

Programsko inženjerstvo

Nositelj: doc. dr. sc. Nikola Tanković

Asistent: mag. inf. Alesandro Žužić

Ustanova: Sveučilište Jurja Dobrile u Puli, Fakultet informatike u Puli



Fakultet informatike u Puli

[4] Komponente

Vue komponente omogućuju razlaganje korisničkog sučelja na manje, modularne i ponovo iskoristive dijelove, čime se poboljšava organizacija i održavanje koda. Njihova upotreba omogućuje logičnu i skalabilnu izgradnju aplikacije, olakšavajući razvoj, testiranje i ponovnu upotrebu elemenata sučelja.

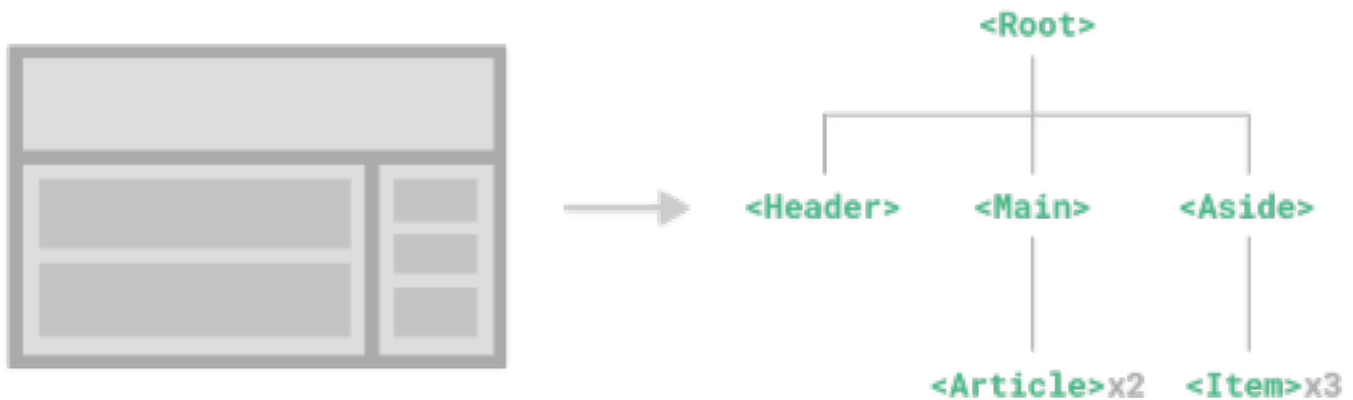


Posljednje ažurirano: 1. travnja 2025.

- Programsko inženjerstvo
- [4] Komponente
 - Vue komponente
 - Definiranje komponenti
 - Korištenje komponenti
 - Prosljeđivanje parametara props
 - defineProps
 - Validacija parametara
 - Statični i dinamični parametri
 - Jednosmjerna veza
 - Događaji komponenti
 - Prosljeđivanje događaja emit
 - Prosljeđivanje argumenata događajima
 - Deklaracija prosljeđenih događaja
 - Slot element
 - Dohvaćanje konteksta unutar slota
 - Zadani sadržaj slota
 - Imenovani slotovi
 - Samostalni zadatak za vježbu 4

Vue komponente

Kao što smo rekli i prije, Vue koristi **komponente** – modularne, višekratno iskoristive dijelove korisničkog sučelja koji sadrže vlastitu logiku, stilove i podatke. Komponente olakšavaju održavanje kôda i omogućuju lakšu skalabilnost aplikacija.



Njihova upotreba donosi brojne prednosti:

- **Modularnost** – Sučelje se dijeli na manje cjeline koje su lakše za upravljanje i testiranje
- **Ponovna upotreba** – Jednom definirane komponente mogu se koristiti na više mjesta u aplikaciji
- **Izolacija** – Svaka komponenta ima vlastite podatke, metode i stilove, čime se izbjegava nepredvidivo ponašanje u aplikaciji
- **Jednostavna komunikacija** – Komponente mogu razmjenjivati podatke putem **parametri** (*props*) i **dogadaja** (*events*)
- **Poboljšana čitljivost i održavanje** – Kod je bolje organiziran i ima definiranu strukturu

Definiranje komponenti

Kada izrađujemo novi projekt, uvijek ćemo imati barem jednu komponentu, a to je `App.vue`. Ono što trebamo uočiti i znati jest da sve komponente u Vue-u imaju nastavak `.vue`, što također znamo i pod nazivom **Single-File Component** (*SFC*).

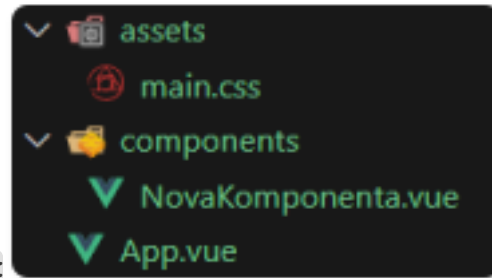
Ponavljanje iz prve skripte:

Komponente se obično pišu u formatu koji nalikuje HTML-u, poznatom kao **Single-File Component** (*SFC*), odnosno `*.vue` datoteke. Kao što ime sugerira, SFC enkapsulira logiku komponente (JavaScript), predložak (HTML) i stilove (CSS) u jednoj datoteci.

Komponente možemo izraditi bilo gdje u projektu, no u pravilu ih organiziramo prema njihovoj namjeni. Komponente koje se više puta koriste u različitim dijelovima aplikacije, ali nisu zasebne

stranice, smještamo u mapu `components`.

- Ako ih koristimo kao stranice unutar aplikacije koja koristi `router`, tada ih stavljamo u mapu `views`.



Smještamo komponentu unutar mape `components`:

Kada želimo izraditi novu komponentu, dovoljno je kreirati datoteku s proizvoljnim, ali valjanim nazivom i nastavkom `.vue`.

Svaka komponenta mora sadržavati `<template>` blok:

```
<script setup>
  // opcionalno
  // logika komponente
  // composition api
</script>

<template>
  <!--obavezno-->
  <!--tijelo komponente-->
  Ahoy!
</template>

<style scoped>
  // opcionalno
  // stil komponente
  // scoped -> označava da se stil primjenjuje samo nad komponentom
</style>
```

Korištenje komponenti

Da bi koristili komponente u našem projektu, moramo je učitati.

Možemo kreirati komponentu pod nazivom `Dekoracija.vue`:

```
<template>
  <div class="flex gap-1 items-center">
    <div class="bg-slate-950 opacity-5 h-2 w-2 rounded-full"></div>
    <div class="bg-slate-950 opacity-10 h-2 w-4 rounded-full"></div>
    <div class="bg-slate-950 opacity-20 h-2 w-8 rounded-full"></div>
    <div class="bg-slate-950 opacity-40 h-2 w-16 rounded-full"></div>
```

```

        <div class="bg-slate-950 opacity-60 h-2 w-32 rounded-full"></div>
        <div class="bg-slate-950 opacity-80 h-6 w-2 rounded-full"></div>
    </div>
</template>

```

Te je unutar `App.vue` možemo učitati:

```

<script setup>
  import Dekoracija from "@/components/Dekoracija.vue";
</script>

<template>
  <div
    class="h-full flex justify-center items-center gap-4 p-16 bg-slate-200">
    <Dekoracija></Dekoracija>

    <p class="text-3xl font-bold text-slate-950">FANCY TITLE</p>

    <Dekoracija class="rotate-180"></Dekoracija>
  </div>
</template>

```



- Možemo primijetiti da smo komponentu učitali pod nazivom `Dekoracija`, iako smo je mogli nazvati proizvoljno. Ipak, uobičajena praksa je da naziv komponente odgovara nazivu datoteke.

Kod učitavanja i korištenja komponenti preporučuje se korištenje **PascalCase** naziva oznaka za komponente kako bi se razlikovale od izvornih HTML elemenata. Iako izvornim HTML oznakama nisu bitna velika i mala slova, kod Vue komponenti se mogu koristiti.

```

<script setup>
  import MojaDekoracija from "@/components/Dekoracija.vue";
  import mojaDekoracija from "@/components/DekoracijaKruzici.vue";
</script>

<template>
  <div
    class="flex justify-center items-center
    gap-4 p-16 bg-slate-200">
    <MojaDekoracija></MojaDekoracija>

```

```

    <p class="text-3xl font-bold text-slate-950">FANCY TITLE</p>
    <!--Preporuka PascalCase-->
    <MojaDekoracija class="rotate-180" />
  </div>

  <div
    class="flex justify-center items-center
    gap-4 p-16 bg-slate-200">
    <!--Izbjegavat-->
    <mojaDekoracija />

    <p class="text-3xl font-bold text-slate-950">FANCY TITLE</p>

    <mojaDekoracija class="rotate-180"></mojaDekoracija>
  </div>
</template>

```



U ovom slučaju, riječ je o dvije različite komponente koje smo učitali pod istim nazivom `moja dekoracija`, pri čemu jedan naziv počinje velikim početnim slovom, a drugi malim. Time ih Vue smatra kao dvije različite komponente. Zato se preporučuje koristiti PascalCase imenovanje i odabrati deskriptivnije nazive koji se jasno razlikuju. U ovom slučaju, bolje bi bilo koristiti nazive poput *DekoracijaLinije* i *DekoracijaKruzici*.

Možemo koristiti `</>` za zatvaranje oznake

Kako bismo nadalje izbjegli stvaranje besmislenih primjera, koristit ćemo koncept digitalne knjižnice. Možemo je zamisliti kao vlastiti ormar s policama, gdje svaka polica predstavlja određenu kolekciju knjiga, a unutar kolekcija nalaze se serijali i pojedinačne knjige.

- ormar → polica → kolekcija → knjiga

To znači da ćemo našu komponentnu hijerarhiju graditi od najmanjih jedinica prema složenijim strukturama. Počet ćemo s komponentom koja predstavlja **knjigu**, zatim ćemo je smjestiti unutar **kolekcije** knjiga, koje će se nalaziti na **policama** unutar **ormara**.

- Tako da krećemo s izradom komponentom `Knjiga.vue` koja će imati dva parametra: *slika* i *naslov*

`Knjiga.vue` komponenta:

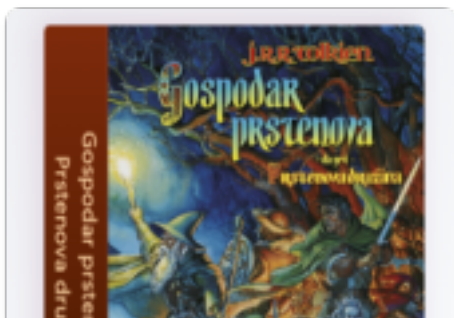
```

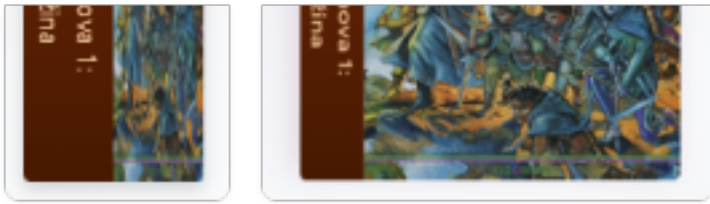
<script setup>
  import { ref } from "vue";
  const slika = ref("./src/assets/images/books/fantasy/lotr_1.png");
  const naslov = ref("Gospodar prstenova 1: Prstenova družina");
  const selected = ref(false);
</script>

<template>
  <div
    class="group/book flex flex-col gap-1 h-fit"
    :class="{ 'active' : selected }">
    <div
      @click="selected = true"
      @mouseleave="selected = false"
      class="relative flex h-64 max-w-12 items-start group-hover/book:rotate-x-10
transition-all duration-300"
      :class="selected ? 'drop-shadow-[-4px_-4px_8px_rgba(224,212,224,1)] -
translate-x-54 scale-125' :
      'drop-shadow-lg group-hover/book:-translate-x-4 group-hover/
book:scale-105'">
      <div
        class="flex items-center justify-center text-orange-200 bg-gradient-to-
b from-orange-900 to-amber-950 rounded-l
        h-64 min-w-12 px-8 text-sm text-center text-pretty transition-all
duration-300 [writing-mode:sideways-rl] origin-right"
        :class="{ '-rotate-y-45' : selected }">
        {{ naslov }}
      </div>

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

```





I sada ako želimo koristiti tu komponentu knjige, učitamo je u App.vue:

```
<script setup>
  import Knjiga from "@/components/Knjiga.vue";
</script>

<template>
  <div
    class="h-full flex flex-col items-center p-8 bg-gradient-to-br
    from-gray-200 via-slate-50 to-stone-300 text-gray-800">
    <div class="text-3xl font-bold">Digitalna Knjižnica</div>
    <div class="flex gap-1 p-8">
      <Knjiga v-for="i in 3" />
    </div>
  </div>
</template>
```



Međutim, sada imamo tri identične knjige, što je i očekivano jer sve imaju isti naziv i sliku definiranu u komponenti. Ako želimo prikazati tri različite knjige, umjesto da stvaramo još dvije identične komponente s različitim nazivima i slikama, možemo im proslijediti podatke kao parametre iz neke liste.

Prosljeđivanje parametara (*props*)

U većini slučajeva, za komponente koje koristimo unutar projekta, želimo da imaju različit sadržaj. Jedan od načina kako to možemo postići je kroz prosljeđivanje podataka, a u tom kontekstu dolaze u igru **parametri** (*props*).

- **Parametri** su Vue atributi koji se mogu registrirati unutar komponente. Ako želimo proslijediti podatke komponenti, trebamo deklarirati popis parametara koje komponenta može prihvatiti, koristeći funkciju `defineProps`.

Funkciji `defineProps` prosljeđujemo listu parametara:

```
const props = defineProps(["naslov"]);
console.log(props.naslov);
```

Tako da u komponenti `Knjiga.vue` možemo promijeniti `script`:

```
<script setup>
  defineProps(["slika", "naslov"]);
</script>
```

Te u `App.vue` onda proslijediti podatke parametrima knjige iz liste:

```
<script setup>
import { ref } from "vue";
import Knjiga from "@/components/Knjiga.vue";
const knjige = ref([
  {
    naslov: "Gospodar prstenova - Prstenova družina",
    slika: "./src/assets/images/books/fantasy/lotr_1.png",
  },
  {
    naslov: "Gospodar prstenova - Dvije kule",
    slika: "./src/assets/images/books/fantasy/lotr_2.png",
  },
  {
    naslov: "Gospodar prstenova - Povratak kralja",
    slika: "./src/assets/images/books/fantasy/lotr_3.png",
  },
]);
</script>

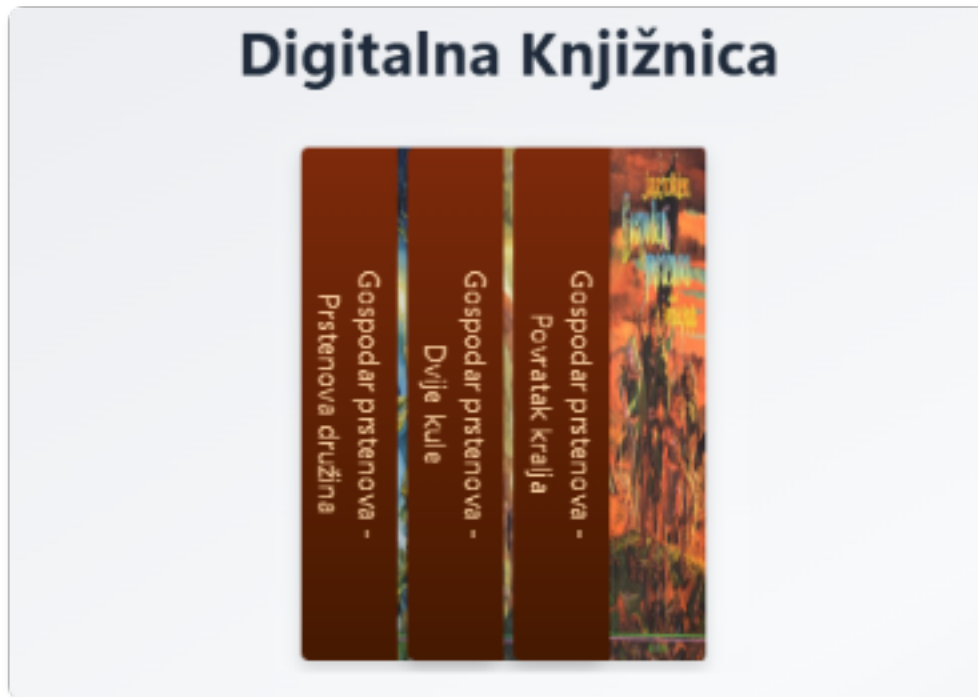
<template>
  <div
    class="h-full flex flex-col items-center p-8 bg-gradient-to-br
```



```

from-gray-200 via-slate-50 to-stone-300 text-gray-800">
<div class="text-3xl font-bold">Digitalna Knjižnica</div>
<div class="flex gap-1 p-8">
  <Knjiga
    v-for="knjiga in knjige"
    :naslov="knjiga.naslov"
    :slika="knjiga.slika" />
</div>
</div>
</template>

```



Sada vidimo da smo uspješno proslijedili podatke komponentama, povezujući parametre kao attribute. Time svaka knjiga ima svoj jedinstveni naziv.

defineProps()

Kao što smo rekli, s pomoću funkcije `defineProps()` deklariramo parametre koje možemo proslijediti komponenti.

- Ako parametre ne spremimo u varijablu, onda im možemo pristupiti samo unutar `<template>` bloka:

```

<script setup>
  defineProps(["mojParametar"]);

  console.log(props.mojParametar); // neće raditi
</script>

```

- Ako želimo pristupati parametrima unutar `<script>` bloka onda ih moramo spremiti u varijablu (u praksi tu varijablu nazivamo *props*):

```
<script setup>
  const props = defineProps(["mojParametar"]);

  console.log(props.foo); // ispisat će vrijednost proslijeđene vrijednosti parametra
</script>
```

- Kada imenujemo parametre s dugačkim nazivima, u pravilu koristimo **camelCase**

```
defineProps({
  nazivKnjige: String,
});
```

- Te kada prosljeđujemo vrijednosti parametrima komponenti, onda u pravilu koristimo **kebab-case**

```
<Knjiga naziv-knjige="Gospodar prstenova - Prstenova družina" />
```

Validacija parametara

Funkcija `defineProps()` može primiti ne samo *polje* parametara, već i *objekt* parametara. U tom slučaju omogućuje nam provjeru i validaciju proslijeđenih vrijednosti, gdje je:

- **ključ** – naziv parametra
- **vrijednost** – tip podataka ili objekt s dodatnom validacijom

```
defineProps({
  // Osnovna provjera tipa
  // (`null` i `undefined` vrijednosti dopuštaju bilo koji tip)
  paramA: Number,

  // Više mogućih tipova
  paramB: [String, Number],

  // Obavezan string
  paramC: {
    type: String,
    required: true,
  },

  // Obavezan ali dopušta null vrijednost
  paramD: {
```

```

    type: [String, null],
    required: true,
  },

  // Broj s zadanom vrijednošću
  paramE: {
    type: Number,
    default: 100,
  },

  // Objekt s zadanom vrijednošću
  paramF: {
    type: Object,
    // Objekt ili polje moraju imati zadanu vrijednost
    // koja se vraća iz funkcije. Funkcija prima sirove
    // proslijeđene parametre kao argument.
    default(rawProps) {
      return { message: "pozdrav" };
    },
  },

  // Prilagođena funkcija za validaciju
  paramG: {
    validator(value, props) {
      // Vrijednost mora biti jedna od navedenih opcija
      return ["uspjeh", "upozorenje", "opasnost"].includes(value);
    },
  },
});

```

- Svi parametri (*props*) su po zadanom **opcionalni**, osim ako nije navedeno `required: true`
- Ako opcionalni parametar (osim Booleana) nije prisutan, njegova vrijednost će biti `undefined`
- Boolean parametri koji nisu prisutni automatski će se postaviti na `false`. Možete promijeniti ovo ponašanje postavljanjem zadane vrijednosti, npr. `default: undefined`, kako bi se ponašao kao ne-Boolean parametar
- Ako je zadana vrijednost definirana, koristit će se ako je vrijednost parametra `undefined`. To uključuje situacije kada parametar nije proslijeđen ili mu je eksplicitno dodijeljena vrijednost `undefined`

Kada validacija parametra ne uspije, Vue će prikazati upozorenje u konzoli

Statični i dinamični parametri

Kao i za atribut, vue dopošta parametre koji su proslijeđeni kao statične i dinamičke vrijednosti:

```

<!-- Statičko dodjeljivanje vrijednosti varijable -->
<Knjiga naslov="Gospodar prstenova - Prstenova družina" />

```

```
<!-- Dinamično dodjeljivanje vrijednosti varijable -->
<Knjiga v-bind:naslov="naslov" />

<!-- Dinamično dodjeljivanje vrijednosti izraza -->
<Knjiga :naslov="volumen + '. ' + naslov" />
```

- Vue omogućuje prosljeđivanje parametara bilo kojeg tipa

```
<!-- Iako je `42` statična vrijednost, koristimo v-bind jer je to JS izraz -->
<Knjiga :broj-stranica="423" />

<!-- Ako je prop naveden bez vrijednosti, pretpostavlja se da je `true` -->
<Knjiga objavljena />

<!-- Iako je `false` statična vrijednost, koristimo v-bind -->
<Knjiga :objavljena="false" />

<!-- Prosljeđivanje polja -->
<Knjiga :zanrovi="['avantura', 'fantazija']" />

<!-- Prosljeđivanje objekta -->
<Knjiga
  :autor="{
    ime: 'John Ronald Reuel',
    prezime: 'Tolkien'
  }" />
```

- Ako želimo proslijediti sva svojstva objekta kao parametre, možemo koristiti `v-bind` bez argumenta

```
const lotrPrvaKnjiga_info = {
  naziv: "Gospodar prstenova - Prstenova družina",
  brojStranica: 423,
  zanrovi: ["avantura", "fantazija"],
  autor: {
    ime: "John Ronald Reuel",
    prezime: "Tolkien",
  },
  objavljena: true,
};
```

```
<Knjiga v-bind="lotrPrvaKnjiga_info" />
```

Jednosmjerna veza

Svi *parametri* imaju **jednosmjernu vezu** od roditelja prema djetetu: kada se vrijednost u roditelju promijeni, ažurira se i u djetetu, ali ne obrnuto.

Svako ažuriranje roditelja ponovno prosljeđuje *parametar* djetetu. Zato **nije preporučljivo** mijenjati *parametre* unutar djeteta – Vue će prikazati upozorenje:

```
const props = defineProps(["naziv"]);

props.naziv = "Novi naziv"; // Neispravno – READONLY!
```

- Jedno rješenje je da koristimo *parametar* kao početnu vrijednost za izradu lokalne kopije:

```
const props = defineProps(["naziv"]);
const localniNaziv = ref(props.naziv); // Lokalna kopija
```

Komponenta ne može izravno mijenjati *parametre*, ali može mijenjati **unutarnja svojstva objekata/nizova** jer se prenose po referenci

```
const props = defineProps(["autor"]);
props.autor.ime = "Novo ime"; // Radi – Nije preporučljivo
```

U pravilu komponenta treba emitirati događaj, a roditelj obaviti mutaciju, no ponekad je poželjno i jednostavnije raditi mutaciju unutar komponente

Događaji komponenti

U prethodnom poglavlju stvorili smo komponentu `Knjiga.vue`, kojoj prosljeđujemo parametre iz liste. Sada možemo kreirati novu komponentu `Kolekcija.vue`, koja će sadržavati više knjiga iz istog serijala, poput sedmologije *Harry Potter* ili trilogije *Gospodar prstenova*.

- Osim prikaza knjiga, omogućit ćemo **dodavanje i brisanje** knjiga iz kolekcije.
 - Svaka knjiga će imati dugme *Ukloni*.
- Kolekcija će sadržavati **parametar** `kolekcija`, koji ćemo prosljeđivati iz roditeljske komponente.
 - Svaka kolekcija sadrži *naslov* i *popis knjiga*.

Prvi korak je implementacija funkcionalnosti za uklanjanje određene knjige iz kolekcije pritiskom na dugme **Ukloni**:

```
<div
  class="text-red-700 font-bold opacity-0 group-hover/book:opacity-100 cursor-pointer
  hover:underline transition"
  @click="$emit('ukloni')">
```

```
    <!--prosljeđivanje događaja ukloni-->
    Ukloni
  </div>
```

Budući da pojedinačne knjige nemaju izravan pristup kolekciji, ne mogu ukloniti same sebe iz nje. Umjesto toga, potrebno je pozvati funkciju unutar komponente `Kolekcija.vue`, koja će se pobrinuti za uklanjanje knjige iz popisa.

- Način na koji to radimo je korištenjem `$emit` događaja, koji omogućuje komponenti `Knjiga.vue` da emitira vlastiti događaj prema `Kolekcija.vue`. Kada se događaj pokrene, `Kolekcija.vue` ga može presresti i ažurirati popis knjiga uklanjanjem odgovarajuće stavke.

```
<script setup>
  import { ref } from "vue";
  import Knjiga from "@/components/Knjiga.vue";
  const props = defineProps(["kolekcija"]);

  function ukloniKnjigu(index) {
    props.kolekcija.knjige.splice(index, 1);
  }
</script>

<template>
  <div class="relative group flex flex-col gap-2">
    <div class="text-lg font-bold">{{ kolekcija.naslov }}</div>
    <div class="flex gap-1">
      <Knjiga
        v-for="(knjiga, i) in kolekcija.knjige"
        :naslov="knjiga.naslov"
        :slika="knjiga.slika"
        @ukloni="ukloniKnjigu(i)" />
      <!--presrećivanje događaja ukloni-->
    </div>
  </div>
</template>
```

Dalje, kod komponente `Kolekcija.vue` možemo dodati mogućnost za dodavanje nove knjige. Definirat ćemo novu varijablu `novaKnjiga` i funkciju koja će omogućiti dodavanje knjige u kolekciju.

```

const novaKnjiga = ref({
  naslov: "",
  slika: "",
});

function dodajKnjigu() {
  props.kolekcija.knjige.push(novaKnjiga.value);
}

```

```

<div class="flex gap-2 opacity-0 group-hover:opacity-100 text-sm transition">
  <input
    type="text"
    class="ring rounded w-24 px-2 bg-amber-100/25"
    v-model="novaKnjiga.naslov"
    placeholder="Naslov..." />
  <input
    type="text"
    class="ring rounded w-24 px-2 bg-amber-100/25"
    v-model="novaKnjiga.slika"
    placeholder="Url slike..." />
  <button
    class="bg-sky-600 rounded px-2 hover:bg-sky-500 text-sky-50 cursor-pointer"
    @click="dodajKnjigu()">
    Dodaj
  </button>
</div>

```

Zatim u `App.vue` komponenti malo preuredimo i dodamo dvije kolekcije:

```

const lotrKolekcija = ref({
  naslov: "Gospodar prstenova",
  knjige: [...]
})
const harryPotterKolekcija = ref({
  naslov: "Harry Potter",
  knjige: [...]
})

```

```

<template>
  <div
    class="h-full flex flex-col items-center bg-gradient-to-br from-gray-200 via-slate-50 to-stone-300 text-gray-800 px-60 pt-8 gap-8">
    <div class="text-3xl font-bold">Digitalna Knjižnica</div>
    <div

```

```

class="flex flex-col w-full pt-4 rounded-xl"
style="background-image: url(https://i.primimg.com/originals/b9/ea/4e/
b9ea4e526fb0fc0fd8038ac713fbb881.jpg)">
<div class="flex gap-8 px-8 pt-4">
  <Kolekcija :kolekcija="lotrKolekcija" />
  <Kolekcija :kolekcija="harryPotterKolekcija" />
</div>
<div
  class="flex items-center bg-orange-950 rounded-lg text-orange-100 text-
xl px-8 py-1">
  Fantazija
</div>
</div>
</div>
</template>

```



Prosljeđivanje događaja `emit()`

Komponente u Vue.js mogu prosljeđivati vlastite događaje koristeći metodu `$emit()`. Ovaj mehanizam omogućava komunikaciju između komponenta roditelja i djece, na način da djete šalje signal roditelju kada se nešto dogodi.

U komponenti možemo prosljeđivati događaj izravno iz `<template>`:

```

<!-- MyComponent.vue -->
<template>
  <button @click="$emit('myEvent')">Klikni me</button>
</template>

```


- U ovom slučaju, kada korisnik klikne dugme, komponenta proslijedi događaj pod nazivom `myEvent`

Roditelj može oslušivati ovaj događaj pomoću `v-on` direktive (kraće `@`):

```
<template>
  <MyComponent @my-event="myFun" />
</template>

<script setup>
  function myFun() {
    console.log("Događaj myEvent je primljen!");
  }
</script>
```

- Ako želimo da se događaj preslušava samo jednom, možemo koristiti `.once` modifikator:

```
<MyComponent @my-event.once="myFun" />
```

Nakon prvog emitiranja, ovaj događaj se više neće preslušavati.

Za razliku od nativnih DOM događaja, događaji koje proslijeđuje komponenta **ne propagiraju se prema gore** kroz hijerarhiju komponenti. Roditelj može čuti samo događaje proslijeđene izravno od svoje djece. Ako je potrebno komunicirati između neizravno povezanih komponenti, preporučuje se korištenje globalnog stanja (*Vuex*, *Pinia*).

Prosljeđivanje argumenata događajima

Često želimo poslati dodatne podatke uz događaj. To se može postići dodavanjem argumenata u `$emit()`.

Pretpostavimo da imamo komponentu koja omogućava povećanje broja za određenu vrijednost:

```
<template>
  <button @click="$emit('increaseBy', 1)">Povećaj za 1</button>
</template>
```

- Ovdje `$emit('increaseBy', 1)` znači da će se uz događaj `'increaseBy'` poslati broj `1`

Roditeljska komponenta može uhvatiti taj argument i koristiti ga:

```
<template>
  <MyButton @increase-by="(n) => count += n" />
</template>
```

```
<script setup>
  import { ref } from "vue";

  const count = ref(0);
</script>
```

- Ako želimo koristiti metodu umjesto inline funkcije, možemo definirati posebnu funkciju:

```
<template>
  <MyButton @increase-by="increaseCount" />
</template>

<script setup>
  import { ref } from "vue";

  const count = ref(0);

  function increaseCount(n) {
    count.value += n;
  }
</script>
```

- Ako je potrebno poslati više podataka, `$emit()` može primiti više argumenata:

```
<button @click="$emit('update', 'Nova vrijednost', 42)">Ažuriraj</button>
```

Roditelj može primiti sve argumente:

```
<MyComponent @update="(msg, num) => console.log(msg, num)" />
```

Deklaracija proslijeđenih događaja

Možemo **deklarirati događaje** koje komponenta emitira. Budući da metoda `$emit`, korištena unutar `<template>`, nije dostupna unutar `<script setup>`, `defineEmits()` vraća ekvivalentnu funkciju koju možemo koristiti umjesto nje:

```
<script setup>
  const emit = defineEmits(["inFocus", "submit"]);

  function buttonClick() {
    emit("submit");
  }
```

```
</script>
```

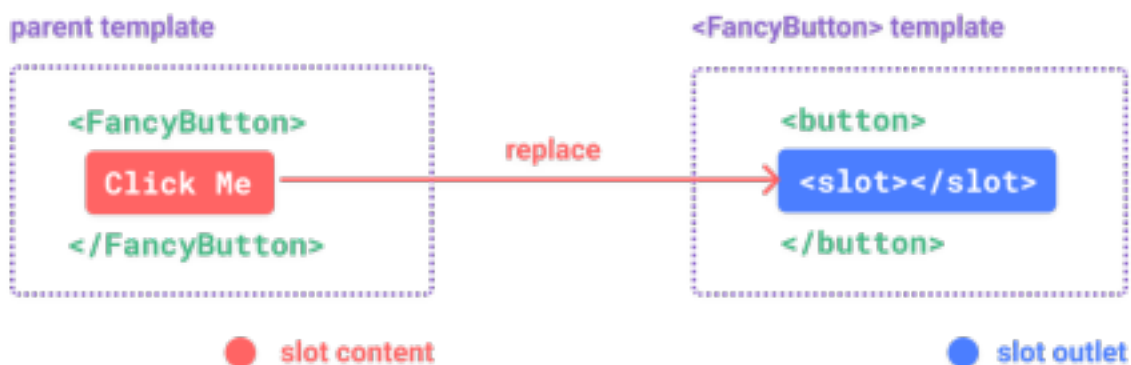
Slot element

Naučili smo da komponente mogu primiti **parametre** (*props*), koji mogu biti JavaScript vrijednosti bilo kojeg tipa. No, što je s predloškom (*template*)? U nekim slučajevima želimo proslijediti dio predloška **komponenti**, kako bi ga ona mogla prikazati unutar vlastitog predloška.

```
<FancyButton>
  Click me! <!-- umetnuti sadžaj-->
</FancyButton>
```

```
<button class="fancy-btn">
  <slot/> <!-- mjesto za umetanje sadržaja -->
</button>
```

Element `<slot>` predstavlja **mjesto za umetanje sadržaja**, označavajući gdje će se prikazati sadržaj koji je roditelj proslijedio djetetu.



- Sada možemo nastaviti s izradom komponente `Polica.vue`, koja će sadržavati više kolekcija unutar istog žanra. Pomoću `<slot/>` elementa dinamički ćemo ubaciti žanr police.

```
<script setup>
  import Kolekcija from "@/components/Kolekcija.vue";
  defineProps(['kolekcije'])
</script>

<template>
  <div class="flex flex-col w-full pt-4 rounded-xl">
```

```

        style="background-image: url(https://i.pinimg.com/originals/b9/ea/4e/
b9ea4e526fb0fc0fd8038ac713fbb881.jpg)">
        <div class="flex gap-8 px-8 pt-4">
            <Kolekcija v-for="kolekcija in kolekcije"
                :kolekcija="kolekcija"/>
        </div>
        <div class="flex items-center bg-orange-950 rounded-lg text-orange-100 text-xl
px-8 py-1">
            <slot/> <!--mjesto za umetanje žanra police-->
        </div>
    </div>
</template>

```

- Zatim ćemo u skladu s tim ažurirati i `App.vue`, tako da pravilno prikaže policu s kolekcijama unutar aplikacije.

```

<script setup>
    import { ref } from 'vue';
    import Polica from "@components/Polica.vue";
    const fantazijaKolekcije = ref([
        {
            naslov: "Gospodar prstenova",
            knjige: [...]
        },
        {
            naslov: "Harry Potter",
            knjige: [...]
        }
    ])
    const scifiKolekcije = ref([
        {
            naslov: "Dina",
            knjige: [...]
        },
    ])
</script>

<template>
    <div class="h-full flex flex-col items-center bg-gradient-to-br from-gray-200 via-
slate-50 to-stone-300 text-gray-800 px-60 pt-8 gap-8">
        <div class="text-3xl font-bold">
            Digitalna Knjižnica
        </div>
        <Polica :kolekcije="fantazijaKolekcije">
            Fantazija
        </Polica>
        <Polica :kolekcije="scifiKolekcije">
            Sci-Fi
        </Polica>
    </div>

```

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

Dohvaćanje konteksta unutar slota

Sadržaj unutar **slot**a ima pristup podacima roditelja, jer je definiran unutar nje.

```
<span>{{ message }}</span>
<FancyButton>{{ message }}</FancyButton>
```

Obje interpolacije `{{ message }}` prikazat će isti sadržaj. Međutim, **slot sadržaj nema pristup podacima djeteta**. Izrazi u Vue predlošcima mogu pristupiti samo kontekstu u kojem su definirani, slično kao u JavaScriptu.

Roditelj vidi samo vlastite podatke, dok dijete vidi samo svoje podatke

Zadani sadržaj slot

Ponekad želimo postaviti **zadani sadržaj slot**a, koji će se prikazati samo ako roditelj ne osigura sadržaj.

```
<button type="submit">
  <slot>Submit</slot> <!-- zadani sadržaj -->
</button>
```

- Ako koristimo ovu komponentu bez slot sadržaja:

```
<SubmitButton />
```

- Tada će se prikazati zadani sadržaj:

```
<button type="submit">Submit</button>
```

- Ali ako osiguramo vlastiti sadržaj:

```
<SubmitButton>Spremi</SubmitButton>
```

- Prikazat će se:

```
<button type="submit">Spremi</button>
```

Imenovani slotovi

Kada komponenta ima više mjesta za umetanje sadržaja, koristimo **imenovane slotove**. Na primjer, komponenta `BaseLayout.vue` može sadržavati različite sekcije:

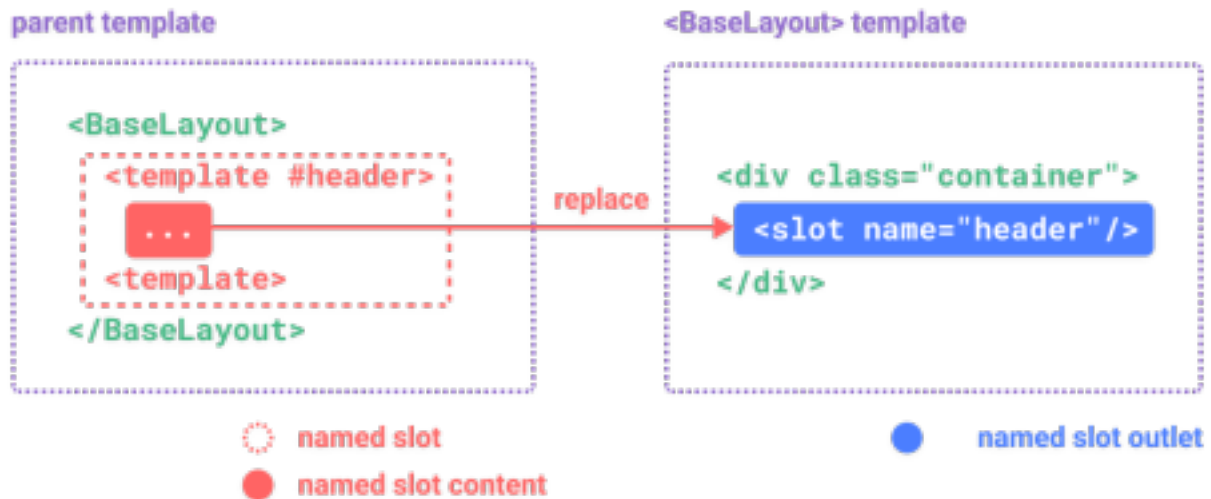
```
<div class="container">
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot> <!-- zadani slot -->
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
```

- U roditelju koristimo `v-slot` direktivu da proslijedimo sadržaj u određeni slot (skraćeno `#`):

```
<BaseLayout>
  <template v-slot:header>
    <h1>Naslov stranice</h1>
  </template>

  <template #default>
    <p>Glavni sadržaj.</p>
  </template>

  <template #footer>
    <p>Kontakt informacije.</p>
  </template>
</BaseLayout>
```



Slot bez imena (<slot>) automatski postaje **zadani slot**, dok se ostali definiraju s name.

- Na kraju, preostaje nam još izraditi završnu komponentu `Ormar.vue`, koja će sadržavati više polica s različitim žanrovima. Time ćemo završiti izradu naše aplikacije.

`Ormar.vue` komponenta:

```

<script setup>
  import Polica from "@/components/Polica.vue";
  defineProps(['police'])
</script>
<template>
  <div class="flex flex-col gap-2">
    <p class="text-3xl">
      <slot />
    </p>
    <div class="h-full flex flex-col">
      <Polica v-for="polica in police"
        :kolekcije="polica.kolekcije">
        {{ polica.zanr }}
      </Polica>
    </div>
  </div>
</template>

```

`App.vue` komponenta:

```

<script setup>
  import { ref } from 'vue';

```

```

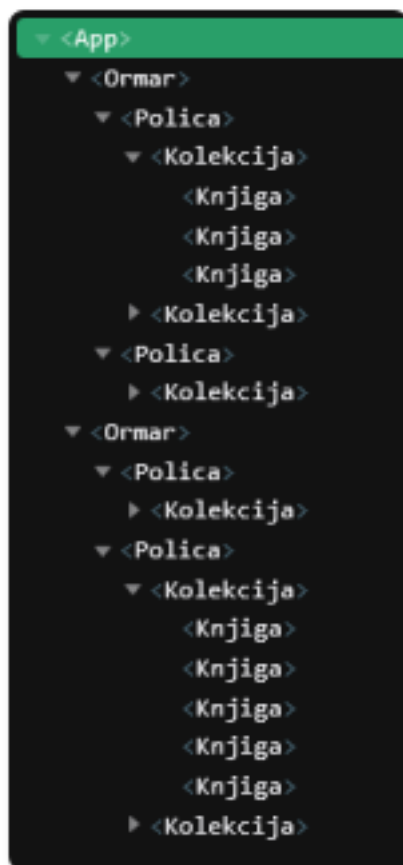
import Ormar from './components/Ormar.vue';
const policaKnjiga = ref([
  {
    žanr: "Fantasy",
    kolekcije: [...]
  },
  {
    žanr: "Sci-Fi",
    kolekcije: [...]
  }
])
const policaStripova = ref([
  {
    žanr: "Dark Fantasy",
    kolekcije: [...]
  },
  {
    žanr: "Superhero",
    kolekcije: [...]
  }
])
</script>

<template>
  <div class="overflow-auto h-full flex flex-col items-center bg-gradient-to-br from-gray-200 via-slate-50 to-stone-300 text-gray-800 py-8 gap-4">
    <div class="text-3xl font-bold">
      Digitalna Knjižnica
    </div>
    <div class="w-full px-48 flex gap-16">
      <Ormar :police="policaKnjiga" class="w-1/2">
        Knjige
      </Ormar>
      <Ormar :police="policaStripova" class="w-1/2">
        Stripovi
      </Ormar>
    </div>
  </div>
</template>

```




Ako sada pogledamo Vue **DevTools**, vidjet ćemo da je struktura projekta organizirana točno onako kako smo na početku postavili hijerarhiju.



Samostalni zadatak za vježbu 4

U ovom zadatku ćete izraditi Vue komponente na temelju postojeće `App.vue` datoteke iz projekta u mapi `Primjeri/primjer_vue_zadatak`.

Trenutno `App.vue` sadrži veliki broj ponavljajućih elemenata, stoga ćemo ga modularizirati stvaranjem sljedećih komponenti:

- *Strukturne komponente*
 - `Header.vue` – Zaglavlje stranice
 - `Footer.vue` – Podnožje stranice
 - `Navbar.vue` – Navigacijska traka, koja će sadržavati:
 - `LinkItem.vue` – Pojedinačna stavka navigacije
- *Stranice*
 - `PagePočetna.vue` – Početna stranica
 - `PageSlika.vue` – Stranica o izradi slike
 - `PageGrafika.vue` – Stranica o izradi grafike
- *Pomoćne komponente*
 - `Separator.vue` – Vizualni razdjelnik sadržaja
 - `JustifiedTextSection.vue` – Sekcija s tekстом poravnatim obostrano