

Napredni razvoj programske podpore za web

**- predavanja -
2021./2022.**

9. Jednostranične web-aplikacije Vue.js

Creative Commons



- slobodno smijete:

- **dijeliti** — umnožavati, distribuirati i javnosti priopćavati djelo
- **prerađivati** djelo



- pod sljedećim uvjetima:

- **imenovanje:** morate priznati i označiti autorstvo djela na način kako je specificirao autor ili davatelj licence (ali ne način koji bi sugerirao da Vi ili Vaše korištenje njegova djela imate njegovu izravnu podršku).
- **nekomercijalno:** ovo djelo ne smijete koristiti u komercijalne svrhe.
- **dijeli pod istim uvjetima:** ako ovo djelo izmijenite, preoblikujete ili stvarate koristeći ga, preradu možete distribuirati samo pod licencom koja je ista ili slična ovoj.



U slučaju daljnjeg korištenja ili distribuiranja morate drugima jasno dati do znanja licencne uvjete ovog djela.

Od svakog od gornjih uvjeta moguće je odstupiti, ako dobijete dopuštenje nositelja autorskog prava.

Ništa u ovoj licenci ne narušava ili ograničava autorova moralna prava.

Tekst licence preuzet je s <http://creativecommons.org/>



Vue.js

a crash course...

routing, components, forms, Vuex

Vue CLI

(prethodno instalirati
Node.js i npm)

```
npm install -g @vue/cli  
(...)
```

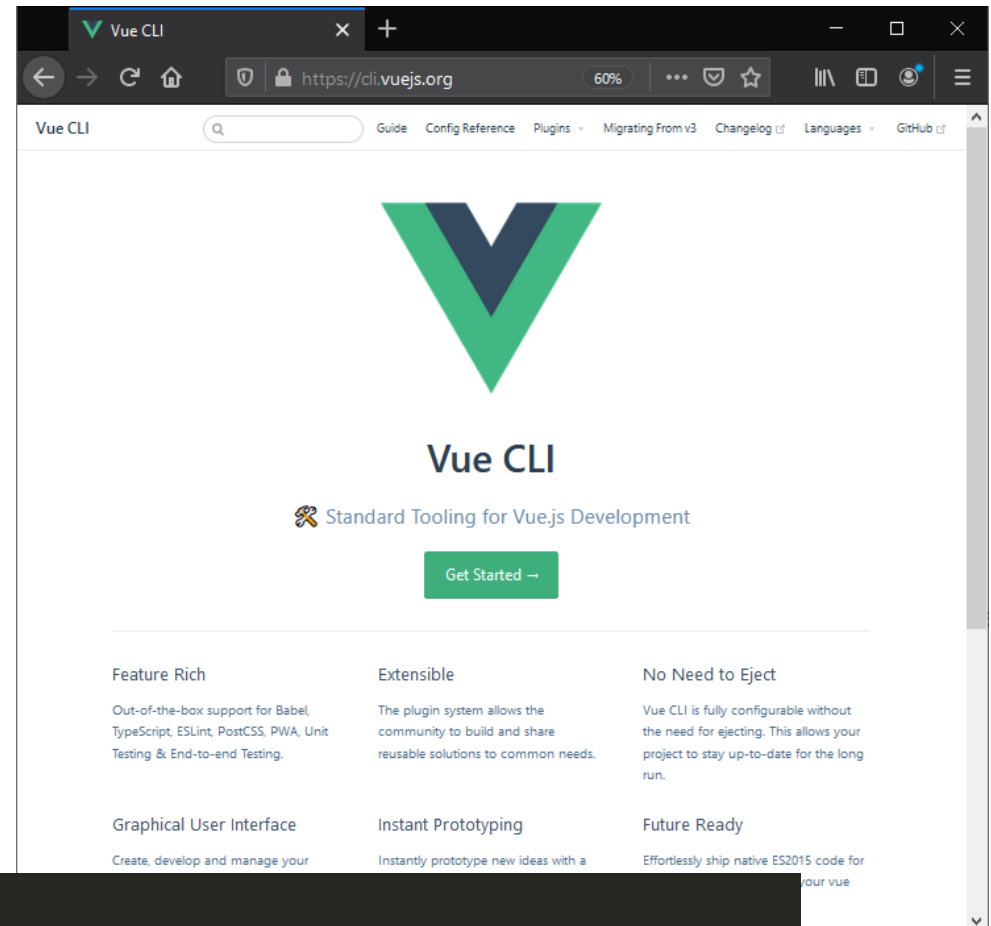
```
vue create fer-demo-spa
```

- Manually select features:

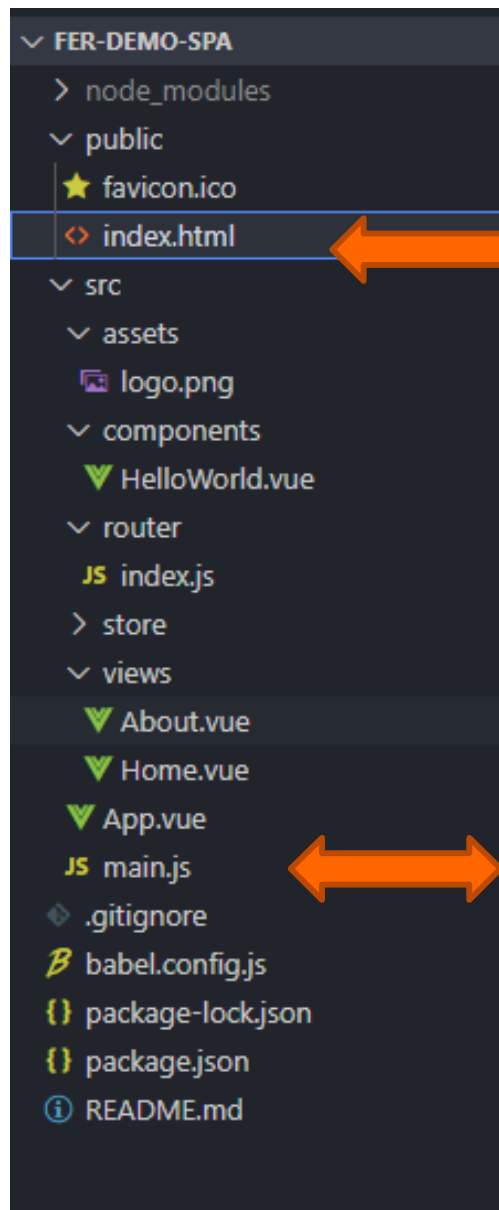
Vue CLI v4.5.13

```
? Please pick a preset: Manually select features  
? Check the features needed for your project: Choose Vue version, Babel, Router, Vuex, Linter  
? Choose a version of Vue.js that you want to start the project with 3.x  
? Use history mode for router? (Requires proper server setup for index fallback in production) Yes  
? Pick a linter / formatter config: Prettier  
? Pick additional lint features: Lint on save  
? Where do you prefer placing config for Babel, ESLint, etc.? In package.json  
? Save this as a preset for future projects? (y/N)
```

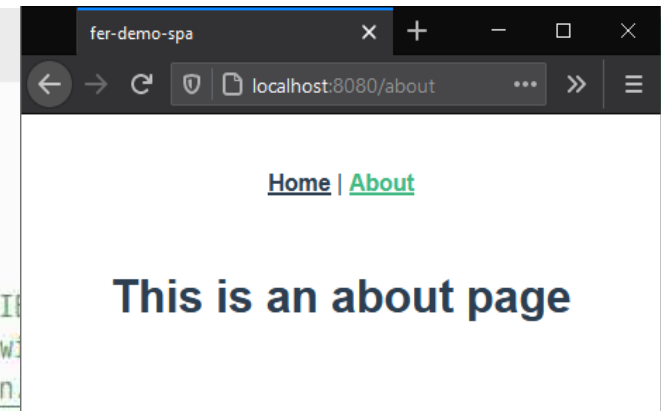
```
✔ Successfully created project fer-demo-spa.  
✔ Get started with the following commands:  
  
$ cd fer-demo-spa  
$ npm run serve
```



Struktura projekta



```
index.html x
public > <> index.html > ...
1  <!DOCTYPE html>
2  <html lang="">
3    <head>
4      <meta charset="utf-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1">
7      <link rel="icon" href="%= BASE_URL %>favicon.ico">
8      <title>%= htmlWebpackPlugin.options.title %>
9    </head>
10   <body>
11     <noscript>
12       <strong>We're sorry but <%= htmlWebpackPlugin.options.title %>
13     </noscript>
14     <div id="app"></div>
15     <!-- built files will be auto injected -->
16   </body>
17 </html>
```



```
JS main.js x
src > JS main.js
1  import { createApp } from "vue";
2  import App from "./App.vue";
3  import router from "./router";
4  import store from "./store";
5
6  createApp(App).use(store).use(router).mount("#app");
```

.vue datoteke

```
<template>  
  (ovdje pišemo HTML)  
</template>
```

```
<script>  
  (ovdje pišemo JS)  
</script>
```

```
<style scoped>  
  (ovdje pišemo CSS)  
</style>
```

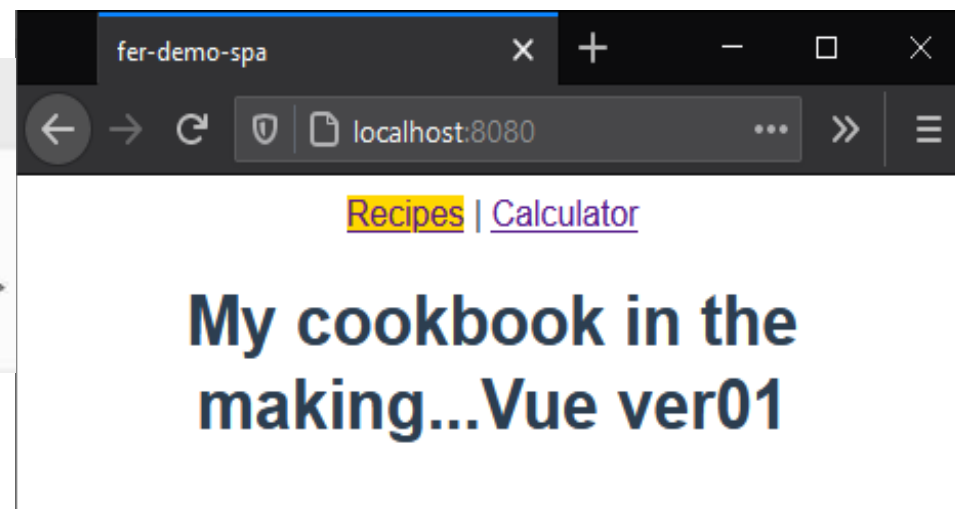
- Imamo tri sekcije:
 - template
 - script
 - style
- Automatski se prevodi u JS
 - Babel
 - Webpack
- Koristi se:
 - View
 - Component

View vs Component

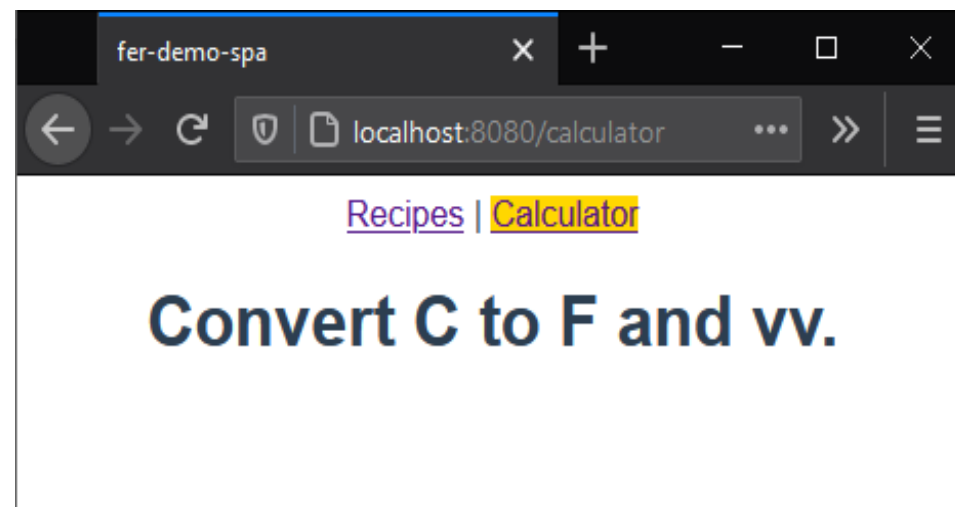
- Primjećujemo dva koncepta u ustroju projekta (i dva različita direktorija):
 - **Views** (src/components)
 - **Components** (src/components)
- Na tehničkoj razini su to iste stvari – **komponente**
 - Vrijedi sve s prethodnog slajda
- Ali semantički:
 - **Views** predstavljaju **stranice** naše **više-stranične** aplikacije i tipično se referenciraju iz usmjerivača (router, uskoro)
 - **Components** predstavljaju (ponovo upotrebljive) komponente koje se koriste na N stranica
- Postoje **različite konvencije** kako ih nazivati i gdje ih pohranjivati (npr. *pages*, *views*, itd.)

Izmijenimo generirani projekt u začetak naše Cookbook aplikacije

```
Recipes.vue U X
src > views > Recipes.vue > ...
1 <template>
2 | <h1>My cookbook in the making...Vue ver01</h1>
3 </template>
```



```
Calculator.vue U X
src > views > Calculator.vue > ...
1 <template>
2 | <h1>Convert C to F and vv.</h1>
3 </template>
4 |
```



Ako koristite **VS Code**, instalirati **Vetur** za *syntax highlighting* i sl.

Tehnička opaska

- Preneseno s: <https://gitlab.com/fer-web2/spa-3/-/blob/master/README.md>

cookbook

Različite verzije možete dohvaćati tako da:

- prvo naravno napravite `git clone git@gitlab.com:fer-web2/spa-3.git`
- `cd spa-3`
- zatim checkout verziju koju hoćete (oznake su u predavanjima), npr.:
 - `git checkout v2.0`
 - `git checkout v3.0`
 - ...

Za verziju (tag) 4+ pokrenuti i fer-cookbook-backend koji vraća recepte, dakle:

- `...\fer-cookbook-backend>node server.js`
- `...\fer-cookbook>npm run serve`

Konfiguracija routera

main.js

```
import router from "../router";
createApp(App)
  .use(router)
  .mount("#app");
```

App.vue

```
<template>
  <div id="nav">
    <router-link to="/">
      Recipes
    </router-link> |
    <router-link to="/calculator">
      Calculator
    </router-link>
  </div>
  <router-view />
</template>
```

router/index.js

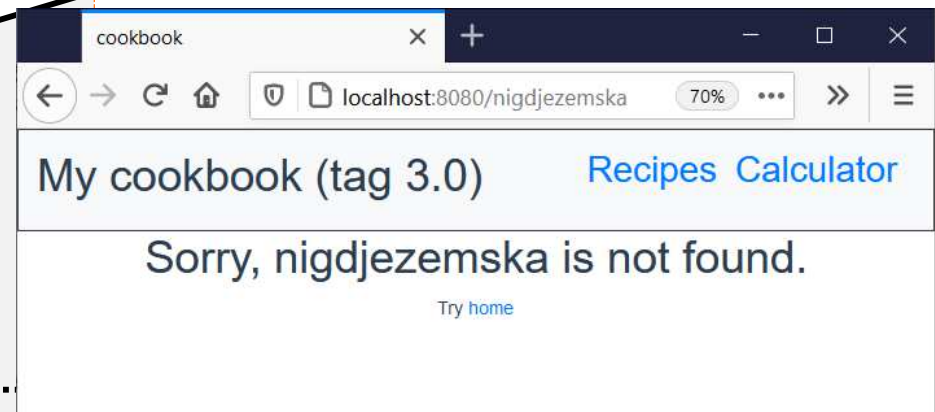
```
import { createRouter, createWebHistory } from "vue-router";
import Recipes from "../views/Recipes.vue";
import Calculator from "../views/Calculator.vue";
const routes = [
  {
    path: "/",
    name: "Recipes",
    component: Recipes,
  },
  {
    path: "/calculator",
    name: "Calculator",
    component: Calculator,
  },
];
const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes,
});
export default router;
```

Dynamic routing, catch-all

router/index.js

```
...  
const routes = [ // postoji i alias: "/"  
  { path: "/", // mogli smo i redirect: "/recipes"  
    component: Recipes,  
  },  
  {  
    path: "/recipes/:id?",  
    component: Recipes,  
  },  
  { path: "/calculator",  
    component: Calculator,  
  },  
  { path: "/*:catchAll(.*)",  
    name: "NotFound",  
    component: NotFound,  
  },  
];  
...
```

Opcionalni (zbog upitnika) parametar, poslije dostupan kao `$route.params.id`



NotFound.vue

```
<template>  
  <h1>Sorry, {{ $route.params.catchAll }}  
    is not found.</h1>  
  Try <router-link to="/">home</router-link>  
</template>
```

catchAll je proizvoljno ime varijable, u zagradi je regex

Recipes.vue 1/2

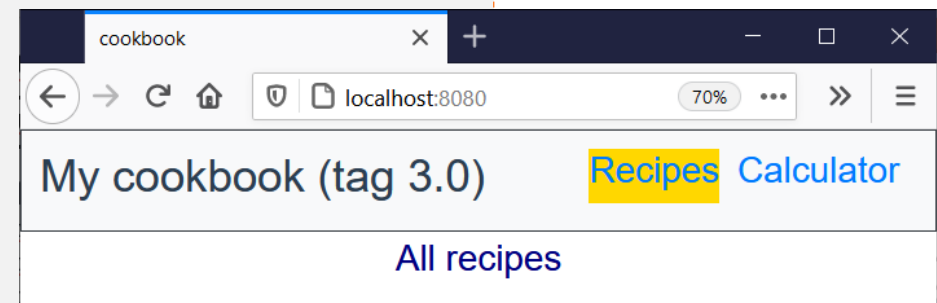
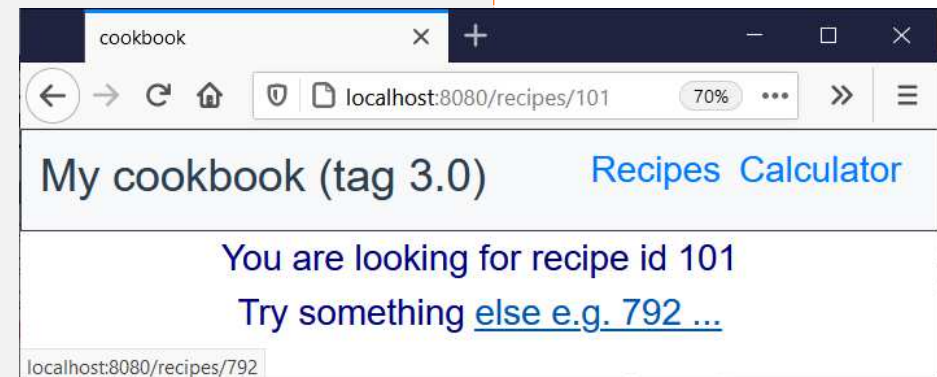
Recipes.vue

```

<template>
  <div v-if="$route.params.id">
    You are looking for recipe id {{ $route.params.id }}
    <br>Try something <router-link :to="'/recipes/' + somethingElse">
      else e.g. {{ somethingElse }} ...
    </router-link>
  </div>
  <div v-else>All recipes</div>
  <ul>
    <li v-for="msg in routeChanges"
      :key="msg">{{ msg }}</li>
  </ul>
</template>
<script>
export default {
  data() {
    return {
      routeChanges: [],
      somethingElse: Math.floor(Math.random() * 1000)
    };
  }, ...
}

```

Primijetiti dvotočku – izraz pod navodnicima se onda tumači kao js expression



Recipes.vue 2/2: watchers, scoped style

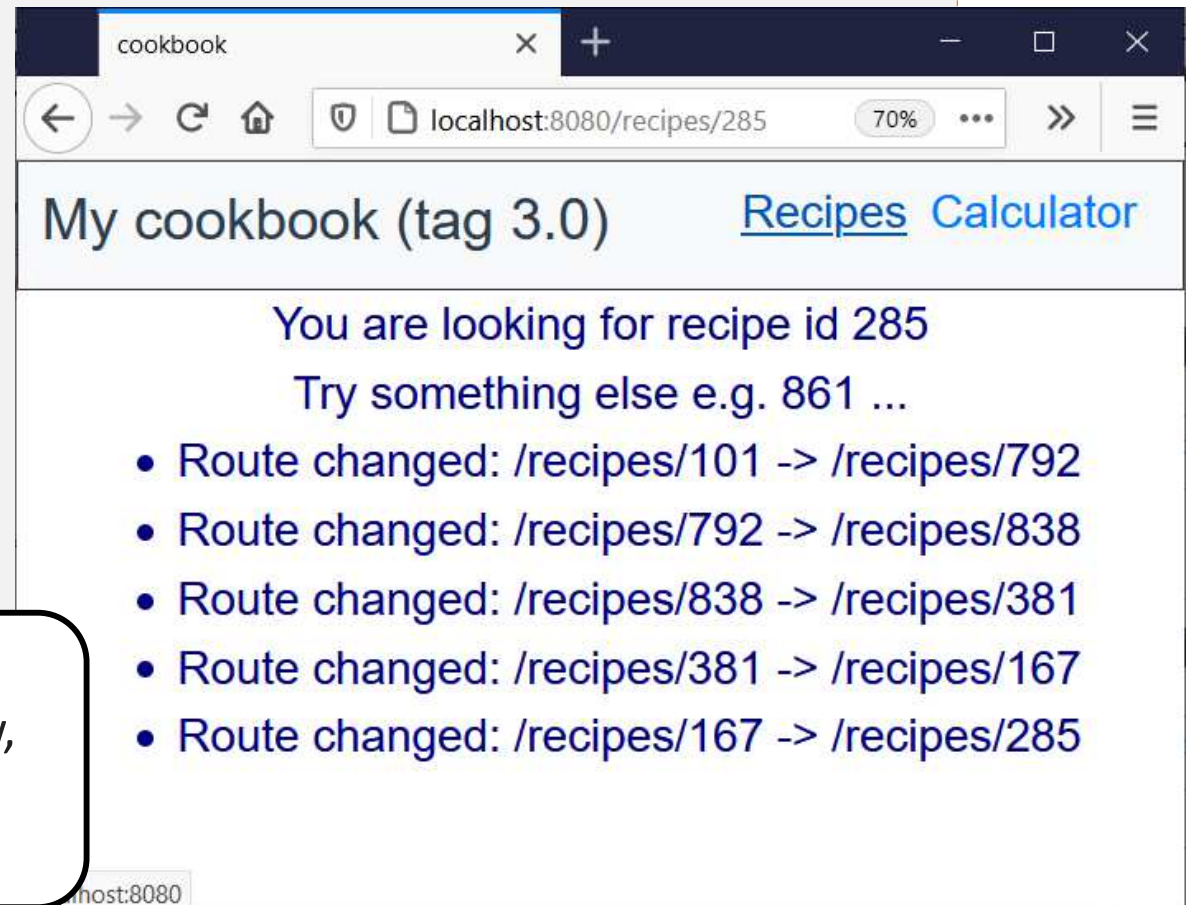
Recipes.vue

```
...
watch: {
  $route(to, from) {
    this.routeChanges.push(`Route changed: ${from.path} -> ${to.path} `);
    this.somethingElse = Math.floor(Math.random() * 1000);
  },
},
};
</script>

<style scoped>
* {
  font-size: 32px;
  color: navy;
}
</style>
```

watch opcija – moguće je pratiti promjene neke varijable!
Inače ne bi detektirali promjenu rute, jer Vue čuva komponentu u memoriji

Primijetiti da je stil primijenjen samo na view, npr. naslov je i dalje crne boje, zašto?



Zadatak za vježbu

Pogledajte *programmatic navigation*, te dodajte na `Recipes.vue` *button* koji će korisnika prebaciti na **slučajni recept**

Što će se dogoditi ako ručno upišemo npr. /recipes/101 ?

- Uočiti razliku:

Ako ga ne konfiguriramo, WS će vjerojatno vratiti 404.



GET /recipes/101

Web server

NotFound.vue

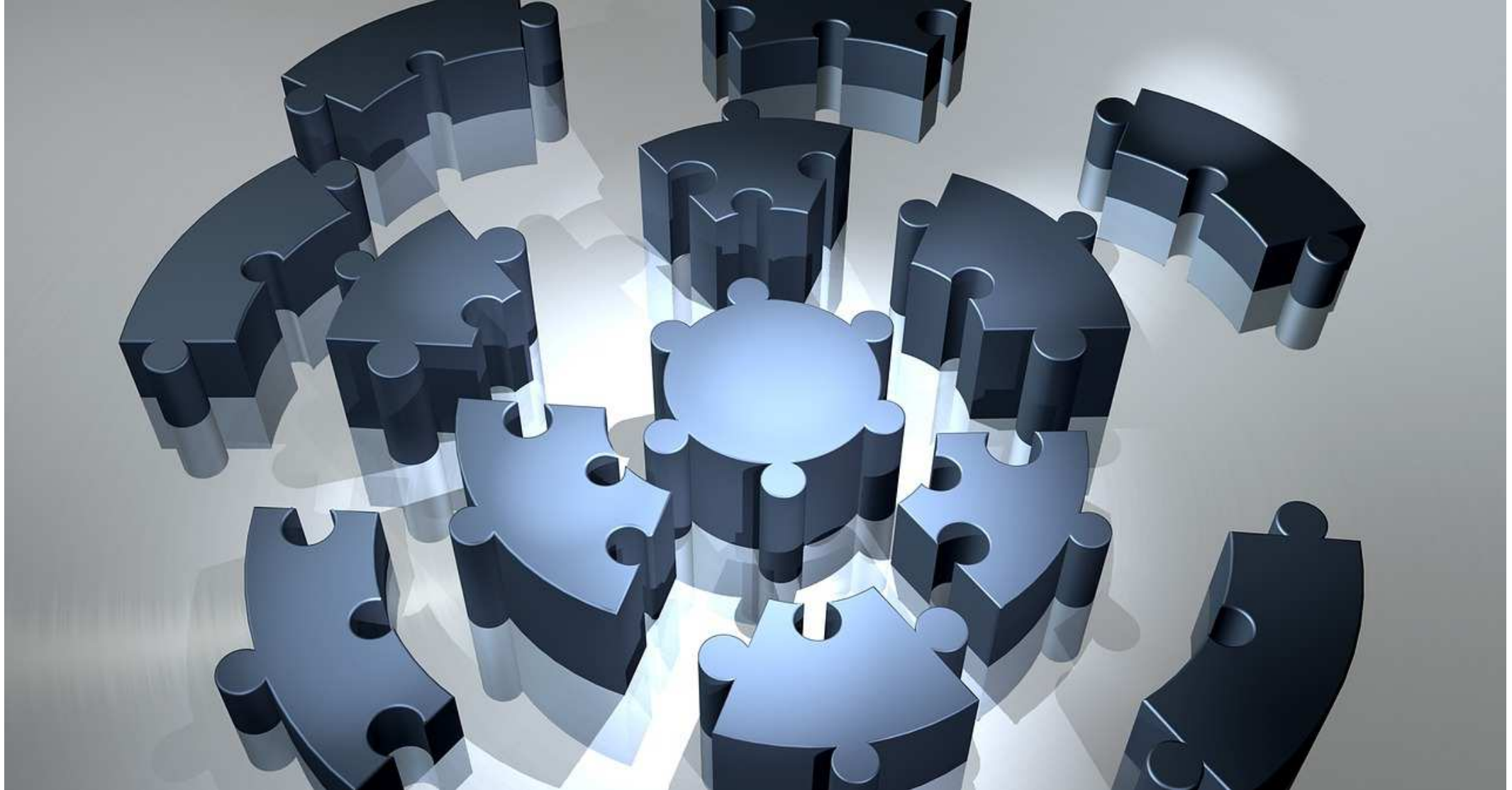
```
<template>
...
Try <router-link to="/recipes/101">
    recipe 101
</router-link>
</template>
```

Slide 15

DM1

ovo je komentar nastavno na zadatak za vježbu? zato sto je slajd izmedju zadatka za vjezbu i komponente

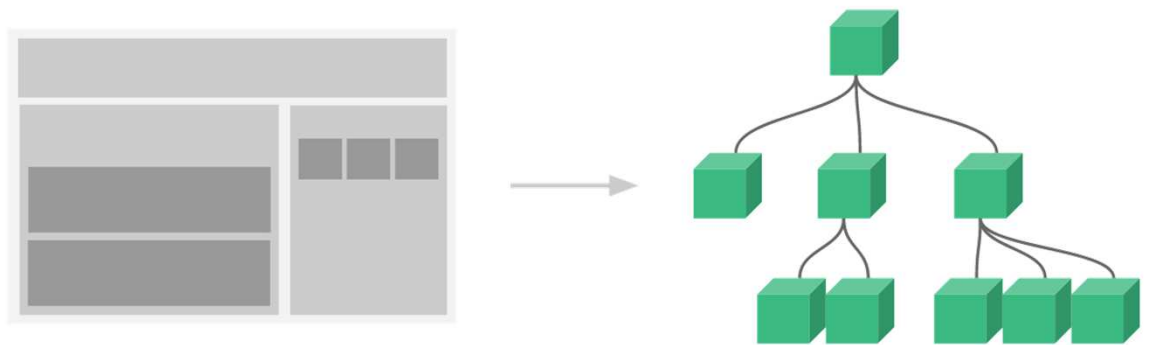
Danijel Mlinarić; 15.9.2021.



Komponente

Komponente

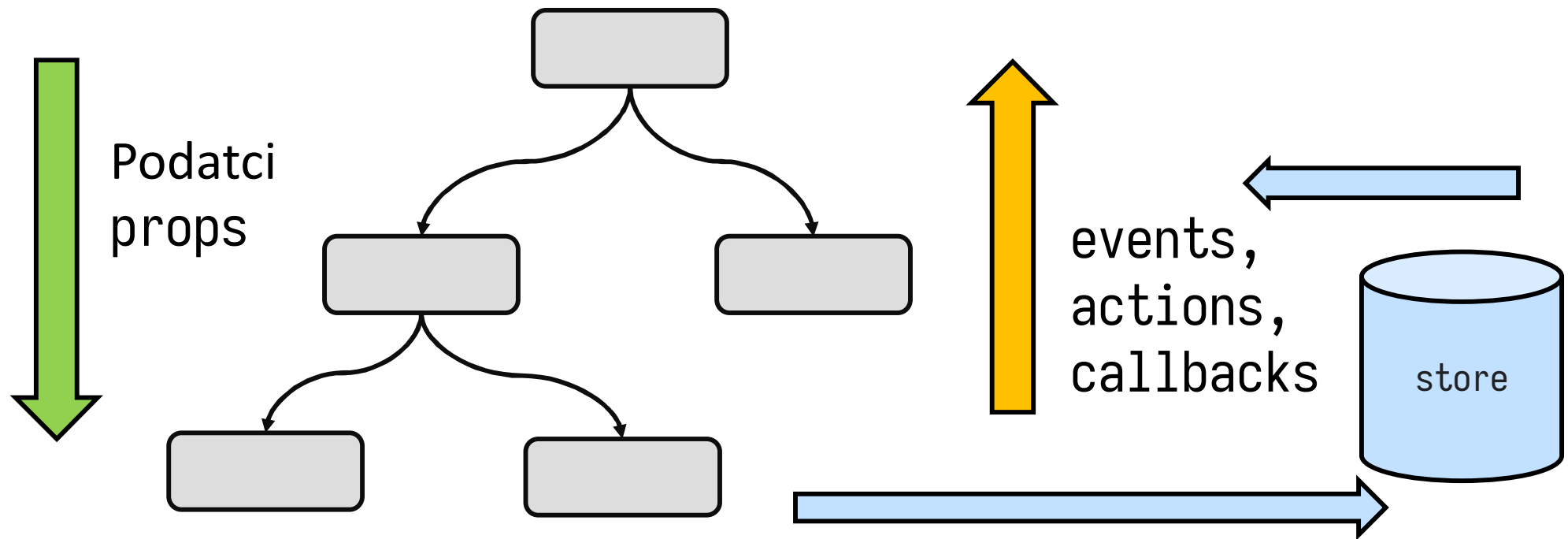
- Kada:
 - Imamo dijelove stranice/markupa koji se ponavljaju
 - Želimo razložiti problem/aplikaciju u manje probleme/aplikacije
- Komponenta je poput mini-aplikacije koja se onda pridijeli aplikaciji
 - Pridjeljujemo im oznaku (tag), tipično dvije riječi odvojene crticom kako bi izbjegli preklapanje s HTML oznakama
- Uobičajeno se aplikaciju ustroji u stablo ugniježđenih komponenti,
npr. zaglavlje, tijelo, podnožje,
popup/dialog, ...



<https://v3.vuejs.org/guide/component-basics.html>

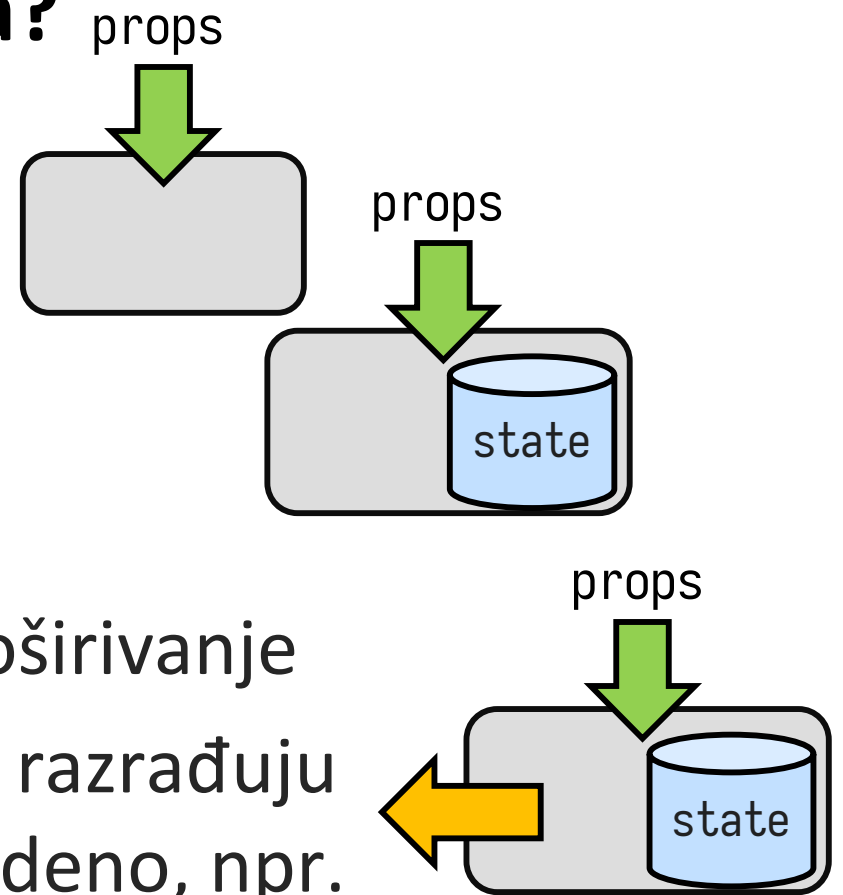
Obrasci komunikacije

- **Roditelj -> dijete**
 - Tipično se komponentama predaju vrijednosti
- **Dijete -> roditelj**
 - Rjeđe, ali ponekad nužno
 - Npr. React ima „*one-way data flow parent → child*“, ali...



Kakva može biti komponenta?

- Tri obilježja:
 1. Što prima?
 2. Ima li vlastito stanje?
 3. Suraduje li s drugima?
 - Po pitanju podataka
 - Omoguće li gniježđenje, proširivanje
- Različiti radni okviri više ili manje razrađuju nomenklaturu s obzirom na navedeno, npr.
 - React ima *class* i *functional* components
 - Angular: *smart* i *dumb/presentational/pure* komponente (samo primaju podatke i iscrtavaju)



Pretvorimo karticu recepta i kalkulator u komponente

tag: v4.0

- Prvo jednostavniju – Calculator
 - Vlastito stanje, ne prima parametre, ne surađuje

main.js

```
import { createApp } from "vue";
import App from "./App.vue";
import router from "./router";
import RecipeCard from './components/RecipeCard.vue';
import TempConverter from './components/TempConverter.vue';
```

```
const app = createApp(App);
app.use(router);
app.component('recipe-card', RecipeCard);
app.component('temp-converter', TempConverter);
app.mount("#app");
```

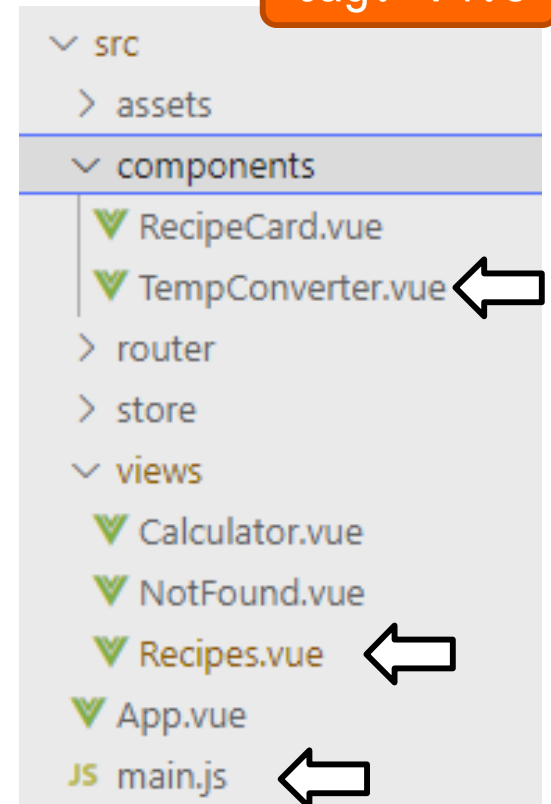
Konvencija: xxx-yyy

Prijavljujemo ih „globalno” čime postaju dostupne u cijeloj aplikaciji.
(moguće je i lokalno, pogledati:

<https://v3.vuejs.org/guide/component-registration.html>)

Views/Calculator.vue

```
<template>
  <div>
    <h2>Calculator</h2>
    <div class="container-fluid p-2 d-
flex justify-content-center">
      <temp-converter></temp-converter>
    </div>
  </div>
</template>
```



TempConverter.vue

- Prvo jednostavniju – Calculator
 - Ima vlasito stanje, ne prima parametre, ne surađuje

components/TempConverter.vue

```
<template>
  <div><h3>Temperature converter</h3>
    <form>
      <div><label>Celsius</label>
      <div><input
        v-model.number="tempCelsius"
        @keyup="c2f"/>
      </div>
    </div>
    <div><label>Fahrenheit</label>
    <div><input
      v-model.number="tempFahrenheit"
      @keyup="f2c"
    />
    </div></div>
  </form></div>
</template>
```

Izbačeni svi stilovi i sitnice koje nisu u fokusu, pogledati

src

Svaki dio se može izostaviti, npr. ovdje nema style

components/TempConverter

```
<script>
export default {
  data() {
    return {
      tempCelsius: 0,
      tempFahrenheit: 32,
    };
  },
  methods: {
    c2f() {
      this.tempFahrenheit = Math.round((this.tempCelsius * 9) / 5 + 32);
    },
    f2c() {
      this.tempCelsius = Math.round(((this.tempFahrenheit - 32) * 5) / 9);
    },
  },
};
</script>
```

struktura:

```
<template>
  ...
</template>
<script >
  ...
</script >
<style>
  ...
</style>
```

■ Domaća zadaća

- Primijenite komponente, dodajte joj „Log it” gumb koji smo već vidjeli
- Listu temperatura možete prikazati kao tooltip ili sl.
- Time će naša komponente proširiti svoje lokalno stanje (pored temperatura C i F)
- Stavite nekoliko istih komponenti na npr. tu istu stranicu
- Što se događa kada otidete na stranicu recepata i vratite se?
 - Pogledajte i keep-alive

Vue props

- Pogledajmo: <https://v3.vuejs.org/guide/component-props.html>

- Static:

```
<blog-post title="My journey with Vue"></blog-post>
```

- Dynamic:

```
<!-- Dynamically assign the value of a variable -->
<blog-post :title="post.title"></blog-post>

<!-- Dynamically assign the value of a complex expression -->
<blog-post :title="post.title + ' by ' + post.author.name"></blog-post>
```

- Mogu se predavati različiti tipovi: number, boolean, array,...

- Object:

- Itd. pogledati dokumentaciju

```
<!-- Even though the object is static, we need v-bind to tell Vue that -->
<!-- this is a JavaScript expression rather than a string. -->
<blog-post
  :author="{
    name: 'Veronica',
    company: 'Veridian Dynamics'
  }"
></blog-post>

<!-- Dynamically assign to the value of a variable. -->
<blog-post :author="post.author"></blog-post>
```


Citat iz dokumentacije:

One-Way Data Flow

All props form a **one-way-down binding** between the child property and the parent one: when the parent property updates, it will flow down to the child, but not the other way around. This prevents child components from accidentally mutating the parent's state, which can make your app's data flow harder to understand.

In addition, every time the parent component is updated, all props in the child component will be refreshed with the latest value. This means you should **not** attempt to mutate a prop inside a child component. If you do, Vue will warn you in the console.

There are usually two cases where it's tempting to mutate a prop:

1. The prop is used to pass in an initial value; the child component wants to use it as a local data property afterwards. In this case, it's best to define a local data property that uses the prop as its initial value:

```
1  props: ['initialCounter'],
2  data() {
3    return {
4      counter: this.initialCounter
5    }
6  }
```

js

2. The prop is passed in as a raw value that needs to be transformed. In this case, it's best to define a computed property using the prop's value:

```
1  props: ['size'],
2  computed: {
3    normalizedSize() {
4      return this.size.trim().toLowerCase()
5    }
6  }
```

js

Note

Note that objects and arrays in JavaScript are passed by reference, so if the prop is an array or object, mutating the object or array itself inside the child component **will** affect parent state.

RecipeCard.vue 1/2 (script, style)

- Nema stanje, prima parametre, ne surađuje

components/RecipeCard.vue

```
<script>
export default {
  props: [
    "id", "image", "name", "description", "cookTime",
    "prepTime", "recipeYield", "datePublished", "url",
    "ingredients"
  ],
  computed: {
    idUrl() {
      return '/recipes' + this.id;
    }
  }
};
</script>
<style scoped>
  div.card-body .badge {
    white-space: pre-wrap;
  }
</style>
```

Template na sljedećem
slajdu, uočiti props

Scoped – lokalni stil,
utječe samo na ovu
komponentu!

views/Recipes.vue

```
<template>
  <div v-if="selectedRecipe">
    Recipe #{{ id }}
    <recipe-card :key="selectedRecipe.id"
      v-bind="selectedRecipe"
    ></recipe-card>
  </div>
  <div v-else>
    <h2>All recipes ({{allRecipes.length}})</h2>
    <div>
      <recipe-card v-for="recipe in allRecipes"
        :key="recipe.id"
        v-bind="recipe"
      ></recipe-card>
    </div>
  </div>
</template>
```

„If you want to pass all
the properties of an object
as props, you can use v-
bind without an argument
(v-bind instead of :prop-
name).”

Slide 25

DM4

props se mogu definirati kao name: {type, default, required}

Danijel Mlinarić; 15.9.2021.

RecipeCard.vue 2/2 (template)

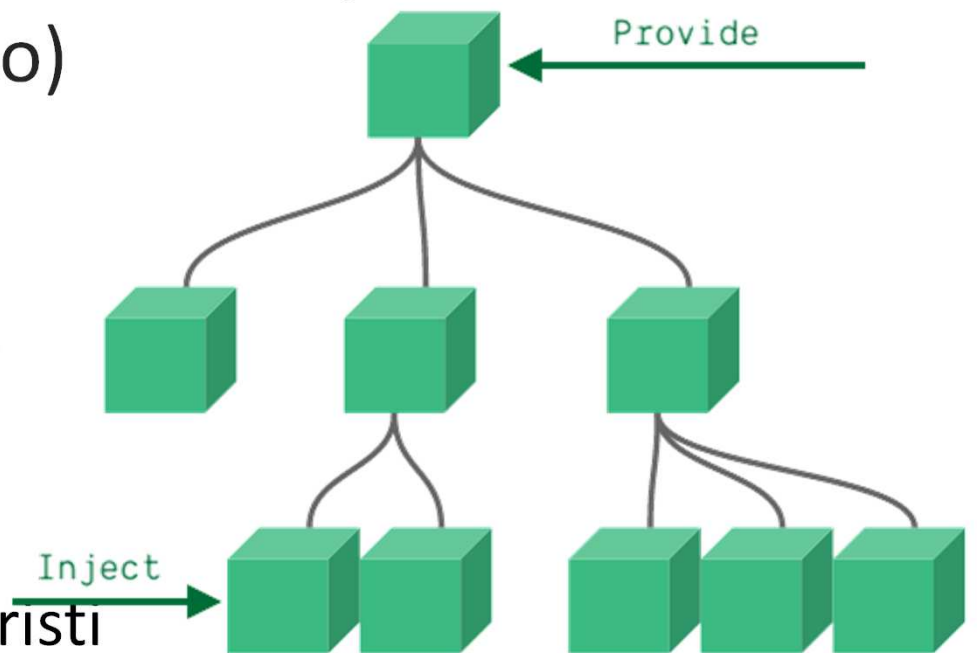
components/RecipeCard.vue

```
<template>
  <div class="card mt-2 mr-2" style="width: 230px">
    <router-link :to="'/recipes/' + id" class="mr-3">
      
    </router-link>
    <div class="card-body">
      <h5 class="card-title">{{ name }}</h5>
      <p class="card-text">{{ description }}</p>
      <span class="badge badge-primary">
        Cook/prep time: {{ cookTime }}/{{ prepTime }}
      </span>
      <span class="badge badge-secondary">Yield: {{ recipeYield }}</span>
      <span class="badge badge-success">Published: {{ datePublished }}</span>
      <a :href="url" target="_blank" class="card-link">
        <span class="badge badge-success">{{ url.substring(0, 20) }}...</span>
      </a>
    </div>
    <details v-if="ingredients">
      <summary><h3>Ingredients</h3></summary>
      <ul class="list-group list-group-flush">
        <li v-for="ingredient in ingredients" :key="ingredient" class="list-group-item">{{ ingredient }} </li>
      </ul>
    </details>
  </div>
</template>
```

Nema ničeg posebno novog,
jednostavno koristimo props
kao normalne varijable kod
definiranja obrasca odnosno
iscrtavanja

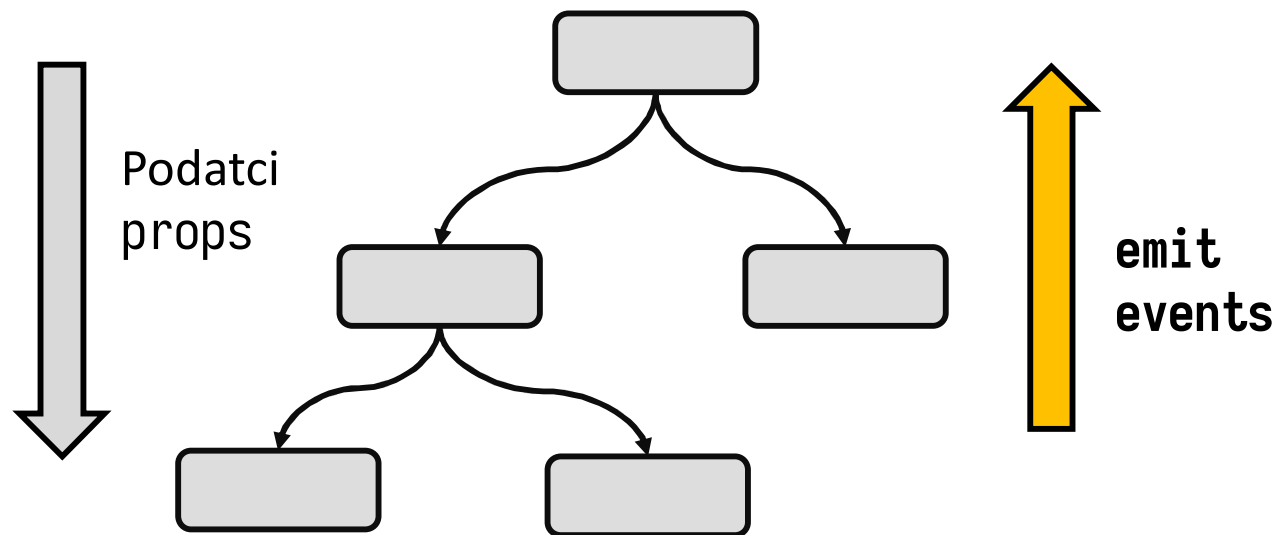
Provide/inject

- <https://v3.vuejs.org/guide/component-provide-inject.html>
- Ako imamo duboku hijerarhiju komponenti i želimo poslati svojstvo (*prop*) ne samo neposrednom djetetu onda to može biti zamorno („štafeta”)
- *provide*: roditelj pruža svojstvo svim svojim nasljednicima (cijelo podstablo)
- *inject*: dijete prijavi što koristi
- „*long-range props*”:
 - Roditelji ne moraju znati tko koristi
 - Djeca ne moraju znati odakle dolazi



Emitiranje događaja

- „Suprotni smjer” – ponekad je potrebno iz komponente nešto javiti roditelju
- Dodajmo u recipe-card gumb za brisanje recepta
 - Komponenta **emitira** događaj brisanja svom roditelju
 - Roditelj se povezuje na emitirane događaje i poduzima odgovarajuće radnje – ovdje izbacuje element iz polja



Emitiranje događaja „deleteRecipe”

components/RecipeCard.vue

Na sljedećem slajdu

```
<template>
  <small-card>
    (... isto kao prije ...)
    <button class="btn btn-danger"
      @click="deleteRecipe">
      Delete
    </button>
  </small-card>
</template>
<script>
export default {
  emits: ["deleteRecipe"],
  (... )
  methods: {
    deleteRecipe() {
      this.$emit('deleteRecipe', {id: this.id});
    }
  }
};
</script>
```

1

2

views/Recipes.vu

```
<template>
  <div v-if="selectedRecipe">
    <h2>Recipe #{{ id }}</h2>
    <recipe-card :key="selectedRecipe"
      v-bind="selectedRecipe"
      @delete-recipe="deleteRecipe"
    ></recipe-card>
  </div></div>
  <div v-else>
    <h2>All recipes ({{allRecipes.length}})</h2><hr><div>
      <recipe-card v-for="recipe in allRecipes"
        :key="recipe.id" v-bind="recipe"
        @delete-recipe="deleteRecipe"
      ></recipe-card>
    </div></div>
</template>
<script>...
methods: {
  deleteRecipe (args) {
    this.allRecipes = this.allRecipes.filter(x => x.id !== args.id);
    if (this.selectedRecipe && this.selectedRecipe.id === args.id) {
      this.selectedRecipe = null;
    }
  }
  ...
}
</script>
```

3

„Like components and props, event names provide an automatic case transformation. If you emit an event from the child component in camel case, you will be able to add a kebab-cased listener in the parent”

Utori (*Slots*)

<https://v3.vuejs.org/guide/component-slots.html>

- Komponente mogu definirati mjesto (utor, slot) u koje će se ugraditi drugi sadržaj (string, HTML, druge komponente...)
- Mi ćemo definirati „karticu” – standardni element našeg korisničkog sučelja
 - u nju možemo stavljati razne sadržaje (recept, kalkulator)

components/SmallCard.vue

```
<template>
  <div>
    <slot></slot>
  </div>
</template>
<style scoped>
  div {
    border: 1px gray solid;
    border-radius: 3px;
    width: 250px;
    box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.19);
    padding: 10px;
    margin: 5px;
  }
</style>
```

Temperature converter

Celsius

Fahrenheit



Buckwheat Cheese Straws

These cheese straws look like wispy tree branches. Wayne calls them cheese twigs, and they never last very long around here. Crispy, cheddar-flecked, and rustic - it's the buckwheat flour that lends these slender creations their convincing shade of brownish gray.

Cook/prep time:
PT10M/PT60M

Yield: Makes about 4 dozen straws.

Published: 2009-08-22

<http://www.101cookbo...>

Ingredients

Delete

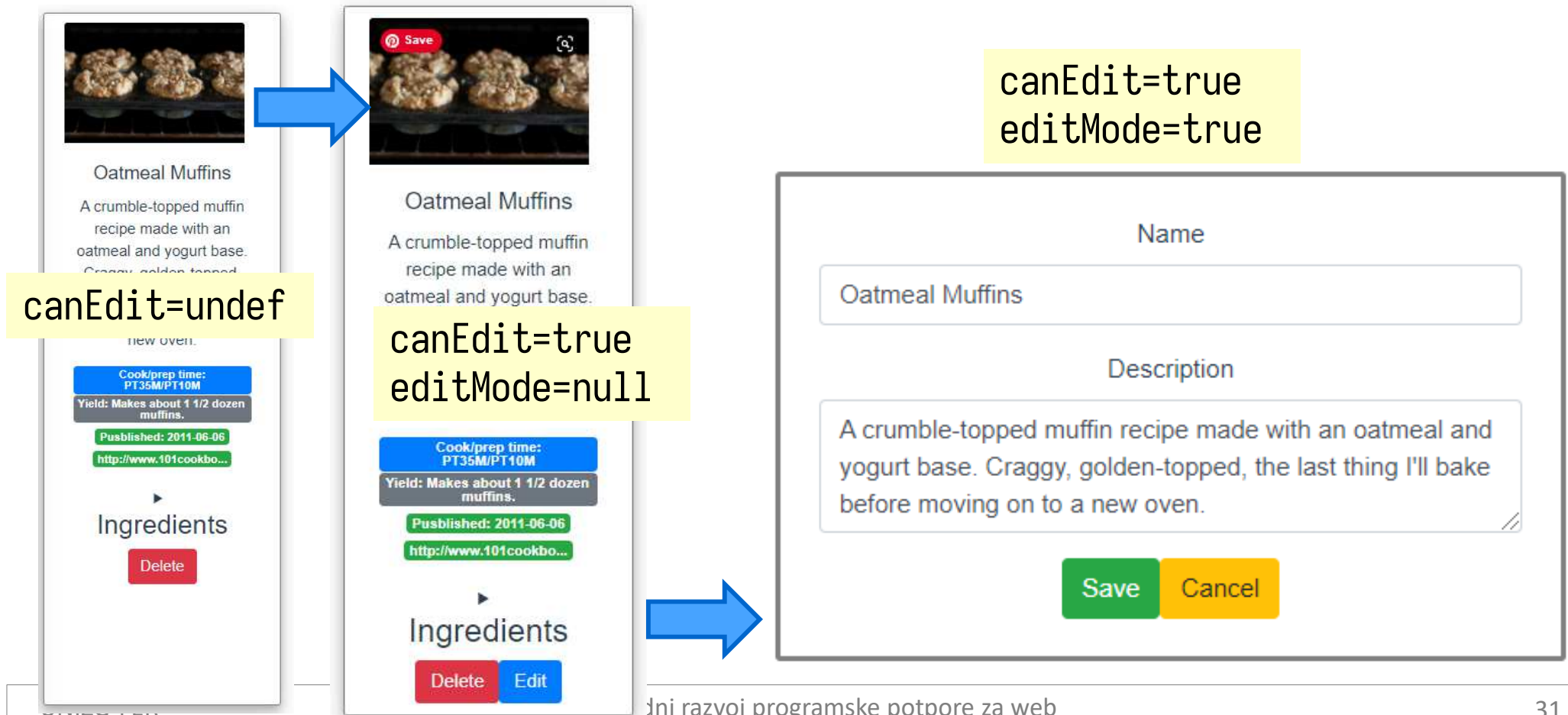
Prijavimo u main.js kao i druge komponente.
Psotoje i napredne opcije, npr. scoped slots, ZOKŽNV

DM5

po novoj verziji je v-slot
Danijel Mlinarić; 15.9.2021.

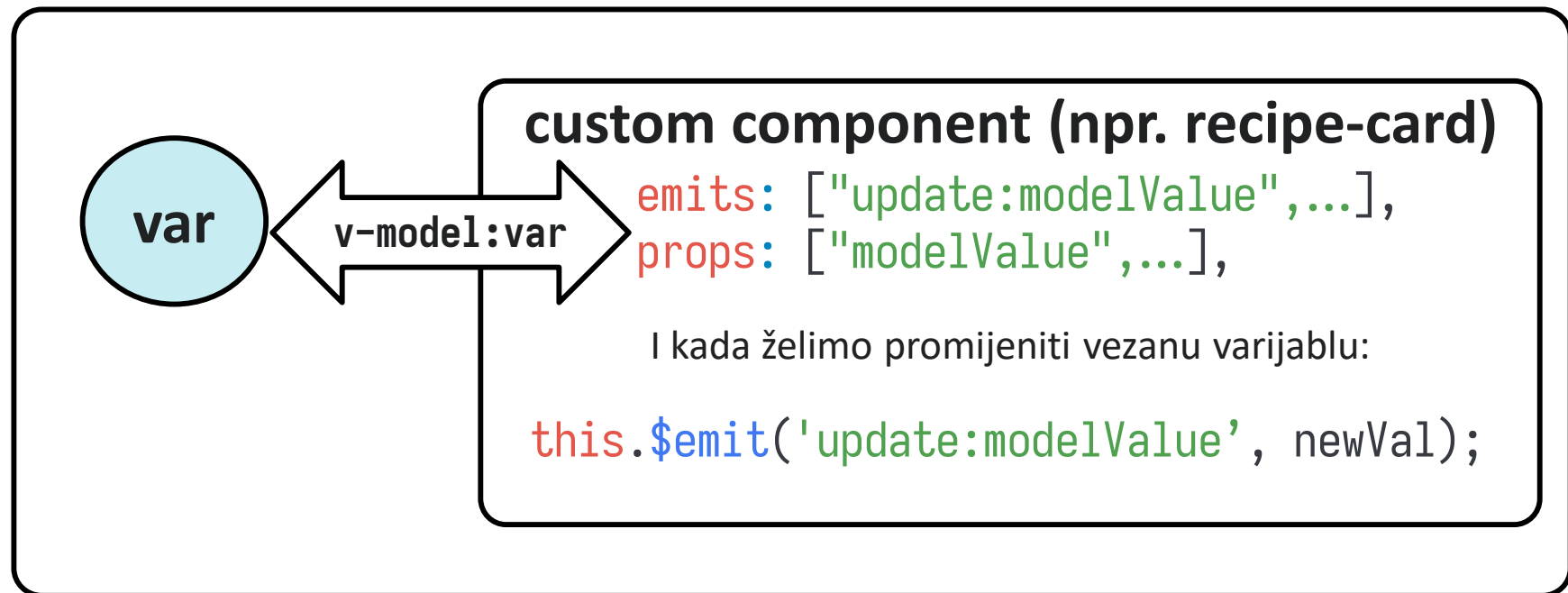
Omogućimo izmjenu recepta!

- Radi jednostavnosti: samo name i description
- Gumb za izmjenu vidljiv samo kada se odabere recept
 - Dodat ćemo prop canEdit koji će to definirati
- Komponenta mijenja izgled s obzirom na editMode



Omogućimo izmjenu recepta!

- Želimo da se izmjene unutar komponente odraze na vanjske podatke odnosno selektirani recept
- Gumb za izmjenu vidljiv samo kada se odabere recept
 - Dodat ćemo prop `canEdit` koji će to definirati
- Komponenta mijenja izgled s obzirom na `editMode`



Povežimo prvo „vanjsku” varijablu

views/recipes.vue

```
<template>
  <div v-if="selectedRecipe">
    <div>
      <recipe-card :key="selectedRecipe.id"
        v-model="allRecipes[selectedRecipeIndex]"
        @delete-recipe="deleteRecipe"
        can-edit="true"
      ></recipe-card>
    </div>
  </div>
  <div v-else>
    <div>
      <recipe-card v-for="(recipe, index)
        in allRecipes"
        :key="recipe.id"
        v-model="allRecipes[index]"
        @delete-recipe="deleteRecipe"
      ></recipe-card>
    </div>
  </div>
</template>
...
```

Nema smisla povezati na lokalnu selectedRecipe jer se to onda neće odraziti na naše polje – zato uvodimo selectedRecipeIndex da možemo odmah direktno povezati s elementom polja `allRecipes[selectedRecipeIndex]`

```
...
<script>
export default {
  ...
  data() {
    return {
      selectedRecipe: null,
      selectedRecipeIndex: -1,
    };
  },
  watch: {
    $route(to, from) {
      this.selectedRecipe = this.allRecipes.find( x => x.id == this.$route.params.id);
      this.selectedRecipeIndex = this.allRecipes.findIndex( x => x.id == this.$route.params.id);
    },
  },
  ...
}
```

Na neki način zloporabimo – ovdje bi nam bili dovoljni props iz v5.0 jer se predaju samo roditelj->dijete jer canEdit nije postavljen pa se ne može ni promijeniti vezana vrijednost

... zatim recipe-card

views/recipes.vue

```
<template>

  <small-card v-if="!editMode">
    (... isto kao prije ...)
    <button v-if="canEdit" @click="editMode = true">
      Edit
    </button>
  </small-card>
  <div v-if="editMode" class="editForm">
    <form @submit.prevent="submitChanges">
      <input type="text" v-model="name">
      <textarea v-
model="description"></textarea>
      <button type="submit">
        Save
      </button>
      <button @click.stop="exitSingleRecipe()">
        Cancel
      </button>
    </form>
  </div>
</template>
```

Izbačeno dosta koda koji nije u fokusu, vidjeti src, jedino u read-only dijelu sad koristimo modelValue.PROP, npr. modelValue.name, modelValue.url, itd.

Save će bubble-up u form submit, a Cancel neće zbog stop modifera

```
<script>
export default {
  emits: ["deleteRecipe", "update:modelValue"],
  props: ["modelValue", "canEdit"],
  data() {
    return {
      editMode: false,
      name: this.modelValue.name,
      description: this.modelValue.description
    }
  },
  methods: {
    exitSingleRecipe() {
      this.$router.push({ path: '/recipes' });
    },
    submitChanges() {
      this.$emit('update:modelValue', {
        id: this.modelValue.id,
        image: this.modelValue.image,
        name: this.name,
        description: this.description,
        cookTime: this.modelValue.cookTime,
        prepTime: this.modelValue.prepTime,
        recipeYield: this.modelValue.recipeYield,
        datePublished: this.modelValue.datePublished,
        url: this.modelValue.url,
        ingredients: this.modelValue.ingredients
      });
      this.exitSingleRecipe();
    }
  }
}
...
```

Slide 34

DM6

kuzim, ali bas mi je `update:modelValue` grd, moze se spomenuti state model. U biti je li bitno za recipe promjena u kartici?

Danijel Mlinarić; 15.9.2021.

DM7

sad vidim sljedeći slajd spominjes stanja :)

Danijel Mlinarić; 15.9.2021.

Stanje aplikacije (*state*)

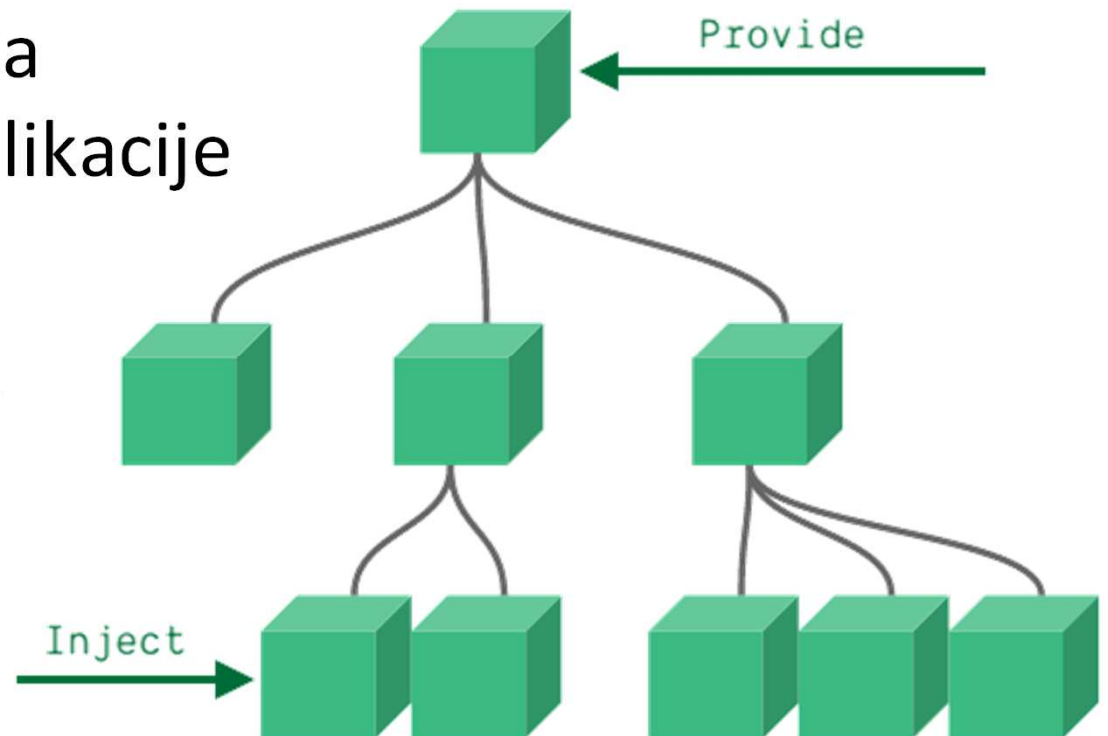


Stanje

- Stanje = trenutni radni skup podataka aplikacije
- Može biti:
 - A. Lokalno stanje** (komponenti, stranica,...)
 - Odnosi se na samo jednu komponentu
 - Npr. logs u Calculator, isEdit u RecipeCard, allRecipes u Recipes
 - B. Globalno stanje**
 - Odnosi se na (potrebno u) više komponenti
 - Npr. sadržaj košarice kod kupovine, prijavljeni korisnik i uloge
 - Je li allRecipes potreban na više mjesta?
- Tipično u aplikaciji imamo i globalno i lokalno stanje

Kako ostvariti globalno stanje? (1/2)

- *Provide/inject?*
- Moguće, ali:
 - U korijensku komponentu bi „nagurali” previše koda
 - Teže upravljivo, teži razvoj
 - Nisu jasni obrasci korištenja, izmjene
 - Podložnije pogreškama
 - Prikladno za manje aplikacije



Kako ostvariti globalno stanje?

(2/2)

- FTSE 😊: uvodimo vanjski entitet koji se bavi globalnim stanjem

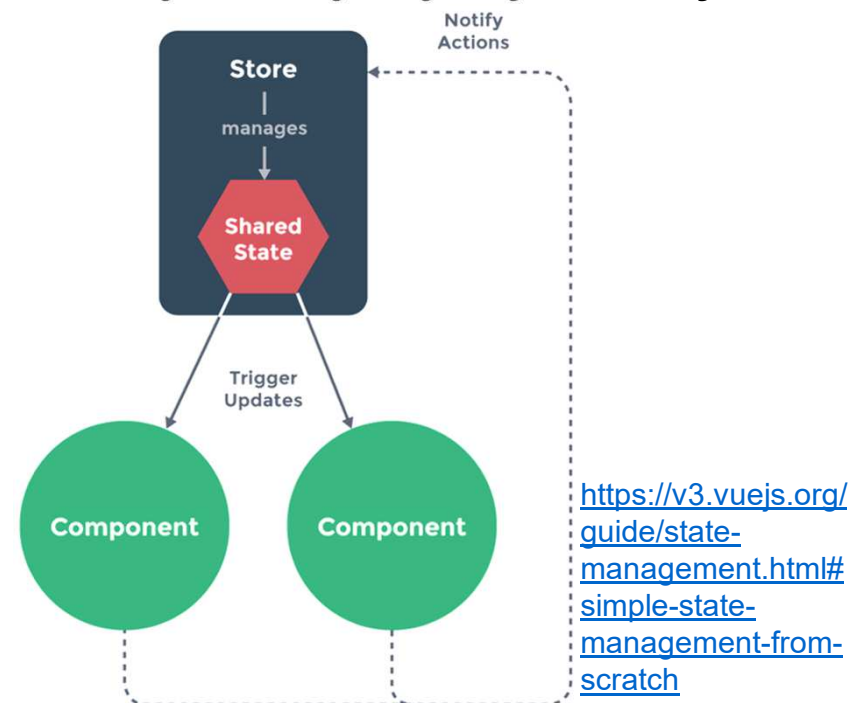
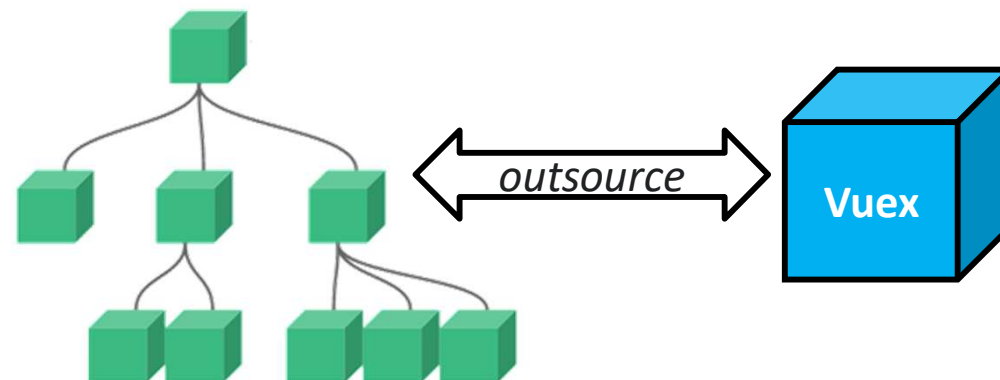
- *Global singleton*

- Provodi nekakav **obrazac** upravljanja stanjem:

- **Strogo definirani načini korištenja**, čitanja i mijenjanja stanja
- Smanjuje mogućnost pogreške
- Olakšava razvoj (devtools, debugging, time-travel)

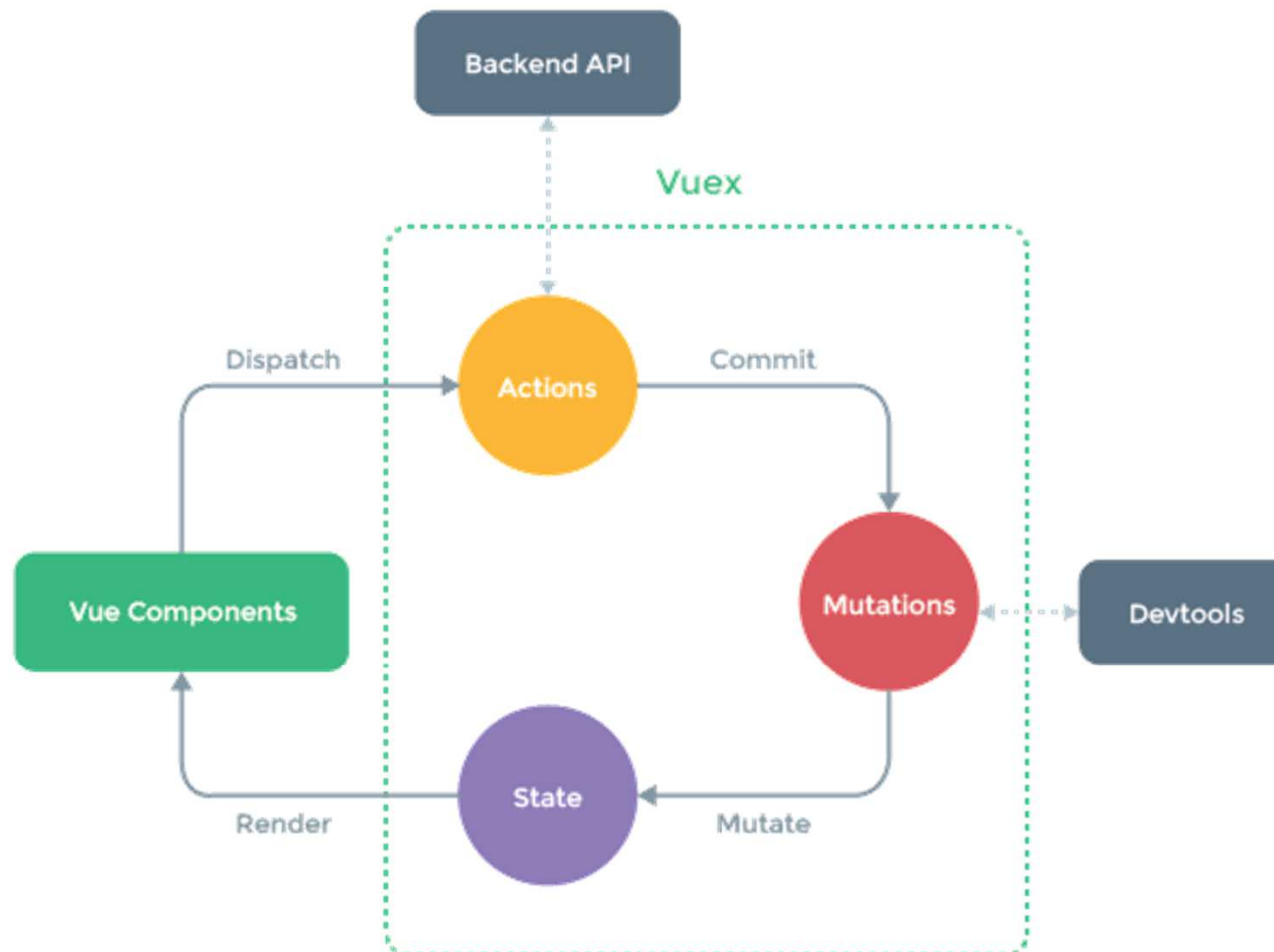
- Npr.:

- Redux, MobX
- Vuex
- NgRx



■ Vuex

- „state management pattern + library for Vue.js applications”
- „strogo definirani način korištenja” (obrazac):



<https://vuex.vuejs.org/#what-is-a-state-management-pattern>

Vuex

- An example of the [most basic Vuex counter app \(opens new window\)](https://vuex.vuejs.org/guide/#the-simplest-store).

<https://vuex.vuejs.org/guide/#the-simplest-store>

```
const store = new Vuex.Store({
  state: {
    count: 0
  },
  mutations: {
    increment: state => state.count++,
    decrement: state => state.count--
  }
})
new Vue({
  el: '#app',
  computed: {
    count () { return store.state.count }
  },
  methods: {
    increment () {
      store.commit('increment')
    },
    decrement () {
      store.commit('decrement')
    }
  }
})
```

Objekt
stanja

```
<div id="app">
  <p>{{ count }}</p>
  <p>
    <button @click="increment">+</button>
    <button @click="decrement">-</button>
  </p>
</div>
```



U ovom minimalnom primjeru direktno referenciramo **store** varijablu, ali inače ćemo prijaviti **store** i Vuex će injektirati **store** u naše komponente te ćemo koristiti npr. `this.$store.commit('increment')`

„strogo def.“: mijenjamo **count** putem mutacija a ne direktno!

Na taj način možemo pratiti promjene, lakši razvoj, debugging, itd.

Slide 40

DM9

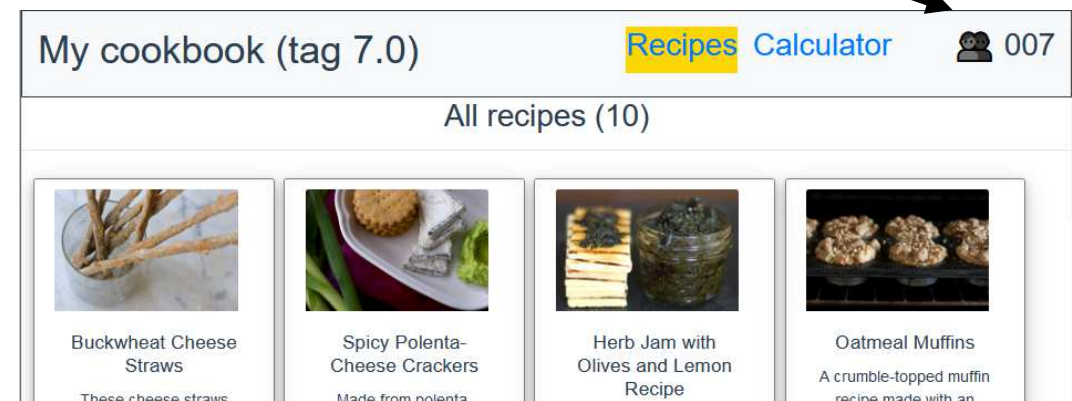
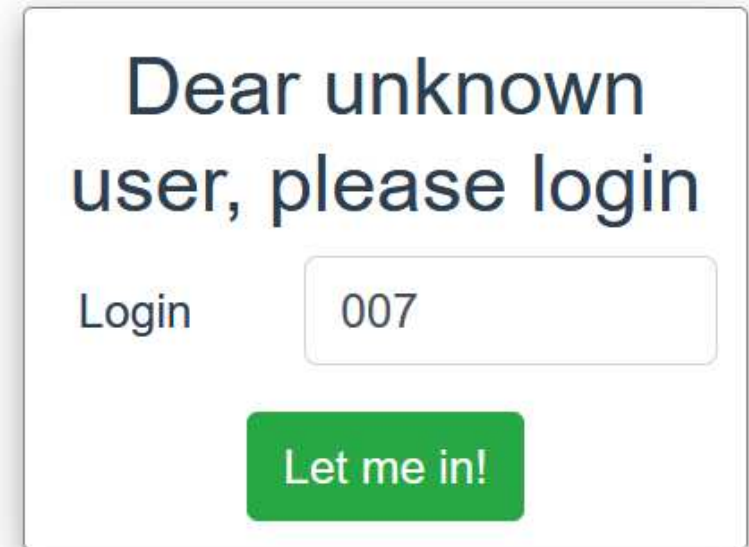
bilo bi zgodno spomenuti zasto su mutacije ok, mi smo ih na kraju izbacili i napravili svoj store, samo nam je kompliciralo pisanje :D

Danijel Mlinarić; 15.9.2021.

Dodajmo prijavljenog korisnika!

■ Plan:

- Store sadrži ime prijavljenog korisnika
- Napraviti ćemo novu komponentu **UserLogin.vue** za prijavu
- U App.vue:
 - Ako **nije** prijavljen - `<user-login/>`
 - Ako je prijavljen – kao prije, ali dodajemo username gore desno
- Store ćemo registrirati i on će biti vidljiv svim komponentama, u našem slučaju trebaju ga **App** i **UserLogin**
- **Store pravila:**
 - Čitamo pomoću **getters**
 - Mijenjamo stanje pomoću **mutations**



Store

store/index.js

```
import { createStore } from "vuex";
export default createStore({
  state: {
    user: null,
  },
  mutations: {
    setUser(store, newUser) {
      store.user = newUser;
    }
  },
  getters: {
    user(store) {
      return `${store.user}`;
    },
    isAuth(store) {
      return !!store.user;
    }
  }
});
```

main.js

```
...
import store from "./store";
import router from "./router";
...
const app = createApp(App);
app.use(store);
app.use(router);
```

Ovime će `store` postati dostupan u svim komponentama kao: `this.$store`

Metode u `mutations` primaju staro stanje i eventualni payload.
Iz komponente ćemo pozvati s:
`this.$store.commit('setUser', '007')`

Getters ćemo pozvati s npr.:
`this.$store.getters.isAuth`

Slide 42

DM8

stvarno je ikona u return za store? :)

Danijel Mlinarić; 15.9.2021.

App.vue

App.vue

```
<template>
  <div v-if="isAuth">
    <div>
      ... isto kao prije, osim:
      <div class="ml-5">{{ $store.getters.user }}</div>
    </div>
    <div v-else class="d-flex justify-content-center p-5">
      <user-login class="mt-5"></user-login>
    </div>
  </template>
<script>
import UserLogin from './components/UserLogin.vue';
export default ({
  components: {
    UserLogin
  },
  computed: {
    isAuth() {
      return this.$store.getters.isAuth;
    }
  }
})
</script>
```

Dodajemo ikonicu i username gore desno.

Sintaksni detalj: dodali smo class u custom component tag!?
Što će biti s njim?

Primijetite lokalnu komponentu!
Samo App.vue treba user-login pa smo ju prijavili lokalno.

Moglo se i bez ovoga, direktno u **v-if** koristiti isti izraz.

UserLogin.vue

Components/UserLogin.vue

```
<template><small-card><h3>Dear unknown user, please login</h3>
  <form @submit.prevent="login">
    <div class="form-group row">
      <label for="celsius" class="col-sm-4 col-form-label">Login</label>
      <div class="col-8">
        <input class="form-control"
          v-model.trim="username" />
      </div>
    </div>
    <button class="btn btn-success" type="submit">Let me in!</button>
  </form>
</small-card></template>
<script>
export default {
  data() {
    return { username: "" };
  },
  methods: {
    login() {
      this.$store.commit('setUser', this.username);
    }
  },
};
</script>
```

Opet koristimo našu karticu 😊

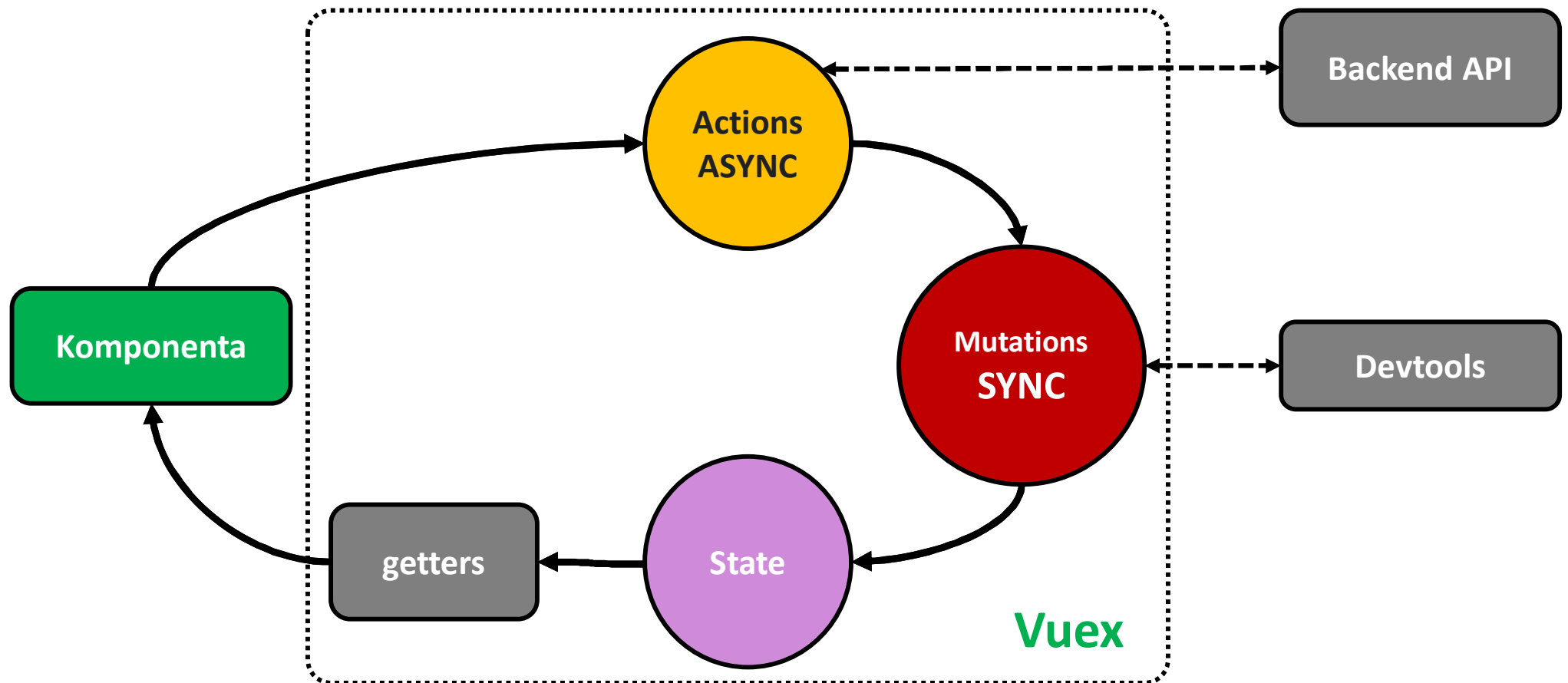
Ništa novo, radimo trim

Lokalna varijabla, ništa novo

Putem mutacije postavljamo korisnika. Ništa više nije potrebno napraviti - App.vue mijenja prikaz i ova komponenta se uklanja

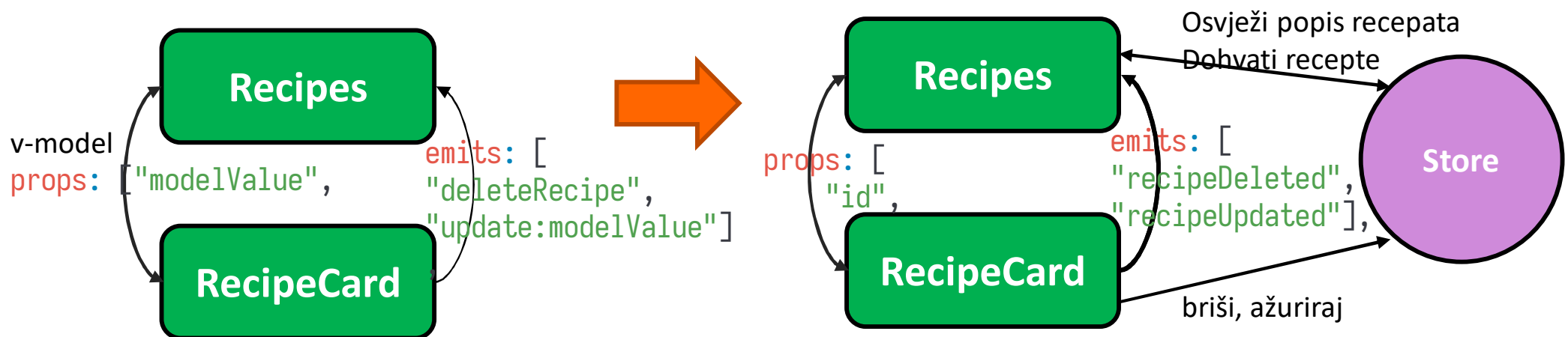
■ Trenutna situacija – imamo mutations i getters

- Ali: mutations moraju biti **sinkrone**! (sekvencijalne promjene stanja)
- Uvodimo: **actions** koje onda pozivaju mutations
 - Smatra se dobrom praksom uvijek koristiti actions



Izmjene

- Dohvat recepata je **asinkrona** funkcija, premjestiti ćemo ju u store (**actions**)
 - Također, tamo ćemo smjestiti i brisanje i ažuriranje recepata (iako oni u ovom primjeru nisu **async**, svejedno ćemo ih staviti u **actions**, pa pozvati **mutations**, u skladu s dobrom praksom)
- Promijenit ćemo ustroj i povezivanje **Recipes** i **RecipeCard**:
 - **prije** su bili spregnuti putem v-model, emit, refresh iz **Recipes**
 - **sada** obje imaju pristup storeu i **Recipe** predaje samo **id**



Pogledajmo prvo Store:

store/index.js

```
export default createStore({
  state: { user: null,
    allRecipes: [] },
  actions: {
    async refreshRecipes(context) {
      if (context.getters.allRecipes.length === 0) {
        try {
          let response = await fetch("http://127.0.0.1:3000/recipes");
          if (response.ok) {
            let recipes = await response.json();
            recipes = recipes.slice(0, 10);
            context.commit("setRecipes", recipes);
          } else { throw new Error("HTTP-Error: " + response.status); }
        } catch (error) { console.error(error); }
      }
    },
    deleteRecipe(context, {id}) {
      context.commit("setRecipes", context.getters.allRecipes.filter((x) => x.id !== id));
    },
    updateRecipe(context, recipe) {
      let newArray = context.getters.allRecipes.map(x => x.id === recipe.id ? recipe : x);
      context.commit("setRecipes", newArray);
    },
  },
});
```

Trebat će nam za
Recipes

```
mutations: { ...
  setRecipes(store, recipes) {
    store.allRecipes = recipes;
  },
},
getters: { ...
  allRecipes(store) {
    return store.allRecipes || [];
  },
  getRecipeById: (state, getters) => (id) => {
    return getters.allRecipes.find(rcp => rcp.id === id);
  },
},
```

Trebat će nam za
RecipeCard,
prvi primjer
gettera s
argumentima

Trivijalno keširanje, razmisliti o opcijama
sinkronizacije s backendom...

Slide 47

DM10

axios za komunikaciju s backendom?

Danijel Mlinarić; 15.9.2021.

DM11

context me ovdje podsjeća na node s web1, ali to je na serveru pa smo imali context koji je mogao biti db, servis, mock...

Danijel Mlinarić; 15.9.2021.

... zatim RecipeCard - on radi samo na temelju

tag: v7.0

id i canEdit (i veze sa Storeom):

components/RecipeCard.js

Ljubaznost: možda će post-festum zanimati onoga tko me pozvao. Primijetiti promjenu naziva eventa - sad je pasiv

```
export default {
  emits: ["recipeDeleted", "recipeUpdated"],
  props: ["id", "canEdit"],
  data() {
    return {
      editMode: false,
      recipe: {url:
    },
  },
  methods: {
    async deleteRecipe() {
      await this.$store.dispatch('deleteRecipe',
        { id: this.id });
      this.$emit('recipeDeleted', { id: this.id });
    },
    async submitChanges() {
      await this.$store.dispatch('updateRecipe', this.recipe);
      this.$emit('recipeUpdated', this.recipe);
      this.exitSingleRecipe();
    },
    exitSingleRecipe() { this.$router.push({ path: '/recipes' }); },
  },
  async created() {
    this.recipe = {... await this.$store.getters.getRecipeById(this.id) };
  }
};
```

Inicijalno prazan, korisnik to neće ni vidjeti, ali na jedno mjestu koristimo url.substring, pa da ne bi bilo undefined.substring

Template je isti, samo sada koristimo recipe.property, npr.

```
<span class="badge badge-secondary">Yield: {{ recipe.recipeYield }}</span>
```

Također, u edit formi smo izbacili vlastite varijable već vežemo direktno na recipe.name i recipe.description – to je u redu jer je taj recipe klonirani element polja iz allRecipes, dakle lokalna kopija, npr.

```
... <div v-if="editMode" class="editForm">
  <form @submit.prevent="submitChanges">
    <div class="form-group">
      <label for="name">Name</label>
      <input type="text" class="form-control" id="name" placeholder="name" v-model="recipe.name">
    </div> ...
```

Kloniramo putem spread operatora.
Mogli smo i u mounted, ali created je ranije u lifecycleu

... i konačno - Recipes

views/Recipes

```
export default {
  props: ["id"],
  data() {
    return { selectedRecipe: null,
      selectedRecipeIndex: -1,
    };
  },
  computed: {
    allRecipes() { return this.$store.getters.allRecipes; }
  },
  methods: {
    recipeUpdated(recipe) {
      console.log("So now i know recipe is updated...", recipe)
    },
    recipeDeleted(args) {
      if (this.selectedRecipe
        && this.selectedRecipe.id === args.id) {
        this.selectedRecipe = null;
      }
    },
  },
  async mounted() {
    await this.$store.dispatch('refreshRecipes');
    this.selectedRecipe = this.allRecipes.find( x => x.id == this.$route.params.id );
  }
};
```

Recipes se bitno pojednostavnio
– sad se brine samo time je li
recept odabran ili ne i kako to
prikazati

Ažuriranje nas zasad ni ne zanima, ali
brisanje da – koristimo „ljubaznost”
odnosno dojavu iz RecipeCard

```
<div v-if="selectedRecipe">
  <h2>Recipe #{{ id }}</h2><br>
  <div class="d-flex justify-content-center">
    <recipe-card :key="selectedRecipe.id"
      v-bind:id="selectedRecipe.id"
      @recipe-updated="recipeUpdated"
      @recipe-deleted="recipeDeleted"
      can-edit="true"
    ></recipe-card>
  </div>
</div>
<div v-else>
  <h2>All recipes ({{$store.getters.allRecipes.length}})</h2>
  <hr>
  <div class="container-fluid p-2 d-flex flex-wrap">
    <recipe-card v-for="recipe in allRecipes"
      :key="recipe.id"
      v-bind:id="recipe.id"
      @recipe-updated="recipeUpdated"
      @recipe-deleted="recipeDeleted"
    ></recipe-card>
  </div>
</div>
```

DM12

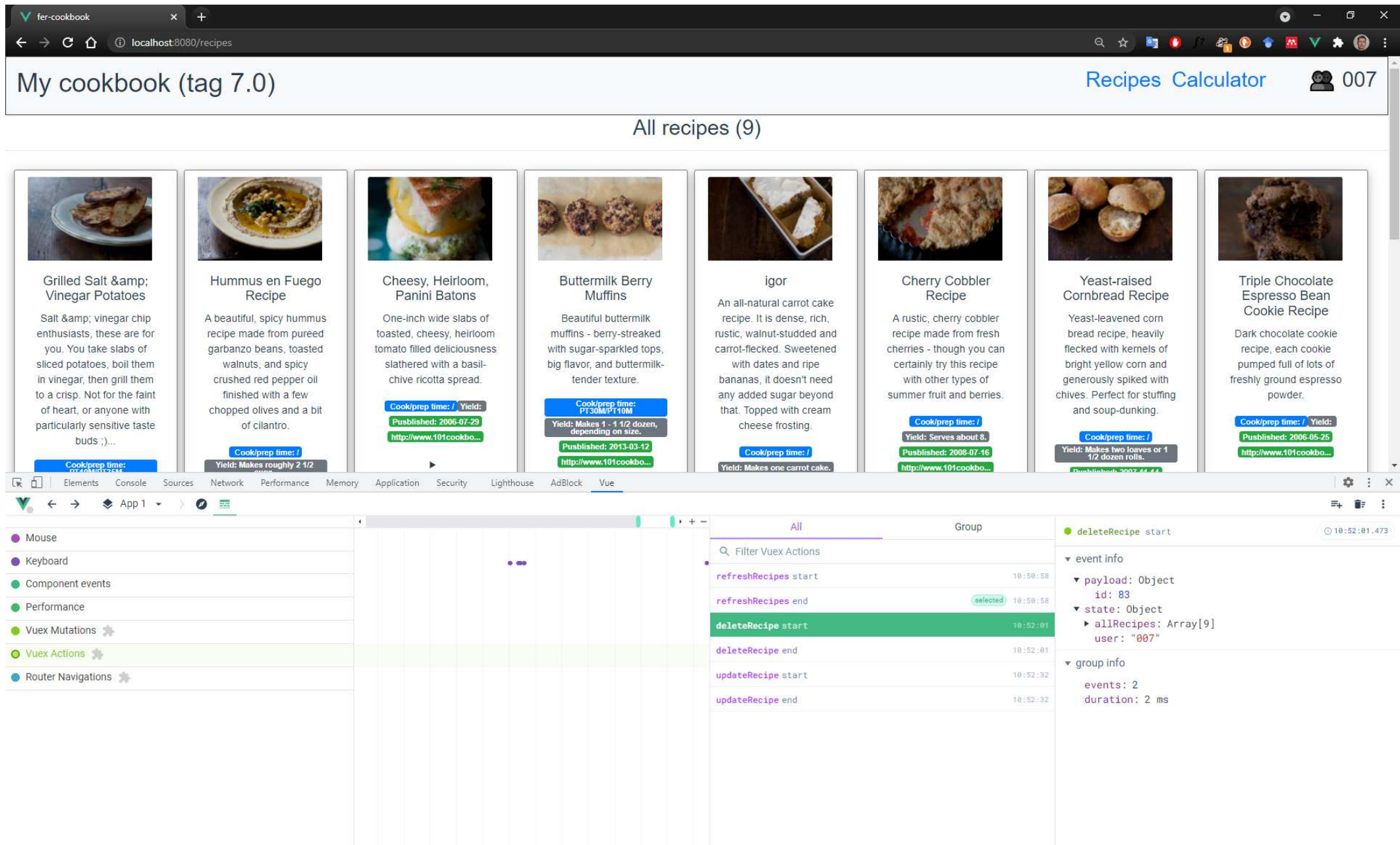
DZ: promijeniti template da
pokazuje spinner dok recepti
još nisu dohvaćeni...

DM12

primjer za `recipeDeleted` može biti counter s količinom recepata

Danijel Mlinarić; 15.9.2021.

Instalirajte *vue devtools* 6 (beta?) (6+ za Vue3)



The screenshot displays a web application titled "fer-cookbook" running on a local server at `localhost:8080/recipes`. The page features a header with "My cookbook (tag 7.0)" and a "Recipes Calculator" button. Below the header, there's a section for "All recipes (9)" showing a grid of recipe cards. Each card includes a recipe image, title, description, and metadata like cook/prepare time, yield, and publish date. The recipes listed are: Grilled Salt & Vinegar Potatoes, Hummus en Fuego Recipe, Cheesy, Heirloom, Panini Batons, Buttermilk Berry Muffins, Igor, Cherry Cobbler Recipe, Yeast-raised Cornbread Recipe, and Triple Chocolate Espresso Bean Cookie Recipe.

Below the recipe grid, the Vue DevTools 6 interface is open, showing the Vuex Actions panel. The panel lists several actions: `refreshRecipes start`, `refreshRecipes end`, `deleteRecipe start` (selected), `deleteRecipe end`, `updateRecipe start`, and `updateRecipe end`. The details for the selected `deleteRecipe start` action are shown on the right, including the payload (an object with `id: 83`) and the state (an object with `allRecipes: Array[9]` and `user: "007"`). The group info shows 2 events and a duration of 2 ms.

■ Ustroj Storea

- Za one koji žele znati više:
 - Pogledati `mapActions`, `mapGetters` – elegantniji način kako preslikati akcije i gettere iz Storea u komponente
 - `Modules` – preustroj Storea u module kada postane prevelik

I još jedna stvar...

npm run build



```
\ Building for production...
```

```
WARNING Compiled with 1 warning  
warning
```

```
4:46:18 PM
```

```
asset size limit: The following asset(s) exceed the recommended size limit (244 KiB).  
This can impact web performance.
```

```
Assets:
```

```
img/disco.b921cfd5.gif (2.06 MiB)
```

File	Size	Gzipped
dist\js\chunk-vendors.a68e7e1d.js	142.14 KiB	50.69 KiB
dist\js\app.a448ee5c.js	15.79 KiB	5.00 KiB
dist\js\chunk-2d0ddf93.8177cced.js	0.48 KiB	0.33 KiB
dist\js\chunk-91eb2748.2e907fc5.js	0.42 KiB	0.30 KiB
dist\css\app.1961192e.css	0.52 KiB	0.34 KiB

```
Images and other types of assets omitted.
```

```
DONE Build complete. The dist directory is ready to be deployed.
```

```
INFO Check out deployment instructions at https://cli.vuejs.org/guide/deployment.html
```

Razvoja i produkcijska okolina

- U razvojnoj okolini:
 - Pokreće se lokalni web-poslužitelj
 - Prevodi se .vue kontinuirano kod svakog snimanja
 - Hot-Module-Replacement (HMR)
 - Datoteke nisu minimizirane
 - Radni okvir daje raskošne provjere i upozorenja
 - Itd.
- U produkcijskoj okolini:
 - SPA je zapravo „samo” HTML + CSS + JS koji zatim trebamo poslužiti pregledniku (static web hosting)
 - Minifikacija HTML+CSS+JS datoteka (+ source maps)
 - Vendor chunk splitting (razdvojiti naš kod od knjižnica)
 - Optimizacija koda
 - Optimizacija učitavanja?

Optimizirajmo učitavanje (dijelova) stranice

- Jedan od nedostataka SPA je veliki inicijalni zahtjev
- Rješenje: učitajmo dijelove aplikacije kad (i ako trebaju)!
- Dodajmo u našu aplikaciju besmislenu „Disco” komponentu i stranicu:



- Učitajmo ju asinkrono – kad (ako) nam zatreba!

Komponenta i stranica su trivijalne:

tag: v8.0

views/Disco.vue

```
<template>
  <div>
    <h2>Disco!!!</h2>
    <div class="container-fluid p-2 d-flex justify-content-center">
      <disco-component></disco-component>
    </div>
  </div>
</template>
```

components/DiscoComponent.vue

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

Prvo učitajmo komponentu asinkrono...

tag: v8.0

main.js

```
import { createApp, defineAsyncComponent } from "vue";
import App from "./App.vue";
import store from "./store";
import router from "./router";
import RecipeCard from './components/RecipeCard.vue';
import TempConverter from './components/TempConverter.vue';
import SmallCard from './components/SmallCard.vue';

const DiscoComponent = defineAsyncComponent(() => import('./components/DiscoComponent.vue'));

const app = createApp(App);
app.use(store);
app.use(router);
app.component('recipe-card', RecipeCard);
app.component('temp-converter', TempConverter);
app.component('small-card', SmallCard);
app.component('disco-component', DiscoComponent);
app.mount("#app");
```


...potom i stranicu

tag: v8.0

router/index.js

```
import { defineAsyncComponent } from "vue";
import { createRouter, createWebHistory } from "vue-router";
import Recipes from "../views/Recipes.vue";
import Calculator from "../views/Calculator.vue";
import NotFound from "../views/NotFound.vue";
// u routeru ne koristiti defineAsyncComponent
const Disco = () => import('../views/Disco.vue');
const routes = [
  {
    path: "/",
    component: Recipes, // mogli smo i redirect: "/recipes"
  }, ...
  {
    path: "/disco",
    component: Disco,
  },
  ...
];
```

Ako pokrenemo i u razvojnoj okolini vidjet ćemo da se datoteke 0.js, 1.js. itd. dohvaćaju kada su potrebne. Kada napravimo production build, onda će to biti posebne chunk datoteke.

Konačno, napravimo build

■ npm run build

```
\ Building for production...
```

```
WARNING Compiled with 1 warning  
warning
```

```
asset size limit: The following asset(s) exceed the recommended size (244  
KiB).
```

```
This can impact web performance.
```

```
Assets:
```

```
img/disco.b921cfd5.gif (2.06 MiB)
```

File	Size	Gzipped
dist\js\chunk-vendors.a68e7e1d.js	142.14 KiB	50.69 KiB
dist\js\app.a448ee5c.js	15.79 KiB	5.00 KiB
dist\js\chunk-2d0ddf93.8177cced.js	0.48 KiB	0.33 KiB
dist\js\chunk-91eb2748.2e907fc5.js	0.42 KiB	0.30 KiB
dist\css\app.1961192e.css	0.52 KiB	0.34 KiB

```
Images and other types of assets omitted.
```

```
DONE Build complete. The dist directory is ready to be deployed.
```

```
INFO Check out deployment instructions at
```

```
https://cli.vuejs.org/guide/deployment.html
```

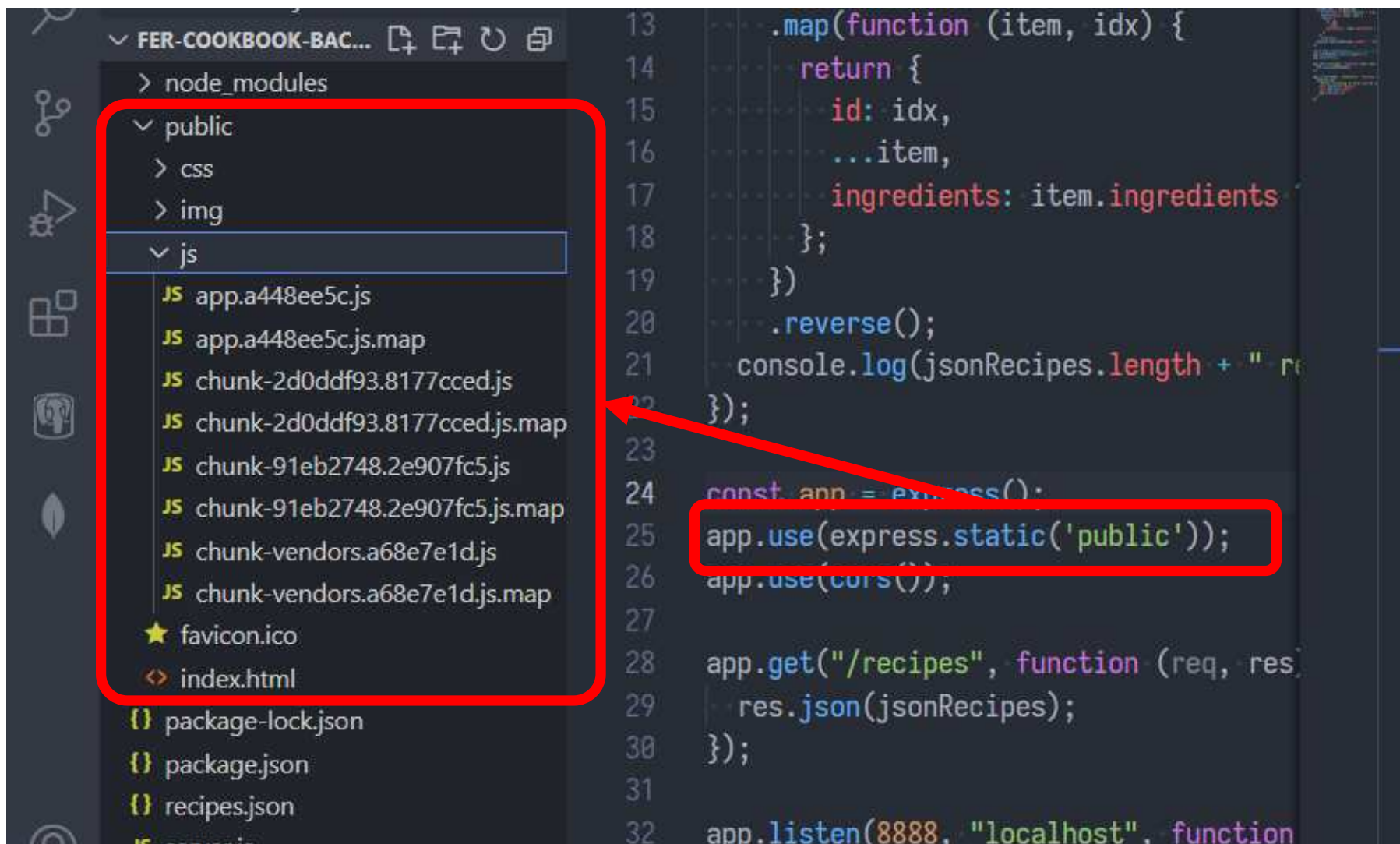
U razvojnoj okolini:

Name	Method	Status	Type	Initiator	Size	T
app.js	GET	200	script	(index)	274 kB	1
chunk-vendors.js	GET	200	script	(index)	3.0 MB	6

Ovisnosti (tuđi kod) u posebnoj datoteci, naš kod u main.js, preostali dijelovi (asinkroni) našeg koda u chunk datoteka i minificirani CSS.

Poslužimo putem naše backend aplikacije

- Kopiramo /dist direktorij koji je napravio vue-cli u /public folder backend



```
13  ... .map(function (item, idx) {
14  ...    return {
15  ...      id: idx,
16  ...      ...item,
17  ...      ingredients: item.ingredients
18  ...    };
19  ...  })
20  ...  .reverse();
21  ...  console.log(jsonRecipes.length + " recipes");
22  ...  });
23
24  const app = express();
25  app.use(express.static('public'));
26  app.use(cors());
27
28  app.get("/recipes", function (req, res) {
29    res.json(jsonRecipes);
30  });
31
32  app.listen(8888, "localhost", function
```

Projekt

- Napraviti SPA aplikaciju u Vue3 radnom okviru
- *Peer assessment* putem Edgara