Klijentske web-aplikacije i vue.js

Vidović, Ana

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Ana Vidović

KLIJENTSKE WEB-APLIKACIJE I VUE.JS

Diplomski rad

Voditelj rada:
doc. dr. sc. Zvonimir Bujanović

Zagreb, rujan, 2018.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Hvala svima koji su doprinijeli nastanku ovoga rada, ponajprije mom mentoru.

Sadržaj

Sadržaj **iv**

Uvod **2**

1 Opis aplikacijskog okvira Vue.js **3**
1.1 Svojstva okvira . . . 3
1.2 Osnovni pojmovi i koncepti . . . 5
1.3 Vue Webpack i Vue CLI . . . 15
1.4 Direktive . . . 16
1.5 Komponente . . . 20
1.6 Rad s formama . . . 25
1.7 Filteri i Mixins . . . 27
1.8 Animacije i tranzicije . . . 31
1.9 HTTP zahtjevi i paket vue-resource . . . 32
1.10 Preusmjeravanje (eng. routing) . . . 36
1.11 Biblioteka Vuex i upravljanje stanjima . . . 40

2 Aplikacija Stock Trader **45**
2.1 Opis aplikacije . . . 45
2.2 Implementacija . . . 47

Bibliografija **53**

Uvod

Vue.js je aplikacijski okvir za izradu web-aplikacija.

Web-aplikacije su klijent-server programi napravljeni tako da se klijentski dio aplikacije otvara u pretraživaču. Najčešći primjeri web-aplikacija su: email aplikacije, aplikacije za online razmjenu poruka, online trgovine, itd.

Prije web-aplikacija klijentski dio klijent-server programa morao je biti individualno instaliran na svako klijentsko računalo. U tom smislu su web-aplikacije uvele preokret i sada je klijentska aplikacija dostupna s bilo kojeg računala koje ima pristup internetskoj mreži, bez prethodne lokalne instalacije. U ranim fazama, svaka web-aplikacija se na klijentskim računalima preuzimala kao statički dokument, gdje se interakcija s korisnikom ostvarivala kroz web-forme ugrađene u HTML (HyperText Markup Language). No svaka ozbiljnija promjena na stranici je zahtjevala kontaktiranje servera koji bi ponovno slao statički dokument (klijentsku aplikaciju) s novim stanjem. Godine 1995. Netscape je predstavio klijentski skriptni jezik JavaScript. On omogućava programerima dodavanje dinamičkih elemenata na stranice, te dinamičku izmjenu sadržaja bez osvježavanja. Nedugo nakon toga, Macromedia je predstavila platformu Flash, zasnovanu na vektorskoj animaciji, koja se ugradi u pretraživač i time omogućava djelovanje skriptnog jezika bez potrebe za kontaktiranjem servera. Tako kreće razvoj web-aplikacija koje su tek unazad desetak godina doživjele nagli razvoj i popularizaciju.

Aplikacijski okviri služe za ubrzanje razvoja web-aplikacija te gotovo svaki programski jezik ima svoj aplikacijski okvir. Najčešće imaju tri sastavna sloja – prezentacijski, aplikacijski (poslovna logika) i podatkovni.

Najvažnija svojstva koje moderna web-aplikacija treba imati su:

- *Progresivnost* – radi na svakom klijentskom računalu, kao da imamo tradicionalnu aplikaciju;
- *Responzivnost* – dobro se ponaša na ekranima svih razlučivosti, na različitim uređajima;
- *Konekcijska neovisnost* – ima ugrađene takozvane “radnike” koji omogućavaju rad offline;

- Daje osjećaj prave desktop aplikacije;
- Osvježava se bez korisnikove intervencije;
- *Sigurnost* – poslužuje se putem HTTPS protokola;
- Lako dostupna;
- Možemo postaviti kraticu na zaslonu koja će nas odvesti na web-lokaciju aplikacije;
- *Djeljivost* - postoji hiper-veza (eng. link) kojom možemo podijeliti aplikaciju s drugima.

U ovom radu ćemo opisati aplikacijski okvir skriptnog jezika JavaScript koji se zove *Vue.js*, a koji je dizajniran s ciljem što lakše izrade web-aplikacija s gore navedenim svojstvima. Rad se dijeli na tri dijela: uvod, opis aplikacijskog okvira Vue.js te opis pripadne prezentacijske aplikacije.

U uvodu smo objasnili što su to web-aplikacije.

U drugom poglavlju opisujemo aplikacijski okvir Vue.js te njegove najvažnije značajke i prednosti.

U zadnjem poglavlju opisujemo aplikaciju koju smo razvili, a koja demonstrira mogućnosti aplikacijskog okvira Vue kroz web-aplikaciju za trgovanje dionicama.

Poglavlje 1

Opis aplikacijskog okvira Vue.js

Vue.js je aplikacijski okvir za izradu web-aplikacija s MVC (Model-View-Controler) arhitekturom koji dopušta i uključivanje raznih dodatnih biblioteka. Pisan je skriptnim jezikom JavaScript. Izradio ga je *Evan You*, nakon rada s AngularJS okvirom u tvrtki Google. Prva verzija je objavljena 2014. godine.

Vue.js je progresivni aplikacijski okvir prvenstveno namijenjen izradi korisničkih sučelja, ali je od početka građen na način da ostavlja dovoljno prostora za nadogradnju i prilagodbu. Jezgrena biblioteka (eng. core library) se fokusira na prezentacijski sloj i lako se kombinira s drugim bibliotekama, pa i drugim projektima. Vue.js je pogodan za izradnju modernih SPA aplikacija (eng. single-page application).

SPA aplikacije su web-aplikacije s jednom stranicom (lokacijom) po kojoj se dinamički navigira i po potrebi mijenja sadržaj. Ovakve aplikacije su popularne zbog kompaktnosti i toga što klijentu pružaju osjećaj desktop aplikacija. Lokacija web-aplikacije se gotovo nikada ne osvježava. Osjećaj prave navigacije i mijenjanja linkova se postiže bibliotekama za preusmjeravanje.

1.1 Svojstva okvira

1. **Predlošci**
   Vue koristi sintaksu HTML predložaka u kojima se pomoću objekata vue-data direktno povezujemo s DOM (eng. document object model) strukturom koja se prikazuje u web-pregledniku. Vue predlošci su validan HTML koji se zatim pozadinski prevodi u virtualne DOM funkcije za prikaz. Zbog svog reaktivnog sustava, Vue zna odrediti minimalni broj DOM komponenti za koje je potreban ponovni prikaz pri promjeni stanja aplikacije. Vue, osim sintakse s predlošcima, dopušta i izravan unos JSX (JavaScript XML) funkcija.

2. **Reaktivnost**
   Jedna od istaknutih značajki okvira Vue je sustav reaktivnosti koji se svodi na čisti JavaScript. Modeli su obični JavaScript objekti. Kada mijenjamo modele, stranica se ažurira. To upravljanje stanjima čini jednostavnim i intuitivnim. Optimiziranost ažuriranja smo spomenuli u odjeljku o predlošcima – Vue komponente prate stanje svojih varijabli. Točno se zna kada treba ažurirati stanje stranice i točno koje komponente treba ažurirati.

3. **Komponente**
   Komponente su jedna od najvećih prednosti i najmoćnijih svojstava aplikacijskog okvira Vue. Vue dozvoljava gradnju prilagođenih i posebnih komponenata. Komponente proširuju obične HTML elemente kako bismo zapakirali kod za ponovnu upotrebu. Komponente su, u suštini, prilagođeni HTML elementi na koje vežemo specijalna ponašanja. Još jedan pogled na komponente jest da su one zapravo Vue instance s predefiniranim ponašanjem i opcijama.

4. **Tranzicije**
   Vue sadrži čitavu lepezu različitih tranzicijskih efekata kod umetanja, brisanja ili ažuriranja elemenata DOM strukture. U to se ubraja i:

   - Automatsko primjenjivanje klasa za CSS tranzicije i animacije;
   - Integracija vanjskih CSS animacijskih biblioteka, kao npr. animate.css ili velocity.css;
   - Korištenje JavaScripta za direktno manipuliranje DOM strukturom tijekom tranzicijskih okidača.

   Kada je element zapakiran u tranzicijsku komponentu te umetnut ili izbrisan, događa se sljedeće:

   a) Vue automatski provjerava postoji li tranzicijski ili animacijski CSS za dani element. Ako postoji, CSS tranzicijske klase će biti dodane/maknute kako je očekivano.
   b) Ako tranzicijska komponenta sadrži JavaScript okidače (eng. hooks), ti okidači će biti pozvani kako je očekivano.
   c) Ako nisu detektirani ni CSS ni JavaScript okidači za dani element, DOM operacije za umetanje i/ili brisanje će biti izvršene odmah sa sljedećim okvirom (okvir se odnosi na okvir pretraživača [4]).

5. **Preusmjeravanje**
   Generalna mana SPA aplikacija je nemogućnost dijeljenja poveznice do točne podstranice. To je zato što SPA šalje klijentu odgovor vezan za samo jedan URL, obično

index.html ili *indeks.vue*. Rješenje ovog problema je u front-end preusmjerivačima (eng. router) koji omogućuju definiranje umjetnih URL-ova baziranih na oznaci hash (#), na primjer: *page.com/#/* . Počevši sa standardom HTML5 najmoderniji pretraživači podržavaju preusmjeravanje bez upotrebe oznake hash. JavaScript biblioteke poput Vue imaju intuitivno sučelje (eng. interface) za mijenjanje prikazanog sadržaja stranice s trenutnim URL-om (neovisno o tome kako je sadržaj promijenjen). Dodatno, front-end preusmjeravanje dopušta namjerne promjene URL putanje kod događaja (eng. event) poput klika na gumb ili poveznicu. Vue sam po sebi ne dolazi s hash-baziranim front-end preusmjerivačem, ali biblioteka otvorenog koda *vue-router* pruža programsko sučelje (eng. API) za mijenjanje URL-a pretraživača, korištenje gumba 'Nazad' (povijest, eng. back-button), resetiranje email lozinke ili verifikaciju poveznice s autentifikacijskim parametrima iz URL-a. Podržano je i ugniježđeno mapiranje ruta do ugniježđenih komponenti te fino razrađena kontrola tranzicije. Izrada SPA aplikacije s front-end preusmjeravanjem uz Vue i vue-router nije složeno. Uz Vue, programeri komponiraju aplikaciju iz malih građevnih blokova, gradeći sve veću aplikaciju. Dodatno, uz vue-router, komponente treba samo mapirati na rutu kojoj pripadaju, te roditeljske (ili korijenske) rute moraju indicirati gdje će se renderirati djeca, što se čini pozicioniranjem posebnih tagova u HTML predlošku roditeljske komponente.

1.2 Osnovni pojmovi i koncepti

Pogodnosti Vue možemo koristiti dodajući skriptu s referencom na minimiziranu biblioteku u HTML, lokalnom instalacijom pomoću alata npm s kojim kontroliramo koje pakete ćemo instalirati, ili pomoću alata CLI koji prema predlošcima generira osnovni kostur i strukturu aplikacije s već instaliranim paketima[3].

```
1 var vm = new Vue({
2 // opcije: el, data, methods, computed...
3 })
```

Listing 1.1: Osnovna instanca Vue

Neke od opcija koje možemo definirati u instanci su:

- el: opcija veže instancu za HTML element unutar kojega će vrijediti pogodnosti okvira Vue

```
1 var vm = new Vue({
2   el: '#app'
3 })
```

U primjeru se instanca veže na element s identifikatorom "id='app'".

- data: opcija definira instanci skup varijabli s kojima će raditi.

```
var vm = new Vue({
  el: '#app',
  data: {
    varijabla1: 'vrijednost',
    varijabla2: 0
  }
})
```

- methods: opcija u kojoj definiramo sve metode koje ćemo koristiti u aplikaciji
- computed: opcija u kojoj definiramo kraće metode (tzv. izračunata svojstva) koje često koristimo
- components: opcija u kojoj navodimo tagove i imena posebnih komponenti koje smo sami definirali

```
var vm = new Vue({
  el: '#app',
  data: {
    varijabla1: 'vrijednost',
    varijabla2: 0
  },
  methods: {
    someMethod() { ... }
  },
  computed: {
    insufficientQuantity() {
      return this.varijabla2 > 5;
    }
  },
  components: {
    'noviTag': novaKomponenta
  }
});
```

Listing 1.2: Instanca Vue s opcijama

Vue uvodi dvije pokrate koje se vrlo često koriste, pa ih navodimo:

- Kod definiranja događaja na HTML elementu '@' je ekvivalentno s 'v-on'.

```
<input v-on:click="metoda"> <!-- isto kao -->
<input @click="metoda">
```

- Kod povezivanja vrijednosti atributa u oba smjera ':' je ekvivalentno s 'v-bind'.

```
<input v-bind:href="url"> <!-- isto kao -->
<input :href="url">
```

Također, objasnimo na primjeru značenje interpolacije stringa. Na primjer:

```
<p> {{ url }} </p>
```

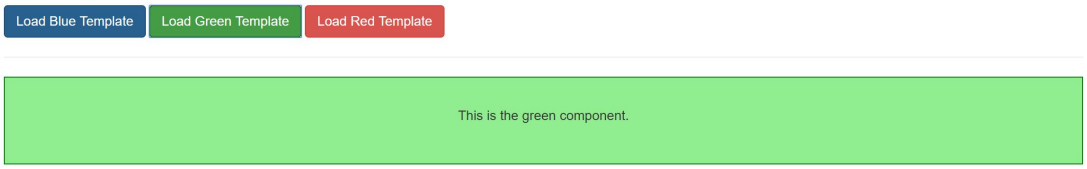
će ispisati pravu vrijednost varijable 'url'. Zbog dvostrukih vitičastih zagrada Vue zna da je izraz unutar zagrada neko data svojstvo instance Vue ili rezultat neke metode pa se taj izraz u pozadini naprije izvrijedni i tek onda se prikazuje vrijednost tog izraza.

Pojasnimo na primjeru jednostavne aplikacije dosad spomenute pojmove pa ćemo ih kasnije detaljnije obraditi.

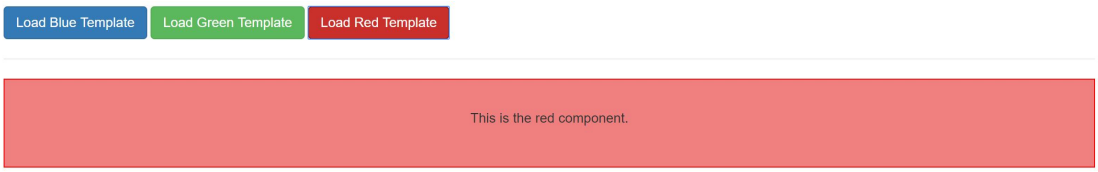
Aplikacija se sastoji od glavne app komponente s HTML predloškom te tri jednostavne prilagođene komponente koje se dinamički izmjenjuju na ekranu na događaj pritiska gumba. Instanca objekta klase Vue je definirana u JavaScript datoteci main.js, gdje smo naveli na koji element će Vue imati utjecaj. U našem primjeru je to element s id oznakom 'app'.

Prikaz ekrana nakon pritiska plavog gumba

Prikaz ekrana nakon pritiska zelenog gumba



Prikaz ekrana nakon pritiska crvenog gumba



Komponenta plave boje (Blue.vue) se učitava odmah po učitavanju aplikacije ili na pritisak plavog gumba. Komponenta zelene boje (Green.vue) se učitava na pritisak zelenog gumba, a komponenta crvene boje (Red.vue) na pritisak crvenog gumba. Opisat ćemo implementaciju spomenutih komponenti i mehanizam njihove izmjene na ekranu.

Prikažimo datoteke s JavaScript kodom gdje se vidi da je novi objekt Vue instance registriran na element s id oznakom 'app', te osnovni HTML kod aplikacije u kojem se nalazi element tipa div s oznakom 'app'.

```
import Vue from 'vue'
import App from './App.vue'

new Vue({
  el: '#app',
  render: h => h(App)
})
```

Listing 1.3: Main.js datoteka

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Vue Components</title>
    <link rel="stylesheet" href="https://maxcdn.cdn.com
    /bootstrap/3.3.7/css/bootstrap.min.css">
  </head>
  <body>
    <div id="app">
    </div>
    <script src="/dist/build.js"></script>
  </body>
</html>
```

Listing 1.4: index.html datoteka

U datoteci main.js definiramo tj. registriramo glavni objekt Vue instance koji se veže za element s id oznakom 'app' te nam omogućava korištenje Vue pogodnosti i funkcija unutar tog elementa.

Nadalje, svaka validna Vue datoteka se sastoji od tri karakteristična dijela: template, script i style dijela.

Template dio se navodi između `<template></template>` tagova, a sadrži HTML kod koji želimo pridružiti našoj komponenti. Jedino ograničenje ovog dijela je da sav HTML kod mora biti zapakiran u jedan korijenski element, u praksi najčešće div element.

Script dio se navodi između `<script></script>` tagova i sadrži sav JavaScript kod vezan uz našu komponentu i njezin HTML predložak: varijable, defincije događaja, itd.

Style dio se navodi između `<style></style>` tagova i sadrži CSS kod koji se definira kao globalni, osim u slučaju navođenja atributa *scoped* unutar style taga - tada se navedeni CSS stil primjenjuje samo na HTML kod u predlošku iste datoteke.

Komponente aplikacije te potrebni događaji su registrirani u datoteci App.vue, u pripadajućem objektu Vue, koji se izvozi (eng. export) te spaja s objektom Vue iz datoteke main.js.

```
<script>
  import Blue from './components/Blue.vue';
  import Green from './components/Green.vue';
  import Red from './components/Red.vue';

  export default {
    data: function () {
      return {
        selectedComponent: "Blue"
      }
    },
    components: {
      appBlue: Blue,
      appGreen: Green,
      appRed: Red
    }
  }
</script>
```

Listing 1.5: App.vue - JavaScript

U JavaScript kodu uvozimo (eng. import) datoteke s prilagođenim komponentama kako bismo ih mogli registrirati unutar objekta Vue instance. Komponente definiramo u objektu *components* u obliku ključ-vrijednost (npr. 'appBlue: Blue', gdje appBlue označava jedinstveni HTML tag za komponentu Blue). U datoteci App.vue definiramo i objekt data sa svojstvom *selectedComponent* kojem vrijednost postavljamo na "Blue", kako bi se po učitavanju aplikacije na odgovarajućem mjestu odmah prikazala plava komponenta. Svojstvo *selectedComponent* je ključno za našu aplikaciju jer se upravo njegova vrijednost mijenja na pritisak gumba i time direktno utječe na promjenu komponente koja se prikazuje.

Kada u konstruktoru za novu Vue instancu prosljeđujemo objekt sa el, data, methods i drugim objektima Vue kopira taj objekt te postavlja *watcher* za svako svojstvo u objektu data. Tako zna kada se koje svojstvo promijenilo i koji dio treba ažurirati. Postavljanje *watcher*-a se događa samo jednom i to je uvijek kroz konstruktor instance. Dakle, mi možemo naknadno definirati neko novo svojstvo za instancu, no tada ono neće biti reaktivno jer se za njega nije postavio *watcher*. *Watcher*-i se postavljaju samo i jedino za one varijable koje su inicijalno poslane konstruktoru u svojstvu data.

Nadalje, jednom kada je Vue instanca definirana, imamo neka njezina svojstva koja valja dodatno istaknuti. To su:

- **$el** - sadržaj HTML komponente na koju je vezana Vue instanca;
- **$data** - omogućava pristup objektu *data* i njegovim atributima;
- **$refs** - pomoću tog svojstva dolazimo do svih DOM elemenata u dosegu instance koji na sebi imaju *ref* tag. Mijenjajući takve elemente mijenjamo direktno DOM strukturu (ne Vue template koji se generira). To za posljedicu ima da prvi puta kada se tako izmjenjeni element ažurira, sve naše promjene će nestati. Vue ne ažurira direktno DOM već prati promjene varijabli i ponovno povlači template koji smo definirali. Svojstvo je, međutim, korisno za označavanje elemenata koje kasnije želimo dohvaćati;

Napomena 1.2.1. *Sva svojstva i metode s prefiksom $ su nativna Vue svojstva, odnosno metode. Nativna svojstva i metode pripadaju jezgrenoj biblioteci Vue i mogu se koristiti bez prethodne registracije ili definicije.*

```
<template>
  <div class="container">
  <div class="row">
    <div class="col-xs-12">
      <br>
      <button class="btn btn-primary" @click="selectedComponent='Blue'">
        Load Blue Template</button>
      <button class="btn btn-success" @click="selectedComponent='Green'"
   >Load Green Template</button>
      <button class="btn btn-danger"
      @click="selectedComponent='Red'">Load Red Template</button>
      <hr>
      <component :is="'app'.concat(selectedComponent)">
        <p> This is the
        {{selectedComponent.toLowerCase()}} component.</p>
      </component>
    </div>
  </div>
  </div>
</template>
```

Listing 1.6: App.vue - predložak

Predložak aplikacije se sastoji od div elementa s klasom *col-xs-12* koja zapravo djeluje na responzivnost tako što govori kako da se aplikacija ponaša na malim ekranima - taj element se treba protezati od lijevog do desnog ruba roditeljskog elementa. Ova, kao i sve

ostale klase pridružene tagovima u gornjem predlošku dolaze iz biblioteke Bootstrap[2]. Dalje se navode tri gumba, od kojih svaki mijenja sadržaj svojstva *selectedComponent* iz skupa varijabli (data) Vue instance te implicitno mijenja prikaz odgovarajuće komponente na ekranu. Ispod gumba se nalazi tag component, Vue-ov ugrađeni tag za komponentu. Tag component je u pozadini obrađen tako da Vue zna i očekuje kod posebne komponente unutar njega. U našem primjeru unutar component taga se nalazi p tag u kojem imamo izraz {{`selectedComponent.toLowerCase()`}}. Ranije smo spomenuli da ovaj postupak zovemo interpolacija stringa. Vrijednost svojstva *selectedComponent* najprije se transformira u riječ pisanu malim tiskanim slovima te se potom konkatenira s ostatkom rečenice.

Pomoću vezanog atributa *:is* Vue zna koju komponentu treba prikazati. S desne strane jednakosti atributa *:is* se navodi ime taga registrirane komponente koju želimo prikazati. U našem primjeru se vrijednost vezanog atributa *:is* izvrijednjuje tako da se spaja riječ 'app' s vrijednošću svojstva *selectedComponent*. Primjetimo da će pritiskom npr. zelenog gumba svojstvo *selectedComponent* poprimiti vrijednost 'Green', te će se atribut *:is* izvrijedniti u riječ 'appGreen' što je upravo ime taga kojem smo u objektu Vue instance pridružili komponentu Green i na ekranu će nam se zaista prikazati zelena (Green) komponenta.

Komponentu je moguće promijeniti uz pomoć JavaScripta tako da u kodu promijenimo vrijednost svojstva *selectedComponent*. To rezultira promjenom komponente na ekranu, jednako kao i pritisak gumba. To je primjer reaktivnosti. Imamo svojstvo koje utječe na prikaz komponente, i nebitno je kako se promijenila vrijednost danog svojstva, akcijom korisnika u sučelju ili pozadinski u JavaScript kodu, dogodit će se očekivana promjena.

U slučaju da ne želimo na istom mjestu ekrana izmjenjivati različite komponente nego ih želimo prikazivati sve u isto vrijeme dovoljno je samo u HTML predlošku navesti prazne tagove s kojima smo registrirali dane komponente u objektu Vue instance. U našem primjeru se unutar samog component taga nalazi paragraf (p) tag koji će se prikazivati u svakoj od naših komponenti. To je zgodna mogućnost koju Vue pruža, a omogućeno je slot tagom unutar HTML predloška komponente.

Slot je poseban tag koji nam Vue pruža kako bismo ugniježđenoj komponenti proslijedili HTML kod iz roditeljske komponente. On je u pozadini obrađen na način da se kod učitavanja HTML koda komponente slot tag zamijeni odgovarajućim HTML kodom iz roditeljske komponente, a koji se nalazi između tagova `<component></component>`, bez obzira što je sam HTML kod naveden u roditeljskoj komponenti.

Za kraj ostavljamo style dio App.vue datoteke. Nemamo definiran poseban stil pa style tag ostavljamo praznim.

```
1  <style></style>
```

Listing 1.7: App.vue - CSS

Pogledajmo sada datoteke prilagođenih komponenti:

```
<template>
  <div>
    <slot></slot>
  </div>
</template>

<script></script>

<style scoped>
  div {
    border: 1px solid blue;
    background-color: lightblue;
    padding: 30px;
    margin: 20px auto;
    text-align: center
  }
</style>
```

Listing 1.8: Blue.vue

Kao što vidimo, predložak se sastoji samo od korijenskog div elementa i unutar njega tag slot. Slot nam omogućava da se p element definiran unutar component taga predloška App.vue prikazuje unutar komponente Blue.vue (dok je komponenta aktivna, tj. dok se prikazuje). U našem slučajnu je takvo prosljeđivanje HTML koda posebno zgodno svojstvo jer nam ne treba prosljeđivanje parametara u Blue.vue komponentu. Mijenjanjem aktivne komponente na ekranu automatski se mijenja i poruka unutar p taga - ovisno o vrijednosti svojstva *selectedComponent*.

Također, imamo definiran stil za danu komponentu koji dolazi unutar taga style s atributom *scoped*. Podsjetimo se, atribut *scoped* nam kaže da je navedeni stil primjenjiv samo na danu komponentu. Da smo stil definirali bez atributa *scoped* stil bi se primjenjivao globalno, odnosno bio bi primjenjiv na sav HTML kod aplikacije, bez obzira kojoj komponenti pripada.

```
<template>
  <div>
    <slot></slot>
  </div>
</template>

<script></script>

<style scoped>
  div {
    border: 1px solid green;
    background-color: lightgreen;
    padding: 30px;
```

```
    margin: 20px auto;
    text-align: center
  }
</style>
```

Listing 1.9: Green.vue

```
<template>
  <div>
    <slot></slot>
  </div>
</template>

<script></script>

<style scoped>
  div {
    border: 1px solid red;
    background-color: lightcoral;
    padding: 30px;
    margin: 20px auto;
    text-align: center
  }
</style>
```

Listing 1.10: Red.vue

Datoteke Green.vue i Red.vue su ekvivalentne s Blue.vue do na definiciju stila. Tri komponente razlikuju se u definiciji pozadinskog obojenja, tj. svaka će obojati element u pripadajuću pozadinsku boju. Primjetimo da u našoj aplikaciji imamo samo jednu Vue instancu kojom kontroliramo izmjenu tri jednostavne komponente.

Teoretski je moguće i sasvim u redu koristiti više instanci, no treba biti pažljiv jer iz svake instance pristupamo samo lokalno definiranim vrijednostima objekta *data*.

Instancama se može pristupiti i izvana, pod uvjetom da smo definiciju instance pospremili u varijablu, na primjer:

```
var vm = new Vue({
  el: '#app',
  data: {
    title: 'New Vue instance'
  }
});
```

Tada instancom možemo manipulirati kao bilo kojim JavaScript objektom. Možemo joj pristupati iz neke druge instance ili s bilo kojeg mjesta u JavaScript kodu, jednostavno pristupajući varijabli kojoj smo pridružili definiciju instance:

```
setTimeout(function() {
  vm.title = 'Updated title';
}, 2000);
```

Navedeni isječak koda će direktno izmijeniti vrijednost svojstva *title* iz gore registriranog objekta data jer smo mu pristupili pomoću varijable vm u koju smo pospremili objekt Vue instance.

Ako nismo sigurni gdje ćemo učitavati instancu ili se HTML kod za danu instancu naknadno dinamički generira, možemo samu instancu kreirati bez svojstva *el* te je naknadno povezati s HTML kodom uz pomoć nativne metode *mount*:

```
vm.$mount('#app');
```

što je ekvivalentno definiranju

```
el: '#app'
```

unutar konstruktora.

U konstruktoru instance možemo definirati i svojstvo *template* gdje navodimo HTML kod koji želimo prikazati u svojoj komponenti, no to je dosta ograničavajuće obzirom da je teško pisati višelinijski kod u običnom stringu. Obično se ne koristi, eventualno za jednostavne jednolinijske kodove.

Vue instance nisu pogodne za koncept ponovne upotrebe, jer se instanca veže za prvi element s definiranim selektorom, a svi pripadajući selektori iza se ignoriraju. Dakle, ako za *el* svojstvo u konstruktoru instance postavimo selektor *'.app'* tada će Vue imati utjecaj samo na prvi DOM element s klasom *app* bez obzira koliko ih još ima nakon njega.

Ono što se u Vue okviru koristi za koncept ponovne upotrebe jesu *komponente*. Njima zadajemo tag koji Vue prepoznaje kao identifikator dane komponente, i koliko god puta ga navedemo u HTML kodu, toliko puta će se prikazati naša komponenta.

Što se tiče osvježavanja prikaza, prilikom promjene modela Vue koristi virtualni DOM. JavaScript je brz sam po sebi, no pristup DOM-u je prilično spor i bilo bi sporo kada bi Vue predložak zbog malih promjena generirao cijeli novi DOM. Zato Vue radi s dodatnim slojem (eng. layer) koji je zapravo kopija DOM-a, to jest virtualni DOM, koji je parsirani JavaScript i brzo mu se pristupa. On stalno prati promjene i uspoređuje Vue instancu s virtualnim DOM-om te u izvornom (eng. native) DOM-u mijenja samo one elemente gdje je uočio razlike umjesto da generira cijeli novi DOM. To Vue čini vrlo brzim.

Napomena 1.2.2. *Nativna metoda $destroy() uništava svu JavaScript logiku vezanu za komponentu. Ne uništava DOM strukturu.*

1.3 Vue Webpack i Vue CLI

Vue.js dolazi u dva osnovna oblika: s ugrađenim kompajlerom i bez njega. Ako želimo raditi sa string predlošcima (eng. template) i dinamički učitavati (eng. mount) komponente tada nam treba verzija s ugrađenim kompajlerom. U protivnom, ako koristimo definirane predloške dovoljan nam je manji paket bez ugrađenog kompajlera. Manji je za 30%, a kompajler je dostupan na razvojnom serveru (eng. development server). Također, razvoj pomoću razvojnog servera nam daje neke dodatne mogućnosti poput pisanja koda u EcmaScript6 formatu, a serverov transpiler (Babel) će ga prevoditi u EcmaScript5 kako bi se sadržaj prikazivao u svim pretraživačima. Razvojni server nam omogućuje i *auto-reload* svojstvo. To znači da se promjene, u kodu, na ekranu vide odmah po učinjenim promjenama.

Jedna od najvećih prednosti korištenja razvojnog servera je upravo to što se aplikacija poslužuje preko njega, odnosno dohvaćamo je u svoj pretraživač sa servera. To znači da se ništa ne kompajlira lokalno i to nam omogućava lakši razvoj i bolju robusnost, bez obzira na računalo i pretraživač koji koristimo. Jedan od alata koji možemo koristiti za lakši početak gradnje aplikacije je *Vue CLI*. Vue CLI ima jednu značajnu prednost i glavni zadatak, a to je pristup *VueJS Project Templates*. Dohvaća nam, ovisno o tome s kojim parametrima ga pokrećemo, prazan ali inicijalno postavljen i posložen projekt. Instaliramo ga pomoću:

```
npm install -g vue-cli
```

Npm stoji za *node package manager*, i da bi ispravno radio trebamo prethodno instalirati *Node.js*. *Node.js* nam nije potreban za razvoj same Vue aplikacije, ali npm paket ga koristi. Zastavica '-g' stoji kao oznaka da se vue-cli instalira globalno na našem računalu. Na Linuxu ili MacOS-u ćemo za instalaciju vue-cli možda trebati administratorska prava i gore navedenoj naredbi dodati prefix sudo. Možemo birati između različitih predložaka (eng. template), neki od njih su:

- *simple* - index.html + Vue CDN import;
- *webpack-simple* - osnovni Webpack tijek rada (eng. workflow);
- *webpack* - složeni Webpack tijek rada;
- *browserify/ browserify-simple* - Browserify tijek rada, vrlo slično kao i Webpack.

Nakon instalacije *Vue-CLI* i odabira predloška, inicijaliziramo projekt i unutar njega instaliramo potrebne pakete pomoću *npm*, naredbama u bash ljusci kako slijedi:

```
1 vue init webpack-simple vue-cli
2 cd vue-cli
3 npm install
```

Recimo da smo odabrali *webpack-simple* kao predložak, i mapu u kojoj se inicijalizira projekt smo nazvali *vue-cli*.

Nakon uspješne instalacije npm paketa, možemo pokrenuti razvojni server naredbom:

```
npm run dev
```

Razvojni server će automatski našu aplikaciju napraviti dostupnom na URL *http://localhost:8080*.

Ukoliko želimo pripremiti aplikaciju za objavljivanje i postavljanje na javno dostupni server, naredba

```
npm run build
```

će minimizirati i optimizirati paket za takvu akciju.

Nakon što smo na jednostavnom primjeru objasnili osnovne elemente Vue aplikacije, u narednim sekcijama ćemo detaljnije opisati svaki od njih.

1.4 Direktive

Direktive su posebne oznake koje na DOM element vežu specifične predefinirane akcije, ovisno o tome koju direktivu koristimo. Postoje gotove Vue direktive: *v-text, v-html, v-model, v-show, v-bind*, itd. Kao što je vidljivo, direktive uvijek započinju s "v-" pa neka ključna riječ.

Direktiva *v-show* spada pod uvjetne direktive i koristi se za prikaz/sakrivanje DOM elementa ovisno o vrijednosti neke boolean varijable. Primjer:

```
<button v-show="uvjet">Gumb</button>
```

Gumb će se prikazivati ako je stanje varijable uvjet jednako 'true',odnosno bit će sakriven ako je vrijednost varijable 'false'. Još jedna takva direktiva je *v-if*. Razlika je u tome da *v-if* direktiva dodaje ili miče element iz DOM strukture dok *v-show* samo postavlja CSS svojstvo *display*, dakle ne mijenja DOM strukturu dokumenta.

Direktiva *v-bind* služi za povezivanje vrijednosti HTML atributa. To znači da će, u našem primjeru, atribut src poprimiti vrijednost varijable imageSrc iz skupa varijabli (data) Vue instance.

```
<img v-bind:src="imageSrc">
```

Direktiva *v-text* služi za prosljeđivanje teksta elementu.

```
<span v-text="msg"></span> <!-- isto kao -->
<span> {{ msg }} </span>
```

Direktiva *v-html* prosljeđuje elementu string koji postavlja za vrijednost innerHTML-a.

```
<div v-html="html"></div>
```

Direktiva *v-for* služi za prikazivanje elemenata niza/liste.

```
<div v-for="item in items">
  {{ item.text }}
</div>
```

Direktive, kao i komponente, možemo sami stvarati te ih definirati i globalno i lokalno. Globalno ih definiramo pomoću

```
Vue.directive('imeKomponente',{ konfiguracija });
```

u datoteci main.js. Slično kao kod komponenti, samo se u prvom argumentu ne prosljeđuje ime taga već ime komponente na koju se direktiva veže.

Direktive se mogu definirati i lokalno. Tada su dostupne samo u komponenti u kojoj su definirane, a definiraju se u svojstvu *directives*.

Postoji pet različitih okidača (eng. hooks) koji su zapravo poput metoda životnog ciklusa:

1. bind(el, binding, vnode) – ova metoda se pokreće čim je direktiva vezana na element.

   el označava element na koji smo vezali direktivu, *binding* označava način na koji smo vezali direktivu (tu određujemo koje varijable i/ili modifikatore možemo prosljeđivati), dok *vnode* označava čvor u virtualnom DOM-u i rijetko se koristi. Vnode bi se trebao koristiti samo za čitanje jer sadrži informacije koje Vue kontrolira i ne bismo ih trebali mijenjati pri izvođenju (eng. runtime).

2. inserted(el, binding, vnode) – pokreće se kad se element na koji smo vezali direktivu kreira u DOM roditeljskom elementu.

3. update(el, binding, vnode, oldVnode) – pokreće se kada ažuriramo komponentu, bitno je naglasiti da se ta metoda poziva kada potomci te komponente nisu još ažurirani. Dodatni argument *oldVnode* sadrži prethodnu vrijednost čvora, budući je aktualna ažurirana na *vnode*. Obje ove vrijednosti su dostupne samo za čitanje.

4. componentUpdated(el, binding, vnode, oldVnode) – pokreće se kada je ažurirana i komponenta i njezini potomci.

5. unbind(el, binding, vnode) – pokreće se kada se direktiva ukloni.

Bind i update se u praksi najčešće koriste. Primjer jednostavne direktive koja samo dodaje elementu pozadinsku boju:


```
<div id="app">
  <p>Text</p>
  <p v-highlight>Some other text</p>
</div>
```

Listing 1.11: HTML

```
Vue.directive('highlight', {
  bind(el,binding,vnode) {
    el.style.background = 'green';
  }
});

new Vue({
  el: '#app'
});
```

Listing 1.12: Vue

Primjer prosljeđivanja vrijednosti direktivi:


```
<div id="app">
  <p>Text</p>
  <p v-highlight="'red'">Some other text</p>
</div>
```

Listing 1.13: HTML

```
Vue.directive('highlight', {
  bind(el,binding,vnode) {
    el.style.background = binding.value;
  }
});

new Vue({
el: '#app'
});
```

Listing 1.14: Vue

Vrijednost koja se nalazi između dvostrukih navodnika kod navođenja direktive sprema se u varijablu *binding.value*. Vrijednost može biti u JSON obliku gdje onda dalje pristupamo atributima objekta. Na taj način možemo proslijediti i više različitih vrijednosti istoj direktivi. Ako kroz vrijednost želimo poslati ime funkcije koju želimo izvršiti, tada toj funkciji pristupamo s *binding.value()* – točno kao da pozivamo tu funkciju.

Primjer prosljeđivanja argumenta direktivi:


```
<div id="app">
  <p>Text</p>
  <p v-highlight:background="'blue'">Some other text</p>
</div>
```

Listing 1.15: HTML

```
Vue.directive('highlight', {
  bind(el,binding,vnode) {
    if (binding.arg == 'background') {
      el.style.background = binding.value;
    }
  }
});

new Vue({
el: '#app'
```

```
11 });
```

Listing 1.16: Vue

Vrijednost argumenta koji prosljeđujemo sprema se u varijablu *binding.arg*. Argumente prosljeđujemo obično u slučajevima kada želimo izvršiti poseban događaj u slučaju postojanja tog argumenta. U slučaju našeg primjera sa slike, da nemamo naveden argument *background* boja pozadine se ne bi mijenjala u plavu.

Primjer prosljeđivanja modifikatora:

```
1 <p v-highlight:background.delayed = "'blue'">Some other text</p>
```

```
1  Vue.directive('highlight', {
2    bind(el,binding,vnode) {
3      var delay = 0;
4      if (binding.modifiers['delayed']) {
5        delay = 2000;
6      }
7      setTimeout(() => {
8        if (binding.arg == 'background') {
9          el.style.background = binding.value;
10       }
11     }, delay);
12   }
13 });
```

Primjer prikazuje kako se u prisustvu modifikatora 'delayed' mijenjanje pozadinske boje odgađa za 2000 milisekundi. Bez navedenog modifikatora nema odgode. Modifikatora možemo dodati više i svi se čuvaju u nizu *binding.modifiers*.

1.5 Komponente

Komponente su zapravo nove Vue instance, s proširenim mogućnostima. Na komponente možemo gledati kao na prilagođene dijelove HTML koda koje sami pišemo i koji se mogu opetovano koristiti na različitim mjestima. Vue ih koristi za koncept ponovne upotrebe. Kako komponente jesu Vue datoteke imaju i karakterističnu strukturu - *template*, *script* i *style*.

Komponente se mogu registrirati kao globalne ili lokalne. Globalno ih registriramo pomoću funkcije *Vue.component()* u datoteci main.js i iznad objekta Vue instance, dok ih lokalno registriramo tako da definiramo komponentu kao novi objekt u zasebnoj datoteci te u objektu Vue instance registriramo komponentu u svojstvu components.

```
Vue.component('click-counter', {
  data: function () {
    return {
      count: 0
    }
  },
  template: '<p v-on:click="count++">You clicked me {{ count }}
   times.</p>'
});

new Vue({...});
```

Listing 1.17: Globalna registracija komponente

Komponente proširuju Vue instancu pa im *data* objekt trebamo proslijediti umjesto da ga deklariramo lokalno. Data treba biti u obliku funkcije koja će vraćati objekt s varijablama koje nam trebaju, kako bi svaka instanca komponente koju definiramo u kodu imala nezavisnu kopiju iste varijable.

```
export default {
  data() {
    return {
      quantity: 0
    }
  }
}
```

Listing 1.18: Primjer objekta data kao funkcije

Ako proslijeđujemo predefinirani *data* objekt komponenti, treba biti svjestan da će sve instance iste komponente pristupati istoj memorijskoj lokaciji. Dakle, ako u jednoj komponenti promijenimo stanje neke varijable, promijenit će se u svim komponentama s istom oznakom.

Komunikacija među komponentama

Komunikacija među komponentama je bitna za aplikacije gdje više komponenata koristi ili prikazuje iste podatke te je stoga potrebno da podaci cirkuliraju kroz aplikaciju.

1. **Proslijeđivanje parametara u smjeru roditelj <-> potomak:**

   Služi za prosljeđivanje vrijednosti varijable roditeljske komponente ugniježđenoj komponenti koja zatim tu vrijednost sprema u svoju lokalnu varijablu, s kojom dalje radi. Varijabla za proslijeđenu vrijednost se definira u posebnom svojstvu *props* u objektu instance ugniježđene komponente. Roditeljska komponenta samu vrijednost šalje navodeći ime vezane varijable u tagu ugniježđene komponente. Primjer:

`<moja-komponenta vezanaVarijabla="name"></moja-komponenta>`. Varijabla koju prenosimo u ugniježđenu komponentu (moja-komponenta) je varijabla "name" iz data objekta. U objektu Vue instance ugniježđene komponente trebamo registrirati vezanu varijablu, kako je navedeno:

```
export default {
  props: ['vezanaVarijabla'],
  data() {
    return {
      ...
    }
  }
}
```

```
Vue.component('textit', {
  props: ['title'],
  data: function () {
    return {
      count: 0
    }
  },
  template: '<h1> {{ title }} </h1>'
});

new Vue({
  el: '#app',
  data: {
    text: 'NASLOV'
  }
});
```

Listing 1.19: Prosljeđivanje parametara pomoću svojstva props

```
<div id='app'>
  <textit title="text"></textit>
</div>
```

Ako, na primjer, u ugniježđenoj komponenti promijenimo vrijednost varijable 'title' i želimo to javiti roditeljskoj komponenti, onda koristimo nativnu metodu *$emit('imeDogađaja', vrijednost);*.

```
Vue.component('textit', {
  props: ['title'],
  data: function () {
    return {
      count: 0
    }
  },
```

Slika 1.1: Izgled komponente

```
  methods:{
    change() {
      this.title="Novi naslov";
      this.$emit('changeTitle', this.title);
    }
  },
  template: '<h1 v-on:click="change"> {{ title }} </h1>'
});
```

Listing 1.20: Prosljeđivanje vrijednosti primitivnih tipova roditeljskoj komponenti - ugniježđena komponenta

U roditeljskoj komponenti na proslijeđenu akciju reagiramo metodom koja se zove jednako kao ime događaja koje smo proslijedili u emit metodi ugniježđene komponente.

```
new Vue({
  el: '#app',
  data: {
    text: 'NASLOV'
  },
  methods: {
    changeTitle(event) {
      this.text = event;
    }
  }
});
```

Listing 1.21: Prosljeđivanje vrijednosti primitivnih tipova roditeljskoj komponenti - roditeljska komponenta

2. **Komunikacija između komponenata iste razine:**

   Komunikacije između ugniježđenih komponenti iste razine se odvija posredstvom roditeljske komponente. Proslijeđeni podaci iz polazne ugniježđene komponente

se prosljeđuju u roditeljsku komponentu pa se automatski ažuriraju sve ostale ugniježđene komponente kojima roditeljska komponenta prosljeđuje iste podatke.

Vue ima i podršku za prosljeđivanje teksta ili HTML koda komponentama. To možemo činiti pomoću rezerviranog taga slot, kao što je ranije spomenuto.

No gdje se slot kompajlira i kako se primjenjuju stilovi na tako proslijeđen dio koda? Svi stilovi koje želimo primijeniti na dio koda koji prosljeđujemo treba definirati u scoped style tagu u ugniježđenoj komponenti, bez obzira što je sam HTML kod napisan u roditeljskoj komponenti. Sve ostalo vezano za proslijeđeni komad koda, osim stila, kontroliramo iz roditeljske komponente. To je važno svojstvo okvira Vue. Kod tako proslijeđenog koda radi čak i interpolacija stringova iz roditeljske komponente, što smo vidjeli na primjeru jednostavne aplikacije. Također je podržano slanje višestrukih slot tagova istoj ugniježđnoj komponenti. No, ako želimo slotove napuniti drugačijim sadržajima, moramo im pridodati imena. Pri pisanju koda u roditeljskoj komponenti treba kod svakog taga navesti kojem slotu pripada. Ako se ne navede ime slota tag će se pridodati predefiniranom slotu, tj. onom bez navedenog imena. Ako se koristimo slotovima unutar predloška komponente potomka, sadržaj između `<slot></slot>` tagova će se uredno prikazivati.

Dostupne su i takozvane dinamičke komponente. Podržano je dinamičko mijenjanje komponenti ovisno o nekom ispunjenom uvjetu, npr. ovisno o pritisku gumba. To je omogućeno pomoću rezerviranog taga

```
<component :is="selectedComponent"></component>
```

Ako se u *selectedComponent* mijenjaju imena tagova komponenti, tada će se automatski renderirati komponenta s danim tagom. Kod takve dinamičke izmjene komponenti one se sa svakom izmjenom uništavaju i ponovno stvaraju svaki puta kod novog prikaza. Međutim, to ponašanje možemo promijeniti tako da cijeli `<component></component>` tag umotamo u još jedan tag `<keep-alive></keep-alive>`. Sada se komponente pri izmjeni neće uništavati nego se njihovo stanje čuva u pozadini i svaki puta s prikazivanjem komponente vidjet ćemo stanje isto kao prije prvotne promjene.

Napredni rad s komponentama nam omogućava i dva nova životna ciklusa, *deactivated()* i *activated()*. To je posebno zgodno baš u slučaju dinamičke izmjene komponenti kada se komponente ne uništavaju. Možemo točno kontrolirati što će nam se dogoditi ako neku komponentu deaktiviramo i/ili aktiviramo.

Napomena 1.5.1. *Dobra praksa je razdvojiti Vue datoteke u mape radi bolje strukture te preglednosti. Obično se sve komponente migriraju u mapu components. Dalje se opet mogu napraviti pod-mape u kojima ćemo organizirati Vue datoteke po tome kojoj komponenti / sučelju pripadaju ili nečem sličnom.*

1.6 Rad s formama

U ovom odjeljku želimo naglasiti jednostavnost rada s formama u okviru Vue, te ukazati na nekoliko zanimljivih pristupa formama.

Vue podržava modifikatore događaja (eng. event modifiers) i modifikatore tipki (eng. key modifiers) izravno u HTML tagovima. Primjer:

```
1 <input @keyup.enter="mojaMetoda">
```

Primjer prikazuje modifikator tipki, pobliže modifikator koji se pokreće otpuštanjem razmaknice. U danom primjeru želimo da se na akciju otpuštanja razmaknice izvrši metoda koju smo nazvali "mojaMetoda".

Primjeri modifikatora događaja:

- .stop - sprječava propagaciju to jest normalno ponašanje te će se izvršiti dana metoda;
- .prevent - sprječava osvježavanje stranice;
- .self - izvrši danu metodu samo ako se događaj dogodio na tom istom elementu, obično se upotrebljava uz događaj klika;
- .once - izvrši metodu samo jednom;

Primjeri ugrađenih modifikatora tipki:

- .enter;
- .tab;
- .delete -podrazumijeva i tipku delete i tipku backspace;
- .esc;
- .space;
- .up;
- .down;
- .left;
- .right;
- .alt;

- .ctrl;

Za ostale tipke koristimo uobičajene kodove. Na primjer, slovo c ima kod 67 pa bi primjer modifikatora na pritisak slova c bio: `<input @keydown.67="metoda">`.

U formama koristimo direktivu *v-model* za povezivanje podataka (eng. data binding) u oba smjera. Jednostavnim rječnikom rečeno, ako neku varijablu koja se prikazuje na sučelju promijenimo direktnom interakcijom sa sučeljem (npr. input polje) tada će se odgovarajuća povezana varijabla ažurirati u pozadini. Obratno, ako se varijabla promijeni u pozadini, odgovarajuća promjena će biti vidljiva i na sučelju. Imamo četiri moguća modifikatora za v-model:

- default - Vue reagira na svaku promjenu, tj. na svaku tipku.
- lazy - promjene u input polju će biti vidljive tek nakon što maknemo fokus s polja.
- trim - briše sve polazne ili krajnje bjeline.
- number - automatski se forsira konverzija stringa u broj.

Primjer:

```
<input v-model="tekst">
```

```
export default {
  data: function () {
    return {
      tekst: "Blue"
    }
  }
}
```

Listing 1.22: Prikaz v-model direktive na input elementu

Pod pretpostavkom da su HTML i JavaScript isječak dio iste datoteke, u input polju će inicijalno biti vidljiv tekst "Blue" i kako ga korisnik korigira na sučelju, simultano će se ažurirati i varijabla data objekta.

Opišimo ukratko kako se direktiva v-model koristi s nekim uobičajenim elementima HTML formi.

Textarea je element za prikaz višelinijskog teksta. Ne radi s interpolacijom stringova pa ako mu želimo proslijediti neku predefiniranu vrijednost trebamo je vezati v-model direktivom. Ako želimo čuvati tekst u formi u kojoj smo ga unijeli (s očuvanim prelascima u novi red), trebamo dodati css stil u style atribut (style="white-space: pre"). Napomena: to je samo zgodan trik CSS-a, nema veze s Vue okvirom.

Vrijednosti checkboxova koje smo naveli u atributu value možemo lako pospremati u neku varijablu s v-model direktivom. Posebno, ako varijablu deklariramo kao niz, tada se vrijednosti svih checkboxova kojima je vrijednost v-model direktive postavljena na danu varijablu (npr. v-model="mojNiz") dodaju u taj niz.

Radiobutton vrijednosti iz atributa value možemo spremati u posebnu varijablu koju definiramo pomoću v-model direktive. Varijabla koju smo naveli pod v-model ne samo da sprema value vrijednost nego se pomoću te varijable radiobuttoni grupiraju - Vue u pozadini obavi sve potrebne radnje za to.

Kod select i option tagova, možemo birati koju vrijednost ćemo po definiciji označavati pomoću vezanog atributa *:selected* u option tagu, ili pomoću v-model direktive u select tagu. Treba biti svjestan da v-model prepisuje *:selected* vezani atribut iz option taga. Dakle *:selected* radi samo kada v-model nije definirano.

Definiranje posebnih komponenti u formama

Vue ima podršku za kreiranje novih elemenata forme. Kako bi nam ovo bilo intuitivno treba razumjeti što i kako radi v-model. On se zapravo sastoji od vezanog atributa *:value* i metode *@input* (ili *@change* ako želimo lazy pristup).

Povezivanje podataka ostvarujemo preko direktive v-model u tagu naše komponente, jednako kao kod svih postojećih elemenata.

Nećemo ulaziti u detalje stvaranja novih elemenata formi, već upućujemo na dokumentaciju[7].

1.7 Filteri i Mixins

Filteri i mixins objekti pomažu kod strukturiranja aplikacije i prilagođenog ispisa sadržaja.

Filteri

Filteri ne mijenjaju sadržaj varijabli iz naše aplikacije, oni samo prilagođavaju prikaz podataka za korisnika. Vue ne dolazi s ugrađenim filterima, tako da sve filtere koji su nam potrebni moramo sami kreirati.

Ako želimo lokalno definirati filter definiramo ih pod svojstvom *filters*. Oni su funkcije pa ih tako i definiramo. U donjem primjeru na varijablu text primjenjujemo filter toUpperCase.

SOME TEXT

Some other text

```
<div id="app">
  <p> {{ text | toUpperCase }} </p>
  <p>Some other text</p>
</div>
```

```
new Vue({
  el: '#app',
  data() {
    return {
      text: 'Some text'
    }
  },
  filters: {
    toUpperCase(value) {
      return value.toUpperCase();
    }
  }
});
```

U samom HTML-u ih prikazujemo po uzoru na Angular 2 ('izraz' | filter). Možemo ih definirati i globalno pomoću *Vue.filter('ime-filtera', definicija);* u datoteci main.js.

Ako na istu vrijednost primjenjujemo više filtera, oni se ulančavaju. Dakle, primjenit će se točno onim redom kojim su navedeni. Vue ne zna pratiti filtere, odnosno, ne prati promjene u podacima na koje se primjenjuje filter, što znači da svaki put kad se osvježi DOM osvježe se i filteri. To je veliki udarac na performanse aplikacijskog okvira. Tome možemo doskočiti mudrim korištenjem *computed* svojstva. Computed svojstva se definiraju u objektu Vue instance pod svojstvom *computed*. Imaju formu funkcija koje vraćaju transformirane vrijednosti. Tako zaobiđemo filtere, tj. iskoristimo ugrađeni filter iz ES6 koji definiramo kako nam je potrebno, a izbjegnemo stalno osvježavanje.


```
<div id="example">
  <p>Original message: "{{ message }}"</p>
  <p>Computed reversed message: "{{ reversedMessage }}"</p>
</div>
```

```
var vm = new Vue({
  el: '#example',
    data: {
      message: 'Hello'
    },
    computed: {
      reversedMessage: function () {
        return this.message.split('').reverse().join('')
    }
  }
})
```

Listing 1.23: Primjer definiranja svojstva computed

Filteri općenito nisu najpoželjnija tehnika zbog svojih ograničenja.

Mixins

Mixins su objekti koje definiramo kako bismo smanjili dupliciranje koda. Dio koda koji nam treba na više mjesta definiramo u posebnoj JavaScript datoteci i zatim objekt s tim kodom koji smo definirali importiramo gdje god nam treba. Mixin se spaja s kodom komponente u koju se importira, ali na način da mixin ništa ne može promijeniti nego on samo dodaje opcije. Kada se kod učitava, prvo se učita kod iz mixina a zatim se učita kod komponente koja ima glavnu riječ i prerađuje metode iz mixina, ako se razlikuju. Dakle mixin je isključivo dodatak, nema ovlasti za ikakve promjene.

Globalni mixini su specijalni slučaj, oni se dodaju svakoj instanci i svakoj komponenti u aplikaciji. Definira se u datoteci main.js pomoću *Vue.mixin({ definicija });*. Lokalno se definiraju pomoću importa JavaScript datoteke, te registracijom u komponenti, pod svojstvom *mixins*, tako da imena svih mixins objekata navedemo u nizu.

Slijedi primjer mixin objekta i registracije u komponenti. U primjeru smo filter za dane u tjednu izdvojili u posebnu JavaScript datoteku koju zatim registriramo u nekoj komponenti.

```
export const dayMixin = {
  data() {
    return {
      days: [Sunday, Monday, Tuesday, Wednesday,
                Thursday, Friday, Saturday],
      filteredText: ''
    }
  },
  computed: {
    filteredDays() {
      return this.days.filter((el) => {
        return el.match(this.filteredText);
      });
    }
  }
}
```

Listing 1.24: Primjer mixin objekta

```
import {dayMixin} from './dayMixin.js';

export default {
  mixins: [dayMixin],
  data() {
    return {
      text: 'Some text.'
    }
  }
}
```

Listing 1.25: Primjer registracije mixin objekta u Vue komponenti

Dalje sve data varijable iz danog mixina koristimo kao da su navedeni u data objektu same komponente. Kao što je prethodno rečeno, mixin objekt se spaja s kodom komponente.

Prednost mixina je to što u svakoj komponenti gdje je definiran (ili u slučaju globalnog mixina, u svim komponentama) postoji lokalna kopija podataka i ne pokazuju svi objekti na istu memorijsku lokaciju. Na primjer, ako u istom sučelju imamo dvije različite komponente s istim mixinom tada ako unutar jedne komponente mijenjamo podatke mixina, podaci mixina unutar druge komponente će ostati netaknuti.

1.8 Animacije i tranzicije

Animacije i tranzicije su vrlo bitne za korisnika, pa tako i za programere koji razvijaju web-aplikaciju. Njihova uloga je na vrlo prirodan i intuitivan način prikazati korisniku gdje se točno nalazi u aplikaciji, što se događa i kako se kretati dalje. Ukratko, pomažu pri boljem snalaženju ili navigiranju. Na primjer, želimo dodati animaciju na dinamički element koji se, ovisno o uvjetu, prikazuje odnosno ne prikazuje.

Vue ima podršku za to kroz omotač (eng. wrapper) *<transition>*. Sve što je unutar tagova *<transition></transition>* može biti animirano, iako inicijalno nije. Dotični tag ima ograničenje - unutar njega samo jedan element može biti animiran u danom trenutku.

CSS klase koje Vue pridružuje takvom elementu su sljedeće: *v-enter, v-enter-active, v-leave, v-leave-active* ili ako tranzicijskom tagu dodamo atribut *name="nekoIme"* tada će te klase odgovarati *nekoIme-enter, nekoIme-enter-action, nekoIme-leave, nekoIme-leave-action*. Klase *-enter i *-leave traju samo jedan okvir, a klase *-action traju do kraja animacije. U CSS klasama možemo koristiti svojstva transition i animation. Svojstvo transition koristimo za postupno pojavljivanje ili nestajanje elementa (eng. fade), dok svojstvo animation više povezujemo s uklizavanjem elementa na ekran ili njegovo otklizavanje s ekrana (eng. slide). Za još bolji efekt često se animacije i tranzicije kombiniraju. Kod tog slučaja važno je odrediti po kojem svojstvu će Vue računati vrijeme animacije/tranzicije. Ako stavimo različita vremena, a ne odredimo koje svojstvo će diktirati duljinu animacije/tranzicije mogu se događati čudne stvari poput ponovnog pojavljivanja ili čak titranja elementa. Tip svojstva određujemo u transition tagu s atributom *type* koji može imati vrijednost transition ili animation. Postoji i atribut *appear* koji kaže Vue da animaciju/tranziciju primjeni odmah po učitavanju elementa.

Animacije/tranzicije se koriste s uvjetnim direktivama *v-if* i *v-show*. CSS klase na njima djeluju potpuno jednako.

Postoje zgodne biblioteke s već gotovim animacijama koje možemo samo uključiti u projekt i koristiti gotove klase, npr. animate.css[1].

Kod ulančanih efekata na elementima bitno je napomenuti da se uvjeti prikaza moraju kontrolirati isključivo s *v-if* (može u kombinaciji s *v-else*), ali ne funkcionira s *v-show*. Isti efekt možemo postići i čistim JavaScriptom jer Vue i to podržava, s tim da se ne koriste klase nego imamo 4 okidača za dolaznu animaciju te 4 okidača za izlaznu. Redom *@before-enter, @enter, @after-enter, @enter-cancelled* te *@before-leave, @leave, @after-leave, @leave-cancelled*.

Za animacijske efekte na listama koristimo novi tag *<transition-group></transition-group>* koji samo postavimo oko *li* elementa liste, s uvjetom da svaki od elemenata u listi mora imati jedinstveni atribut key kako bi ga Vue prepoznao te prikazao.

1.9 HTTP zahtjevi i paket vue-resource

Vue-resource[8] je poseban paket rađen upravo za aplikacijski okvir Vue.js. Njegova namjena je učiniti slanje i primanje HTTP zahtjeva prilično jednostavnim.

Vue-resource se može dodati u aplikaciju pomoću npm-a ili dodajući odgovarajuću skriptu u *index.html*. Ukoliko vue-resource instaliramo pomoću npm-a, naredbom:

```
npm install vue-resource
```

u kod datoteke main.js treba dodati sljedeće naredbe:

```
var Vue = require('vue');
var VueResource = require('vue-resource');
Vue.use(VueResource);
```

Vue.use(*neki-plugin*) je nova Vue metoda na razini globalnog Vue objekta, koja u suštini govori Vue.js-u da doda navedeni plugin u Vue.js jezgrenu biblioteku. Drugim riječima, zapravo proširujemo jezgrenu biblioteku. U ovom slučaju je to vue-resource koji će nam omogućiti jednostavno slanje i primanje HTTP zahtjeva.

Vue-resource paket koristi *promise* koncept, koji je ugrađen u novije verzije JavaScripta, tako da uvijek dobivamo povratnu informaciju o zahtjevu. To je dobro za asinkrone zahtjeve. *Promises* nam obećavaju povratne informacije o zahtjevu, koji se ne mora uvijek trenutno izvršiti. Može se koristiti globalno sa *Vue.http* ili lokalno pomoću *this.$http*.

Slanje POST zahtjeva serveru

Uz vue-resource, za POST zahtjev jednostavno koristimo npr. objekt *$http* i metodu post u kojoj kao argumente prosljeđujemo *URL* servera na koji šaljemo podatke, te sam objekt s podacima koje želimo poslati. Primjer:

```
export default {
  data() {
    return {
      user: {
        username: '',
        email: ''
      }
    };
  },
  methods: {
    submit() {
      this.$http.post('server-url', this.user)
        .then(response => {
```

```
14            console.log(response); /*povratne informacije*/
15          }, error => {
16            console.log(error); //informacije o gresci
17          });
18      }
19    }
20  }
```

Slanje GET zahtjeva serveru

Za izvođenje GET zahtjeva možemo opet koristiti objekt *$http* i metodu get kojoj je jedini argument *URL* servera s kojeg želimo dohvatiti podatke.

Zanimljivo je što kod objekta response ne dobivamo čiste ekstrahirane podatke koje očekujemo, već također dobivamo *promise* objekt. To je zato što je to asinkroni zahtjev i podatke možda nećemo dobiti trenutno.

Da bismo dobili objekt koji očekujemo, ulančavamo dvije *.then()* metode, gdje prva vraća response objekt koji je zapravo *promise*, a druga će sadržavati ekstrahirani objekt koji želimo.

```
1  export default {
2    data() {
3      return {
4        user: {
5          username: '',
6          email: ''
7        },
8        users: []
9      };
10   },
11   methods: {
12     submit() {
13       this.$http.get('server-url')
14         .then(response => {
15           return response.json(); //konvertira response iz
16                   //stringa u javascript objekt
17         }).then(data => {
18         //ovdje imamo ocekivani data objekt s kojim dalje mozemo
19         //manipulirati
20           const result = [];
21           for (let key in data) {
22             result.push(data[key]);
23           }
24           this.users = result;
25         });
26     }
```

```
27   }
28 }
```

U drugoj *then()* metodi koristimo for petlju za pristup samom objektu user-a jer ih dobivamo kao niz jedinstvenih ID-ja koji potom sadrže user objekte.

Datoteka main.js, iznad Vue instance, je centralno mjesto u kojem treba postaviti globalne postavke HTTP zahtjeva. Tako možemo jednom definirati URL-ove koji nam trebaju i kasnije ih koristiti. Možemo, na primjer, definirati URL za kasnije pomoću:

```
Vue.http.options.root='predefiniraniUrl';
```

Ovdje objektu http pristupamo bez prefiksa $(dolar) jer nismo u Vue instanci nego u globalnom Vue objektu. Tada u metodi post ili get smijemo URL ostaviti kao prazan string (‘’), koji će se automatski napuniti predefiniranim URL-om.

Interceptori

Interceptori su niz metoda koje želimo izvršiti sa svakim zahtjevom. Definiramo ih također u globalnom Vue objektu, pomoću:

```
Vue.http.options.interceptors.push((request, next) =>{..});
```

Svaki interceptor ima prepoznatljivu strukturu: prima metode *request* i *next*, gdje metoda *next* omogućava zahtjevu da se izvrši do kraja. Primjer nam pokazuje interceptor koji svaki zahtjev tipa POST presreće te pretvara i zatim dovršava kao zahtjev tipa PUT:

```
1  import Vue from 'vue'
2  import VueResource from 'vue-resource'
3  import App from './App.vue'
4
5  Vue.use(VueResource);
6
7  Vue.http.options.root = 'serverUrl';
8  Vue.http.interceptors.push((request, next) => {
9    console.log(request);
10   if (request.method == 'POST') {
11     request.method = 'PUT'; //svaki POST zahtjev ce se izvrsiti
12                   //kao PUT zahtjev
13   }
14   next(); //zbog ovoga se zahtjev izvrsi do kraja
15 });
16
17
```

```
new Vue({
  el: '#app',
  render: h => h(App)
});
```

Dani primjer je slučaj request-interceptora.

Također, možemo napraviti i response-interceptor, tako da u metodu *next()* dodamo response argument i njegovu lambda funkciju, kao u primjeru u kojem se povratne informacije o zahtjevu transformiraju na način da se za zahtjeve tipa GET vrati samo ektrahirani odgovor, bez ostalih informacija o zahtjevu:

```
import Vue from 'vue'
import VueResource from 'vue-resource'
import App from './App.vue'

Vue.use(VueResource);

Vue.http.options.root = 'serverUrl';
Vue.http.interceptors.push((request, next) => {
  console.log(request);
  if (request.method == 'POST') {
    request.method = 'PUT';
  }
  next( response => {
    response.json = () => { return  {message: response.body}}
    //izmjenjuje json metodu za GET zahtjeve
  });
});


new Vue({
  el: '#app',
  render: h => h(App)
});
```

Na ovaj način možemo modificirati podatke iz response-a, kao i pripadajuće metode, kako je prikazano.

Uz vue-resource, osim objekta *this.$http* postoji i objekt *this.$resource* u kojem možemo napraviti sami svoj resource. Vue-resources imaju nekoliko gotovih metoda koje možemo koristiti. To su: get('GET'), save('POST'), query('GET'), update('PUT'), remove('DELETE'), delete('DELETE'). Međutim, *$resource* nam najviše olakšava te smanjuje dupliciranje koda, kada mu proslijedimo niz prilagođenih metoda koje ćemo izvršavati na više mjesta.

Vue-resources su samo alternativa za obične get i post HTTP zahtjeve. Nisu uvijek bolje rješenje, ali postoje kao opcija. Na programeru je da odluči što mu je korisnije. Postoje i *URL predlošci* kroz koje možemo dinamički mijenjati URL na koji se povezujemo. U njima pomoću varijabli dinamički mijenjamo vrijednost putanje.

1.10 Preusmjeravanje (eng. routing)

Ruter, ili preusmjerivač, je alat bez kojeg je nezamislivo razvijati SPA aplikacije (*Single Page Applications*). On nam pomaže da simuliramo preusmjeravanje jedine stranice (obično *index.html*) u SPA aplikaciji, na način da se URL mijenja, kao i prikaz, pri čemu tehnički ostajemo i dalje na istoj stranici.

U slučaju SPA aplikacija ne komuniciramo više s nekim vanjskim serverima koji nam šalju potrebne podatke pa da pomoću Vue.js-a samo parcijalno uređujemo podatke ili njihov prikaz, nego u SPA aplikacijama Vue.js upravlja svim podacima, cijelom aplikacijom, i kontrolira baš sve. Za to nam treba preusmjerivač koji omogućava spomenute simulacije i time okviru Vue.js otvara vrata za gradnju i održavanje velikih i kompleksnih aplikacija.

Preusmjerivač koji koristi Vue.js je *vue-router* [9]. S preusmjerivačem sve ranije spomenute tehnike funkcioniraju potpuno isto, s prednošću da možemo izmjenjivati različite komponente na istom mjestu na ekranu, i tako dinamički učitavati HTML kod. Aktivira se dodavanjem skripte

```
<script src="https://unpkg.com/vue-router/dist/vue-router.js"></script>
```

u zaglavlje (eng. header), ili instaliranjem pomoću npm paketa:

```
npm install vue-router
```

Potrebno je dodati i sljedeće retke

```
import VueRouter from 'vue-router';
vue.use(VueRouter);
```

u datoteku main.js.

Za preumjeravanja koja ćemo definirati koristimo poseban objekt, koji može biti i u posebnoj JavaScript datoteci. Objekt koji definiramo je niz objekata, gdje svaki objekt u nizu predstavlja neko preusmjerenje. Objekti preusmjerenja se definiraju kako slijedi (pod pretpostavkom da su komponente User i Order ranije već definirane). Pri tome se pristupom URL-u *user* prikazuje komponenta User, a pristupom URL-u *order* prikazat će se komponenta Order. Točni URL kojem se pristupa bi bio: *http://localhost:8080/#/user*.

```
export const routes= [
  { path: '/user', component: User},
  { path: '/order', component: Order}
]
```

Ako *path* postavimo na prazan string(*path:''*) tada definiramo predefiniranu komponentu, koja će se vidjeti po učitavanju.

Sam preusmjerivač definiramo samo jednom, dakle uvijek imamo jednu instancu, i to kroz konstruktor u datoteci main.js:

```
const router = new VueRouter({
  routes: routes //objekt definiran u prethodnom primjeru
});

new Vue({
  el:'#app',
  routes, //gore je konstruktor, ovako se registrira router
  render: h => h(App)
});
```

U HTML kodu preusmjerivač nam je automatski omogućen tagom:

```
<router-view></router-view>
```

Kod linkova se pojavljuje " # ", koji preusmjerivaču govori da nakon učitavanja *index.html* stranice pristupi našoj JavaScript datoteci s definiranim preusmjerenjima i učita komponentu koja odgovara trenutnom URL-u. Moguće je ostvariti i preusmjeravanje bez pojave hash znaka, u posebnom modu *history* koji definiramo u konstruktoru rutera, uz preduvjet da serveru na kojem se izvršava aplikacija postavimo da uvijek vraća *index.html* - čak i kod greške sa statusom 404. History mod onemogućava " # " mod, koji se inače podrazumijeva ako *mode: 'history'* nije navedeno.

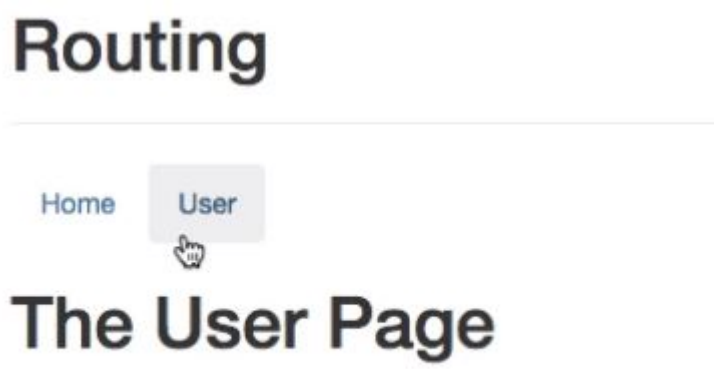
Navigacijski izbornik

Vrlo lako možemo definirati izbornik različih ruta u obliku navigacijskog izbornika. To omogućavamo pomoću taga:

```
<router-link to="/path-for-component">Name of component</router-link>
```

Primjer:

```
<template>
  <ul class="nav nav-pills">
    <li role="presentation">
      <router-link to="/">Home</router-link>
    </li>
    <li role="presentation">
      <router-link to="/user">User</router-link>
    </li>
  </ul>
</template>
```



Ako u pretraživaču analiziramo navigacijske gumbe, vidjet ćemo da oni jesu upravo *<a>* elementi koje Vue definira umjesto nas. Naravno, ovaj primjer s elementima *li* i *router-link* nije jedinstveni način izvođenja navigacijskog izbornika, ali pokazuje mogućnosti paketa vue-router.

Dinamičko preusmjeravanje i parametri

Ako želimo u JavaScript funkciji dinamički preusmjeravati komponente, npr. na klik gumba, to možemo izvesti pomoću *this.$router.push('/route')*. Na taj način stavljamo novu rutu na stog preusmjerenja pa smo sigurni da će nam *Nazad* i *Naprijed* gumbi ispravno raditi.

Ako želimo dodati parametre u rutu u objektu gdje definiramo rute, u *path*, dodajemo */:parameter*. Na primjer,

localhost:80/user/:id

dok u HTML kodu u tagu *router-link*, u atributu *to*, navodimo egzaktnu vrijednost parametra, na primjer,

localhost:80/user/6

Parametre rute možemo i dinamički dohvaćati u metodama. Primjer dohvata *id* parametra iz gornjeg primjera bi bio:

this.$route.params.id

Izvršavanje neke funkcije nakon promjene parametra moguće je pomoću watcher-a, kao što je prikazano u primjeru:

```
export default {
  data() {
    return {
      id: this.$route.params.id
    }
  },
  watch: {
    '$route'(to, from) {
      //zelimo nadzirati promjene u rutama
      //to - nova ruta na koju smo preusmjereni
      //from - prethodna ruta s koje smo preusmjereni
      this.id = to.params.id;
    }
  },
  methods: {
    navigateToHome() {
      this.$route.push('/'); //push na stog, kako bismo mogli
                  //aktivirati gumb za nazad
    }
  }
}
```

Na ovaj način, u varijabli *id* ćemo uvijek imati ažurnu vrijednost parametra.

Ugiježđena preusmjerenja

Ugniježđena preusmjerenja se definiraju kako bi dobili osjećaj navigacije po već ugniježđenoj podstranici. Na primjer, ako imamo aplikaciju za bilo kakvu uslužnu djelatnost, negdje na stranici imamo popis korisnika (popis korisnika je jedna komponenta). Prirodno je da na toj listi korisnika možemo kliknuti na nekoga od njih i pogledati detaljne informacije o korisniku. Prikaz detaljnih informacija bi bila podstranica podstranice (liste korisnika) i njezin URL zapravo proširuje URL do liste korisnika. Definiramo ih u nizu gdje su definirana i osnovna preusmjerenja, na način da objekt s preusmjerenjem proširimo atributom *children*.

```
export const routes= [
  { path: '', component: Home},
  { path: '/user', component: User, children:[
    {path: ':id', component: UserDetail},
    {path: ':id/edit', component: UserEdit}
  ]}

]
```

Da bismo vidjeli ugniježđene komponente i u roditeljskoj komponenti, koja je primarno preusmjerenje, trebamo tag *router-view*.

```
<template>
  <div>
    <h1>The User Page</h1>
    <hr>
    <button @click="navigateToHome" class="btn btn-primary">Go Home
    </button>
    <hr>
    <router-view></router-view>
  </div>
</template>
```

Za lakše upravljanje i održavanje linkova preusmjerenja, rutama možemo dati imena. To radimo tako da proširimo objekt preusmjerenja:

```
{ path: '/user/:id', component: UserDetail, name: 'userDetail'}
```

Tada se sa svakog mjesta u kodu, iz bilo koje komponente, možemo preusmjeriti na danu komponentu navodeći ime odgovarajuće rute te vrijednosti paramet(a)ra.

```
<router-link tag="button"
  :to="{ name: 'userDetail', params: { id: $route.params.id } }"
  class="btn btn-primary"> Show User Details </router-link>}
```

1.11 Biblioteka Vuex i upravljanje stanjima

Vuex je uzorak za upravljanje stanjima u aplikaciji i biblioteka s podrškom za Vue aplikacije. Služi kao centralno mjesto za pohranu varijabli za sve komponente u aplikaciji, s ciljem očuvanja konzistencije podataka i lakšeg upravljanja podacima. Preporuča se za veće aplikacije. Ako imamo veći broj komponenti u aplikaciji, međusobna komunikacija među njima može lako postati neuredna. Čak i korištenjem zajedničkog kanala (eng. event

bus), koji se brzo zaguši velikim brojem poruka, postaje teško pratiti promjene. Stoga za velike aplikacije postoji drugačije, pametnije rješenje - biblioteka Vuex.

Glavna politika ove biblioteke je da postoji centralizirano stanje. Sve varijable koje se koriste u više komponenti se spremaju na takvo centralno mjesto ili se od tamo uzimaju potrebne vrijednosti. Obično se takvo skladište realizira kao objekt u posebnoj JavaScript datoteci koja se nalazi u mapi *store*. Vuex instaliramo pomoću:

```
npm install vuex
```

te registriramo pomoću:

```
1 import Vuex from 'vuex';
2 Vue.use(Vuex);
```

i na kraju instanciramo pomoću

```
export const store = new Vuex.Store({...});
```

objekta unutar kojeg čuvamo *store* objekt u kojem su globalne varijable. Jednom kada smo definirali skladište varijabli, trebamo glavnu Vue instancu obavijestiti o skladištu tako da *store* objekt registriramo u Vue instanci pod atributom *store*. Tako registriranom skladištu možemo pristupiti od svuda u aplikaciji, pomoću Vue varijable *this.$store.state.imeVarijable*.

Getteri

Getteri se definiraju kako bi dohvaćali trenutno stanje iz skladišta te posluživali podatke komponentama koje koriste dane varijable. Pomoću njih možemo prilagođavati izlaznu vrijednost u ovisnosti o varijabli, npr. ako znamo da nekim komponentama treba višekratnik neke varijable tada kroz predefinirani *getter* možemo vraćati točno taj višekratnik. Gettere definiramo u samom Vuex.Store() objektu u svojstvu *getters*. Pristupa im se slično kao i varijablama: *this.$store.getters.imeGettera*.

Mutations

Mutations su setteri koji mijenjaju vrijednosti u objektu s globalnim varijablama. Definiramo ih u objektu Vuex.Store() kao i gettere, u svojstvu *mutations*.

To je opet niz metoda koje želimo izvoditi (kao i getteri). Ove metode nemaju povratni tip nego se varijablama postavljaju nove vrijednosti. Pozivamo ih iz drugih komponenti pomoću *this.$store.commit('imeMutationa')*.

```
const state = {
  funds: 15000
};
const getters = {
  funds (state) {
    return state.funds;
  }
};
const mutations= {
  'ENLARGE_FUNDS' (state, addition) {
    stock.funds += addition;
  }
};

export default {
  state,
  getters,
  mutations
}
```

Listing 1.26: Primjer gettera i settera

Primjer nam pokazuje jednostavne primjere gettera i settera za varijablu 'funds' iz store objekta komponente. Getter nam jednostavno vraća vrijednost varijable dok je setter povećava za iznos koji mu proslijedimo kao argument u funkciji. Vidimo da u objektu instance Vue koji izvozimo svojstva nisu u obliku ključ-vrijednost nego su navedena samo imena svojstava. To je novina iz EcmaScript 6 standarda. Ukoliko objekt pridružen svojstvu dijeli isto ime sa svojstvom, objekt se ne mora posebno navoditi nego se podrazumijeva (na primjer, state je jednako kao da piše state:state).

Kod mutacija postoji jedno veliko ograničenje - moraju se događati sinkrono. Dakle, bilo koja varijanta setTimeout metode ili nešto tome slično neće raditi prema očekivanjima, jer se gubi *"watcher"* varijable. Mutacije mijenjaju stanje varijabli odmah, kako ne bi došlo do zabune kod redoslijeda primanja obavijesti. Cijelu tu prepreku s asinkronim mutacijama možemo zaobići koristeći dodatno definiranu akciju koja će zatim spremati promjene kada smo gotovi s asinkronim poslovima.

Akcije (eng. actions) su funkcije koje izvršavaju zadanu mutaciju nakon asinkronih radnji, ako postoje. U akcijama promjene nisu nužno trenutne. Akcije se definiraju jednako kao i mutacije - u objektu Vuex.Store(), samo u svojstvu *actions*. Spremanje promjena aktiviramo metodom *commit* u kojoj navodimo ime mutacije za koju želimo da se izvrši nakon asinkronog posla, kako je gore i spomenuto. Same akcije se mogu pokretati metodom dispatch, primjer:

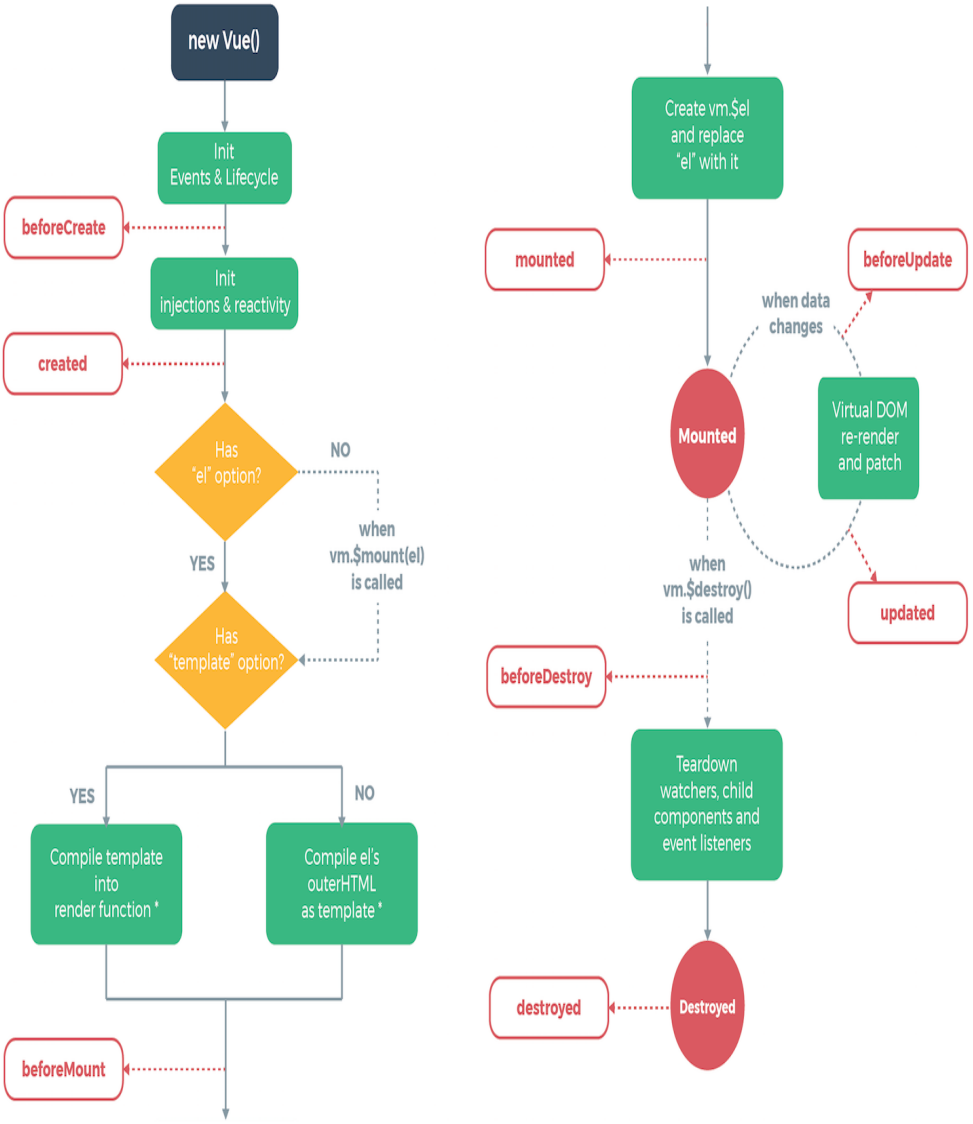
store.dispatch('imeMutationa')

Primjetimo da ovdje nemamo znak *$* ispred *store*, jer u ovom slučaju *store* je globalna varijabla (nije nativni element).

Dobre prakse

Dobra praksa je koristiti akcije čak i kod sinkronih metoda, iako nije nužno. Preporuka za strukturu aplikacije u kojoj se koristi Vuex je da modulariziramo datoteke i razdijelimo ih po funkcionalnostima. Najčešća struktura na koju nailazimo: unutar mape *store* kreiramo mapu *modules* i u njoj čuvamo JavaScript datoteke razdijeljene po funkcionalnostima, gdje svaka datoteka ima svoje objekte s vlastitim varijablama *state, getters, mutations, actions*. Na kraju datoteke sve objekte izvozimo (exportamo) kako bismo ih u *store* objektu mogli registrirati u atributu *modules*. Ako imamo dosta metoda koje se ne odnose niti na jedan modul, nego recimo na zaglavlje (eng. header) ili podnožje (eng. footer) stranice - koji su svuda prisutni, te skupine metoda možemo također definirati izvan samog *store* objekta, ali unutar mape *store* u JavaScript datotekama koje također kasnije registriramo u *store* objektu. Jednako tako možemo stvoriti podmapu *data* unutar mape *src* kako bismo u nju spremali JavaScript datoteke s podacima pojedinih komponenti. Vue će znati da se unutar mape data nalaze data objekti odgovarajućih komponenti.

Dobro je biti svjestan svih životnih ciklusa Vue instance kako bismo određene metode mogli izvršavati u pravo vrijeme. Ilustracija životnih ciklusa Vue instance je prikazana na slici 1.2.



Slika 1.2: Životni ciklus Vue instance

Poglavlje 2

Aplikacija Stock Trader

Aplikacija koju ćemo opisati obuhvaća veliki dio svojstava i mogućnosti aplikacijskog okvira Vue.js, koje smo prethodno objasnili. Aplikacija oponaša burzu dionicama, te dinamički mijenja sadržaje u komponentama. Na taj način prezentira komunikaciju među komponentama, za što koristimo paket Vuex. Koristi se i paket vue-router, za dinamičko mijenjanje komponenti unutar aplikacije.

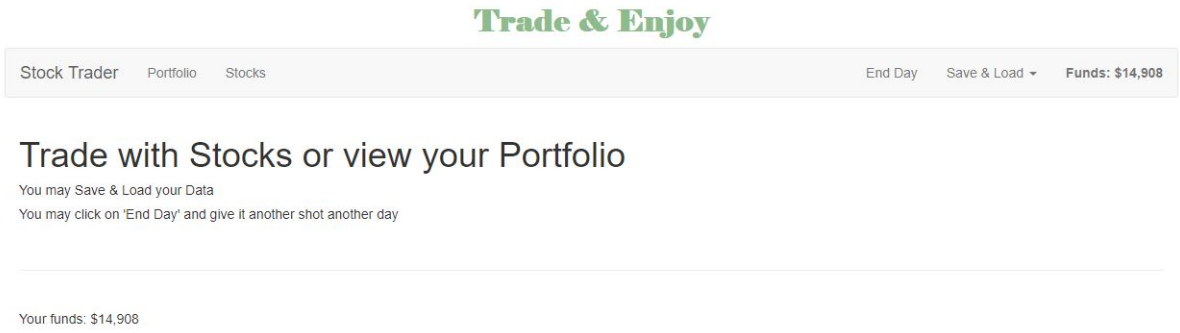
2.1 Opis aplikacije

Aplikacija je podijeljena u 3 osnovna ekrana: ekran *Home*, ekran *Portfolio* te ekran *Stocks*. Ekran *Home* služi za osnovne informacije o načinu funkcioniranja aplikacije, te opis pojedinih mogućnosti. Na ekranu *Portfolio* vidimo vlastite kupljene dionice, njihovu prodajnu cijenu i stanje budžeta. Na ekranu *Stocks* su prikazane sve dionice koje se nude na burzi, trenutni budžet te cijene dionica.

Imamo i zaglavlje koje se provlači kroz sve ekrane, a sastoji se od naslova aplikacije i navigacijske trake koja nam nudi mogućnosti promjene ekrana, kao i uvid u trenutni budžet te akcije za završetak dana ili spremanje/učitavanje podataka iz baze. Akcija za završetak dana mijenja cijenu dionica koja može varirati od 50% cijene do 150% cijene.

Slika 2.1: Izgled zaglavlja

Za ljepši i moderniji izgled aplikacije koristimo biblioteku *Bootstrap*. Na ekranu *home* se vide opće informacije vezane za navigaciju.

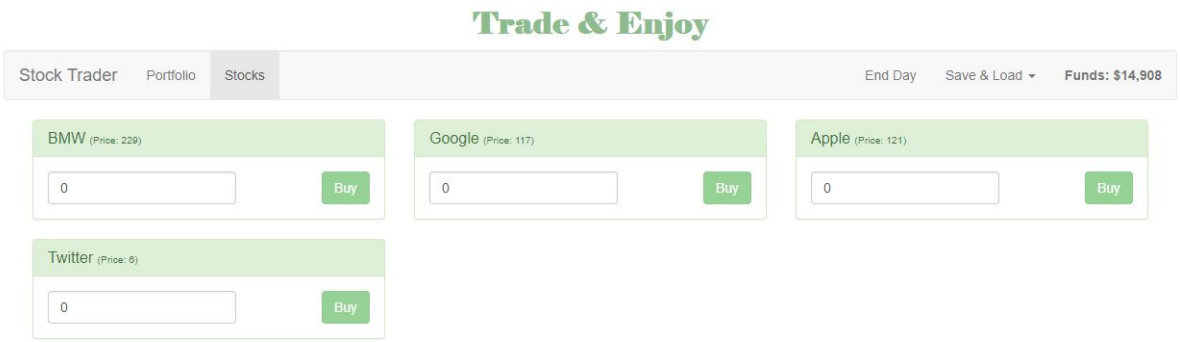


Slika 2.2: Prikaz ekrana home

Na ekranu *portfolio* se vide kartice kupljenih dionica na kojima je prikazano ime dionice, količina kupljenih dionica te trenutna cijena dionica, kao i polje inbox za unos željenog broja dionica koje želimo prodati. Kartice su izrađene kao bootstrap paneli s klasom *.panel-info*, kako bi dobile plavi obrub.

Slika 2.3: Primjer ekrana portfolio

Na ekranu *stocks* imamo listu svih dionica za trgovanje s gotovo jednako formatiranim karticama kao na ekranu *portfolio*. Razlika je što nemamo navedenu količinu (jer se dionice nude za kupovinu), te imaju klasu *.panel-success* kako bi bile zelene boje.



Slika 2.4: Primjer ekrana stocks

2.2 Implementacija

Aplikacija je implementirana u duhu dobrih praksi za Vue i Vuex. Vuex nam pomaže kod bolje organizacije koda te preglednosti, jednako kao i kod lakšeg upravljanja stanjima u aplikaciji. U aplikaciji smo, kroz terminal, instalirali paket Vuex naredbom:

```
npm install vuex
```

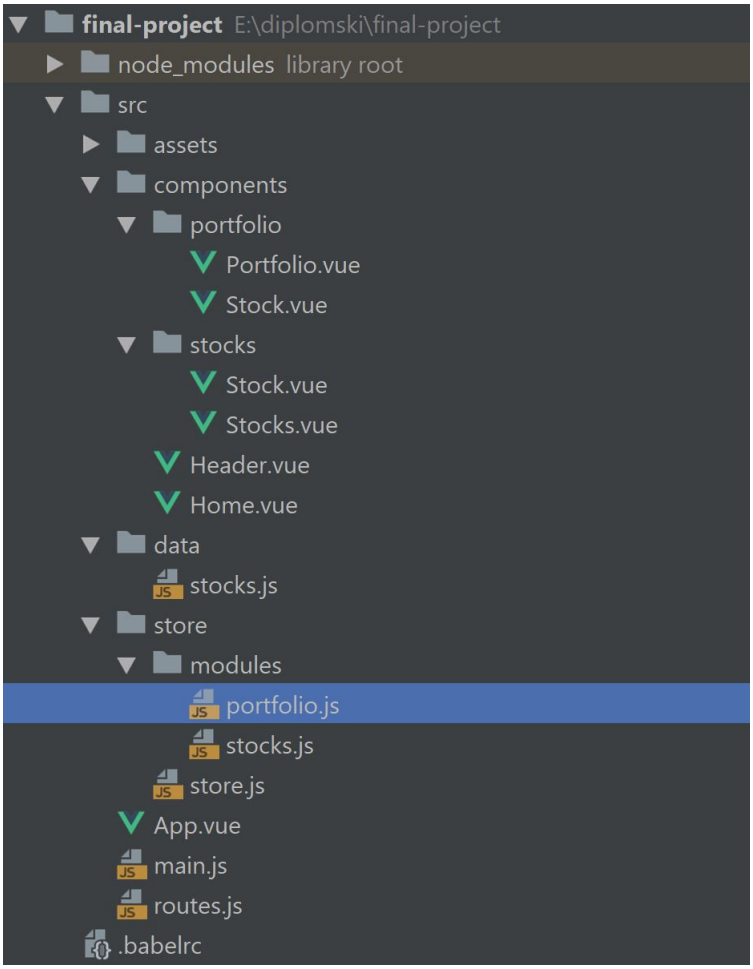
Napravili smo datoteku *store.js* u kojoj smo definirali objekt *store*, prema ranije opisanoj strukturi, te unutar mape *store* napravili još jednu mapu *modules* za dodatne JavaScript datoteke razdijeljene po modulima, kao što nam nalažu dobre prakse. Također stvorili smo i mapu *data* za pospremanje podataka (data objekata) pojedinih komponenti.

Sve funkcije za mijenjanje stanja budžeta te cijene i količine dionica rade na temelju *mutations*-a, odnosno *actions*-a. Također, sve podatke dohvaćamo pomoću *getters*-a, karakterističnо za Vuex.

Pomoću paketa *vue-router*, kojeg smo instalirali pomoću

```
npm install vue-router
```

napravili smo navigacijski izbornik. Izbornik se nalazi u zaglavlju koje je definirano u datoteci *Header.vue*.



Slika 2.5: Prikaz strukture

Sva preusmjerenja smo definirali samo na jednom mjestu, u datoteci *routes.js*. Ako dođe do ikakvih promjena to je jedino mjesto koje bismo trebali promijeniti.

```
export const routes = [
  {path: '/', component: Home},
  {path: '/portfolio', component: Portfolio},
  {path: '/stocks', component: Stocks}
]
```

Listing 2.1: Preusmjerenja definirana u routes.js datoteci

```
<div class="col-xs-12">
  <transition name="slide" mode="out-in">
    <router-view></router-view>
  </transition>
</div>
```

Listing 2.2: Prikaz HTML koda datoteke App.vue koji nam omogućuje tranzicijske efekte i preusmjeravanje

Napokon, za HTTP zahtjeve instalirali smo paket vue-resource. On nam treba za pospremanje i učitavanje podataka - Save & Load Data koje imamo u padajućem izborniku u zaglavlju. Aplikacija nema implementirano logiranje korisnika, no podatke o dionicama na tržištu i kupljenim dionicama na pritisak gumba Load uzima iz online baze, u koju i sprema trenutno stanje dionica na pritisak gumba Save. Po učitavanju stranice, inicijalni popis dionica za tržište uzima iz JavaScript objekta.

Imamo dva osnovna modula u koja dijelimo aplikaciju: portfolio i stock. U modulu portfolio imamo dvije komponente: Portfolio.vue i Stock.vue. Komponenta Portfolio se sastoji liste Stock komponenti. Komponenta Portfolio je roditeljska komponenta, a Stock ugniježđena komponenta. Portfolio sadrži getter za dohvaćanje niza kupljenih dionica sa svim potrebnim podacima, dok komponenta Stock ima definiranu akciju za prodaju dionica koja koristi mutaciju `SELL_STOCK`.

```
methods: {
  ...mapActions({
      placeSellOrder: 'sellStock'
    }),
  sellStock() {
    const order = {
      stockId: this.stock.id,
      stockPrice: this.stock.price,
      quantity: this.quantity
    };
    this.placeSellOrder(order);
  }
}

```

Listing 2.3: Isječak JavaScript koda iz datoteke portfolio/Stock.vue

...mapActions je primjer operatora proširenja iz EcmaScript 2015, u kojem navodimo više objekata koji se svi registriraju kao akcije (unutar jednog objekta, umjesto da svaki registriramo sam za sebe). Ovdje imamo samo jednu akciju, ali pravilo vrijedi općenito.

```
stockPortfolio (state, getters) {
  return state.stocks.map(stock => {
    const record = getters.stocks
            .find(element => element.id == stock.id);
    return {
      id: stock.id,
      quantity: stock.quantity,
      name: record.name,
      price: record.price
    }
  });
}
```

Listing 2.4: Getter kupljenih dionica

Getter kombinira globalni objekt u kojem čuvamo sve dionice koje se nude na tržištu s lokalnim stocks objektom u kojem držimo sve dionice koje smo kupili jer se povratni podaci mapiraju iz oba objekta.

```
'SELL_STOCK' (state, {stockId, quantity, stockPrice}) {
  const record = state.stocks
            .find(element => element.id == stockId);
  if (record.quantity > quantity) {
    record.quantity -= quantity;
  }
  else {
    state.stocks.splice(state.stocks.indexOf(record), 1);
  }
  state.funds += stockPrice*quantity;
}
```

Listing 2.5: Mutacija SELL_STOCK

Mutacija smanjuje količinu kupljenih dionica ukoliko broj prodanih dionica nije veći od broja kupljenih. Ako je broj dionica za prodaju jednak broju kupljenih dionica, ista dionica se izrezuje iz niza kupljenih dionica. Budžet se lienarno povećava, ovisno o broju prodanih dionica.

U stocks modulu također imamo dvije komponente: Stocks.vue i Stock.vue. Primijetimo da imamo dvije naizgled jednake komponente Stock.vue u dva različita modula. Svaka je definirana u posebnoj mapi pripadajućeg modula, odnosno roditeljske komponente. Komponenta Stocks se sastoji od liste vlastitih komponenti Stock te pomoću metode computed i gettera definiranog u objektu store dohvaća listu dostupnih dionica. Komponenta Stock definira metodu s akcijom za kupovinu dionica gdje se izvršava mutacija `BUY_STOCK`.

```
export default {
  components: {
    appStock: Stock
    },
  computed: {
    stocks() {
      return this.$store.getters.stocks;
    }
  }
}
```

Listing 2.6: Stocks Vue objekt

```
'BUY_STOCK' (state, {stockId, quantity, stockPrice}) {
  const record = state.stocks
          .find(element => element.id == stockId);
  if (record) {
    record.quantity += quantity;
  }
  else {
    state.stocks.push({
      id: stockId,
      quantity: quantity
    });
  }
  state.funds -= stockPrice*quantity;
}
```

Listing 2.7: Mutacija BUY_STOCK

Za tečnije i ljepše pomicanje panela dodali smo tranzicijske, odnosno animacijske klase kako bi paneli klizili sa ekrana odnosno na ekran.

```
body {
  padding: 30px;
}

.slide-enter-active {
  animation: slide-in 200ms ease-out forwards;
}
.slide-leave-active {
  animation: slide-out 200ms ease-out forwards;
}

@keyframes slide-in {
  from {
    transform: translateY(-30px);
```

```
15      opacity: 0;
16    }
17    to {
18      transform: translateY(0);
19      opacity: 1;
20    }
21  }
22
23  @keyframes slide-out {
24    from {
25      transform: translateY(0);
26      opacity: 1;
27    }
28    to {
29      transform:  translateY(-30px);
30      opacity: 0;
31    }
32  }
```

Listing 2.8: Sadržaj style dijela datoteke App.vue

Navedimo ostatak akcija i odgovarajućih mutacija koje koristimo:

- akcija *randomizeStocks* koja izvršava mutaciju `RANDOM_STOCKS` - ona se provodi na pritisak gumba 'End day' u zaglavlju ekrana.
- akcija *loadData* koja izvršava http zahtjev na online bazu i dohvaća podatke, te potom izvršava mutacije `SET_STOCKS` i `SET_PORTFOLIO` koje postavljaju varijable niza dionica za prodaju te niza dionica za kupovinu i stanje budžeta (ekran portfolio), redom.

Cjelokupni kod aplikacije Stock Trader dostupan je na priloženom CD-u. Ova aplikacija napravljena je na temelju online udžbenika [6] za Vue okvir.

Bibliografija

[1] *Animate.css*, https://github.com/daneden/animate.css, pristupljeno u srpnju, 2018.

[2] *Bootstrap*, https://getbootstrap.com/, pristupljeno u srpnju 2018.

[3] *Instalacija paketa Vue*, https://vuejs.org/v2/guide/installation.html, pristupljeno u srpnju, 2018.

[4] *Referenca na definiciju animacijskog okvira preglednika*, https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame, pristupljeno u srpnju, 2018.

[5] *Tutorijal-Laracast*, https://laracasts.com/series/learn-vue-2-step-by-step, pristupljeno u srpnju, 2018.

[6] *Tutorijal-Udemy*, https://www.udemy.com/vuejs-2-the-complete-guide, pristupljeno u srpnju, 2018.

[7] *Vue komponente*, https://vuejs.org/v2/guide/components.html, pristupljeno u srpnju 2018.

[8] *Vue-resource*, https://github.com/pagekit/vue-resource, pristupljeno u srpnju, 2018.

[9] *Vue-router*, https://router.vuejs.org/, pristupljeno u srpnju, 2018.

[10] *Web-application*, https://en.wikipedia.org/wiki/Web_application, pristupljeno u lipnju, 2018.

[11] *Web lokacija aplikacijskog okvira Vue*, https://vuejs.org/, pristupljeno u lipnju, 2018.

Sažetak

U ovom radu smo predstavili i objasnili JavaScript aplikacijski okvir pod nazivom Vue.js koji se koristi za izradu modernih web-aplikacija. Objasnili smo njegove značajke i glavne prednosti. Izradom prvo vrlo jednostavne, a potom i naprednije web-aplikacije na primjerima smo objasnili mogućnosti aplikacijskog okvira Vue.js. Glavna prednost Vue.js okvira je mogućnost izrade modernih SPA (Single-Page Applications) aplikacija i vrlo dobra optimiziranost same biblioteke. Samim time aplikacije izrađene pomoću Vue.js aplikacijskog okvira imaju vrlo dobre predispozicije da budu brze i bez zastajkivanja.

Summary

In this graduate thesis we have introduced and explained JavaScript framework Vue.js which is used for building modern web-applications. We've explained its core features and main advantages. Building one simple and one more complex sample web-application gave us an opportunity to show all feature possibilities of Vue.js through examples. Vue's main advantage is the possibility of building modern SPA applications and a nicely optimized core library. With these properties of the framework, applications built with Vue.js are more likely to be fast and smooth.

Životopis

Ana Vidović je rođena 17. srpnja 1992. godine u Zagrebu. Završila je osnovnu školu Frana Krste Frankopana. Zatim upisuje XV. gimnaziju u Zagrebu, matematičko-informatički smjer. Po završetku srednje škole upisuje preddiplomski studij Matematike na Prirodoslovno-matematičkom fakultetu u Zagrebu, koji završava 2016. godine i nastavlja diplomski studij Matematika i računarstvo na istom fakultetu. Unazad dvije i pol godine radi uz studij, zadnjih godinu i pol dana kao programer u programskom jeziku Java.