SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 187

APLIKACIJA ZA VIZUALIZACIJU I UREĐIVANJE MODELA IT SUSTAVA I PRIPADAJUĆE IMPLEMENTACIJE SIGURNOSNIH POLITIKA

Karlo Lončar

Zagreb, Lipanj 2023.

Zahvaljujem se Bogu, svojoj obitelji i prijateljima za svu podršku koju su mi ukazali u životu.

Sadržaj

Uvod

U današnjem razvijenom svijetu koji je postao iznimno povezan putem interneta, skoro svaka kompanija je prisiljena izgraditi i održavati vlastiti informacijski sustav kako bi neprestano izvršavala svoje funkcije i pružala usluge svojim korisnicima. S obzirom na brzi tehnološki napredak i sve veću povezanost, složenost mrežnih struktura značajno raste. Potrebno je naglasiti kako je sigurnost postala sve veći problem u takvim kompleksnim sistemima zbog nedostatka sveobuhvatnih pravila koji mogu osigurati sisteme od svih mogućih prijetnji. Takve strukture često nastaju prirodno kroz dulji vremenski period ovisno o svojim specifikacijama i zahtjevima te prateći iskustva i najbolje prakse izrade. Jedan od glavnih problema koji se nameće u ovom kontekstu jest ograničena mogućnost za edukaciju i testiranje ovih kompleksnih mreža putem simuliranih okruženja. Razlog za to uglavnom leži u činjenici da mnoge kompanije, prvenstveno iz sigurnosnih razloga, nisu spremne omogućiti pristup svojim mrežnim sustavima kako ne bi izložile svoje sustave mogućim opasnostima.

S obzirom na navedene izazove, javlja se potreba za drugačijim pristupom - izradom vlastitog sustava koji bi omogućio generiranje mrežnih struktura te njihovu simulaciju i testiranje. Međutim, takav sistem je po svojoj prirodi izuzetno kompleksan i zahtjeva veliku pažnju prilikom dizajniranja. Bitno je da se, uz samu strukturu i funkcionalnost, razmisli i o korisničkom sučelju. Korisničko sučelje mora omogućiti lako korištenje, vizualizaciju i uređivanje mreže prema potrebi korisnika, čime se omogućuje lakše razumijevanje i manipulacija kompleksnih mrežnih struktura.

Cilj je omogućiti široku primjenu uz aplikaciju koja može služiti kao jedan od alata u izradi sistema koja se dalje može upotrijebiti u simulacijama kibernetičkih napada, istraživanjima propusta mreža te po mogućnosti kao dodatan izvor podataka za treniranje neuronskih mreža radi razvijanja još sofisticiranijih alata. Kroz ovaj rad, nastoji se objasniti kako je dizajnirano korisničko sučelje takve aplikacije koje bi zadovoljilo spomenute zahtjeve.

U prvom poglavlju se opisuju informacijsko tehnološki sistemi, postojeći prototip za generiranje sustava te zahtjevi vizualizacije generiranih sustava. U drugom poglavlju se raspravlja o korištenim tehnologijama pri izradi sučelja te u trećem poglavlju način izrade

osnovne strukture sučelja. U četvrtom poglavlju je obrazložen način izrade simulacije za vizualizaciju te obrada i uređivanje podataka generiranih sistema. U petom i šestom poglavlju se diskutiraju problemi, ograničenja te moguće nadogradnje aplikacije i konačno slijedi zaključak u kojem se opisuje konačno rješenje te popis korištene literature pri izradi rada i objašnjenje korištenih skraćenica.

1. Informacijsko tehnološki sustavi

Informacijsko tehnološki sustavi [1] (engl. *Information technology systems*), skraćeno ITS, su sustavi koji obuhvaćaju sve informacijske procese koji se koriste u poslovanju neke organizacije uključujući programsku opremu (engl. *hardware*), softvere (engl. *software*), posrednički sloj (engl. *middleware*), alate, baze podataka, tehničke i poslovne informacije, dokumente, registracije, isprave i slične dokumente koji se pohranjuju i koriste u raznovrsnim poslovnim procesima. Područje ITS-a obuhvaća proces upravljanja tim sustavima i tehnologijom koju koriste. Svi aspekti, od jednostavne IT podrške do razvoja sofisticiranih mrežnih rješenja, spadaju pod ovu široku kategoriju [2]. Uviđa se aspekt sigurnosti u upravljanju ovakvim sustavima zbog njihove veličine i kompleksnosti. U takvim složenim sustavima je bitno pomno istražiti osjetljive točke u sustavu i u međusobnoj komunikaciji elemenata sustava. Mnogi sigurnosni stručnjaci su zaduženi za taj posao, ali najveći problem predstavlja kompleksnost sustava što dovodi do učestalih ljudskih propusta te pogrešaka u podešavanju sigurnosnih pravila.

Automatizacija takvog procesa bi uvelike olakšala proces osiguravanja sustava. Prvi korak se očituje u podizanju sigurnosnih zahtjeva na najvišu razinu apstrakcije kako bi korisnici mogli na jednostavan način iznijeti svoje zahtjeve na govornoj razini te analiziranjem tih zahtjeva bi se kroz razvijeni proces generiralo rješenje na najnižoj tehničkoj razini koja je potrebna kako bi svi elementi sustava primijenili generirana svojstva u svojim postavkama.

Uzevši automatizaciju spomenutog procesa kao dugoročni cilj, u znanstvenom radu [3] je izrađen prototip koji pokušava modelirati IT sustave te će biti opisan u nastavku.

1.1. Generator informacijsko tehnoloških sustava

Prototip za generiranje ITS-a, nadalje oslovljavan kao generator, je programsko rješenje koje uz pomoć predefiniranih unosa i raznim izračunima generira sistem koji zadovoljava unesene sigurnosne zahtjeve. Detaljno obrazloženje izrade te svojstva generatora se nalazi u spomenutom znanstvenom radu, ali će u nastavku biti objašnjeni ključni aspekti generatora koji su potrebni za daljnju razradu teme. Generator kao ulaz zahtjeva dvije cjeline:

- Predlošci i pravila (engl. *templates and rules*)
- Parametri (engl. *parameters*)

Predlošci u ovom kontekstu označavaju nacrt sustava, tj. generičan dio ITS-a koji se inicijalizira i specijalizira primjenom sigurnosnih pravila ovisno o potrebi. Predlošci opisuju programske pakete, uloge zaposlenika i organizacijske servise:

- Programski paketi - najčešće uključuju lokalne i mrežne ovisnosti, odobrene servise na računalima zaposlenika, apstraktnu mjeru programske opreme i slično.
- Uloge zaposlenike - povezuju zaposlenike s informacijama o njihovim zahtjevima za servisima, što opisuje potrebne programe i usluge čije radne stanice zaposlenika moraju moći podržavati.
- Usluge koje organizacija pruža - opisuju servise koje organizacija mora pružati svojim vanjskim korisnicima putem interneta.

Naravno, u izradi sistema specifičnog za potrebe korisnika potrebni su parametri koji pobliže opisuju traženi sustav. Ti parametri su odvojeni u sljedeće atribute:

- Podskupine uloga zaposlenika (engl. *Employee role subgroups*) - specificiraju pripadnost zaposlenika u skupinu ljudi s određenim ovlastima i ulogama u organizaciji te broj takvih zaposlenika.
- Zbirke podataka (engl. *Data collections*) - opisuju koji podatci moraju biti pristupačni kako bi pružali potporu organizaciji.
- Pružene vanjske usluge (engl. *Provided external services*) - opisuje listu vanjski servisa koja organizacija mora pružati korisnicima.
- Mrežne politike (engl. *Network policies*) - opisuju bilo kakva pravila o dopuštenim i zabranjenim akcijama koje se odvijaju u mrežnom segmentu.

Raznovrsni parametri mogu određivati i kombinirati veliki broj različitih sistema, najčešće ovisno o veličini organizacije. Potrebno je naglasiti kako generator ima određeni stupanj slobode pri obradi parametara kada su izostavljeni ili ne postoji kvalitetna veza između njih.

Također, generator podržava određena pravila pri generiranju sustava. Istaknuti skup pravila su načini rada zaposlenika (engl. *Employee mode*) u kojem se referira na način

stvaranja podatkovnih skupova ovisno o samom zaposleniku, način rada s uslugama (engl. *Service mode*) u kojem se referira na način kreiranja podatkovnih skupova ovisno o uslugama koje organizacija pruža te programski način rada gdje se referira na način kreiranja podatkovnih skupova ovisno o softveru. U sklopu tih pravila se još ističu pravila mrežne segmentacije (engl. *Network segmentation rules*) koji specificiraju zahtjeve koji se tiču samo određenog dijela mreže za podatkovne skupove te softverske podrške.

Nakon pružanja svih spomenutih zahtjeva, generator će moći generirati sustav koji zadovoljava te zahtjeve. Generator prolazi kroz brojni niz koraka i provjera te naknadnog ručnog testiranja kako bi stvorio što kvalitetniji sustav koji bi u teoriji mogao replicirati sustave poput onih koji se koriste u poslovnom svijetu.

Kao rezultat se generira datoteka u JSON formatu [4] pod nazivom *results.json* koja sadrži sve potrebne informacije o novo generiranom sustavu. Sadrži informacije o entitetima mreže, njihovim softverskim paketima i preostalim konfiguracijskim podatcima, uloge zaposlenika i pravila ponašanja entiteta u mreži te potrebne informacije o vezama između tih entiteta.

1.2. Vizualizacija generiranih sustava

S trenutnim razumijevanjem mogućnosti i uporabe generatora dobiva se temeljna cjelina aplikacije. Kako bi korisnik mogao bez poteškoća koristiti taj proces u izradi vlastitih sustava, prvo se mora omogućiti njegov unos zahtjeva putem korisničkog sučelja. Naravno, korisnik ne želi javno dijeliti svoje zahtjeve ili generirane sustave što potiče izradu korisničkog profila s ugrađenim sigurnosnim funkcionalnostima. Korisnik vjerojatno želi generirati veliki broj različitih sustava te pratiti njihove specifikacije i povijest izmjena. Uzevši to u obzir, korisniku se mora pružiti pregled svih prijašnjih generiranih sustava te prostor gdje on može raditi željene izmjene. Uzevši to u obzir, postoji potreba izrade poslužitelja koji će spremati korisnikove zahtjeve, generirane sustave te bilo kakve izmjene sustava u obliku verzija.

Takav poslužitelj će slati zahtjeve na generator putem vanjskog API sučelja [5] čime obavlja ulogu posrednika u procesu generiranja i vizualizacije sustava. Kako bi korisnik mogao predočiti i razumjeti generirani sustav, potrebno je izraditi vizualni prikaz rezultata te prilagoditi rezultat generatora radi što kvalitetnijeg prikaza uz intuitivan prikaz sustava i alata koji se koriste u njegovoj izmjeni i proučavanju.

Konačno, korisniku se mora omogućiti opcija preuzimanja konfiguracije sustava u obliku JSON formata s kojom može nastaviti rad u drugim alatima. Odabir i obrazloženje odabranih tehnologija koje zadovoljavaju sve navedene korisničke zahtjeve se opisuje u sljedećem poglavlju.

2. Korištene tehnologije

Kako bi se uspješno izradile ove funkcionalnosti aplikacije provedeno je dugotrajno istraživanje postojeći radnih okvira, biblioteka i raznih ostalih tehnologija kako bi se na što kvalitetniji i lakši način izgradila aplikacija. Bitni faktori su vremensko ograničenje, preferencija prema tehnologijama koje su besplatne za korištenje i mogućnost proširenja aplikacije u budućnosti.

2.1. D3.js

Kao najbitniji dio izrade sučelja se izdvojio vizualni prikaz mreže koju korisnik mora moći pregledavati te mora moći među djelovati s njom kako bi radio željene izmjene. S obzirom na to da je izrada takve vizualizacije kompleksan posao, odlučilo se istražiti biblioteke koje trenutno postoje koje bi pomogle pri rješavanju zadanog problema. Nakon dužeg istraživanja izabrana je JavaScript biblioteka D3.js.

2.1.1. Pregled

D3.js [6] je biblioteka otvorenog kôda za vizualizaciju podataka bazirana na JavaScript-u. Poznata i kao D3, služi kao alat koji se temelji na web standardima, uključujući SVG, HTML i CSS tehnologije, pružajući neusporedivu fleksibilnost u stvaranju dinamičkih, interaktivnih grafika vođenih podacima. Izvorno razvijen kao nasljednik ranijeg Protovis okvira, D3 je od lansiranja verzije 2.0.0 u kolovozu 2011. godine postao ključan alat za vizualizaciju podataka na webu [7].

D3 omogućava korisnicima da stvaraju raznovrsne vizualizacije, od jednostavnih dijagrama do složenih interaktivnih grafičkih prikaza. Pruža programerima potpunu kontrolu nad svim aspektima vizualizacije, uključujući boje, oblike, stilove, animacije i tranzicije. Bez obzira na svoju strmu krivulju učenja, rezultati koji se mogu postići pomoću D3.js-a su impresivni. Ova biblioteka je ključna komponenta mnogih biblioteka za izradu grafikona i nalazi široku primjenu u različitim sektorima, uključujući znanstvena istraživanja, novinarstvo podataka i analizu weba[8].

Od izdanja verzije 4.0.0 [9] u lipnju 2016. godine, D3.js je transformiran iz jedinstvene biblioteke u kolekciju manjih, modularnih biblioteka koje se mogu koristiti neovisno. Ciljani modul koji je upotrijebljen u ovoj izradi je modul *d3-force* koji će biti objašnjen kasnije.

2.1.2. Povezivanje podataka

Ključni koncept u D3 je povezivanje podataka. D3 omogućuje da se proizvoljni podatci povežu za DOM [10], a zatim primjene odabrane transformacije vođene podacima na dokumentu. Osnovni način na koji D3 obraduje podatke jest kroz tri ključna izbora: unos (engl. *enter*), ažuriranje (engl. *update*) i izlaz (engl. *exit*) [11].

- Izbor *enter* je skup elemenata koji trebaju biti dodani na stranicu (tj., novi elementi podataka koji nemaju odgovarajuće DOM elemente). Ovaj izbor se obično koristi za dodavanje novih elemenata na stranicu.
- Izbor *update* se odnosi na elemente koji su već prisutni i za koje postoje odgovarajući podaci. Ovaj izbor se obično koristi za ažuriranje postojećih elemenata na stranici.
- Izbor *exit* uključuje sve elemente za koje ne postoje odgovarajući podaci (tj., DOM elementi za koje nema odgovarajućih podataka). Ovaj izbor se obično koristi za uklanjanje elemenata sa stranice.

D3 omogućuje učitavanje podataka iz različitih izvora i u različitim formatima, uključujući CSV, TSV i JSON formatu. Funkcije d3.csv, d3.tsv i d3.json koriste se za učitavanje podataka u navedenim formatima. Svaka od ovih funkcija vraća obećanje koje se ispunjava kada se podaci učitaju i obrađuju.

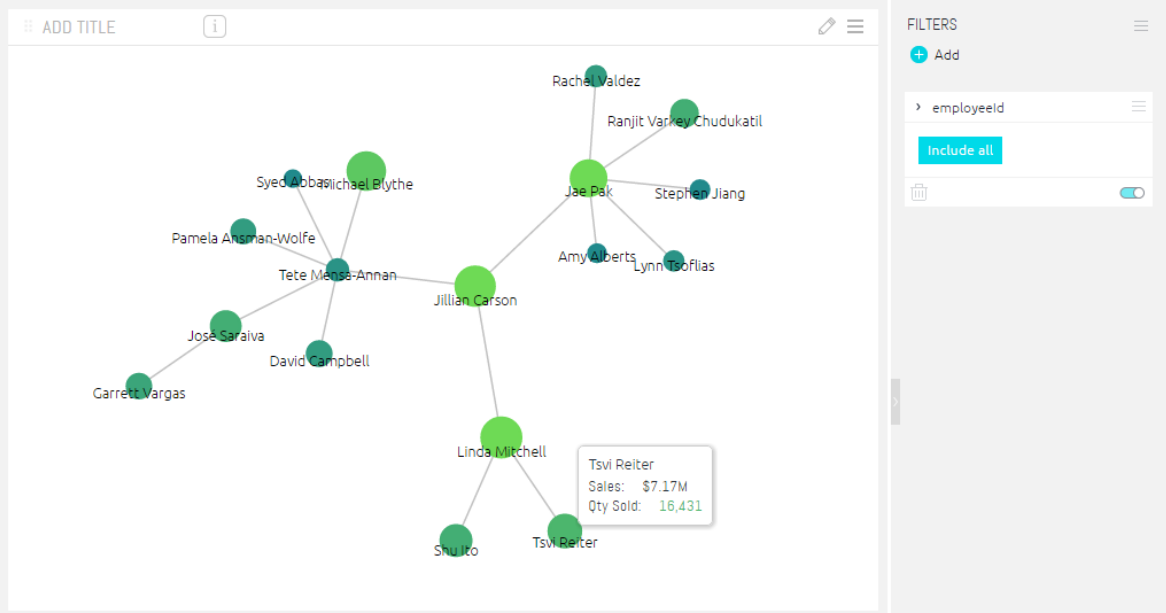
Kada se podaci učitavaju, često je potrebno izvršiti dodatno formatiranje ili transformaciju podataka prije nego što se mogu koristiti u vizualizaciji. Na primjer, numerički podaci često se učitavaju kao tekst i moraju se pretvoriti u brojeve. D3.js pruža niz alata za formatiranje i manipulaciju podacima, uključujući sortiranje, filtriranje, mapiranje i smanjivanje što je detaljno objašnjeno u njegovoj detaljnoj dokumentaciji [12].

2.1.3. Raspored simulacije sila

Raspored simulacije sila (engl. *Force Simulation Layout*) je jedan od predložaka za izradu interaktivne vizualizacije kojeg pruža D3 [13]. Ovaj raspored koristi koncepte fizike i simulira sile kako bi utvrdio optimalnu poziciju elemenata (najčešće čvorova i veza u mrežnom grafikonu) unutar vizualizacije. Bazira se na algoritmima za crtanje grafova koji koristi različite izračune i simulacije prirodnih sila [14]. Kada se koristi ovaj raspored, svaki čvor u mreži tretira se kao pojedinačni objekt s određenim silama koje djeluju na njega. Ove sile mogu uključivati gravitaciju (privlači čvorove prema centru), naboj (odbija čvorove jedan od drugoga) i veze (privlači povezane čvorove zajedno).

Funkcija *d3.forceSimulation()* stvara simulaciju. Postoji mogućnost dodavanja različitih sila u simulaciju koristeći *force* metodu. D3 pruža nekoliko prethodno definiranih sila, a one korištene u izradi će biti objašnjene u četvrtom poglavlju.

Tražeći prikladno rješenje za vizualizaciju sustava, konceptualno se orijentiralo na primjer sa slike (Slika 2.1):



Slika 2.1 Konceptualni prikaz vizualizacije sustava [15]

Informacijski sustav se može prikazati na sličan način tako što će elementi mreže biti predstavljeni čvorovima koji sadrže podatke elemenata koji se mogu koristiti u vizualizaciji ili u procesiranju podataka uz pomoćne funkcije te će veze između mrežnih elemenata biti predstavljene poveznicama kako bi se prikazao međuodnos elemenata.

2.2. Angular

Angular je radni okvir (engl. *framework*) [16] za izgradnju korisničkog sučelja web aplikacija koji ima zanimljivu povijest te veliku primjenu u poslovnom svijetu. Angular se karakterizira kao okvir koji ima strmu liniju učenja, ali veliku isplativost ulaganja vremena u njegovo savladavanje. Kao i slični okviri, sadrži vlastitu strukturu i pravila izrade sučelja koji će biti pobliže objašnjeni u ostatku poglavlja.

2.2.1. Povijest

Googleov stručnjak Miško Hevery je inicirao osnivanje razvojnog okvira kao sekundarni projekt s ciljem olakšavanja izgradnje web aplikacija za vlastite interne projekte [17]. U 2010. godini, ovaj je okvir službeno lansiran kao otvoreni izvorni kôd (engl. *open-source*) za korisničko sučelje pod imenom AngularJS. Zasnivao se na MVC arhitekturi [18] te je kombinirao HTML, CSS i JavaScript tehnologije u svojoj strukturi.

S obzirom na to da AngularJS nije izvorno koncipiran kao formalan projekt izgradnje sučelja, kroz nekoliko godina suočio se s problemima inherentnim svojoj osnovnoj strukturi. Alternativna rješenja počela su nuditi funkcionalnosti koje nisu bile predviđene pri njegovoj izgradnji. Jedan od ključnih aspekata bio je razvoj unakrsnih platformi (engl. *cross-platform*) uz skalabilnost dizajna korisničkog sučelja, što gospodin Hevery nije originalno predvidio pri pokretanju projekta. Tijekom razdoblja 2014.-2015., AngularJS prošao je kroz svoje prvo veliko restrukturiranje.

U svjetlu pojavljivanja novih arhitektura kod konkurenata, primarno MVVM arhitekture [19] kod React.js-a, zamišljena je nova struktura temeljena na komponentama. Rekonstrukcijom okvira stvoren je AngularJS 2.0, koji je potom preimenovan u Angular, a za označavanje verzija okvira koristi se notacija "major.minor.patch". Trenutačna verzija koja je korištena u ovom radu je Angular 15. U međuvremenu, JavaScript je zamijenjen TypeScript-om [20] u ulozi programiranja logičkih funkcija sučelja.

TypeScript je nadogradnja JavaScript-a, koji zadržava originalne funkcionalnosti, ali ih proširuje novim konceptima kao što su tipovi i unaprijeđene mogućnosti testiranja. Budući da web preglednici ne podržavaju TypeScript, Angular automatski konvertira TypeScript kôd u JavaScript kôd putem svog sastavljača (engl. *compiler*).

2.2.2. Primjena

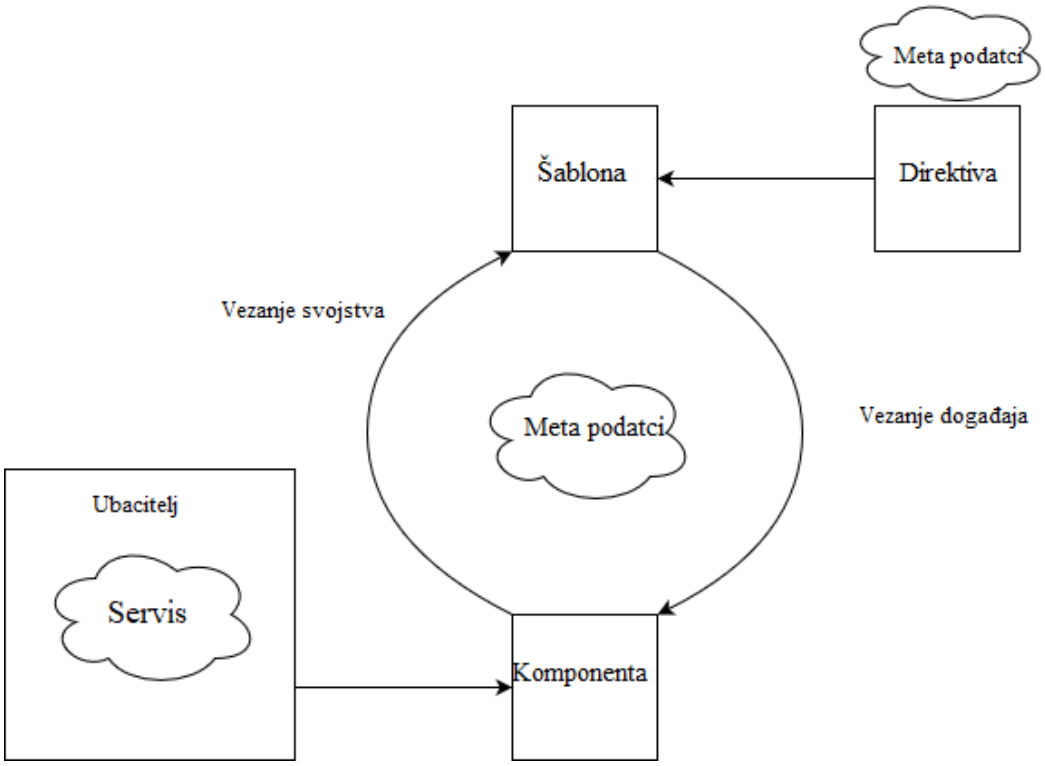
U 2020. godini, Angular je ostvario impresivne rezultate [21]. Programeri ga prepoznaju kao jedan od najboljih alata za izgradnju web stranica, a TypeScript je rangiran kao treći najpopularniji programski jezik za kreiranje web aplikacija. TypeScript nudi prednosti u vidu ugrađenih provjera i testiranja koji se odvijaju u pozadini tijekom razvoja kôda, omogućavajući brzo identificiranje i sprječavanje potencijalnih problema u ranoj fazi razvoja aplikacija. Dodatno, uz odgovarajuću podršku web preglednika, moguće je izravno u pregledniku ispravljati greške u TypeScript kôdu, što doprinosi boljoj efikasnosti, testiranju i izgradnji.

Jedna od prednosti Angular-a nad ostalim okvirima je standardizirana struktura kôda, brza izgradnja aplikacija te sposobnost realizacije većih projekata zbog dobre preraspodjele kôda i njegove ponovljive upotrebe. Google pruža dugotrajnu potporu u razvoju i održavanju radnog okvira te poveća njegovu upotrebu u svojim aplikacijama. Angular pridaje veliku važnost konzistentnosti kôda, prepoznajući probleme i potencijalne troškove koji mogu proizaći iz nedostatka konzistencije. Kôd u Angular-u je ponovno upotrebljiv zbog podjele na komponente. Na primjer, komponenta koja izračunava statistiku na jednoj stranici može se koristiti za istu funkciju na drugoj stranici jednostavnim uvozom. Konzistentnost i ponovna upotrebljivost rezultiraju poboljšanjem čitljivosti kôda, čime se značajno povećava produktivnost programera. Uzimajući u obzir navedene karakteristike, nije iznenađujuće da su neke od najpopularnijih web stranica i aplikacija bazirane na Angular-u, uključujući Gmail, Paypal, Lego i UPS [22].

Angular nudi brzinu i izvanredne performanse u svojim aplikacijama, pretvarajući predloške u visoko optimiziran kôd za izvršavanje preko JavaScript virtualnih strojeva, te pruža podršku najčešće korištenim radnim okvirima i poslužiteljskim jezicima poput 'Node.js', '.NET' i 'PHP' za gotovo trenutno procesiranje HTML i CSS datoteka. Da bi se uštedjelo na resursima i poboljšala brzina učitavanja stranica, Angular dijeli kôd na segmente koji se učitavaju po potrebi za procesiranje stranica koje se trenutno pregledavaju, dok se ostatak učitava u pozadini ili po potrebi. Svojim vlastitim korisničkim sučeljem za unos naredbi (engl. *Command Line interface*), Angular nudi ubrzane procese generiranja kôda, postavljanja lokalnog poslužitelja, testiranja i pripreme kôda aplikacije za produkcijsko okruženje.

2.2.3. Struktura

Komponenta (engl. component) je temeljna građevna jedinica koja odgovara jednom prikazu na ekranu, funkcionalnoj cjelini ili čak cijeloj stranici. Komponente su međusobno povezane, omogućujući laku komunikaciju i razmjenu informacija između komponenata na istoj ili različitoj upravljačkoj razini koje nastaju grananjem. Komponente se oslanjaju na usluge koje pružaju specifične funkcionalnosti nevezane za izravan prikaz komponente. Prikaz pojednostavljene arhitekture prikazan je na slici (Slika 2.2):



Slika 2.2 Prikaz pojednostavljene arhitekture Angular-a

Komunikacija podatkovne logike komponente i predloška koji se prikazuje na ekranu korisnika je pojednostavljena mehanizmima razmjene informacija, bazirana na MVVM arhitekturi. Ti mehanizmi su sljedeći:

- Vezanje događaja (engl. *Event binding*) - je jednosmjerno povezivanje koje osluškuje promjene na predlošku i pri određenim promjenama omogućava pozivanje metoda i funkcija u komponenti te izmjenu vrijednosti varijabli u komponenti.
- Vezanje svojstva (engl. *Property binding*) - je jednosmjerno povezivanje gdje se podaci i vrijednosti varijabli prikazuju na elementima predloška.

- Dvosmjerno vezanje podataka (engl. *Two-way data binding*) - je dvosmjerno povezivanje koje sklapa vezanje svojstva s vezanjem događaja koristeći *ngModel* direktivu. Pri korisnikovom unosu podataka na predlošku dio komponente koji je vezan za taj dio automatski ažurira vrijednosti varijabli te ažurira elemente prikaza koji su vezani na te iste varijable komponente. Na primjer, korisnik unosi svoje ime i istovremeno se na ekranu ispisuje njegov unos imena.

Moduli, komponente i servisi su zapravo klase koje koriste dekoratore (engl. *decorators*). Ovi dekoratori označavaju sastavljaču (engl. *compiler*) o kakvom tipu datoteka se radi. Označavaju tip strukture te pružaju meta podatke (engl. *meta data*) koji označavaju kako će se oni koristiti i interpretirati u aplikaciji.

Meta podatci komuniciraju radnom okviru kako će procesirati klasu. Za primjer se uzima glavna komponenta *app.component.ts*:

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
```

U ovoj komponenti se može uočiti da `@Component` označava sastavljaču da je ova datoteka zapravo komponenta. Sadrži popratne meta podatke koji pobliže opisuju strukturu komponente: `selector` označava sastavljaču o kojoj komponenti se radi kada naiđe na istoimeni naziv u etiketi (engl. *tag*) HTML kôda. U pozadini sastavljač takvu specifičnu etiketu zamijeni HTML datotekom označenoj u `templateUrl`. `styleUrls` definira CSS datoteku čije klase povezuje sa `templateUrl` datotekom radi izrade vlastito odabranog oblikovanja prikaza.

Modul (engl. *module*) obavlja funkciju upravitelja komponenata gdje oponaša funkcionalnu cjelinu grupacijom komponenti, servisa i direktiva te uvozom drugih modula. Cilj je obuhvatiti funkcije i podatke na jednom skupu samo tamo gdje su potrebne. S tim pristupom se kôd vrlo jednostavno organizira tijekom svog razvoja te se komunikacija komponenti pojednostavljuje podizanjem razine komunikacije na module.

Komponente i dalje komuniciraju međusobno unutar jedne cjeline tj. modula, ali komunikacija s komponentama koje se nalaze izvan zajedničkog modula se odvija uz pomoć komunikacije uspostavljene između modula koji su zaduženi za te komponente.

Svaka Angular aplikacija sadrži barem jedan modul - korijenski modul, konvencionalno imenovan *AppModule*. Struktura modula je označena dekoratorom *@ngModule* koji označava datoteku kao modul. U dekoratoru možemo prilagoditi modul i povezivati ga s ostatkom aplikacije.

Osnovni opisni elementi *@ngModule* dekoratora su sljedeći:

- Deklaracije (engl. *Declarations*) – deklariraju komponente za koje će modul u pitanju biti zadužen upravljanjem.
- Izvozi (engl. *Exports*) – navode imena komponenti za koje se želi da ih uvoze drugi moduli i koriste u svojim funkcijama.
- Uvozi (engl. *Imports*) – navode imena modula i komponenata koje se žele uvesti u modul kako bi ih i on mogao koristiti.
- Pružatelji usluga (engl. *Providers*) – definiraju servise kako bi ih modul mogao koristiti unutar svojih deklariranih komponenti.
- Pokretanje (engl. *Bootstrap*) – označava glavni prikaz aplikacije tj. korijenske komponente koja sadrži sve ostale prikaze i komponente. Samo korijenski modul bi trebao sadržavati to svojstvo.

Servis (engl. *service*) je komponenta koja je označena posebnim dekoratorom *@Injectable().* Koristi se u slučaju kada se želi upravljati podatcima ili programskom logikom koja nije nužno vezana za specifičnu komponentnu te se želi omogućiti njeno dijeljenje na više mjesta u aplikaciji po potrebi.

Servisi mogu biti ubačeni (engl. *injected*) u komponentu kao ovisnost (engl. *dependency*) čime se izgrađuje modularan, ponovno uporaban i efikasan kôd. Ubacivanje ovisnosti je pristup ugrađivanja servisa u komponente kako bi se odvojili zadatci koje izvršava sama komponenta i koje izvršavaju funkcije servisa. Prakticira se pristup da se svaka komunikacija s poslužiteljem ili različitim drugim API-em te internetskim uslugama provodi kroz ubačene servisne funkcije dok komponenta koristi samo rezultate tih servisnih funkcija kako bi upravljala svojim prikazima stranice.

Pri korištenju servisa koriste se određeni elementi koji upravljaju promjenama i događajima koji se događaju u aplikaciji. Angular je poznat po upotrebi specifične funkcionalnosti reagiranja na događaje. Dijelovi tih funkcionalnosti su osluškivači, subjekti i mehanizam pretplate.

Osluškivači (engl. *observables*) pružaju potporu u upravljanju porukama između dijelova aplikacije i izvan nje. Njihova primjena je učestala kao tehnika upravljanja događajima u aplikaciji, kod asinkronog programiranja i upravljanja s više vrijednosti istovremeno. Mehanizam osluškivača je softverskog oblikovanja u kojem objekt, zvan subjekt (engl. *subject*), održava listu svojih ovisnih dijelova, zvanih osluškivačima i automatski ih obavještava o nadziranim promjenama. Ovaj mehanizam je sličan „publish/subscribe“ dizajnerskom pristupu [23].

Ključan dio mehanizma je da promjene i informacije koje se prenose kao tok podataka (engl. *data-stream*), nisu uključene u proces izvođenja sve dok se ne pretplate (engl. *subscribe*) na taj osluškivač. Nakon pretplaćivanja preuzima se tok podataka i mogu se procesirati zamijećene promjene ili odgovori na zahtjeve.

Najčešća primjena servisa je u pozivima na API poslužitelja koji se koriste u razmjeni poruka između korisničkog sučelja i poslužitelja koji sačinjavaju aplikaciju gdje se zahtjevi sa sučelja šalju prema poslužitelju te se prate njihova stanja preko osluškivača koji primaju odgovore od poslužitelja. Pretplativši se na te osluškivače mogu se koristiti dohvaćene informacije s poslužitelja u funkcijama korisničkog sučelja.

Jedna od najvećih prednosti Angular-a su njegove direktive. Direktiva (engl. *directive*) je proširenje HTML etiketa s raznim prilagodljivim funkcionalnostima koje nude više opcija pri oblikovanju stranice. Pošto su prikazi u Angular-u dinamični, pri procesiranju prikaza prate se instrukcije zadane direktivama. Direktive uvode mogućnost povezivanja korisničkog unosa s logikom programa i vrijednostima varijabli, prikazivanju i sakrivanju određenih HTML elemenata ovisno o vlastito zadanim uvjetima. Postoje dva tipa direktiva: strukturalna direktiva i atributna direktiva.

Strukturalna direktiva mijenja prikaz tako da dodaje, otklanja ili zamjenjuje određene elemente u prikazu. Karakterizira ju sintaksa: „*directiveName“ te su najčešći primjeri „*ngIf“ i „*ngFor“.

Atributna direktiva mijenja izgled ili ponašanje određenog elementa. Izgleda kao obični HTML atribut te je tako i dobila ime. Primjer:

```
„<input [(ngModel)]=[“user.name“]>
<p> This is your input:{{user.name}}</p>
```

Direktiva `[(ngModel)]` uključuje dvostranu vezu podataka te u ovom primjeru osigurava da će se objekt `user` u komponenti mijenjati ovisno o onom što korisnik upiše.

Time se osigurava automatsko ažuriranje unutarnjih varijabli korisničkim upisom te se ista takva varijabla može ispisivati na prikazu. Direktive uvode i opciju postavljanja cijelih komponenti unutar vlastitih etiketa bilo gdje u HTML datoteci. Na primjer, ako se izgradi kalkulator u jednoj komponenti s vlastitim predloškom, ta se komponenta može uvrstiti unutar *div* etikete te će se sastavljač pobrinuti da se taj kalkulator tamo pravilno smjesti sa svim svojim funkcionalnostima i oblikovanjem preko klasa CSS datoteke vezane za njega.

Za prikaz određenih elemenata i stranica tj. navigacijom aplikacije, brine se Angular-ov sistem usmjeravanja (engl. *routing*) koji definira kako će se i kada učitavati određene komponente. Jedno od velikih koristi kojeg omogućava usmjeritelj je koncept zvan „lazy-loading“ [24]. Umjesto da se sve datoteke i resursi učitavaju pri dolasku na aplikaciju, što bi usporilo njeno vrijeme rada, usmjeritelj ima opciju učitavanja samo onih modula i komponenti koje se trenutno prikazuju dok se ostatak aplikacije može učitavati u pozadini ili isto po potrebi. Tako se ubrzava rad aplikacije te nudi kvalitetnije korisničko iskustvo.

2.3. Ostale tehnologije i biblioteke

Uz spomenutu biblioteku za vizualizaciju i radni okvir, korištene su tehnologije i alati za pripomoć u izradi dizajna korisničkog sučelja te generiranja kodnih isječaka za temeljne cjeline sučelja. Navod i kratki opisi tehnologija i alata se nalaze u nastavku.

2.3.1. Angular CLI

Angular CLI [25] je pomoćni alat razvijen od programera Angular-a koji omogućava generiranje učestalog kôda na brži i efikasniji način. Nudi dodatne korisne opcije kao što je automatizirano uvođenje usmjeravanja komponenti, posluživanja privremenog lokalnog poslužitelja za pokretanje i izgradnju aplikacije te mogućnost sastavljanja kôda za produkciju. Preko Node.js biblioteke nudi široku opciju generiranja dijelova kôda preko „ng“ funkcija. Na primjer, umjesto pisanja vlastitih HTML, CSS i TS dijelova komponente pod imenom „Component_A“, upisom naredbe „`ng generate component Component_A`“ automatski se generiraju te datoteke i spajaju s ostatkom aplikacije.

Instalacija Angular CLI-a se izvršava u terminalu uz pomoć node.js-ovog menadžera paketa *npm*. Izvođenjem samo jedne komandne linije „`npm install -g @angular/cli`“, mogu se koristiti svi alati koje pruža Angular CLI.

U razvoju aplikacije najčešća komanda koja se koristi je „`ng serve`" s kojom se generira lokalni poslužitelj u pozadini koji će se koristiti za pokretanje sučelja te se svakom izmjenom u kôdu automatski ažuriraju elementi stranice.

Kada je sučelje spremno za povezivanje s poslužiteljem koristi se komanda „`ng build`" koja sklapa datoteke u zadani direktorij odakle će poslužitelj koristiti HTML i JavaScript datoteke za prikaz aplikacije.

Zamjećuje se velika lakoća korištenja CLI-a koji obavlja složene zadatke spajanja datoteka, sklapanja kôda te pokretanja aplikacije kroz par kratkih i jednostavnih linija kôda.

2.3.2. Angular Material

Angular Material [26] je biblioteka modula koja pruža jedinstveni izgled koji daje dojam modernog i poslovnog oblikovanja korisničkog sučelja. Izgrađena i održavana od strane Google-a, uvodi vlastita rješenja u izradi oblikovanja sučelja aplikacije. Nudi širok raspon alata za izradu tablica, gumbi, raspodjela elemenata stranice, prikaza kalendara i sličnih elemenata. Njihov je uvoz jednostavan te Google nudi odlične i jednostavne upute kako koristiti te module. Također nudi uvoz posebnih dijelova modula umjesto cijele biblioteke modula kako bi se iskoristili samo potrebni dijelovi. Uz dobro napisanu dokumentaciju s punu primjena za najčešće uporabe, s lakoćom se mogu izabrati, prilagoditi i implementirati elementi biblioteke u svoje projekte.

2.3.3. Angular Reactive Forms

Angular reaktivni obrasci (engl. *Reactive forms*) [27] pripadaju službenom Angular modulu za pripomoć u kreiranju i upravljanju dinamičkih, kompleksnih i konzistentnih obrazaca. Reaktivni obrasci koriste eksplicitan i nepromjenjiv pristup za upravljanje stanjem obrasca u određenom trenutku. Svaka promjena stanja obrasca vraća novo stanje, čime se održava integritet modela između promjena. Reaktivni obrasci građeni su oko promatranih tokova, gdje su unos i vrijednosti formi pruženi kao tokovi ulaznih vrijednosti, kojima se može pristupiti sinkrono.

Reaktivni obrasci također pružaju jednostavan put do testiranja su podaci dosljedni i predvidljivi kada su zatraženi. Bilo koji pretplatitelji tokova imaju pristup da sigurno manipuliraju tim podacima. Reaktivni obrasci se razlikuju od drugih obrazaca vođenih

predloškom na različite načine. Reaktivni obrasci pružaju sinkroni pristup modelu podataka, nepromjenjivost s operaterima promatranja, te praćenje promjena kroz promatrane tokove.

Ovaj modul rješava kritičan problem generiranja dinamičkog obrasca za korisnikov unos konfiguracije tako što sebe uspješno i konzistentno nadograđuje dodatnim poljima za unos, ovisno o potrebi korisnika, čime omogućava unos konfiguracija s opširnim skupom pravila i velikim brojem parametra.

2.3.4. FontAwesome

FontAwesome [28] je alat za fontove i ikone koji je stvorio Dave Gandy. To je iznimno popularan resurs koji koriste web programeri i dizajneri zbog svoje opsežne biblioteke skalabilnih vektorskih ikona, koje se mogu prilagoditi po veličini, boji, sjenci koristeći CSS atribute. FontAwesome ikone se lako integriraju na web stranice i nude bolju skalabilnost i kvalitetu od tradicionalnih slikovnih ikona. Njihova kompatibilnost s D3 bibliotekom je presudna u odabiru ovog alata jer obje biblioteke koriste SVG u pozadini za prikazivanje vizualnih sadržaja.

3. Implementacija osnovnog dijela sučelja

Pri implementaciji korisničkog sučelja korištene su najbolje prakse, osobno iskustvo i znanje stečeno u prijašnjim godinama tijekom izrade završnog rada te radno iskustvo u odabranom radnom okviru. Korišten je modularni pristup gdje su svi dijelovi sučelja odvojeni u logičke cjeline između kojih je uspostavljena pravilna komunikacija tako i veza između sučelja i poslužitelja. U ovom poglavlju će biti prikazani isječci kôda pri izradi sučelja te obrazložen proces izrade.

3.1. Izrada osnovne strukture aplikacije

Sučelje je podijeljeno na glavne komponente odnosno module koji predstavljaju istovjetnost s prikazom određene stranice koja je prikazana korisniku ovisno o ruti na kojoj se korisnik nalazi u tom trenutku. Glavne stranice su podijeljene na:

- *Login* – stranica koja sadrži obrazac za ispunjavanje korisnikovih vjerodajnica kako bi sigurno pristupio ostatku sučelja.
- *Registration* – stranica koja sadrži obrazac za izradu novog korisničkog računa upisivanjem proizvoljnog korisničkog imena te šifre.
- *Systems* – stranica koja sadrži prikaz svih korisnikovih već unesenih konfiguracija te obrazac za stvaranje nove konfiguracije čime se sukladno generira novi sustav.
- *System details* – stranica koja sadrži vizualizaciju generiranog sustava gdje korisnik može pregledavati, uređivati i nadograđivati takav sustav.

Uz te komponente je izgrađen i servis *system.service.ts* koji provodi svu komunikaciju s poslužiteljem kroz HTTP zahtjeve (engl. *HTTP requests*) uz pomoć definiranih funkcija za unos konfiguracija te zaprimanja generiranih sustava. Implementiran je i servis *auth.service.ts* koji izvršava funkcije autorizacije i autentifikacije te *http-handler.interceptor.ts* koji presreće sve http zahtjeve te obavlja zadaću provjere i obrade pogrešaka (engl. *error*) nastale u komunikaciji s poslužiteljem. Ti servisi se ugrađuju u komponentne koje zahtijevaju njihove funkcionalnosti u svojim operacijama.

3.2. Autentifikacija i Autorizacija

Autentifikacija i autorizacija [29] igraju važnu ulogu u sigurnosti tako što obavljaju funkciju odobravanja pristupa korisnicima do osjetljivih podataka i usluga. Autentifikacija provjerava je li korisnik uistinu onaj koji se predstavlja te autorizacija provjerava i odobrava pristup određenim resursima autentificiranom korisniku ovisno o njegovim definiranim dopuštenjima.

Autentifikacija i autorizacija mogu biti implementirane na razne načine. Odabrano je korištenje JWT [30] preko kojeg će se obavljati navedene funkcije jer je jedan od dostupnih metoda i jednostavan je za korištenje.

JWT je standard koji omogućava izmjenu tokena preko kojih se provodi autentifikacija i autorizacija u web aplikacijama. Temelji se na digitalnim potpisima i kodiranju informacija u obliku JSON objekta. Prilikom validacije korisnika, šalju se njegove vjerodajnice na poslužitelj. Nakon uspješne provjere, poslužitelj izrađuje dva tokena: *access token* i *refresh token* koji sadrže enkodirane (engl. *encoded*) informacije o korisniku, poput identifikatora ili uloge. Potom se token šalje korisniku. Kada korisnik želi pristupiti zaštićenim rutama ili resursima, šalje *access* token kao dio zahtjeva u zaglavlju autorizacije. Poslužitelj provjerava ispravnost tokena te smatra korisnika autentificiranim i autorizira njego pristup traženim resursima. Definirano je trajanje *access token*-a u trajanju od 15 minuta te *refresh token*-a u trajanju 60 minuta od trenutka nastanka. Kada *access token* istekne, koristi se *refresh token* kako bi se generirao novi *access token* i preskočila situacija ponovnog provjeravanja korisnika kao što je prikazanu u ispisu (Ispis 3.1):

```
refreshToken() {
    const refreshToken = this.getToken('refreshToken');
    return this._http.post(this.apiURL + '/users/rest-
auth/token/refresh/', {refresh: refreshToken}).pipe(
        tap((response: any) => {
            this.setToken('accessToken', response.access);
        })
    );
  }
```

Ispis 3.1 – Programski odsječak za osvježivanje tokena

Angular HTTP Interceptor [31], u nastavku interceptor, je posrednički softver koji može presresti odlazne HTTP zahtjeve i dolazne odgovore. Omogućuje programerima da obrade

zahtjeve prije nego što su poslani ili prije nego što dođu do aplikacije. Interceptori se mogu koristiti za izmjenu zahtjeva, dodavanje zaglavlja (tokeni za autentifikaciju), bilježenje informacija, rukovanje pogreškama ili čak ponovno slanje neuspjelih zahtjeva. Budući da pružaju središnje mjesto za takve manipulacije, pomažu u održavanju kôda čišćim i lakšim za održavanje.

Interceptor je aktivan u obostranoj komunikaciji s poslužiteljem, prije nego se šalju zahtjevi na poslužitelj i kada se primaju odgovori s poslužitelja on prvi obrađuje odgovore te ih prosljeđuje dalje. Prije slanja zahtjeva interceptor će provjeriti postoji li token u lokalnu pohranu (engl. *local storage*), te će ga dodati u *header* http zahtjeva kao što je prikazano u ispisu (Ispis 3.2):

```
intercept(
    request: HttpRequest<any>,
    next: HttpHandler
  ): Observable<HttpEvent<any>> {
    if (this.authService.getToken('accessToken')) {
      request = this.addToken(
        request,
        this.authService.getToken('accessToken')
      );
    }

    return next.handle(request).pipe(
      catchError((error) => {
        if (error instanceof HttpErrorResponse &&
error.status === 401) {
          return this.handle401Error(request, next);
        } else {
          return this.handleErrors(error);
        }
      })
    );
  }
```

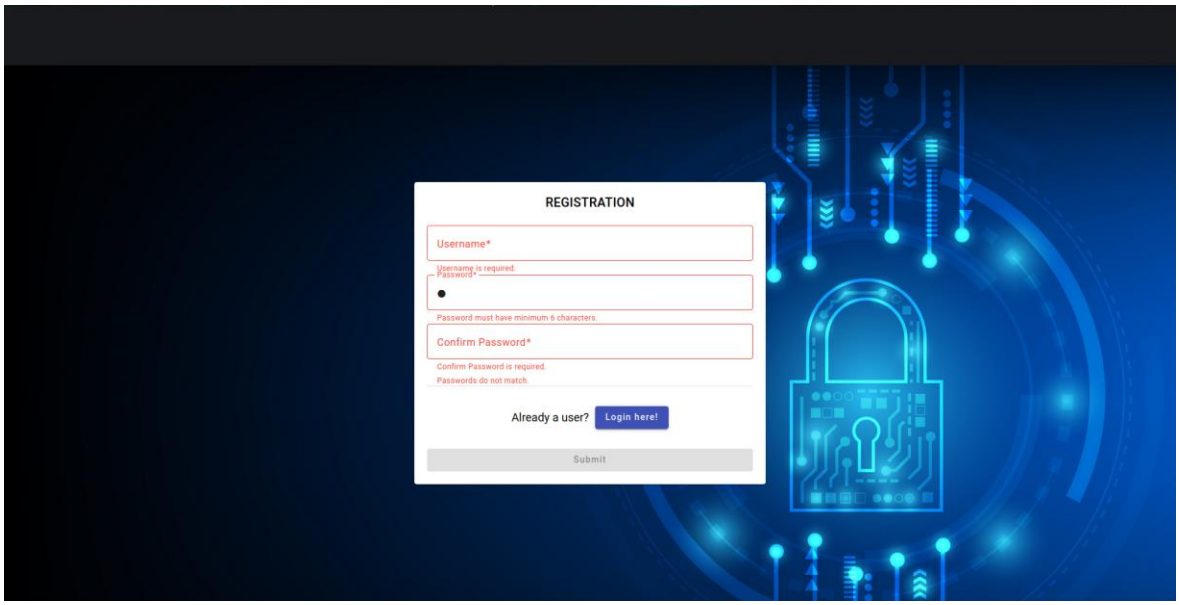
Ispis 3.2 – Programski odsječak interceptora za upravljanje HTTP pozivima

Istovremeno, pri zaprimanju odgovora od strane poslužitelja provjerava postoji li problem u autentifikaciji. U tom slučaju obavlja funkciju osvježavanja tokena te ponavlja prvotno poslani zahtjev. U slučaju primanja ostalih pogrešaka od strane poslužitelja stvara prozor na korisnikovom ekranu sa sakupljenim informacijama o pogrešci.

Ugrađivanjem spomenutih tehnologija se uspostavlja visoka razina sigurnosti korisnika u uporabi sučelja koja minimizira potrebu korisnika da konstantno provjerava validaciju svoga računa.

3.3. Pristupanje sučelju

Kako bi pristupio sučelju korisnik mora proći kroz proces registracije ispunjavanjem obrasca s proizvoljnim korisničkim imenom i šifrom na komponenti *registration* prikazan na slici (Slika 3.1):



Slika 3.1 Prikaz *Registration* komponente

Obrazac je izgrađen uz pomoć reaktivnog obrasca te su uvedene provjere, tj. validacije kako bi se osigurao pravilan unos i smanjio besmislen kontakt s poslužiteljem slanjem neispravnih podataka. Reaktivne forme daju na izbor predefinirane validatore [32] koji mogu poslužiti u većini slučajeva, ali je potrebno složiti napredniju validaciju obrasca uz pomoć prilagođenog validatora kako bi zadovoljio dodatan uvjet. Dodatan uvjet je da je polje *Password* istovjetno vrijednosti polja *Confirm Password*. Tek kada korisnik zadovolji sve uvjete mu se dopušta slanje obrasca na obradu. U ovom ispisu (Ispis 3.3) se uviđa inicijalizacija obrasca:

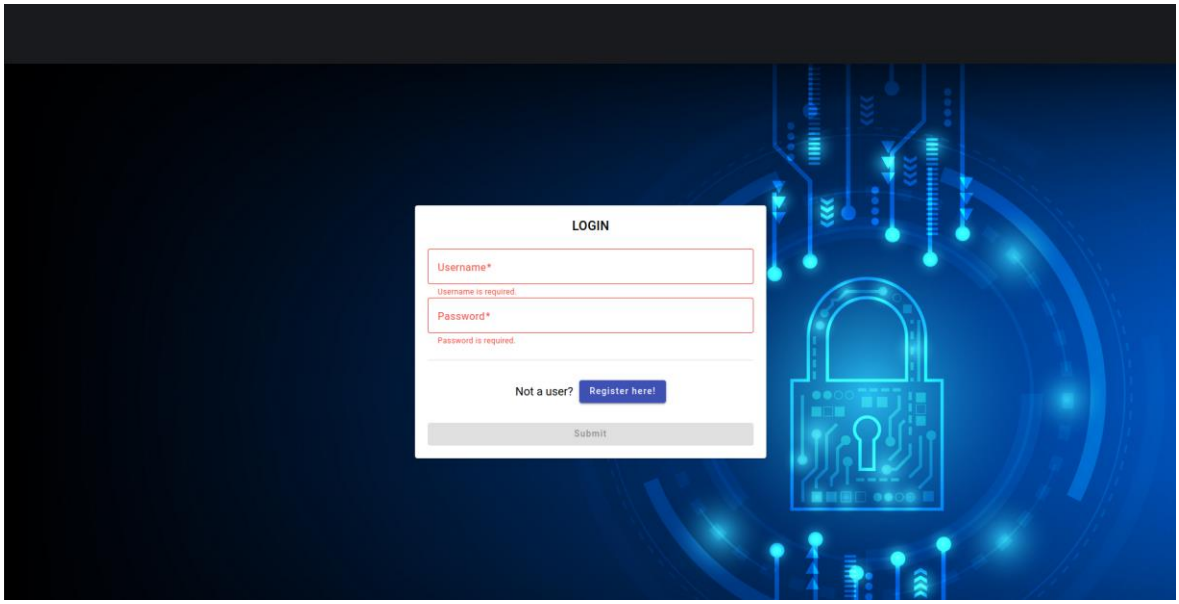
```
intializeForm(): void {
    this.registrationForm = this.fb.group({
      username : ["", [Validators.required,
Validators.maxLength(30), Validators.minLength(4)]],
```

```
      password : ["", [Validators.required,
Validators.minLength(6)]],
      passwordConfirm : ["", [Validators.required,
Validators.minLength(6)]]
    }
    ,
      {
        validator : registrationValidator
      }
    );
  }
...
function registrationValidator(control: AbstractControl): {
[key: string]: boolean } | null {
const username = control.get('username');
  const password = control.get('password');
  const passwordConfirm = control.get('passwordConfirm');

  const errors = {};
...
if (password.value !== passwordConfirm.value) {
    errors['passwordMismatch'] = true;
  }

  return Object.keys(errors).length ? errors : null;
}
```

Ispis 3.3 – Programski odsječak za validaciju obrasca registracije

Funkcija `registrationValidator` sakuplja svaki propust u zadovoljavanju uvjeta polja obrasca u listu `errors` u obliku poruka o pogreškama te ovisno o porukama koji se nalaze u listi, ispod odgovarajućih polja se ispisuju poruke pogreške na sučelju kako bi upozorile korisnika gdje griješi. U trenutku kada je lista `errors` prazna omogućen je gumb (engl. *button*) za slanje obrasca na obradu. U trenutku pritiska na gumb *Submit*, prikupljaju se podatci iz polja te se šalje zahtjev prema poslužitelju. U slučaju uspješne registracije korisnika se automatski preusmjerava na stranicu *Login*, a u slučaju pogreške interceptor ispisuje pogrešku na sučelju s odgovarajućom statusnom šifrom i porukom pogreške.

Dolaskom na stranicu *Login*, korisnik mora unijeti svoje vjerodajnice u obrazac koji je izgrađen na sličan način kao u prethodnom paragrafu. Stranica je prikazana na slici (Slika 3.2):



Slika 3.2 Prikaz *Login* komponente

Nakon zadovoljavanja uvjeta obrasca korisnik šalje podatke prema poslužitelju te se odvijaju procesi prikazani u ispisu (Ispis 3.4):

```
login(username: string, password: string) {
    return this._http
      .post(this.apiURL + '/users/rest-auth/login/', {
username, password })
      .pipe(
        tap((response: any) => {
          this._isLoggedIn$.next(true);
          this.setToken('accessToken', response.access);
          this.setToken('refreshToken', response.refresh);
        })
      );
  }
```

Ispis 3.4 – Programski odsječak za login korisnika

Korisnikove vjerodajnice se šalju na krajnju točku (engl. *endpoint*) za autentifikaciju i pri uspješnoj provjeri se pretplaćuje na odgovor poslužitelja koji šalje objekt *response* s dva atributa: *access* i *refresh* koji sadrže generirane tokene. Oni se spremaju u lokalnu pohranu korisnikova preglednika te se automatski preusmjerava na stranicu *System*. U slučaju korisnikova odjavljivanja se šalje zahtjev na poslužitelja koji briše korisnikove tokene te se

oni istovremeno brišu iz njegove lokalne pohrane u pregledniku što je prikazano u ispisu (Ispis 3.5):

```
public logout() {
    return this._http
      .post(this.apiURL + '/users/rest-auth/logout/', {})
      .pipe(finalize(() => localStorage.clear()));
  }
```

Ispis 3.5 – Programski odsječak za logout korisnika

Naposljetku se korisnik preusmjerava na stranicu *Login*.

3.4. Usmjeravanje i Route Guard

Sve rute su definirane u datoteci *app.routing-module.ts* uz pomoć korištenja metodologije *lazy-loading*. Umjesto da se svi moduli i komponente učitavaju odjednom pri pokretanju aplikacije, *lazy-loading* omogućuje da se dijelovi aplikacije učitavaju samo kada su stvarno potrebni. Kada se koristi *lazy-loading*, moduli se dijele na manje dijelove ili „chunkove". Svaki *chunk* predstavlja određenu funkcionalnosti ili skup komponenti. Kada korisnik pristupi određenoj ruti ili izvrši neku akciju koja zahtijeva određeni *chunk*, taj se *chunk* učitava samo tada.

Komponente se povezuju u module koji obavljaju međusobnu komunikaciju i svoju ulogu u preusmjeravanju svojih dječjih ruta (engl. *children routes*). Na primjer, Modul *System* sadrži komponentnu S*ystem* i komponentnu *System-details*. Time se postiže veća brzina učitavanja prikaza sučelja jer se učitava samo ona stranica na kojoj se korisnik trenutno nalazi, a preostale komponente se učitavaju u pozadini. Podjelom na module se također štedi na prostoru jer se biblioteke uvoze (engl. *import*) samo po potrebnim modulima umjesto kroz cijelu aplikaciju.

Bitno je spomenuti kako je postavljena i funkcionalnost Angular Route Guard-a [33], koji se aktivira pri preusmjeravanju na zadane rute kao što je prikazano u ispisu (Ispis 3.6):

```
canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<boolean | UrlTree> {
    return this.isLoggedIn().pipe(
```

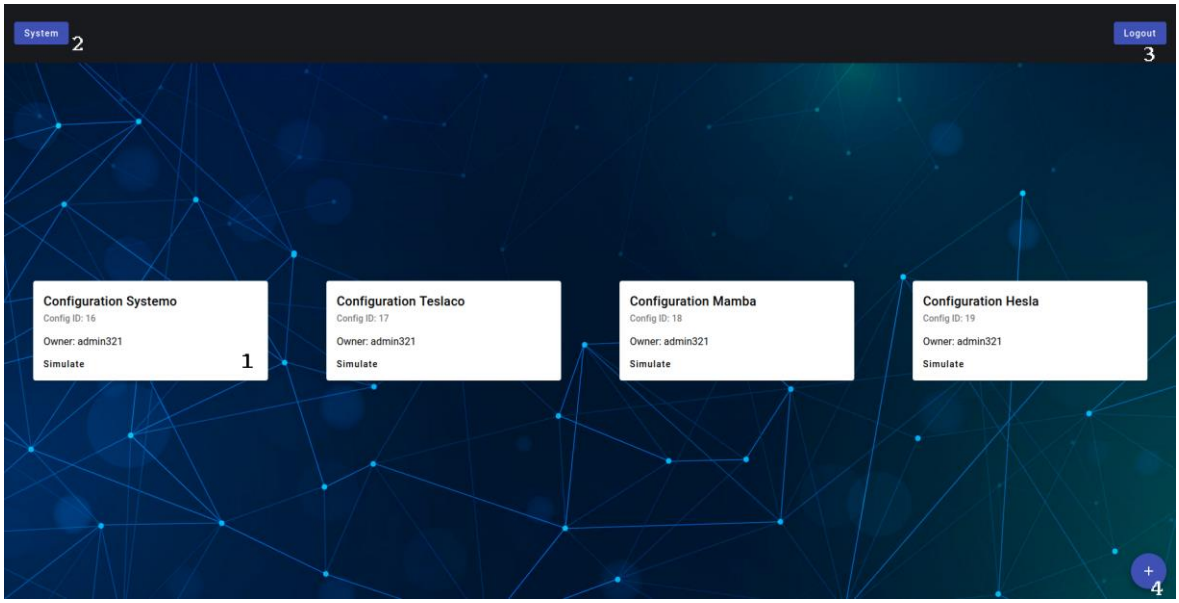
```
      map(
        (isLoggedIn) => {
          if (!isLoggedIn) {
            return this._router.parseUrl('/login');
          } else {
            return true;
          }
        }
      )
    );
  }
userCheck(): Observable<boolean> {
    return this._http.get(this.apiURL + '/users/rest-
auth/user/').pipe(
      map(() => true),
      catchError((error) => {
        // If the error status is 401, return an Observable
of false
        return of(false);
      })
    );
  }
```

Ispis 3.6 – Programski odsječak za provjeru korisnika ovisno o trenutnoj ruti

Kada korisnik želi pristupi *System* ili *System-details* komponenti, aktivira se funkcija `canActivate` koja provjerava je li korisnik pravilno ulogiran u sučelje, odnosno da li ima važeće tokene. Kod neuspješne provjere preusmjerava korisnika na *Login* komponentu te ne prikazuje nikakav osjetljivi sadržaj korisnika. Ispravnim pristupom na rutu, odobrava korisnika te mu prikazuje sadržaj tih komponenti koje sadrže glavne funkcionalnosti sučelja.

4. Implementacija glavnih funkcionalnih zahtjeva

Nakon uspješne validacije korisnika on dolazi na komponentu *System* te se pojavljuje prikaz sa slike (Slika 4.1):

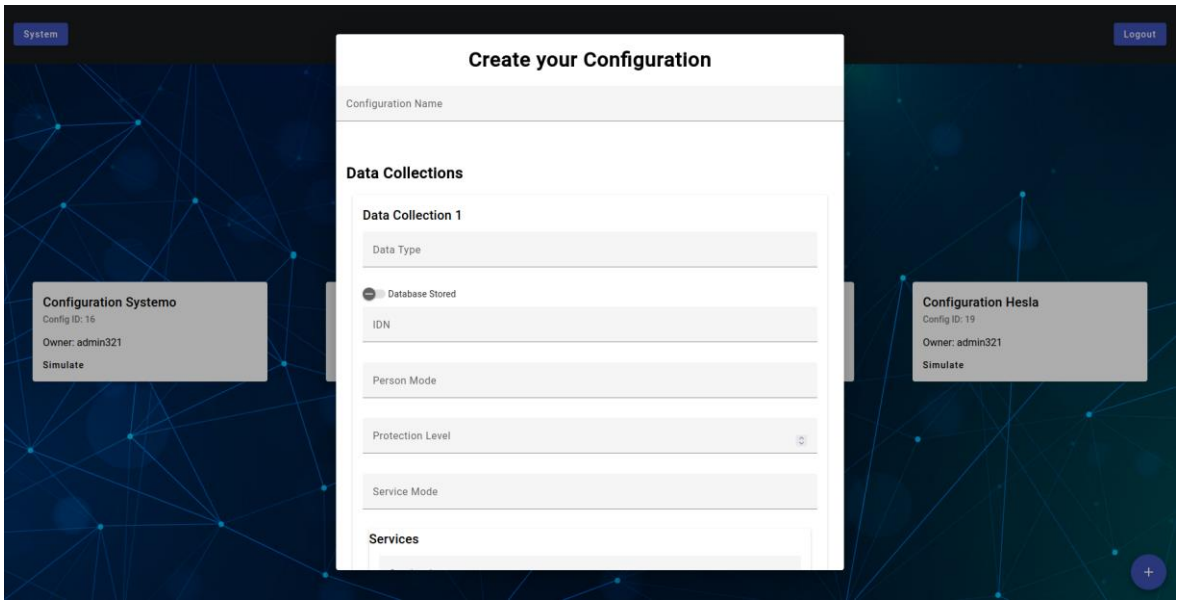


Slika 4.1 Prikaz *System* komponente

Korisnik dobiva jednostavan pregled svih prijašnje unesenih konfiguracija. Na svakoj kartici, označene oznakom 1, može uvidjeti ime konfiguracije, identifikator te svoje korisničko ime. Unos nove konfiguracije se odvija klikom miša na gumb u donje desnom kutu označen oznakom 4, te se otvara obrazac koji je opisan u nastavku. Također, korisnik se može vratiti na ovu rutu u bilo kojem trenutku klikom na gumb *System* označen oznakom 2 te se može odjaviti iz aplikacije klikom na gumb *Logout* označen oznakom 3.

4.1. Unos konfiguracije

Kako bi korisnik mogao unijeti predefinirane unose objašnjene u prvom poglavlju potrebno je izraditi adekvatan unos podataka. S obzirom na to da je unos dinamičan u smislu broja zahtjeva, koristi se svojstvo reaktivnih obrazaca dinamičnog generiranja polja za unos. Obrazac za unos korisnikove konfiguracije je prikazan na slici (Slika 4.2):



Slika 4.2 Prikaz obrasca za unos konfiguracije

Od korisnika se zahtjeva da unese ime svoje konfiguracije te ispuni potrebne podatke u formatima teksta, broja ili istinitosti ovisno o tipu polja tj. podataka. Obrazac se inicijalizira u ispisu (Ispis 4.1):

```
export class AddConfigDialogueComponent implements OnInit {
  dataForm: FormGroup;
  title : string;
  constructor(
    private fb: FormBuilder,
    public dialogRef:
MatDialogRef<AddConfigDialogueComponent>
  ) {
    this.dataForm = this.fb.group({
      data_collections: this.fb.array([
        this.createDataCollection()
      ]),
      employee_groups: this.fb.array([
        this.createEmployeeGroup()
      ]),
      network_policies: this.fb.array([
        this.createNetworkPolicy()
      ]),
      provided_external_services: this.fb.array([
        this.createProvidedExternalService()
      ]),
    });
```

```
      this.title = '';
    }
```

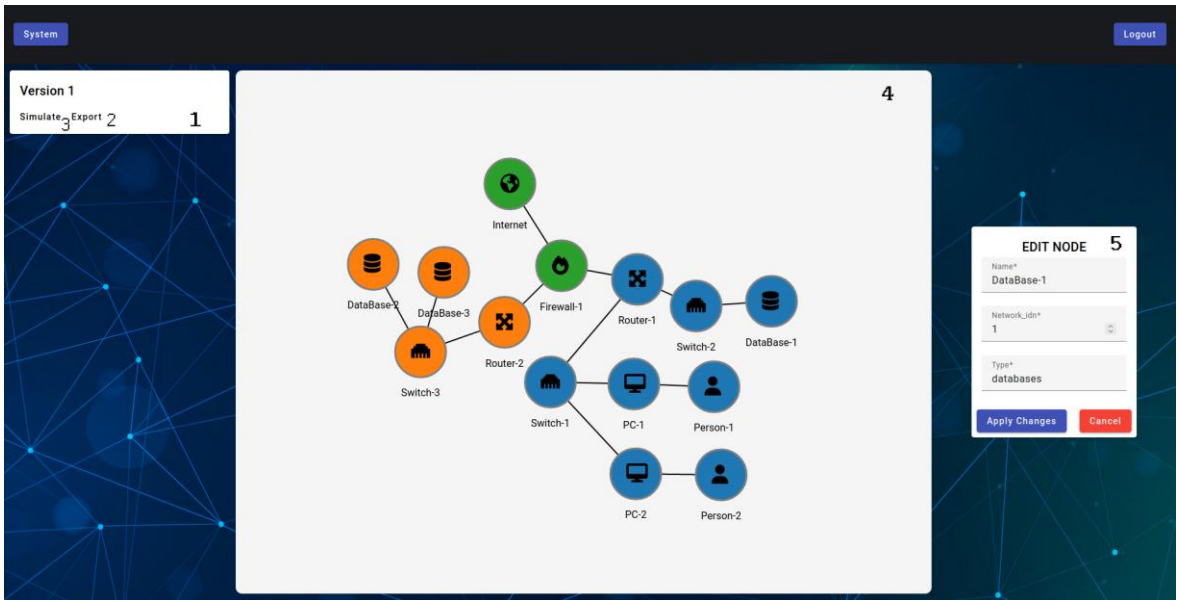
Ispis 4.1 – Programski odsječak za inicijaliziranje obrasca za unos konfiguracije

Koristeći reaktivne obrasce omogućava se dinamično nadograđivanje polja obrasca koje se dodaju pozivom funkcija kao što je `createDataCollection()` Te se nadodaje u listu atributa tipa `fb.array`. Te pomoćne funkcije se koriste kako bi korisnik sam mogao nadograđivati obrazac. Na primjer, pritiskom na gumb *Add data collection*, ponovno se poziva ista pomoćna funkcija `createDataCollection()`.

Nakon što korisnik završi s ispunjavanjem obrasca, šalje se zahtjev na poslužitelj i osvježava se prikaz svih korisnikovih konfiguracija kako bi ažurirali prikaz koji uključuje obrazac koji je maloprije ispunjavao. Jednom kad se korisnik odluči da želi vidjeti vizualizaciju generiranog sustava, klikom na gumb *Simulate*, usmjerava se na komponentu *System-details* što je opisano u nastavku.

4.2. Formatiranje i vizualizacija podataka

S korisnikovim izborom sustava s kojem želi interaktirati, korisnik dolazi na komponentu *System-details* čiji je sadržaj podijeljen na tri dijela: lijevi prikaz sadrži popis svih verzija sustava u obliku liste kartica gdje je kartica označena oznakom 1, srednji prikaz koji je označen oznakom 4 sadrži samu vizualizaciju sustava trenutno odabrane verzije i desni prikaz nudi opciju pregleda postavka odabranog čvora te njegove izmjene pomoću reaktivnog obrasca označenog oznakom 5 što je prikazano na slici (Slika 4.3):



Slika 4.3 Prikaz *System-details* komponente

Tijekom učitavanja stranice odvijaju se različiti procesi koji se mogu vidjeti u sljedećem ispisu (Ispis 4.2):

```
constructor(
    private dialog : MatDialog,
    private fb : FormBuilder,
    private route : ActivatedRoute,
    private _systemService : SystemService
  ){
    library.add(fas, far, fab);
    this.typeMap = {
      'person': '\uf007',
      'computers' : '\uf108',
      'databases' : '\uf1c0',
      'switch' : '\uf796',
      'routers': '\uf31e',
      'firewalls' : '\uf06d',
      'internet' : '\uf0ac'
    };
    this.graph = {
      nodes : [],
      links : []
    };
    this.nodeRadius = 40;
  }
```

```
  ngOnInit(): void {
    //this.initializePage();
    this.routeSub = this.route.params.subscribe((params :
any) => {
      this.id = params['id'];

this._systemService.getEditedConfigurations(this.id).subscrib
e((result : any)=>{
        this.versions.push(result);
      })
    });
    this.intitializeForm();

  }
  ngAfterViewInit() {
    this.inputGraphData();
  }
```

Ispis 4.2 – Program odsječak s funkcijama inicijalizacije komponente

Prvo se definiraju pomoćne varijable kao što su `typeMap` preko koje se definiraju ikone čvorova ovisno o tipu čvora, te se definira prazni objekt pomoću varijable `graph` koji sadrži atribute `nodes` i `links`. Zatim se poziva funkcija `ngOnInit()` koja dohvaća identifikator sustava preko identifikatora varijable rute te se ona koristi pri dohvaćaju informacija o sustavu.

U funkciji `afterViewInit()` se poziva funkcija `inputGraphData()` koja je zaslužna za ubacivanje podataka o sustavu te je prikazana u ispisu (Ispis 4.3):

```
this._systemService.getEditedConfigurations(this.id).subscrib
e((configurations : any) => {
      this.currentVersion = configurations;
      this.graph.nodes =
configurations.api1_response_json_edit.nodes;
      this.graph.links =
configurations.api1_response_json_edit.links;
      this.simulation = this.createChart();
    });
```

Ispis 4.3 – Programski odsječak za dohvat podataka o sustavu

Informacije o sustavu s meta podacima se spremaju u varijablu `currentVersion` te se popunjavaju informacije o čvorovima i linkovima kako bi se naposljetku pozvala funkcija `createChart()` koja generira vizualnu simulaciju putem D3 Force Layout modula.

Unutar funkcije `createChart()` se kreće s definiranjem SVG elementa u kojem će se odvijati simulacija, što je prikazano u ispisu (Ispis 4.4):

```
let mainPanel = d3.select('.main-panel');
let mainPanelWidth = +mainPanel.style('width').replace('px',
'');
let mainPanelHeight =
+mainPanel.style('height').replace('px', '');
const svg = d3.select('svg');
const centerX = mainPanelWidth / 2;
const centerY = mainPanelHeight / 2;
```

Ispis 4.4 – Programski odsječak za definiranje prikaza grafa

Definiraju se općenita svojstva SVG elementa poput njegove visine i duljine te njegove koordinate centra. Zatim je potrebno definirati čvorove i njihove poveznice što je prikazano u ispisu (Ispis 4.5):

```
var nodes = this.graph.nodes;
    var linksSimple = this.graph.links;
    var links = [];
    linksSimple.forEach(function (linkObj) {
      var sourceNode = nodes.filter(function (n) {
          return n.idn === linkObj.source;
        })[0],
        targetNode = nodes.filter(function (n) {
          return n.idn === linkObj.target;
        })[0];
      links.push({
        source: sourceNode,
        target: targetNode
      });
    });
```

Ispis 4.5 – Programski odsječak za prilagodbu čvorova i poveznica

Na tvorničkim postavkama D3 force layout-a, poveznice se povezuju preko indeksa čvorova što nije praktično u ovom slučaju. Varijabla `linksSimple` transformira svoje podatke u varijablu `links` koje se povezuju putem njihovog identifikatora što je ujedno i

njihov jedinstveni identifikator te se time zadovoljavaju zahtjevi konfiguracije poveznica u D3.

Nakon inicijalizacije potrebnih podataka slijedi korak definiranja simulacije što je prikazano u sljedećem ispisu (Ispis 4.6):

```
const forceNode = d3.forceManyBody().strength(-200);
const forceLink = d3.forceLink({ nodes, links
}).distance(50);
var simulation = d3
   .forceSimulation(nodes)
   .force('link', forceLink)
   .force('charge', forceNode)
   .force('collide', d3.forceCollide(50))
   .force('x', d3.forceX())
   .force('y', d3.forceY())
   .force('center', d3.forceCenter(centerX, centerY));
```

Ispis 4.6 – Programski odsječak za definiranje simulacije grafa

Pri definiranju simulacije potrebno je definirati sve moguće sile koje se odvijaju te im dodijeliti vrijednosti preko kojih računaju iznos svojih sila. Detaljnu dokumentaciju, definiciju i parametre pomoćnih funkcija force layout modula se može vidjeti u [12], dok će se u nastavku dati kratko objašnjenje.

Prvo se u simulaciju ugrađuju čvorovi funkcijom `forceSimulation(nodes)`, te se dodaje sila s atributom `'link'` koja je definirana u varijabli `forceLink`. Ona definira silu privlačnosti između povezanih čvorova kako bi oni ostali na razumnoj distanci jedno od drugog. U nastavku se definira sila atributa `'charge'` putem funkcije `forceNode` koja je zaslužna za odbijanje čvorova koji su povezani međusobno u slučaju kad su na bliskoj međusobnoj udaljenosti. Nakon toga se definira sila atributa `'collide'` koja je zaslužna za odbijanje nepovezanih čvorova. Ta sila je ključna kako se čvorovi tijekom pomicanja ne bi preklapali međusobno. Konačno sila s atributom `'center'` definira gravitacijsku silu koja djeluje na sve čvorove u simulaciji i privlači ih prema centru simulacijskog prostora. S njom se osigurava da se čvorovi neće udaljiti od centra u nedogled.

Kombinacijom svih definiranih sila, u pozadini se odvijaju razni optimirani izračuni sila te se pokretanjem simulacije, nakon kratkog vremena, postiže potpuni balans sila. U trenutku

postizanja tog balansa svi čvorovi miruju što definira temeljnu situaciju simulacije u kojoj se može započeti interaktiranje.

Definiranje čvorova, poveznica i sila je ključan korak simulacije, ali je potrebno definirati i grafičke elemente koji će se iscrtavati na sučelju te ih povezati s varijablama čvorova i poveznica. U ispisu (Ispis 4.7) uviđamo stvaranje poveznica:

```
var lines = svg
      .append('g')
      .selectAll('line')
      .data(links)
      .enter()
      .append('line')
      .attr('stroke-width', '2px')
      .attr('stroke', 'black');
```

Ispis 4.7 – Programski odsječak za definiranje poveznica

Funkcijom `append('g')` se definira grupa elemenata koji će predstavljati poveznice te sa `selectAll('line')` se koristi predefinirani atribut SVG alata koji opisuje da će biti nacrtana ravna crta na sučelju između čvorova. S `data(links)` se povezuje grafički element poveznica s podatkovnim dijelom poveznica te s funkcijama `enter()` i `append()` se napokon iscrtava grafički element. Na sličan način se definiraju čvorovi čiji je prikaz u ispisu (Ispis 4.8):

```
var node = svg
      .append('g')
      .selectAll('g')
      .data(nodes)
      .enter()
      .append('g')
      .attr('class', 'node');

var circles = node
      .append('circle')
      .attr('r', this.nodeRadius)
      .attr('cx', (d) => d.x)
      .attr('cy', (d) => d.y)
      .attr('fill', (d) => color(d.network_idn))
      .attr('stroke', 'gray')
      .attr('stroke-width', '3px')
```

```
        .attr('opacity', 1)
```

Ispis 4.8 – Programski odsječak za definiranje čvorova

Jedina bitna razlika je u tome što čvorovi mogu sadržavati više grafičkih elemenata koji su povezani s njima te su se takvi dijelovi odvojili u pomoćne varijable kao što je `circles`. Bitno je uočiti D3 mogućnost u korištenju lambda funkcija [34] pri definiciji atributa kao što je prikazano u `attr('fill', (d) => color(d.network_idn))`. D3 zamjenjuje klasični način programiranja tako što se orijentira na pristup definicije željenog rezultata. Funkcijom `color(d.network_idn)`, D3 će u pozadini implementirati boju kruga za svaki čvor u skupu podataka ovisno o `networkd_idn` vrijednosti tog čvora. Na sličan način su definirani grafički prikazi ikona na čvorovima ovisno o tipu entiteta čvora, *tooltip* koji prikazuje osnovne informacije čvora te tekst ispod svakog čvora koji prikazuje vrijednost atributa *name* čvora, odnosno naziv čvora.

Nakon povezivanja podataka i definiranja grafičkih elemenata simulacije preostaje funkcionalni aspekt simulacije što je opisano u nastavku.

4.3. Funkcionalnosti simulacije

Pri interakciji čvorova ulogu izračuna sila obavlja sama simulacija, ali se grafički prikaz čvorova ne mijenja. Kako bi grafički prikaz konstantno pratio položaje čvorova nastalih izračunima sila potrebno je sinkronizirati čvorove s njihovim grafičkim prikazom što je postignuto u ispisu (Ispis 4.9):

```
simulation.on('tick', () => {
      circles.attr('cx', (node) => node.x).attr('cy', (node)
=> node.y);
      circles
        .attr('r', this.nodeRadius)
        .attr('fill', (d) => color(d.network_idn));
      icons
        .attr('x', (node) => node.x - 15)
        .attr('y', (node) => node.y + 10)
        .text(d => this.typeMap[d.type]);
      textNodeTitle
        .text((d) => d.name)
        .attr('x', (node) => node.x)
        .attr('y', (node) => node.y + 40 + 20);
```

```
    lines
      .attr('x1', (link) => link.source.x)
      .attr('y1', (link) => link.source.y)
      .attr('x2', (link) => link.target.x)
      .attr('y2', (link) => link.target.y);
  });
```

Ispis 4.9 – Programski odsječak za ažuriranje simulacije

U pomoćnoj funkciji atributa `'tick'`, poziva se lambda funkcija koja je zaslužna za ponovno definiranje položaja svih grafičkih elemenata u simulaciji te se ona izvršava pri bilo kojoj promjeni u koordinatama bilo kojeg člana simulacije. Dodatno, u pripremi za prikaz velikog broja čvorova, izgrađena je i funkcionalnost zumiranja (engl. *zoom*) što je prikazano u ispisu (Ispis 4.10):

```
svg.call(d3.zoom().on('zoom', zoomed));
function zoomed(event){
      const transform = event.transform;
      svg.selectAll('g.node')
      .attr('transform', transform);
      svg.selectAll('line')
        .attr('stroke-width', (d) => 2 + 2*transform.k)
        .attr("transform", transform);
      svg.selectAll('text')
        .style('font-size', (d) => 30 / transform.k);
      }
```

Ispis 4.20 – Programski odsječak za definiranje svojstva zumiranja grafa

D3 koristi koeficijent `k` pri izračunu svih transformacija članova simulacije. U pozadini se korisniku samo priviđa da on zumira, dok zapravo svi elementi povećavaju ili smanjuju svoju veličinu. S ovom opcijom je dodatno omogućeno i pomicanje po grafu.

Što se tiče opcije razmještanja čvorova u simulaciji, korisniku se omogućuje opcija kliktanja na čvor te držanjem tog klika, može proizvoljno razmještati odabrani čvor po simulaciji. Opisana funkcionalnost se nalazi u ispisu (Ispis 4.11):

```
function drag(simulation) {
      function dragstarted(event, d) {
        if (!event.active)
simulation.alphaTarget(0.003).restart();
        d.fx = d.x;
        d.fy = d.y;
```

```
    }

    function dragged(event, d) {
      d.fx = event.x;
      d.fy = event.y;
    }

    function dragended(event, d) {
      if (!event.active) simulation.alphaTarget(0);
      d.fx = null;
      d.fy = null;
    }

    return d3
      .drag()
      .on('start', dragstarted)
      .on('drag', dragged)
      .on('end', dragended);
  }
```

Ispis 4.11 – Programski odsječak za definiranje funkcija interakcija s mišem korisnika

Bitno je naglasiti da tijekom korisnikovog pomicanja čvorova simulacijski izračuni moraju usporiti radi uštedivanja resursa što se postiže funkcijom `simulation.alphaTarget(0.003)`. Pri završetku pomicanja simulacija se ponovno pokreće te se vraća na početnu brzinu obrade. Također je potrebno povezati reaktivni obrazac s D3 događajem (engl. *event*) kada korisnik želi mijenjati postavke čvora u obrascu označenim oznakom 5 na slici (Slika 4.3), što se događa u sljedećem ispisu:

```
function handleMouseClick(event, datum) {
    self.selectedNode = datum;
    self.editForm();
}
```

Iz argumenta `datum` se izvlače sve informacije o odabranom čvoru te se postavlja globalna varijable `selectedNode` s tim vrijednostima što aktivira prikazivanje obrasca za uređivanje. Pokreće se funkcija `editForm` koja ispunjava obrazac s trenutnim vrijednostima čvora koje korisnik napokon može uređivati. Nakon završetka korisnikovih izmjena i pritiskom na gumb *Apply changes* unutar obrasca označenog oznakom 5 na slici (Slika 4.3), aktivira se funkcija `applyChanges`, što je prikazano u sljedećem ispisu (Ispis 4.12):

```
applyChanges() : void {
    if (this.selectedNode) {
      Object.assign(this.selectedNode, this.nodeForm.value);
      this.graph.nodes = this.graph.nodes.map(node =>
node.idn === this.selectedNode.idn ? this.selectedNode :
node);
      this.nodeForm.reset();
      this.selectedNode = null;
      this.simulation.nodes(this.graph.nodes);
      this.simulation.alpha(0.001).restart();
    }
  }
```

Ispis 4.32 – Programski odsječak za postavljanje novih podataka čvora

Prvo se radi duboka kopija (engl. *deep copy*) vrijednosti u varijablu `selectedNode` te se pronalazi izmijenjeni čvor u listi svih čvorova po njegovom identifikatoru i ažuriraju se njegove vrijednosti. Nakon toga se obrazac resetira, tj. prazni, kao i globalna varijabla `selectedNode`. Naposljetku se u simulaciju ugrađuje izmijenjena lista čvorova te se simulacija ponovno pokreće kako bi se prikaz čvora na ekranu odmah ažurirao.

Konačno, korisnik nakon provedenih izmjena ima opciju stvaranja nove verzije pritiskom na gumb *Save changes* gdje se njegova trenutno izmijenjena verzija šalje na poslužitelj. Uz to je omogućeno preuzimanje verzije sustava u formatu zip datoteke [35] klikom na gumb *Export* u kartici verzije, što je označeno oznakom 3 na slici (Slika 4.3) te pokretanje nove simulacije klikom na gumb *Simulate* koji je označen oznakom 2 na slici (Slika 4.3).

5. Diskusija

Nakon izrade sučelja, može se zaključiti kako su početni uvijeti zadovoljeni te je upotreba sučelja adekvatna, jednostavna i fluidna. Korisnik vrlo kratkim postupkom može izraditi vlastiti račun, unijeti svoju konfiguraciju te vidjeti rezultat generatora. Vizualno minimalistički pristup se pokazao uspješnim gdje je fokus stavljen na preglednost i lakše shvaćanje strukture sustava umjesto da se korisnika teretilo obujmom informacija, tranzicijama i animacijama tijekom pregleda.

Naravno, na kraju se uviđaju brojni problemi i ograničenja sučelja koji su nastali ili tijekom izrade ili neovisno o izradi zbog nedostatka iskustva i znanja autora. Većina vremena koja je planirana za diplomski rad je uložena na učenje ključne biblioteke D3.js.

D3.js uistinu ima veliki broj resursa, uputa i jedno od najkvalitetnijih dokumentacija, ali čim se pokušavalo izraditi nešto specifično ili kompliciranо za potrebe ovog rada, broj resursa se uvelike smanjio i mnoge stvari su ostale nedorečene. Otprilike polovica D3 implementacije je izrađeno pristupom pokušaja i pogreške te dubokim traženjem grešaka u kôdu (engl. *debugging*), specifično u DOM elementima kojima upravlja D3.

Tijekom procesa sinkronizacije Angular i D3.js funkcija pojavio se problem dijeljenja konteksta između D3 događaja te konteksta komponente u kojoj se simulacija nalazi. Također, nadzire se općeniti problem intenzivnog računanja sila u simulatoru, neovisno o izboru alata za vizualizaciju, gdje performanse prikaza počinju naglo padati između stotinjak i tisuću čvorova te nije prikladno unositi konfiguracije namijenjene za izradu sustava koje bi koristile ogromne organizacije.

Nadalje, jedan od najvećih problema je nedostatak kvalitetnog transformatora između izlaza generatora i prosljeđivanja podataka u simulator na pravilan način. Simulator zahtjeva sve informacije o čvorovima i poveznicama u obliku liste objekata čvorova i liste objekata poveznica s predefiniranim atributima, što nažalost nije slučaj u izlazu generatora gdje su informacije o čvorovima i poveznicama raspršene na različite atribute u zapisu izlaza generatora. Bez tog transformatora nije bilo moguće testirati sučelje s izlazima generatora te postoji velika mogućnost da su određene greške još uvijek prisutne koje se moraju u budućnosti otkloniti.

6. Budući rad

S dobro razvijenim temeljima sučelja, preostaje puno prostora za nadogradnju. D3.js biblioteka nudi široki izbor animacija, prijelaza i transformacija koje mogu učiniti simulaciju puno interaktivnijom i zanimljivom. Dodatno, mogu se mijenjati informacije koje se žele prikazati korisniku i isključiti one koje korisnik ne smije mijenjati.

S obzirom na to da je sučelje napravljeno s ciljem modularnosti, uz kratke promjene se može koristiti u simulaciji sustava koji su generirani od različitih generatora dok god se format ulaska podataka u simulaciju ne mijenja.

Još jedno proširenje se očituje u izradi dodatne razine sigurnosti, tj. uvođenjem pojma grupe korisnika koji pripadaju istoj organizaciji. Ti korisnici bi mogli dijeliti svoje konfiguracije i sustave međusobno u svakodnevnom radu.

Konačno, sučelju se nije posvetilo puno vremena s perspektive dizajna te tu ostaju mogućnosti poput povećavanja ergonomičnosti te uvođenje skalabilnost sučelja za različite veličine ekrana, prvenstveno rezolucije mobilnih korisnika.

Zaključak

U ovom radu je uspješno izrađeno sučelje aplikacije korištenjem aktualnih besplatnih alata i tehnologija uz praćenje najboljih praksi izrade. Omogućuje se jednostavan unos konfiguracije željenog sustava te vizualni prikaz generiranih informacijskih sustava. Omogućen je sljedeći korak u izradi simuliranih informacijskih sustava koji se mogu koristiti u daljnje svrhe edukacije, treniranja i uporabe u drugim alatima.

Korisniku je pružen prostor za vizualizaciju s dobrim performansama gdje korisnik ima potpunu kontrolu nad generiranim sustavom nad kojim može raditi potrebne izmjene kako bi zadovoljio sve specifične zahtjeve i što bliže približio sustav stvarnosti. Sučelje se može koristiti ne samo s opisanim generatorom, već i s bilo kojim tehnologijama dok god zadovoljavaju pravila ulaza simulatora.

Nažalost, najveći propust u izradi je bilo skoro nepostojeće testiranje aplikacije što je sigurno ostavilo par propusta. Smatra se da se moglo izgraditi kvalitetnije sučelje provođenjem testiranja i korisničkog ispitivanja kako bi se povratnom informacijom sučelje što bolje prilagodilo korisnikovim potrebama.

Nakon završetka projekta, uz svo skupljeno znanje kroz rad i istraživanje, nameće se mišljenje kako je ovaj projekt mogao biti izgrađen u drugim radnim okvirima koji su kompatibilniji s D3 bibliotekom. Također, tijekom izrade su se pojavljivale mnoge izmjene zbog rasprava u vezi načina povezivanja sučelja s poslužiteljem što je odužilo izradu. Takvi problemi su se mogli izbjeći detaljnijim istraživanjem mogućnosti i ograničenja radnih okvira, tehnologija i alata koji su se koristili pri izradi. Naravno, takvi problemi se tek uočavaju tijekom izrade projekta te su ponekad teški za predvidjeti prije početka same implementacije.

Literatura

[1] Lawinsider, *Information Technology Systems definition*, Lawinsider. Poveznica: https://www.lawinsider.com/dictionary/information-technology-systems; pristupljeno: 28. svibnja 2023.

[2] Javapoint, *Introduction to Information System*, Javapoint. Poveznica: https://www.javatpoint.com/cyber-security-information-system-introduction; pristupljeno: 17. lipnja 2023.

[3] Kovačević I., Groš S., Đerek A. *Automatically generating models of IT systems*. 10. izdanje. IEEE Access 10 (2022): 13536-13554.

[4] MDN Contributors, *Working with JSON*, MDN Web Docs, (2023, svibanj). Poveznica: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON; pristupljeno: 25. svibnja 2023.

[5] Amazon AWS, *What Is An API (Application Programming Interface)?*, Amazon AWS. Poveznica: https://aws.amazon.com/what-is/api/; pristupljeno: 16. lipnja 2023.

[6] Bostock M., Ogievetsky V., Heer J., *D3: Data-Driven Documents*. 1. izdanje. IEEE Trans. Visualization \& Comp. Graphics (Proc. InfoVis): 2011.

[7] Bostock M., *For Protovis Users*, Web Archive, (2012). Poveznica: https://web.archive.org/web/20120806072124/http://mbostock.github.com/d3/tutorial/protovis.html;pristupljeno: 31. svibnja 2023.

[8] Gill N.S., *Introduction to D3.Js Library and its Use Cases*, Xenostack (2022, veljača). Poveznica: https://www.xenonstack.com/blog/d3js; pristupljeno: 19. lipnja 2023.

[9] Bostock M., *d3*, Github (2016, lipanj). Poveznica: https://github.com/d3/d3/releases/tag/v4.0.0; pristupljeno: 23. svibnja 2023.

[10] MDN Contributors, *Introduction to the DOM*, MDN Web Docs (2023, svibanj). Poveznica: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction; pristupljeno: 2. lipnja 2023.

[11] Tutorials Teacher, *Data Binding in D3*, Tutorials Teacher. Poveznica: https://www.tutorialsteacher.com/d3js/data-binding-in-d3js; pristupljeno: 15. svibnja 2023.

[12] Bostock M., *D3 API Reference*, Github. Poveznica: https://github.com/d3/d3/blob/main/API.md; pristupljeno: 13. lipnja 2023.

[13] Bostock M., *d3-force*, Github. Poveznica: https://github.com/d3/d3-force; pristupljeno: 13. lipnja 2023.

[14] Wikipedia, Force-directed graph drawing, Wikipedia. Poveznica: https://en.wikipedia.org/wiki/Force-directed_graph_drawing; pristupljeno: 11. lipnja 2023.

[15] Intapiuser, *D3 Force Directed Network Diagram*, Sisense (2023, ožujak). Poveznica: https://community.sisense.com/t5/knowledge/d3-force-directed-network-diagram/ta-p/9108; pristupljeno: 9. lipnja 2023.

[16] Angular, *Official Angular Website*, Angular. Poveznica: https://angular.io/; pristupljeno: 15. lipnja 2023.

[17] Gavigan D., *The History of Angular*, Medium. Poveznica: https://medium.com/the-startup-lab-blog/the-history-of-angular-3e36f7e828c7, pristupljeno: 20. svibnja 2023.

[18] MDN Contributors, *MVC*, MDN Web Docs (2023, lipanj). Poveznica: https://developer.mozilla.org/en-US/docs/Glossary/MVC; pristupljeno: 21. lipnja 2023.

[19] Gallardo E. G., *What is MVVM Architecture?*, Builtin (2023, siječanj). Poveznica: https://builtin.com/software-engineering-perspectives/mvvm-architecture; pristupljeno: 16. lipnja 2023.

[20] Adams D., *Learn TypeScript – The Ultimate Beginners Guide*, Free Code Camp. Poveznica: https://www.freecodecamp.org/news/learn-typescript-beginners-guide/; pristupljeno: 25. svibnja 2023.

[21] Stack Overflow, *2020 Developer Survey*, *Stack Overflow* (2020, veljača). Poveznica: https://insights.stackoverflow.com/survey/2020#technology-web-frameworks-all-respondents2; pristupljeno: 2. lipnja 2023.

[22] WireDelta, *10 Most Popular Angular Websites of 2020*, WireDelta (2020, srpanj). Poveznica: https://wiredelta.com/10-most-popular-angular-websites-of-2020/, pristupljeno: 13. lipnja 2023.

[23] Ably, *What is Pub/Sub? The Publish/Subscribe model explained*, Ably (2023, svibanj). Poveznica: https://ably.com/topic/pub-sub; pristupljeno: 7. lipnja 2023.

[24] Angular, *Lazy-loading feature modules*, Angular (2022, svibanj). Poveznica: https://angular.io/guide/lazy-loading-ngmodules; pristupljeno: 15. lipnja 2023.

[25] Angular, *CLI Overview and Command Reference*, Angular. Poveznica: https://angular.io/cli; pristupljeno: 28. svibnja 2023.

[26] Angular, *Angular Material*, Angular. Poveznica: https://material.angular.io/; pristupljeno: 17. lipnja 2023.

[27] Angular, *Reactive forms*, Angular (2022, veljača). Poveznica: https://angular.io/guide/reactive-forms; pristupljeno: 31. svibnja 2023.

[28] Madole R., *Font-Awesome*, Github. Poveznica: https://github.com/FortAwesome/Font-Awesome; pristupljeno: 13. lipnja 2023.

[29] OneLogin, *Authentication vs. Authorization*, OneLogin. Poveznica: https://www.onelogin.com/learn/authentication-vs-authorization; pristupljeno: 6. lipnja 2023.

[30] Auth0, *JSON Web Tokens*, Auth0. Poveznica: https://auth0.com/docs/secure/tokens/json-web-tokens; pristupljeno: 19. svibnja 2023.

[31] Paredes D., *Angular Basics: Intro to Angular Interceptors*, Telerik (2022, prosinac). Poveznica: https://www.telerik.com/blogs/angular-basics-intro-interceptors; pristupljeno: 21. svibnja 2023.

[32] Hussain A., *Custom Form Validators*, CodeCraft (2018, prosinac). Poveznica: https://codecraft.tv/courses/angular/advanced-topics/basic-custom-validators/; pristupljeno: 12. lipnja 2023.

[33] Chenkie R., *Angular Authentication: Using Route Guards*, Medium (2017, srpanj). Poveznica: https://medium.com/@ryanchenkie_40935/angular-authentication-using-route-guards-bf7a4ca13ae3; pristupljeno: 24. svibnja 2023.

[34] MDN Contributors, *Arrow function expressions*, MDN Web Docs. Poveznica: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions; pristupljeno: 31. svibnja 2023.

[35] Dropbox, *What is a ZIP file?*, Dropbox. Poveznica: https://experience.dropbox.com/resources/what-is-a-zip-file; pristupljeno: 19. lipnja 2023.

Sažetak

Aplikacija za vizualizaciju i uređivanje modela IT sustava i pripadajuće implementacije sigurnosnih politika

Rad se bavi tematikom informacijsko tehnoloških sustava, te njihovih problema u sadašnjici, prvenstveno o činjenici da mnogi nisu javno dostupni za uporabu iz sigurnosnih razloga što sprječava potencijal proučavanja, testiranja i razvoja takvih sistema. Uzima se drugačiji pristup – izrada vlastitih sustava uz početne predefinirane zahtjeve koji generirani sustav mora zadovoljavati. U radu se opisuje izrada korisničkog sučelja aplikacije korištenjem radnog okvira Angular i implementacija biblioteke D3.js kako bi se omogućila olakšana uporaba procesa generiranja simuliranih informacijsko tehnoloških sustava koji bi imali veliku primjenu u daljnjem razvoju te njihova vizualizacija.

Ključne riječi: Informacijsko tehnološki sustavi, Angular, D3.js, Angular CLI, Angular Material

Summary

Application for visualization and editing of IT system models and corresponding implementation of security policies

The paper deals with the subject of information technology systems, and their problems in the present, primarily about the fact that many are not publicly available for use due to security reasons, which prevents the potential of studying, testing, and developing such systems. A different approach is taken - creating own systems with initial predefined requirements that the generated system must satisfy. The paper describes the creation of a user interface of the application using the Angular framework and the implementation of the D3.js library in order to enable easier use of the process of generating simulated information technology systems which would have great use in further development and their visualization.

Keywords: Information technology systems, Angular, D3.js, Angular CLI, Angular Material

Skraćenice

ITS	*Information Technology System*	Informacijsko tehnološki sustav
JSON	*Javascript Object Notation*	Javascript Objekta Notacija
API	*Application Programming Interface*	Sučelje za Programiranje Aplikacije
D3	*Data-Driven Documents*	Dokumenti Vođeni Podacima
SVG	*Scalable Vector Graphics*	Skalabilna vektorska grafika
HTML	*Hypertext Transfer Protocol*	asinkroni način prijenosa
CSS	*Cascading Style Sheets*	Kaskadni Stilski Listovi
DOM	*Document Object Model*	Model Objekta Dokumenta
CSV	*Comma Separated Values*	Vrijednosti Odvojene Zarezom
TSV	*Tab Separated Values*	Vrijednosti Odvojene Tabulatorom
MVC	*Model-View-Controller*	Model Prikaz Kontroler
HTTP	*HyperText Transfer Protocol*	Protokol za Prijenos Hiperteksta
JWT	*JSON Web Token*	JSON Web Token
MVVM	*Model-View-ViewModel*	Model Prikaz ViewModel