Web aplikacija za upravljanje domenskim konfiguracijama s IntelliSense mogućnostima

Todić, Juraj

Sveučilište Josipa Jurja Strossmayera u Osijeku

Odjel za matematiku

Sveučilišni preddiplomski studij Matematika i računarstvo

Web aplikacija za upravljanje domenskim konfiguracijama s IntelliSense mogućnostima

Završni rad

Mentor:

izv. prof. dr. sc.
Domagoj Matijević

Komentor:

mag. ing. comp. Ivan Kordaso
Enea Software d.o.o.

Kandidat:

Juraj Todić

Osijek, 2023

Web application for managing domain configurations with IntelliSense

Sažetak

Ovaj rad bavi se razvojem web aplikacije za upravljanje specifičnim domenskim konfiguracijama s dodanim IntelliSense mogućnostima. Objasnit ćemo strukturu web aplikacije, funkcionalnosti backend i frontend dijela aplikacije, pokazat ćemo kako smo implementirali validaciju specifičnih domenskih konfiguracija te objasniti implementaciju Monaco uređivača koda kojeg ćemo proširiti s IntelliSense mogućnostima. Ovaj projekt napravljen je u suradnji s tvrtkom ENEA.

Ključne riječi

domenska konfiguracija, Monaco uređivač koda, IntelliSense mogućnosti

Summary

This paper deals with the development of a web application for managing specific domain configurations with added IntelliSense capabilities. We will explain the structure of the web application, functionality of the backend and frontend part of the application, we will also show how we implemented validation of specific domain configurations and we will explain the implementation of the Monaco code editor, which we will extend with IntelliSense features. This project was done in collaboration with the company ENEA.

Keywords

domain configuration, Monaco code editor, IntelliSense features

Sadržaj

1 | Uvod

Web aplikacija je softver koji radi u web pregledniku. Omogućuje pristup složenim funkcijama bez instaliranja ili konfiguriranja softvera.
Domenski specifična konfiguracija odnosi se na praksu prilagođavanja softverskih aplikacija ili sustava kako bi se zadovoljile jedinstvene potrebe određene domene, industrije ili slučaja upotrebe umjesto korištenja pristupa koji odgovara širokom spektru situacija. To je način da se softver učini prilagodljivijim i učinkovitijim u rješavanju specifičnih zahtjeva različitih sektora, u konačnici poboljšavajući njegovu upotrebljivost i performanse unutar tih domena.
Web aplikacija za upravljanje domenskim konfiguracijama s IntelliSense mogućnostima omogućava pregled, izmjenu, dodavanje, uklanjanje i validaciju specifičnih domenskih konfiguracija pohranjenih u bazi podataka.

U nastavku ovog rada objasnit ćemo strukturu web aplikacije, funkcionalnosti backend i frontend dijela aplikacije, definirati domenski specifični jezik kako bismo validirali sadržaj domenske specifikacije te ćemo pokazati implementaciju Monaco uređivača koda kojeg ćemo proširiti s IntelliSense mogućnostima. Web aplikacija dostupna je na GitHub-u [11], zajedno s kratkim videom koji pokazuje osnovne funkcionalnosti. [12]

2 | Opis web aplikacije

Web aplikacije sastoje se od frontend i backend dijela. Frontend dio web aplikacije je onaj koji korisnici mogu vidjeti i s kojim mogu komunicirati preko grafičkog korisničkog sučelja (GUI). Dizajn, navigacijski izbornici, tekst, slike, video zapisi itd. čine frontend dio web aplikacije.
Backend je serverska strana web aplikacije. Backend pohranjuje i raspoređuje podatke i osigurava da sve na frontend strani radi na željen način. Backend je dio web aplikacije koji ne dolazi u izravan kontakt s korisnicima tj. korisnik ne može vidjeti niti može komunicirati s njim. Backend je zadužen za upravljenje podacima kao što su npr. komunikacija s bazom podataka, validacija podataka, dostavljanje i primanje podataka od strane frontend-a i sl.
Obje strane moraju međusobno komunicirati i djelovati kao jedinstvena jedinica kako bi se poboljšala funkcionalnost web stranice.

Svrha ove web aplikacije je mogućnost pregleda, izmjene, dodavanja, uklanjanja i validacije domenskih konfiguracija. Web aplikacija koristi popularni Spring framework kao backend, dok se za frontend koristi React, te Monaco uređivač koda, kao React komponenta. Uređivač koda proširen je osnovnim IntelliSense mogućnostima kao što su označavanje sintakse (engl. syntax highlighter) i dovršavanje/prijedlog koda (engl. code completion)

2.1 Korištene tehnologije

Spring je najpopularniji open-source framework za izgradnju aplikacija u programskom jeziku Java, razvijen s ciljem da pojednostavi složen i glomazan proces razvoja softvera u Javi. Pruža robusnu i modularnu infrastrukturu za razvoj jednostavnih i složenih aplikacija temeljenih na Javi, s fokusom na jednostavnost razvoja, skalabilnost i mogućnost održavanja. [1]

React, poznat i kao React.js ili ReactJS, je JavaScript open-source biblioteka koja se koristi za brzu i učinkovitu izradu korisničkih sučelja (UI) web aplikacija. U Reactu se aplikacije razvijaju stvaranjem komponenti za višekratnu upotrebu. Ove su komponente pojedinačni dijelovi konačnog sučelja koji, kada se sastave, čine cjelokupno korisničko sučelje aplikacije. Razvio ga je i održava Facebook.

Monaco uređivač koda lagan je i brz uređivač koda koji se prvenstveno koristi za ugrađivanje mogućnosti uređivanja koda u web aplikacije. To je open-source

projekt koji je razvio Microsoft i isti je uređivač koji pokreće Visual Studio Code (VS Code), jedan od najpopularnijih uređivača koda među programerima.

Za potrebe baze podataka koristi se MySQL. MySQL je open-source sustav za upravljanje relacijskim bazama podataka koji se široko koristi za pohranu i upravljanje strukturiranim podacima. To je jedan od najpopularnijih i široko prihvaćenih sustava baza podataka u svijetu

2.2 Struktura domenski specifične konfiguracije

Specifična domenska konfiguracija ima sljedeću strukturu:

```
Node {
  # comment1
  Sub-Node-1-1 = Value
  Sub-Node-1-2 {
    Sub-Node-2-1 = Value
    # comment2
    Sub-Node-2-2 {
      ...Sub-Node-N {

      }
    }
    Sub-Node-2-3 {
      # comment3
      Value1
    }
    Sub-Node-2-4 {
      Value1, Value2, ...ValueN
    }
    Sub-Node-2-5 {
      Value1,
      Value2, # comment4
      ...ValueN
    }
  }
}
```

Slika 2.1: Struktura specifične domenske konfiguracije

2.3 Baza podataka

Baza podataka kao komponenta web aplikacije strukturirana je i organizirana zbirka podataka koja pohranjuje informacije koje web aplikacija koristi za obavljanje svojih funkcija.
Pruža mogućnosti pohrane, dohvaćanja i upravljanja podacima te zbog toga ima središnju ulogu u omogućavanju web uslugama da obavljaju različite funkcije, odgovaraju na zahtjeve klijenata te pohranjuju i održavaju podatke potrebne za učinkovit rad usluge.
Naša web aplikacija ima MySQL bazu podataka implementiranu u MySQL Workbench softveru.

Baza podataka koju koristimo sastoji se od jedne tablice sljedećeg oblika:

ID(int)	Type (char(255))	Name (char(255))	Active(boolean)	Content(longblob)
1	Type1	Type1Name1	0	sadržaj konf. 1v1
2	Type2	Type2Name1	1	sadržaj konf. 2
3	Type1	Type1Name2	1	sadržaj konf. 1v2

Tablica 2.1: Tablica u bazi podataka

Struktura specifične domenske konfiguracije tada će izgledati ovako:

```
Type1 {
  Type1Name1 {
    Property1 = Value1
    Property2 = Value2
    # comment1
    Property3 {
      Value3
    }
  }
}
```

Slika 2.2: Struktura specifične domenske konfiguracije prema tablici u bazi podataka

3 | Backend

Na backend-u je implementiran REST servis u Spring-u koji se spaja na MySQL bazu podataka te je odgovoran za pružanje funkcionalnosti koja omogućuje rad web aplikacije. Bavi se obradom podataka, pohranom i upravljanjem, kao i obavljanjem raznih zadataka povezanih s funkcionalnošću aplikacije

U ovom završnom radu nećemo se fokusirati na backend stranu web aplikacije, no ukratko ćemo opisati funkcionalnosti koje backend ima kako bi ispunio svoju zadaću.

3.1 CRUD operacije

CRUD (Create, Read, Update, Delete) je akronim koji označava skup temeljnih operacija ili funkcija koje se obično koriste u razvoju baza podataka i web aplikacija za interakciju s podacima u bazi podataka i manipuliranje njima. Ove operacije odgovaraju četirima osnovnim radnjama koje se mogu izvesti nad podacima:

- Create (HTTP POST zahtjev): Operacija za dodavanje novih zapisa podataka ili unosa u bazu podataka. U kontekstu baze podataka, to obično znači umetanje novog retka u tablicu s navedenim vrijednostima podataka.
- Read (HTTP GET zahtjev): Operacija za dohvaćanje podataka iz baze podataka. Ova operacija omogućuje postavljanje upita i pristup podacima kako bi se pregledali ili koristili.
- Update (HTTP PUT zahtjev): Operacija za izmjenu ili ažuriranje postojećih zapisa podataka u bazi podataka. To uključuje promjenu jednog ili više atributa, a da pritom njegov jedinstveni identifikator ostane netaknut.
- Delete (HTTP DELETE zahtjev): Operacija za uklanjanje zapisa podataka iz baze podataka. Nakon izvršenja operacije "Delete", podaci povezani sa zapisom obično se trajno uklanjaju.

3.2 Validacija domenskih konfiguracija

Na backend-u je dodatno definirana funkcionalnost validacije. U svrhu ove funkcionalnosti definirana je posebna gramatika te lekser i parser za specifičnu domensku konfiguraciju. Gramatika, lekser i parser tri su temeljne komponente u

procesu transformacije izvornog koda domenske konfiguracije u strukturirani prikaz koji računalo može razumjeti.

- Gramatika: Formalna specifikacija sintakse ili strukture jezika. Definira pravila i obrasce koji određuju kako se valjane rečenice ili izrazi oblikuju u tom jeziku.
- Lekser: Izvodi tokenizaciju, identificiranje i kategoriziranje elemenata u izvornom kodu u tokene, koji služe kao ulazni podaci za parser.
- Parser: Uzima tokene koje je proizveo lekser i analizira njihovu sintaktičku strukturu prema pravilima definiranim gramatikom.

Ove komponente rade zajedno kako bi osigurale da je izvorni kod domenske konfiguracije sintaktički i semantički valjan.

Gramatika, lekser i parser generirani su pomoću ANTLR4 (ANOther Tool for Language Recognition) alata za generiranje gramatike, leksera i parsera. ANTLR je popularan izbor alata za stvaranje specifičnih domenskih jezika i općenito za izvođenje različitih oblika analize teksta.

```
dsc:    type EOF;

type: WORD '{' typename '}' ;

typename: WORD '{' pair+ '}' ;

pair
    : object
    | array
    | property
    ;

property
    : WORD '=' value+
    | WORD '=' '"' value+ '"'
    ;

object: WORD '{' pair+ '}' ;

array
    :  WORD '{' value (',' value)* '}'
    |  WORD '{' value+ '}'
    ;

value
    :   '"' WORD '"'
    |   '"' NUMBER '"'
    |   '"' 'true' '"'
    |   '"' 'false' '"'
    |   '"' 'null' '"'
    |   WORD
    |   NUMBER
    |   'true'
    |   'false'
    |   'null'
    ;
```

Slika 3.1: Definicija parsera za specifičnu domensku konfiguraciju u ANTLR-u

```
WORD
    : (UPPERCASE | LOWERCASE | DIGIT | ESC)+
    | '"' (UPPERCASE | LOWERCASE | DIGIT | ESC)+ '"'
    ;

fragment LOWERCASE  : [a-z] ;
fragment UPPERCASE  : [A-Z] ;
fragment DIGIT : [0-9] ;

fragment ESC :   '\\' (["\\/bfnrt] | UNICODE) ;
fragment UNICODE : 'u' HEX HEX HEX HEX ;
fragment HEX : [0-9a-fA-F] ;


NUMBER
    :   '-'? INT '.' [0-9]+ EXP? // 1.35, 1.35E-9, 0.3, -4.5
    |   '-'? INT EXP             // 1e10 -3e4
    |   '-'? INT                 // -3, 45
    ;
fragment INT :   '0' | [1-9] [0-9]* ; // no leading zeros
fragment EXP :   [Ee] [+\-]? INT ; // \- since - means "range" inside [...]


COMMENT : '#' ~[\r\n\f]* -> skip ;
WHITESPACE : [ \t\r\n] -> skip;
```

Slika 3.2: Definicija leksera za specifičnu domensku konfiguraciju u ANTLR-u

4 | Frontend

Pokretanjem React aplikacije se pozivaju komponente *BrowserRouter* i *NavBar* iz *index.js* datoteke. *BrowserRouter* je komponenta iz *react-router-dom* biblioteke. Ta se biblioteka koristi za usmjeravanje i navigaciju u React aplikacijama. Komponenta *BrowserRouter* koristi se za omogućavanje usmjeravanja na strani klijenta (eng. client-side routing) u React aplikaciji. Ovaj način usmjeravanja omogućuje promjenu sadržaja prikazanog u pregledniku bez ponovnog učitavanja cijele stranice što pruža glatkije i bolje korisničko iskustvo jer se korisnici mogu kretati između različitih pogleda ili stranica jednostranične aplikacije (eng. single-page application) bez čekanja na odgovore poslužitelja.[2]

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <NavBar/>
      <Routes>
        <Route path='/' element={<App/>}></Route>
        <Route path='DscEdit/:id' element={<DscEditor/>}></Route>
      </Routes>
    </BrowserRouter>
  </React.StrictMode>
);
```

Slika 4.1: *BrowserRouter* React komponenta i njene rute

Definirane su dvije rute. Prva ruta, putanje "/" ("localhost:3000/"), usmjerava na početnu stranicu na kojoj se generira *App* komponenta koja zauzvrat generira *Table* komponentu, tj. tablicu domenskih konfiguracija.

```
function App() {
  return (
    <div className="App">
      <div>
        <Table />
      </div>
    </div>
  );
}
```

Slika 4.2: React komponenta *App*

Druga ruta, putanje "DscEdit/:id" (vidi sliku 4.1) koristi se za usmjeravanje na komponentu *DscEditor* koja služi za uređivanje specifične domenske konfiguracije danog ID-a. Sjetimo se, ID je primarni ključ u bazi podataka te služi kao unikatni identifikator domenske konfiguracije (vidi 2.1).

NavBar komponenta predstavlja navigacijsku traku na vrhu stranice, koja je uvijek vidljiva, neovisno o drugim komponentama koje su prikazane na web pregledniku. Na navigacijskoj traci prikazan je Mathos logo te React komponenta *Link*, koja je također komponenta *react-router-dom* biblioteke i pruža način za stvaranje navigacijske veze unutar React aplikacije. Pritiskom na gumb *Tablice* otvara se putanja "/" ("localhost:3000/"), što nas usmjerava na početnu stranicu. [3]

```
function NavBar() {
  return (
    <nav className="flex flex-row p-3 bg-slate-400
     items-center justify-between rounded-b-lg py-2">
      <section className="flex flex-row items-center justify-between">
        <img src={logo} alt="" className="App-logo w-12 h-12" />
        <Link
          to={"/"}
          className="text-black no-underline rounded-full
           bg-slate-500 text-lg px-2 ml-3 hover:scale-105 duration-150"
        >
          Tablice <FontAwesomeIcon icon={faTable} />
        </Link>
      </section>
      <section className="text-2xl">Domain Configuration Editor</section>
    </nav>
  );
}
export default NavBar;
```

Slika 4.3: *NavBar* komponenta

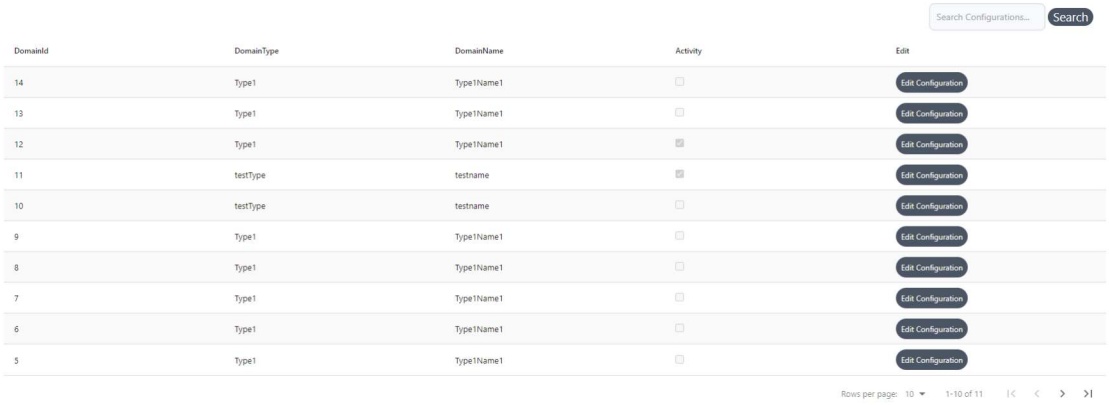
4.1 Pojmovi

U cilju boljeg shvaćanja načina rada React frontend dijela web aplikacije definirat ćemo sljedeće pojmove.

- React stanje (*eng. state*) : JavaScript objekt koji sadrži podatke koji se mogu mijenjati tijekom vremena i utjecati na ponašanje komponente. State je temeljni koncept u Reactu i koristi se za upravljanje i praćenje informacija unutar komponente. Komponente Reacta mogu imati vlastito stanje, a kada se stanje komponente promijeni, React automatski ponovno renderira komponentu kako bi prikazao ažurirano stanje. [4]
- React hook : Omogućuje funkcionalnim komponentama upravljanje stanjem, nuspojavama i drugim React značajkama. Hookovi pružaju način da se "priključite" Reactovim značajkama stanja i životnog ciklusa bez potrebe za pisanjem klasnih komponenta. [5]
- useState : Ovaj hook omogućuje funkcionalnim komponentama upravljanje stanjem lokalne komponente. Obično se koristi za upravljanje podacima koji se mogu mijenjati tijekom vremena unutar komponente.[5]
- useEffect : Ovaj hook omogućuje funkcionalnim komponentama izvođenje nuspojava, kao što je dohvaćanje podataka, manipulacija DOM-om ili pretplata na događaje. [5]

4.2 Tablica

Tablica prikazuje sve specifične domenske konfiguracije koje se nalaze u bazi podataka. Stupce čine atributi *Id*, *Type*, *Name* i *Active* specifične domenske konfiguracije (vidi 2.1) te *EditConfiguration* gumb koji preko komponente *Link* usmjerava na putanju "DscEdit/id" koja služi za uređivanje odabrane domenske konfiguracije. Tablica ima funkcionalnost sortiranja, paginacije te filtriranja korištenjem trake za pretraživanje.



Slika 4.4: Tablica domenskih konfiguracija u web pregledniku

Table komponenta predstavlja glavnu tablicu u kojoj su prikazane domenske konfiguracije. Za kreiranje tablice korištena je već postojeća *DataTable* komponenta iz *react-data-table-component* biblioteke. *DataTable* komponenta ima ugrađeno sortiranje i paginaciju, a definirana je mogućnost filtriranja korištenjem trake za pretraživanje.

4.2.1 Dohvaćanje domenskih konfiguracija iz baze podataka

Funkcija *handleFetchAllDomains* šalje HTTP GET zahtjev "api/v1/configurations" na backend endpoint koji pokreće upit nad bazom podataka i vraća odgovor frontendu. Odgovor sadrži *Id* , *Type*, *Name* i *Active* atribute svake domenske kofiguracije u bazi podataka (vidi 2.1). Ti se podaci spremaju u React state *fetchedData*. Funkcija *handleFetchAllDomains* poziva se na useEffect hook-u, tj. podaci se spremaju u *fetchedData* state pri učitavanju stranice.

```
//FETCHING CONFIGURATIONS
const handleFetchAllDomains = async () => {
  fetch("api/v1/configurations?pageSize=100")
    .then((res) => res.json())
    .then((data) => {
      setFetchedData(data.content);
    });
};

//FETCHING ALL THE DOMAINS ON MOUNTING
useEffect(() => {
  handleFetchAllDomains();
}, []);
```

Slika 4.5: *handleFetchAllDomains* funkcija zadužena za dohvaćanje domenskih konfiguracija

4.2.2 Prikaz podataka u tablici

Kao što je ranije spomenuto, za tablicu korištena je *DataTable* React komponenta [6] koja je definirana na sljedeći način:

```
return (
  <DataTable
    title="Domain Configurations"
    columns={columns}
    data={filteredItems}
    defaultSortField="DomainId"
    striped
    pagination
    subHeader
    subHeaderComponent={subHeaderComponent}
  />
);
```

Slika 4.6: *DataTable* React komponenta

Komponenti *DataTable* pridruženi su atributi [8]:

- *title*: naslov prikazan u zaglavlju tablice
- *columns*: polje koji predstavlja stupce tablice
- *data*: polje u kojem su podaci koji će se ispisati u tablici
- *defaultSortField*: postavlja stupac *DomainId* kao zadan stupac za sortiranje
- *pagination*: omogućava paginaciju
- *subHeader*: prikazuje sekundarno zaglavlje između tablice i zaglavlja tablice
- *subHeaderComponent*: komponenta koja se prikazuje kao sekundarno zaglavlje, u našem slučaju to je traka za pretraživanje

Polje *columns* [7] definirano je na sljedeći način :

- *name*: ime stupca
- *selector*: funkcija potrebna za prikaz podataka
- *sortable*: atribut zadužen za sortibilnost
- *cell*: omogućuje zamjenu ćelije tablice vlastitom prilagođenom komponentom, u našem slučaju to je *editButton* funkcija koja sadrži React *Link* koji usmjerava na stranicu za uređivanje odabrane domenske konfiguracije

```
//COLUMNS FOR TABLE
const columns = [
  {
    name: "DomainId",
    selector: (row) => row.id,
    sortable: true,
  },
  {
    name: "DomainType",
    selector: (row) => row.type,
    sortable: true,
  },
  {
    name: "DomainName",
    selector: (row) => row.name,
    sortable: true,
  },
  {
    name: "Activity",
    selector: (row) => {
      return (
        <label>
          <input
            type="checkbox"
            disabled="disabled"
            defaultChecked={row.active}
          />
        </label>
      );
    },
  },
  {
    name: "Edit",
    cell: (row) => editButton(row.id, row.type, row.name, row.active),
  },
];
```

Slika 4.7: Polje *columns*

```
//EDIT BUTTON FOR SENDING INFO TO MONACO
const editButton = (id, domainType, domainName, activity) => (
  <Link
    to={"DscEdit/" + id}
    state={{
      Id: id,
      DomainType: domainType,
      DomainName: domainName,
      Activity: activity,
    }}
  >
    <button className=" p-2  bg-gray-600 rounded-full text-xs text-white hover:scale-105 duration-75 ">
      Edit Configuration
    </button>
  </Link>
);
```

Slika 4.8: *editButton* funkcija

Polju *data* mogli smo jednostavno pridružiti React state *fetchedData* i tablica bi bila ispunjena svim potrebnim podacima domenskih konfiguracija. Međutim, kako bismo uveli funkcionalnost filtriranja korištenjem trake za pretraživanje polju *data* pridružen je React state *filteredItems*, drugim riječima React state *filteredItems* čine podaci state-a *fetchedData* koji su prošli kroz proces filtriranja korištenjem trake za pretraživanje. U svrhu ovoga definirana je nova React komponenta *FilterComponent*.

```
const FilterComponent = ({ filterText, onFilter, onClear }) => (
  <>
    <input
    className="bg-gray-50 border border-gray-300 text-gray-900
     text-sm rounded-lg focus:ring-blue-500 focus:border-blue-500 block w-200 p-2.5 "
      id="search"
      type="text"
      placeholder="Search Configurations..."
      value={filterText}
      onChange={onFilter}
    />
    <button className='px-2 mx-1  bg-gray-600 rounded-full text-[2vh]
     text-white hover:scale-105 duration-150' onClick={onClear}>Search</button>
  </>
);
```

Slika 4.9: React komponenta *FilterComponent*

Za filtriranje podataka koristimo *useMemo* React hook [9] koji omogućuje spremanje rezultata kalkulacija između renderiranja stranice. Drugim riječima, ponaša se kao predmemorija (*eng. cache*) kako bismo spremili rezultate filtriranja podataka u React state *filterText*.

```
//CREATING FILTER COMPONENT ON TOP AND HANDLING PAGINATION TOGGLE
const subHeaderComponent = useMemo(() => {
  const handleClear = () => {
    if (filterText) {
      setResetPaginationToggle(!resetPaginationToggle);
      setFilterText("");
    }
  };
  return (
    <FilterComponent
      onFilter={(e) => setFilterText(e.target.value)}
      onClear={handleClear}
      filterText={filterText}
    />
  );
}, [filterText, resetPaginationToggle]);

//FILTERING DATA
const filteredItems = fetchedData.filter(
  (item) =>
    JSON.stringify(item).toLowerCase().indexOf(filterText.toLowerCase()) !==
    -1
);
```

Slika 4.10: Filtriranje podataka

4.3 Uređivanje domenskih konfiguracija

U svrhu uređivanja domenskih konfiguracija koristi se Monaco uređivač koda kao React komponenta. Monaco uređivač koda najpoznatiji je po tome što pokreće VS Code uređivač koda. Monaco ćemo proširiti IntelliSense mogućnostima označavanje sintakse (eng. syntax highlighter), validacije, dovršavanja/predlaganja koda (eng. code completion) za našu specifičnu domensku konfiguraciju.

Kao što smo ranije spomenuli, domensku konfiguraciju možemo uređivati pritiskom na gumb *Edit Configuration* željene domenske konfiguracije u tablici domenskih konfiguracija (vidi sliku 4.4). Pritiskom na *Edit Configuration* pokreće se *editButton* funkcija (vidi sliku 4.8) koja koristeći React *Link* usmjerava na putanju "DscEdit/" te konkatenira *Id* odabrane domenske konfiguracije. Primjerice, ako pritisnemo *Edit Configuration* gumb domenske konfiguracije Id-a 6, usmjereni smo na putanju "localhost:3000/DscEdit/6". Ova ruta React komponente *BrowserRouter* generira *DscEditor* komponentu (vidi sliku 4.1).

```
# comment
Type1 { # comment
    # comment
    Type1Name1 { # comment
        Property1 = Value1 # comment
        Property2 = Value2
        # comment
        Property3 { # comment
            Value3 # comment
        } # coment
        Property4 { # comment
            Property41 = Value41 # comment
            # comment
            Property42 = Value42
            # comment
            Property43 {
                Value431, Value432, Value433, Value434 # comment
            }
            Property44 = Value44
            Property45 {
                Value451
            }
            Property46 { Value461 } # coment
            Property47 { Value471, Value472 } # coment
            Property48 { Value481 Value482 Value483 } # coment
        }
        Property5 {
            Property51 {
                Property511 { Value5111 Value5112 Value5113 }
                Property512 = Value512
                Property513 {
                    Value5131, Value5132
                }
            }
            Property52 { Value521 }
            Property53 { Value531, Value532 }
            Property54 { Value541 Value542 Value543 }
        }
        Property6 = Value6
```

Slika 4.11: Prikaz web preglednika na ruti "localhost:3000/DscEdit/6", tj. prikaz sadržaja domenske konfiguracije s ID-jem 6 u Monaco uređivaču koda

4.3.1 Učitavanje sadržaja domenske konfiguracije

Sadržaj domenske konfiguracije spremljen je u React state-u *dscContent*. Funkcija *getConfiguration* šalje HTTP GET zahtjev "/api/v1/configurations/Id" na backend endpoint koji pokreće upit nad bazom podataka i vraća odgovor frontend-u. Odgovor sadrži *Content* atribut domenske konfiguracije odabranog ID-ja, tj. njen sadržaj (vidi 2.1). Pošto je sadržaj domenske konfiguracije u bazi podataka tipa *longblob*, funkcija *getConfiguration* ujedno i pretvara *longblob* tip podatka u tekst koji se može prikazati u uređivaču.

```
useEffect(() => {
  getConfiguration();
},[]);

const getConfiguration = () => {
  axios
    .get(`/api/v1/configurations/${Id}`)
    .then((res) => {
      // Base64 string
      var content = res.data.content;

      // Text from base64
      var text = decodeURIComponent(window.atob(content));
      setDscContent(text);
    })
    .catch((err) => console.log(err.toJSON()));
};
```

Slika 4.12: *getConfiguration* funkcija

4.3.2 Monaco uređivač koda

Monaco uređivač koda uvest ćemo kao React komponentu [10]. Monaco komponenta definirana je na sljedeći način:

```
<Editor
  height="80vh"
  defaultLanguage="dscLang"
  defaultValue={dscContent}
  value={dscContent}
  theme="dscLang-theme"
  onChange={(value) => {
    setDscContent(value);
  }}
  beforeMount={handleEditorWillMount}
  onMount={handleEditorDidMount}
/>
```

Slika 4.13: Monaco React komponenta

- *height*: visina uređivača koda
- *defaultLanguage*: zadani jezik trenutnog modela
- *defaultValue*: zadana vrijednost trenutnog modela
- *value*: vrijednost trenutnog modela
- *theme*: tema za izgled Monaco uređivača koda
- *onChange*: događaj koji se izvodi kada se promijeni sadržaj trenutnog modela
- *beforeMount*: događaj koji se izvodi prije učitavanja uređivača
- *onMount*: događaj koji se izvodi u trenutku kada se model učitao

Zadani jezik *dscLang* definiran je na sljedeći način:

```
monaco.languages.register({ id: "dscLang" });
monaco.languages.setMonarchTokensProvider("dscLang", {
  bracketPairColorization: true,
  tokenizer: {
    root: [
      [/#.*/, "comment"],
      [/".*?"/, "string"],
      [/(\S+\s)(=)/, ["naming", "eq"]],
      [/(\S+\s)({)/, ["naming", "brackets"]],
    ],
  },
});

//zagrade
const config = {
  brackets: [
    ["{", "}"],
    ["(", ")"],
    ["[", "]"],
  ],
  surroundingPairs: [
    { open: "{", close: "}" },
    { open: "[", close: "]" },
    { open: "(", close: ")" },
    { open: "<", close: ">" },
    { open: "'", close: "'" },
    { open: '"', close: '"' },
  ],
  autoClosingPairs: [
    { open: "{", close: "}" },
    { open: "[", close: "]" },
    { open: "(", close: ")" },
    { open: "'", close: "'", notIn: ["string", "comment"] },
    { open: '"', close: '"', notIn: ["string", "comment"] },
  ],
};
monaco.languages.setLanguageConfiguration("dscLang", config);
```

Slika 4.14: *dscLang* jezik domenskih konfiguracija

U okviru definiranja jezika također je definirano dovršavanje koda u smislu automatskog zatvaranja zagrada, navodnika itd.

Tema za izgled Monaco uređivača *dscLang-theme* definirana je na sljedeći način:

```
monaco.editor.defineTheme("dscLang-theme", {
  base: "vs-dark",
  inherit: true,
  folding: true,
  rules: [
    { token: "string", foreground: "#F87475" },
    { token: "comment", foreground: "#AA852E", fontStyle: "italic" },
    { token: "naming", foreground: "#AD4A81" },
  ],
  colors: {
    "editor.background": "#292929",
    "editorCursor.foreground": "#B8B8B8",
    "editor.lineHighlightBackground": "#383838",
    "editorLineNumber.foreground": "#585858",
    "editor.selectionBackground": "#393939",
    "editor.inactiveSelectionBackground": "#717171",
  },
});
```

Slika 4.15: *dscLang-theme* tema za izgled Monaco uređivača

Jezik i temu potrebno je definirati prije učitavanja Monaco uređivača, stoga definicije jezika (vidi sliku 4.14) i temu (vidi sliku 4.15) enkapsuliramo u *handleEditorWillMount* funkciju koja se pridružuje *beforeMount* događaju Monaco uređivača (vidi sliku 4.13).

Zadana početna vrijednost Monaco uređivača jest sadržaj odabrane domenske konfiguracije, tj. React state *dscContent* pridružen je *defaultValue* svojstvu Monaco uređivača (vidi sliku 4.13). React *useEffect* hook pokreće *getConfiguration* funkciju. Time se *getConfiguration* pokreće pri otvaranju web stranice te se tada u React state-u *dscContent* sprema sadržaj domenske konfiguracije (vidi sliku 4.12) kojeg Monaco uređivač može iskoristiti kao zadanu vrijednost za prikaz.

Nakon svake promjene domenske konfiguracije u uređivaču novi se sadržaj sprema u React state *dscContent*. Ovaj proces obavlja ugrađena funkcionalnost Monaco uređivača *onChange* (vidi sliku 4.13)

4.3.3 Validacija domenske konfiguracije

Pritiskom na gumb "Validate Configuration!" u Monaco uređivaču poziva se funkcija *validateConfiguration* koja šalje HTTP POST zahtjev "/api/v1/configurations/validate" na backend endpoint, on pokreće validaciju sadržaja domenske konfiguracije i vraća odgovor frontend dijelu. Odgovor koji backend vraća objasnit ćemo na sljedećem primjeru. Recimo da je sadržaj domenske konfiguracije sljedeći:

```
# comment
THIS IS WRONG
Type1 { # comment
    # comment
    Type1Name1 { # comment
        Property1 = Value1 # comment
        Property2 = Value2
        # comment
        Property3 { # comment
            Value3 # comment
        } # coment
        Property4 { # comment
            Property41 = Value41 # comment
            # comment
            Property42 = Value42
            # comment
            Property43 {
                Value431, Value432, Value433, Value434 # comment
            }
            Property44 = Value44
            Property45 {
                Value451
            }
            Property46 { Value461 } # coment
            Property47 { Value471, Value472 } # coment
            Property48 { Value481 Value482 Value483 } # coment
        }
    } # comment
} # comment
# comment
```

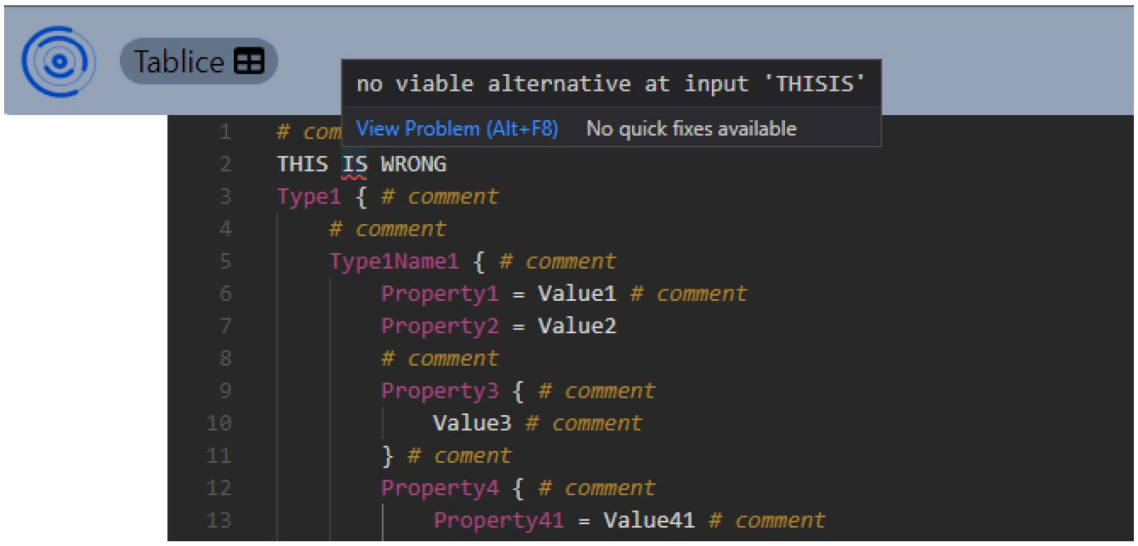
Slika 4.16: Sadržaj domenske konfiguracije

Prisjetimo se kako struktura domenske konfiguracije treba izgledati (vidi sliku 2.2). Uočimo da "THIS IS WRONG" predstavlja krivo napisani dio domenske konfiguracije. Odgovor koji će frontend dio primiti od backend-a nakon validacije izgleda ovako:

```
[
    {
        "hint": "IS",
        "line": 2,
        "charPositionInLine": 5,
        "message": "no viable alternative at input 'THISIS'",
        "isValid": false
    }
]
```


Slika 4.17: Odgovor koji frontend prima od backend dijela nakon obavljene validacije

Odgovor sadrži informaciju koja govori je li domenska konfiguracija važeća, te ako nije, vraća atribute *hint*, *line*, *charPositionInLine* i *message*. Ovi atributi nose informaciju gdje se nalazi neispravan dio koda kako bi se taj dio mogao iscrtati u uređivaču. Na ovaj način je implementirano prepoznavanje greške i njeno označavanje.



Slika 4.18: Označavanje pogreške u Monaco uređivaču

Funkcija *validateConfiguration* implementirana je na sljedeći način:

```
const validateConfiguration = () => {
  // from text to base64
  const base64content = btoa(dscContent);
  // from base 64 to longblob
  var file = b64toBlob(base64content, "application/octet-stream");

  var formData = new FormData();
  formData.append("file", file, file.name);

  axios
    .post("/api/v1/configurations/validate", formData)
    .then((res) => {
      if (res.data.length > 0) {
        setValidateMessage("Domain Configuration is NOT valid!");
        res.data.forEach((i) => {
          markers.push({
            startLineNumber: i.line,
            endLineNumber: i.line,
            startColumn: i.charPositionInLine + 1,
            endColumn: i.charPositionInLine + i.hint.length + 1,
            message: i.message,
            severity: monaco.MarkerSeverity.Error,
          });
        });
      } else {
        setValidateMessage("Domain Configuration is valid!");
      }
      monaco.editor.setModelMarkers(
        editorRef.current.getModel(),
        "owner",
        markers
      );
    })
    .catch((err) => console.log(err));
};
```

Slika 4.19: *validateConfiguration* funkcija

Prije nego što se pošalje HTTP POST zahtjev na backend, sadržaj domenske konfiguracije se mora pretvoriti iz teksta u tip *longblob* koji se koristi u bazi podataka (vidi 2.1). U tu svrhu koristimo ugrađenu funkciju *btoa* koja pretvara tekst u *base64* tip podatka te definiramo funkciju *b64toBlob* koja pretvara *base64* u *longblob* tip podatka.

```
const b64toBlob = (b64Data, contentType, sliceSize) => {
  contentType = contentType || "";
  sliceSize = sliceSize || 512;

  var byteCharacters = atob(b64Data);
  var byteArrays = [];

  for (var offset = 0; offset < byteCharacters.length; offset += sliceSize) {
    var slice = byteCharacters.slice(offset, offset + sliceSize);

    var byteNumbers = new Array(slice.length);
    for (var i = 0; i < slice.length; i++) {
      byteNumbers[i] = slice.charCodeAt(i);
    }

    var byteArray = new Uint8Array(byteNumbers);

    byteArrays.push(byteArray);
  }

  console.log(byteArrays);

  return new File(byteArrays, makeid(20), { type: contentType });
};

// Get random string
const makeid = (length) => {
  var result = "";
  var characters =
    "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
  var charactersLength = characters.length;
  for (var i = 0; i < length; i++) {
    result += characters.charAt(Math.floor(Math.random() * charactersLength));
  }
  return result;
};
```

Slika 4.20: *b64toBlob* funkcija

4.3.4 Spremanje domenske konfiguracije

Pritiskom na pritiskom na gumb "Save Configuration" (vidi sliku 4.11) poziva se funkcija *saveConfiguration* koja šalje HTTP POST zahtjev "/api/v1/configurations/" na backend endpoint, koji sprema sadržaj domenske konfiguracije u bazu podataka. Na backend dijelu implementirana je validacija pri svakom spremanju domenske konfiguracije, tako da je nemoguće spremiti neispravnu domensku konfiguraciju.

```
const saveConfiguration = () => {
  // from text to base64
  const base64content = btoa(dscContent);
  // from base 64 to longblob
  var file = b64toBlob(base64content, "application/octet-stream");

  var formData = new FormData();
  formData.append("file", file, file.name);
  formData.append("type", DomainType);
  formData.append("name", DomainName);
  formData.append("active", active);

  axios
    .post("/api/v1/configurations/", formData)
    .then((res) => {
      setSaveMessage("Domain Configuration saved successfully!");
    })
    .catch((err) => {
      console.log(err);
      setSaveMessage("Domain Configuratin saved unsuccessfully!");
    });
};
```

Slika 4.21: *saveConfiguration* funkcija

U slučaju bilo kakve promijene sadržaja odabrane domenske konfiguracije, stvara se nova domenska konfiguracija s novim jedinstvenim identifikatorom (Id-jem), tj. stvara se novi redak u tablici baze podataka. (vidi 2.1)

Tijekom uređivanja domenske konfiguracije također je moguće pritisnuti tipku "Activate Configuration!" (vidi 4.11) u kojem slučaju se trenutnoj domenskoj konfiguraciji atribut *Active* (vidi 2.1) mijenja iz *false* u *true*.

5 | Zaključak

Tema ovog završnog rada bila je razvoj web aplikacije za upravljanje domenskim konfiguracijama s IntelliSense mogućnostima. Domenska konfiguracija je specifična struktura podatka prilagođena jedinstvenim slučajima upotrebe unutar tvrtke ENEA. Web aplikacija omogućava pregled, izmjenu, dodavanje, uklanjanje i validaciju domenskih konfiguracija pohranjenih u bazi podataka. Kako bi se validirale domenske konfiguracije i uspješno primjenile IntelliSense mogućnosti definiran je domenski specifičan jezik pomoću ANTLR alata za generiranje gramatike, leksera i parsera. Web aplikacija izrađena je koristeći Spring framework kao backend, dok se za frontend koristila React biblioteka. Backend je zadužen za upravljanje podacima kao što su npr. komunikacija s bazom podataka, validacija podataka, dostavljanje i primanje podataka od strane frontend-a i sl. Za potrebe uređivanja domenskih konfiguracija koristi se Monaco uređivač koda kao React komponenta. Uređivač koda proširen je IntelliSense mogućnostima označavanja sintakse i dovršavanja/prijedloga koda.
Web aplikacija dostupna je na GitHub-u [11], zajedno s kratkim videom koji pokazuje osnovne funkcionalnosti. [12]

Literatura

[1] \https://www.tutorialspoint.com/spring/spring_overview.htm

[2] \https://reactrouter.com/en/main/start/overview

[3] \https://reactrouter.com/en/main/components/link

[4] \https://www.geeksforgeeks.org/reactjs-state/

[5] \https://legacy.reactjs.org/docs/hooks-overview.html

[6] https://react-data-table-component.netlify.app/?path=/docs/getting-started-intro--page

[7] \https://react-data-table-component.netlify.app/?path=/docs/api-columns--page

[8] \https://react-data-table-component.netlify.app/?path=/docs/api-props--page

[9] \https://react.dev/reference/react/useMemo

[10] \https://www.npmjs.com/package/@monaco-editor/react#usage

[11] \https://github.com/jurajtodic/DSC-Editor

[12] \https://rawcdn.githack.com/jurajtodic/DSC-Editor/a5d5f81f8261bd2ef51b619be725cf8bd14993db/demo_video.mp4