Vežba 4

U ovoj vežbi napravićemo vue.js aplikaciju koja se povezuje na REST API i čita podatke. Koristićemo RAF Alumni servis - direktorijum bivših studenata. Aplikaciju ćemo na kraju build-ovati i postaviti na node.js express.

Gotova vežba dostupna je na repozitorijumu https://github.com/prvul/sj-v4
Možete da kod sebe klonirate, uradite npm install da bi se povukli potrebni dependency moduli i da pokrenete aplikaciju sa npm run serve

Uputstvo za samostalni rad

Za samostalni rad/projekat napravićete po ovom modelu svoju aplikaciju koja se povezuje na REST API vašeg servisa napravljenog u Vežbi 3. Build-ovana aplikacija treba da se postavi na APP Servis koji ste pravili u Vežbi 3, a koji servira html stranice adminskog interfejsa. Adminski interfejs prebacite u podfolder /admin a vue.js aplikaciju postavite na home rutu /

Prilagođavanje adminske aplikacije iz vežbe 3

Obratite pažnju da treba da prilagodite linkove. Možete da menjate sve linkove a možete da iskoristite <base> tag koji služi da postavi osnovnu putanju za relativne linkove (https://www.w3schools.com/tags/tag_base.asp). Dodajte <base href="http://localhost:8000/admin/"> u <head> svake stranice. Ovo možete i sa find-replace nad celim projektom - uradite find za <head> i replace sa <head><base href="http://localhost:8000/admin/"> i to bi trebalo da reši stvar bez puno muke.

Kreiranje vue.js aplikacije

1. Instalirajte vue-cli globalno, pomoću ovog alata ćemo praviti vue.js aplikaciju
    a. npm install -g vue-cli
2. Napravite aplikaciju
    a. vue create vezba4vue
    b. Ispratite upitnik
        i. Izaberite vue2
    c. Dobićete kreiranu strukturu fajlova i foldera. Ovo je zapravo node aplikacija. Pogledajte package.json - scripts.serve i scripts.build omogućavaju dodavanje inicijalnih konzolnih poziva prilikom pokretanja i build-ovanja aplikacije.
3. Pokrenite aplikaciju
    a. npm run serve
        i. Pazite u kom ste folderu, ako je kreiran podfolder prvo cd podfolder pa onda npm run serve
    b. Otvorite link u browseru
    c. Ovo je razvojni server, ne koristi se u produkciji. Bavićemo se produkcijom kasnije.
    d. Ako menjate fajlove ne morate da restartujete ovo, razvojni server prati promene nad fajlovima i sam se restartuje i osvežava u browseru - pokrenete ga jednom i zaboravite na konzolu.

Deployment vue.js aplikacije

1. Zaustavite produkcioni server u konzoli (ctrl+c)
2. Izvršite
    a. npm run build
    b. Dobićete novi folder /dist u kome su spremni svi fajlovi aplikacije. Ovo treba da prebacimo na web server koji će servirati aplikaciju.
3. Napravite novi folder, van foldera projekta, nazovite ga vezba4node
4. U tom folderu izvršite u konzoli
    a. npm install express
5. Napravite app.js
6. U app.js postavite osnovni express servis sa static middleware

```
const express = require('express');
const path = require('path');
const app = express();
app.use( express.static( path.join(__dirname, 'static', 'dist') ) );
app.listen(8000);
```

7. Napravite folder /static
8. Kopirajte /dist u /static
9. Pokrenite express servis
    a. node app
10. Otvorite u browseru localhost:8000, trebalo bi da vidite vue.js aplikaciju istu kao što ste je dobili sa npm run serve
    a. Naravno, sa tom razlikom što je ovde servirana kompajlirana verzija sa web servera

U nastavku vežbe dovoljno je da pokrenete u folderu vue aplikacije npm run serve i tako razvijate i testirate aplikaciju. Deployment proces radite na kraju, kada aplikaciju u konačnom obliku treba staviti na server - tj. kada budete spremni da predate vaš projekat.

Ukratko o strukturi vue aplikacije

/public sadrži sve javne resurse-slike, css fajlove, ikonice itd.
/public/index.html je entry point fajl, u kome se nalazi `<div id="app"></div>` i u koji se na kraju fajla injektuju build-ovani js fajlovi aplikacije. Ovaj div služi kao mesto gde će aplikacija biti ubačena u stranicu. GUI aplikacije se kreira dinamički i kada je spreman radi se nešto tipa document.getElementById("app").appendChild(vueAppRootElement);
/src folder sadrži fajlove aplikacije, i kako budemo razvijali aplikaciju ovo će biti sve punije fajlovima i folderima.
/src/main.js sadrži inicijalizacioni kod i ovo je zapravo početni skript aplikacije.
/src/App.vue je glavna komponenta, koju main.js koristi i koja se injektuje u onaj div u index.html.
/src/components/HelloWorld.vue je primer komponente, koja se koristi u App.vue.

Primetite da je struktura ova .vue fajla ista:
<template>

html kod te komponente, u kome su html tagovi i vue tagovi-komponente
</template>
<script>
javascript kod te komponente, gde imamo potrebne import-e a zatim export u kome se definiše objekat komponente i sadržaj te komponente
</script>
<style>
css kod. ako se u <style> doda atribut *scoped* onda će css biti specifičan za komponentu
</style>

Pravljenje prve komponente - header

Ova komponenta treba da prikazuje naslov stranice-sadržaja koji se prikazuje. Naslov se dobija spolja, iz komponente koja u sebi sadrži header (npr. spisak studenata, jedan student)

1. Kopirajte /src/components/HelloWorld.vue u Header.vue
2. Počistite fajl od suvišnog koda. Iz <template> i <style> dela obrišite sve
3. U <template> ubacite jedan div sa klasom header, u njemu h1
4. U h1 kao sadržaj stavite ispis varijable *title* ovako:

```
<div class="header">
  <h1>{{ title }}</h1>
</div>
```

5. U style tag dodajte atribut *scoped*, ako ga nema, i dodajte css za klasu header kojim postavljate centralno poravnanje teksta i font Gloock.

```
<style scoped>
.header{
  font-family: 'Gloock', serif;
  text-align: center;
}
</style>
```

6. Da bi font bio dostupan potrebno je uključiti font. Ovo ćemo da namestimo globalno, u App.vue, u <style>

```
@import url('https://fonts.googleapis.com/css2?family=Gloock&display=swap');
```

Ovo je google font, link možete generisati na fonts.google.com za izabrani font

7. U <script> delu definišite komponentu. Ime je Header a od svojstava ima samo title

```
export default {
  name: 'Header',
  props: {
    title: String
  }
}
```

Dodavanje komponente u App.vue

8. Otvorite App.vue
9. U <script> promenite postojeći import Helloworld tako da se importuje Header. U defaults objektu, u components, uklonite HelloWorld i dodajte Header. Ovako u komponentu uključujemo i registrujemo druge komponente koje će se sadržati u njoj.

```
<script>
import Header from './components/Header.vue'

export default {
  name: 'App',
  components: {
    Header
  }
}
</script>
```

10. U <template> dodajte komponentu. Komponenta se zove Header, pa će tag biti <Header /> Pošto komponenta ima property *title* tu vrednost treba proslediti komponenti preko istoimenog atributa.

```
<template>
  <div>
    <Header title="RAF Alumni"/>
  </div>
</template>
```

11. Title bi trebalo da bude promenjiv, zavisno od toga šta se prikazuje, pa bi trebalo da tu vrednost čitamo iz promenljive (iz *stanja* aplikacije, uslovno rečeno).
    a. U <script> delu, u default objekat dodajte metod data() koji vraća objekat sa svojstvom title.

```
export default {
  name: 'App',
  components: {
    Header
  },
  data(){
    return{
      headerTitle: "RAF Alumni"
    }
  }
}
```

    b. U komponenti Header dodajte v-bind: ispred naziva atributa - time se vrednost atributa tretira kao naziv svojstva u default.data()

```
<Header v-bind:title="headerTitle"/>
```

    c. Umesto v-bind:title možete da pišete i samo :title

12. Pošto komponenta Header sada uzima title iz svojstva objekta aplikacije, očekivano bi bilo da promena vrednosti ovog svojstva utiče na promenu sadržaja u komponenti Header. Ovo možemo jednostavno da isprobamo dodavanjem input polja koje ćemo vezati za title.
    a. U App.vue ispod <Header> dodajte <input> Sa v-model="headerTItle" povežite vrednost inputa sa svojstvom headerTitle. Ovo ima dvojaku ulogu: koristiće headerTItle kao vrednost koja se prikazuje u input polju, i kada se desi promena vrednosti input polja ažuriraće se i headerTitle, čime se posledično ažurira i dinamično ispisani sadržaj u Header-u.
    ```
    <input v-model="headerTitle" type="text">
    ```
    b. Isprobajte ovo. Trebalo bi da se prilikom otvaranja stranice u input polju ispisuje vrednost naslova, a ako promenite tekst u input polju promeniće se i veliki naslov stranice.
    c. Obrišite ovaj <input> jer nam neće trebati u nastavku. Videli smo kako funkcioniše dvosmerno povezivanje.

13. Sa http://alumni.raf.edu.rs preuzmite logo i sačuvajte u /src/assets preko postojećeg fajla.

Komponenta za listanje studenata sa API

Sada ćemo da napravimo komponentu koja prikazuje listu studenata, sa paginacijom. Potrebno je da prilikom učitavanja aplikacije pošaljemo inicijalni poziv kojim dohvatamo niz id-eva svih studenata. To ćemo sačuvati unutar komponente aplikacije. Zatim treba da definišemo promenljivu u kojoj čuvamo trenutno prikazanu stranu, takođe unutar komponente aplikacije. Dodaćemo i dva linka za napred-nazad koji menjaju vrednost trenutne strane (inkrement i dekrement). U komponentu za prikaz liste prosledićemo kao parametar podniz studenata, zavisno od trenutno prikazane strane. U komponenti pratimo promenu vrednosti ovog parametra i kada se desi promena prolazimo kroz niz i za svakog studenta šaljemo novi zahtev u kome dohvatamo detaljne podatke i tako formiramo niz studenata za prikaz. Za prikaz jednog studenta pravimo komponentu koja kao parametar dobija objekat tog studenta. Komponenta koja prikazuje listu studenata to radi tako što za svaki element niza izabranih studenata (na trenutnoj stranici) prikaže po komponentu za prikaz jednog studenta.

Inicijalno dohvatanje studenata sa API i stanje u komponenti aplikacije

14. Otvorite App.vue
15. U data() dodajte svojstva sviStudenti i current

```
data(){
  return{
    headerTitle: "RAF Alumni",
    sviStudenti: null,
    current: 0
  }
}
```

16. Ispod komponente <Header> dodajte potrebne elemente, sa v-on:click, ili, kraće, @click pridružujete event handler. Ovo je u kontekstu komponente, pa ćemo funkcije definisati kao metode u komponenti.

```
<div>
  <button @click="prev()">Prethodno</button>
  ...
  <button @click="next()">Sledece</button>
</div>
```

17. U <script> delu ispod data(){} dodajte methods objekat u kome ćemo deklarisati metode next() i prev(). Dodaćemo i uslove koji sprečavaju da se ode u minus i preko kraja liste. Uzećemo da lista sadrži 10 elemenata.

```
data(){
  ...
},
methods:{
  next(){
    if( this.current * 10 < this.sviStudenti.length ){
      this.current++;
    }
  },
  prev(){
    if( this.current > 0){
      this.current--;
    }
  }
}
```

18. Pandan događaju load ovde je *mounted* - trenutak kada se konstruisana aplikacija umeće u glavni div kontejner u index.html. Definisaćemo mounted() i tu staviti inicijalni fetch() koji dohvata niz svih studenata. Na alumni.raf.edu.rs/rs/api nalazi se jednostavan API.

```
data(){
    ...
},
methods:{
    ...
},
mounted(){
  fetch("http://alumni.raf.edu.rs/rs/api/list")
  .then( res=>res.json() )
  .then( data=>{
    this.sviStudenti = data;
  });
}
```

Kreiranje komponenti za prikaz studenata

Prvo ćemo da napravimo komponentu koja prikazuje jednog studenta, na osnovu objekta datog kao parametar.

19. Napravite novi fajl /components/StudentSingle.vue
20. Kopirajte iz Header.vue kod, kako bismo imali početnu strukturu.
21. U default objektu definišite name: "StudentSingle" a u props stavite samo student: Object

```
export default {
  name: 'StudentSingle',
  props: {
    student: Object
  }
}
```

22. Analizirajte strukturu objekta studenta koji se dobija sa API. Evo jednog primera: http://alumni.raf.edu.rs/rs/api/diplomac/5515
23. U <template> bloku napravićemo jedan div sa klasom student, unutar koga želimo da prikažemo sliku, ako je ima, i ime i prezime.
    a. U objektu studenta imamo atribute slika, ime, prezime, ali se slike nalaze na http://alumni.raf.edu.rs/images/slike/

```
<div class="student">
  <img v-if="student.slika"
     :src="'http://alumni.raf.edu.rs/images/slike/'+student.slika" />
  <p v-else>Nema slike</p>

  <h3>{{ student.ime }} {{ student.prezime }}</h3>
</div>
```

24. Možemo da dodamo i nešto css-a da ovo izgleda malo pristojnije. Evo primera:

```
<style scoped>
.student{
    width:150px;
    display:inline-block;
    margin:10px;
    vertical-align: top;
}
.student img{
    width:150px;
    height:150px;
    object-fit:cover;
}
.student p{
    height:150px;
    margin:0;
```

```
        background-color:silver;
        text-align:center;
    }
    </style>
```

Sada ćemo da naparvimo komponentu koja prikazuje listu studenata, koristeći komponentu StudentSingle za prikaz svakog studenta u nizu.

25. Napravite novi fajl /components/StudentiList.vue
26. Kopirajte iz Header.vue kod, kako bismo imali početnu strukturu.
27. U default objektu podesite name komponente na StudentiList
28. Importujte komponentu StudentSingle i registrujte je u default.components
29. Dodajte svojstvo studentiIDs: Array, koje će u sebi sadržati id-eve studenata koje treba prikazati na trenutnoj stranici (glavna komponenta aplikacije će na osnovu current i sviStudenti da izdvoji deo niza koji treba da se prikaže).
30. U data() dodajte niz studenti, koji će sadržati objekte studenata koje dovlačimo sa API-ja jedan po jedan, na osnovu id-eva iz studentIDs. Ovo ćemo prosleđivati u komponente StudentSingle

```
import StudentSingle from '@/components/StudentSingle.vue';
export default {
    name: 'StudentiList',

    components: {
        StudentSingle
    },

    data() {
        return {
            studenti: []
        }
    },

    props: {
        studentiIDs: Array
    },
```

31. Prilikom učitavanja aplikacije, na mount, želimo da prođemo kroz niz id-eva studenata, svaki od njih dohvatimo sa API-ja, i tako dobijeni objekat ubacimo u niz studenti.

```
mounted() {
    this.studentiIDs.map( obj => {
        fetch(`http://alumni.raf.edu.rs/rs/api/diplomac/${obj.id}`)
            .then( obj => obj.json())
            .then( res => this.studenti.push(res));
    });
```

```
    },
```

32. Isto to želimo da radimo i svaki put kada se niz studentiIDs promeni (menja se kada se klikne na prev/next). Ovde treba da pratimo promene svojstva studentiIDs i pridružimo funkciju koja će se pozvati kada se to desi. Funkcija kao argumente prima newValue i oldValue.

```
    watch: {
        studentiIDs(nVal) {
            this.studenti = [];
            nVal.map( obj => {

fetch(`http://alumni.raf.edu.rs/rs/api/diplomac/${obj.id}`)
                    .then( obj => obj.json())
                    .then( res => this.studenti.push(res));
            });
        }
    }
```

33. Konačno, u <template> treba da dodamo generisanje komponenti za prikaz svakog studenta, na osnovu niza studenti. Možemo da koristimo v-for, kojim se naznačava da komponenta treba da se instancira za svaki element niza.

```
    <template>
        <div>
            <StudentSingle v-for="student in studenti"
                    :key="student.id" :student="student"/>
        </div>
    </template>
```

Dodavanje liste u glavnu komponentu apiikacije

34. Otvorite App.vue
35. Importujte i registrujte komponentu StudentiList

```
import StudentiList from '@/components/StudentiList.vue'

export default {
  name: 'App',
  components: {
    Header, StudentiList
  },
...
```

36. u <template> deo, na kraj dodajte komponentu. Želimo da bind-ujemo atribut *studenti* na deo niza sviStudenti, u skladu sa trenutnom stranicom. Kada koristimo :atribut="" vrednost u

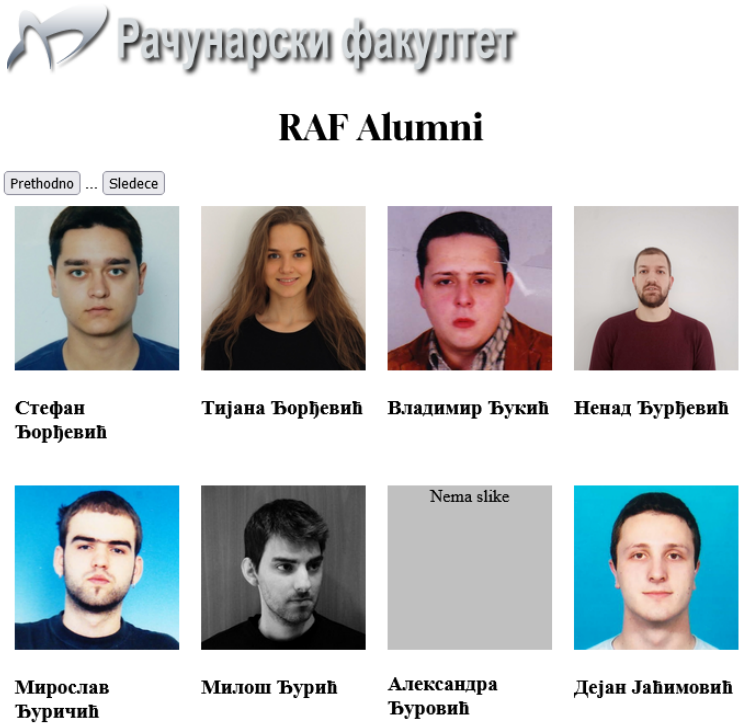
zagradama tretira se kao izraz i evaluira se, pa možemo da napišemo i nešto složenije od naziva promenljive.

```
<StudentiList :studentiIDs="sviStudenti.slice(current*10, current*10+10)"/>
```

37. Inicijalno, prilikom pokretanja aplikacije, niz će biti prazan pa želimo da prikažemo poruku "Lista još nije spremna" (ili neki ajax loader). Sa v-if možemo da pridružimo uslov na osnovu koga se komponenta prikazuje ili sakriva na stranici, pa ćemo da zapravo uradimo ovako nešto:

```
<StudentiList v-if="sviStudenti" :studentiIDs="sviStudenti.slice(current*10,
current*10+10)"/>
<p v-else="sviStudenti">Lista još nije spremna</p>
```

38. Proverite da li sve radi. Trebalo bi da izgleda kao na slici



Drugi deo: rute i stranice

U ovom delu napravićemo vue.js aplikaciju koja ima više stranica-ruta. Da bismo to uradili potrebno je prilikom kreiranja aplikacije da izaberemo router komponentu. Moguće je ovo dodati i naknadno, ali nije preporučljivo jer može doći do prepisivanja bitnih delova ključnih fajlova aplikacije, pošto se router ubacuje u konfiguraciju projekta.
Kako funkcionišu rute? URL može u sebi da sadrži i deo koji počinje sa # a koji se tretira kao ID elementa na stranici. U opštem slučaju browser ovo tumači kao link ka određenom elementu na stranici i prilikom otvaranja stranice skrolovaće do mesta gde element sa tim id-em počinje. Ovo može da se iskoristi i kao ulazni parametar aplikacije, tako što će aplikacija prilikom učitavanja stranice da dohvati url, izdvoji ovaj id i to tretira kao npr. naziv stranice koja treba da se inicijalno

prikaže - u vue.js to bismo izveli tako što u App.vue mount-ujemo komponentu zavisno od vrednosti ovog parametra. Router je mehanizam koji mapira ovakve, ali i obične linkove (po patternima koje zadate) na komponente-stranice koje treba da se prikažu.
Moguće je hijack-ovati browser tako da se menja url a da se aplikacija-stranica ne učitava ponovo cela sa web servera. Problem koji ovde nastaje je Back i Forward u browseru, tj. kretanje kroz istoriju. History mode služi da spreči reloadovanje stranice kada se ide unazad-unapred kroz istoriju browsera, pa se to prebacuje na router da mount-uje odgovarajuću komponentu.

1. Napravite novu vue.js aplikaciju
    a. vue create vezba-4-2
2. Izaberite manual configuration (poslednja opcija)
3. Uključite router
4. Prihvatite history mode (upišite Y)
5. Kada se projekat napravi primetićete nekoliko novih elemenata u strukturi projekta
    a. Folder /router u sebi sadrži index.js, u kome ćemo definisati rute ka stranicama aplikacije
    b. Folder /views sadrži komponente tih stranica. I dalje je App.vue glavna komponenta, ali će zavisno od rute mountovati odgovarajuću view komponentu. Ove komponente su po svojoj strukturi iste kao i komponente u /components, ali im je namena drugačija - ovo su, uslovno rečeno, "glavne" komponente, "route handler" na neki način.
    c. U main.js nalazi se import za router i u konstruktoru Vue objekta nalazi se router instanca
6. Otvorite App.vue, primetite u <template> bloku dva linka, ovog oblika
    a. `<router-link to="/about">About</router-link>`
    b. Ovo će generisati <a> tag čiji je href (link) takav da odgovara ruti.
7. Primetite i tag `<router-view/>`
    a. Ovo je mesto gde će router mount-ovati potrebnu view komponentu, zavisno od rute.

Stranica Home

8. Iz prethodne vežbe kopirajte komponente Header, StudentiList i StudentSingle u folder /components
9. App.vue u prethodnoj vežbi ćemo prebaciti u /views/Home.vue - kopirajte kompletan kod iz jednog fajla u drugi.
    a. U <img> tagu promenite src tako da umesto . koristi @ kako bi pravilno konstruisali putanju (komponenta više nije u root folderu projekta pa . kao current folder ne daje tačnu referencu. @ će se tretirati kao root folder projekta)
    `<img src="@/assets/logo.png">`
    b. <img> tag premestite u App.vue, iznad ,jer je ovo element koji se prikazuje kao zaglavlje svake stranice.

Stranica za prikaz jednog studenta

Želimo da studenti u listi imaju link ka detaljnom prikazu, gde će se prikazati kompletna biografija izabranog studenta. Link može da bude u obliku /student/:id

Analizirajte strukturu objekta studenta koji se dobija sa API. Evo jednog primera: http://alumni.raf.edu.rs/rs/api/diplomac/5515

10. Napravite fajl /views/Student.vue i kopirajte Home.vue kao osnovu koda, i počistite sav višak, ostavite samo osnovni kostur
11. Importujte i registrujte komponentu Header. U data() dodajte svojstvo student:null.
12. U mounted() treba da dohvatimo ID iz rute, pa na osnovu toga da pustimo API zahtev i dohvatimo objekat studenta. Sa this.$route.params pristupamo parametrima iz url-a

```
<script>
  import Header from '@/components/Header.vue'

  export default {
    name: 'Student',
    components: {
      Header
    },
    data(){
      return{
        student: null
      }
    },
    mounted(){
      let studentID = this.$route.params.id;

fetch(`http://alumni.raf.edu.rs/rs/api/diplomac/${studentID}`)
      .then( res=>res.json() )
      .then( data=>{
        this.student = data;
      });
    }
  }
</script>
```

13. U <template> bloku treba da prikažemo podatke o studentu. Prvo ćemo da uključimo komponentu Header u koju ćemo da prosledimo ime i prezime studenta, iz objekta student
14. Zatim dodajemo sliku, ako je ima (kao u komponenti StudentSingle)

```
<template>
    <div class="student">
        <Header :title="student.ime + ' ' + student.prezime"/>

        <img class="slika" v-if="student.slika"
            :src="'http://alumni.raf.edu.rs/images/slike/'+student.slika" />
        <p v-else>Nema slike</p>
```

15. Otvaramo div sa klasom .info u kome ćemo da ispišemo stečene diplome i zaposlenja
    a. Dodajte h3 naslov *Studije*
    b. Dodajte div u kome ispisujete datum diplomiranja i stečeno zvanje, za osnovne i master studije, ako postoje

```
<div class="info">
    <h3>Studije</h3>
    <div v-if="student.studije.osnovne">
        {{student.studije.osnovne.datum_diplomiranja}}
        {{student.studije.osnovne.steceno_zvanje}}
    </div>
    <div v-if="student.studije.master">
        {{student.studije.osnovne.datum_diplomiranja}}
        {{student.studije.osnovne.steceno_zvanje}}
    </div>
```

16. Dodajte h3 naslov *Zaposlenja*, a ispod toga div koji ćete sa v-for da instancirate za svaki element niza student.zaposlenja. Ispišite početak i kraj zaposlenja, kompaniju i radno mesto.

```
    <h3>Zaposlenja</h3>
    <div class="zaposlenje" v-for="zaposlenje in student.zaposlenja"
        :key="zaposlenje.id" :zaposlenje="zaposlenje">
        <span class="period">
            {{zaposlenje.pocetak}} - {{zaposlenje.kraj}}
        </span>
        <br>
        <b>{{zaposlenje.kompanija}}</b> {{zaposlenje.radno_mesto}}
    </div>
</div>
```

17. Evo i nešto css-a koji možete da dodate da bi ovo izgledalo elementarno uredno

```
<style scoped>
    .slika{
        float:left; margin-right:20px; margin-bottom:20px;
        width:200px;
    }
    .student{
        text-align:left;
    }
    .info{
        overflow:auto;
    }
    .zaposlenje{
        margin-bottom:20px;
    }
    .zaposlenje .period{
        font-size: 80%;
    }
```

```
</style>
```

18. Pređite u /components/StudentSingle.vue
19. Na div.student dodajte događaj click u kome ćete pozvati funkciju kojoj prosleđujete id studenta.

```
<div class="student" @click="kliknutStudent(student.id)">
```

20. Ta funkcija treba da inicira prelaz na stranicu/rutu Student, sa parametrom id studenta

```
methods:{
    kliknutStudent(id){
        this.$router.push({ name: 'Student', params: { id: id }
});
    }
}
```

21. Otvorite router/index.js i dodajte rutu *Student*
    a. U const routes nizu obrišite element About
    b. Dodajte element niza, objekat, u kome je definisana putanja, naziv putanje i komponenta (u views folderu) koju treba mount-ovati u <router-view/> u App.vue

```
{
  path: '/student/:id',
  name: 'Student',
  component: Student
}
```

22. Otvorite App.vue i obrišite <router-link> za About, pošto ovu stranicu više nemamo.
23. Isprobajte aplikaciju. Trebalo bi da se na klik na studenta otvara stranica sa detaljnim podacima o studentu, i da je url u odgovarajućem obliku.

Bootstrap-vue

Bootstrap komponente upakovane su u vue kompomente i mogu se koristiti kao već gotov resurs koji je prilično bogat mogućnostima. Ovde ćemo videti kako da instaliramo bootstrap-vue u projekat, dodamo bootstrap tabelu za spisak kompanija koje dohvatamo sa API.
Postupak podešavanja i dokumentaciju možete naći na https://bootstrap-vue.org/docs

1. U konzoli prekinite pokrenuti npm run serve sa ctrl+c pa izvršite
    a. npm install vue bootstrap bootstrap-vue
2. Otvorite /public/index.html i postojeći meta tag viewport zamenite sledećim <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
3. Otvorite main.js i registrujte bootstrap-vue. Dodajte sledeće nakon reda `import router`

```
import { BootstrapVue, IconsPlugin } from 'bootstrap-vue'

// Import Bootstrap and BootstrapVue CSS files (order is
important)
import 'bootstrap/dist/css/bootstrap.css'
import 'bootstrap-vue/dist/bootstrap-vue.css'
```

```
// Make BootstrapVue available throughout your project
Vue.use(BootstrapVue)
// Optionally install the BootstrapVue icon components plugin
Vue.use(IconsPlugin)
```

Kompanije

Sada ćemo napraviti novu stranicu u kojoj se prikazuje spisak kompanija u kojima rade studenti, upotrebom bootstrap tabele.

1. Napravite views/Kompanije.vue
2. U <template> delu postavite header komponentu i zadajte statički title

```
<template>
    <div>
        <Header title="Kompanije u kojima rade naši diplomci"/>
    </div>
</template>
```

3. U <script> delu importujte Header, postavite potrebne detalje komponente i interni niz *kompanije* u kome će se čuvati niz kompanija dohvaćenih sa API-ja. Prilikom mount-ovanja komponente treba sa API preuzeti niz kompanija i smestiti u *kompanije*. To ćemo zatim prikazati u bootstrap tabeli.

```
<script>
import Header from '@/components/Header.vue'

export default {
  name: 'Kompanije',
  components: {
    Header
  },
  data(){
    return{
      kompanije: null
    }
  },
  mounted(){
    fetch(`http://alumni.raf.edu.rs/rs/api/kompanije`)
    .then( res=>res.json() )
    .then( data=>{
      this.kompanije = data;
    });
  }
}
</script>
```

4. <style> blok ostavite prazan.
5. Otvorite router/index.js i dodajte rutu za ovu stranicu-komponentu

```
,{
    path: '/kompanije',
    name: 'Kompanije',
    component: Kompanije
  }
```

6. Dodajte import za ovu komponentu

```
import Kompanije from '../views/Kompanije.vue'
```

7. Otvorite App.vue i dodajte link ka ovoj stranici u navigaciju

```
<router-link to="/kompanije">Kompanije</router-link>
```

8. Otvorite views/Kompanije.vue, sada ćemo da dodamo bootstrap tabelu koja će da ispiše sve kompanije iz niza *this.kompanije*. Ref: https://bootstrap-vue.org/docs/components/table
    a. U <template> ispod <Header> ubacite komponentu b-table

```
<b-table striped hover :items="kompanije"></b-table>
```

9. Pogledajte kako ovo izgleda. Dobili smo jednostavnu tabelu koja ispisuje sve atribute objekata. Ovo nam nije baš idealno jer prikazuje previše kolona.
    a. Dodajte u <b-table> atribut :fields koji ce sadržati niz naziva kolona-atributa objekata koje želimo da prikažemo

```
<b-table striped hover :items="kompanije" :fields="fields">...
```

    b. Definišite niz *fields* u data()

```
fields: ["naziv","logo"]
```

10. Spisak sada prikazuje samo dve kolone. *fields* možemo dodatno da opišemo detaljima i prilagodimo prikaz. Želimo da se *naziv* prikaze kao *Kompanija*, da *logo* nema naslov u zaglavlju tabele, i da uključimo mogućnost sortiranja po koloni naziva.

```
fields: [
  {
    key: "naziv",
    sortable: true,
    label: "Kompanija"
  },
  {
    key: "logo",
    label:""
  }
]
```

11. Dalje, želimo da umesto naziva slike prikažemo sliku. Sve slike nalaze se u http://alumni.raf.edu.rs/images/kompanije/ pa treba da ovo dodamo putanji. Iskoristićemo opciju formatter - pridružujemo funkciju koja obrađuje vrednost.

```
{
    key: "logo",
    label:"",
    formatter: value=>{
      if(value)
```

```
            return
"http://alumni.raf.edu.rs/images/kompanije/"+value;
          else return null;
        }
      }
```

12. I ovu vrednost/putanju do slike treba prikažemo u ćelijama kolone kao <img>. U <b-table> ćemo dodati <template> za tu kolonu. Unutar <template> zadajemo šablon sadržaja ćelije, kome prosleđujemo objekat za prikazivanje (objekat kompanije, atribut *logo*).

```
<b-table striped hover :items="kompanije" :fields="fields">
  <template #cell(logo)="data">
    <img v-if="data.value" :src="`${data.value}`" />
  </template>
</b-table>
```

    a. sa v-if prikazujemo <img> samo ako postoji putanja do slike, tj. ako formatter vrati vrednost a ne null. Ako kompanija nema logo (ima takvih slučajeva), <img> neće ni biti renderovan

13. Ako je sadržaj tabele centriran, idite u App.vue i pogledajte <style> blok, naći ćete #app text-align:center. Možete da u Kompanije.vue dodate <style scoped> i css deklaraciju

```
.table{
  text-align:left;
}
```

14. Paginacija takođe postoji kao gotova komponenta i može jednostavno da se poveže sa tabelom. Prvo ćemo da tabeli dodamo id="tabelaKompanije"

```
<b-table ... id="tabelaKompanije">
```

15. Zatim pre <b-table> dodajemo komponentu <b-pagination> sa atributima koji povezuju paginaciju sa tabelom i izvorom podataka, kao i sa trenutnom stranicom i brojem redova po stranici, koje moramo da dodamo.

```
<b-pagination
  v-model="currentPage"
  :total-rows="rows"
  :per-page="perPage"
  aria-controls="tabelaKompanije"
></b-pagination>
```

16. Dodajemo currentPage i perPage u data()

```
data(){
  return{
    perPage:10,
    currentPage: 1,
    ...
```

17. Dodajemo rows kao *izračunatu* vrednost (zavisno od dužine niza this.kompanije)

```
computed:{
  rows(){
    return this.kompanije.length;
  }
```

```
},
```

18. Sada je paginacija spremna, i treba još da u <b-table> dodamo perPage i currentPage. <b-table> u konačnom obliku izgleda ovako:

```
<b-table
    striped
    hover
    :items="kompanije"
    :fields="fields"
    :per-page="perPage"
    :current-page="currentPage"
    id="tabelaKompanije">
      <template #cell(logo)="data">
        <img v-if="data.value" :src="`${data.value}`" />
      </template>
  </b-table>
```

Time smo završili ovaj deo vežbe. Videli smo kako da postavimo vue.js aplikaciju, da pokupimo podatke sa API-ja, napravimo komponente, dodamo rute i stranice (koje su takođe komponente, ali izdvojene) i kako da uključimo bootstrap-vue i koristimo gotove komponente - tabelu i paginaciju.

U nastavku ćemo videti šta je vuex i kako da bolje organizujemo kod. Ukoliko budete krenuli da pravite svoj projektni zadatak sad očekujte da nakon sledećeg časa radite refaktorizaciju koda, tako da sve fetch() i sl. pozive i pomoćne funkcije izmestite na za to predviđena mesta.