**Vue.js – OSNOVE JAVASCRIPT FRAMEWORK-a**

**INSTALACIJA:** <https://vuejs.org/v2/guide/installation.html>

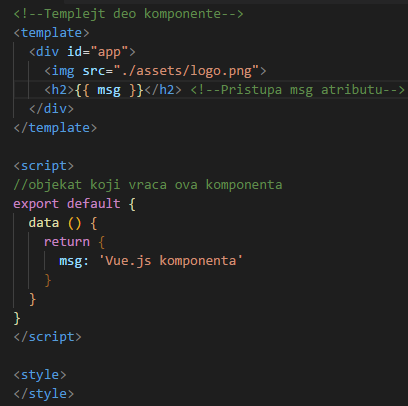
* Ako je upitanju samo jedan projekat onda se koristi script element koji je dat na sajtu odnosno CDN. On se pise u <head> html elementa kako bi se includovao, ali ce se js pisati u externom js fajlu, koji se inkluduje na kraju <body> elementa, gde ce da bude ceo Vue.js kod.

<script src="https://cdn.jsdelivr.net/npm/vue@2.5.13/dist/vue.js"></script>

* Ako se kreiraju SPA (single page applications) ili kompleksniji projekti onda se vue.js instalira u globalu na kompu preko vue.js CLI-a. On je dobar jer se pokrece na lokalnom serveru i kod se kompajlira prvo na kompu pa se tek onda izvrsava na browseru. Omogucava i koriscenje ES6.

1. Skine se Node.js
2. Otvori se cmd (sa **node –v** se proverava da li je instaliran node.js)
3. Vue.js se Instalira sa **npm install –g vue-cli**
4. Kada je instalirano sad se kreira folder u kome ce da se napravi projekat, ali pre toga postavimo putanju do tog foldera sa **cd** pa gde
5. Kreiramo sa **vue init <template\_name> <project\_name>** (template name vec kreirani neki delovi projekta od strane vue developera da se mi ne mucimo, ovde koristimo webpack-simple ima ih jos 5-6 sto se koriste)
6. Onda bi trebalo da iskoce neka pitanja vezana za projekat (sass je framework za css i ne mora da se instalira)
7. Nakon toga pokazu se koraci sta treba da se uradi – prvo **cd ime\_foldera, npm install, npm run dev** (Nakon cega bi trebao da se prikaze neki demo prikaz sajta u Chome-u). **CMD MORA DA BUDE UPALJEN DA BI RADIO LOCALHOST:8080!**
8. To je to!

**SADRŽAJ FOLDERA:**

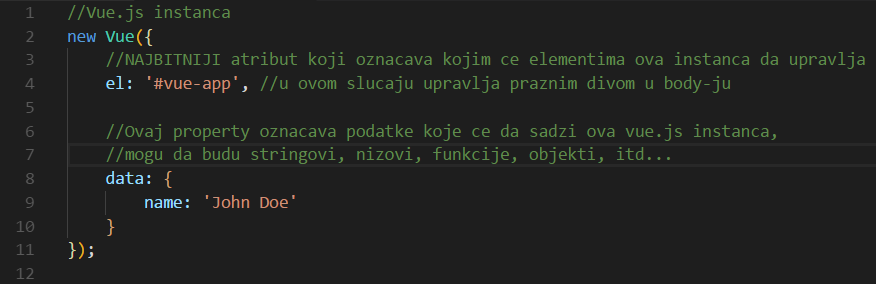
* **Node\_modules** folder u kome smo instalirali neke stvari i to ne treba da se dira.
* **Src** folder u kome ce da ide sav kod koji kreiramo znaci najbitniji folder. U njemu se nalaze:
  + **Assets** folder u koji smestamo sve slike koje zelimo da koristimo
  + **main.js** je fajl koji se pokrece kada pokrecemo aplikaciju, ovo kontrolise sve otpocetka. Ovde importujemo sve falove koje budemo dodavali. I ovde se nalazi Vue.js instanca koja kontrolise onaj div u index.html, i u toj instanci se renderuju sve komponente koje budemo kreirali i renderuju se u element koji kontrolise ta vue.js instanca.
  + **app.vue** je root component. Ovo je obicna vue.js komponenta koja se samo pise u fajlu, a ne kao Vue.component, i ima extenziju .vue. Znaci u njoj imamo template element i onda pisemo izmedju njega bilo koji html kod, a kada se kreira obicna komponenta ovo bi sve bilo u template atributu komponente. Takodje unutar ovog fajla imamo i script element u kome exportujemo objekat koji je kod obicnog kreiranja komponente drugi parametar. I imamo style element koji kontrolise izgled te komponente. Po defaultu tu se nalazi neki content tako da to moze da se obrise, ali trebalo bi da se ostave samo tri stvari **<template>, <script> i <style>,** i to da u script moramo da imamo export default, i u templejtu mora da postoji jedan root div u koji se ubacuju svi drugi elementi (ne moze da postoje dva razlicita root diva).
* **.gitignore** falj za git koji ignoruje neke fajlove kada se upload na git folder
* **Index.html** fajl u kome ce da se unosi html kod – inicijalno u njemu je ucitana skripta kojom se po defaultu kada se instalira otvara ona demo strana
* **Package.json** sadrzi sve informacije o nasem projektu i sve sta smo instalirali
* **Webpack.config.js** fajl koji nam omogucava koriscenje ES6 i svega ostalog
* Ostali fajlovi se generišu kada se build-uje projekat

Slika: Kako izgleda .vue fajl (komponenta)

**KREIRANJE APLIKACIJE SA CDN-om I OSNOVE VUE.JS-a**

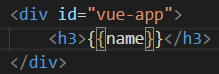
**1. KORAK – Vue.js Instanca**

* Instanca se kreira u posebnom .js fajlu. Ova vue.js instanca se moze koristiti za kontrolu dela nase aplikacije ili cele aplikacije. Vise instanci se koristi ako npr. na sajtu imamo widgete pa svaka instanca kontrolise jedan.
* Kako ova instanca kontrolise aplikaciju? U vitičastim zagradama se ovom objektu prosledjuju neke opcije (metode) šta ce aplikacija da radi.

**Dodavanje property-ja Vue.js instanci**

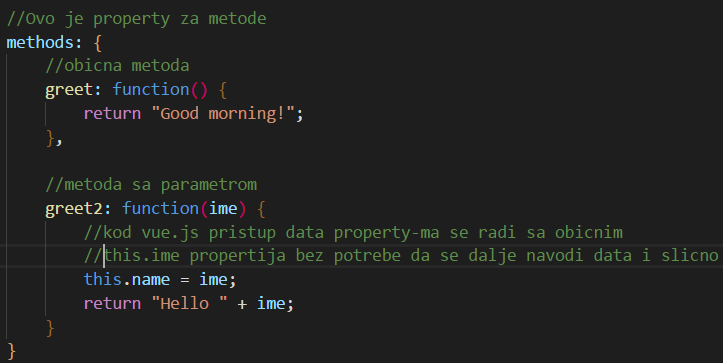
* **el: property** oznacava kojim ce se HTML elementima upravljati ova instanca. U ovom slucaju u html dokumentu postoji prazan div u body-u sa id-jem vue-app, i zato po id-ju nalazimo taj prazan div kojim ce se upravljati.
* **data: property** cuva sve podatke koji ova vue.js instanca ce da prikazuje na stranici. To mogu da budu stringovi, nizovi, funkcije, objekti, itd... U ovom slucaju postavimo property name da ima vrednost John Doe.

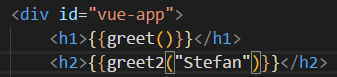
**Prikaz data vrednosti vue instance u html elementu**

* ****Vue.js prikazuje stvari na strani slicno kao kod php-a MySmarty, znaci kao template. Posto smo u js fajlu naveli kojim objektom ce da upravlja Vue instanca sada ako zelimo da ispisemo na strani neku stvar iz data property-ja onda u tom elementu koji se kontroliše samo stavimo vitičaste zagrade i u njima naziv property-ja koji se nalazi u vue instanci data property-ju, i on ce da prikaze tu vrednost.

I tako mozemo da prikazujemo koliko hocemo vrednosti.

**Poziv metoda iz vue instance**

* Isto kao i za podatke koji se smestaju u data property, metode se kod vue instance smestaju u methods: property. Pozivaju se isto u okviru HTML-a isto kao i data property-ji samo uz ().



**Pristup property-ijma vue instance**

* Kod vue.js pristup data property-ma se radi sa obicnim this.ime-property-ja bez potrebe da se dalje navodi data i slicno. Znaci oni su direktno vezani za vue instancu.

**DataBinding**

* DataBinding se koristi za spajanje atributa html elemenata sa vrednostima vue instance. Npr. ako zelimo da postavimo href u <a> html elementu da pokazuje na vrednost porperty-ja „website“ kod vue instance to ne moze da se uradi sa obicnim vitičastim zagradama, vec mora da se koristi kljucna reč **v-bind: .** Znaci ona kaze vue da spoji nesto sa nekim podatkom iz istance. Može i skraćeno da se piše samo sa **:** bez v-bind. U nasem primeru smo postavili v-bind za website property (ime property-ja se u ovom slučaju piše bez vitičastih zagrada i pod navodnicima), a : za name property kod inputa.

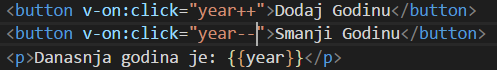
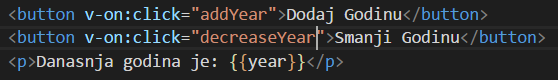


**Obrada html tag-a u Vue.js**

* I vue.js podrzava DOM, tako da i ovde moze dinamicki da se ubacuje html na stanu. Ako zelimo da direktno u vue instanci kreiramo html kod, onda da bi smo prikazali to u html fajlu moramo da unutar odredjenog html elementa dodamo atribut **v-html=“naziv propery-ja“.** I onda ce vue unutar tog elementa da postavi taj html element koji se nalazi kao string u nekom od property-ja.

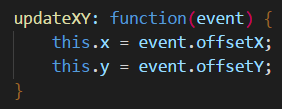
**Obrada evenat-a u Vue.js -** [**https://vuejs.org/v2/guide/events.html**](https://vuejs.org/v2/guide/events.html)

* U javaScriptu postoje razni eventi koji se u Vue.js jednostavno obradjuju sa atributom u HTML-u sa **v-on:“naziv eventa“ = „sta ce se izvrsiti“.** Sta ce se izvrsiti mozemo direktno da napisemo u HTML-u, ali to ne bi trebalo da se praktikuje jer je bolje da se odatle pozove metoda iz externog .js fajla i da se ona izvrsi. Isto kao i kod v-bind: i ovde moze krace da se zapise umesto v-on, kao **@”naziv-eventa”** Neki eventi koji postoje: click, doubleclick, hover, itd...



Direktno u HTML-u Poziv metoda

* Kada se triggeruje bilo koji event tada se kreira objekat **event** koji mozemo da koristimo u funkcijama koje se tada pokrecu. Najcesce se on koristi za koordinate misa tj. offsetX i offsetY.



Kraci zapis

event objekat

**Trigger event samo jednom (Event Modifiers)**

* Moguće je namestiti da se event koji se postavi na neki button ili nesto drugo triggeruje samo prvi put i onda svaki sledeci put nece se nista desiti. To se radi pomocu event modifier koji se zove **once i** ide odmah posle naziva eventa.

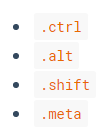
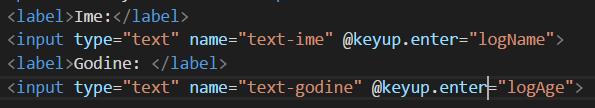


Samo jednom moze da se klikne na button i posle nista

Dodatni Modifieri

**Obrada keyboard evenat-a -** [**https://www.w3schools.com/jsref/dom\_obj\_event.asp**](https://www.w3schools.com/jsref/dom_obj_event.asp)

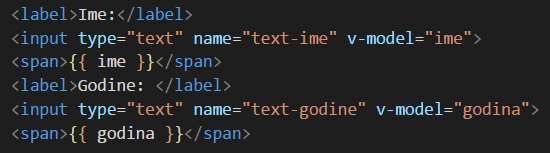
* Kao i obicne evente moguce je i obraditi evente kada se nesto unosi sa tastature, odnosno kad se pritisne i pusti neki taster. Postoje keyup, keydown i keypress. Keyup ce se aktivirati svaki put kada se pusti neki taster. Keydown i keypress se aktivira svaki put kada se pritisne neki taster. Pomoću event modifiers-a mozemo da prekinemo triggerovanje ovih evenata svaki put koriscenjem npr. enter ili alt modifiera.



Kada se unese neki tekst i pritisne enter onda ce se aktivirati funkcija

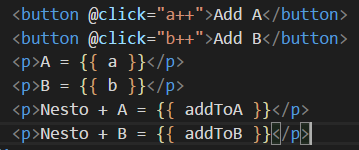
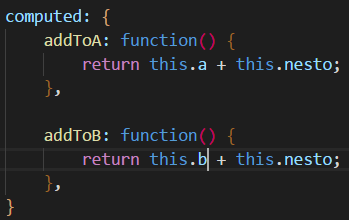
Svi keyboard modifieri

**Two-way data binding – Uporedo updejtovanje vrednosti podataka (v-model)**

* Npr. ako zelimo da uporedo prikazujemo nesto dok korisnik to unosi onda se koristi two-way data binding. Kada korisnik nesto unosi ono automatski menja vrednost zadatog propertija na unetu vrednost. Koristi se ključna reč **v-model = „naziv-propertyja koji se menja“,** i nije potrebdno da se kombinuje sa keyboard eventima.

Kako se unosi ime on se odmah stampa u span element tj. Vrednost property-ja se automatski menja i prikazuje se u span elementu

**Computed properties – izvrsavanje tacno jedne funkcije**

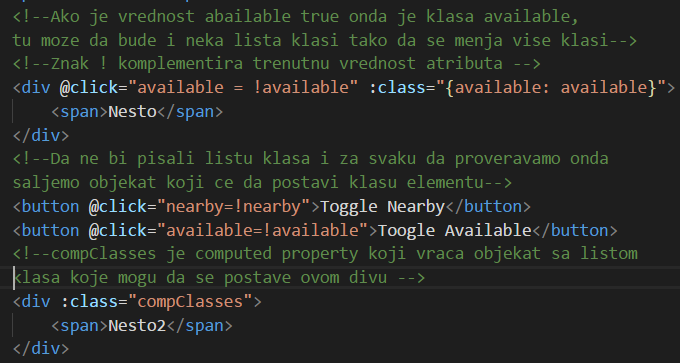
* Kod vue.js se desava da pri bilo kom eventu kada se menja neka vrednost desava se da se pozivaju sve funkcije na strani, iako ce samo jedna funkcija biti izvrsena i vrednost samo jedne promenljive ce se promeniti idalje ce se pokrenuti sve druge funkcije bez da ista urade. Da bi se to sprecilo i da bi se tacno jedna funkcija pozivala kada treba koriste se **Computed Properties** koje su iste kao i metode samo sto se ne pozivaju kao one vec kao obicna promenljiva. Oni se pozivaju samo kad se menja tacno ta promenljiva za koju su namenjeni. U vue.js instanci oni se nalaze u posebnom objektu **computed:**.

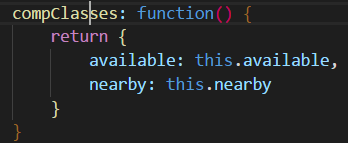
addToA i addToB su Computed Properties

Pisu se isto kao obicne funkcije

**Dinamički CSS – postavljanje atributa dinamički**

* Unutar HTML-a pomoću vue.js mozemo dinamicki da postavljamo i menjamo atribute u zavisnosti od vue.js vrednosti u instanci. Mora da se piše unutar vitičastih zagrada. I onda za sve vrednosti koje su true postavice se ime klase na tu vrednost (moze imati i vise klasa).

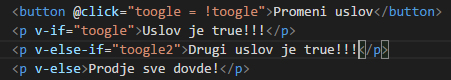




Primer direktnog proveravanja i sa objektom

**If – else uslov (v-if & v-else & v-else-if)**

* Kod vue.js if uslov se pise sa kljucnom recju **v-if=“boolean“.** On je postavljen kao atribut nekog HTML elementa, koji kada je true onda je taj element vidljiv, a kada je false onda je hidden. Takodje imamo i else granu koja se pise **v-else-if=“boolean“.**

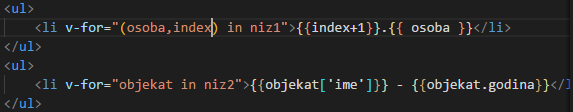


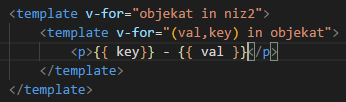
Ako je toogle true prikazace se prvi paragraf, ako je false proverice se toogle2 i prikaza ce se drugi paragraf, a ako oba su false onda ce se prikazati poslednji else paragraf.

**Prikaz i sakrivanje elemenata u zavisnosti od uslova (v-show)**

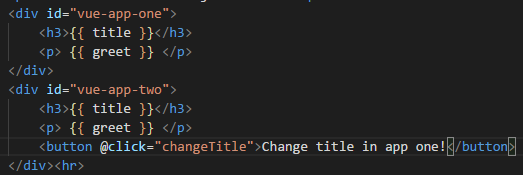
* Iako ovu funkciju moze da uradi i if uslov, postoji poseban vue.js atribut koji je namenjen za to. Za to se koristi kljucna rec **v-show=“boolean“,** koja prikazuje element ako je boolean true, a sakriva ako je boolean false. Razlika izmedju if i show jeste u tome sto ako je boolean false if uslov ga u opste na prikazuje u DOM-u, a show samo postavi atribut display na none (koji sakriva element), ali se element idalje vidi u DOM-u.

**For petlja (v-for)**

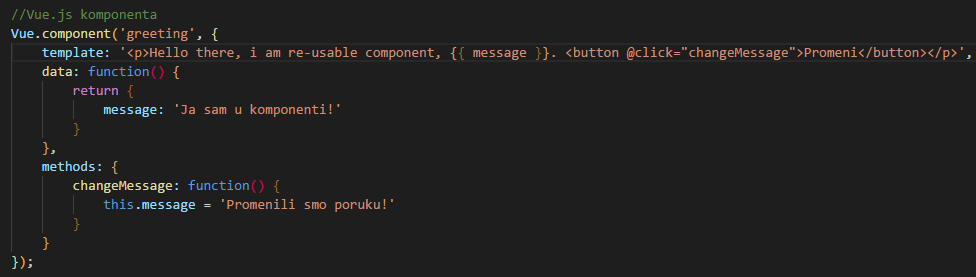
* Kao i svuda for petlja se koristi za prolazak kroz elemente niza. Kod vue.js se koristi kljucna rec **v-for=“element in niz“.** Slicno kao foreach u drugim jezicima. Element je element kome se pristupa i svaku iteraciju pokazuje na drugi element u nizu, a niz je jedan kroz koji prolazimo. Ako zelimo i index elementa u nizu onda moramo da stavimo **(element,index)** bas u ovom redosledu i onda preko promenljive index imam koji je index elementa u nizu.

* Takodje moguce je sa v-for da se prolazi kroz objekat i njegove atribute bez da se zna koji su atributi. **(val,key)** i mora bas u ovom redosledu, val je vrednost atributa, a key je ime tog atributa. **Template** element u ovom slucaju nema nikakve veze jer on se ne prikazuje u DOM-u vec sluzi samo da se prikazu elementi unutar njega, tako da ce samo u DOM-u da se vide paragrafi za svaki objekat.

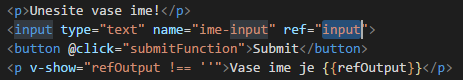
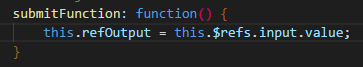
**Više Vue.js instanci**

* Svaka vue.js instanca moze da kontrolise drugi deo stranice. To radimo tako sto dve razlicite promenljive dodamo Vue objekat. I onda preko tih promenljivih mozemo da pristupamo tom objektu tj. tom delu strane. A ako zelimo da pristupimo nekom atributu prve instance iz druge instance to radimo kao sa bilo kojim objektom **one.atribut.**

**Vue.js components**

* Vue.js komponente su kao templejti u OOP. Definisu se pre svega i onda se samo koriste u vue instancama. Kreira se sa kljucnom recju **component – Vue.component(„Naziv“, objekat).** Objekat je isto kao i kod Vue instance odredjuje se sta sadrzi komponenta i zasta sluzi. U taj objekat ide atribut **template: „neki html kod ili nesto“.** Ta komponenta se u HTML fajlu kreira pomoću naziva koji smo stavili kad smo deklarisali komponentu. Npr. **<“Naziv“></“Naziv“>. (**Naravno ovaj kod mora da bude unutar Vue.js instanci da bi vue.js znao da je rec o njegovoj komponenti). Kada postavimo ovaj „element“ na stranu, na stani ce biti prikazano samo ono sto se nalazi u template atributu, i samo ce to da se vidi u inspect strane, a ne ta komponenta. OVA KOMPONENTA SE KORISTI SAMO U HTML-u.
* Takodje komponente mogu imati i **data** atribut tj. da imaju neke vrednosti, samo za razliku od Vue instance ovde je data atribut kao **funkcija** koja vraca objekat, dok je kod instance data objekat sam po sebi. Razlog ovome jeste to sto komponentu mozemo da kreiramo vise puta i ako promenimo nesto u data funkciji onda ce to ostati samo za tu kreiranu kopiju, a kada bi data bila objekat onda bi se to update za sve kopije komponente.
* Takodje mogu i metode da se kreiraju ali oni se kreiraju isto kao i kod Vue.js instance.

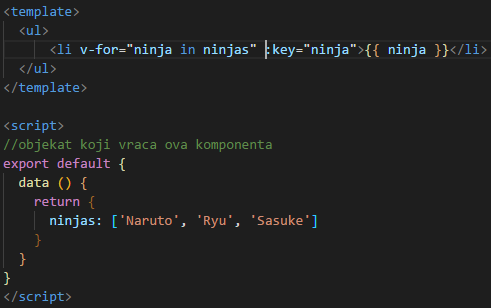
**Refs**

* Refs (reference) se koriste da se pristupi HTML elementima i njihovim atributima iz Vue.js instance. Npr. za input text da pristupimo sta je uneto, mozemo i bilo kom drugom elementu da pristupimo i da vratimo InnerHtml ili InnerText. On se postavlja kao atribut elementu kljucnom recju **ref=“neki naziv tom elementu“.** U Vue.js instanci svim ref pristupamo pomocu **this.$refs –** this misli se na sve ref u tom vue elementu, $refs vraca niz svih ref koji su u tom vue html elementu. Ako zelimo nekom konkretnom ref elementu da pristupimo ide se onda ime tog ref-a i onda koji atribut trazimo npr. **this.$refs.ime.value –** vraca vrednost elementa koji ima ref ime.

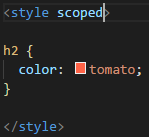
**KREIRANJE WEB APLIKACIJE SA VUE.JS CLI**

**(za ovo je potrebno da je cmd uvek otvoren zbog localhost-a)**

**Ugnježdavanje više komponenti (Nesting components)**

* Root komponenta je u app.vue, a sve ostale komponente mozemo da stavimo unutar te root komponente, ali kod vecih aplikacija ona je izdeljena na vise komponenti koje se prikazuju kada trebaju (npr. Header komponenta, Article komponenta i Footer komponenta i onda ih nestujemo u root komponentu, a onda Header komponenta da ima Login komponentu i Link komponentu). Ovo radimo da ne bi kreirali komponente svaki put kad nam zatrebaju nego kreiramo jednu komponentu i nestujemo gde nam treba.
* Sve nove komponente kreiramo unutar **src foldera** sa extenzijom **.vue.** Kada kreiramo komponentu sa svim njenim podacima tada je nestujemo tj. importujemo u App.vue tj. glavnu aplikaciju. Ali pre toga moramo da registrujemo komponentu: globalno ili lokalno. Globalno to znaci da ta kreirana komponenta moze da se nestuje u bilo koju drugu komponentu, a lokalno znaci da se ona koristi samo unutar te komponente za koju se registruje.
* **Globalno** registrovanje se radi u **main.js** fajlu kako bi sve druge komponente mogle da vide tu komponentu. Sintaksa je **import „Neko ime komponente“ from ’putanja do komponente’,** nakon toga kreiramo komponentu sa **Vue.component(’kako\_ce\_da\_bude\_tag\_u\_HTML’, „ime te komponente“).** I onda se ova komponenta koristi u bilo kojoj drugoj komponenti pomocu taga koji smo registrovali za nju: **<“tag\_komponente“></“tag\_komponente“>.** 3 slika pokazuje izgled te komponente.
* **Lokalno** registrovanje se radi unutar komponente u koju cemo samo da koristimo tu drugu komponentu. I to se radi u **<script>** delu te komponente. Importuje se isto kao i kod globalnog nacina, i onda se u export default objektu dodaje atribut components (koji sadrzi informacije o lokalnim komponentama) i unutra navodimo prvo **’tag\_ime’: „Neko ime komponente“.** Koriscenje je isto kao kod globalne registracije.

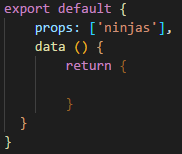
**CSS komponenti (lokalni css)**

* Svaka komponenta kada se kreira ima tri elementa <template>, <script> i <style>. Ovaj poslednji element se koristi za stilizovanje komponenti i to je deo za CSS kod. On radi isto kao da imamo externi css samo sto se ovo direkto odnosi na tu komponentu.
* Ako nestujemo neku komponentu lokalno u drugu onda ce sav stil iz te root komponente preci i na tu nestovanu, tj. ona ce da nasledi sve stilove roditeljske komponente. Ali ako hocemo da svaka komponenta ima svoj stil koji napisemo u njenom fajlu onda koristimo kljucnu rec **scoped** kao atribut za tag **<style>, tj. <style scoped>**. Tako ce onda sav stil da se odnosi samo na tu komponentu. Vue.js im dodaje neki jedinstveni id tako da se onda elementi razlikuju i zbog toga nema preklapanja stilova.

**Greška kada se koristi v-for**

* Kada se koristi v-for potrebno je da se koristi pored toga i **v-bind:key=“id“ ili :key=“id“.** Razlog tome je sto vue.js pri prolazu kroz elemente mora da prati neki id za svaki od njih tako je potrebno da se prosledi neki jedinstveni id za svaki element kako ne bi prikazivao gresku. Ovo se nista ne odnosi na projekat vec samo je potrebno zbog Vue.js-a.

**Props**

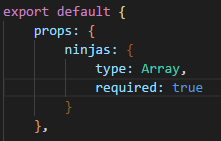
* Props se koriste za slanje podataka iz root komponente u komponentu u koju je nestovana. Da bi se primio props tj. neka vrednost potrebno je da se u export default objektu komponente koja prima props definise atribut **props: [],** koji je sam po sebi niz i u koga se upisuje sta ocekujemo da primimo, tj. moramo da navedemo ime promenljive koju cemo da primimo kako bi se pomocu te promenljive posle toj vrednosti moglo da se pristupa kao da je napisana u **data** delu. A props se salje tako sto se u root komponenti, tamo gde je u <template> elementu napisana nestovana komponenta, na nju se doda atribut **v-bind:**“**ime\_promenljive“=“naziv\_lokalne“,** ime\_promenljive je ime promenljivađe koju smo u nestovanu komponentu nazvali u onom nizu props, a naziv lokalne promenljive je naziv one promenljive cija se vrednost salje toj nestovanoj komponenti. Takodje ovom props se pristupa kao da je u data delu sa **this.ime\_propsa** (za metode i computed).



Slanje props-a u root komponenti

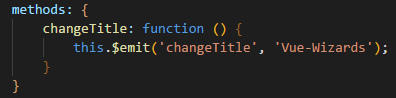
Prijem propsa

**Validacija Props-a**

* Prilikom slanja propsa mozemo da definisemo sta ocekujemo da stigne. Npr. ocekujemo niz ili string ili broj itd. Tada **props** atribut postaje objekat, a svi props postaju takodje objekti koji imaju svoje atribute koji navode sta se ocekuje da stigne, tj. definisemo **type** i da li je potrebno da se posalje **required**, inace nece da radi aplikacija. Ovo je jedan od bitnih stvari i treba se kroistiti kad se radi sa props.
* **Kada se kao props salje niz ili objekat treba znati da bilo koja promena u tom objektu ili nizu u nested komponenti bice izvrsena i u root komponenti, tj. bice promenjen uopste taj niz ili objekat tako da ce svi da vide tu promenu. Ali ako se salje string ili neki primitivni tip onda ce promene biti vidljive samo lokalno, tj. unutar te komponente.**

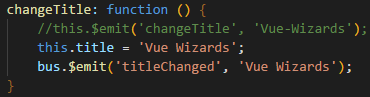
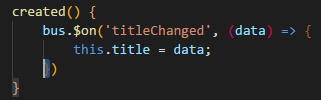
**Eventi**

* Eventi su kao slotovi i signali u QT-u. Jedna komponenta aktivira event koji root komponenta osluskuje i kada se aktivira izvrsi se neka metoda. Ovo se koristi kada neku promenu zelimo da posaljemo svim komponentama tj. da se vrednost ne promeni samo u lokalnoj komponenti nego u globalu i da sve komponente imaju promenjenu vrednost.
* Koraci kod eventa su: 1) aktiviranje eventa, 2) slusanje eventa i reagovanje na event. Lokalna komponenta akrivira event, root komponenta slusa i reaguje na event.

1. Slanje se radi pomocu metode u kojoj pomocu kljucne reči **this.$emit** aktiviramo event. Sintaksa je: **this.$emit(„naziv\_eventa“, „niz\_parametara“).** Naziv eventa moze da bude bilo šta ali mora da se zapamti jer ce se po tom nazivu osluskivati event, a niz parametara jeste ono sto taj event salje odnosno šta hocemo da promenimo (moze i samo jedan parametar da se posalje). Naravno ovaj event se aktivira npr. na button ili na click.
2. Osluskivanje se radi u root komponenti jer ona mora da prosledi sve promene drugim komponentama. Na odgovarajucu komponentu od koje se ocekuje event postavimo **v-on:“naziv\_eventa“=“neka\_funkcija($event)“.** Naziv eventa je ime eventa koji smo dali u lokalnoj komponenti, neka funkcija je funkcija koja ce da se aktivira na event i **$event** jesu parametri koji ce event da posalje (niz ili jedan parametar). Kada se event aktivira event i odgovarajuca funkcija, sve ostale komponente ima da dobiju promenjene vrednosti promenljivih.



**Event bus**

* Je ustvari vue.js objekat koji se kreira u komponentama koje zele da komuniciraju bez posredstva root komponente (ovo je u nekim situacijama bolje nego obicni eventi). Znači kreira se po jedan objekat u obe komponente. U objektu jedne komponente se emituje, a u objektu druge komponente osluskuje.
* Prvo se u main.js kreira instanca vue.js kao: **export const bus = new Vue();** export zato sto cemo da ga koristimo u drugim komponentama i tamo importujemo, ne mora da se zove bus moze bilo sta. Drugo, importujemo ovaj bus u sve komponente koje ce da ga koriste: **import { bus } from '../../main';.** Trece, emitujemo event sa: **bus.$emit('naziv\_eventa', 'parametri');** koristimo bus da emitujemo event. Cetvrto, u komponenti koja osluskuje moramo da koristimo **LifeCycle Hooks created()** koja se koristi da proveri da li je nesto kreirano, a u tome osluskujemo event sa: **bus.$on(„naziv\_eventa“, function (data) {}.** Naziv eventa je onaj koji smo dali kad smo emitovali, a funkcija je ona koja se aktivira kada se event aktivira, a data su oni parametri koje smo prosledili (ne mora da se zove data).

Emitujemo event Slusamo event

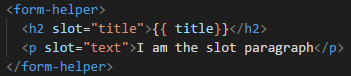


Exportovanje i imporotvanje bus-a

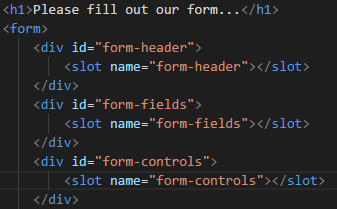
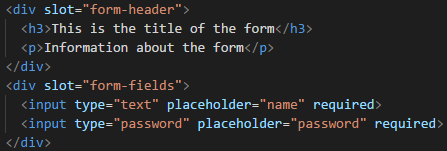
**Life-cycle hooks**

* Kada kreiramo neki objekat on ide kroz neki ciklus prilikom kreiranja (life-cycle), i u vue.js postoje definisane funkcije koje se aktiviraju u odredjenom ciklusu zivota instance. Ove funkcije se u vue.js zovu **hooks.** Ovo vazi i za komponente jer su i one instance. I Sve ove funkcije idu u export default objekat kao i **data()** (tako se pisu).
* Prva od tih funkcija je **beforeCreated() {},** u koju mozemo nesto da radimo pre nego što se kreira neka instanca. Nakon toga je funkcija **created() {},** koja se aktivira kada je objekat kreiran, i u nju mozemo da radimo nesto sta ce se aktivirati cim se objekat kreira, i ovde je najbolje da se fatch bilo koja data (npr iz baze podataka). Sledeca funkcija je **beforeMount() {},** ona se aktivira kada je objekat kreiran ali nije se jos prikazao na strani tj. jos nije usao u DOM. Posle toga funkcija **mounted() {},** kada se ubaci u DOM onda se aktivira ova funkcija i onda mozemo da menjamo bilo sta iz DOM-a tog objekta. Nakon toga kada je sve vec prikazano na strani mi mozemo da menjamo neke stvari, a kad vue.js detektuje neke promene onda se aktivira prvo funkcija **beforeUpdate() {},** gde mozemo da menjamo bilo sta pre nego sto se vrednost necega promeni. Kada se promeni onda imamo sledecu funkciju **updated() {}**, koja se aktivira kada se promene prikazu na strani i tada imamo pristup update DOM-u. I na kraju ako hocemo da obrisemo nesto aktivira se funkcija **beforeDestroy() {},** i na kraju kada se obrise funkcija **destroyed() {}.**
* **Test moze da se uradi u svakoj ovoj funkciji se stavi alert().**

**Slotovi**

* Slotovi se u vue.js koriste kako bi se komponentama prosledio neki HTML kod i onda on ubacio u njihov template. Slotovi se ubacuju pomocu tag-a u HTML-u **<slot></slot>** i izmedju se ne pise nista jer se to izmedju pise u komponenti iz koje se prosledjuje. U komponenti koja prosledjuje HTML kod (to je uvek roditeljska komponenta koja salje nested komponenti) on se pise izmedju tagova komponente kojoj se salje kod. Takodje moze postojati vise slotova i onda da se HTML kod koji se salje da se razdvoji po tim slotovima. To se radi tako sto se kod koda koji se salje postavi atribut **slot=“ime\_slota“** , a kod komponente koji prima kod, slot tagu se stavi atribut **name=“ime\_slota“.** Takodje preko slota mozemo da prikazemo neku vrednost atributa komponente sto salje.



* Slotovi dobijaju smisao kod formi, gde se omogucava da se kreira neki template formi koji stalno mogu da se koriste i da se menja ono sto neko zeli. Konkretno receno u nested komponenti se postave slotovi koji ce da se koriste i onda se u root komponenti u svaki od tih slotova stavlja ono sto zelimo, navodeci samo tacno ime tog slota.

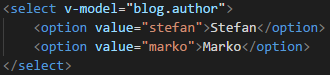
**Dinamičke komponente**

* Vue.js nam omogucava da odredimo koja ce komponenta da se dinamički pojavi na stranici. Za to se koristi vue.js html tag **<component></component>** i u njega moramo da bind ime komponente koja ce da se dinamicki prikaze. To se radi sa **:is=“promenljiva“,** promenljiva nosi naziv komponente koja se prikazuje. Ta promenljiva moze npr. da se toggle i onda ce na svaki button click da se prikaze druga komponenta.
* Kada promenimo sa jedne forme na drugu, sve sto smo napisali u prvoj bice obrisano prilikom otvaranja druge. Ako zelimo da to izbegnemo koristi se vue.js tag **<keep-alive></keep-alive>,** unutar koga se navodi komponenta koja se nikada ne brise iako se ona hide idalje ostaju svi podaci na njoj.

**Checkbox binding**

* Vue.js nam omogucava da sa v-model bindujemo i checkbox-eve tj. kada se check neki box da se odmah vidi sta je checked. Ako imamo vise checkbox-eva onda mozemo da stavimo da se svi koji su stiklirani pa njihova vrednost doda u taj niz.

**Select box binding**

* Isto kao i za checkbox i za sve ostalo, tako mozemo i select box da bindujemo.

**Custom directives**

* V-for, v-if, v-model, itd...Sve su to vue.js direktive i sve imaju zajednicko v- na pocetku, ali ako ne postoji neka direktiva koja nam treba moze se kreirati neka nova (custom) direktiva. Custom direktive mozemo da kreiramo na lokalnom ili na globalnom nivou. Na globlanom nivou kreiramo ih u **main.js** fajlu koristeci sintaksu **Vue.directive(„ime\_direktive“, objekat).** Ime direktive je ono ime sto ce da stoji pored v- znaci ovde se pise bez tog v-, a objekat je ustvari sta ce da radi ta direktiva. Kao i sve ostalo i directive imaju life hooks funkcije i ovde se najcesce koristi **bind(el, binding, vnode)** funkcija. Ona se aktivira kada se direktiva napise kod nekog elementa, vnode nece cesto da se koristi i to je neki virtuelni node u DOM-u, binding sadrzi sve informacije o bindovanju sa elementom tj. ako imamo nesto da prosledimo kada napisemo direktivu ili tako nesto, el je element za koji je vezana direktiva.Kod binding vrednost koju prosledimo sa = dobija se kao **binding.value,** a argument koji prosledimo sa : dobijamo sa **binding.arg.**

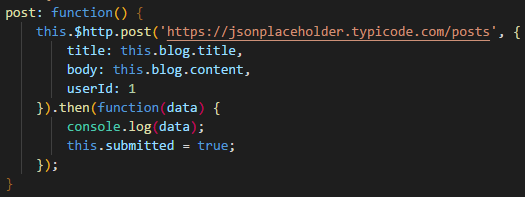


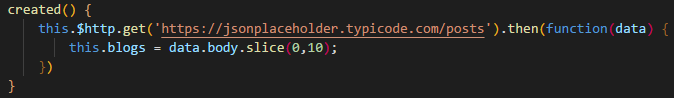




**HTTP Requests -** [**https://github.com/pagekit/vue-resource**](https://github.com/pagekit/vue-resource)

* Za kreiranje HTTP requesta koristi se plug in **vue-resource.** Prvo se on instalira u folder projekta koji se pravi, sa **npm install vue-resource.** Kada se on instalira mozemo da proverimo u fajlu **package.json** i potrazimo da li u dependencies pise vue-resource i verzija. Nakon toga moramo u **main.js** da importujemo ovaj plugin i da naznacimo da ocemo da koristimo, to radimo sa:
* Ovi requests-i se koriste za kompunikaciju sa bazom podataka, tj. sa slanje i primanje. Za proveru rada aplikacije bez da se koristi prava baza podataka moze da se koristi onine servis <https://jsonplaceholder.typicode.com/> koji sadrzi neke od metoda koje mozemo da koristimo kao sa pravom bazom (svi podaci su po defaultu kreirani), i na sajtu se nalaze sintaxe za sve requests-e.
* **POST –** sintaksa za post metodu je **this.$http.post(„url\_za\_bazu“, {objekat\_koji\_saljemo})**. $http je unutar onog vue-resource koji smo instalirali, post predstavlja naziv metode (u ovom slucaju post), url za bazu je url koji se koristi za post deo, i sledeci parametar post metode je objekat koji saljemo, pazeci na to da ima sve atribute kao sto su kolone u nasoj bazi podataka. Nakon toga mozemo da stavimo i **.then()** metodu koja ce da izvrsi nesto kada se posalju podaci.



* **GET -** sintaksa za get metodu je slicna kao za post metodu: **this.$http.get(„url\_za\_bazu“).** Ovaj HTTP request je najbolje izvrsiti kada se kreira komponenta koja treba da prikaze te podatke tj. u vue life hook funkciji **created().** Za razliku od post metode gde je nakon toga then() neobavezno kod get metode je obavezno jer mora da se kaze sta ce se raditi sa podacima, znaci **this.$http.get(„url\_za\_bazu“).then(function(data) {}).** Ova funkcija ce se izvrsiti cim se prikupe podaci iz baze sa get metodom, data parametar funkcije jesu ti prikupljeni podaci i jos neke informacije (uradi console.log(data) za proveru), i u toj funkciji trebamo negde da smestimo te prikupljene podatke.