GENERIRANJE BACKEND DOKUMENTACIJE

Marin, Borna

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERAU OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

GENERIRANJE BACKEND DOKUMENTACIJE

Završni rad

Borna Marin

Osijek, 2021.

SADRŽAJ

1. UVOD

Cilj ovog završnog rada je izrada internet aplikacije kao pomoć razvojnim inženjerima tijekom izrade internet aplikacije. Predstavljena je konkretna internet aplikacija koja automatizirano pravi dokumentaciju između korisničkog i serverskog sučelja za neku općenitu internet aplikaciju koja je u procesu izrade. Dokumentiranje korisničkog i serverskog sučelja je od velike važnosti jer se korisnička i serverska strana aplikacije obično izrađuje odvojeno, odnosno izrađuju ih dva programera zasebno. Predstavljena aplikacija poboljšava komunikaciju između programera korisničkog i serverskog sučelja, a time pruža uštedu u vremenu. Komunikacija između programera je vrlo bitna tijekom izrade internet aplikacije, razvijanja njezinih značajki te rješavanja nastalih problema u tom procesu. S obzirom da se na komunikaciju odvaja značajan dio vremena, dokumentiranje korisničkog i serverskog sučelja je od velike važnosti. Tijekom razvoja internet aplikacije, čest je problem promijena osobe koje izrađuje određenu internet aplikaciju. U tom slučaju, drugoj osobi dolazi do poteškoća u shvaćanju napisanog programa, jer program nije lako čitljiv. S druge strane, jasan i čitljiv program ostvaruje se detaljnim komentiranjem i opisivanjem svih njegovih dijelova. Pored toga, bitno je jednostavno i brzo navigiranje kroz dokumentaciju koristeći moderne tehnologije. Za tu svrhu korišten je NuxContent kao jedna od modernih okolina za izradu i dokumentiranje internet aplikacija. Njegovu implementacija podržava Tailwind Css, Nuxt i Vue. U radu su navedene prednosti i nedostaci u odnosu na slične i/ili starije tehnologije. Nadalje, u radu se pobliže objašnjava pojedina tehnologija te su također prikazani i najbolji načini korištenja s primjerima programskog koda. Drugo poglavlje donosi prikaz korištenih tehnologija, a kroz njegova potpoglavlja prikazane su Vue i Nuxt razvojne okoline, Nuxt Content, Javascript i Tailwind Css. U trećem poglavlju je prikazan proces izrade aplikacije uz detaljnija pojašnjenja pristupa problemu i načina rješavanja istih. Na samom kraju rada dan je zaključak sveukupnog procesa i tematike problema ovoga rada.

1.1. Zadatak završnog rada

Zadatak ovog završnog rada je izraditi internet aplikaciju za automatizirano dokumentiranje kao pomoć razvojnim inženjerima. Aplikacija treba na jasan i učinkovit način automatizirano dokumentirati sučelje između korisničke i serverske strane internet aplikacije. Potrebno je opisati postupak izrade opisane aplikacije, kao i njenu funkcionalnost. Aplikaciju je potrebno usporediti s već postojećim i sličnim rješenjima.

2. KORIŠTENE TEHNOLOGIJE

U ovom poglavlju opisuju se tehnologije korištene pri izradi aplikacije za potrebe ovog završnog rada. Pobliže objašnjavamo Vue i Nuxt okolinu, Nuxt Content proširenje, Tailwind css i neke moderne alate Javascripta.

2.1. Vue i Nuxt okolina

Vue.js je sučelje tehnologija koju koristimo za stvaranje internetskih aplikacija i stranica. Dva glavna konkurenta su joj React.js i Angular.js od Facebooka i Googla. Vue je nastao nakon oba konkuretna te iz tog razloga je imao mogućnost pokupiti korisne značajke obje okoline [1].

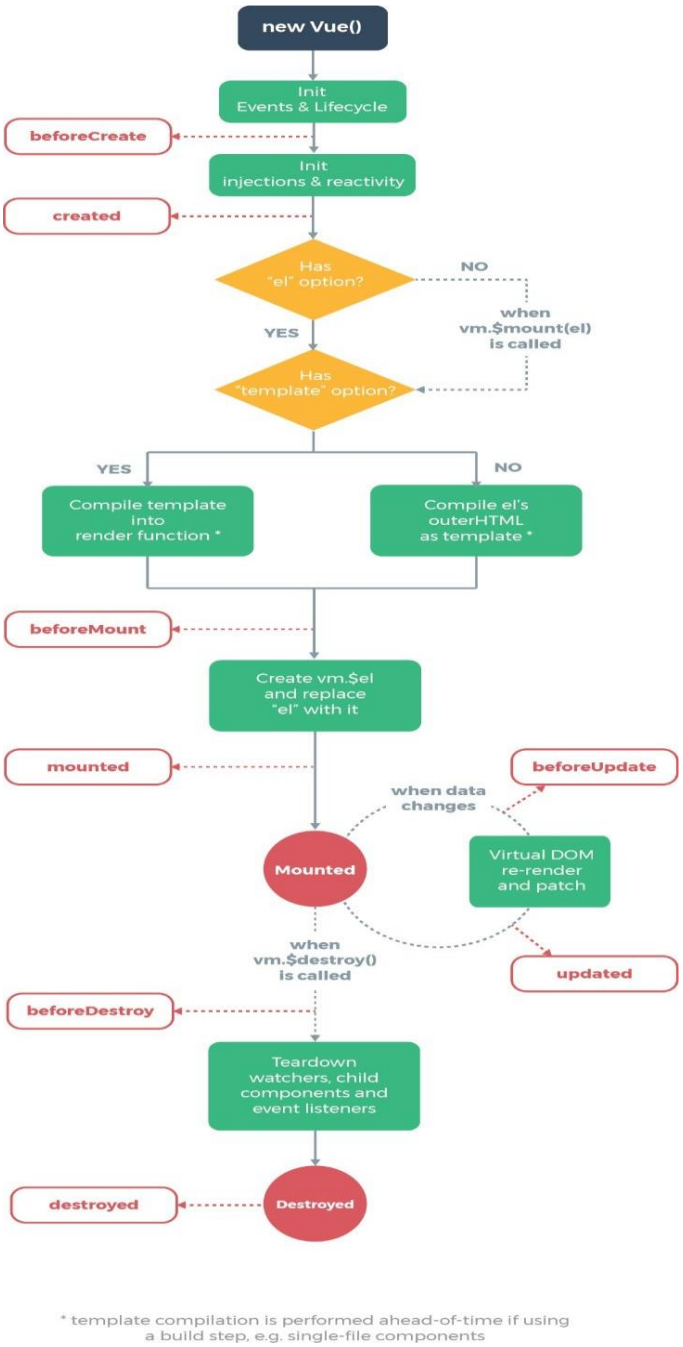
Osnovna ideja razvoja Vue-a je postići dobre rezultate uz što manje napora, tako da korisnik može stvarati kompleksne komponente uz samo nekoliko linija programa. Vue.js je također izvrstan za rad s komponentama jer mu je potrebno relativno malo dodatnih troškova jer komponente s jednom datotekom mogu pohraniti sve programske i Markup jezike kao što su HTML, CSS i JavaScript u jednu datoteku. Naravno u praksi se ulaže ogromna količina vremena u razvijanje arhitekture komponenta zbog glomaznosti projekata, ali u svojoj srži vue teži jednostavnosti [1].

Vue.js izvrstan je alat za programere jer su sve njegove funkcije lako dostupne. Radi lakšeg korištenja, programeri mogu lako imenovati funkciju kako im se sviđa. Svaki segment može imati zasebne funkcije, što olakšava prilagodbu aplikacije prema individualnim zahtjevima [1].

Nuxt je frontend okolina izgrađena oko Vue-a koji nudi sjajne razvojne značajke kao što su iscrtavanje na strani poslužitelja, automatski generirane rute, poboljšano upravljanje meta oznakama i poboljšanje za SEO [2].

Jedan od bitnijih featura Nuxt-a je renderanje na serveru (engl. Server side rendering) i životni ciklus (engl. Lifecycle) koji se poziva dolaskom na novu rutu. Na slici 2.1. u nastavku je prikazan Vue životni ciklus [2].

Prvo nam se kreira vue instanca koja poziva početak životnog ciklusa. U crvenim oblačićima vidimo životne cikluse koji zapravo predstavljaju metode gdje mi kao programeri možemo ubaciti svoju logiku i na taj način manipulirati kreiranjem Nuxt instance. Dio se obavlja na serveru, već spomenuto iscrtavanje na poslužitelju, a dio na klijentu. Destroyed (uništavanje) je metoda koja se obavlja tek nakon odlaska sa stranice. [2]



Slika 2.1. Nuxt/Vue životni ciklus

2.2. Nuxt Content

Nuxt content je dodatni Npm module aplikacije. Datoteke prilažemo u direktorij te iz njega dohvaćamo Markdown, JSON, YAML, XML ili CSV datoteke preko ugrađenog api-a.API je sličan kao MongoDB, ponaša se kao GIT-based Headless CMS [2].

Neke od prednosti [2]:

- Brzina učitavanja projekta u developmentu
- Vue komponente napisane u Markdown-u
- Integriran search
- Podržano generiranje statičnih stranica
- QueryBuilder Api

Instalira se kao npm ili yarn paket. Dodajemo ga u nuxt.config.js datoteku (konfiguracijska datoteka). U direktoriju imamo JSON datoteku koja sadrži sve podatke potrebne za internet aplikaciju. Ti podaci mogu biti lokalni ili dohvaćeni sa servera, također možemo ih spojiti sa Socket tehnologijom kako bi imali razmjenu podataka servera i klijenta u stvarnome vremenu [2].

Sami moduli nam globalno ubrizgaju sadržajnu instancu, što znači da joj možemo pristupiti bilo gdje u projektu koristeći taj sadržaj. Ako instanci želimo pristupiti u životnim ciklusima projekta koji se obavljaju na serveru ili u direktoriju za proširenja koristimo context (sadržaj.) Sadržaj kao parametar prima putanju do datoteke iz koje čitamo podatke npr. $content('articles', params.slug) -> /articles/${params.slug}. Nakon toga sve metode slažemo jednu iza druge ovisno na koji način želimo sortirati i filtrirati podatke. Zadnja metoda u nizu je uvijek ona koja zapravo odrađuje dohvaćanje [2].

```
const [prev, next] = await this.$content('articles')
      .only(['title', 'path'])
      .sortBy('date')
      .where({ isArchived: false })
      .surround('article-2'

     .fetch()
```

Slika 2.2. Asinkrono dohvaćanje podataka

Renderiranje podataka - možemo koristiti Nuxt Content komponentu ili sami napraviti komponentu koja iscrtava dohvaćene podatke. Slika 2.3 prikazuje manualno integriranu komponentu koristeći tehnologije kao Tailwind Css i plain Html [3].

```
<template>
  <article>
    <h1>{{ page.title }}</h1>
    <nuxt-content :document="page" />
  </article>
</template>
```

Slika 2.3. Nuxt content integracija u Html

2.3. Javascript

Javascript je trenutno najzastupljeniji jezik za kreiranje internet aplikacija. Trenutno ga koriste tri vodeće okoline za razvoj internet aplikacija i stranica, a to su Vue, React i Angular. Prije nego se objasni način dohvaćanja podataka moraju se pojasniti dva alata Javascript jezika koja koristimo u projektu. A to su Obećanja (engl. Promises) i Async Await.

Obećanja (engl. Promises) su korišteni kada se piše asinkroni programski kod u Javascriptu. Prije se za manipuliranje većeg broja asinkronih operacija koristio povratne pozive, ali oni stvaraju pomutnju u programu koju je teško održavati i zapravo često vodi do neodrživog programa. Obećanja na jednostavan način rješavaju problem te omogućuju bolji pristup pogreškama [3].

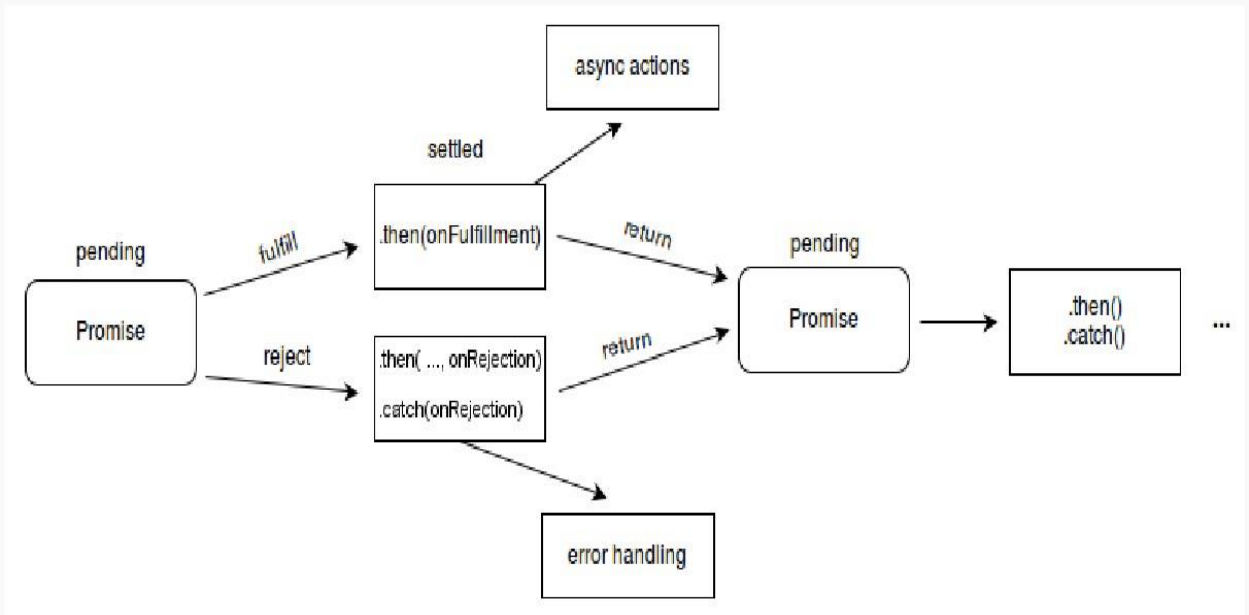
Obećanja su zamjena za vrijednost koja nije nužno poznata pri stvaranju obećanja. Omogućuje povezivanje rukovatelja s vrijednošću uspjeha asinkrone akcije ili razlogom neuspjeha. To omogućuje asinkronim metodama da vraćaju vrijednosti poput sinkronih metoda: umjesto da odmah vrate konačnu vrijednost, asinkrona metoda vraća obećanje da će vrijednost dati u nekom trenutku u budućnosti [3]. Na slici 2.4. u nastavku prikazano je kroz shemu kako funkcionira izvršavanje obećanja.

Prednosti korištenja Obećanja:

1. Čitljiviji programski kod
2. Lakše manipuliranje većim brojem asinkronih operacija
3. Bolja tečnost i kontrola asinkrone logike
4. Bolje rukovanje pogreškama

Obećanja imaju 4 stanja:

1. Ispunjeno: Radnja u vezi s obećanjem uspjela
2. Odbijeno: Radnja u vezi s obećanjem nije uspjela
3. Na čekanju: Obećanje se još čeka, tj. još nije ispunjeno ili odbijeno
4. Riješeno: Obećanje je ispunjeno ili odbijeno



Slika 2.4. Izvršavanje obećanja

Funkcija na slici 2.5 prikazuje jednostavan način korištenja obećanja. Većinom se koriste za operacije koje traju neko određeno vrijeme npr. koriste se za API pozive. Bitno je da se zna da pozivanje metode myPromise.then izvršava tek nakon što su obećanja pridružen myPromise varijabli.

```
let myPromise = new Promise(function(myResolve, myReject) {
  let x = 0;
  if (x == 0) {
     myResolve("OK");
  } else {
     myReject("Error");
  }
});
myPromise.then(
 function(value) {myDisplayer(value);},
 function(error) {myDisplayer(error);});
```

Slika 2.5. Jednostavan primjer obećanja poziva

Asinkroniziranje/čekanje - asinkrona funkcija je funkcija deklarirana s ključnom riječi asink, a ključna riječ čekanje dopuštena je unutar njih. Ključne riječi asink i čekanje omogućuju pisanje asinkronog ponašanja zasnovanog na obećanjima u čišćem stilu, izbjegavajući potrebu za izričitim konfiguriranjem lanca obećanja [3].

Asinkrone funkcije mogu sadržavati nula ili više izraza čekanja. Izrazi čekanja(eng. Await) čine da se funkcije koje vraćaju obećanja ponašaju kao da su sinkrone obustavljajući izvršavanje dok se vraćeno obećanje ne ispuni ili odbije. Riješena vrijednost obećanja tretira se kao povratna vrijednost izraza čekanje. Korištenje asinkroniranja (eng. Async) i čekanja (eng. Await) omogućuje korištenje običnih pokušaj (eng. try) / uhvatiti (eng. catch) blokova oko asinkronog programa [3].

Asinkrone funkcije uvijek vraćaju obećanje. Ako povratna vrijednost asinkrone funkcije nije obećanje, implicitno se omotana obećanjem. Primjer jednog životnog ciklusa: bitno je shvatiti da pomoću ključne riječi čekanje (eng. Await) ispred metode, Javascript Execution Context čeka izvršenje funkcije te ovisno o vraćenom obećanju nastavlja rad [3]. U nastavku rada na slici 2.6. prikazano je asinkroniranje – čekanje (eng. sync-await).

```
async function sequentialStart() {
 console.log('==SEQUENTIAL START==')

 // 1. Execution gets here almost instantly
 const slow = await resolveAfter2Seconds()
 console.log(slow) // 2. this runs 2 seconds after 1.

 const fast = await resolveAfter1Second()
 console.log(fast) // 3. this runs 3 seconds after 1.}
```

Slika 2.6. Asinkroniranje – Čekanje (eng. Async – Await)

2.4. Tailwind css

Tailwind css je korisnost prva okolina koja ima riješene neke probleme vezane uz dizajniranje i kreiranje današnjih internetskih stranica i aplikacija. Odnosi se na praksu korištenja skupa semantičkih klasa koje imaju jednu posebnu svrhu, poput postavljanja boje pozadine ili mijenjanja težine fonta. Što znači da za svaki css daje vjetar u leđa i pruža klasu koja referira taj posjed. Čest problem u današnjem razvoju je arhitektura css-a koja ostavlja dobar prosjek za SEO i Google Crawlere. Kroz takav proces se dopušta manipuliranje h1, h2 html tagova bez da Google Crawleri zamjere, što donosi nove opcije za dizajniranje internetskih stranica/aplikacija. Iako se na više mjesta koriste iste klase, tailwind i dalje ne krši DRY princip (engl. Dont Repeat Yourself - ne ponavljaj se) jer svaka ta klasa referencira na isti css posjed za svaki html node koji ga koristi. Dodatno pruža na mogućnost dodavanja tailwind.config.js datoteke koja je ništa više, nego izvoz konfiguracijskog objekta. Ovakva vrsta dokumentacije također koristi alat kao Nuxt Content što mi je zapravo bila i inspiracija za pisanje ovoga rada. Cijeli kontent dokumentacije je statičan sto omogućuje nevjerojatno brzo navigiranje kroz dokumentaciju, a to je nešto prema čemu moj projekt cilja [4].

3. PROCES IZRADE APLIKACIJE

Prije same izrade aplikacije bitno je znati na koji način razvojna okolina radi u pozadini. Pri pokretanju građenja "zapeče" nam se nuxt.config.js datoteka. Ona sadrži sve konfiguracijske podatke potrebne za razne module koje koristimo kroz projekt kao i neke generalne stvari npr. $axios module koji se koriste za dohvaćanje podataka sa servera, ima početno postavljen Base-url koji se čita iz env.js datoteke (engl. Enviorment file). Package.json datoteka sadrži popis svih modula i npm paketa te njihove trenutne verzije, također daje mogućnost manipuliranja verzijama svakog pojedinog modula.

Kod Nuxt projekta jedna od bitnijih stvari je arhitektura direktorija i datoteka, dok će u nastavku biti prikazani nekih od njih.

3.1. Glavni direktoriji

Najkorišteniji direktorij u razvoju aplikacije je komponentni direktorij. On čuva sve komponente koje koristimo u projektu. Šta je komponenta? Komponenta je dio programa kojoj je svrha da se iznova koristi na više mjesta. Služi za odvajanje programskog koda koji je Reusable na način da se doda u bilo koju drugu datoteka te spoji s njom. Poštuje princip DRY načela koje kaže ne ponavljaj se. Program postaje čitljiviji, a svaka promjena u budućnosti automatski lakša. Direktorij sa stranicama odgovoran je za url strukturu internetske stranice ili aplikacije [5].

Ako posjetimo na internetskoj aplikaciji url "domena/Docs/12" dolazimo na poseban page koji se nalazi pod docs/id. Datoteka se učitava zajedno sa svim njenim dodacima te se prikazuje na ekranu. ID jedinamičan naziv za neku datoteku ili direktorij. On nam zapravo služi većinom za posjećivanjesinglica. Ako neki entitet ima više djece, svako to dijete ima neki svoj id i određene podatke koji su vezani na njega. Preko ID-a renderamo istu datoteku kao i za svako drugo dijete, ali pridohvaćanja sa servera šalje se u body api rute i taj id koji dohvaća specificirane podatke ovisneo njemu. Ovakva značajka uvelike olakšava rad i arhitekturu [5].

3.2. Layout

Layout ili izgled je okvirni dio internetske aplikacije. On se pojavljuje na svakoj stranici dok se njegovsadržaj mijenja ovisno o url-u koji se posjećuje. Kako se koristi vuetify kao dodatni module zastrukturu html-a mogu se koristiti njegovi html elementi kao <v-app-bar>, <v-navigation- drawer> itd. Prednost korištenja je automatski pravilna struktura html-a za SEO i Google Crawlere. U vuetify dokumentaciji može se naći mnoštvo alata za uređivanje izgleda i manipuliranje logike elemenata. Jedan od bitnijih je v-model koji služi kao pohranastanja određenog elementa.Ako mu se pridoda neka bool varijabla, mutiranjem varijable iz točnog u netočno i obrnuto uklanja se i dodaje taj cijeli blok na ekranu korisnika. Također se mogu drugi elementi uvjetovatiovisno o tuđim v-modelima. Pomoću Javascripta može se nadjačati (engl. Override), dobiti I postaviti metoda koristeći računski posjed [5]. U nastavku je na slici 3.1. prikazano nadjačanjeV- modela, dok je na slici 3.2. prikazana struktura layouta.

```
computed: {
  drawer: {
    get() {
      return 'nesto'
    },
    set(){
      this.mapData({x: 'nesto'})
    }
  }
},
```

Slika 3.1. Nadjačanje V - Modela

Izgled se generalno koristi za lagano navigiranje kroz stranice internetske aplikacije, odlogiravanje i ulogiravanje.

Slika 3.2. Struktura layouta

3.3. Jedna komponenta

Dokumentacija se prikazuje u dinamično napravljenim kutijama. Za renderanje svih objekata nekog arraya koristi se Nuxt značajka v-for. On dopušta da se iterira po arrayu i zasvaki objekt koji ima kao dijete stvori zasebnu instancu komponente. Većinom ih se drži u nekom kontejneru koji ima definiran grid kako bi komponente bile jedne ispod drugih. Kroz sliku 3.3. prikazan je primjer iteriranja u html-u.

```
<v-list>
  <v-list-item v-for="item in items" :key="item.title" link>
    <v-list-item-icon>
      <v-icon>{{ item.icon }}</v-icon>
    </v-list-item-icon>
    <v-list-item-content>
      <v-list-item-title>{{ item.title }}</v-list-item-title>
    </v-list-item-content>
  </v-list-item>
</v-list>
```

Slika 3.3. Primjer iteriranja u html-u

Ključ (:key=”item.title”) koji se vidi ljudi često zaborave staviti ili misle da jednostavno nije toliko bitan jer vizualno oku nije očito. Ali njegova svrha je vrlo važna pogotovo za velike projektekoji barataju sa ogromnom količinom podataka. Ključ nam služi kako bi vue znao razlikovati svaku komponentu koju smo stvorili. Ako se uzme tipa jedan objekt iz arraya i promijeni neku njegovu vrijednost (engl. key value pair) Vue ponovo rendera samo točno tu komponentu čiji je objekt mutiran. Dakle ako se ne stavi ključ na html na kojem je v-for i mutira bilo koji objekt iz arraya, Vue ponovo rendera sve komponente što nije dobro za optimizaciju aplikacije. Na slici 3.4. u nastavku prikazana je dinamična tablica.

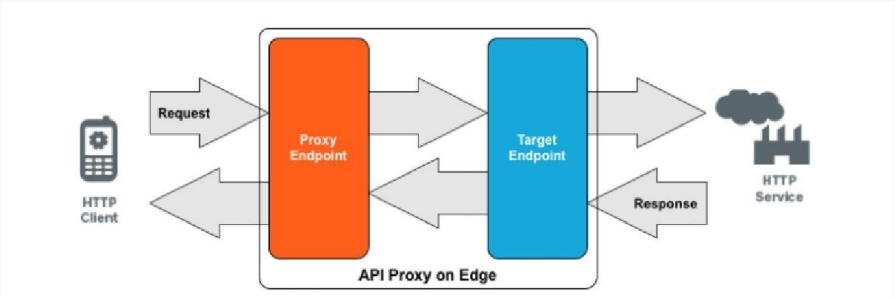
```
<v-card class="mt-2" elevation="1">
  <v-simple-table>
    <template v-slot:default>
      <thead>
      <tr>
        <th class="text-left">
          Key
        </th>
        <th class="text-left">
          Type
        </th>
        <th class="text-left">
          Required
        </th>
      </tr>
      </thead>
      <tbody>
      <tr
        v-for="value in item.values"
        :key="item.key"
      >
        <td>{{ value.key }}</td>
        <td>{{ value.type }}</td>
        <td>
          <v-icon...>
          <v-icon...>
        </td>
      </tr>
      </tbody>
    </template>
  </v-simple-table>
</v-card>
```

Slika 3.4. Dinamična tablica

Kao omotač (engl. Wrapper) komponente koristi se V-Card koji daje stil i elevaciju, a kao strukturu iliti kostur komponente, koristi se V-Simple-Table koji omogućuje iteraciju i definiranje izgleda jednog reda tablice pomoću plain Html-a, kao što vidimo na slici iznad.

3.4. Url

Prije nego se objasni dohvaćanje podataka i njihovo prikazivanje u tablici mora se objasniti svrha i namjena podataka koji se dokumentiraju, a to su Api zahtjevi. Većina modernih aplikacija ima odvojen pozadine od sučelja. Što znači da mobilne aplikacije ili internetske aplikacije komuniciraju sa istim pozadinskim serverom i dohvaćaju podatke kriptirane i definiranena isti način. Na slici 3.5.u nastavku prikazan je url protok zahtjeva.



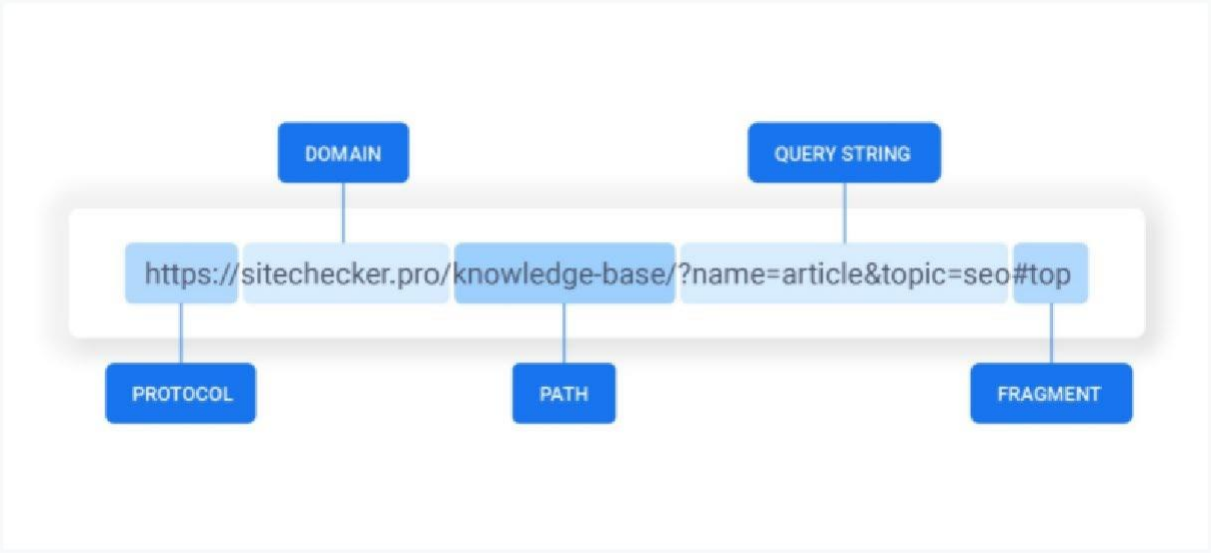
Slika 3.5. Url protok zahtjeva

Pozadinski programeri definiraju rute (engl. Path) do određenih podataka. Svaka ruta ima određenu konfiguraciju koja se može mutirati. Tipa korisnik može tražiti podatke sortirane ili filtrirane na određeni način.

Svaka frontend aplikacija mora izgraditi Builder za Api Requestove.

1. Protokol
2. Domena
3. Put
4. Niz upita
5. Ulomak

Https ili http - određuje sigurnosni protokol. Nakon toga definira se domena, koja se većinom sprema u Environment datoteku jer sasvim je uredu da se ta varijabla “zapeče” napokretanju aplikacije. Path je dio koji ovisi od Requesta do Requesta. On zapravo govori serveru koji se podaci žele dohvatiti, mutirati, obrisati. Query String se koristi kao dodatni key value parovi, većinom im je svrha progurati dodatne informacije do servera koji inače nisu definirane direktno u Path-u. Tipa odrediti način sortiranja ili filtriranja. Na slici 3.6. u nastavku prikazana je url struktura.

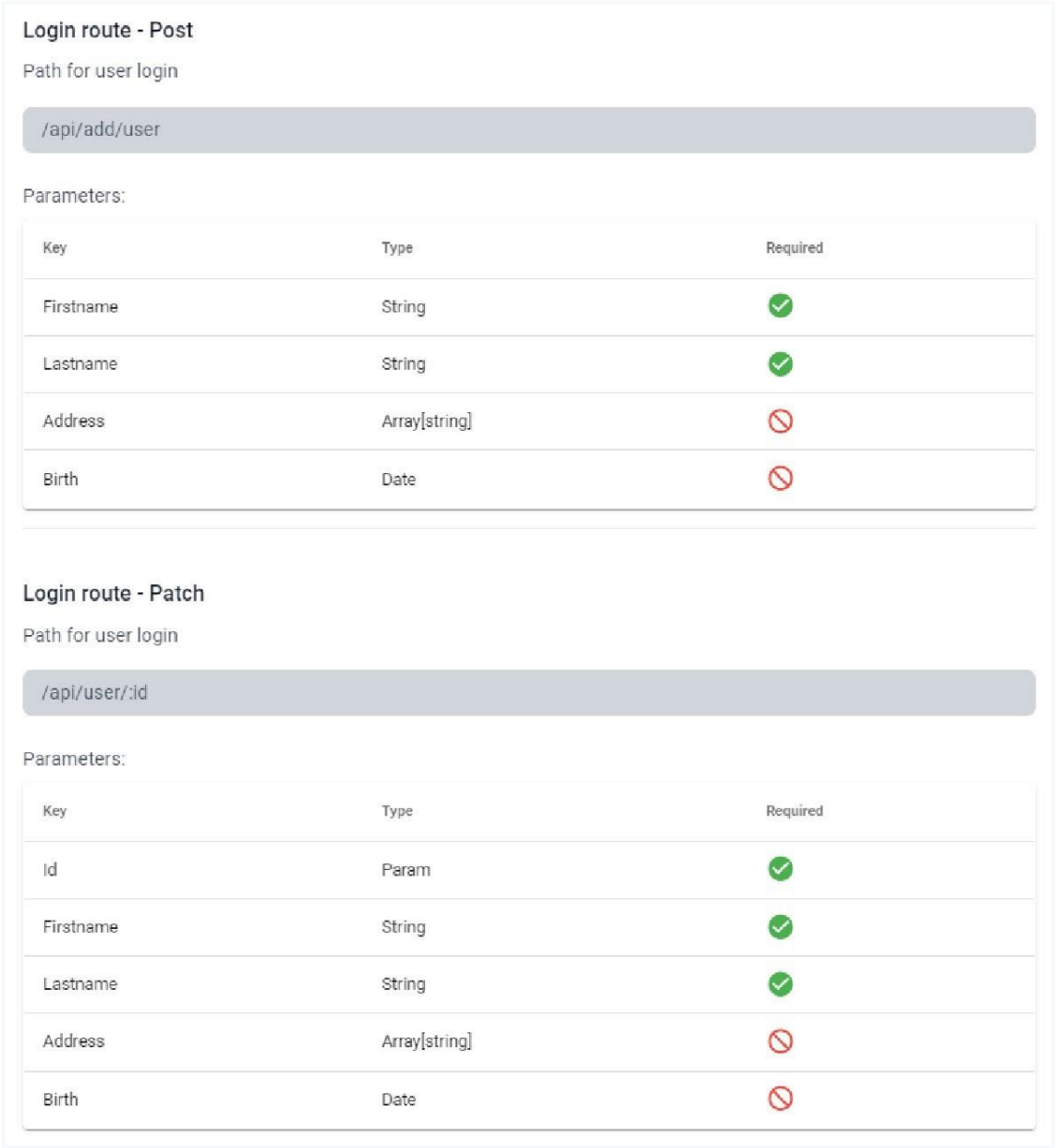


Slika 3.6. Url struktura

Nakon što je definiran url path potrebno je reći serveru koju se radnju pokušava odraditi na zadanim podacima. U nastavku na slici 3.7. prikazan je vizualni primjer komponente o kojoj se govori.

Primjer na korisnicima (engl. Userima).

- Post - dodavanja ili stvaranje novog korisnika u sustav
- Patch - editiranje već postojećeg korisnika
- Delete - brisanje postojećeg korisnika
- Get - dohvaćanje podataka nekog korisnika



Slika 3.7. Vizualni primjer komponente o kojoj se govori

Primjer kako to izgleda u aplikaciji. Svaki sučeljski programer svakodnevno spoji veći broj Api ruta. Koristeći dokumentaciju štedi se vrijeme pozadinskim programerima. Iz slike se može vidjeti informacije kao staza i tip rute. Koje vrijednosti prima u body parametrima i kojega su tipa. Npr. prvo ime je varijabla u koju se sprema ime nekog korisnika, ona je tipa string i mora se poslati ako se želi da server propusti zahtjev. Razlika između postavljanja rute i popravljanja je u tome što popravljanje rute ima i dinamičku vrijednost ID koja ovisi od korisnika do korisnika.

4. ZAKLJUČAK

U ovom završnom radu dan je prikaz modernih tehnologija za razvijanje internetskih aplikacija. Pobližeje objašnjen tok podataka između raznih komponenata i stranica. Svaka veća firma vodi računao dokumentiranju programskog koda čime se osigurava veća čitljivost samoga programa. Dokumentacija ubrzava proces uvođenja novih ljudi na projekte i omogućuje brži i jednostavniji razvoj. U ovom radu koristio se Nuxt Content modul koji omogućava spremanje podataka u jednu JSON datoteku te pruža Api za lagano dohvaćanje, brisanje, sortiranje i filtriranje podataka. Većina danas dostupnih dokumentacija koristiisti princip rada, a to je statično čuvanje podataka koje smanjuje broj zahtjeva koje klijent radi prema poslužitelju, što omogućava brže pretraživanje i navigiranje kroz aplikaciju. Nuxt okolina daje na raspolaganje poslužiteljsko iscrtavanje, što uvelike pomaže jer poslužitelj vraća već kreirani html dokument s unaprijed popunjenim podacima koji su mapirani prije iscrtavanja stranice. Krajnji cilj developmenta je ubrzavanje i optimiziranje programskog koda na internetskoj aplikaciji što na koncu donosi i bolju SEO ocjenu.

LITERATURA

[1] Vue dokumentacija
https://vuejs.org/v2/guide/

[2] Nuxt dokumentacija
https://nuxtjs.org/docs/get-started/installation

[3] Nuxt-content dokumentacija
https://content.nuxtjs.org/

[4] Tailwind css dokumentacija
https://tailwindcss.com/docs

[5] Nuxt-content službeni primjeri programskog koda
https://github.com/nuxt/content

SAŽETAK

Cilj ovog završnog rada bio je izraditi Internetsku aplikaciju koju služe sučeljski programeri kao strukturiran prikaz api ruta kreiranih od strane pozadinskih programera. U početku ovog završnog rada nalazi se kratki uvod u moderne tehnologije koje se danas koriste u programiranju. Nadalje, objašnjeni su Javascript jezik, Api rute, Nuxt i Vue okolina, Tailwind css okolina te NuxtContent module. Glavni dio pruža uvid u način kreiranja komponenti te generalne tehnike stvaranja internetske aplikacija.

ABSTRACT

The aim of this final paper was to create a Web application that serves frontend developers as a structured view of api routes created by backend developers. At the beginning of this seminar there is a brief introduction to modern technologies used in todays web development. Furthermore we explained, Javascript language, Api routes, Nuxt and Vue environment, Tailwind css environment and Nuxt Content module. The main part provides an insight into how to create components and general techniques for creating web applications.

ŽIVOTOPIS

Borna Marin rođen je 19. kolovoza 1997. godine u Vinkovcima. Pohađao je Osnovnu školu Vladimira Nazora u Vinkovcima gdje biva uspješan učenik (završio je svih 8 razreda s prosjekom 5,00). 2012. godine upisuje Gimnaziju Matije Antuna Reljkovića u Vinkovcima, matematički smjer. 2016. godine polaganjem državne mature završava srednju školu te iste godine upisuje preddiplomski sveučilišni studij na Fakultetu za elektrotehniku, računarstvo i informacijske tehnologije Osijek, smjer Računarstvo. U vrijeme pisanja ovog završnog rada zaposlen je kao student frontend developer u Gauss Development-u. Vrlo se dobro služi engleskim jezikom te posjeduje B2 diplomu.

PRILOZI

1. Cd