

**Veleučilište Hrvatsko zagorje Krapina**

**FILIP ŽGELA**

**Web-Aplikacija za praćenje i evidenciju mrežnih priključaka na preklopnicima**

**Završni rad**

**Krapina, 2025.**



**Veleučilište Hrvatsko zagorje Krapina**

**Stručni preddiplomski studij Informatike**

**Web-Aplikacija za praćenje i evidenciju mrežnih priključaka na preklopnicima**

**Završni rad**

|  |  |
| --- | --- |
| **Mentor:**  **Stjepan Šalković, dipl.inf.univ.spec.oec.** | **Student:**  **Filip Žgela, matični broj** |

**Krapina, rujan 2025.**



**Veleučilište Hrvatsko zagorje Krapina**

Krapina,

**ZADATAK ZAVRŠNOG RADA**

|  |  |
| --- | --- |
| ***Pristupnik:*** |  |
| ***Matični broj:*** |  |
| ***Prijediplomski studij:*** |  |

|  |  |
| --- | --- |
| Naslov rada: |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Znanstveno područje: |  | Znanstveno polje: |  |

Opis zadatka:

|  |
| --- |
|  |

Mentor: Voditelj studija:

Datum uručenja zadatka: Datum predaje završnog rada:

**SAŽETAK**

Ovaj završni rad opisuje razvoj web-aplikacije za praćenje i evidenciju mrežnih priključaka u poslovnom okruženju. Sustav omogućuje unos, pregled, uređivanje i pretraživanje mrežnih uređaja (switcheva), portova, VLAN-ova te pripadajućih lokacija, katova i soba. Backend aplikacije razvijen je u Spring Boot okviru uz korištenje JPA i Hibernate tehnologije za rad s bazom podataka. Frontend dio izrađen je u React.js pomoću Axios biblioteke za komunikaciju s REST API-em. Posebna pažnja posvećena je funkcionalnostima kao što su kaskadni dropdowni i pretraga po imenu korisnika i utičnici. Aplikacija je testirana pomoću Postmana i ručnim unosom podataka kroz korisničko sučelje. Rezultat je jednostavno i funkcionalno rješenje prilagođeno krajnjem korisniku.

**Ključne riječi:** mrežna infrastruktura, switch, port, VLAN, React, Spring Boot, REST API

**SUMMARY**

This thesis presents the development of a web application for monitoring and managing network connections in a business environment. The system enables the input, viewing, editing, and searching of network switches, ports, VLANs, and their associated locations, floors, and rooms. The backend is built using the Spring Boot framework with JPA and Hibernate for database handling. The frontend is developed in React.js, utilizing the Axios library for communication with the REST API. Special attention is given to functionalities such as cascading dropdowns and search by socket label or user name. The application was tested using Postman and manual data input via the user interface. The result is a simple and efficient solution tailored for end-user needs.

**Keywords:** network infrastructure, switch, port, VLAN, React, Spring Boot, REST API

**Sadržaj**

[1. Uvod 1](#_Toc206703622)

[1.1. Predmet i cilj rada 1](#_Toc206703623)

[1.2. Izvori podataka 2](#_Toc206703624)

[1.3. Korištene tehnologije i razvojno okruženje 2](#_Toc206703625)

[1.4. Sadržaj i struktura rada 3](#_Toc206703626)

[2. Opis problema i funkcionalni zahtjevi 5](#_Toc206703627)

[2.1. Izazovi u upravljanju mrežnim priključcima 5](#_Toc206703628)

[2.2. Potrebe i zahtjevi korisnika 5](#_Toc206703629)

[2.3. Specifikacija funkcionalnosti sustava 7](#_Toc206703630)

[3. Razrada aplikacije 9](#_Toc206703631)

[3.1. Pojmovnik i definicije 9](#_Toc206703632)

[3.2. Logička struktura mreže 10](#_Toc206703633)

[3.3. Planirano ponašanje aplikacije 11](#_Toc206703634)

[4. Modeliranje baze podataka 12](#_Toc206703635)

[4.1. Entiteti i relacije 12](#_Toc206703636)

[4.2. Prikaz i objašnjenje ER modela 12](#_Toc206703637)

[4.3. Struktura ključnih tablica 14](#_Toc206703638)

[4.4. Integracija baze s aplikacijom (JPA/Hibernate) 16](#_Toc206703639)

[5. Backend implementacija 22](#_Toc206703640)

[5.1. Struktura backenda 22](#_Toc206703641)

[5.2. Pregled REST arhitekture 24](#_Toc206703642)

[5.3. Implementacija slojeva (kontroleri, servisi, repozitoriji) 26](#_Toc206703643)

[5.4. Maven alat za upravljanje projektom 31](#_Toc206703644)

[6. Frontend rješenje i korisničko sučelje 34](#_Toc206703645)

[6.1. Struktura frontenda 34](#_Toc206703646)

[6.2. HTML struktura i interaktivni elementi 35](#_Toc206703647)

[6.3. JavaScript logika i povezivanje s REST API-jem 37](#_Toc206703648)

[6.4. Dinamičko dohvaćanje podataka (kaskadni dropdowni) 38](#_Toc206703649)

[6.5. Uporaba Axios biblioteke 39](#_Toc206703650)

[7. Git i Github 41](#_Toc206703651)

[7.1. Praktična primjena Gita i GitHuba 41](#_Toc206703652)

[8. Prikaz ključnih funkcionalnosti 44](#_Toc206703653)

[8.1. Pregled i unos switch uređaja 44](#_Toc206703654)

[8.2. Upravljanje priključcima i VLAN-ovima 47](#_Toc206703655)

[8.3. Pretraživanje po imenu korisnika i utičnici 47](#_Toc206703656)

[9. Testiranje 49](#_Toc206703657)

[9.1. Metodologija testiranja 49](#_Toc206703658)

[9.2. Testiranje backend API-ja 49](#_Toc206703659)

[9.3. Testiranje korisničkog sučelja 53](#_Toc206703660)

[10. Zaključak 57](#_Toc206703661)

[11. Popis literature 59](#_Toc206703662)

[12. Prilozi 61](#_Toc206703663)

[12.1. Popis slika 61](#_Toc206703664)

[12.2. Popis tablica 62](#_Toc206703665)

[12.3. Popis kȏdova 62](#_Toc206703666)

# Uvod

Razvoj i održavanje mrežne infrastrukture ključni su zadaci u većini organizacija, posebno onih koje ovise o stabilnoj i sigurnoj IT podršci. Evidencija mrežnih priključaka, uređaja i njihovih veza često se vodi ručno, što dovodi do nepreciznosti i povećava mogućnost grešaka. Upravo iz tih razloga, razvijena je aplikacija koja omogućuje centralizirano upravljanje mrežnim uređajima i olakšava svakodnevne zadatke administratorima mreže.

U suvremenom poslovnom okruženju pouzdana mrežna infrastruktura predstavlja osnovu za gotovo sve poslovne procese – od interne komunikacije i dijeljenja resursa, do pristupa internet servisima i korištenja specijaliziranih poslovnih aplikacija. Svaki prekid u radu mreže može rezultirati financijskim gubicima, padom produktivnosti ili otežanom komunikacijom zaposlenika, stoga je kvalitetno vođenje evidencije od presudne važnosti.

Tradicionalni načini bilježenja podataka o mrežnim priključcima – poput Excel tablica ili papirnatih shema – pokazuju se kao lošiji odabir jer otežavaju pretraživanje, nisu stabilni i skloni su ljudskim pogreškama. Kako mreže postaju sve kompleksnije, raste i potreba za digitalizacijom procesa, automatizacijom zadataka te boljim alatima za praćenje i održavanje.

Ovaj rad bavi se razvojem web-aplikacije koja pruža jednostavan, ali učinkovit način za upravljanje mrežnim preklopnicima i njihovim priključcima. Kroz implementaciju modernih tehnologija u backendu i frontendu, aplikacija omogućuje unos, pregled, pretraživanje i ažuriranje podataka na intuitivan način, a istovremeno osigurava skalabilnost i mogućnost daljnjeg razvoja. Poseban naglasak stavljen je na praktične funkcionalnosti kao što su pretraga po utičnici ili korisniku, dinamički prikazi podataka te jasno strukturirana baza, što u konačnici značajno olakšava rad administratora mreže.

## Predmet i cilj rada

U središtu ovoga rada nalazi se web-aplikacija koje će se koristiti za praćenje, evidenciju i upravljanje portovima, odnosno mrežnim priključcima na mrežnim preklopnicima (switchevima). Nastala je iz stvarne potrebe kao produkt razgovora s korisnicima koji rade na održavanju mrežne infrastrukture te evidenciju vode ručno, što otežava posao i utječe na kvalitetu obavljenoga posla.

Cilj rada bio je izraditi funkcionalno softversko rješenje koje bi omogućilo unos, pregled i ažuriranje podataka o mrežnim preklopnicima i portovima, povezati portove sa sobama i korisnicima, omogućiti brzi pronalazak informacije prema nazivu utičnice ili korisnika te, konačno, pojednostaviti administraciju mrežne opreme. Ovakav sustav trebao bi ubrzati proces rada, smanjiti greške pri vođenju evidenciju i omogućiti korisnicima da imaju brži i jednostavniji pristup i upravljanje informacijama koje se tiču održavanja mreže u njihovom okruženju.

## Izvori podataka

Podaci korišteni unutar aplikacije spremljeni su u bazu podataka, a kreirani su prema stvarnim podacima zapisanima u papirnatom obliku i/ili Excel dokumentu koje korisnici koriste pri obavljanju posla. U samoj izradi aplikacije koristili su se testni podaci koji nisu stvarni podaci u vlasništvu tvrtke, već simulacija istih.

Podaci su na početku u bazu zapisani ručno, no nakon kreiranja korisničkog sučelja uneseni su, uređivani i/ili brisani koristeći samo sučelje. To je, ujedno, bio i jedan od načina testiranja funkcionalnosti, detaljnije opisan kasnije u radu.

## Korištene tehnologije i razvojno okruženje

Razvoj je započet tako što je backend dio projekta kreiran na stranici Spring Initializr i koristi moderne tehnologije i alate koji će omogućiti jednostavniju nadogradnju sustava ako u budućnosti bude potrebe za istom. Spring Initializr služi za brzo generiranje Spring Boot projekta s osnovnim ovisnostima [1]. Pojednostavljuje konfiguraciju te stvara kostur projekta [1]. Nakon konfiguracije osnovnih postavki u backendu, konfiguriran je i frontend dio.

Backend, odnosno poslužiteljski dio sustava, razvijen je koristeći sljedeće tehnologije:

* Java 21 – programski jezik za implementaciju logike sustava
* Spring Boot 3.4.4. – okvir za razvoj REST API-ja
* Spring Data JPA – sloj za pristup bazi podataka

Frontend razvijen je koristeći:

* HTML, C i Boostrap – oblikovanje korisničkog sučelja
* JavaScript – za logiku i upravljanje stanjima
* React (JavaScript biblioteka) – za izradu modernog korisničkog sučelja
* Axios – za komunikaciju s backendom koristeći HTTP zahtjeve

Alati i razvojno okruženje:

* IntelliJ IDEA – za razvoj backend dijela aplikacije
* Visual Studio Code – za uređenje i razvoj frontend dijela
* Postman – za testiranje REST API-ja
* pgAdmin i dBeaver – za upravljanje PostgreSQL bazom podataka
* Git i Github – za verzioniranje koda

## Sadržaj i struktura rada

Struktura rada osmišljena je tako da prati slijed razvoja aplikacije – od analize problema, preko implementacije, do testiranja. U uvodu su definirani predmet i cilj rada, te izvori i tehnologije korištene u razvoju. Drugo poglavlje donosi pregled problema i funkcionalnih zahtjeva korisnika koji su poslužili kao smjernice za projektiranje aplikacije. Treće poglavlje pojašnjava pojmove i logiku mrežne domene, dok četvrto prikazuje bazu podataka, njen model, tablice i način povezivanja s aplikacijom putem JPA/Hibernate tehnologije.

Peto poglavlje posvećeno je implementaciji backend dijela aplikacije – uključujući REST arhitekturu, slojeve i način komunikacije s bazom. U šestom poglavlju opisano je frontend rješenje razvijeno u Reactu, s naglaskom na povezivanje s API-jem i prikaz podataka kroz dinamičke komponente. Sedmo poglavlje sadrži osnovno o Gitu i Githubu te praktičnu primjenu istoga u radu. Osmo poglavlje donosi prikaz ključnih funkcionalnosti kao što su unos uređaja, upravljanje portovima i pretraga. U devetom poglavlju opisana je metodologija testiranja i alati korišteni za provjeru funkcionalnosti. Na kraju, u zaključku su istaknute prednosti ovakvog sustava i mogući pravci daljnjeg razvoja.

# Opis problema i funkcionalni zahtjevi

Ručni, nedigitalizirani način evidencije i upravljanja mrežnom infrastrukturom u današnje je digitalno doba gotovo neprihvatljiv, čak i u situacijama kada se kombinira s papirnatom dokumentacijom. Ovakav pristup može dovesti do pogrešaka, gubljenja informacija, neusklađenosti podataka, nedostatka strukture i slično. Upravo zbog takvih, ali i drugih sličnih problema, razvijen je sustav o kojemu je riječ u ovome radu. Cilj je centralizirati upravljanje mrežnom infrastrukturom pružajući korisniku jednostavno, no funkcionalno te uvijek dostupno korisničko sučelje.

## Izazovi u upravljanju mrežnim priključcima

Tijekom upravljanja mrežnom infrastrukturom javljaju se brojni izazovi, među kojima su:

* Nepreglednost postojećih zapisa – podaci o portovima, lokacijama i povezanim uređajima često su nepovezani i teško dostupni.
* Ručno vođenje evidencije – povećava mogućnost grešaka i otežava ažuriranje podataka.
* Teško pronalaženje lokacije priključka – kada dođe do problema u mreži, često je teško utvrditi gdje se točno nalazi odgovarajući port.
* Nedostatak integracije s drugim informacijama – npr. koji je korisnik povezan s određenim portom, na kojem katu se nalazi uređaj, koji je VLAN itd.

Aplikacija razvijena u sklopu ovog rada rješava navedene izazove pružanjem strukturiranog i digitaliziranog upravljanja mrežnim priključcima.

## Potrebe i zahtjevi korisnika

Iz izazova opisanih u prethodnom poglavlju proizašao je popis zahtjeva korisnika, koji je vidljiv u tablici u nastavku rada. Popis je sastavljen tijekom višestrukih razgovora s krajnjim korisnicima, kako bi se dobio što jasniji dojam o tome što sustav treba sadržavati.

Tablica 1 Popis zahtjeva korisnika

|  |  |  |
| --- | --- | --- |
| **Rbr.** | **Zahtjev korisnika** | **Opis** |
| 1 | Pregled svih switch uređaja | Korisnik može vidjeti popis svih unesenih switcheva s pripadajućim informacijama. |
| 2 | Dodavanje novog switch uređaja | Korisnik može unijeti novi switch putem obrasca, s podacima kao što su model, IP adresa itd. |
| 3 | Uređivanje i brisanje postojećeg switcha | Korisnik može ažurirati ili ukloniti postojeće podatke o switchu. |
| 4 | Upravljanje portovima | Za svaki switch korisnik može unijeti, urediti ili obrisati podatke o portovima. |
| 5 | Dodjela VLAN-a portovima | Korisnik može definirati VLAN za svaki port. |
| 6 | Povezivanje portova sa sobama, katovima i lokacijama | Portovi su logički povezani s fizičkom lokacijom. |
| 7 | Evidencija korisnika ili uređaja spojenih na port | Korisnik može unijeti ime osobe ili naziv uređaja spojenog na port. |
| 8 | Pretraga po utičnici | Korisnik može unijeti oznaku utičnice i pronaći pripadajući switch i port. |
| 9 | Pretraga po imenu korisnika | Korisnik može pronaći na kojem je portu spojen određeni korisnik. |
| 10 | Upravljanje VLAN popisom | Korisnik može unositi, mijenjati i brisati VLAN-ove. |

Na temelju potreba i zahtjeva korisnika utvrđene su funkcionalnosti koje će web-aplikacija imati, a koje su detaljnije opisane u sljedećem potpoglavlju.

## Specifikacija funkcionalnosti sustava

Na temelju korisničkih zahtjeva definirane su konkretne funkcionalnosti koje web-aplikacija mora omogućiti. Funkcionalnosti su grupirane prema logičkim cjelinama koje odgovaraju stvarnim zadacima korisnika tijekom upravljanja mrežnom opremom i evidencijom portova.

Aplikacija omogućuje:

* Upravljanje mrežnim uređajima (switchevima):
  + Pregled svih unesenih switch uređaja s prikazom pripadajućih podataka (model, IP adresa, lokacija, tip i vrsta veze).
  + Dodavanje, izmjena i brisanje switch uređaja putem obrasca na korisničkom sučelju.
* Upravljanje portovima:
  + Unos i pregled portova pojedinog switcha.
  + Svakom portu moguće je dodijeliti VLAN, broj utičnice, broj sobe i podatak o povezanoj osobi ili uređaju.
  + Omogućena je izmjena i brisanje postojećih zapisa.
* Pretraga i filtriranje:
  + Pretraga po nazivu utičnice kako bi se brzo pronašao switch i port na koji je spojen uređaj.
  + Pretraga prema imenu korisnika za lakše lociranje mrežne veze određenog zaposlenika.
* Upravljanje fizičkom infrastrukturom:
  + Evidencija lokacija, katova i soba u objektima.
  + Povezivanje svakog port priključka sa stvarnim fizičkim položajem u prostoru.
* VLAN administracija:
  + Upravljanje popisom VLAN-ova.
  + Dodjela VLAN-a svakom portu prema potrebi mrežne konfiguracije.

Sve funkcionalnosti dostupne su putem preglednog web sučelja razvijenog u Reactu. Korištenjem REST API-ja ostvarena je komunikacija između frontend i backend dijela sustava.

# Razrada aplikacije

Kako bi se uspješno razvila i implementirala aplikacija za upravljanje mrežnim priključcima, nužno je razumjeti temeljne pojmove i odnose između elemenata mreže. U ovom poglavlju definiraju se ključni izrazi, opisuje se logička struktura mreže na kojoj se temelji sustav te se pojašnjava planirano ponašanje aplikacije s obzirom na poslovna pravila i korisničke potrebe.

## Pojmovnik i definicije

Ovo potpoglavlje definira ključne pojmove koji se koriste unutar aplikacije i sustava nabrojane i opisane u tablici u nastavku.

Tablica 2 Osnovni pojmovi i definicije

|  |  |
| --- | --- |
| **Pojam** | **Definicija** |
| **Switch** | Uređaj koji analizira mrežni promet i usmjerava podatke izravno s jednog priključka na drugi. Zahvaljujući prepoznavanju MAC adresa, preklopnik zna na kojem se priključku nalazi koje računalo i omogućuje istovremeni prijenos podataka većem broju korisnika bez smanjenja performansi mreže [2, str. 24]. |
| **Port** | Port na mrežnom switchu je fizička ili logička točka povezivanja koja omogućuje prijenos podataka između uređaja i mreže, pri čemu može služiti za komunikaciju unutar jedne VLAN mreže (access port), između više VLAN-ova (trunk port) ili kao kombinacija oba (hybrid port) [3]. |
| **VLAN** | VLAN omogućuje grupiranje računala unutar mreže prema logičkim kriterijima, neovisno o njihovoj stvarnoj fizičkoj lokaciji. Računala unutar iste virtualne mreže mogu se nalaziti na različitim lokacijama, ali zahvaljujući konfiguraciji, međusobno komuniciraju kao da su spojena u istu mrežnu cjelinu [4]. |
| **Lokacija** | U kontekstu ovoga rada – zgrada u kojoj se nalazi mrežna oprema. |
| **Kat (floor)** | Etaža u zgradi gdje se nalazi mrežna oprema. |
| Soba (room) | Prostorija u kojoj se koristi mrežni priključak i u kojoj se nalaze korisnici ili uređaji. |
| Korisnik | Osoba koja koristi mrežni priključak, odnosno zaposlenik ustanove. |

## Logička struktura mreže

Logička struktura mreže osmišljena je kako bi osigurala jasnoću, preglednost i jednostavno upravljanje svim mrežnim priključcima u organizaciji. Struktura je oblikovana hijerarhijski, uz međusobne veze između mrežnih uređaja, fizičkih lokacija i krajnjih korisnika.

Svaki switch unutar sustava pridružen je lokaciji, zatim određenom katu, a na kraju i konkretnoj sobi u kojoj se fizički nalazi. Ovakva organizacija omogućuje pregled mrežne opreme po zgradama i prostorijama. Unutar svakog switcha nalaze se portovi koji su jedinstveni za taj switch i mogu biti povezani s određenom sobom u zgradi. Na taj način jasno je gdje svaki port fizički vodi.

Na svaki port može biti povezan određeni korisnik ili uređaj, čime se prati tko koristi koji priključak u mreži. Ova razina evidencije omogućuje precizno praćenje, jednostavno dijagnosticiranje problema i lakše održavanje.

Sustav je osmišljen tako da omogućuje skalabilnost i buduće nadogradnje, primjerice dodavanje novih lokacija ili switcheva bez narušavanja postojeće strukture.

## Planirano ponašanje aplikacije

Aplikacija se sastoji od dva osnovna dijela:

* Backend – razvijen u Spring Bootu, pruža REST API za upravljanje entitetima kao što su switch, port, lokacija, kat, soba, VLAN itd.
* Frontend – temeljen na HTML-u, CSS-u i JavaScriptu, omogućuje korisniku unos, pregled i izmjenu podataka putem preglednika.

Planirane funkcionalnosti uključuju dohvaćanje svih entiteta (GET), unos novih zapisa (POST), uređivanje postojećih zapisa (PUT), brisanje zapisa (DELETE) i filtriranje podataka na temelju korisničkog unosa (pretraga po imenu korisnika, oznaci utičnice...).

Uz to, korisnik kroz sučelje upravlja vezama između portova i fizičkih lokacija (sobe, katovi), što omogućuje detaljan nadzor nad fizičkom i logičkom strukturom mreže.

# Modeliranje baze podataka

U svrhu implementacije web-aplikacije izrađen je relacijski model baze podataka koji obuhvaća sve ključne entitete vezane uz evidenciju mrežnih priključaka, njihovu fizičku lokaciju te VLAN-ove. Baza je osmišljena tako da omogućava jednostavno proširivanje, učinkovitu pohranu i brz pristup podacima.

## Entiteti i relacije

Aplikacija koristi sljedeće glavne entitete:

* Switches – predstavljaju mrežne uređaje koji se fizički nalaze u određenim lokacijama, katovima i sobama.
* Ports – fizički priključci na switchu, povezani s VLAN-ovima, sobama i korisnicima.
* Rooms, Floors, Locations – hijerarhijska organizacija fizičkog prostora.
* Vlans – omogućuju segmentaciju mreže.
* Users – korisnici sustava s podacima o pristupu (admin, odnosno trenutno jedna vrsta korisnika).

Relacije među entitetima su definirane tako da je svaki switch je povezan s točnom lokacijom, katom i sobom. Svaki port pripada točno jednom switchu, a port može biti povezan s room i opcionalno s korisnikom ili uređajem. Floor pripada točno jednoj lokaciji, a room točno jednom flooru.

## Prikaz i objašnjenje ER modela

Dijagram vidljiv na slici 1 prikazuje vezu između entiteta pomoću stranih ključeva. Na primjer: room\_id, floor\_id, i location\_id u tablici switches povezuju mrežni uređaj s fizičkom lokacijom. Port\_number i vlan\_id u tablici ports određuju detalje o mrežnom priključku. Tablica users sadrži atribute poput username, password\_hash, email, is\_admin, i koristi se za autentikaciju i autorizaciju korisnika sustava. Port\_statistics je takozvani *view* koji bi služio dohvaćanju i prikazu statističkih podataka o portovima te se trenutno ne koristi jer korisnik u ovom trenutku ne treba tu funkcionalnost, no implementiran je zbog potencijalne implementacije u bližoj budućnosti.

A screenshot of a computer

AI-generated content may be incorrect.

Slika 1 ER dijagram baze podataka

## Struktura ključnih tablica

Tablice u bazi podataka napravljene su tako da što jednostavnije i jasnije prikazuju podatke koji se koriste u aplikaciji. Svaki dio sustava – poput switch uređaja, portova, lokacija, soba ili VLAN-ova – ima svoju tablicu. Time se omogućuje da se podaci lakše upisuju, mijenjaju i pregledavaju.

Većina tablica je povezana međusobno pomoću vanjskih ključeva. Na primjer, svaki port pripada samo jednom switchu, a svaki switch pripada nekoj lokaciji, katu i sobi. Te veze između tablica osiguravaju da svi podaci budu povezani i da se ne može dogoditi, primjerice, da se unese port bez da postoji odgovarajući switch.

Svaka tablica ima svoj glavni (primarni) ključ koji je jedinstven broj (tipa int). Tekstualni podaci – poput naziva, oznaka i opisa – pohranjuju se kao varchar. Sve tablice su povezane s Java klasama (entitetima) putem JPA (Java Persistence API) koji omogućuje jednostavan rad s bazom iz same aplikacije.

Ovakva struktura baze olakšava korištenje aplikacije, omogućuje njezino nadograđivanje te čini podatke preglednima i pouzdanima.

Tablica 3 Struktura tablice „switches”

|  |  |  |
| --- | --- | --- |
| **Naziv stupca** | **Tip podatka** | **Opis** |
| id | serial4 | Primarni ključ – identifikator switch uređaja |
| name | varchar | Naziv switcha |
| model | varchar | Model uređaja |
| ip\_address | varchar | IP adresa switcha |
| type | varchar | Tip uređaja |
| connection | varchar | Vrsta povezivosti |
| location\_id | int (FK → locations) | Lokacija gdje se uređaj nalazi |
| room\_id | int (FK → rooms) | Soba u kojoj je uređaj instaliran |
| floor\_id | int(FK → floors) | Kat na lokaciji |
| inventory\_number | varchar | Inventarski broj (opcionalno) |
| broj\_portova | int | Ukupan broj portova (opcionalno) |
| active | Bool | Oznaka aktivnosti korisnika |

Tablica 4 Struktura tablice „ports”

|  |  |  |
| --- | --- | --- |
| **Naziv stupca** | **Tip podataka** | **Opis** |
| id | int | Primarni ključ |
| port\_number | varchar | Broj porta |
| label | varchar | Naziv utičnice |
| vlan\_id | Int | Broj VLAN-a dodijeljenog portu |
| room\_name | varchar | Oznaka sobe (tekstualna) |
| connected | boolean | Je li port spojen |
| user\_full\_name | varchar | Ime korisnika ili uređaja |
| switch\_id | int (FK → switches) | Veza na switch |

Tablica 5 Struktura tablice „vlans” za opis definiranih VLAN-ova

|  |  |  |
| --- | --- | --- |
| **Naziv stupca** | **Tip podataka** | **Opis** |
| id | int | Primarni ključ |
| vlan\_number | Int | Broj VLAN-a |
| ip\_range | varchar | Raspon IP-ja |
| description | varchar | Opis |

Tablica 5 Struktura tablice „locations”

|  |  |  |
| --- | --- | --- |
| **Naziv stupca** | **Tip podataka** | **Opis** |
| id | int | Primarni ključ |
| name | varchar | Naziv lokacije |
| description | varchar | Opis |
| active | bool | Oznaka aktivnosti |

## Integracija baze s aplikacijom (JPA/Hibernate)

Za povezivanje baze podataka s aplikacijom korišten je JPA (Java Persistence API) u kombinaciji s Hibernateom. Hibernate i JPA olakšavaju razvoj jer smanjuju količinu ručnog pisanja SQL upita. Entity Manager je glavni mehanizam JPA-e koji upravlja životnim ciklusom entiteta i pruža metode za spremanje, dohvaćanje, ažuriranje i brisanje podataka [5]. Uz navedene prednosti JPA-ja, dodatno pomaže Hibernate jer podržava različite baze podataka, omogućava sinkronizaciju izmjena u bazi podataka jednom kada razvojni tim napravi promjene u backendu i slično [6]. Ovaj pristup omogućava da se nad bazom podataka radi na objektno orijentirani način, što znači da se redovi u tablicama predstavljaju kao Java objekti.

Svaka tablica u bazi ima svoju odgovarajuću JPA entitetsku klasu, koja je označena s anotacijom @Entity. Atributi tih klasa povezani su s kolonama u tablici pomoću anotacija poput @Column, @Id, @GeneratedValue, @ManyToOne, @OneToMany i slično. Time se jasno definira struktura podataka i odnosi među entitetima.

Na primjer, klasa Switches sadrži atribute poput name, model, ipAddress, kao i reference na druge entitete (Location, Room i sl.). Na taj način se iz aplikacije direktno može pristupiti podacima iz povezanih tablica, bez potrebe za ručnim pisanjem SQL upita.

Za pristup podacima koriste se Spring Data JPA repozitoriji, koji omogućuju izvođenje osnovnih operacija nad bazom (poput spremanja, dohvaćanja, ažuriranja i brisanja) bez dodatnog pisanja koda [6]. Sve operacije nad podacima obavljaju se kroz repozitorije, što čini kod čišćim i lakšim za održavanje. Povezanost aplikacije i baze omogućena je kroz konfiguraciju u application.properties datoteci, gdje su definirani podaci o konekciji na bazu (URL, korisničko ime, lozinka, i sl.).

Primjer primjene ovih tehnologija vidljiv je u kodu na primjeru atributa s*witches* u nastavku. U navedenim primjerima JPA omogućuje mapiranje Java klasa na tablice u bazi podataka koristeći anotacije poput @Entity i @Table, dok Hibernate kao implementacija JPA-a izvršava te operacije u pozadini (poput spremanja, dohvaćanja i ažuriranja podataka). Repository sloj koristi JPA metode (save, findAll, itd.) koje Hibernate automatski prevodi u SQL upite prema bazi. U određenim slučajevima zbog samih funkcionalnosti aplikacije, ponekad su pisani ručniupiti koristeći @Query anotaciju. U servisnoj i kontrolerskoj klasi se primjena Hibernate i JPA-ja vidi indirektno – konstruktorskim *injectionom* repozitorijske klase.

* Entitet – Switches

**@**Entity

**@**Table(name="switches")

**@**SQLDelete(sql = "UPDATE switches SET active = false WHERE id = ?")

**@**Where(clause = "active = true")

public class Switches {

**@**Id

**@**GeneratedValue(strategy= GenerationType.IDENTITY)

**@**Column(name="id")

private Long id;

**@**Column(nullable = false)

private boolean active = true;

**@**Column(name="model")

private String model;

**@**Column(name="ip\_address")

private String ipAddress;

**@**Column(name="name")

private String name;

**@**Column(name="type")

private String type;

**@**Column(name="connection")

private String connection;

**@**Column(name = "broj\_portova")

private Integer brojPortova;

//Getteri i setteri

}

Kȏd 1 Primjer entity klase

* Repository klasa – SwitchRepository

package com.networkapp.demo.repository;

import com.networkapp.demo.entity.Switches;

import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface SwitchRepository extends JpaRepository**<**Switches, Long**>** {

List**<**Switches**>** findByActiveTrueOrderByIdAsc();

}

Kȏd 2 Primjer repository klase

* Service klasa – SwitchService

package com.networkapp.demo.service;

//importovi

**@**Service

public class SwitchService {

private final SwitchRepository switchRepository;

private final LocationRepository locationRepository;

private final RoomRepository roomRepository;

private final PortRepository portRepository;

public SwitchService(SwitchRepository switchRepository,

LocationRepository locationRepository,

RoomRepository roomRepository,

PortRepository portRepository) {

this.switchRepository = switchRepository;

this.locationRepository = locationRepository;

this.roomRepository = roomRepository;

this.portRepository = portRepository;

}

/**\*\* Vraća SAMO aktivne switch-eve, sortirane po ID-u rastuće. \*/**

**public List<Switches> getAllSwitches() {**

return switchRepository.findByActiveTrueOrderByIdAsc();

}

/**\*\***

\* Dohvati jedan switch PO ID-u bez filtriranja po active.

\*/

public Optional**<**Switches**>** getSwitchById(Long id) {

return switchRepository.findById(id); // ⬅️ promjena: bez .filter(...active...)

}

/**\*\***

\* Kreiraj/uredi switch:

\* - poveži lokaciju/sobu ako su poslani

\* - osiguraj active=true za nove

\* - portove automatski kreiraj samo kod novog unosa (ne i kod uređenja)

\*/

**@**Transactional

public Switches saveSwitch(Switches sw) {

// Poveži lokaciju ako je poslana (ID referenca)

if (sw.getLocationId() != null && sw.getLocationId().getId() != null) {

locationRepository.findById(sw.getLocationId().getId())

.ifPresent(sw::setLocationId);

}

// Poveži/kreiraj sobu:

if (sw.getRoomId() != null) {

if (sw.getRoomId().getId() != null) {

roomRepository.findById(sw.getRoomId().getId())

.ifPresent(sw::setRoomId);

} else {

String brojSobe = sw.getRoomId().getNumber();

if (brojSobe != null && !brojSobe.trim().isEmpty()) {

Rooms soba = roomRepository.findByNumber(brojSobe)

.orElseGet(() **->** {

Rooms novaSoba = new Rooms();

novaSoba.setNumber(brojSobe);

return roomRepository.save(novaSoba);

});

sw.setRoomId(soba);

} else {

sw.setRoomId(null);

}

}

}

// Novi zapisi neka budu aktivni

if (sw.getActive() == null) {

sw.setActive(true);

}

// Je li ovo novi zapis? (id == null prije spremanja)

boolean isNew = (sw.getId() == null);

// Spremi switch

Switches savedSwitch = switchRepository.save(sw);

// Portove kreiramo SAMO kod novog switcha (da izbjegnemo duplikate pri uređenju)

if (isNew && sw.getBrojPortova() != null && sw.getBrojPortova() **>** 0) {

for (int i = 1; i **<**= sw.getBrojPortova(); i++) {

Ports port = new Ports();

port.setSwitchId(savedSwitch);

port.setPortNumber(String.valueOf(i));

port.setConnected(false);

port.setActive(true);

portRepository.save(port);

}

}

return savedSwitch;

}

/**\*\***

\* "Brisanje" = SOFT DELETE:

\* - portovi tog switcha -> active=false

\* - switch -> active=false (preko @SQLDelete ili ručno)

\*/

**@**Transactional

public void deleteSwitch(Long id) {

Optional**<**Switches**>** opt = switchRepository.findById(id);

if (opt.isEmpty()) return;

Switches sw = opt.get();

if (Boolean.FALSE.equals(sw.getActive())) return; // već neaktivan

// 1) Portovi **->** active=false

portRepository.softDeleteBySwitchId(id);

// 2) Switch **->** active=false

switchRepository.deleteById(id); // **@**SQLDelete pretvara u UPDATE active=false

}

}

Kȏd 3 Primjer service klase

* Controller klasa – SwitchController

package com.networkapp.demo.controller;

import com.networkapp.demo.entity.Switches;

import com.networkapp.demo.service.SwitchService;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

**@**RestController

**@**RequestMapping("/api/switches")

**@**CrossOrigin(origins = "\*")

public class SwitchController {

private final SwitchService service;

public SwitchController(SwitchService service) {

this.service = service;

}

**@**GetMapping

public List**<**Switches**>** getAll() {

return service.getAllSwitches(); // vraća samo aktivne

}

**@**GetMapping("/{id}")

public ResponseEntity**<**Switches**>** getById(**@**PathVariable Long id) {

return service.getSwitchById(id)

.map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());

}

**@**PostMapping

public Switches create(**@**RequestBody Switches sw) {

return service.saveSwitch(sw);

}

**@**PutMapping("/{id}")

public ResponseEntity**<**Switches**>** update(**@**PathVariable Long id, **@**RequestBody Switches sw) {

return service.getSwitchById(id).map(existing **->** {

sw.setId(id);

return ResponseEntity.ok(service.saveSwitch(sw));

}).orElse(ResponseEntity.notFound().build());

}

**@**DeleteMapping("/{id}")

public ResponseEntity**<**Void**>** delete(**@**PathVariable Long id) {

service.deleteSwitch(id);

return ResponseEntity.noContent().build();

}

}

Kȏd 4 Primjer controller klase

Navedene klase, kao i cjelokupna struktura i implementacija backenda i frontenda detaljnije je obuhvaćena sljedećim poglavljem.

# Backend implementacija

Backend dio aplikacije izrađen je korištenjem Java Spring Boot okvira, koji omogućuje brzu i strukturiranu izradu REST API-ja. Podaci se pohranjuju i dohvaćaju iz PostgreSQL baze podataka uz pomoć JPA i Hibernate mehanizama za mapiranje objekata. Arhitektura backenda podijeljena je u slojeve: kontrolerski sloj za obradu zahtjeva, servisni sloj za poslovnu logiku te repozitorijski sloj za pristup podacima. Ovakva podjela, detaljnije opisana u sljedećim potpoglavljima, osigurava čitljivost, ponovnu iskoristivost i lakše održavanje sustava.

## Struktura backenda

Projekt je razvijen u okviru Spring Boot okvira koristeći standardnu Maven strukturu direktorija. Na slici se vidi hijerarhija direktorija i datoteka unutar projekta demo.

A screenshot of a computer

AI-generated content may be incorrect.

Slika 2 Backend struktura

Cilj je ovakve podjele odvojiti logiku poslovanja, upravljanje zahtjevima i pristup podacima radi jednostavnije čitljivost koda, ali i lakšeg održavanja te budućeg proširivanja aplikacije. Jedan je od većih izazova pri preuzimanju tuđeg projekta upravo razumijevanje istoga, stoga je naglasak bio i na tome da projekt bude strukturiran smisleno te da je osobi koja će raditi na eventualnom daljnjem razvoju bude većina toga jasna upravo iz čitanja samoga koda.

Struktura vidljiva na slici detaljnije je objašnjena u nastavku:

* src/main/java/com.networkapp.demo – osnovni paket aplikacije koji sadrži sve glavne programske module, podijeljene po slojevima:
  + controller – sadrži REST kontrolere koji upravljaju HTTP zahtjevima (npr. SwitchController, RoomController, LocationController itd.),
  + service – obuhvaća servisne klase koje sadrže poslovnu logiku (npr. SwitchService, LocationService),
  + repository – sadrži JPA repozitorije koji omogućuju komunikaciju s bazom podataka (npr. SwitchRepository, RoomRepository),
  + entity – definira JPA entitete koji se mapiraju na tablice baze podataka (npr. Switches, Ports, Locations),
  + config – služiti za definiranje sigurnosnih i CORS konfiguracija,
  + DemoApplication.java – glavna klasa koja sadrži main metodu i služi kao ulazna točka za pokretanje cijele aplikacije,
  + src/main/resources – direktorij koji sadrži resurse koji se ne prevode:
    - application.properties – konfiguracijska datoteka u kojoj se definiraju podaci za povezivanje s bazom (JDBC URL, korisničko ime, lozinka), kao i Hibernate parametri (ddl-auto, show-sql).
    - src/test/java – predviđen za smještaj JUnit testova
    - target – automatski generiran direktorij nakon buildanja projekta, sadrži kompajlirane .class datoteke i ostale artefakte.
    - pom.xml – Mavenova konfiguracijska datoteka koja sadrži ovisnosti, verzije Spring Boota, pluginove i postavke buildanja projekta.

## Pregled REST arhitekture

REST (Representational State Transfer), prema definiciji CERT-a [7], odnosi se na arhitekturni stil koji je najčešće zastupljen pri samoj izradi web-aplikacija. REST se oslanja na HTTP aplikacijski protokol koji se standardno koristi prilikom pregledavanja weba. Zapravo, cijeli web prostor na internetu koristi REST arhitekturu [7].

U ovome radu korištene su sljedeće HTTP metode:

* GET – za dohvaćanje resursa
* POST – za kreiranje novih resursa
* PUT – za ažuriranje postojećih resursa
* DELETE – služi za brisanje postojećih resursa, iako je u radu radi očuvanja podataka uglavnom korištena soft delete opcija.

Na konkretnom primjeru, GET /api/switches dohvaća listu svih switcheva u bazi podataka, POST api/switches omogućuje unos novih switcheva u bazu podataka, PUT /api/switches/{id} ažuriranje switcheva, a DELETE /api/switches/{id} brisanje istih.

Podaci se iz baze dohvaćaju ili spremaju putem servisnog sloja i repozitorija, dok se klijentu šalju u JSON formatu. To omogućuje frontend aplikaciji (HTML + JavaScript) da dinamički prikazuje podatke ili ih šalje natrag poslužitelju.

Primjer JSON formata bio bi:

{

"id": 1,

"name": "Switch A1",

"model": "Cisco Catalyst 2960",

"ipAddress": "192.168.1.1/32",

"type": "access",

"connection": "ethernet",

"inventoryNumber": "INV001",

"locationId": {

"id": 1,

"name": "Zgrada A"

},

"floorId": {

"id": 1,

"number": "1"

},

"roomId": {

"id": 1,

"number": "101"

}"

}

Kȏd 5 Primjer podataka u JSON formatu

U sustavu za praćenje i evidenciju mrežnih priključaka REST API-ji su implementirani u kontrolerskim klasama, poput sljedeće:

package com.networkapp.demo.controller;

import com.networkapp.demo.entity.Ports;

import com.networkapp.demo.repository.PortRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.Collections;

import java.util.List;

**@**RestController

**@**RequestMapping("/api/ports")

**@**CrossOrigin(origins = "http://localhost:3000")

public class PortController {

private final PortRepository portRepository;

**@**Autowired

public PortController(PortRepository portRepository) {

this.portRepository = portRepository;

}

**@**GetMapping("/search")

public List**<**Ports**>** searchPorts(**@**RequestParam(required = false) String label,

**@**RequestParam(required = false) String user) {

if (label != null && !label.isBlank()) {

return portRepository.findByLabelContainingIgnoreCase(label);

} else if (user != null && !user.isBlank()) {

return portRepository.findByUserFullNameContainingIgnoreCase(user);

}

return Collections.emptyList();

}

**@**GetMapping("/switch/{switchId}")

public List**<**Ports**>** getPortsBySwitch(**@**PathVariable Long switchId) {

return portRepository.findBySwitchId\_Id(switchId);

}

**@**PutMapping("/{id}")

public Ports updatePort(**@**PathVariable Long id, **@**RequestBody Ports updatedPort) {

return portRepository.findById(id)

.map(port **->** {

port.setPortNumber(updatedPort.getPortNumber());

port.setLabel(updatedPort.getLabel());

port.setUserFullName(updatedPort.getUserFullName());

port.setConnected(updatedPort.getConnected());

port.setVlanId(updatedPort.getVlanId());

port.setRoomName(updatedPort.getRoomName()); // ✅ ispravljeno na roomName

port.setSwitchId(updatedPort.getSwitchId());

return portRepository.save(port);

})

.orElseThrow(() **->** new RuntimeException("Port s ID " **+** id **+** " nije pronađen"));

}

}

Kȏd 6 Implementacija kontrolerske klase

Osim samih kontrolera, kao što je već ranije opisano, sustav je građen i na temelju servisnih i repozitorijskih klasa, koje su detaljnije opisane u sljedećem potpoglavlju.

## Implementacija slojeva (kontroleri, servisi, repozitoriji)

U ovoj aplikaciji, kako je već navedeno, primijenjen je standardni troslojni pristup: Controller (ulazna točka HTTP zahtjeva), Service (poslovna logika i transakcije) i Repository (pristup bazi podataka). Takva podjela omogućuje bolju testabilnost, čitljivost i lakše proširenje sustava.

Kontroleri služe kao ulazni sloj: primaju zahtjeve, prosljeđuju ih servisima i vraćaju rezultate u JSON formatu. U ovom projektu kontroleri rade direktno s JPA entitetima (bez DTO‑a), što ubrzava razvoj i pojednostavljuje primjer, no u budućnosti se može uvesti DTO sloj radi boljeg odvajanja modela baze od vanjskog API‑ja. Primjer implementacije kontrolerskog sloja vidljiv je u primjeru koda 6.

Servisni sloj u Spring Boot aplikaciji služi kao most između kontrolera i repozitorija. U njemu se nalazi poslovna logika koja obrađuje zahtjeve prije nego što se podaci pošalju u bazu podataka ili vrate korisniku. Servisne klase su anotirane s @Service, čime ih Spring registrira kao beanove i omogućuje njihovo automatsko umetanje (@Autowired) u druge slojeve aplikacije [8]. U nastavku je vidljiv primjer servisne klase u cijelosti, dok je u kodu 8 prikaz metode za dohvaćanje svih switcheva, pri čemu servisna klasa zapravo injektira repozitorijsku klasu kako bi podaci bili dohvaćeni.

package com.networkapp.demo.service;

import com.networkapp.demo.entity.Switches;

import com.networkapp.demo.entity.Rooms;

import com.networkapp.demo.entity.Ports;

import com.networkapp.demo.repository.SwitchRepository;

import com.networkapp.demo.repository.LocationRepository;

import com.networkapp.demo.repository.RoomRepository;

import com.networkapp.demo.repository.PortRepository;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

import java.util.Optional;

**@**Service

public class SwitchService {

private final SwitchRepository switchRepository;

private final LocationRepository locationRepository;

private final RoomRepository roomRepository;

private final PortRepository portRepository;

public SwitchService(SwitchRepository switchRepository,

LocationRepository locationRepository,

RoomRepository roomRepository,

PortRepository portRepository) {

this.switchRepository = switchRepository;

this.locationRepository = locationRepository;

this.roomRepository = roomRepository;

this.portRepository = portRepository;

}

/**\*\* Vraća SAMO aktivne switch-eve, sortirane po ID-u rastuće. \*/**

**public List<Switches> getAllSwitches() {**

return switchRepository.findByActiveTrueOrderByIdAsc();

}

/**\*\***

\* Dohvati jedan switch PO ID-u BEZ filtriranja po active.

\* Ovo je bitno kako PUT/GET /api/switches/{id} ne bi vraćao 404

\* za neaktivne (soft-deleted) zapise.

\*/

public Optional**<**Switches**>** getSwitchById(Long id) {

return switchRepository.findById(id); // ⬅️ promjena: bez .filter(...active...)

}

/**\*\***

\* Kreiraj/uredi switch:

\* - poveži lokaciju/sobu ako su poslani

\* - osiguraj active=true za nove

\* - portove automatski kreiraj SAMO kod novog unosa (ne i kod uređenja)

\*/

**@**Transactional

public Switches saveSwitch(Switches sw) {

// Poveži lokaciju ako je poslana (ID referenca)

if (sw.getLocationId() != null && sw.getLocationId().getId() != null) {

locationRepository.findById(sw.getLocationId().getId())

.ifPresent(sw::setLocationId);

}

// Poveži/kreiraj sobu:

if (sw.getRoomId() != null) {

if (sw.getRoomId().getId() != null) {

roomRepository.findById(sw.getRoomId().getId())

.ifPresent(sw::setRoomId);

} else {

String brojSobe = sw.getRoomId().getNumber();

if (brojSobe != null && !brojSobe.trim().isEmpty()) {

Rooms soba = roomRepository.findByNumber(brojSobe)

.orElseGet(() **->** {

Rooms novaSoba = new Rooms();

novaSoba.setNumber(brojSobe);

return roomRepository.save(novaSoba);

});

sw.setRoomId(soba);

} else {

sw.setRoomId(null);

}

}

}

// Novi zapisi neka budu aktivni

if (sw.getActive() == null) {

sw.setActive(true);

}

// Je li ovo novi zapis? (id == null prije spremanja)

boolean isNew = (sw.getId() == null);

// Spremi switch

Switches savedSwitch = switchRepository.save(sw);

// Portove kreiramo SAMO kod novog switcha (da izbjegnemo duplikate pri uređenju)

if (isNew && sw.getBrojPortova() != null && sw.getBrojPortova() **>** 0) {

for (int i = 1; i **<**= sw.getBrojPortova(); i++) {

Ports port = new Ports();

port.setSwitchId(savedSwitch);

port.setPortNumber(String.valueOf(i));

port.setConnected(false);

port.setActive(true);

portRepository.save(port);

}

}

return savedSwitch;

}

/**\*\***

\* "Brisanje" = SOFT DELETE:

\* - portovi tog switcha -> active=false (bulk)

\* - switch -> active=false (preko @SQLDelete ili ručno)

\*/

**@**Transactional

public void deleteSwitch(Long id) {

Optional**<**Switches**>** opt = switchRepository.findById(id);

if (opt.isEmpty()) return;

Switches sw = opt.get();

if (Boolean.FALSE.equals(sw.getActive())) return; // već neaktivan

// 1) Portovi **->** active=false

portRepository.softDeleteBySwitchId(id);

// 2) Switch **->** active=false

switchRepository.deleteById(id); // **@**SQLDelete pretvara u UPDATE active=false

}

}

Kȏd 7 Servisna klasa SwitchService.java

U kontekstu ove aplikacije, servisne klase obavljaju funkcije kao što su dohvat svih entiteta (npr. switch uređaja, portova), dodavanje novih zapisa, pronalaženje zapisa po ID-ju, ažuriranje postojećih i njihovo brisanje. Osim osnovnih operacija, u servisni sloj se može smjestiti i dodatna logika poput provjera valjanosti ili povezivanja više entiteta u jednoj operaciji. Metoda u nastavku dohvaća sve preklopnike i prosljeđuje ih kontroleru, a kontroler odgovor vraća u JSON formatu.

public List**<**Switches**>** getAllSwitches() {

return switchRepository.findAll();

}

Kȏd 8 Implementacija metode servisne klase

Repozitorijski sloj (Repository layer) u Spring Boot aplikaciji odgovoran je za komunikaciju s bazom podataka. U ovoj aplikaciji koristi se Spring Data JPA, čime se omogućuje jednostavno rukovanje entitetima bez potrebe za pisanjem SQL upita. Umjesto ručnog pisanja koda za pristup bazi, dovoljno je definirati sučelje (interface) koje nasljeđuje JpaRepository [9].

public interface SwitchRepository extends JpaRepository**<**Switches, Long**>** {

}

Kȏd 9 Implementacija metode repozitorijske klase

Osim troslojne podjele, dodatna sigurnost postiže se i rukovanjem izuzecima, za sada na minimalnoj razini, no uz planiranu nadogradnju i povećanje kvalitete koda. Jedan od primjera rukovanja izuzecima vidljiv je u servisnoj klasi JpaUserDetailsService gdje se baca iznimka *UsernameNotFound* ako korisnik nije pronađen u bazi.

package com.networkapp.demo.service;

import com.networkapp.demo.repository.UserRepository;

import org.springframework.security.core.userdetails.\*;

import org.springframework.stereotype.Service;

import java.util.Collections;

**@**Service

public class JpaUserDetailsService implements UserDetailsService {

private final UserRepository userRepository;

public JpaUserDetailsService(UserRepository usersRepository) {

this.userRepository = usersRepository;

}

**@**Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

return userRepository.findByUsername(username)

.map(u **->** User.withUsername(u.getUsername())

.password(u.getPasswordHash()) // plain lozinka iz baze

.authorities(Collections.emptyList())

.build())

.orElseThrow(() **->** new UsernameNotFoundException("Korisnik nije pronađen: " **+** username));

}

}

Kȏd 10 Primjer rukovanja iznimkama

Sve navedeno osnovni je dio strukture backenda, no podjela koja olakšava održavanje sustava prisutna je i u frontendu, o čemu više slijedi u nastavku rada.

## Maven alat za upravljanje projektom

Apache Maven je alat za upravljanje projektima i sustav za automatizaciju izgradnje (build system) koji se primarno koristi u Java projektima. Omogućuje deklarativno upravljanje ovisnostima, izgradnjom, testiranjem i distribucijom softverskih paketa putem jednog konfiguracijskog XML dokumenta – pom.xml (Project Object Model). Ključna prednost Mavena je njegova sposobnost da pojednostavi upravljanje projektima, osiguravajući dosljednost i ponovljivost build procesa bez ručne intervencije [10].

U ovom projektu, Maven je korišten za konfiguraciju svih potrebnih ovisnosti kao što su Spring Boot Starter Web – za REST API-je, Spring Boot Starter Data JPA – za pristup bazi podataka, PostgreSQL driver – za povezivanje s bazom podataka, Spring Boot Starter Security – za autentifikaciju i autorizaciju.

Primjer pom.xml datoteke:

**<**?xml version="1.0" encoding="UTF-8"?**>**

**<**project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd"**>**

**<**modelVersion**>**4.0.0**<**/modelVersion**>**

**<**parent**>**

**<**groupId**>**org.springframework.boot**<**/groupId**>**

**<**artifactId**>**spring-boot-starter-parent**<**/artifactId**>**

**<**version**>**3.4.4**<**/version**>**

**<**relativePath/**>** <!-- lookup parent from repository -->

**<**/parent**>**

**<**groupId**>**com.networkapp**<**/groupId**>**

**<**artifactId**>**demo**<**/artifactId**>**

**<**version**>**0.0.1-SNAPSHOT**<**/version**>**

**<**name**>**demo**<**/name**>**

**<**description**>**Demo project for Spring Boot**<**/description**>**

**<**url/**>**

**<**licenses**>**

**<**license/**>**

**<**/licenses**>**

**<**developers**>**

**<**developer/**>**

**<**/developers**>**

**<**scm**>**

**<**connection/**>**

**<**developerConnection/**>**

**<**tag/**>**

**<**url/**>**

**<**/scm**>**

**<**properties**>**

**<**java.version**>**21**<**/java.version**>**

**<**/properties**>**

**<**dependencies**>**

**<**dependency**>**

**<**groupId**>**org.springframework.boot**<**/groupId**>**

**<**artifactId**>**spring-boot-starter-data-jpa**<**/artifactId**>**

**<**/dependency**>**

**<**dependency**>**

**<**groupId**>**org.springframework.boot**<**/groupId**>**

**<**artifactId**>**spring-boot-starter-web**<**/artifactId**>**

**<**/dependency**>**

**<**dependency**>**

**<**groupId**>**javax.xml.bind**<**/groupId**>**

**<**artifactId**>**jaxb-api**<**/artifactId**>**

**<**version**>**2.3.1**<**/version**>**

**<**/dependency**>**

**<**dependency**>**

**<**groupId**>**org.springframework.boot**<**/groupId**>**

**<**artifactId**>**spring-boot-starter-security**<**/artifactId**>**

**<**/dependency**>**

**<**dependency**>**

**<**groupId**>**org.springframework.boot**<**/groupId**>**

**<**artifactId**>**spring-boot-devtools**<**/artifactId**>**

**<**scope**>**runtime**<**/scope**>**

**<**optional**>**true**<**/optional**>**

**<**/dependency**>**

**<**dependency**>**

**<**groupId**>**org.postgresql**<**/groupId**>**

**<**artifactId**>**postgresql**<**/artifactId**>**

**<**scope**>**runtime**<**/scope**>**

**<**/dependency**>**

**<**dependency**>**

**<**groupId**>**io.jsonwebtoken**<**/groupId**>**

**<**artifactId**>**jjwt**<**/artifactId**>**

**<**version**>**0.9.1**<**/version**>**

**<**/dependency**>**

**<**dependency**>**

**<**groupId**>**org.springframework.boot**<**/groupId**>**

**<**artifactId**>**spring-boot-starter-test**<**/artifactId**>**

**<**scope**>**test**<**/scope**>**

**<**/dependency**>**

**<**/dependencies**>**

**<**build**>**

**<**plugins**>**

**<**plugin**>**

**<**groupId**>**org.springframework.boot**<**/groupId**>**

**<**artifactId**>**spring-boot-maven-plugin**<**/artifactId**>**

**<**/plugin**>**

**<**/plugins**>**

**<**/build**>**

**<**/project**>**

Kȏd 11 Pom.xml datoteka projekta

Maven koristi središnji repozitorij iz kojeg automatski preuzima ovisnosti i po potrebi ih lokalno sprema, što dodatno ubrzava radni proces. U kombinaciji sa Spring Boot frameworkom, Maven je postao standard u modernom razvoju web-aplikacija temeljenih na Javi.

# Frontend rješenje i korisničko sučelje

Korisničko sučelje izrađeno je korištenjem React biblioteke, čime je omogućena izrada dinamične i responzivne aplikacije. Frontend omogućuje korisniku da komunicira s aplikacijom putem forme i tabličnih prikaza, dok se komunikacija s backendom ostvaruje putem HTTP zahtjeva (Axios). Korištenjem hookova kao što su useState i useEffect postiže se upravljanje stanjem i dohvat podataka u stvarnom vremenu. Svi podaci koji se prikazuju na sučelju dolaze iz baze putem REST API-ja.

## Struktura frontenda

Frontend aplikacija razvijena je korištenjem React biblioteke, što omogućuje komponentni pristup i jednostavno upravljanje stanjem aplikacije. Projektna struktura unutar src/ direktorija organizirana je u više cjelina:

* api/ – sadrži axiosInstance.js i authApi.js datoteke za konfiguraciju i slanje HTTP zahtjeva prema backendu,
* components/ – glavna komponentna mapa u kojoj se nalaze sve vizualne i logičke cjeline aplikacije: Dashboard, Login, Lokacije, Portovi, PretragaUticnice, Switch, Sidebar, Topbar itd.
* context/ – sadrži AuthContext.jsx koji definira kontekst autentikacije i omogućuje prijenos korisničkih podataka kroz komponente.

Glavni ulazni sloj aplikacije nalazi se u App.js, a ostale datoteke poput App.css i index.js inicijaliziraju aplikaciju i globalne stilove.

Opisana struktura vidljiva je na slici u nastavku.

A screenshot of a computer

AI-generated content may be incorrect.

Slika 3 Frontend struktura

## HTML struktura i interaktivni elementi

React koristi JSX sintaksu koja kombinira JavaScript i HTML. Svaka komponenta sadrži vlastitu strukturu sučelja. Na primjer, komponenta Portovi.jsx prikazuje tablicu portova sa svim pripadajućim podacima, dok komponenta PretragaUticnice.jsx omogućuje korisniku unos oznake utičnice te vraća pripadajući rezultat.

Korišteni su standardni HTML elementi unutar React komponenti: <input>, <form>, <table>, <button>, <select>, uz odgovarajuće event listenere (npr. onClick, onChange, onSubmit).

U tablici u nastavku prikazani su primjeri korištenja navedenoga.

Tablica 5 Korištenje HTML i interaktivnih elemenata

|  |  |  |
| --- | --- | --- |
| **Element** | **Primjer** | **Opis elementa** |
| <h2> | <h2>Pretraga po utičnici</h2> | Podnaslov |
| <input> | **<**input className="form-control" placeholder="Upiši ime utičnice..." /**>** | Input polje za unos kriterija pretrage |
| <button > | **<**input className="form-control" placeholder="Upiši ime utičnice..." /**><**button className="btn btn-primary mb-3"**>**Traži**<**/button**>** | Gumb za pokretanje pretrage< |
| <table> | **<**table className="table table-bordered"**>**  **<**thead**>**  **<**tr**>**  **<**th**>**Switch**<**/th**>**  **<**th**>**IP**<**/th**>**  **<**th**>**Port**<**/th**>**  **<**th**>**Utičnica**<**/th**>**  **<**th**>**Korisnik**<**/th**>**  **<**/tr**>**  **<**/thead**>**  **<**tbody**>**  <!-- Redovi rezultata -->  **<**/tbody**>**  **<**/table**>** | Tablica koja ispisuje switcheve i njihove podatke |
| alert | **<**div className="alert alert-warning"**>**  Nema rezultata za traženu utičnicu.  **<**/div**>** | Obavijest o nepostojanju rezultata za tražene kriterije |
| spinner (interaktivni element) | **<**span className="spinner-border spinner-border-sm me-2"**><**/span**>** | *Spinner* koji se prikazuje dok se dohvaćaju podaci |

Primjer korištenja spinnera vidljiv je na ekranu za prijavu, gdje se pritiskom na gumb Prijava učitava spinner koji do trenutka učitanja početne stranice prikazuje tekst „Prijava…”.

A screenshot of a computer

AI-generated content may be incorrect.

Slika 4 Loading spinner pri prijavi u sustav

## JavaScript logika i povezivanje s REST API-jem

Povezivanje s backend servisima obavljeno je pomoću axios biblioteke koja pojednostavljuje rad s HTTP zahtjevima. Sve funkcije koje komuniciraju s REST endpointima nalaze se u authApi.js i drugim datotekama unutar api/ direktorija.

Primjer poziva za dohvat svih switcheva:

const getAllSwitches = async () =**>** {

const response = await axiosInstance.get("/api/switches");

return response.data;

};

Kȏd 12 Primjer API poziva

Podaci se dohvaćaju u useEffect() hookovima i spremaju u lokalno stanje (useState). Na taj način omogućeno je ažuriranje prikaza bez ponovnog učitavanja stranice. Navedeno ima ključnu ulogu kada govorimo o dinamičkim podacima. Naime, useState omogućuje definiranje i ažuriranje stanja unutar funkcionalnih komponenti. Na primjer, u komponenti za pretragu koristi se za praćenje vrijednosti unesenih u input polja, kao i za pohranu rezultata pretrage [10]:

const [uticnica, setUticnica] = useState("");

const [rezultati, setRezultati] = useState([]);

Kȏd 13 Primjer upotrebe useStatea

Ovaj pristup omogućava reaktivno ažuriranje korisničkog sučelja – čim se stanje promijeni, komponenta se automatski ponovno renderira. useEffect se koristi u slučaju poput dohvaćanja podataka s poslužitelja pri učitavanju neke komponente [11].

## Dinamičko dohvaćanje podataka (kaskadni dropdowni)

U korisničkom sučelju aplikacije primijenjen je obrazac kaskadnog odabira (eng. cascading dropdowns), gdje je izbor vrijednosti u jednoj padajućoj listi ovisan o prethodno odabranoj vrijednosti. Na primjer, korisnik najprije odabire lokaciju, zatim se dinamički prikazuju dostupni katovi povezani s tom lokacijom, a nakon toga i pripadajuće sobe.

Ovakva funkcionalnost realizirana je u Reactu pomoću useEffect hooka, koji omogućuje reakciju na promjene stanja (selectedLocationId) te dohvat podataka s poslužitelja koristeći fetch() metodu. Svaki put kada korisnik promijeni lokaciju, pokreće se API poziv koji dohvaća sve katove povezane s tom lokacijom, a zatim i sobe. Time se osigurava da korisnik uvijek ima prikazane samo one opcije koje su relevantne za njegov prethodni odabir, čime se smanjuje mogućnost pogrešaka i povećava korisnička učinkovitost.

Ovakav obrazac detaljno je objašnjen i u vodiču na FreeCodeCampu, gdje se objašnjavaju ključne faze implementacije, uključujući definiranje stanja, asinkroni dohvat podataka i upravljanje zavisnostima između elemenata forme [13]. Primjena ove funkcionalnosti vidljiva je u nastavku:

useEffect(() =**>** {

if (selectedLocationId) {

fetch(`/api/floors/location/${selectedLocationId}`)

.then((res) =**>** res.json())

.then((data) =**>** setFloors(data));

}

}, [selectedLocationId]);

Kȏd 14 Primjena kaskadnog dropdowna

## Uporaba Axios biblioteke

Axios je JavaScript biblioteka koja omogućuje slanje HTTP zahtjeva s klijentske strane. U ovoj aplikaciji Axios se koristi za komunikaciju s backend REST API-jem, omogućujući dohvat podataka (GET), slanje podataka (POST), ažuriranje (PUT) i brisanje (DELETE). Axios nudi jednostavniji i pregledniji sintaksni oblik u odnosu na fetch() API, uz dodatne funkcionalnosti poput automatskog pretvaranja JSON odgovora i globalnog postavljanja zaglavlja [14].

U kȏdu 15 koristi se useEffect() hook koji se izvršava prilikom prvog učitavanja komponente. Unutar hooka koriste se dva GET zahtjeva pomoću Axios instance: jedan za dohvat svih switch uređaja i drugi za dohvat dostupnih lokacija. Nakon uspješnog odgovora, podaci se spremaju u pripadajuća stanja, što omogućuje njihovo prikazivanje ili daljnju obradu u korisničkom sučelju.

useEffect(() =**>** {

axiosInstance.get("/api/switches")

.then((res) =**>** {

const sortirani = [...res.data].sort((a, b) =**>** a.id **-** b.id);

setSwitchevi(sortirani);

})

.catch((err) =**>** console.error("Greška pri dohvaćanju switcheva:", err));

axiosInstance.get("/api/locations")

.then((res) =**>** setLokacije(res.data))

.catch((err) =**>** console.error("Greška pri dohvaćanju lokacija:", err));

}, []);

Kȏd 15 Dohvaćanje podataka o switchevima i lokacijama pomoću Axiosa

# Git i Github

Git je sustav za upravljanje verzijama otvorenog kôda koji omogućuje praćenje promjena u izvornoj datoteci i podržava razvoj softverskih rješenja kroz suradnju cijelog tima [15]. Za razliku od centraliziranih sustava za kontrolu verzija, Git omogućuje svakom korisniku da lokalno ima cijelu povijest projekta, što značajno ubrzava rad i omogućuje potpunu kontrolu bez stalne povezanosti s udaljenim poslužiteljem.

GitHub je web-platforma za pohranu i suradnju temeljena na Git sustavu. Osim hostanja Git repozitorija, GitHub nudi i dodatne alate za upravljanje projektima, kao što su: issues, pull requests, wiki, dokumentacija i CI/CD integracije [16].

U kontekstu izrade ove aplikacije, Git i GitHub su korišteni za:

* praćenje svih promjena na frontendu i backendu,
* upravljanje verzijama,
* sigurnu pohranu cijelog projekta u oblaku.

## Praktična primjena Gita i GitHuba

Na početku rada, inicijaliziran je Git repozitorij naredbom git init, a projekt je podijeljen u dva glavna dijela – backend (zavrsni-rad-back) i frontend (zavrsni-rad-front), svaki u svojem repozitoriju.

U svakoj fazi razvoja redovno su izvršavane naredbe:

* git add .
* git commit -m "Dodana funkcionalnost pretrage po utičnici"
* git push origin master

GitHub je korišten za hostanje oba repozitorija i omogućavanje pregleda koda iz bilo kojeg preglednika, kao i izradu sigurnosne kopije projekta. Na slikama u nastavku prikazan je izgled repozitorija i hijerarhijska struktura foldera (npr. src/main/java, src/components, api, controller, service, itd.).

A screenshot of a computer

AI-generated content may be incorrect.

Slika 5 Frontend Github repozitorij

A screenshot of a computer

AI-generated content may be incorrect.

Slika 6 Backend Github Repozitorij

A screenshot of a computer program

AI-generated content may be incorrect.

Slika 7 Primjer Git naredbi za dohvaćanje i prikaz statusa repozitorija

Jedna od važnih prednosti bila je mogućnost vraćanja na prethodne verzije projekta u slučaju grešaka, što je u više navrata znatno ubrzalo otklanjanje problema tijekom razvoja. Korištenjem naredbe git log moguće je pregledavati sve prethodne promjene zajedno s opisima i autorima commitova.

Git i GitHub su ključni alati u modernom razvoju softvera. Njihova primjena u ovom projektu omogućila je efikasnu organizaciju rada, sigurnost koda i veću fleksibilnost pri eksperimentiranju s različitim funkcionalnostima. Zahvaljujući verzioniranju, cijeli tijek razvoja aplikacije je dokumentiran i lako je moguće vratiti se na ranije verzije ako se pojavi potreba.

# Prikaz ključnih funkcionalnosti

U ovom poglavlju prikazane su ključne funkcionalnosti aplikacije kroz primjere korisničkog sučelja. Funkcionalnosti su implementirane koristeći React na frontend strani i Spring Boot na backend strani. Korisnicima je omogućeno jednostavno upravljanje mrežnom infrastrukturom kroz intuitivno i pregledno sučelje.

## Pregled i unos switch uređaja

Korisnici mogu pregledati sve unesene mrežne uređaje (switcheve) u obliku tablice koja sadrži osnovne informacije poput naziva, modela, IP adrese, tipa veze i pripadajuće lokacije. Dostupna je opcija za filtriranje prema lokaciji, kao i mogućnosti dodavanja novog uređaja klikom na gumb „Dodaj novi switch”.

A screenshot of a computer

AI-generated content may be incorrect.

Slika 8 Prikaz tablice sa switchevima

Unos novog uređaja uključuje popunjavanje forme s poljima kao što su: naziv, model, IP adresa, tip (access/core), veza (ethernet/fiber), lokacija, kat, soba i inventarni broj. Dropdowni za katove i sobe se dinamički ažuriraju ovisno o odabranoj lokaciji, što je realizirano kaskadnim dohvaćanjem podataka.

Moguće je i filtriranje switcheva prema lokacijama radi bržeg dolaska do informacije, pogotovo kada je korisnik na terenu.

A screenshot of a computer

AI-generated content may be incorrect.

Slika 9 Filtriranje switcheva po lokaciji

Popis lokacija nalazi se u zasebnoj tablici, a nove lokacije, s jednostavnim opisnim informacijama – nazivom i opisom – moguće je dodati odabirom opcije Lokacije u izborniku, a zatim pritiskom gumba za dodavanje nove lokacije. Osim toga, moguće je uređivati i brisati postojeće lokacije, pri čemu se kod uređivanja polja popunjavaju podacima iz baze.

A screenshot of a computer

AI-generated content may be incorrect.

Slika 10 Popis lokacija

A screenshot of a computer

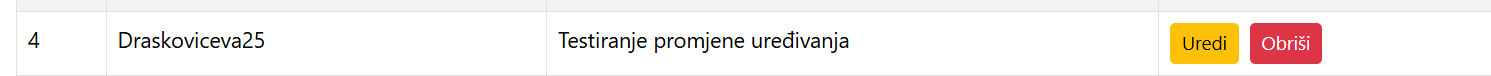
AI-generated content may be incorrect.

Slika 11 Dodavanje nove lokacije

A screenshot of a computer

AI-generated content may be incorrect.

Slika 12 Uređivanje postojeće lokacije



Slika 13 Uspješna izmjena opisa lokacije

## Upravljanje priključcima i VLAN-ovima

Klikom na gumb „Portovi” za pojedini switch otvara se stranica s pregledom svih njegovih priključaka (portova). Za svaki port prikazane su informacije o broju porta, oznaci utičnice, VLAN-u, sobi, statusu povezivanja i imenu korisnika.

A screenshot of a computer

AI-generated content may be incorrect.

Slika 14 Prikaz portova na Switchu A1

Korisnik može uređivati pojedini port klikom na gumb „Uredi”, čime se otvara modalni prozor s formom. Moguće je ažurirati broj porta, oznaku utičnice, VLAN, naziv sobe te korisnika. Također se može označiti je li port povezan ili ne.

## Pretraživanje po imenu korisnika i utičnici

Aplikacija omogućuje brzo pretraživanje mrežnih priključaka prema oznaci utičnice ili imenu korisnika. Na posebnoj stranici za pretragu korisnik unosi jedan od ta dva podatka, a rezultat pretrage prikazuje sve relevantne portove, zajedno s pripadajućim switchem, IP adresom, brojem porta i korisničkim imenom. U slučaju da nema rezultata, aplikacija prikazuje poruku upozorenja. Funkcionalnost je korisna za brzo identificiranje priključka prema stvarnoj fizičkoj utičnici ili zaposleniku.

A screenshot of a computer

AI-generated content may be incorrect.

Slika 15 Pretraga po nazivu utičnice

A screenshot of a computer

AI-generated content may be incorrect.

Slika 16 Pretraga utičnice po korisniku

# Testiranje

Testiranje aplikacije ima ključnu ulogu u osiguravanju ispravnosti i stabilnosti funkcionalnosti implementiranih na frontendu i backendu. Tijekom razvoja provedeno je funkcionalno testiranje kako bi se utvrdilo da svi dijelovi aplikacije rade prema očekivanjima krajnjeg korisnika.

## Metodologija testiranja

Testiranje je provedeno ručno koristeći korisničko sučelje i Postman za provjeru slanja i dohvaćanja podataka, odnosno provjeru API poziva. Cilj testiranja bio je provjeriti:

* ispravno dohvaćanje i prikaz podataka iz baze (GET zahtjevi),
* dodavanje novih podataka putem obrazaca (POST),
* ažuriranje postojećih entiteta (PUT),
* brisanje zapisa iz baze (DELETE),
* ponašanje aplikacije u slučaju greške (npr. nepostojeći ID, prazna polja, nedostupna lokacija),
* validaciju obrazaca na frontend strani.

Korištenjem različitih kombinacija ulaza, ispravnih i neispravnih podataka i sličnih metoda provjerene su različite funkcionalnosti aplikacije kako bi ih korisnik mogao koristiti bez nailaska na prepreke.

## Testiranje backend API-ja

Backend API testiran je pomoću Postman alata. Postman je platforma za razvoj i testiranje API-ja koja omogućuje jednostavno slanje HTTP zahtjeva, analizu odgovora i upravljanje različitim okruženjima. Prema službenoj stranici Postmana [17], alat je dizajniran kako bi ubrzao rad razvojnih timova kroz mogućnost testiranja, dokumentiranja i dijeljenja API-ja. Podržava različite metode autentifikacije, uključujući Basic Auth i Bearer tokene, što ga čini pogodnim za rad s modernim sigurnosnim protokolima poput JWT-a. Postman omogućuje korisnicima organizaciju zahtjeva u kolekcije, automatizirano izvođenje testova i brojne druge mogućnosti. Zbog svoje preglednosti i funkcionalnosti postao je jedan od najčešće korištenih alata u razvoju aplikacija temeljenih na REST arhitekturi [18].

Provjerene su sve CRUD operacije za entitete: locations, switches i ports. U nastavku je primjer POST zahtjeva za prijavu te GET zahtjeva s prijavljenim i neprijavljenim korisnikom.

Da bismo razumjeli odgovore koje dobivamo testirajući u Postmanu, moramo razumjeti što znači koji HTTP status kod koji se prikaže [18].

Tablica 6 Popis HTTP status kodova[[1]](#endnote-1)

|  |  |  |
| --- | --- | --- |
| **Status kod** | **Naziv** | **Opis** |
| 200 | OK | Zahtjev je uspješno obrađen. |
| 201 | Created | Novi resurs je uspješno kreiran. |
| 204 | No Content | Zahtjev je uspješno obrađen, ali nema povratnih podataka. |
| 301 | Moved Permanently | Resurs je trajno premješten na drugu adresu. |
| 302 | Found | Resurs je privremeno premješten. |
| 400 | Bad Request | Neispravan zahtjev (npr. pogrešan JSON format). |
| 401 | Unauthorized | Autentikacija nije provedena ili nije valjana. |
| 403 | Forbidden | Pristup je zabranjen iako je korisnik autentificiran. |
| 404 | Not Found | Traženi resurs nije pronađen. |
| 409 | Conflict | Sukob, najčešće kod pokušaja duplikata unosa. |
| 500 | Internal Server Error | Neodređena greška na strani poslužitelja. |
| 503 | Service Unavailable | Poslužitelj nije trenutačno dostupan (npr. održavanje). |

Primjer testiranja GET zahtjeva, zajedno s odgovorom, korisnika koji nije prijavljen u sustavu te onoga koji je, vidljiv je na slikama u nastavku.

1. GET <http://localhost:8083/api/switches> (bez uspješne prijave u sustav)

A screenshot of a computer

AI-generated content may be incorrect.

Slika 17 Neuspješno dohvaćanje podataka u Postmanu

Greška 403 Forbidden u Postmanu znači da je zahtjev poslan, ali je poslužitelj odbio izvršiti ga jer korisnik nije autoriziran, što znači da je autentikacija i autorizacija uspješno implementirana u sustav.

Nakon toga testirali smo prijavu u sustav u Postman alatu tako što smo koristili POST metodu <http://localhost:8083/users/login>, a u tijelu zahtjeva slali smo korisničko ime i lozinku u *raw* JSON formatu. Zahtjev je bio uspješan, što potvrđuje status 200 OK.

A screenshot of a computer

AI-generated content may be incorrect.

Slika 18 Prijava u sustav u Postmanu

Nakon uspješne prijave u sustav ponovno je poslan GET zahtjev za dohvaćanje switcheva, ovoga puta uspješno (slika 19).

A screenshot of a computer

AI-generated content may be incorrect.

Slika 19 Uspješan GET zahtjev u Postmanu

## Testiranje korisničkog sučelja

Na frontend strani testirane su funkcionalnosti pomoću ručnog testiranja u pregledniku Chrome. Provjeravana je prijava s ispravnim i neispravnim korisničkim podacima, dohvaća li se popis switcheva ispravno pri učitavanju stranice, dodavanje novog switcha kroz formu, reagira li forma na nedostatke (npr. neodabrana lokacija), dohvaćaju li se katovi i sobe ispravno nakon odabira lokacije (kaskadni dropdown), može li se uređivati postojeći zapis i prikazuju li se promjene odmah u tablici, reagira li aplikacija na greške i prikazuje li korisniku korisne poruke, kao i radi li pretraga po korisniku i utičnici kako je zamišljeno.

A screenshot of a phone

AI-generated content may be incorrect.

Slika 20 Pokušaj unosa switcha bez lokacije

Kada pokušamo dodati switch bez odabira lokacije iz kaskadnog dropdowna, dobivamo obavijest da je potrebno odabrati lokaciju jer to polje u bazi ne može imati vrijednost null.

Testirano je i brisanje dodanog switcha koristeći soft delete opciju. Odabirom gumba „Obriši” javlja se dijaloški okvir koji nas traži potvrdu brisanja (slika 21).

A screenshot of a computer

AI-generated content may be incorrect.

Slika 21 Dijaloški okvir za potvrdu brisanja

Prije brisanja vrijednost stupca *active* u tablici ima vrijednost true (slika 22) za navedeni switch, a nakon potvrde brisanja i realizacije soft delete opcije, vrijednost se mijenja u false (slika 23).

A screenshot of a computer

AI-generated content may be incorrect.

Slika 22 Aktivni testni switch

A screenshot of a computer

AI-generated content may be incorrect.

Slika 23 Neaktivni testni switch

8.4. Rezultati testiranja

Pri testiranju su se pojavile neke kritične greške, poput omogućene prijave s neispravnim korisničkim podacima ili nemogućnosti brisanja zapisa zbog veze s drugim tablicama i slično, no sve su kritične greške otklonjene. Identificirane su manje nepravilnosti (npr. validacija unosa sobe u slučaju nepostojećeg broja), koje su ispravljene u hodu.

Aplikacija odgovara zadanim funkcionalnim zahtjevima, omogućava jednostavno korištenje, pravovremeno reagira na pogreške i nudi dobar pregled podataka kroz intuitivno korisničko sučelje, no ima i prostora za napredak i nadogradnju prema budućim potrebama korisnika.

# Zaključak

Tijekom izrade ove web-aplikacije pokazalo se koliko digitalizacija i automatizacija procesa mogu unaprijediti svakodnevno upravljanje mrežnom infrastrukturom. Do sada su se svi podaci o mrežnim priključcima, uređajima i lokacijama vodili ručno – u papirnatom obliku ili u nepreglednim Excel dokumentima. Ovakav način rada bio je sklon pogreškama, otežavao je održavanje, a dodatno je usporavao rješavanje tehničkih problema i identifikaciju korisnika na mreži. S obzirom na to, glavni cilj ovog projekta bio je razviti stabilnu i jednostavnu web-aplikaciju koja će služiti kao centralno mjesto za evidenciju, pretragu i upravljanje mrežnim uređajima i priključcima.

Kroz rad je primijenjena troslojna arhitektura, što znači da su frontend, backend i baza podataka jasno odvojeni. Na taj način postignuta je modularnost sustava – svaki sloj razvijan je neovisno, što omogućuje jednostavniju nadogradnju, testiranje i održavanje. Backend aplikacija razvijena je pomoću Spring Boot frameworka i povezana s PostgreSQL bazom podataka, dok je korisničko sučelje izrađeno u Reactu, koristeći Axios za komunikaciju s REST API-jem. Podaci o switch uređajima, portovima, VLAN-ovima, sobama, katovima i lokacijama strukturirani su u skladu s relacijskim modelom baze podataka, a Hibernate i JPA omogućili su jednostavnu integraciju.

Jedan od izazova u razvoju bio je omogućiti intuitivno korisničko sučelje koje se može koristiti i bez naprednog tehničkog znanja. U tu svrhu implementirani su elementi kao što su kaskadni dropdowni za filtriranje lokacija, katova i soba, dinamičko dohvaćanje portova za svaki switch, modalni prozori za unos i uređivanje, te validacija obrazaca. Poseban naglasak stavljen je na funkcionalnosti pretrage, koje omogućuju pronalaženje informacija na temelju korisnika ili oznake utičnice – što je od ključne važnosti za svakodnevno održavanje mreže.

Testiranje aplikacije provedeno je pomoću alata Postman i kroz ručno testiranje u stvarnom korisničkom okruženju. Pokazalo se da sustav uspješno podržava sve osnovne CRUD operacije i korektno reagira na greške, čime je potvrđena njegova stabilnost. Povratne informacije od krajnjih korisnika bile su pozitivne uz nekoliko konstruktivnih kritika koje su realizirane, posebno u pogledu preglednosti podataka i brzine pristupa informacijama.

Završetkom ovog rada kreiran je funkcionalan sustav koji na osnovnoj razini rješava trenutne probleme krajnjih korisnika i ostavlja prostor za daljnje nadogradnje.

Ova aplikacija predstavlja primjer kako se informatička rješenja mogu primijeniti u kontekstu mrežne administracije s ciljem postizanja veće učinkovitosti, preciznosti i organizacije rada.

# Popis literature

[1] GeeksforGeeks, „Spring Initializr in Spring Boot”, GeeksforGeeks, 2025. [Online]. Dostupno na: <https://www.geeksforgeeks.org/springboot/spring-initializr/>

[2] B. Plančić, Sigurnost web-aplikacija temeljenih na REST arhitekturi, diplomski rad, PMF, Sveučilište u Zagrebu, 2019.

[3] GeeksforGeeks, „What is a Switch Port in Networking?, ” GeeksforGeeks, 2025. [Online]. Dostupno na: <https://www.geeksforgeeks.org/computer-networks/what-is-a-switch-port/>

[4] CERT.hr, „Virtualne lokalne mreže (VLAN) – Nacionalni CERT, ” CERT.hr, 2006. [Online]. Dostupno na: <https://www.cert.hr/NCVlan>

[5] J. Stafford, „Java Persistence API (JPA) Notes,” ejava-springboot, 2025. [Online]. Dostupno na: <https://jcs.ep.jhu.edu/ejava-springboot/coursedocs/content/pdf/jpa-notes.pdf>

[6] Hibernate.org, „Hibernate ORM,” Hibernate, 2025. [Online]. Dostupno na: <https://hibernate.org/orm/>

[7] CERT.hr, „Sigurnost HTTP API-ja,” CERT.hr, 2020. [Online]. Dostupno na: <https://www.cert.hr/wp-content/uploads/2020/03/Sigurnost_HTTP_API-ja.pdf>

[8] GeeksforGeeks, „Spring Boot Service Annotation with Example, ” GeeksforGeeks, 2025. [Online]. Dostupno na: <https://www.geeksforgeeks.org/advance-java/spring-boot-service-annotation-with-example/>

[9] GeeksforGeeks, „Spring Repository Annotation with Example, ” GeeksforGeeks, 2025. [Online]. Dostupno na: <https://www.geeksforgeeks.org/springboot/spring-repository-annotation-with-example/>

[10] Apache Software Foundation, „Introduction to the POM,” Maven Project, 2025. [Online]. Dostupno na: <https://maven.apache.org/pom.html>

[11] React, „Hooks API Reference,” React Official Documentation, 2025. [Online]. Dostupno na: <https://reactjs.org/docs/hooks-intro.html>

[12] React, „Using the Effect Hook,” React Official Documentation, 2025. [Online]. Dostupno na: <https://legacy.reactjs.org/docs/hooks-effect.html>

[13] G. Rajan, „How to Build Dependent Dropdowns in React,” freeCodeCamp, 2025. [Online]. Dostupno na: <https://www.freecodecamp.org/news/how-to-build-dependent-dropdowns-in-react/>

[14] Axios, „Axios Documentation – Introduction,” axios-http.com, 2025. [Online]. Dostupno na: <https://axios-http.com/docs/intro>

[15] S. Cacon i B. Straub, *Pro Git*, Apress, 2014. [Online]. Dostupno na: <https://git-scm.com/book/en/v2>

[16] GitHub Docs, “About GitHub,” GitHub Documentation, 2024. [Online]. Dostupno na: <https://docs.github.com/en/get-started/quickstart/about-github>

[17] Postman, „What is Postman?,” Postman.com, 2025. [Online]. Dostupno na: <https://www.postman.com/product/what-is-postman/>

[18] Postman, „What Are HTTP Status Codes?,” Postman Blog, 2025. [Online]. Dostupno na: <https://blog.postman.com/what-are-http-status-codes/>

# Prilozi

## Popis slika

[Slika 1 ER dijagram baze podataka 13](#_Toc206703744)

[Slika 2 Backend struktura 22](#_Toc206703745)

[Slika 3 Frontend struktura 35](#_Toc206703746)

[Slika 4 Loading spinner pri prijavi u sustav 37](#_Toc206703747)

[Slika 5 Frontend Github repozitorij 42](#_Toc206703748)

[Slika 6 Backend Github Repozitorij 42](#_Toc206703749)

[Slika 7 Primjer Git naredbi za dohvaćanje i prikaz statusa repozitorija 43](#_Toc206703750)

[Slika 8 Prikaz tablice sa switchevima 44](#_Toc206703751)

[Slika 9 Filtriranje switcheva po lokaciji 45](#_Toc206703752)

[Slika 10 Popis lokacija 45](#_Toc206703753)

[Slika 11 Dodavanje nove lokacije 46](#_Toc206703754)

[Slika 12 Uređivanje postojeće lokacije 46](#_Toc206703755)

[Slika 13 Uspješna izmjena opisa lokacije 47](#_Toc206703756)

[Slika 14 Prikaz portova na Switchu A1 47](#_Toc206703757)

[Slika 15 Pretraga po nazivu utičnice 48](#_Toc206703758)

[Slika 16 Pretraga utičnice po korisniku 48](#_Toc206703759)

[Slika 17 Neuspješno dohvaćanje podataka u Postmanu 51](#_Toc206703760)

[Slika 18 Prijava u sustav u Postmanu 52](#_Toc206703761)

[Slika 19 Uspješan GET zahtjev u Postmanu 53](#_Toc206703762)

[Slika 20 Pokušaj unosa switcha bez lokacije 54](#_Toc206703763)

[Slika 21 Dijaloški okvir za potvrdu brisanja 55](#_Toc206703764)

[Slika 22 Aktivni testni switch 55](#_Toc206703765)

[Slika 23 Neaktivni testni switch 56](#_Toc206703766)

## Popis tablica

[Tablica 1 Popis zahtjeva korisnika 6](#_Toc206703767)

[Tablica 2 Osnovni pojmovi i definicije 9](#_Toc206703768)

[Tablica 3 Struktura tablice „switches” 14](#_Toc206703769)

[Tablica 4 Struktura tablice „ports” 15](#_Toc206703770)

[Tablica 5 Korištenje HTML i interaktivnih elemenata 36](#_Toc206703771)

[Tablica 6 Popis HTTP status kodova 50](#_Toc206703772)

## Popis kȏdova

[Kȏd 1 Primjer entity klase 17](#_Toc206703773)

[Kȏd 2 Primjer repository klase 18](#_Toc206703774)

[Kȏd 3 Primjer service klase 20](#_Toc206703775)

[Kȏd 4 Primjer controller klase 21](#_Toc206703776)

[Kȏd 5 Primjer podataka u JSON formatu 25](#_Toc206703777)

[Kȏd 6 Implementacija kontrolerske klase 26](#_Toc206703778)

[Kȏd 7 Servisna klasa SwitchService.java 29](#_Toc206703779)

[Kȏd 8 Implementacija metode servisne klase 29](#_Toc206703780)

[Kȏd 9 Implementacija metode repozitorijske klase 30](#_Toc206703781)

[Kȏd 10 Primjer rukovanja iznimkama 30](#_Toc206703782)

[Kȏd 11 Pom.xml datoteka projekta 32](#_Toc206703783)

[Kȏd 12 Primjer API poziva 37](#_Toc206703784)

[Kȏd 13 Primjer upotrebe useStatea 38](#_Toc206703785)

[Kȏd 14 Primjena kaskadnog dropdowna 38](#_Toc206703786)

[Kȏd 15 Dohvaćanje podataka o switchevima i lokacijama pomoću Axiosa 40](#_Toc206703787)

Slika na kojoj se prikazuje tekst, Font, skeč, crta

Opis je automatski generiran

**VELEUČILIŠTE HRVATSKO ZAGORJE KRAPINA**

**POTVRDA**

Potvrđujem da je student Filip Žgela izradio završni rad pod naslovom:

**NASLOV RADA**

u skladu sa zadanim zadatkom, tezama i pravilima struke. Rad je izrađen prema Uputama za izradu završnog rada, testiran je na plagijarizam i zadovoljava uvjete izvornosti. Potvrđujem da se rad može uvezati i predati.

**Mentor:**

Stjepan Šalković, dipl. inf. univ. spec. oec.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Krapina, rujan 2025.

Slika na kojoj se prikazuje tekst, Font, skeč, crta

Opis je automatski generiran

**VELEUČILIŠTE HRVATSKO ZAGORJE KRAPINA**

**IZJAVA O AKADEMSKOJ ČESTITOSTI**

Ja, Filip Žgela, izjavljujem da je moj završni rad pod naslovom *Web-aplikacija za praćenje i evidenciju mrežnih priključaka na preklopnicima* izvorni rezultat mojeg rada, da se temelji na mojim istraživanjima, te da se u njegovoj izradi nisam koristio drugim izvorima osim onima koji su u njemu navedeni u bilješkama i popisu literature. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da nije prepisan iz necitiranog rada te da nijedan dio rada ne krši bilo čija autorska prava. Također izjavljujem, da nijedan dio rada nije iskorišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

Sadržaj mojega rada u potpunosti odgovara sadržaju obranjenoga rada.

Svojim vlastoručnim potpisom potvrđujem da sam suglasan da se moj završni rad javno objavi na internetu.

U Krapini, rujan 2025.

Filip Žgela

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. [↑](#endnote-ref-1)