Vizualizacija peer-to-peer (P2P) distribuirane mreže u Rust programskom jeziku

Blašković, Ivan

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

VIZUALIZACIJA PEER-TO-PEER (P2P) DISTRIBUIRANE MREŽE U RUST PROGRAMSKOM JEZIKU

Rijeka, rujan 2024.

Ivan Blašković

0069093409

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

VIZUALIZACIJA PEER-TO-PEER (P2P) DISTRIBUIRANE MREŽE U RUST PROGRAMSKOM JEZIKU

Mentor: prof. dr. sc. Kristijan Lenac

Rijeka, rujan 2024.

Ivan Blašković

0069093409

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 24.03.2024.

Zavod: Zavod za računarstvo
Predmet: Operacijski sustavi

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Ivan Blašković (0069093409)**
Studij: Sveučilišni prijediplomski studij računarstva (1035)

Zadatak: **Vizualizacija peer-to-peer (P2P) distribuirane mreže u Rust programskom jeziku / Visualization of a peer-to-peer (P2P) distributed network in the Rust programming language**

Opis zadatka:

Razviti knjižnicu u programskom jeziku Rust namijenjenu vizualizaciji peer-to-peer (P2P) distribuirane mreže, poput blockchain mreže, koja omogućava praćenje stanja mreže i pojedinačnih čvorova u stvarnom vremenu. Vizualizacija treba biti dinamična, s mogućnošću konfiguracije tijekom korištenja i interaktivnim elementima.

Rad mora biti napisan prema Uputama za pisanja diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 20.03.2024.

Mentor:
prof. dr. sc. Kristijan Lenac

Predsjednik povjerenstva za završni ispit:
prof. dr. sc. Miroslav Joler

IZJAVA O SAMOSTALNOJ IZRADI RADA

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2024.

Ivan Blašković

ZAHVALA

Zahvaljujem se mentoru prof. dr. sc. Kristijanu Lencu na pomoći pri izradi ovog rada, svom prenesenom znanju te svakom danom savjetu i korisnom razgovoru.

Najviše se zahvaljujem svojoj obitelji što su me pratili, savjetovali, podržavali i ohrabrivali, kako tijekom studija, tako i tijekom cijelog života. Moji uspjesi ne bi bili mogući bez vas.

Također se želim zahvaliti svim prijateljima i kolegama s fakulteta na svakoj pruženoj podršci, savjetu i zajednički provedenom vremenu.

SADRŽAJ

POPIS SLIKA

1. Uvod

Uporaba peer-to-peer mreža svakodnevno se povećava. To je prvenstveno zbog prednosti koje one nude, a glavne među njima su decentralizacija i distribucija. Peer-to-peer mreže su decentralizirane mreže, što znači da u mreži ne postoji centralni entitet. Time mreža ne zavisi od jedne točke ili čvora, čime se povećava otpornost na kvarove, napade i cenzuru. Prednost distribucije odnosi se na svojstvo peer-to-peer mreža da se zadaci koje mreža obavlja izvršavaju na više čvorova, a ti čvorovi međusobno dijele svoje resurse poput procesne snage i memorije.

Zadatak završnog rada bila je izrada knjižnice za vizualizaciju peer-to-peer mreža u Rust programskom jeziku. Rust programski jezik moderan je programski jezik koji se ističe svojom sigurnosti, brzinom i konkurentnosti. Napravljen je s ciljem povećanja sigurnosti tijekom izvođenja, a time i s ciljem sigurnosti varijabli i podataka. Kod izrade Rust programskog jezika uzeti su dobri principi iz mnogih drugih programskih jezika.

U ovom se radu opisuje izrada knjižnice za vizualizaciju peer-to-peer mreža, način na koji svaki od njezinih dijelova funkcionira te kako se njome upravlja. Uz navedeno opisuju se pojmovi vezani uz peer-to-peer mreže te ostale stvari koje se koriste u navedenoj knjižnici.

2. Peer-to-peer mreže

Uporaba peer-to-peer (P2P) mreža svakodnevno raste s obzirom na brojne prednosti i mogućnosti takvog tipa mreže. Prema [1], peer-to-peer mreža je vrsta mreže u kojoj je svaki čvor te mreže istovremeno i poslužitelj i klijent. Iz toga proizlazi da peer-to-peer mreže su distribuirane i decentralizirane mreže. Distribuirane mreže su mreže kod kojih se podaci dijele i pohranjuju na više čvorova te mreže. Decentralizirane mreže su one mreže kod kojih ne postoji jedinstveni centralni poslužitelj, već svaki čvor te mreže može vršiti ulogu poslužitelja što je suprotno od poznatije klijent – poslužitelj mrežne arhitekture. Definicija peer-to-peer mreže prema [1] glasi: „Distribuirana mrežna arhitektura može se nazvati Peer-to-Peer (P-to-P, P2P, …) mrežom ako sudionici dijele dio vlastitih hardverskih resursa (procesna snaga, kapacitet pohrane, kapacitet mrežne veze, pisače, …). … Oni su dostupni drugim sudionicima izravno, bez prolaska posredničkim subjektima. Sudionici takve mreže su stoga pružatelji resursa (usluga i sadržaja) kao i zahtjevatelji resursa (usluga i sadržaja).“

Peer-to-peer mreže se prema postojanju centralnog entiteta dijele na čiste peer-to-peer mreže i hibridne peer-to-peer mreže, dok se prema organizaciji čvorova mogu podijeliti na strukturirane i nestrukturirane peer-to-peer mreže.

2.1. Čiste peer-to-peer mreže

Čiste peer-to-peer mreže nemaju centralni entitet koji kontrolira mrežu, već svi čvorovi imaju potpuno jednak status i važnost. Svaki čvor može tražiti ili nuditi sadržaj bez posrednika, a komunikacija se odvija direktno između čvorova. Uz nepostojanje centralnog entiteta, prema [1], čiste peer-to-peer mreže su one kod kojih se bilo koji proizvoljno odabran čvor te mreže može ukloniti iz mreže bez ikakvih gubitaka, odnosno da mreža i dalje jednako radi. Primjer čiste peer-to-peer mreže je Gnutella, jedna od prvih peer-to-peer mreža napravljena za dijeljenje datoteka.

2.2. Hibridne peer-to-peer mreže

Hibridne peer-to-peer mreže su kombinacija peer-to-peer i klijent – poslužitelj arhitekture mreže. Centralni entitet se koristi za indeksiranje i pretragu podataka, a sam prijenos podataka obavlja se direktno između čvorova. Iako ova vrsta mreže djelomično koristi ideologiju klijent – poslužitelj

arhitekture, razlika je što čvorovi hibridne peer-to-peer mreže dijele svoje resurse, dok se kod klijent – poslužitelj mreže koriste isključivo resursi poslužitelja. Jedna od najpoznatijih mreža ovog tipa je BitTorrent kod koje poslužitelji pronalaze na kojem se čvoru nalaze koji dijelovi datoteka, ali se sama razmjena podataka odvija direktno između čvorova, odnosno korisnika.

2.3. Strukturirane peer-to-peer mreže

Strukturirane peer-to-peer mreže koriste algoritme za organizaciju čvorova u specifične strukture kao što su distribuirane tablice identifikacijskih oznaka (eng. distributed hash table – DHT). Uporaba takvih tablica omogućava efikasniju pretragu mreže jer se podaci organiziraju tako da se lako može locirati gdje se određeni sadržaj nalazi.

2.4. Nestrukturirane peer-to-peer mreže

U nestrukturiranim peer-to-peer mrežama nema unaprijed definirane strukture, već čvorovi nasumično komuniciraju jedni s drugima. Iako je ovaj tip mreže jednostavniji za implementaciju, pretrage u takvim mrežama mogu biti manje efikasne jer se svaki upit mora propagirati kroz veliki broj čvorova, što povećava vrijeme pretrage i korištenje mrežnih resursa.

2.5. Prednosti peer-to-peer mreže

Peer-to-peer mreže donose brojne prednosti u odnosu na centralizirane mrežne arhitekture, što je jedan od glavnih razloga njihove popularnosti. Najveća i najpopularnija prednost je decentralizacija odnosno odsustvo centralnog entiteta ili poslužitelja. Time mreža ne zavisi od jedne točke ili čvora, čime se povećava otpornost na kvarove, napade i cenzuru. S obzirom da čvorovi peer-to-peer mreže dijele svoje resurse, omogućava se bolja iskoristivost raspoloživih resursa. U većini slučajeva mreža postaje efikasnija s porastom broja korisnika. Time se također omogućava skalabilnost – dodavanjem novih čvorova u mrežu, povećavaju se ukupni resursi. Za razliku od klijent – poslužitelj mrežne arhitekture, kod koje povećanje broja korisnika može preopteretiti poslužitelj, kod peer-to-peer mreža porast broja korisnika uglavnom rezultira porastom ukupne snage mreže.

2.6. Nedostaci peer-to-peer mreže

Iako peer-to-peer mreže imaju brojne prednosti, od kojih su samo neke navedene iznad, one također imaju i nedostatke te dolaze sa izazovima koje je važno uzeti u obzir. Iako je decentralizacija glavna prednost peer-to-peer mreže, ona ima i jedan veliki nedostatak. Pošto ne postoji centralizirana kontrola, zlonamjerni korisnici mogu ući u mrežu i pokušati distribuirati neki maliciozni softver ili pak pokušati neovlašteno pristupiti podacima što predstavlja veliki sigurnosni rizik, a anonimnost koju peer-to-peer mreže pružaju dodatno olakšavaju takve maliciozne radnje. Upravo zbog navedene anonimnosti, peer-to-peer mreže se često koriste za ilegalne aktivnosti poput distribucije nezakonitih sadržaja što predstavlja značajan izazov za zakonodavce i regulatore.

Što se tiče same arhitekture mreže, značajni nedostaci su nesigurna stabilnost i mogućnost neravnomjernog opterećenja. Kako peer-to-peer mreže zavise o svojim čvorovima koji u svakom trenutku mogu biti isključeni iz mreže, stabilnost mreže može biti upitna. Na primjer, ako mnogo čvorova odjednom napusti mrežu, dostupnost resursa može naglo pasti što može negativno utjecati na rad mreže. Iz istog razloga može doći i do neravnomjernog opterećenja čvorova. Neki čvorovi mogu biti preopterećeni velikim brojem zahtjeva, dok su drugi relativno neaktivni, što može dovesti do degradacije performansi mreže te povećanju kašnjenja u razmjeni podataka.

2.7. Primjena peer-to-peer mreže

Primjena peer-to-peer mreža svakodnevno raste. Najpoznatija primjena je kod blockchain tehnologije koja između ostalog stoji iza kriptovaluta. Kod njih se transakcije bilježe u distribuiranoj mreži čvorova, bez potrebe za centralnom institucijom poput banke. Takav pristup omogućava sigurnost i transparentnost transakcija, dok se istovremeno smanjuje mogućnost prevara i manipulacije podacima.

Peer-to-peer mreže se također koriste kod dijeljenja datoteka. Mreže kao što su Napster i BitTorrent omogućuju korisnicima diljem svijeta međusobno dijeljenje datoteka, medijskih sadržaja, softvera i otvorenog koda, ali i sadržaja prikupljenog ilegalnim aktivnostima. Slična primjena je i distribuirano računanje kod koje korisnici dobrovoljno daju resurse svojih računala za izračun složenih znanstvenih i matematičkih problema.

3. Programski jezik Rust

Rust je moderan programski jezik koji se ističe po svojoj sigurnosti, brzini i konkurentnosti. Nastao 2006. godine, a njegov glavni autor je Graydon Hoare. Prva stabilna verzija objavljena je u svibnju 2015. godine. Prema [2], cilj jezika je omogućiti programerima da stvaraju visokoučinkovite aplikacije bez potrebe za žrtvovanjem sigurnosti i pouzdanosti sustava. Rust se često uspoređuje s jezicima poput C i C++, no razlikuje se po svom sustavu upravljanja memorijom koji eliminira značajan broj grešaka i sigurnosnih propusta.

Rust je statički tipiziran jezik, što znači da se tipovi podataka određuju u vrijeme kompiliranja. Njegov sintaktički dizajn i koncepti inspirirani su različitim jezicima poput C++, Haskell, Ruby i sličnih, no nudi poboljšanja koja rješavaju mnoge uobičajene probleme programera, posebno u vezi s upravljanjem memorijom.

Rust je poznat po svojim visokim performansama. Budući da Rust nema sakupljač smeća (eng. garbage collector), omogućuje vrlo precizno upravljanje memorijom, što je slično kao u C i C++ programskim jezicima. To znači da programeri mogu pisati kod koji se izvršava blizu hardvera, što je posebno korisno u područjima poput sustava u stvarnom vremenu, igara ili ugradbenih sustava. U kombinaciji s modelom vlasništva nad varijablama i memorijom te posudbi istih, Rust omogućava programerima da pišu optimizirane programe bez ugrožavanja sigurnosti sustava. [2]

Paralelizam je također jedna od glavnih prednosti Rust-a. U tradicionalnim jezicima, paralelizam često dovodi do kompleksnih grešaka koje su teške za otkrivanje i otklanjanje. Rust-ov sustav vlasništva i posudbi omogućuje siguran paralelizam jer osigurava da podaci ne mogu biti istovremeno promijenjeni iz više dretvi, a time se izbjegavaju problemi poput uvjeta utrke. Korištenje paralelizma u Rust-u je jednostavnije i sigurnije nego u mnogim drugim jezicima, što ga čini odličnim izborom za razvoj softvera koji zahtijeva visoku razinu konkurentnosti.

3.1. Rust paket kao knjižnica ili aplikacija

Programski jezik Rust nudi jednostavnu definiciju nekog paketa kao knjižnice ili aplikacije. Označavanje paketa kao knjižnice vrši se kreiranjem *lib.rs* datoteke u korijenu paketa. Ta je datoteka ulazna datoteka te knjižnice i u njoj bi se trebale nalaziti knjižnične funkcije dostupne korisnicima koji žele implementirati tu knjižnicu u svoj paket, odnosno sustav. Ukoliko se paket

koristi kao zasebna aplikacija, umjesto *lib.rs* datoteke potrebno je kreirati *main.rs* datoteku u korijenu paketa te unutar te datoteke treba definirati glavu funkciju naziva „*main*“. Ta je funkcija početna točka kod kreiranja izvršne datoteke. Iz ovog je vidljivo kao je promjena paketa iz knjižnice u aplikaciju i obratno vrlo jednostavna. Treba samo zamijeniti *lib.rs* datoteku sa *main.rs* datotekom ili obratno. Uz to je poželjno i promijeniti definicije funkcija u tim datotekama. Unutar *main.rs* datoteke obavezno mora bit *main* funkcija, dok unutar *lib.rs* datoteke je poželjno da funkcije budu imenovane prema onome što izvršavaju kako bi korisnicima te knjižnice bila olakšana implementacija u svoj paket ili sustav.

Rust također omogućava hibridne pakete, odnosno pakete koji sadrže i *main.rs* i *lib.rs* datoteke. Imajući obje datoteke u korijenu paketa, taj se paket može koristiti i kao knjižnica i kao zasebna aplikacija. Kod izrade hibridnih paketa, poželjno je unutar *Cargo.toml* datoteke u korijenu paketa definirati različite nazive za knjižnicu i aplikaciju. Uz nazive je potrebno definirati i putanje *main.rs* i *lib.rs* datoteka. Na ovaj se način sprječavaju eventualne greške.

3.2. Cargo – alat za upravljanje paketima

Rust dolazi s bogatim ekosustavom alata koji olakšavaju razvoj, a jedan od njih je Cargo. To je alat za upravljanje paketima i gradnju projekata. Cargo automatski upravlja ovisnostima, omogućuje jednostavno testiranje koda i generiranje dokumentacije. Paketni sustav Rust-a također omogućuje jednostavno dijeljenje i preuzimanje knjižnica, čineći cijeli proces razvoja bržim i učinkovitijim. Uz pomoć Cargo alata moguće je između ostalog kreirati novi paket odnosno projekt, pokrenuti paket u razvojnom okruženju, provjeriti ispravnost koda te generirati izvršnu datoteku tog paketa za konfiguraciju računala na kojemu se kod piše, ali i za mnoge druge konfiguracije i operacijske sustave koje ovaj alat podržava.

3.3. Zajednica i podrška

Jedan od ključnih čimbenika uspjeha Rust-a je aktivna i podržavajuća zajednica programera. Zajednica Rust-a poznata je po svojoj uključenosti i spremnosti na pomoć, što je pridonijelo brzom rastu uporabe ovog jezika. Rust je otvorenog koda, a mnogi doprinosi dolaze od zajednice, čime se osigurava stalno poboljšanje jezika i alata. Njegova dokumentacija je opsežna i dobro napisana, što olakšava učenje jezika čak i za početnike. Osim toga, postoje primjeri, vodiči i forumi na

kojima programeri mogu dobiti pomoć i podijeliti svoja iskustva, iako ih je manje nego za druge popularnije i duže upotrebljavane jezike poput C i C++ programskih jezika.

Rust zajednica razvila je širok spektar knjižnica za različite svrhe, od web razvojnih alata, sustava baza podataka, pa sve do alata za strojno učenje. Ovaj bogat ekosustav čini Rust svestranim jezikom koji se može koristiti u mnogim različitim domenama, a popularan je i u blockchain tehnologiji. Rust također nudi izvrsne mogućnosti za integraciju s drugim jezicima, posebno s C i C++. Zahvaljujući tome, Rust se može koristiti u postojećim projektima napisanima u ovim jezicima, omogućujući postupni prijelaz ili dodavanje novih značajki bez potrebe za potpunim prepisivanjem cijelog koda.

Rust je programski jezik čija je upotreba još uvijek u usponu te ona svakodnevno raste. Zahvaljujući različitim domenama upotrebe te mnogim prednostima, a pogotovo sigurnosti izvršavanja, vrlo je vjerojatno da će postati jedan od najpopularnijih i najkorištenijih programskih jezika.

4. Implementacija knjižnice za vizualizaciju peer-to-peer mreža

Izrađena knjižnica za vizualizaciju peer-to-peer omogućava korisnicima da stvore željenu mrežu dodavajući čvorove i veze neke mreže u grafičkom sučelju. Dodavanje čvorova je moguće i učitavanjem GEXF (Graph Exchange XML Format) ili GraphML datoteka u kojima je zapisana konfiguracija mreže, odnosno čvorova i veza. Izgled i atributi tih datoteka opisani su u Prilogu 1. Mrežu iz grafičkog sučelja je također moguće spremiti u GEXF ili GraphML datoteku kako bi se lakše dijelila ili sačuvala za kasniju upotrebu bez potrebe za ponovnim pojedinačnim definiranjem svih čvorova i veza. Svakom od čvorova moguće je definirati identifikacijsku oznaku, naziv, IP adresu te hardversku i softversku konfiguraciju čvora što uključuje informacije o procesoru, radnoj memoriji (RAM), sekundarnoj memoriji za spremanje podataka (ROM), propusnosti mrežnog sučelja, operacijskom sustavu i softveru koji se na tom čvoru izvršava.

Knjižnica također sadržava i aplikacijsko programsko sučelje (eng. application programming interface – API) kojim se omogućuje promjena atributa i statusa dostupnosti čvorova preko internetske mreže. Ukoliko neki čvor postane nedostupan ili je uklonjen iz mreže, slanjem zahtjeva na točku sučelja za označavanje čvora nedostupnim, čvor čija se identifikacijska oznaka nalazi u tom zahtjevu se u grafičkom sučelju označava sivom bojom što predstavlja da je taj čvor u tom trenutku nedostupan. Ukoliko čvor ponovno postane dostupan u mreži, slanjem zahtjeva na drugu točku programskog sučelja, čvor se u grafičkom sučelju označava crnom bojom što predstavlja da je u tom trenutku ponovno dostupan. Detalji o točkama programskog sučelja i izgledu zahtjeva i odgovora opisani su u Prilogu 2.

Osim upotrebe opisanog programskog sučelja, korisnik ima mogućnost upotrebe javnih knjižničnih funkcija koje omogućuju istu funkcionalnost kao i programsko sučelje spomenuto ranije. Time se omogućuje upravljanje ovom knjižnicom za vizualizaciju iz nekih drugih knjižnica ili aplikacija koje koriste ovu knjižnicu. Detalji o spomenutim funkcijama i primjer upotrebe opisani su u Prilogu 3.

Ova knjižnica koristi *egui* knjižnicu i *eframe* radno okruženje koji služe za izradu grafičkog sučelja, a za izradu točaka izmjene podataka aplikacijskog programskog sučelja, kreiranje više dretvi na kojima se izvršavaju dijelovi ove knjižnice te asinkrono pozivanje funkcija aplikacijskog programskog sučelja korištena je knjižnica *tokio* i radno okruženje *axum*. Izrađena knjižnica za vizualizaciju također koristi druge pomoćne knjižnice koje se koriste za obradu GEXF i GraphML datoteka te obradu zahtjeva pristiglih na aplikacijsko programsko sučelje.

4.1. GEXF – Graph Exchange XML Format

GEXF format jedan je od najkorištenijih formata za dijeljenje i spremanje složenih mreža, što uključuje podatke o čvorovima i vezama tih mreža. Izrada ovog standarda zapisa mreža počela je 2007. godine zajedno sa Gephi projektom, odnosno sustavom za vizualizaciju mreža koja koristi upravo GEXF format zapisa [3].

Sintaksa GEXF datoteka temelji se na XML-u (eng. Extensible Markup Language) i sadrži nekoliko ključnih elemenata. Korijenski element je <*gexf*>, koji definira osnovne informacije o verziji formata. Unutar njega nalaze se elementi poput <*graph*>, koji opisuje strukturu grafa, te atribut *mode* koji određuje je li graf statičan ili dinamičan. Čvorovi (eng. nodes) i bridovi (eng. edges) definirani su pomoću elemenata <*node*> i <*edge*>, pri čemu svaki čvor ima jedinstveni identifikator, a brid povezuje dva čvora putem atributa *source* i *target*.

Pošto je GEXF format napravljen s ciljem olakšavanja prijenosa mrežne topologije, nudi mnoge prednosti za zapis podataka o mreži u odnosu na druge formate zapisa grafova i mreža. Jedna od prednosti je fleksibilnost. Ovaj format ima jasno definiranu sintaksu i ograničenja, ali omogućuje korisnicima dodavanje proizvoljnih atributa kako bi mogli prilagoditi datoteku svojim potrebama. Dodatni atributi označavaju se elementima <*attribute*>, koji definira naziv atributa, te <*attvalue*>, kojim se definira vrijednost atributa za neki element u mreži, poput čvora ili veze. Mogućnost određivanje hijerarhije elemenata u mreži još je jedna od prednosti koje ovaj format nudi, a nije često prisutna u drugim formatima. Time se također može prikazati više čvorova kao jedan, što je pogodno kod grupiranja i paralelizma čvorova. Ovaj format također podržava dinamične mreže, odnosno moguće je zapisivati promjene u mreži, što također nije čest slučaj kod ostalih formata zapisa mreža i grafova. [3]

4.2. GraphML

GraphML je sveobuhvatan i jednostavan format za zapis grafova. Baziran je na XML-u, jednostavne je sintakse i vrlo sličan GEXF formatu. Za razliku od GEXF formata, koji je prvenstveno napravljen za zapis računalnih mreža, GraphML format je više orijentiran prema zapisu grafova te zato nema toliko mogućnosti za zapis mreža kao GEXF format. Neovisno o ovim ograničenjima, njegova upotreba u zapisu mreža je vrlo česta zbog njegove duge postojanosti, a razvoj ovog formata počeo je 2000. godine. Kao i GEXF format, GraphML format također omogućava dodavanje dodatnih atributa i određivanje hijerarhije. [4]

Ključni elementi unutar GraphML datoteke su <*graphml*>, <*graph*>, <*node*> i <*edge*>. Element <*graphml*> je korijenski element koji sadrži cijelu strukturu grafa. Unutar njega, element <*graph*> predstavlja sam graf i može biti usmjeren ili neusmjeren, što se definira atributom *edgedefault*. Svaki je čvor, kao i u GEXF formatu, definiran pomoću elementa <*node*> i ima jedinstveni identifikator putem atributa *id*, a bridovi se definiraju elementom <*edge*> koji povezuje dva čvora putem atributa *source* i *target*. Dodatne informacije kao što su atributi i metapodaci mogu se dodavati koristeći elemente <*data*> koji omogućuju prilagodbu zapisa grafa prema specifičnim potrebama korisnika.

4.3. Struktura knjižnice

Datotečna struktura knjižnice vidljiva je na slici 4.1. S obzirom da je ova knjižnica napravljena kao hibridni paket, odnosno paket koji je istovremeno i knjižnica i zasebna aplikacija, moguće je uočiti i *lib.rs* i *main.rs* datoteke u korijenu paketa. U ovom slučaju, *lib.rs* datoteka sadrži funkcije za inicijalizaciju i pokretanje vizualizatora te funkcije za promjenu atributa čvorova i statusa dostupnosti čvora u mreži, dok *main.rs* datoteka sadrži samo *main* funkciju u kojoj se pozivaju funkcije za inicijalizaciju i pokretanje vizualizatora iz *lib.rs* datoteke. Time se može reći da zapravo ovaj paket, gledan kao zasebna aplikacija, koristi samog sebe kao knjižnicu.

Slika 4.1. Struktura paketa knjižnice za vizualizaciju

Na slici 4.1. je također moguće vidjeti i module ove knjižnice koji su nazvani *gui_control*, *nff_utils*, *api* te *models*. Modul *models* sadrži isključivo istoimenu datoteku s nastavkom *„.rs“* u kojoj su definirane strukture podataka korištene u ovoj knjižnici. Svaki od ostalih modula ima istoimenu datoteku s nastavkom *„.rs“* te istoimeni direktorij u kojemu se nalaze njihovi podmoduli. Modul *gui_control* sadrži podmodule i funkcije potrebne za izradu, crtanje i upravljanje grafičkim korisničkim sučeljem. Modul *nff_utils* sadrži podmodule s funkcijama potrebnim za obradu GEXF i GraphML datoteka, dok *api* modul sadrži podmodule i funkcije potrebne za aplikacijsko programsko sučelje (API).

4.4. Glavne strukture podataka

Unutar *models* modula se nalaze strukture podataka korištene u ovoj knjižnici. Najznačajnija struktura je *MyApp* koja definira sve potrebne varijable za izvršavanje ove knjižnice. Unutar te strukture se nalaze varijable za spremanje čvorova i veza, varijable koje se odnose na cijelu knjižnicu te varijable za spremanje svih stanja knjižnice koje je potrebno pratiti prilikom izvršavanja koda. Struktura *MyApp* nalazi se na slici 4.2.

```rust
pub struct MyApp {
    pub(crate) state: Arc<Mutex<State>>,
    pub(crate) nodes_arc: Arc<Mutex<Vec<Node>>>,
    pub(crate) links_arc: Arc<Mutex<Vec<Link>>>,

    pub(crate) left_side_panel_width: f32,

    pub(crate) dragging: bool,
    pub(crate) dragged_node_id: Option<String>,
    pub(crate) mouse_drag_delta: Vec2,
    pub(crate) zoom: f32,

    pub(crate) show_error: bool,
    pub(crate) error_message: String,

    pub(crate) show_node_names: bool,
    pub(crate) node_popup: Option<Node>,
    pub(crate) node_default_radius: f32,

    pub(crate) adding_node: bool,
    pub(crate) new_node: Node,
    pub(crate) show_input_dialog: bool,
    pub(crate) new_node_pos: Pos2,

    pub(crate) deleting_node: bool,
    pub(crate) show_delete_dialog: bool,
    pub(crate) node_to_delete: Option<Node>,
    pub(crate) left_click_released: bool,

    pub(crate) adding_link: bool,
    pub(crate) deleting_link: bool,
    pub(crate) first_node_selected: Option<Node>,
    pub(crate) second_node_selected: Option<Node>,

    pub(crate) node_editing: bool,
    pub(crate) node_to_edit: Option<Node>,
    pub(crate) node_to_edit_id: Option<String>,
    pub(crate) node_to_edit_name: Option<String>,
}
```

Slika 4.2. Struktura stanja knjižnice nazvana „MyApp“

Unutar *models* modula nalaze se i strukture za čvorove i veze. Strukturu čvora moguće je vidjeti na slici 4.3. Struktura sadržava identifikacijsku oznaku čvora, njegov naziv, IP adresu te hardversku i softversku konfiguraciju što uključuje informacije o procesoru, radnoj memoriji (RAM), sekundarnoj memoriji (ROM), propusnosti mrežnog sučelja, operacijskom sustavu i softveru koji se na tom čvoru izvršava. Svi podaci osim identifikacijske oznake su opcionalni.

```
pub struct Node {
    pub id: String,
    pub name: String,
    pub(crate) center: egui::Pos2,
    pub(crate) radius: f32,
    pub(crate) color: Color32,
    pub ip_addr: String,
    pub cpu: String,
    pub ram: String,
    pub rom: String,
    pub os: String,
    pub network_bw: String,
    pub software: String,
}
```

Slika 4.3. Struktura čvora

Na slici 4.4. moguće je vidjeti strukturu veze. Ta struktura sadržava isključivo identifikacijske oznake čvorova koje određena veza povezuje.

```
pub(crate) struct Link {
    pub(crate) node1_id: String,
    pub(crate) node2_id: String,
}
```

Slika 4.4. Struktura veze

Kao što je vidljivo na slici 4.2., podaci o čvorovima i vezama se spremaju unutar *Arc*, *Mutex* i *Vec* struktura. Najprije se struktura čvora uključuje u *Vec* strukturu čime se dobiva vektor, odnosno lista s više čvorova. Zatim se taj vektor uključuje u *Mutex* strukturu. *Mutex*, skraćeno od engleskog naziva Mutual Exclusion, je struktura koja omogućava sigurnu promjenljivu (eng. mutable) zaštitu podataka u višedretvenom okruženju. Prilikom korištenje *Mutex* strukture, pristup podacima koji su zaštićeni *Mutex*-om je ekskluzivan. To znači da samo jedna nit u isto vrijeme može dobiti pristup podacima, odnosno kada jedna nit zaključa *Mutex* strukturu, sve druge niti koje joj pokušavaju pristupiti moraju čekati da se *Mutex* otključa od strane prve. *Mutex* struktura se zatim uključuje u *Arc* strukturu. *Arc*, skraćeno od engleskog naziva Atomic Reference Counting, je pametni pokazivač koji omogućava dijeljenje podataka između više niti na siguran način. *Arc* zapravo

omogućava višestruko dijeljenje podataka, a broj referenci na te podatke se automatski ažurira. Kombinacija *Arc* i *Mutex* struktura omogućava sigurno dijeljenje i mijenjanje podataka između niti. Na ovaj se način dobiva sigurnost podataka o čvorovima i vezama, pogotovo zbog pristupa tim podacima iz različitih dijelova knjižnice.

4.5. Višedretvenost

Višedretvenost označava da se aplikacija izvodi paralelno na više dretvi. Kod ove knjižnice za vizualizaciju peer-to-peer mreža koriste se dvije dretve, a dodatne se dretve kreiraju prilikom obrade podataka pristiglih na API. Početna i glavna dretva služi za inicijalizaciju samog vizualizatora te pokretanje grafičkog korisničkog sučelja i iscrtavanje njegovih elemenata na ekranu. Druga se dretva kreira iz ove početne pomoću *tokio* knjižnice i služi za pokretanje poslužitelja za API. Kreiranje te druge dretve vidljivo je na slici 4.5. Na toj se drugoj dretvi izvršava kod koji čeka pristigli zahtjev na API. Po pristiglom zahtjevu, kreira se dodatna, privremenu dretva koja obrađuje pristigli zahtjev. Nakon obrade zahtjeva, podaci se šalju na početnu dretvu pomoću *mpsc* modula. Taj je modul često korišten za komunikaciju među dretvama, a nalazi se u standardnoj knjižnici Rust programskog jezika. Naziv samog modula (*mpsc*) dolazi od engleskog „multi-producer, single-consumer“ čime se označava da on podržava više pošiljatelja, ali samo jednog primatelja. To je zapravo i način na koji funkcionira. Preko unaprijed definiranog kanala na kojemu će dretve komunicirati, moguće je slati poruke s više različitih dretvi, odnosno pošiljatelja. Te se poruke dodaju u red, ali isključivo jedna dretva koja ima ulogu primatelja ih može čitati redom kojim se šalju.

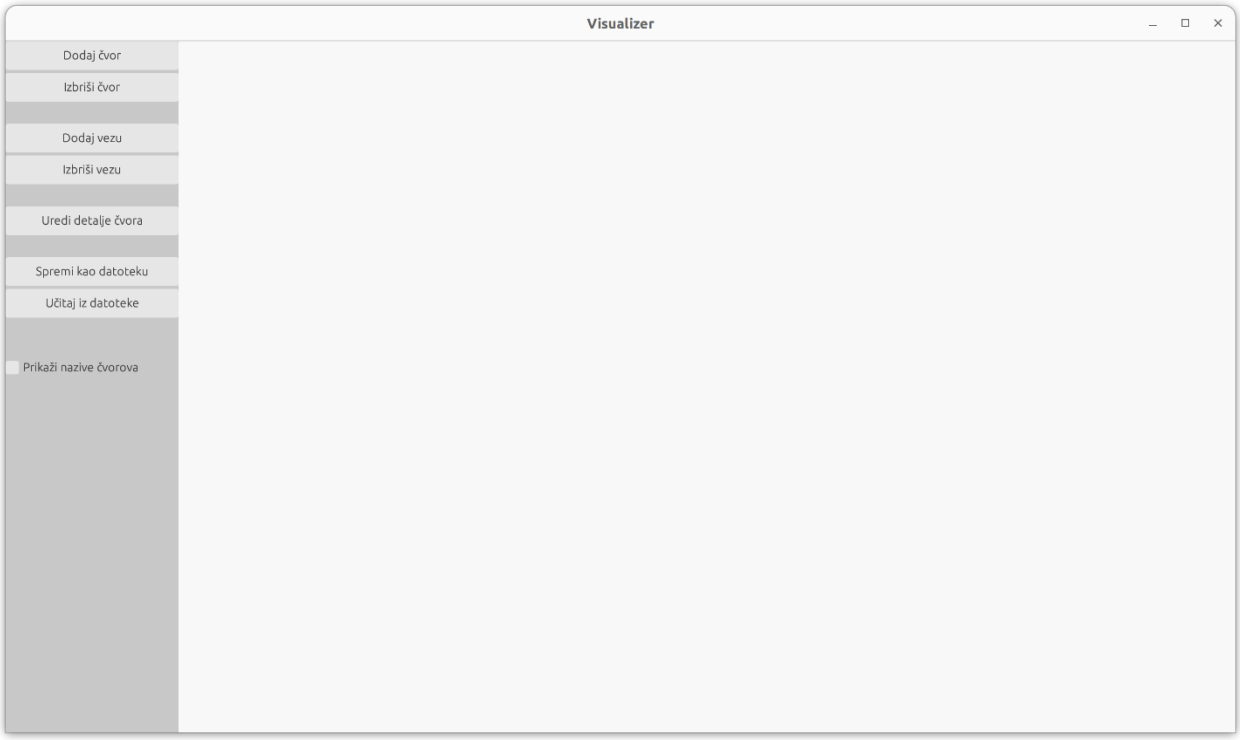
```
let state_clone: Arc<Mutex<State>> = state.clone();
let runtime: Runtime = tokio::runtime::Runtime::new().unwrap();
let runtime_handle: Handle = runtime.handle().clone();
runtime.spawn(future: async move {
    api::web_server(state_clone).await;
});
```

Slika 4.5. Kreiranje dretve poslužitelja za API

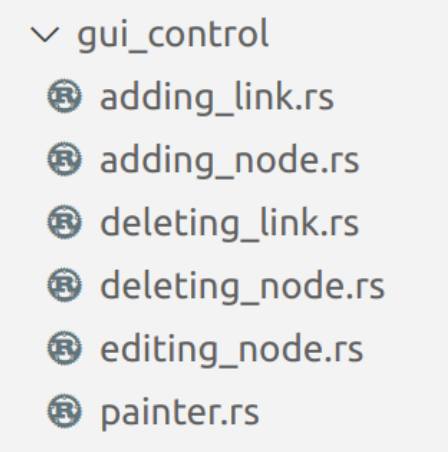
4.6. Grafičko korisničko sučelje

Modul *gui_control* sadržava kod i podmodule koji upravljaju grafičkim sučeljem. Datoteka *gui_control.rs* sadržava kod koji kreira kontrolnu ploču s lijeve strane prozora sa svim potrebnim gumbima. Ona također upravlja i prikazom pomoćnih prozora koji se koriste prilikom izrade i uređivanja konfiguracije peer-to-peer mreže koja se vizualizira. Izgled grafičkog sučelja prilikom pokretanja knjižnice vidljiv je na slici 4.6.

Modul *gui_control* sadržava podmodule koji su vidljivi na slici 4.7., a koji sadržavaju funkcije koje se pozivaju iz *gui_control.rs* datoteke prema potrebi te ovisno o stanju u kojem se knjižnica nalazi u tom trenutku. Imena tim podmodulima su dodijeljena prema funkciji koju obavljaju. Tako je modul *painter* zadužen za crtanje peer-to-peer mreže, odnosno njenih čvorova i veza, te se njegove funkcije pozivaju konstantno. Ostali moduli se koriste po potrebi, a služe za dodavanje i brisanje čvorova i veza te uređivanje podataka o čvorovima.



Slika 4.6. Grafičko sučelje prilikom pokretanja knjižnice za vizualizaciju



Slika 4.7. Podmoduli za grafičko sučelje

4.7. Obrada GEXF i GraphML datoteka

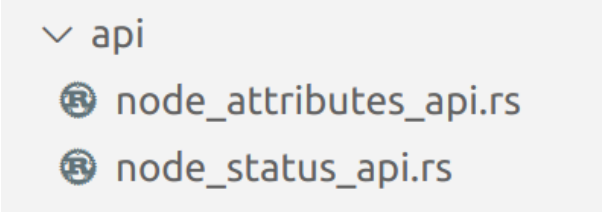
Unutar kontrolne ploče na grafičkom sučelju, nalazi se gumb za učitavanje konfiguracije mreže iz GEXF ili GraphML datoteke te gumb za izradu spomenutih datoteka koje će sadržavati trenutnu konfiguraciju mreže vidljivu u knjižnici za vizualizaciju. Klikom na neki od ta dva gumba otvara se prozor za odabir datoteke kod učitavanja, odnosno za odabir putanje na kojoj će se nova datoteka spremiti prilikom kreiranja nove datoteke. Nakon korisnikovog odabira, poziva se odgovarajuća funkcija iz *nff_utils* modula. Te funkcije provjeravaju o kojoj je vrsti datoteke riječ te pozivaju odgovarajuću funkciju iz određenog podmodula, a oni su vidljivi na slici 4.8. Imena podmodula sadržavaju dvije riječi odvojene donjom crtom. Prva riječ označava format datoteke (GEXF ili GraphML), a druga označava radi li se o podmodulu za učitavanje nove ili spremanje postojeće mreže (*„saver“* označava podmodul za spremanje, a *„parser“* podmodul za obradu učitane datoteke).

Slika 4.8. Podmoduli za obradu GEXF i GraphML datoteka

Podmoduli za obradu učitane datoteke čitaju odabranu datoteku te dodavaju elemente u strukture čvorova i veza, koji su opisani u poglavlju 4.4. Suprotno od toga, podmoduli za spremanje postojeće konfiguracije peer-to-peer mreže pretvaraju spremljene strukture u elemente koje GEXF i GraphML formati podržavaju, a oni su dodatno opisani u Prilogu 1.

4.8. Aplikacijsko programsko sučelje (API)

Za aplikacijsko programsko sučelje zadužen je *api* modul sa svojim podmodulima koji su vidljivi na slici 4.9. Podmodul *node_attributes_api.rs* zadužen je za promjenu atributa čvora i sadrži sve potrebne funkcije za obradu zahtjeva pristiglog na programsko sučelje. Isto tako, podmodul *node_status_api.rs* zadužen je za promjenu statusa čvorova, odnosno promjenu pojedinog čvora iz statusa nedostupan u dostupan i obratno, te sadržava se potrebne funkcije za obradu pristiglih zahtjeva te promjenu statusa čvora i njegove boje u grafičkom sučelju.



Slika 4.9. Podmoduli aplikacijskog programskog sučelja

Kao što je već opisano u poglavlju 4.5., sa početne se dretve kreira nova dretva namijenjena za aplikacijsko programsko sučelje. Na toj se dretvi izvodi funkcija iz *api* modula nazvana *web_server*, a vidljiva je na slici 4.10. Ta funkcija definira rute programskog sučelja, povezuje ih sa funkcijama iz podmodula te pokreće poslužitelj. Poslužitelj čeka dolazni zahtjev te po primitku zahtjeva poziva funkciju zaduženu za obradu tog zahtjeva.

```rust
pub async fn web_server(state: Arc<Mutex<State>>) {
    let state_for_node_down: Arc<Mutex<State>> = state.clone();
    let state_for_node_up: Arc<Mutex<State>> = state.clone();
    let state_for_node_update: Arc<Mutex<State>> = state.clone();

    let app: Router = Router::new() Router
        .route(
            path: "/node_down",
            method_router: post(handler: move |payload: Json<UpdateNodeRequ…| node_down(payload, state_for_node_down)))
        .route(
            path: "/node_up",
            method_router: post(handler: move |payload: Json<UpdateNodeRequ…| node_up(payload, state_for_node_up)))
        .route(
            path: "/node_update",
            method_router: post(handler: move |payload: Json<UpdateNodeRequ…| node_attributes_update(payload, state_for_node_update)));


    let addr: SocketAddr = SocketAddr::from(([0, 0, 0, 0], 8020));

    axum::Server::bind(&addr) Builder<AddrIncoming>
        .serve(app.into_make_service()) Server<AddrIncoming, IntoMakeService<…>>
        .await Result<(), Error>
        .unwrap();
}
```

Slika 4.10. API poslužitelj

Poslužitelj za programsko sučelje napravljen je uz pomoć *axum* radnog okvira. *Axum* je mrežni radni okvir razvijen na temelju *tokio* i *hyper* knjižnica. Cilj *axum* radnog okvira je pružiti jednostavan, ali moćan način za izradu mrežnih aplikacija s fokusom na performanse i sigurnost. Jedna od glavnih značajki *axum* radnog okvira je njegova potpuna integracija s asinkronim modelom programiranja u Rust-u. Zahvaljujući *tokio runtime*-u, *axum* omogućava učinkovitu obradu velikog broja paralelnih zahtjeva, čime se postižu visoke performanse. Ovaj radni okvir koristi HTTP biblioteku visokih performansi nazvanu *hyper* za rukovanje HTTP zahtjevima i odgovorima, što dodatno doprinosi njegovoj učinkovitosti.

Axum se ističe i ekspresivnim API-jem, koji omogućava lako definiranje ruta uz pomoć *Router* modula, što je i vidljivo na slici 4.10. Ovaj funkcionalni stil programiranja omogućava razvoj aplikacija koje su lako čitljive i održive, bez potrebe za puno nepreglednog i nepotrebnog koda. Još jedna važna značajka *axum* radnog okvira je mogućnost ekstrakcije podataka iz HTTP zahtjeva. Ova funkcionalnost omogućava jednostavno izdvajanje podataka iz URL parametara te tijela i zaglavlja zahtjeva. Korištenjem Rust-ovog sustava tipova i zaštite podataka, *axum* radni okvir osigurava sigurnost tijekom kompilacije, minimizirajući mogućnost grešaka tijekom izvršavanja programa. Ovo omogućava programerima da razvijaju stabilne i sigurne aplikacije.

Poslužitelj na slici 4.10., uz pomoć *axum* radnog okvira, pri primitku API zahtjeva, kreira novu privremenu dretvu na kojoj se izvršava funkcija jednog od *api* podmodula koji su zaduženi za točku API-ja na koju je zahtjev pristigao. Ta funkcija zatim obrađuje zahtjev te podatke šalje na prethodno definirani kanal *mpsc* modula opisanog u poglavlju 4.5.

Sa početne se dretve kreira nova dretva koja čeka poruke sa privremenih dretvi koje obrađuju zahtjeve poslane na API, a kreiranje nove dretve i kod koji se izvršava vidljiv je na slici 4.11. Kod čeka dolazak poruke preko *mpsc* modula. Ta poruka sadržava objekt sa podacima o čvoru koje je potrebno promijeniti i novi privremeni kanal *mpsc* modula preko kojeg se na dretvu pošiljatelja šalje povratna informacija o obradi pristiglih podataka. Pri primitku poruke zaključavaju su podaci o čvorovima spremljenim u *MyApp* strukturi. Zatim se u vektoru čvorova pronalazi traženi čvor i poziva funkcija koja ažurira njegove podatke prema zahtjevu. Nakon ažuriranja čvora se poziva funkcija iz *egui* knjižnice zadužena za ponovno crtanje grafičkog sučelja kako bi promjene bile odmah vidljive. Time je ažuriranje čvora završilo, a funkciji koja je obrađivala API zahtjev se vraća povratna informacija koja može biti „OK“, ako su promjene uspješno primijenjene, ili „Čvor nije pronađen“ ukoliko ne postoji čvor s identifikacijskom oznakom iz zahtjeva primljenog na API. Funkcija po primitku povratne informacije istu proslijedi korisniku u obliku JSON (JavaScript Object Notation) paketa kao odgovor na poslani zahtjev.

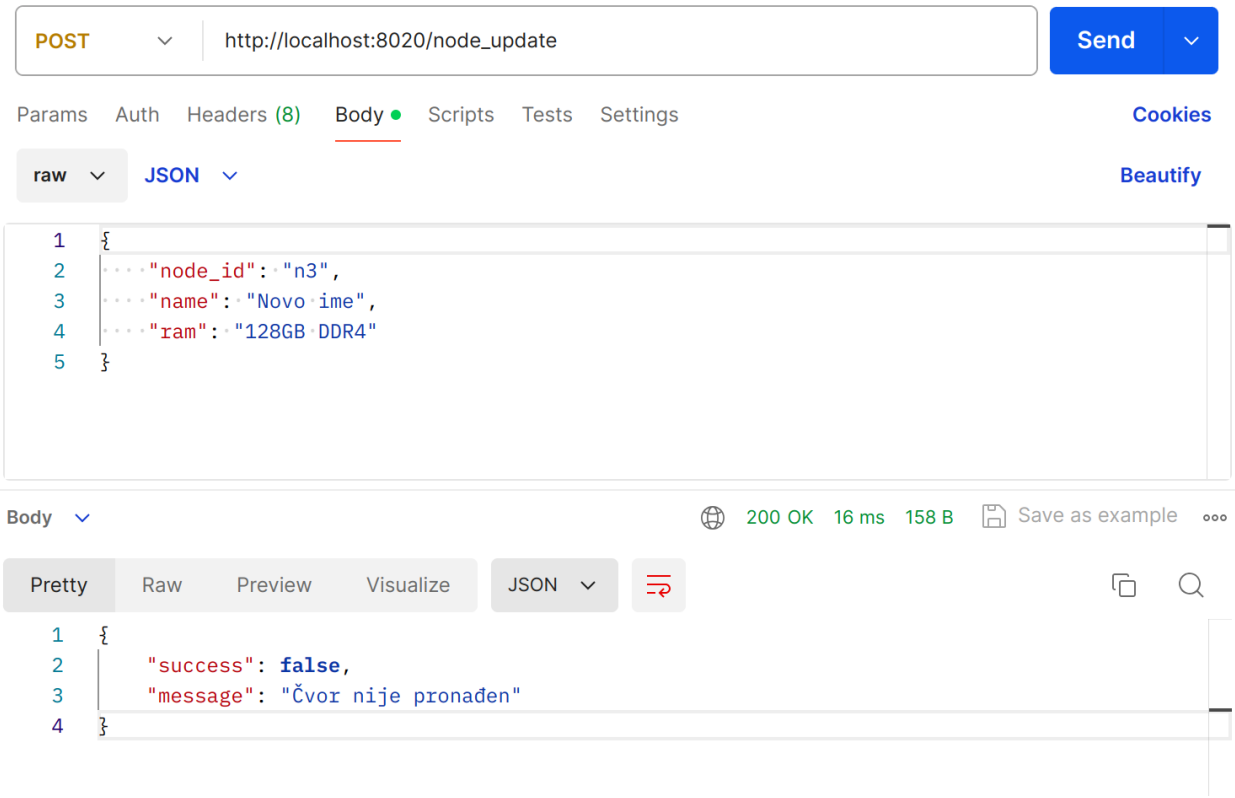
```
// primanje podataka sa API-ja
runtime_handle.spawn(future: async move {
    while let Some((update_node_request: UpdateNodeRequest, resp_tx: Sender<ApiResponse>)) = rx.recv().await {
        let mut nodes: MutexGuard<'_, Vec<Node>> = nodes_arc.lock().unwrap();

        if let Some(node: &mut Node) = nodes.iter_mut().find(|n: &&mut Node| n.id == update_node_request.node_id) {
            api::update_node(node, update_request: update_node_request);

            if let Some(ctx: &Context) = &state_clone.lock().unwrap().ctx {
                ctx.request_repaint();
            }
            let _ = resp_tx.send(ApiResponse { success: true, message: "OK".to_string()});
        } else {
            let _ = resp_tx.send(ApiResponse { success: false, message: "Čvor nije pronađen".to_string() });
        }
    }
});
```

Slika 4.11. Kod za promjenu podataka o čvoru prilikom slanja zahtjeva na API

Na slici 4.12. vidljivo je slanje zahtjeva iz aplikacije Postman na točku programskog sučelja za promjenu atributa čvora. Na slici je od vrha prema dnu moguće vidjeti HTTP metodu i putanju točke programskog sučelja, izgled zahtjeva koji se šalje te primljeni odgovor programskog sučelja koji je na samom dnu slike. Detaljne informacije o svakoj o točki aplikacijskog programskog sučelja te informacije o tipu, strukturi i izgledu zahtjeva i odgovora programskog sučelja nalaze se u Prilogu 2.



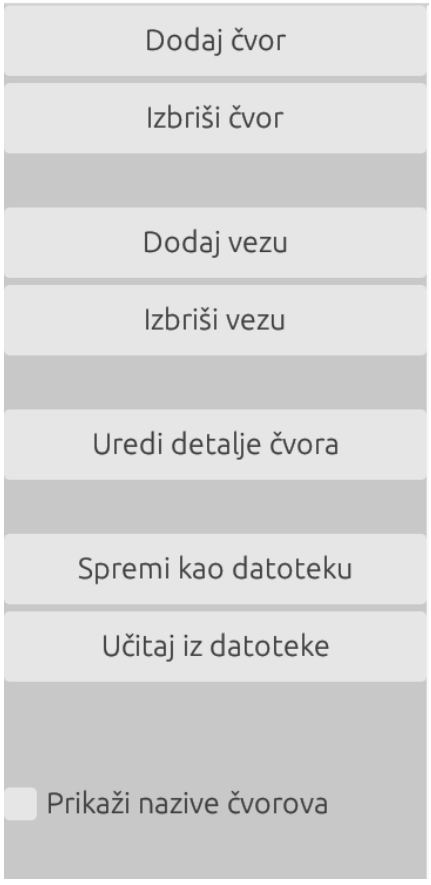
Slika 4.12. Slanje zahtjeva na API točku za izmjenu atributa čvora iz aplikacije Postman

5. Korištenje knjižnice za vizualizaciju peer-to-peer mreža

Prilikom pokretanja izrađene knjižnice za vizualizaciju peer-to-peer mreža, otvara se prozor s grafičkim sučeljem prikazanim na slici 4.6. S lijeve se strane nalazi kontrolna ploča pomoću koje se uređuje konfiguracija peer-to-peer mreže, dok je ostatak prozora namijenjen za prikaz mreže.

5.1. Kontrolna ploča

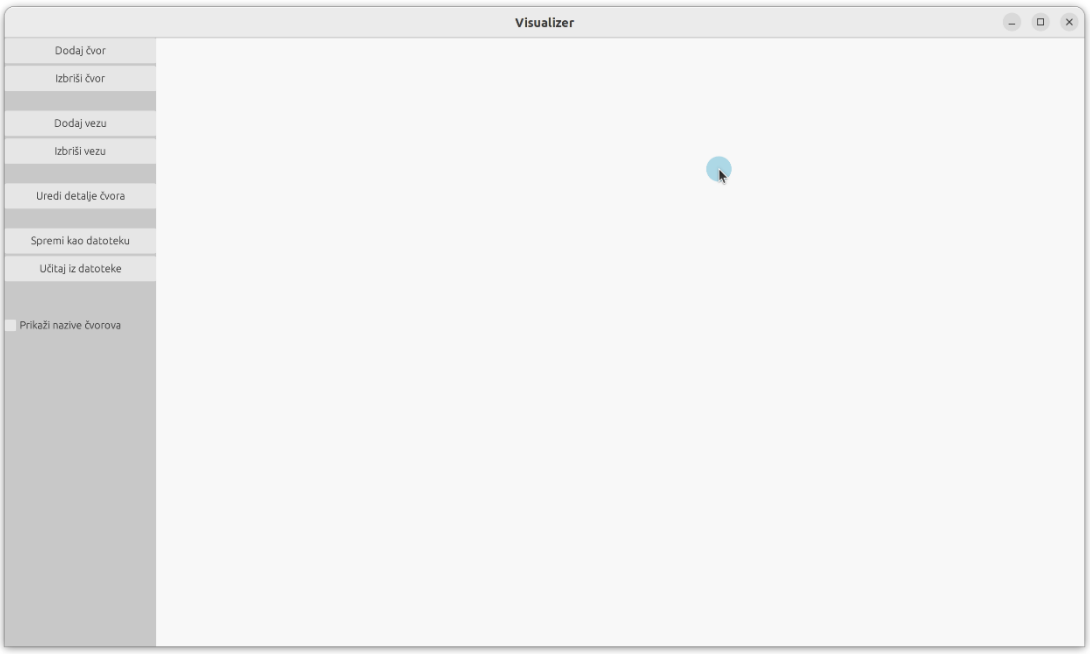
Na slici 5.1. vidljiva je kontrolna ploča. Na njoj se redom nalaze gumbi za dodavanje i brisanje čvorova, dodavanje i brisanje veza, uređivanje podataka o čvoru, spremanje konfiguracije mreže u GEXF ili GraphML datoteku te učitavanje konfiguracije mreže iz istih. Ispod njih nalazi se kvadratić za izbor uz kojeg stoji tekst „Prikaži nazive čvorova“.



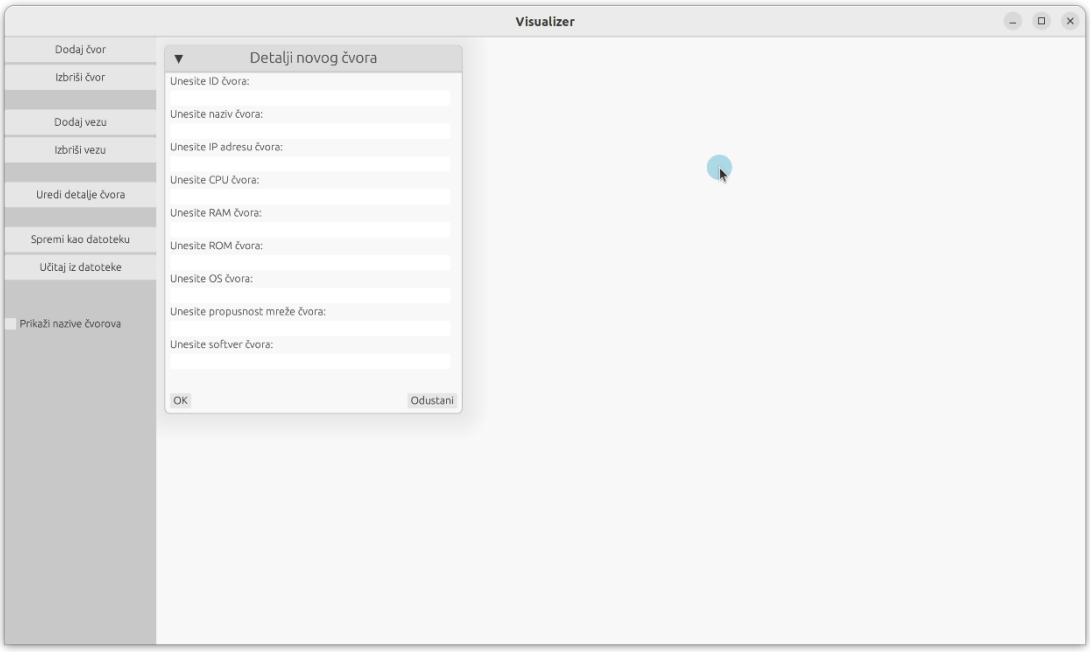
Slika 5.1. Kontrolna ploča

5.2. Dodavanje novog čvora

Klikom na gumb *Dodaj čvor* na kontrolnoj ploči, knjižnica ulazi u stanje za dodavanje novog čvora. Tada se konstantno prikazuje plavi krug na poziciji miša, što je vidljivo na slici 5.2. Plavi se krug također pomiče s pomakom miša. Time se omogućuje odabir pozicije novog čvora. Lijevim klikom miša se odabire pozicija novog čvora te se plavi krug smješta na tu poziciju, odnosno više ne prati miš te se otvara novi prozor u kojeg je moguće unijeti podatke o novom čvoru, a on je prikazan na slici 5.3.

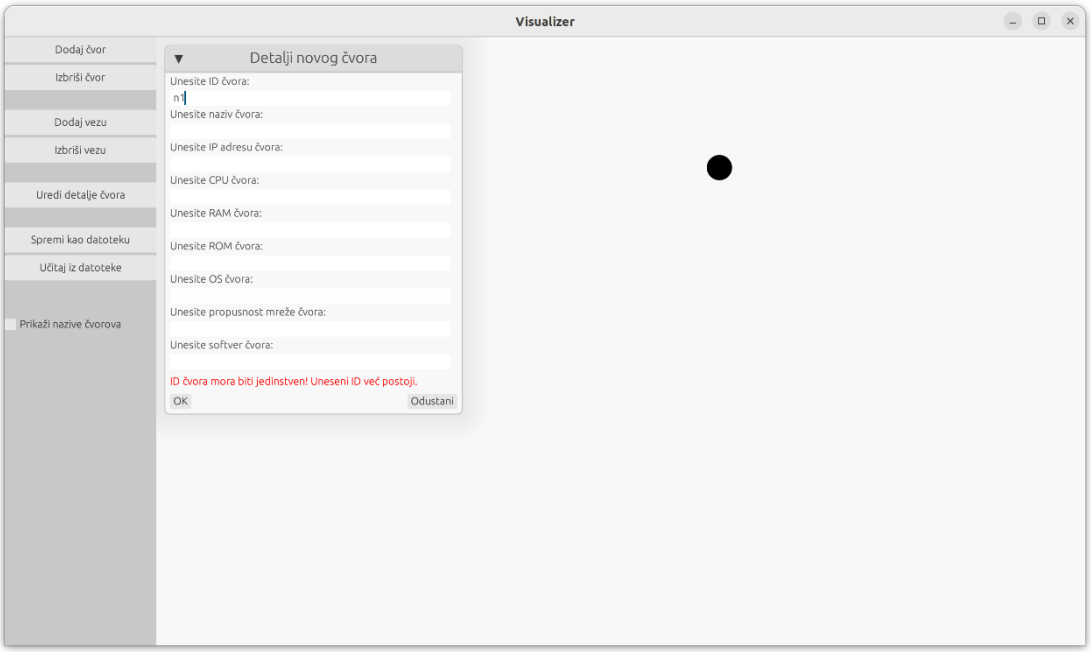


Slika 5.2. Odabir pozicije novog čvora



Slika 5.3. Prozor za unos podataka o novom čvoru

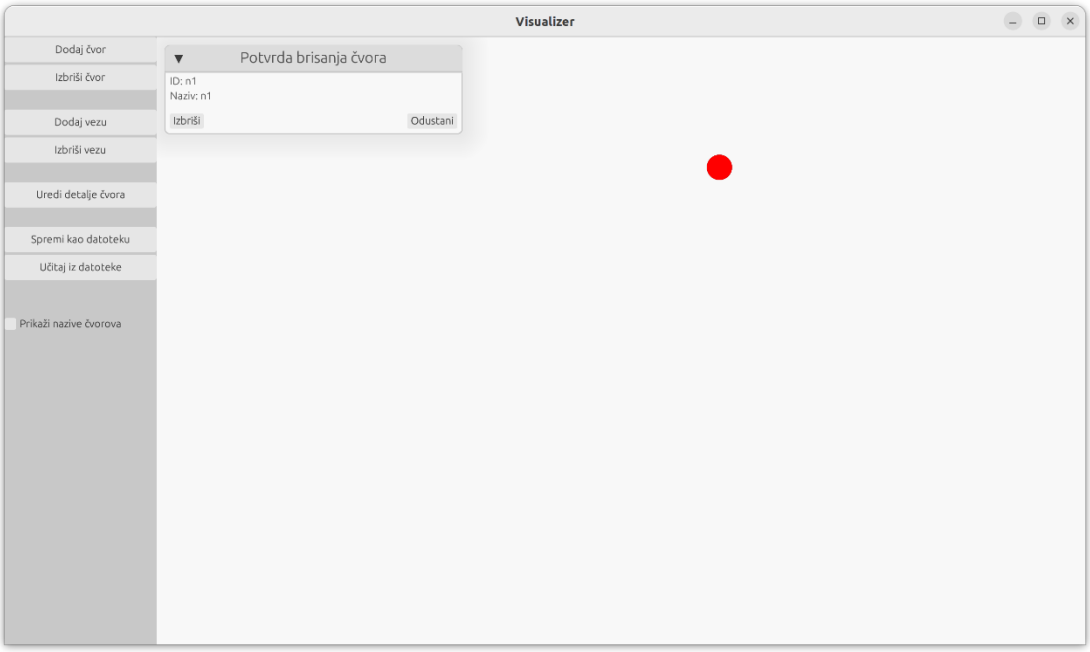
U prozor prikazan na slici 5.3., obavezno je unijeti samo identifikacijsku oznaku (ID) i naziv čvora, dok su ostali podaci opcionalni. Važno je da je ID čvora jedinstven za taj čvor, a ukoliko korisnik unese ID kojeg posjeduje neki od već postojećih čvorova, prikazuje se poruka koja govori da ID mora biti jedinstven. Ta je poruka vidljiva na slici 5.4. Također, ako korisnik unese naziv nekog od već postojećih čvorova, prikazati će mu se poruka kako već postoji čvor s takvim nazivom, ali korisnik može koristiti isti naziv za više čvorova. Klikom na gumb *OK*, dodaje se novi čvor s upisanim podacima te se njegova boja mijenja u crnu, a klikom na *Odustani* se proces dodavanja novog čvora prekida.



Slika 5.4. Poruka da ID mora biti jedinstven prilikom dodavanja novog čvora

5.3. Brisanje čvora

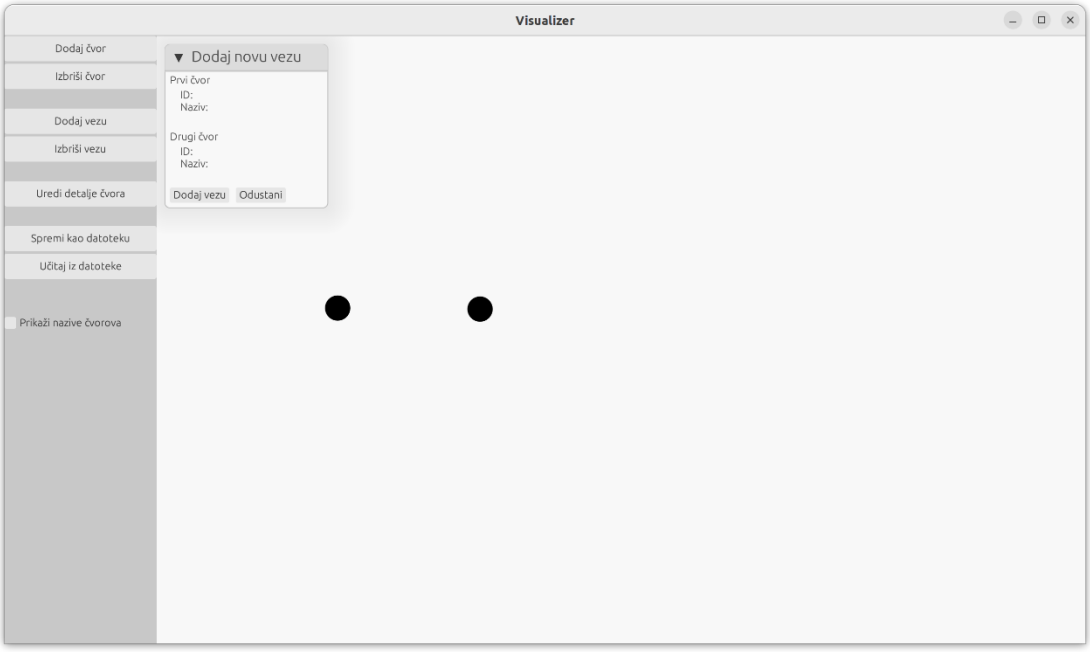
Klikom na gumb *Izbriši čvor* na kontrolnoj ploči, knjižnica ulazi u stanje za brisanje čvorova. Dok se nalazi u tom stanju, kada korisnik prijeđe mišem preko nekog čvora, on promijeni boju u crvenu. Na taj se način odabire čvor za brisanje. Klikom na neki čvor on postaje trajno crven te se otvara novi prozor. Taj prozor služi za potvrdu brisanja čvora, a kao što je i vidljivo na slici 5.5., u njemu je ispisana identifikacijska oznaka i naziv čvora koji je odabran za brisanje. Klikom na gumb *Izbriši* potvrđuje se brisanje te je čvor uklonjen iz vizualizacije, a klikom na gumb *Odustani* prekida se proces brisanja te se boja čvora vraća u onu koja je bila prije odabira čvora.



Slika 5.5. Potvrdni prozor za brisanje čvora

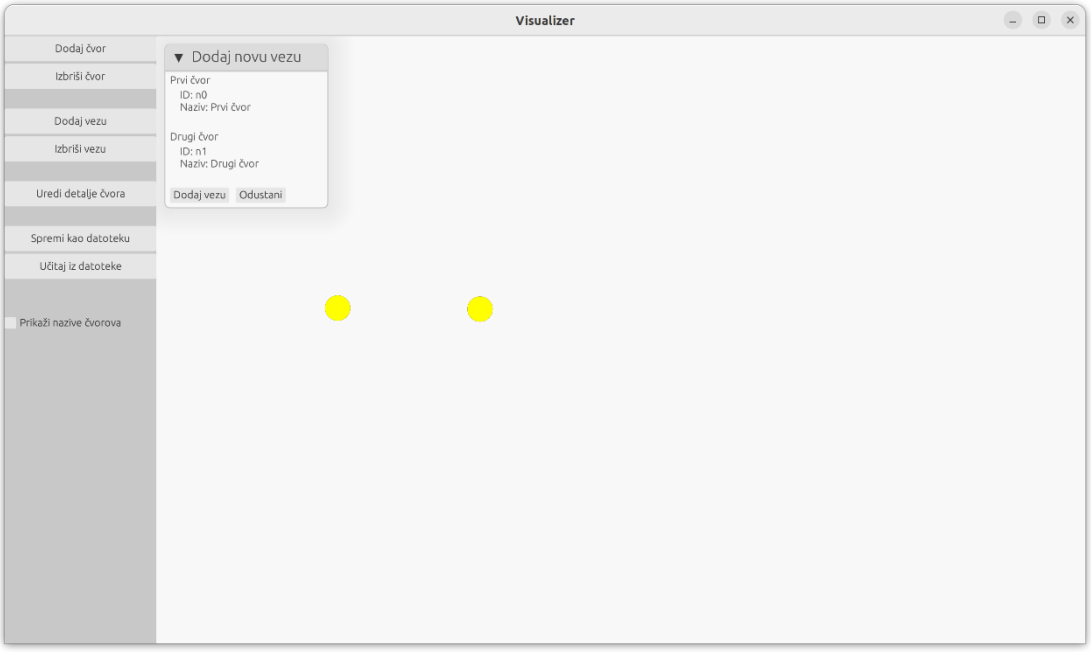
5.4. Dodavanje nove veze

Klikom na gumb *Dodaj vezu* na kontrolnoj ploči, otvara se prozor prikazan na slici 5.6. Novootvoreni prozor prikazuje podatke o čvorovima koji će se povezati. Dok je taj prozor otvoren, ukoliko korisnik prijeđe mišem preko nekog od čvorova, on će privremeno promijeniti boju u žutu. Žutom se bojom označava čvor na koji će se dodati nova veza. Lijevim klikom miša na neki čvor, on postaje trajno žut te se njegovi podaci pojavljuju u prozoru.



Slika 5.6. Prozor za dodavanje nove veze prije odabira čvorova

Kao što je vidljivo na slici 5.7., nakon odabira oba čvora između kojih će se dodati veza, podaci oba čvora se nalaze u prikazanom prozoru te su oba čvora žute boje. Klikom na gumb *Dodaj vezu* u tom prozoru, dodaje se nova veza koja je odmah vidljiva na ekranu, a klikom na gumb *Odustani*, prekida se proces dodavanja nove veze. U oba se slučajeva boje odabranih čvorova vraćaju u prijašnje.



Slika 5.7. Prozor za dodavanje nove veze nakon odabira čvorova

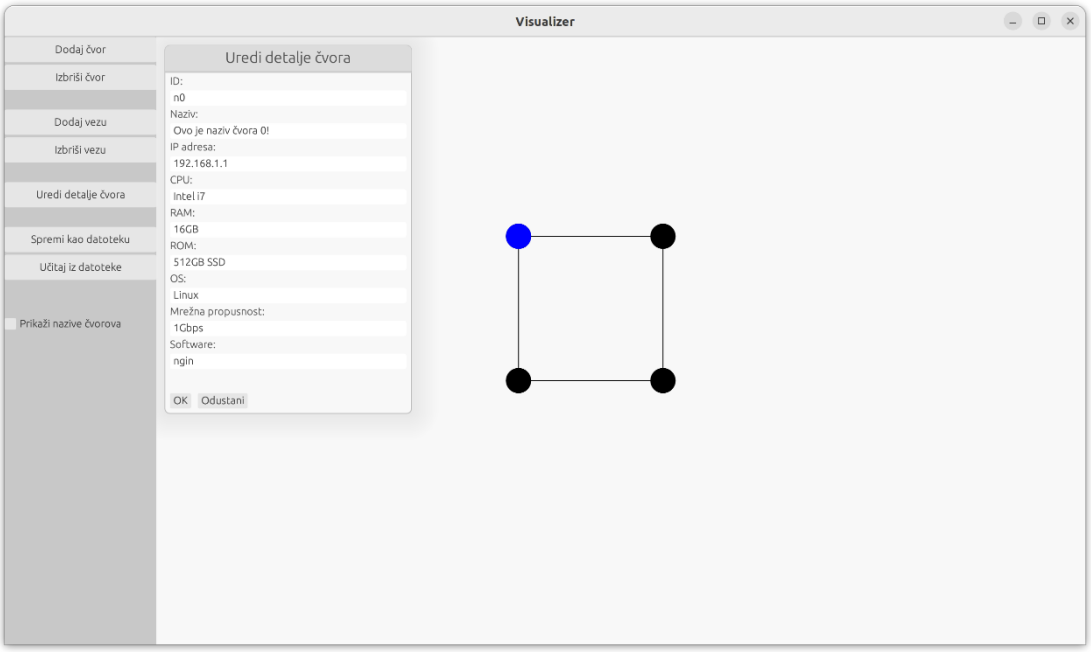
5.5. Brisanje veze

Klikom na gumb *Izbriši vezu* na kontrolnoj ploči, otvara se prozor sličan onome kod dodavanja nove veze. Ponašanje sustava je također isto kao i kod dodavanja nove veze, samo što će se prilikom klika na gumb *Izbriši vezu*, veza između dva odabrana čvora obrisati.

5.6. Uređivanje detalja o čvoru

Klikom na gumb *Uredi detalje čvora* na kontrolnoj ploči, knjižnica ulazi u stanje za uređivanje. U tom stanju, kada korisnik prijeđe mišem preko nekog od čvorova, od promijeni boju u plavu. Tom bojom se označava čvor čiji se podaci žele mijenjati. Lijevim klikom na neki čvor, potvrđuje se odabir. U tom trenutku čvor trajno postaje plave boje i otvara se novi prozor sa svim podacima o čvoru, što je vidljivo na slici 5.8. U otvorenom prozoru je moguće uređivati sve podatke o odabranom čvoru, a klikom na gumb *OK,* spremaju se napravljene izmjene. Kao i kod svih prozora

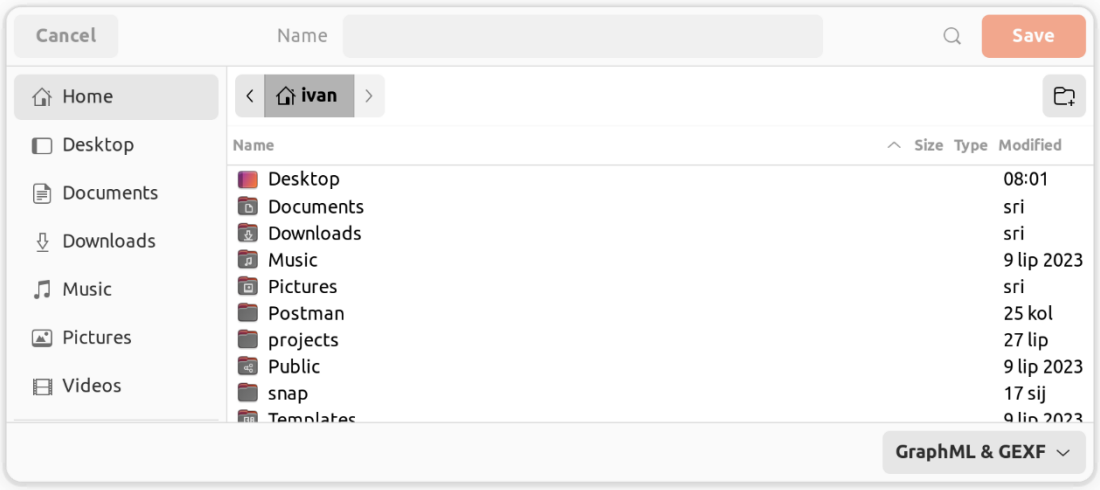
do sada opisanih, klikom na gumb *Odustani*, prekida se uređivanje te se promjene napravljene u otvorenom prozoru neće primijeniti.



Slika 5.8. Prozor za uređivanje detalja o čvoru

5.7. Spremanje konfiguracije mreže iz grafičkog sučelja u GEXF ili GraphML datoteku

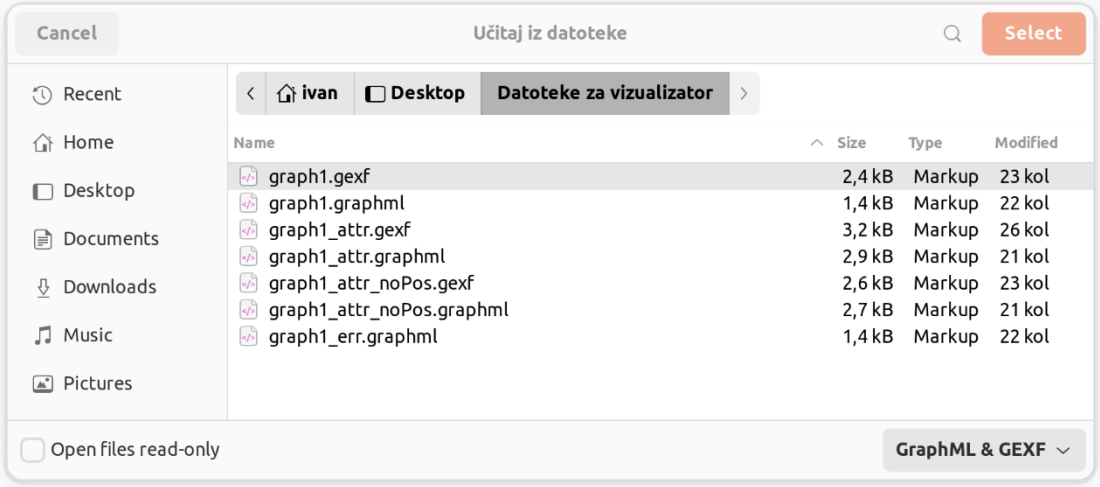
Klikom na gumb *Spremi kao datoteku* na kontrolnoj ploči, otvara se prozor za odabir putanje za spremanje, odnosno mjesta u pohrani gdje će se nova datoteka kreirati. Prozor je vidljiv na slici 5.9. Korisnik u tom prozoru mora navesti i željeno ime nove datoteke. Poželjno je da definira i datotečni nastavak, odnosno format datoteke, koji može biti *„.gexf“* ili *„.graphml“*. Ukoliko je definirani datotečni nastavak *„.gexf“*, knjižnica će na toj putanji kreirati novu GEXF datoteku, a ako je definirani datotečni nastavak *„.graphml“*, knjižnica će kreirati novu GraphML datoteku. Ukoliko datotečni nastavak nije definiran, knjižnica će automatski dodati *„.gexf“* nastavak te kreirati datoteku u GEXF formatu. Ako pak korisnik definira neki drugi datotečni nastavak, knjižnica će mu prikazati novi prozor s porukom kako odabrani format nije podržan.



Slika 5.9. Prozor za odabir putanje nove GEXF ili GraphML datoteke

5.8. Učitavanje postojeće GEXF ili GraphML datoteke

Klikom na gumb *Učitaj iz datoteke* na kontrolnoj ploči, otvara se prozor za odabir datoteke iz memorije računala koji se nalazi na slici 5.10. Odabirom željene datoteke, knjižnica čita njen sadržaj, isti obrađuje, a po obradi prikazuje mrežu na ekranu. Ukoliko korisnik odabere datoteku formata koji nije podržan, u novom će mu se prozoru prikazati poruka koja govori kako odabrani format nije podržan.

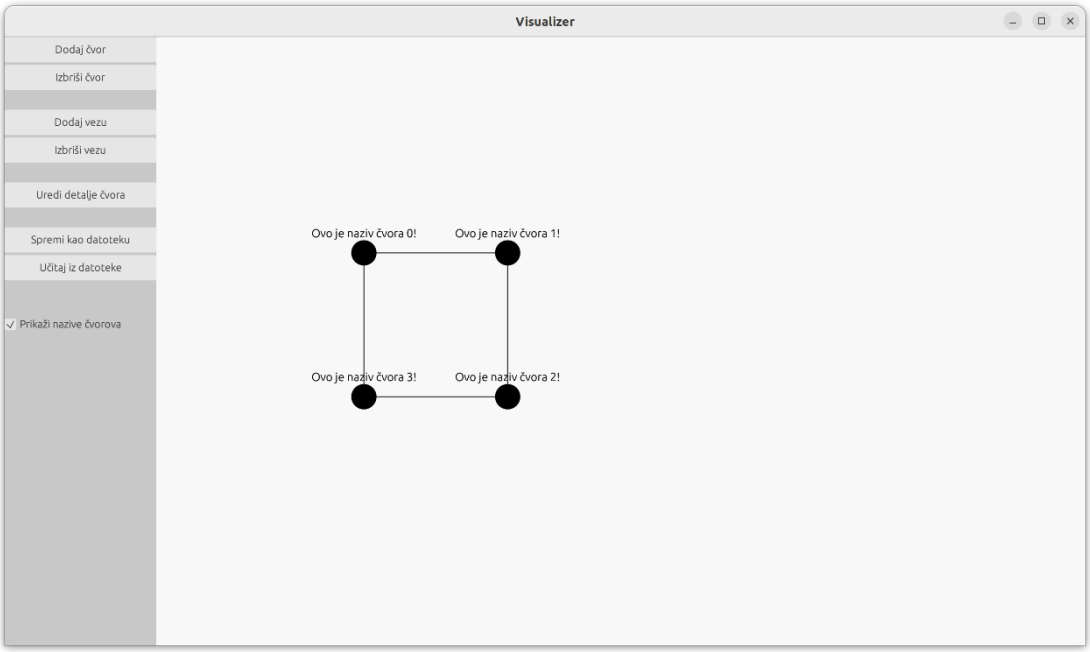


Slika 5.10. Odabir GEXF ili GraphML datoteke za učitavanje

Knjižnica postavlja čvorove prema atributima za definiranje pozicije opisanih u Prilogu 1. Ukoliko u datoteci nedostaje podatak za poziciju nekog čvora, knjižnica će odrediti njegovu poziciju tako da ga pozicionira između svih čvorova s kojima je povezan. Ukoliko pak odabrana datoteka ne sadrži podatke o poziciji za niti jedan čvor, čvorovi će bit postavljeni na ekran jedan do drugoga. Na taj način korisniku je dana mogućnost da pomakne čvorove prema vlastitoj želji.

5.9. Stalni prikaz naziva čvorova

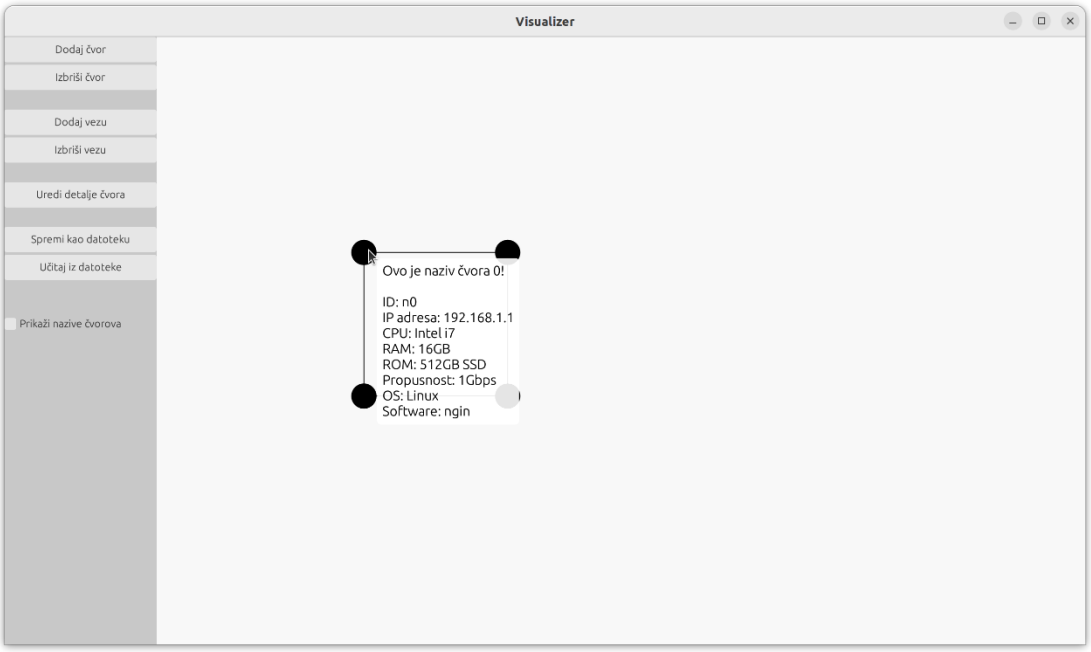
Odabirom, odnosno označavanjem, kvadratića za izbor na dnu kontrolne ploče uz kojeg piše „Prikaži nazive čvorova“, iznad svakog se čvora prikazuje njegov naziv. Isto je vidljivo na slici 5.11. Nazivi će se prikazivati sve dok je kvadratić za izbor označen.



Slika 5.11. Konstantni prikaz naziva čvorova

5.10. Prikaz detalja čvora prelaskom mišem

U izrađenoj je knjižnici omogućeno prikazivanje svih detalja o čvoru kada korisnik postavi miš iznad nekog čvora. Postavljanjem miša iznad čvora, u novom će se prozorčiću prikazati svi njegovi detalji kako je vidljivo na slici 5.12. Kada korisnik pomakne miš tako da više nije iznad čvora, prozorčić s detaljima će nestati. Navedena mogućnost korisna je za istraživanje mreže.



Slika 5.12. Prikaz detalja o čvoru kada je miš pozicioniran iznad čvora

5.11. Pomicanje čvorova i mreže te zumiranje

Klikom na neki čvor omogućuje se njegovo pomicanje po ekranu. Sve dok korisnik drži lijevi gumb miša, odabrani čvor će pratiti poziciju miša. Kada korisnik otpusti lijevi gumb miša, čvor će ostati na poziciji miša u tom trenutku.

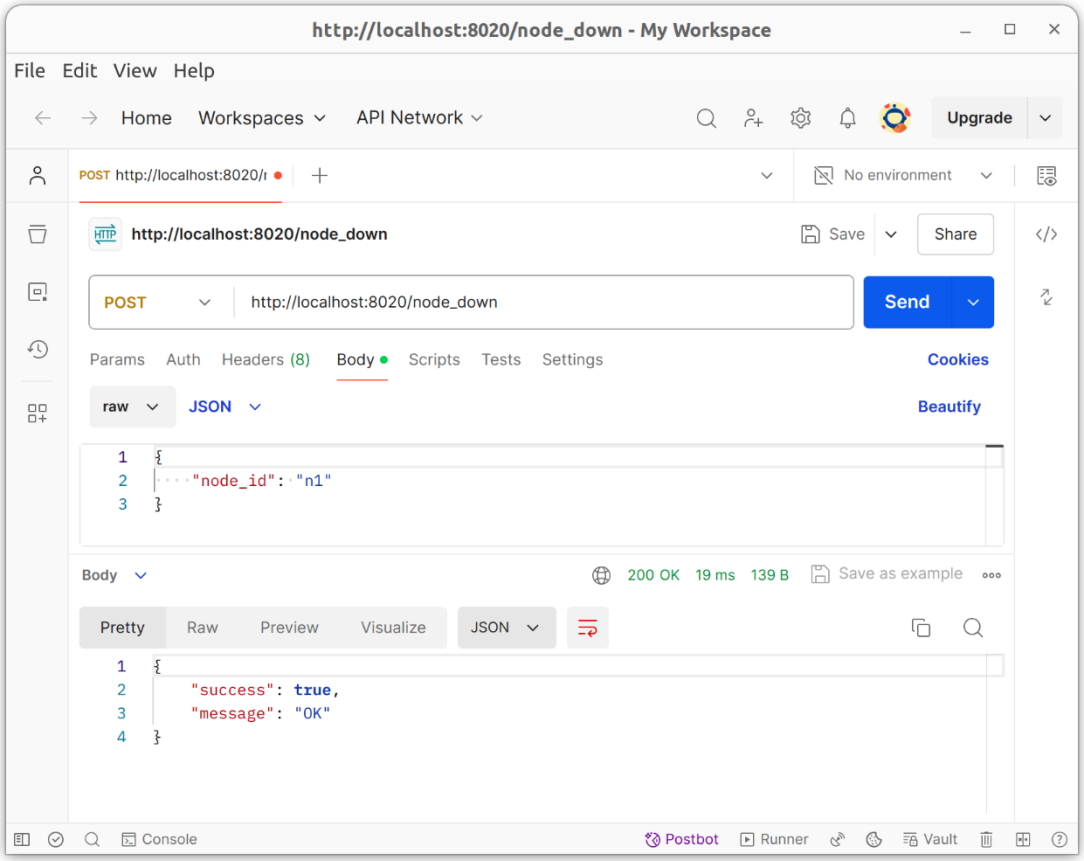
Također je moguće upravljati prikazom cijele mreže. Rotiranjem kotačića za zumiranje na mišu, omogućeno je zumiranje mreže. Ono se izvršava na način da se čvorovi udaljavaju ili približuju. Kada korisnik drži kotačić miša pritisnutim, omogućeno je pomicanje cijele mreže. Dokle god korisnik drži kotačić pritisnutim, svi čvorovi mreže prate pomicanje miša.

5.12. Promjena detalja i statusa čvora putem API-ja

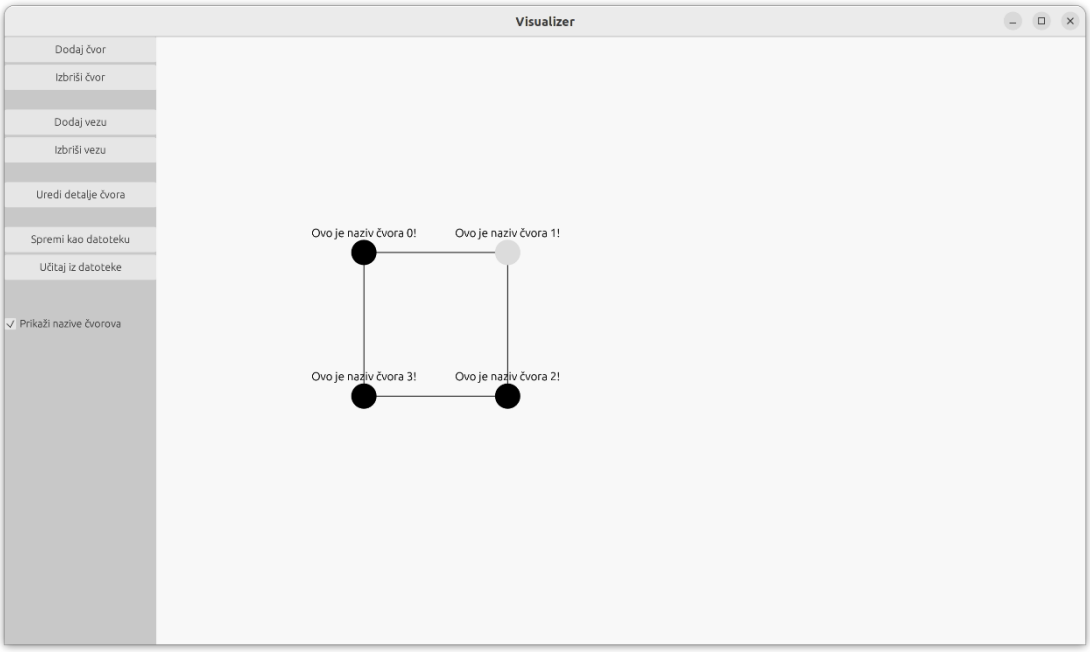
Kako je već opisano u poglavlju 4.8., promjena atributa čvora te njegovog statusa moguća je slanjem zahtjeva na aplikacijsko programsko sučelje. Popis svih točaka API-ja te detalji o strukturi zahtjeva i odgovora za sve točke, nalaze se u API dokumentaciji u Prilogu 2.

Slanjem zahtjeva na točku API-ja za promjenu statusa čvora u stanje nedostupnosti, prikazanog na slici 5.13., koji je u ovom slučaju poslan iz aplikacije *Postman*, njegova se boja u prozoru

vizualizacije mijenja u svijetlo sivu, što je prikazano na slici 5.14., a korisnik u odgovoru API točke dobiva potvrdnu informaciju u vezi zahtjeva što je vidljivo pri dnu slike 5.13.



Slika 5.13. Slanje zahtjeva za nedostupnost čvora iz aplikacije Postman



Slika 5.14. Promjena boje čvora nakon primitka zahtjeva za nedostupnost na točki API-ja

Suprotno od toga, slanjem zahtjeva na točku API-ja zaduženu za promjenu statusa čvora u stanje dostupnosti, boja čvora se u prozoru vizualizacije mijenja u crnu. Također je moguće mijenjati atribute čvora slanjem zahtjeva na API točku za promjenu istih kako je opisano u dokumentaciji u Prilogu 2.

Kako je već ranije navedeno, umjesto ovog API-ja moguće je koristi i javne knjižnične funkcije opisane u Prilogu 3. One se koriste ako je potrebno primijeniti ranije navedene promjene iz projekta koji koristi ovu knjižnicu za vizualizaciju peer-to-peer mreža, dok je API namijenjen za korištenje od strane vanjskih aplikacija i korisnika koji nisu direktno povezani s knjižnicom.

6. Zaključak

Kako peer-to-peer mreže imaju brojne prednosti te kako se zbog tih prednosti uporaba peer-to-peer mreža svakodnevno povećava, moguće je zaključiti da peer-to-peer mreže imaju znatnu ulogu u svijetu informatike i računarstva te je vrlo vjerojatno da će njihova upotreba biti još češća u budućnosti.

Iz ovog je rada vidljivo kako se može napraviti knjižnica za vizualizaciju peer-to-peer mreža te kako je Rust programski jezik vrlo konkurentan jezik zbog svoje sigurnosti. Ta je sigurnost prednost zbog koje popularnost Rust programskog jezika raste, iako mala nepromišljenost prilikom dizajniranja sustava može rezultirati velikim promjenama koda u fazi razvoja, što je rjeđi slučaj u ostalim popularnih programskim jezicima. Također se može vidjeti kako napravljena knjižnica ima mnogo dijelova koji moraju zasebno funkcionirati kako bi vizualizacija bila uspješna. Izrađena knjižnica vrlo je koristan alat, ali ju je moguće nadograditi izradom brojnih sporednih funkcija kako bi se njezina korisnost još dodatno povećala.

Literatura

[1] Schollmeier R.: "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," Proceedings First International Conference on Peer-to-Peer Computing, pp. 101-102, 2001.

[2] Službena internetska stranica programskog jezika Rust: https://www.rust-lang.org/ – pristupljeno 20.8.2024.

[3] Službena internetska stranica GEXF formata: https://gexf.net/ – pristupljeno 22.8.2024.

[4] Službena internetska stranica GraphML formata: http://graphml.graphdrawing.org/ – pristupljeno 22.8.2024.

Prilog 1: Specifikacije GEXF i GraphML datoteka

Kako bi knjižnica za vizualizaciju peer-to-peer mreža mogla uspješno pročitati sadržaj GEXF i GraphML datoteka koje se u nju učitaju, sadržaj tih datoteka mora biti prema službenim specifikacijama navedenih formata, a dodatni atributi koje ova knjižnica podržava moraju biti definirani prema ovim specifikacijama.

Dodatni atributi čvorova koje knjižnica podržava

- **name** – naziv čvora (kao niz znakova) – samo za GraphML datoteke jer GEXF format prema svojim specifikacijama omogućuje zapis naziva čvora atributom **label** u <node> elementu (primjer za GEXF format: <node id="n0" label="Naziv čvora n0">)
- **pos_x** – pozicija čvora u grafičkom sučelju – vrijednost apscise u pikselima (cijeli broj) ako se grafičko sučelje smatra pravokutnim koordinatnim sustavom s ishodištem u gornjem lijevom kutu i pozitivnim vrijednostima prema desnoj strani prozora sučelja (namijenjeno prvenstveno za internu uporabu, a ako nije definirano knjižnica će sama definirati vrijednost za koju smatra da je najbolji izbor prilikom vizualizacije)
- **pos_y** – pozicija čvora u grafičkom sučelju – vrijednost ordinate u pikselima (cijeli broj) ako se grafičko sučelje smatra pravokutnim koordinatnim sustavom s ishodištem u gornjem lijevom kutu i pozitivnim vrijednostima prema donjem dijelu prozora sučelja (namijenjeno prvenstveno za internu uporabu, a ako nije definirano knjižnica će sama definirati vrijednost za koju smatra da je najbolji izbor prilikom vizualizacije)
- **ip_addr** – IP adresa čvora (kao niz znakova)
- **cpu** –procesor kojeg čvor koristi (kao niz znakova)
- **ram** – radna memorija (kao niz znakova)
- **rom** – sekundarna memorija za pohranu podataka (kao niz znakova)
- **os** – operacijskog sustava kojeg čvor koristi (kao niz znakova)
- **network_bw** – propusnost mrežnog sučelja čvora (kao niz znakova)
- **software** – softver koji se na čvoru izvršava (kao niz znakova)

Svi navedeni atributi su opcionalni te se svi atributi osim *pos_x* i *pos_y* zapisuju se kao niz znakova kako bi korisnik imao mogućnost prilagodbe navedenih atributa prema vlastitoj želji ili potrebi.

Atributi *pos_x* i *pos_y* namijenjeni su prvenstveno za internu uporabu. Ako nisu definirani u datoteci, knjižnica će sama definirati vrijednost za koju smatra da je najbolji izbor prilikom vizualizacije. Ukoliko se pojedini čvorovi nalaze na mjestu kontrolne ploče grafičkog sučelja, isti će biti pomaknuti u desnu stranu kako bi bili vidljivi.

Elementi GEXF datoteke

Prema službenim specifikacijama GEXF formata, svaka datoteka ovog tipa mora imati korijenski element **<gexf>** u kojem se definiraju prostor imena atributom **xmlns** i verzija atributom **version**. Zatim slijedi **<graph>** element koji definira jedan graf, a može imati atribut **mode**, koji može imati vrijednost **static** ili **dynamic** ovisno prati li se mreža kroz vrijeme, i atribut **defaultedgetype** koji definira zadani tip veze, a može poprimiti vrijednost **undirected** za neusmjerene veze ili **directed** za usmjerene veze.

Zatim se definiraju svi dodatni atributi unutar **<attributes>** elementa sa atributom **class** kojim se određuje za koje se elemente datoteke definiraju ti atributi. Sami se atributi definiraju pomoću **<attribute>** elementa.

Slijedi **<nodes>** element unutar kojeg se definiraju svi čvorovi. Pojedinačni se čvor definira sa elementom **<node>** koji ima obavezan atribut **id** koji sadrži jedinstvenu identifikacijsku oznaku čvora te opcionalan atribut **label** kojim se definira naziv čvora. Dodatni se atributi definiraju unutar **<attvalues>** elementa postavljenog unutar **<node>** elementa čvora za kojeg se atributi definiraju. Svaki se atribut definira elementom **<attvalue>** koji ima atribute **for** i **value**. Atribut **for** sadrži identifikacijsku oznaku atributa, a **value** sadrži vrijednost tog atributa.

Nakon čvorova, potrebno je odrediti veze. To se radi unutar elementa **<edges>**. Svaka se veza određuje elementom **<edge>** koji prima atribute **id** (jedinstvena identifikacijska oznaka veze), **source** (ishodišni čvor veze) i **target** (odredišni čvor veze). Kod neusmjerenih veza nije bitno koji će čvor bit ishodišni, a koji odredišni.

Elementi GraphML datoteke

Prema službenim specifikacijama GraphML formata, svaka datoteka ovog tipa mora imati korijenski element **<graphml>** u kojem se definiraju prostor imena atributom **xmlns** i shema uz pomoć atributa **xmlns:xsi** i **xsi:schemaLocation**. Zatim se definiraju svi dodatni atributi pomoću **<key>** elementa. Svaki **<key>** element ima atribute **id** (kojim se određuje identifikacijska oznaka atributa), **for** (kojim se određuje za koje se elemente datoteke definiraj taj atribut), **attr.name** (kojim se definira naziv tog atributa) te **attr.type** (kojim se definira vrsta podatka koji taj atribut može poprimiti).

Zatim slijedi **<graph>** element koji definira jedan graf, koji ima atribute **id** (koji predstavlja jedinstvenu identifikacijsku oznaku tog grafa) te **edgedefault** (koji definira zadani tip veze, a može poprimiti vrijednost **undirected** za neusmjerene veze ili **directed** za usmjerene veze).

Zatim se definiraju čvorovi elementima **<node>** koji imaju obavezan atribut **id** koji sadrži jedinstvenu identifikacijsku oznaku čvora. Dodatni se atributi definiraju elementima **<data>** postavljenim unutar **<node>** elementa čvora za kojeg se atributi definiraju. Svaki **<data>** element ima atribut **key** koji sadrži identifikacijsku oznaku atributa, a unutar tog **<data>** elementa se definira vrijednost za taj atribut.

Nakon čvorova, potrebno je odrediti veze. To se radi elementima **<edge>** koji primaju atribute **id** (jedinstvena identifikacijska oznaka veze), **source** (ishodišni čvor veze) i **target** (odredišni čvor veze). Kod neusmjerenih veza nije bitno koji će čvor bit ishodišni, a koji odredišni.

Primjer pravilno formatirane GEXF datoteke

```
<?xml version="1.0" encoding="UTF-8"?>
<gexf xmlns="http://gexf.net/1.2" version="1.2">
  <graph mode="static" defaultedgetype="undirected">
    <attributes class="node">
      <attribute id="pos_x" title="pos_x" type="integer"/>
      <attribute id="pos_y" title="pos_y" type="integer"/>
      <attribute id="ip_addr" title="ip_addr" type="string"/>
      <attribute id="cpu" title="cpu" type="string"/>
      <attribute id="ram" title="ram" type="string"/>
      <attribute id="rom" title="rom" type="string"/>
      <attribute id="os" title="os" type="string"/>
      <attribute id="network_bw" title="network_bw" type="string"/>
      <attribute id="software" title="software" type="string"/>
    </attributes>
    <nodes>
      <node id="n0" label="Naziv čvora 0">
        <attvalues>
          <attvalue for="pos_x" value="250"/>
          <attvalue for="pos_y" value="150"/>
          <attvalue for="ip_addr" value="192.168.1.1"/>
          <attvalue for="cpu" value="Intel i7"/>
          <attvalue for="ram" value="16GB"/>
          <attvalue for="rom" value="512GB SSD"/>
          <attvalue for="os" value="Linux"/>
          <attvalue for="network_bw" value="1Gbps"/>
          <attvalue for="software" value="ngin"/>
        </attvalues>
      </node>
      <node id="n1" label="Naziv čvora 1">
        <attvalues>
          <attvalue for="ip_addr" value="192.168.1.2"/>
          <attvalue for="cpu" value="Intel i5"/>
          <attvalue for="ram" value="8GB"/>
          <attvalue for="rom" value="256GB SSD"/>
          <attvalue for="os" value="Windows"/>
          <attvalue for="network_bw" value="100Mbps"/>
        </attvalues>
      </node>
    </nodes>
    <edges>
      <edge id="e0" source="n0" target="n1"/>
    </edges>
  </graph>
</gexf>
```

Primjer pravilno formatirane GraphML datoteke

```
<?xml version="1.0" encoding="UTF-8"?>
<graphml xmlns="http://graphml.graphdrawing.org/xmlns"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
      http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">

   <key id="name" for="node" attr.name="name" attr.type="string"/>
   <key id="pos_x" for="node" attr.name="pos_x" attr.type="integer"/>
   <key id="pos_y" for="node" attr.name="pos_y" attr.type="integer"/>
   <key id="ip_addr" for="node" attr.name="ip_addr" attr.type="string"/>
   <key id="cpu" for="node" attr.name="cpu" attr.type="string"/>
   <key id="ram" for="node" attr.name="ram" attr.type="string"/>
   <key id="rom" for="node" attr.name="rom" attr.type="string"/>
   <key id="os" for="node" attr.name="os" attr.type="string"/>
   <key id="network_bw" for="node" attr.name="network_bw" attr.type="string"/>
   <key id="software" for="node" attr.name="software" attr.type="string"/>

   <graph id="G" edgedefault="undirected">
      <node id="n0">
         <data key="name"Naziv čvora 0</data>
         <data key="pos_x">250</data>
         <data key="pos_y">150</data>
         <data key="ip_addr">192.168.1.1</data>
         <data key="cpu">Intel i7</data>
         <data key="ram">16GB</data>
         <data key="rom">512GB SSD</data>
         <data key="os">Linux</data>
         <data key="network_bw">1Gbps</data>
         <data key="software">ngin</data>
      </node>
      <node id="n1">
         <data key="name">Naziv čvora 1</data>
         <data key="ip_addr">192.168.1.2</data>
         <data key="cpu">Intel i5</data>
         <data key="ram">8GB</data>
         <data key="rom">256GB SSD</data>
         <data key="os">Windows</data>
         <data key="network_bw">100Mbps</data>
      </node>

      <edge id="e0" source="n0" target="n1"/>
   </graph>
</graphml>
```

Prilog 2: API dokumentacija

Čvor nije dostupan	
Opis	API točka za promjenu dostupnosti čvora u mreži. Slanjem zahtjeva na ovu točku označava se da pojedini čvor više nije dostupan u mreži.
Port	8020
URI	/node_down
HTTP metoda	POST
Tip zahtjeva	Application/JSON
Zaglavlje zahtjeva	
Tijelo zahtjeva	JSON objekt koji sadrži vrijednost: • node_id: String, obavezan – označava čvor koji postaje nedostupan
Tip odgovora	Application/JSON
Zaglavlje odgovora	
Tijelo odgovora	JSON objekt sa vrijednostima: • success: boolean, obavezan – označava je li radnja uspješna • message: String, obavezan – povratna informacija o obradi zahtjeva Ukoliko je obrada zahtjeva uspješna, success će biti „true“, a message će biti „OK“ te se u vizualizatoru čvor označava nedostupnim. Ukoliko je došlo do greške prilikom obrade zahtjeva, success će biti „false“, a message će sadržavati poruku o grešci.

Čvor je dostupan	
Opis	API točka za promjenu dostupnosti čvora u mreži. Slanjem zahtjeva na ovu točku označava se da je pojedini čvor ponovno dostupan u mreži.
Port	8020
URI	/node_up
HTTP metoda	POST
Tip zahtjeva	Application/JSON
Zaglavlje zahtjeva	
Tijelo zahtjeva	JSON objekt koji sadrži vrijednost: • node_id: String, obavezan – označava čvor koji ponovno postaje dostupan
Tip odgovora	Application/JSON
Zaglavlje odgovora	
Tijelo odgovora	JSON objekt sa vrijednostima: • success: boolean, obavezan – označava je li radnja uspješna • message: String, obavezan – povratna informacija o obradi zahtjeva Ukoliko je obrada zahtjeva uspješna, success će biti „true“, a message će biti „OK“ te se u vizualizatoru čvor označava ponovno dostupnim. Ukoliko je došlo do greške prilikom obrade zahtjeva, success će biti „false“, a message će sadržavati poruku o grešci.

Promjena atributa čvora	
Opis	API točka za promjenu atributa pojedinog čvora. Čvor koji se želi mijenjati definira se njegovom ID oznakom, a atributi koje je moguće mijenjati nalaze u definiciji tijela zahtjeva.
Port	8020
URI	/node_update
HTTP metoda	POST
Tip zahtjeva	Application/JSON
Zaglavlje zahtjeva	
Tijelo zahtjeva	JSON objekt koji sadrži sljedeće vrijednosti: • node_id: String, obavezan – označava čvor koji se mijenja • name: String, opcionalan – novo ime čvora • ip_addr: String, opcionalan – nova IP adresa čvora • cpu: String, opcionalan – nova CPU specifikacija čvora • ram: String, opcionalan – nova RAM specifikacija čvora • rom: String, opcionalan – nova ROM specifikacija čvora • os: String, opcionalan – novi operacijski sustav čvora • network_bw: String, opcionalan – nova specifikacija propusnosti mreže kartice čvora • software: String, opcionalan – novi softver čvora Atribute koji se ne mijenjaju nije potrebno definirati.
Tip odgovora	Application/JSON
Zaglavlje odgovora	
Tijelo odgovora	JSON objekt sa vrijednostima: • success: boolean, obavezan – označava je li radnja uspješna • message: String, obavezan – povratna informacija o obradi zahtjeva Ukoliko je obrada zahtjeva uspješna, success će biti „true“, a message će biti „OK“. Ukoliko je došlo do greške prilikom obrade zahtjeva, success će biti „false“, a message će sadržavati poruku o grešci.

Prilog 3: Javne knjižnične funkcije

init_visualizer	
Opis	Inicijalizira i vraća novi objekt koji predstavlja glavni objekt knjižnice za vizualizaciju. Ova funkcija se koristi za stvaranje početnog stanja knjižnice prije nego što se pokrene vizualizacija.
Definicija	pub fn init_visualizer() -> MyApp
Parametri	
Povratna vrijednost	MyApp – funkcija vraća instancu strukture MyApp koja je spremna za daljnje korištenje i pokretanje vizualizacije

start_visualizer	
Opis	Pokreće glavni vizualizaciju, inicijalizira sve potrebne dretve i započinje eframe grafičko korisničko sučelje (GUI). Funkcija također pokreće aplikacijsko programsko sučelje (API).
Definicija	pub fn start_visualizer(mut self) -> Result<(), eframe::Error>
Parametri	
Povratna vrijednost	Result<(), eframe::Error> – za uspješno izvršavanje vizualizacije funkcija vraća 'Ok(())', dok u slučaju greške vraća 'eframe::Error'

node_down	
Opis	Postavlja čvor u stanje nedostupnosti tako da mu promijeni boju u svijetlo sivu. Ova se funkcija poziva kad neki čvor postane nedostupan u mreži.
Definicija	pub fn node_down(&self, node_id: String) -> Result<(), String>
Parametri	node_id: String – ID čvora koji treba postaviti kao nedostupan
Povratna vrijednost	Result<(), String> – ako je čvor pronađen i boja mu je promijenjena, funkcija vraća 'Ok(())', a u slučaju greške vraća 'Err("Čvor nije pronađen")'

node_up	
Opis	Postavlja čvor u stanje dostupnosti tako da mu promijeni boju u crnu. Ova se funkcija poziva kad neki čvor ponovno postane dostupan u mreži.
Definicija	pub fn node_up(&self, node_id: String) -> Result<(), String>
Parametri	node_id: String – ID čvora koji treba postaviti kao dostupan
Povratna vrijednost	Result<(), String> – ako je čvor pronađen i boja mu je promijenjena, funkcija vraća 'Ok(())', a u slučaju greške vraća 'Err("Čvor nije pronađen")'

change_node_attributes	
Opis	Mijenja atribute čvora na temelju zahtjeva formatiranog u obliku strukture *UpdateNodeRequest* iz modula *models*. Promjena boje čvora nije dozvoljena unutar ove funkcije jer je rezervirana za funkcije *node_down* i *node_up* te se vraća greška ako se pokuša promijeniti boju.
Definicija	pub fn change_node_attributes(&self, update_node_request: UpdateNodeRequest) -> Result<(), String>
Parametri	update_node_request: models::UpdateNodeRequest – struktura koja sadrži nove atribute čvora koji se trebaju promijeniti, a može imati slijedeće vrijednosti: • node_id: String, obavezan – označava čvor koji se mijenja • name: String, opcionalan – novo ime čvora • ip_addr: String, opcionalan – nova IP adresa čvora • cpu: String, opcionalan – nova CPU specifikacija čvora • ram: String, opcionalan – nova RAM specifikacija čvora • rom: String, opcionalan – nova ROM specifikacija čvora • os: String, opcionalan – novi operacijski sustav čvora • network_bw: String, opcionalan – nova specifikacija propusnosti mreže kartice čvora • software: String, opcionalan – novi softver čvora Atribute koji se ne mijenjaju nije potrebno definirati.
Povratna vrijednost	Result<(), String> – ako je čvor uspješno ažuriran, vraća 'Ok(())', a u slučaju greške vraća 'Err("Čvor nije pronađen")'

Primjer uporabe knjižničnih funkcija

U programskom kodu koji slijedi, prikazana je upotreba i pozivanje knjižničnih funkcija na jednom vrlo jednostavnom primjeru. Programski kod ovog primjera inicijalizira i pokreće vizualizaciju, čeka deset sekundi pa zatim označuje odabrani čvor nedostupnim. Nakon toga čeka još pet sekundi te isti čvor označava ponovno dostupnim. Zatim ponovno čeka pet sekundi te ažurira podatke o odabranom čvoru.

U ovom je primjeru važno kreirati (ili učitati iz datoteke) čvor *„n0“* unutar deset sekundi između pokretanja vizualizatora i poziva funkcije za označavanje čvora nedostupnim. U suprotnom funkcionalnost ovog koda neće biti vidljiva u grafičkom sučelju knjižnice za vizualizaciju.

Programski kod primjera:

```
use p2p_network_visualizer::init_visualizer;
use p2p_network_visualizer::models::UpdateNodeRequest;

use std::thread::{sleep, spawn};
use std::time::Duration;

fn main() {
    // inicijalizacija vizualizatora
    let app = init_visualizer();
    let app2 = app.clone();

    // kreiranje nove dretve u kojoj će se pozivati funkcije za promjenu statusa i atributa čora
    spawn(move || {
        sleep(Duration::from_secs(10));                 // čekanje 10 sekundi
        let _ = app.node_down("n0".to_string());        // označavanje čvora „n0“ nedostupnim

        sleep(Duration::from_secs(5));                  // čekanje 5 sekundi
        let _ = app.node_up("n0".to_string());          // označavanje čvora „n0“ dostupnim

        sleep(Duration::from_secs(5));                  // čekanje 5 sekundi

        // promjena atributa čvora „n0“
        let _ = app.change_node_attributes(UpdateNodeRequest {
            node_id: "n0".to_string(),
            name: Some("Novi naziv".to_string()),
            cpu: Some("Dvojezgreni".to_string()),
            ..Default::default()
        });
    });

    // pokretanje vizualizacije i grafičkog sučelja
    let _ = app2.start_visualizer();
}
```

Sažetak

U ovom je radu opisana izrada knjižnice za vizualizaciju peer-to-peer mreža u programskom jeziku Rust. Objašnjeni su svi ključni pojmovi koji se vežu za peer-to-peer mreže, Rust programski jezik te navedenu knjižnicu. Neki od njih su višedretvenost, formati za spremanje konfiguracije mreža te aplikacijsko programsko sučelje. Uz navedeno, opisano je i kako se izrađena knjižnica koristi te koje su sve njezine mogućnosti.

Ključne riječi: peer-to-peer mreže, Rust programski jezik, knjižnica, vizualizacija

Abstract

This paper describes the development of a library for the visualization of peer-to-peer networks in the Rust programming language. All key terms related to peer-to-peer networks, the Rust programming language and the specified library are explained. Some of them are multithreading, formats for saving network configuration and application programming interface. In addition to the above, it is also described how the created library is used and what all its possibilities are.

Keywords: peer-to-peer networks, Rust programming language, library, vizualisation