

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

[1] Vue.js osnove

Vue je JavaScript okvir (framework) za izradu korisničkih sučelja. Građen je na standardima HTML-a, CSS-a i JavaScript-a i pruža **deklarativno** programiranje bazirano na **komponentama** što omogućuje efikasnu izradu korisničkih sučelja bilo koje kompleksnosti.



Posljednje ažurirano: 25.2.2025.

Sadržaj

- Programsko inženjerstvo
- [1] Vue.js osnove
 - Sadržaj
 - Uvod
 - Single Page Application SPA
 - Primjer jednostavne vounter aplikacije
 - Minimalni primjer Vue aplikacije
 - Komponentni pristup u Vue.js
 - Primjer liste voća koristeći samo html
 - Primjer liste voća koristeći html i js
 - Primjer liste voća koristeći Vue
 - Primjer liste voća koristeći Vue i komponentu
 - Primjer liste voća i liste povrća koristeći Vue i komponente
- Postavljanje aplikacije/projekta
 - Koraci za postavljanje projekta:
 - Struktura projekta
 - App.vue
 - main.js
 - main.css
- Tailwind
 - Prednosti Tailwind-a
 - Izravno pisanje u class atributu
 - Lakše održavanje kôda

- Jednostavna responzivnost
- Konfiguracija
- Instalacija Tailwind-a
- Tailwind Osnove
 - Boje – Lista svih boja:
 - Text
 - Boja teksta
 - Veličina fonta
 - Stil fonta
 - Poravnanje teksta
 - Prelom teksta
 - Veličine
 - Razmaci
 - Flex
 - Kako centrirati div?
- Samostalni zadatak za vježbu 1

Uvod

Vue.js je JavaScript okvir koji se koristi za izradu dinamičkih i interaktivnih web aplikacija temeljenih na promjenjivim podacima (data-driven). Omogućuje učinkovito ažuriranje korisničkog sučelja bez potrebe za ponovnim učitavanjem cijele stranice.

Single Page Application (SPA)

Vue se često koristi za izradu **Single Page Applications (SPA)** – aplikacija koje učitavaju jednu HTML stranicu i dinamički mijenjaju njezin sadržaj ovisno o interakciji korisnika, bez potrebe za ponovnim učitavanjem stranice s poslužitelja.

U SPA arhitekturi:

- Sav **routing** (navigacija između različitih dijelova aplikacije) odvija se u pregledniku, umjesto na poslužitelju.
- Vue Router omogućuje navigaciju između "stranica" bez osvježavanja cijelog sadržaja.
- API pozivi prema backendu koriste se za dohvaćanje podataka, često u JSON formatu, putem **Axios** ili Fetch API-ja.
- Vuex ili Pinia mogu se koristiti za upravljanje stanjem aplikacije kako bi podaci ostali dosljedni kroz cijelu aplikaciju.

Ovakav pristup poboljšava korisničko iskustvo jer aplikacija radi brzo i fluidno, bez zastoja uzrokovanih ponovnim učitavanjem stranice.

Primjer jednostavne vounter aplikacije

U sljedećem primjeru implementiran je jednostavan brojač koji se inkrementira klikom na gumb:

```
<!DOCTYPE html>
<html lang="hr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Minimalna aplikacija bez Vue.js</title>
</head>
<body>
  <div id="app">
    <button id="counterButton">
      Count is: 0
    </button>
  </div>

  <script>
    const counterButton = document.getElementById("counterButton");
    let count = 0;

    counterButton.addEventListener("click", () => {
      count++;
      counterButton.textContent = `Count is: ${count}`;
    });
  </script>
</body>
</html>
```

Count is: 0 ➡ Count is: 4

Možemo uočiti nekoliko stvari:

- dohvaćanje **HTML elementa** koji prikazuje brojač
- dodavanje **event listenera** za "click"
- ažuriranje *vrijednosti* unutar **DOM-a**

Minimalni primjer Vue aplikacije

Ako bismo istu funkcionalnost implementirali pomoću **Vue.js**, onda ne moramo ručno ažurirati DOM:

```
<!DOCTYPE html>
<html lang="hr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Minimalna Vue aplikacija</title>
  <script src="https://unpkg.com/Vue@3/dist/Vue.global.js"></script>
</head>
<body>
  <div id="app">
    <button @click="count++"> <!-- Event handling -->
      Count is: {{ count }} <!-- Template syntax -->
    </button>
  </div>

  <script>
    const { createApp, ref } = Vue;

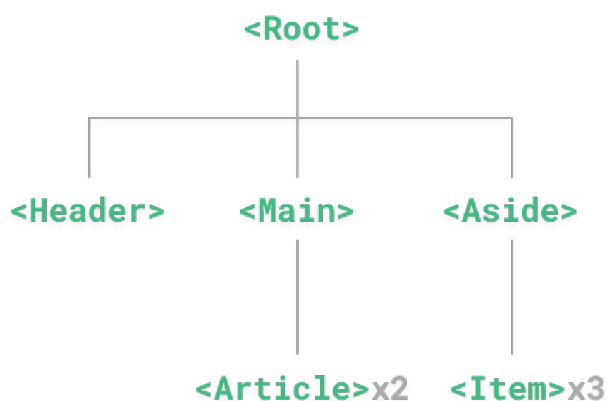
    createApp({
      setup() {
        return {
          count: ref(0) // Reactive variable
        };
      }
    }).mount("#app");
  </script>
</body>
</html>
```

Gornji primjer pokazuje dvije ključne značajke:

- **Deklarativno renderiranje** – Vue koristi *template sintaksu* unutar HTML-a kako bi prikazao dinamičke podatke. Promjene u varijabli `count` automatski se prikazuju u DOM-u.
- **Reaktivnost** – Vue koristi reaktivne varijable (ovdje `ref(0)`) kako bi pratio promjene i ažurirao korisničko sučelje bez potrebe za ručnim manipuliranjem DOM-a.

Komponentni pristup u Vue.js

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.



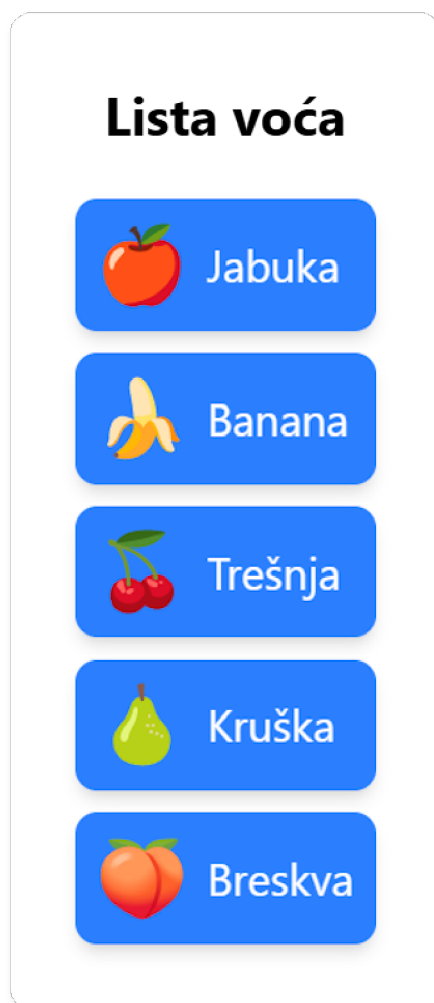
U sljedećim primjerima prikazat ćemo postupni prijelaz od običnog HTML-a do Vue komponenti, detaljno objašnjavajući svaki korak tog procesa. Kao konkretan primjer, izradit ćemo aplikaciju koja prikazuje listu voća.

Primjer liste voća koristeći samo html

```

6      <div class="p-6 bg-white shadow-lg rounded-lg">
7        <h2 class="text-xl font-bold mb-4 text-center">
8          Lista voća
9        </h2>
10       <ul class="space-y-2">
11         <li class="p-2 bg-blue-500 text-white rounded-lg
12           shadow-md flex items-center space-x-2">
13           <svg xmlns="http://www.w3.org/2000/svg"
14             width="32" height="32"
15             viewBox="0 0 128 128"><path fill="#dc0d28" d="m45
16             <span>Jabuka</span>
17         </li>
18         <li class="p-2 bg-blue-500 text-white rounded-lg
19           shadow-md flex items-center space-x-2">
20           <svg xmlns="http://www.w3.org/2000/svg"
21             width="32" height="32"
22             viewBox="0 0 128 128"><path fill="#fff8e0" d="M11
23             <span>Banana</span>
24         </li>
25         <li class="p-2 bg-blue-500 text-white rounded-lg
26           shadow-md flex items-center space-x-2">
27           <svg xmlns="http://www.w3.org/2000/svg"
28             width="32" height="32"
29             viewBox="0 0 128 128"><path fill="#af0c1a" d="m79
30             <span>Trešnja</span>
31         </li>
32         <li class="p-2 bg-blue-500 text-white rounded-lg
33           shadow-md flex items-center space-x-2">
34           <svg xmlns="http://www.w3.org/2000/svg"
35             width="32" height="32"
36             viewBox="0 0 128 128"><path fill="#b7d118" d="M64
37             <span>Kruška</span>
38         </li>
39         <li class="p-2 bg-blue-500 text-white rounded-lg
40           shadow-md flex items-center space-x-2">
41           <svg xmlns="http://www.w3.org/2000/svg"
42             width="32" height="32"
43             viewBox="0 0 128 128"><path fill="#69a246" d="M10
44             <span>Breskva</span>
45         </li>
46       </ul>
47     </div>

```



Možemo primijetiti da se u ovom pristupu ponavlja isti kôd za svaku stavku, pri čemu je jedina razlika naziv stavke i ikona. Ova duplicirana logika može otežati održavanje aplikacije, jer svaki put kad dođe do promjene, potrebno je ažurirati svaku instancu tog obrasca.

Međutim, korištenjem osnovnih JavaScript tehnika, možemo riješiti ove izazove tako da dinamički punimo listu voća manipulirajući DOM elementima, čime se smanjuje potreba za ponavljanjem istog kôda na više mjesta.

Primjer liste voća koristeći html i js

```
6      <div class="p-6 bg-white shadow-lg rounded-lg">
7          <h2 class="text-xl font-bold mb-4 text-center">
8              Lista voća
9          </h2>
10         <ul id="fruitList" class="space-y-2">
11             <!-- Elementi će biti dodani putem JavaScript-a -->
12         </ul>
13     </div>
14
15     <script>
16         const fruits = [
17             {
18                 name: 'Jabuka',
19                 icon: `<svg xmlns="http://www.w3.org/2000/svg" width="32"
20             },
21             {
22                 name: 'Banana',
23                 icon: `<svg xmlns="http://www.w3.org/2000/svg" width="32"
24             },
25             {
26                 name: 'Trešnja',
27                 icon: `<svg xmlns="http://www.w3.org/2000/svg" width="32"
28             },
29             {
30                 name: 'Kruška',
31                 icon: `<svg xmlns="http://www.w3.org/2000/svg" width="32"
32             },
33             {
34                 name: 'Breskva',
35                 icon: `<svg xmlns="http://www.w3.org/2000/svg" width="32"
36             },
37         ];
38
39         const fruitList = document.getElementById('fruitList');
40
41         fruits.forEach(fruit => {
42             const listItem = document.createElement('li');
43             listItem.classList = "p-2 bg-blue-500 text-white rounded-lg s
44             listItem.innerHTML = `
45                 ${fruit.icon}
46                 <span>${fruit.name}</span>
47             `;
48             fruitList.appendChild(listItem);
49         });
50     </script>
```

Kroz ovaj pristup uspjeli smo riješiti problem ponavljanja kôda, čime smo postigli veću efikasnost i održivost. Međutim, HTML kôd za dinamičko generiranje stavki sada više nije odvojen u zasebnom HTML dokumentu, već je smješten unutar skripte, što može povećati složenost održavanja i čitljivosti kôda, jer se logika za prikazivanje sadržaja miješa s logikom za upravljanje podacima.

Korištenjem Vue.js-a, možemo pojednostaviti strukturu aplikacije i odvojiti HTML od skripte.

Primjer liste voća koristeći Vue

```
1  <script setup>
2    import { Icon } from "@iconify/vue";
3    import { ref } from "vue"
4
5    const fruitList = ref([
6      {
7        name: "Jabuka",
8        icon: "noto:red-apple"
9      },
10     {
11       name: "Banana",
12       icon: "noto:banana"
13     },
14     {
15       name: "Trešnja",
16       icon: "noto:cherries"
17     },
18     {
19       name: "Kruška",
20       icon: "noto:pear"
21     },
22     {
23       name: "Breskva",
24       icon: "noto:peach"
25     }
26   ]);
27 </script>
28
29 <template>
30   <div class="p-6 bg-white shadow-lg rounded-lg">
31     <h2 class="text-xl font-bold mb-4 text-center">
32       Lista voća
33     </h2>
34     <ul class="space-y-2">
35       <li v-for="item in fruitList" class="p-2 bg-blue-500 text-white">
36         <Icon :icon="item.icon" width="32" height="32" />
37         <span>{{ item.name }}</span>
38       </li>
39     </ul>
40   </div>
41 </template>
```

Možemo primijetiti da smo, umjesto da HTML kôd smjestimo unutar skripte, sada integrirali JavaScript unutar HTML-a. Korištenjem Vue ugrađenih direktiva, uspjeli smo pojednostaviti i učiniti kôd čitljivijim, a u isto vrijeme sačuvali istu funkcionalnost kao i prije.

Sljedeći korak koji možemo poduzeti je pretvoriti pojedinačnu stavku voća u zasebnu komponentu.

Primjer liste voća koristeći Vue i komponentu

▼ ListItem.vue

```
1  <script setup>
2    import { Icon } from "@iconify/vue";
3    defineProps({
4      name: String,
5      icon: String,
6    })
7  </script>
8
9  <template>
10    <li class="p-2 bg-blue-500 text-white rounded-lg
11      shadow-md flex items-center space-x-2">
12      <Icon :icon="icon" width="32" height="32" />
13      <span>{{ name }}</span>
14    </li>
15  </template>
```

▼ App.vue

```
1  <script setup>
2    import ListItem from './components/ListItem.vue';
3    import { ref } from "vue"
4
5  > const fruitList = ref([ ...
26  ]);
27  </script>
28
29  <template>
30    <div class="p-6 bg-white shadow-lg rounded-lg">
31      <h2 class="text-xl font-bold mb-4 text-center">
32        Lista voća
33      </h2>
34      <ul class="space-y-2">
35        <ListItem v-for="item in fruitList"
36          :name="item.name" :icon="item.icon" />
37      </ul>
38    </div>
39  </template>
```

Na ovaj način izdvajamo kôd stavke u zasebnu Vue datoteku koju poslije možemo koristiti u drugim dijelovima aplikacije. Takve zasebne Vue datoteke nazivaju se Single-File Components.

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

Primjer liste voća i liste povrća koristeći Vue i komponente

```
1 <script setup>
2 import ListItem from './components/ListItem.vue';
3 import { ref } from "vue"
4
5 > const fruitList = ref([ ...
26 ]]);
27 > const vegetableList = ref([ ...
48 ]]);
49 </script>
50
51 <template>
52 <div class="p-6 bg-white shadow-lg rounded-lg">
53 <h2 class="text-xl font-bold mb-4 text-center">
54   Lista voća
55 </h2>
56 <ul class="space-y-2">
57 <ListItem v-for="item in fruitList"
58   :name="item.name" :icon="item.icon"/>
59 </ul>
60 </div>
61
62 <div class="p-6 bg-white shadow-lg rounded-lg">
63 <h2 class="text-xl font-bold mb-4 text-center">
64   Lista povrća
65 </h2>
66 <ul class="space-y-2">
67 <ListItem v-for="item in vegetableList"
68   :name="item.name" :icon="item.icon"/>
69 </ul>
70 </div>
71 </template>
```



Postavljanje aplikacije/projekta

U sljedećim koracima instalirat ćemo sve potrebne alate za razvoj Vue.js projekta.

Koraci za postavljanje projekta:

1. Instalacija uređivača kôda (Code Editor) Visual Studio Code

- Potrebno je preuzet i instalirat Visual Studio Code s code.visualstudio.com



- Alternativno preuzet i instalirat VSCodium s vscodium.com



- Visual Studio Code bez telemetrije
- manje ekstenzija

2. Instalacija Node.js

- Potrebno je preuzet i instalirat Node.js verziju `>=18.3` s nodejs.org
- Instalacija se provjerava pomoću naredbe:

```
npm -v
```

3. Kreiranje Vue projekta

- Vue projekt kreira se pomoću naredbe:

```
npm create vue@latest
```

4. Odabir naziva projekta

- Prilikom kreiranja projekta, potrebno je unijeti željeni naziv
- Taj naziv će odrediti ime mape u kojoj će se generirati svi potrebni resursi

```
> npm create vue@latest  
  
> npx  
> create-vue  
  
Vue.js - The Progressive JavaScript Framework  
? Project name: » primjer_vue_aplikacije
```

5. Podešavanje postavki projekta

- U ovom koraku nije potrebno birati dodatne opcije, pa za sve stavke odabiremo **No**
- Kasnije, kada budemo koristili Router i Piniu, za te opcije ćemo odabrati **Da**

```
✓ Project name: ... primjer_vue_aplikacije  
✓ Add TypeScript? ... No / Yes  
✓ Add JSX Support? ... No / Yes  
✓ Add Vue Router for Single Page Application development? ... No / Yes  
✓ Add Pinia for state management? ... No / Yes  
✓ Add Vitest for Unit Testing? ... No / Yes  
✓ Add an End-to-End Testing Solution? » No  
✓ Add ESLint for code quality? » No
```

6. Pokretanje projekta

- Ući u direktorij projekta

```
cd primjer_vue_aplikacije
```

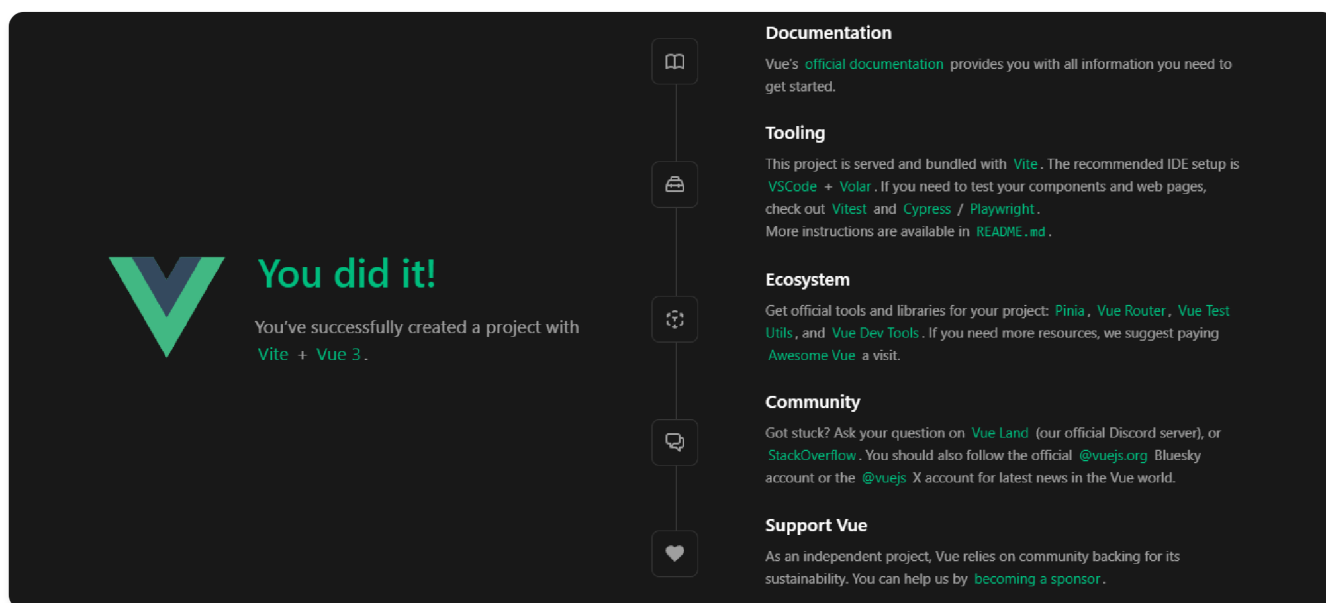
- Za instalaciju svih potrebnih paketa potrebno je pokrenuti sljedeću naredbu:

```
npm install
```

- Za pokretanje projekta u razvojnem načinu (*Hot-Reload*) s automatskim osvježavanjem koristi se naredba:

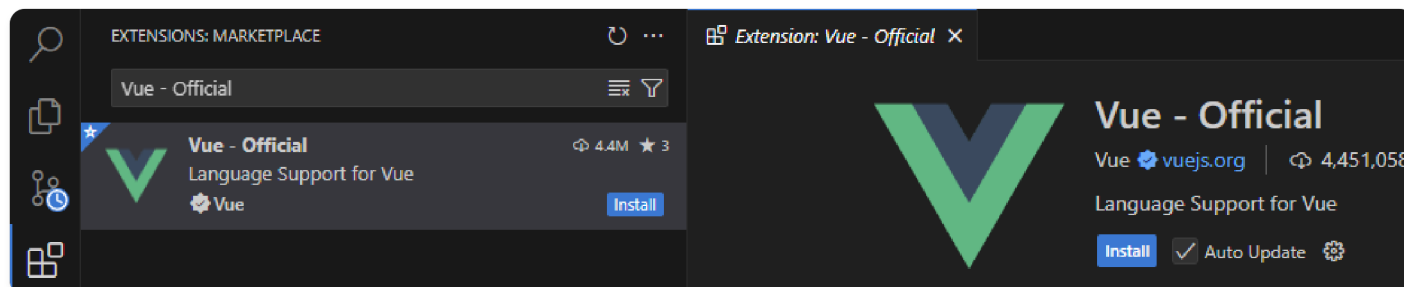
```
npm run dev
```

Početni izgled aplikacije:



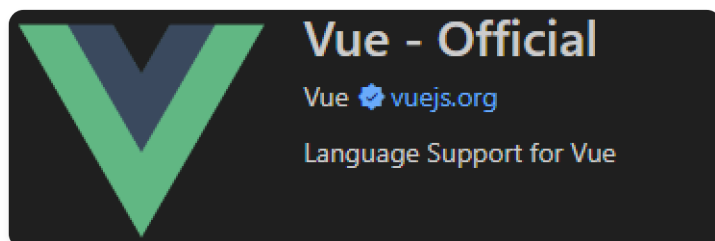
7. VS Code Ekstenzije

Za lakši rad s projektom potrebno je instalirati nekoliko ekstenzija. U VS Code-u potrebno je otvoriti *lijevi bočni izbornik*, kliknuti na *Extensions* (ili pritisnuti `Ctrl + Shift + X`), zatim upisati naziv željene ekstenzije u pretraživač, pronaći je i kliknuti *Install*.

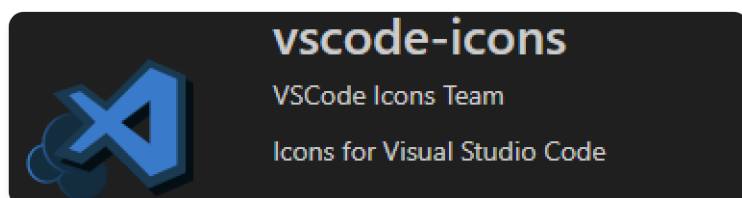


Preporučene ekstenzije:

- **Vue - Official** – podrška za Vue sintaksu



- **vscode-icons** – ikone datoteka i mapa

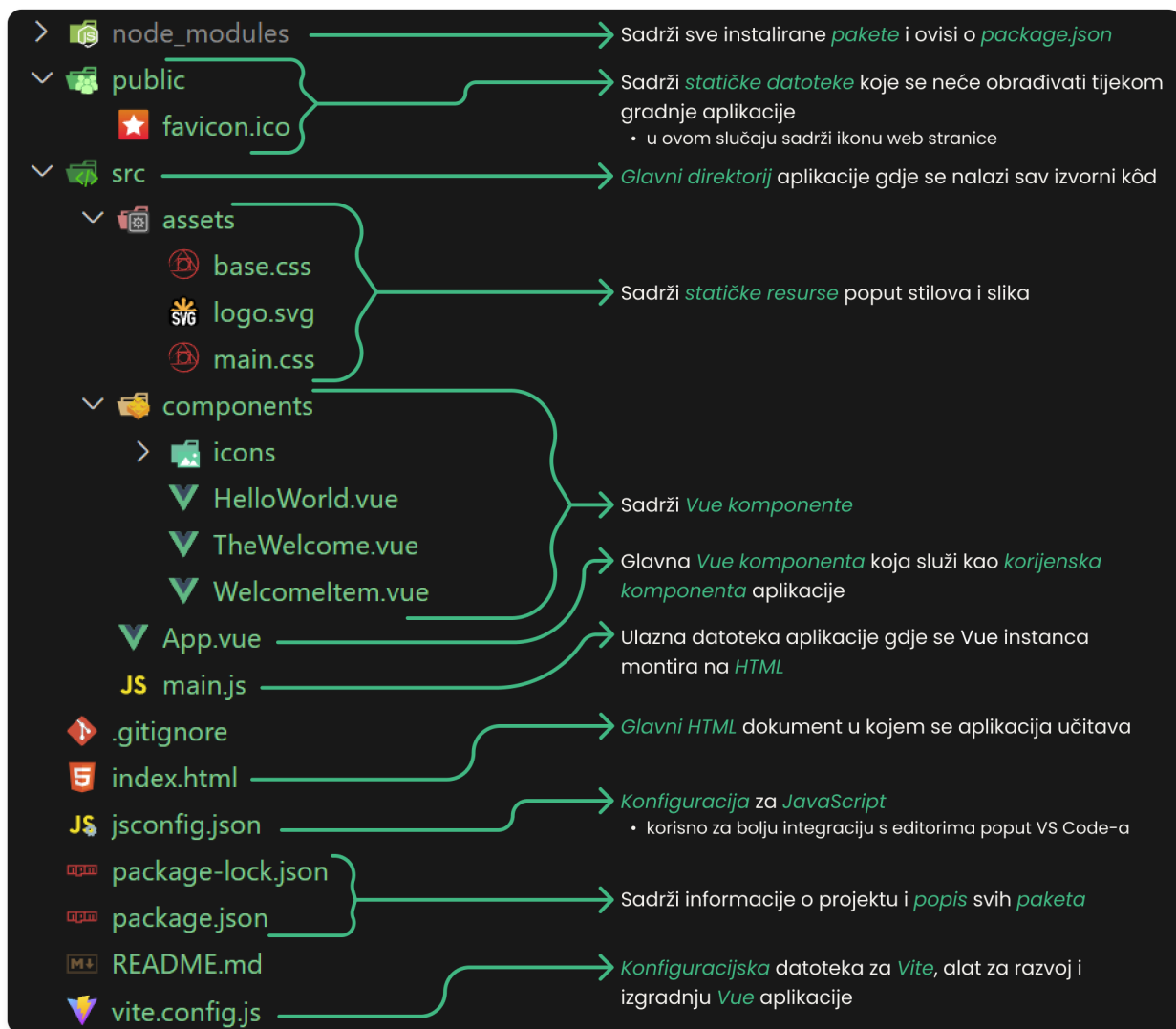


- Tailwind CSS IntelliSense



Struktura projekta

Struktura novokreiranog projekta izgleda kao što je prikazano na slici ispod:



Trenutno projekt sadrži unaprijed definirane datoteke i komponente koje služe kao primjer jednostavne aplikacije s linkovima na dokumentaciju i druge korisne materijale. Budući da nam za početak treba čisti kostur aplikacije, izvršit ćemo sljedeće korake unuta `src` mape:

1. Uklanjanje suvišnih datoteka i mapa

- Izbrisati sve datoteke i podmape unutar mape `components/`
- Izbrisati `base.css` i `logo.svg` iz mape `assets/`

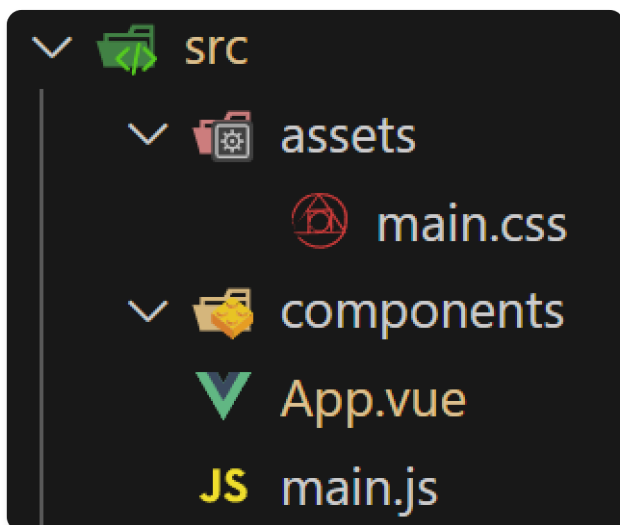
2. Čišćenje stilova

- Isprazniti sadržaj `main.css` kako bi bio potpuno prazan.

3. Priprema glavne komponente

- Očistiti `App.vue` tako da sadrži samo osnovnu strukturu s `script setup`, `template` i `style` blokovima

Izgled `src` mape nakon izvršenih koraka:



App.vue

`App.vue` je **glavna vue komponenta** koja se koristi za montira. Unutar nje ubacujemo sve druge komponente i rute na druge stranice aplikacije koje budemo kasnije radili pomoću `router` paketa.

Izgled `App.vue` nakon čišćenja:

```
<script setup>
  // JS - logika komponente
</script>

<template>
  <!-- HTML - struktura i sadržaj komponente -->
</template>

<style scoped>
  /* CSS - stil komponente */
</style>
```

Nakon što smo uklonili suvišne datoteke i očistili projekt, `src` mapa sada sadrži samo osnovne datoteke potrebne za daljnji rad. U ovom trenutku, stranica bi trebala biti potpuno prazna, tako da možemo u `App.vue` unutar `template` bloka napisati "Hello, world!".

```
<template>
  Hello, World!
</template>
```

Hello, World!

main.js

Main.js je glavna **JavaScript** datoteka unutar koje se inicijalizira i montira vue aplikacija.

Struktura main.js datoteke:

```
import './assets/main.css' // učitavanje glavnog stila aplikacije
                             // koji se primjenjuje nad cijelom aplikacijom

import { createApp } from 'vue' // učitavanje funkcije za kreiranje nove vue instance
import App from './App.vue'    // učitavanje komponente App.vue

createApp(App).mount('#app')    // kreiranje vue instance koristeći App.vue komponentu
                                // montiranje komponente na div s id-em #app
```

Ako pogledamo index.html datoteku:

```
<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="UTF-8">
    <link rel="icon" href="/favicon.ico">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vite App</title>
  </head>

  <body>
    <div id="app">
      <!--ovdje se montira App.vue-->
    </div>
    <script type="module" src="/src/main.js"></script>
  </body>
</html>
```

Vidimo da dokument sadrži div s id-jem #app, na koji Vue montira komponentu App.vue. Također, sadrži module script koji učitava main.js, čime se pokreće sama aplikacija.

Iako Vue omogućuje kreiranje i montiranje više instanci istovremeno, u praksi se obično koristi samo jedna. Također, ime glavne komponente i id elementa na koji se montira mogu biti proizvoljni, ali se pridržavamo standardne konvencije.

Primjer s dvije instance:

```
<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="UTF-8">
    <link rel="icon" href="/favicon.ico">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Vite App</title>
  </head>

  <body>
    <div id="app">
      <!--ovdje se montira App.vue-->
    </div>
    <div id="druga_app">
      <!--ovdje se montira MojApp.vue-->
    </div>
    <script type="module" src="/src/main.js"></script>
  </body>
</html>
```

```
import './assets/main.css'

import { createApp } from 'vue'
import App from './App.vue'
import MojApp from './MojApp.vue'

createApp(App).mount('#app')
createApp(MojApp).mount('#druga_app')
```

main.css

Datoteka `main.css` služi za primjenu globalnih stilova na cijelu aplikaciju ili za učitavanje vanjskih CSS okvira poput **Tailwind**-a.

Kako bismo osigurali dosljedan prikaz aplikacije i olakšali rad sa stilovima, dodajemo sljedeća CSS pravila:

```
html, body, #app {
  height: 100%;
  margin: 0;
}
```

Tailwind

Tailwind CSS je utility-first CSS okvir koji omogućuje brzo i efikasno stiliziranje web aplikacija. Umjesto unaprijed definiranih komponenti (kao u Bootstrapu), Tailwind koristi klase koje direktno primjenjuju pojedinačne stilove, što omogućuje veću fleksibilnost i prilagodljivost dizajna. Za razliku od tradicionalnih CSS frameworka (poput Bootstrapa), ne prisiljava korištenje unaprijed definiranih komponenti.



Ako želimo poboljšati "Hello, world!" primjer i učiniti ga vizualno zanimljivijim, možemo koristiti standardni `style` blok i **CSS** za stilizaciju.

Primjer koristeći standardni `style` blok i **CSS**:

```
<template>
  <div class="mojDiv">
    <span class="mojSpan">
      Hello, World!
    </span>
  </div>
</template>

<style scoped>
  .mojDiv {
    background: linear-gradient(135deg, oklch(0.257 0.09 281.288) 0%, oklch(0.13 0.028
261.692) 100%);
    height: 100%;
    display: flex;
    justify-content: center;
    align-items: center;
  }
  .mojSpan {
    transition: all 0.3s ease-in-out;
    color: oklch(0.967 0.003 264.542);
    font-family: monospace;
    font-size: 4rem;
    font-weight: bold;
  }
  .mojSpan:hover {
    color: oklch(0.789 0.154 211.53);
    scale: 125%;
    letter-spacing: 0.25rem;
  }
</style>
```


Hello, World!

Primjer koristeći **Tailwind**:

```
<template>
  <div class="bg-gradient-to-br from-indigo-950 to-gray-950 h-full flex justify-center items-center">
    <span class="transition-all duration-300 ease-in-out text-gray-100 font-mono text-6xl font-bold
      hover:text-cyan-400 hover:scale-125 hover:tracking-[0.25rem]">
      Hello, World!
    </span>
  </div>
</template>

<style scoped>
</style>
```

Možemo uočiti da smo u drugom primjer potpuno ispraznili style i pisali gotove dinamičke *Tailwind* **klase** direktno u `template`.

Prednosti Tailwind-a

Izravno pisanje u class atributu

Tailwind omogućuje izravno stiliziranje u `class` atributu bez potrebe za pisanjem zasebnog CSS-a. Umjesto definiranja zasebnih CSS pravila za svaki element, Tailwind koristi unaprijed definirane klase.

Ako pišemo CSS klasu za povećat veličinu fonta:

```
<p class="customFontSize">
  Some text
</p>
```

```
.customFontSize {
  font-size: 1.25rem /* (20px) */;
}
```

Možemo umjesto toga direktno u `class` pisat Tailwind klasu `text-xl`:

```
<p class="text-xl">
  Some text
</p>
```

Lakše održavanje kôda

Smanjuje potrebu za pisanjem dodatnih CSS datoteka i olakšava održavanje kôda. Nema potrebe za traženjem gdje su definirani stilovi za određeni element jer se svi stilovi nalaze unutar `class` atributa tog elementa. Budući da svaka Tailwind klasa primjenjuje samo jedno svojstvo, stilizacija je jasna, modularna i lako prilagodljiva.

Jednostavna responzivnost

Tailwind nudi jednostavan sustav za prilagodbu različitim veličinama ekrana pomoću prefiksa (`sm:`, `md:`, `lg:`, `xl:`). Nema potrebe za pisanjem posebnih `@media` upita – sve se rješava unutar `class` atributa

Koristeći tradicionalan media query:

```
<template>
  <div class="mojDiv">
    <h1>Prilagodljiv tekst</h1>
  </div>
</template>

<style scoped>
  .mojDiv {
    height: 100vh;
    width: 100%;
    display: flex;
    justify-content: center;
    align-items: center;
    background-color: #1f2937; /* bg-gray-800 */
  }

  h1 {
    color: white;
    font-weight: bold;
    font-size: 1.5rem; /* text-2xl */
  }

  @media (min-width: 640px) { /* sm */
    h1 {
      font-size: 1.875rem; /* text-3xl */
    }
  }

  @media (min-width: 768px) { /* md */
    h1 {
      font-size: 2.25rem; /* text-4xl */
    }
  }

  @media (min-width: 1024px) { /* lg */
    h1 {
      font-size: 3rem; /* text-5xl */
    }
  }

  @media (min-width: 1280px) { /* xl */
    h1 {
      font-size: 3.75rem; /* text-6xl */
    }
  }
</style>
```

Koristeći Tailwind responzivni dizajn:

```
<template>
  <div class="h-screen flex justify-center items-center bg-gray-800">
    <h1 class="text-white font-bold text-2xl sm:text-3xl md:text-4xl lg:text-5xl xl:text-6xl">
      Prilagodljiv tekst
    </h1>
  </div>
</template>
```

Konfiguracija

Tailwind pruža konfiguraciju, gdje možemo definirati boje, fontove i rasporede koji će se koristiti kroz cijelu aplikaciju. Time osigurava dosljedan vizualni stil u cijelom projektu.

Primjer prilagođene teme:

```
@import "tailwindcss";

@theme {
  --color-FIPU-blue: rgb(14, 165, 233);
}
```

```
<template>
  <h1 class="text-FIPU-blue">
    FIPU blue
  </h1>
</template>
```

Instalacija Tailwind-a

Tailwind ćemo integrirati i instalirati unutar našeg Vue projekta koristeći **Vite**, koji je već uključen prilikom kreiranja Vue aplikacije. Instalaciju Tailwinda možete i pratiti na njihovoj stranici:

<https://tailwindcss.com/docs/installation/using-vite>

Koraci za instalaciju:

1. Instalacija **Tailwind CSS**

- u `terminal` treba napisati sljedeću komandu

```
npm install tailwindcss @tailwindcss/vite
```

- instalacija se vrši pomoću naredbe `npm`, budući da zapravo instaliramo **npm paket**. U kasnijim predavanjima koristit ćemo isti postupak za instalaciju drugih paketa koje ćemo upotrebljavati u projektu.

2. Konfiguracija Vite plugina

- unutar Vite konfiguracijske datoteke `vite.config.js` dodajemo sljedeći kôd

```
import { fileURLToPath, URL } from 'node:url'

import { defineConfig } from 'vite'
import vue from '@vitejs/plugin-vue'
import vueDevTools from 'vite-plugin-vue-devtools'
import tailwindcss from '@tailwindcss/vite' // <- učitavanje tailwindcss paketa

export default defineConfig({
  plugins: [
    vue(),
    vueDevTools() // <- dodan plugin
  ],
  resolve: {
    alias: {
      '@': fileURLToPath(new URL('./src', import.meta.url))
    },
  },
})
```

3. Učitavanje Tailwind CSS-a

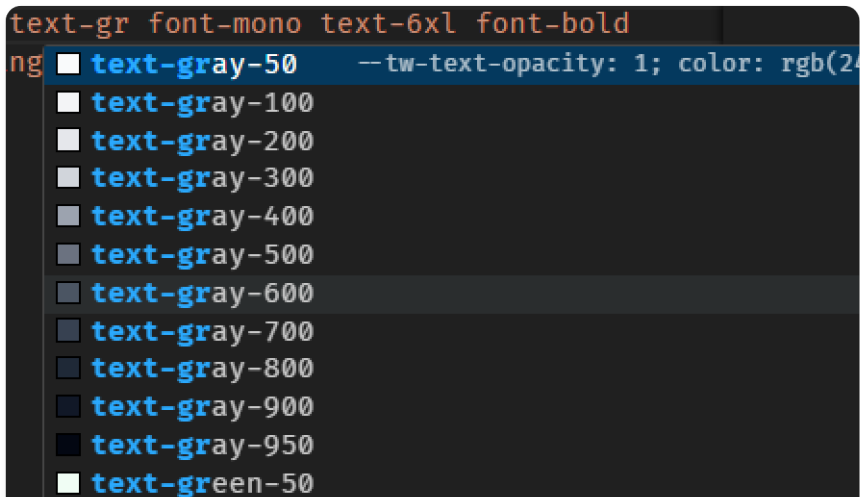
- unutar `main.css` datoteke dodajemo sljedeću liniju kôda čime uključujemo Tailwind CSS

```
@import "tailwindcss";
```

- sada možemo koristiti Tailwind klase

4. Fix za Tailwind CSS IntelliSense

- ako smo instalirali ekstenziju `Tailwind CSS IntelliSense` postoji velika šansa da neće raditi nakon prva 3 koraka
- da bi ekstenzija proradila potrebno je dodati praznu datoteku `tailwind.config.js` u projekt
- nakon dodavanje ove datoteke ekstenzija bi trebala raditi kako spada



Tailwind Osnove

U ovom djelu ćemo proći osnove Tailwind klasa kao što su *boje*, *text*, *veličine*, *razmaci*, *flex*, *stanja*, *responzivnost* i *teme*. Sva dokumentacija Tailwinda dostupna je na stranici:

<https://tailwindcss.com/docs/styling-with-utility-classes>

Boje

Tailwind dolazi s mnogo predefiniranih boja koje se mogu direktno koristiti za oblikovanje teksta, pozadina, gradijenata i obruba. Za korištenje boje, koristi se ime atributa, naziv boje i broj svjetline (manji broj označava svjetliju boju, veći broj tamniju).

Primjer korištenja boja:

```
<div class="
  text-lime-300
  bg-zinc-800
  border-sky-700
  border-2
">
  Some text
</div>
```

Some text

Lista svih boja:

	50	100	200	300	400	500	600	700	800	900	950
Red:											
Orange:											
Amber:											
Yellow:											
Lime:											
Green:											
Emerald:											
Teal:											
Cyan:											
Sky:											
Blue:											
Indigo:											
Violet:											
Purple:											
Fuchsia:											
Pink:											
Rose:											
Slate:											
Gray:											
Zinc:											
Neutral:											
Stone:											
B&W:											

U slučaju da želimo koristiti vlastitu boju, umjesto naziva boje i njene svjetline, pišemo **hex** vrijednost unutar uglatih zagrada, npr. `[#fff]`.

Primjer korištenja vlastitih boja:

```
<div class="
  text-[#bbf451]
  bg-[#27272a]
  border-[#0069a8]
  border-2
">
  Some text
</div>
```

Text

Najčešće korišteni podatak u bilo kojoj aplikaciji, koji čini većinu web stranice, jest tekst. Zato je važno da je pravilno oblikovan te da se jasno razlikuju različiti tipovi teksta, poput naslova, zaglavlja, tijela, sekcija itd.

Ključni stilovi koji se najčešće koriste za oblikovanje teksta su:

- **boja teksta** (*text color*)
- **veličina fonta** (*font size*)
- **stil fonta** (*font style*)
- **poravnanje teksta** (*text align*)
- **prelom teksta** (*text wrap*)

Boja teksta

- Za postavljanje boje teksta, koristi se ista metoda kao što je spomenuto u poglavlju [Boje](#), `text-(naziv boje)-(broj svjetline)` npr: `text-lime-300`

Veličina fonta

- Za veličinu fonta koriste se sljedeće klase:

Klasa	Vrijednost
<code>xs</code>	0.75 rem (12px)
<code>sm</code>	0.875rem (14px)
<code>base</code>	1 rem (16px)
<code>lg</code>	1.125rem (18px)
<code>x1</code>	1.25 rem (20px)
<code>2x1</code>	1.5 rem (24px)
<code>3x1</code>	1.875rem (30px)
<code>4x1</code>	2.25 rem (36px)
<code>5x1</code>	3 rem (48px)
<code>6x1</code>	3.75 rem (60px)
<code>7x1</code>	4.5 rem (72px)
<code>8x1</code>	6 rem (96px)
<code>9x1</code>	8 rem (128px)
<code>[custom]</code>	<code>[(number) px], [(number) rem]</code>

text-xs

text-sm

text-base

text-lg

text-xl

text-2xl

text-3xl

text-4xl

text-5xl

text-6xl

text-7xl

text-8xl

text-9xl

text-[26px]

Stil fonta

- Pod stil spadaju klase: *podcrtano*, *precrtano*, *kurziv* i *debljina fonta*:

underline

~~line-through~~

italic

font-bold

Poravnanje teksta

- Pod poravnanje teksta spadaju klase: *lijevo*, *centrirano*, *desno* i *obostrano*

Klasa	Opis
<code>left</code>	poravnava tekst uz lijevi rub elementa.
<code>center</code>	poravnava tekst u centar elementa
<code>right</code>	poravnava tekst uz desni rub elementa
<code>justify</code>	raspoređuje tekst po širini elementa

text-left

Diam et duo accusam
lorem sed et at tempor
est clita. Gubergren
invidunt at rebum eos et
est sanctus.

text-center

Diam et duo accusam
lorem sed et at tempor
est clita. Gubergren
invidunt at rebum eos et
est sanctus.

text-right

Diam et duo accusam
lorem sed et at tempor
est clita. Gubergren
invidunt at rebum eos et
est sanctus.

text-justify

Diam et duo accusam
lorem sed et at tempor
est clita. Gubergren
invidunt at rebum eos et
est sanctus.

Prelom teksta

- Ove klase nam kontroliraju kako se tekst prelomi unutar elementa:

Klasa	Opis
<code>wrap</code>	tekst se normalno prelomi unutar elementa
<code>nowrap</code>	tekst se ne prelama unutar elementa
<code>balance</code>	tekst se prelomi tako da se ravnomjerno rasporedi po linijama
<code>pretty</code>	tekst se prelomi tako da zadnja linija ne sadrži samo jednu riječ

text-wrap

Vero dolore sanctus est
diam sed

text-nowrap

Vero dolore sanctus est diam sed

text-balance

Vero dolore sanctus
est diam sed

text-pretty

Vero dolore sanctus est
diam sed

Veličine

Svi elementi imaju svoju visinu i širinu, koje ponekad želimo prilagoditi prema potrebi. Elementima zato možemo namjestiti njihovu:

- visinu (h) i širinu (w)
- maksimalnu visinu ($max-h$) i maksimalnu širinu ($max-w$)
- minimalnu visinu ($min-h$) i minimalnu širinu ($min-w$)

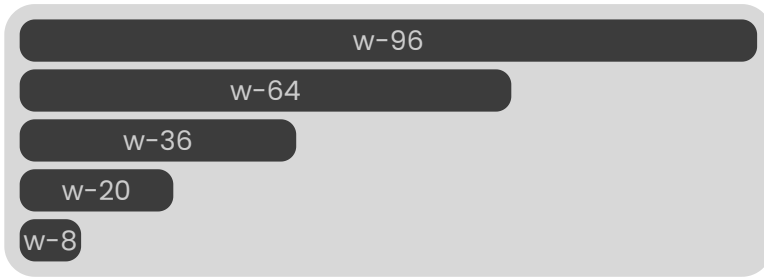
U Tailwindu možemo na više načina mjenjati veličinu elementa, koristeći:

- fiksne veličine: `w-<number>`, npr. `w-16 = 4 * 16 = 64px`
- omjere: `w-<fraction>`, npr. `w-1/4 = 25%`
- predefinirane klase: `w-screen`, `w-fit`, `w-full`
- prilagođene veličine: `w-[123px]`, `w-[55%]`

`width`, `height` i `size` dijele iste klase:

Klasa	Opis
<code>screen</code>	element zauzima veličinu prozora aplikacije
<code>fit</code>	element zauzima veličinu prema svom sadržaju (djece)
<code>full</code>	element zauzima punu veličinu dostupnog prostora (100% širine i visine)
<code>min</code>	element ima minimalnu veličinu, ovisno o sadržaju
<code>max</code>	element ima maksimalnu veličinu, ali može se smanjivati prema potrebama
<code>px</code>	element je veličine 1px (piksel)

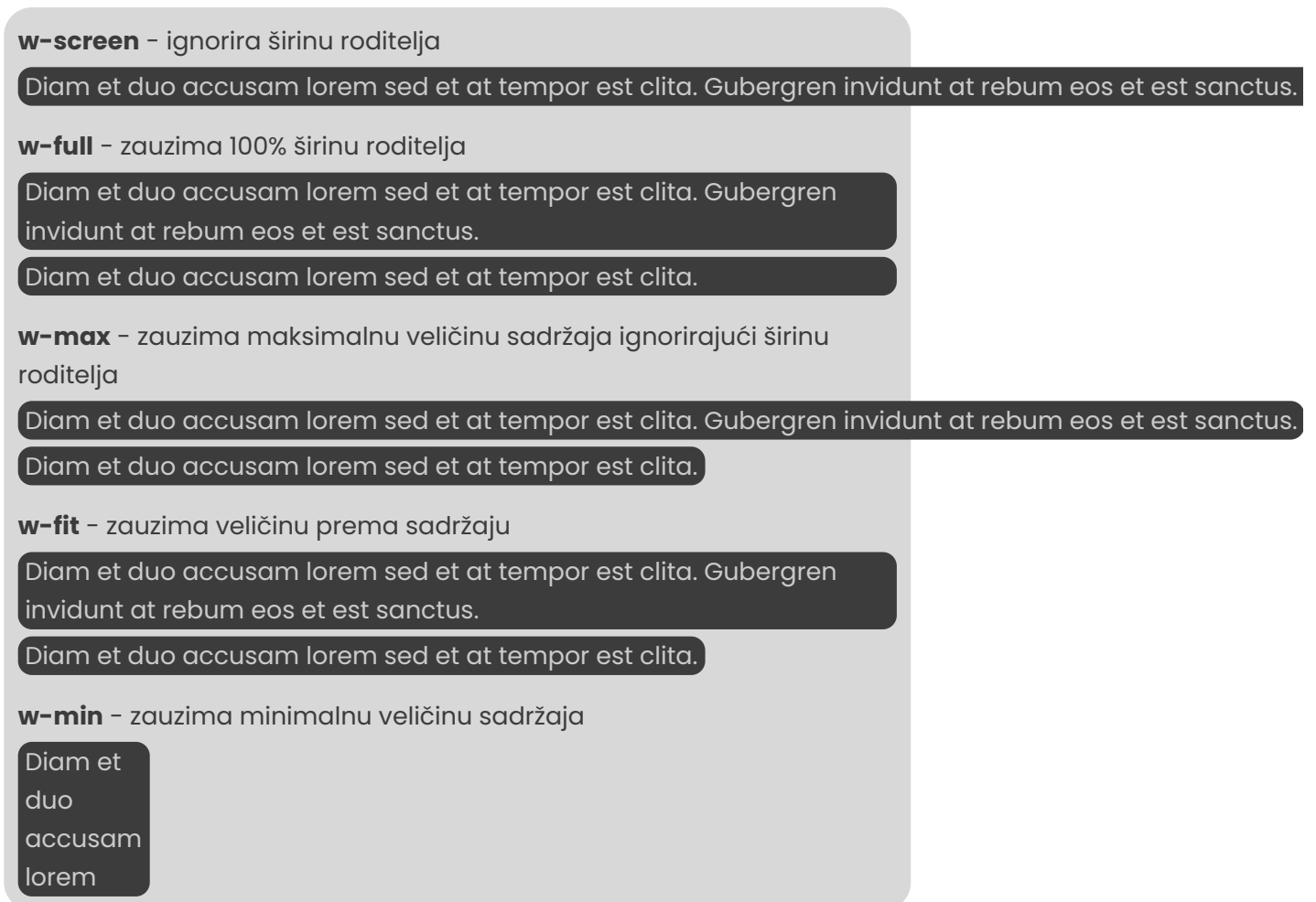
Primjer fiksne veličine `w-<veličina>` (veličina * 4px):



Primjer omjera `w-<omjer>`:



Primjer predefiniranih klasa:



Primjer prilagođenih veličina:

w-[255px]

w-[13%]

U slučaju da želimo primjeniti istu vrijednost visini i širini onda koristimo `size` klasu, npr. `size-24`

size-42

size-1/2

Razmaci

Kod stiliziranja elemenata često je potrebno definirati razmake između njih ili unutar njih. U Tailwindu, razmaci se mogu kontrolirati pomoću:

- **margin** (*m*) – vanjski razmak oko elementa
- **padding** (*p*) – unutarnji razmak unutar elementa

Razmaci se mogu definirati koristeći:

- Fiksne veličine: `m-<number>`, `p-<number>` (npr. `m-4`, `p-8` – `number * 4px`)
- Predefinirane klase: `m-auto`, `p-0`, `p-px`
- Prilagođene veličine: `m-[35px]`

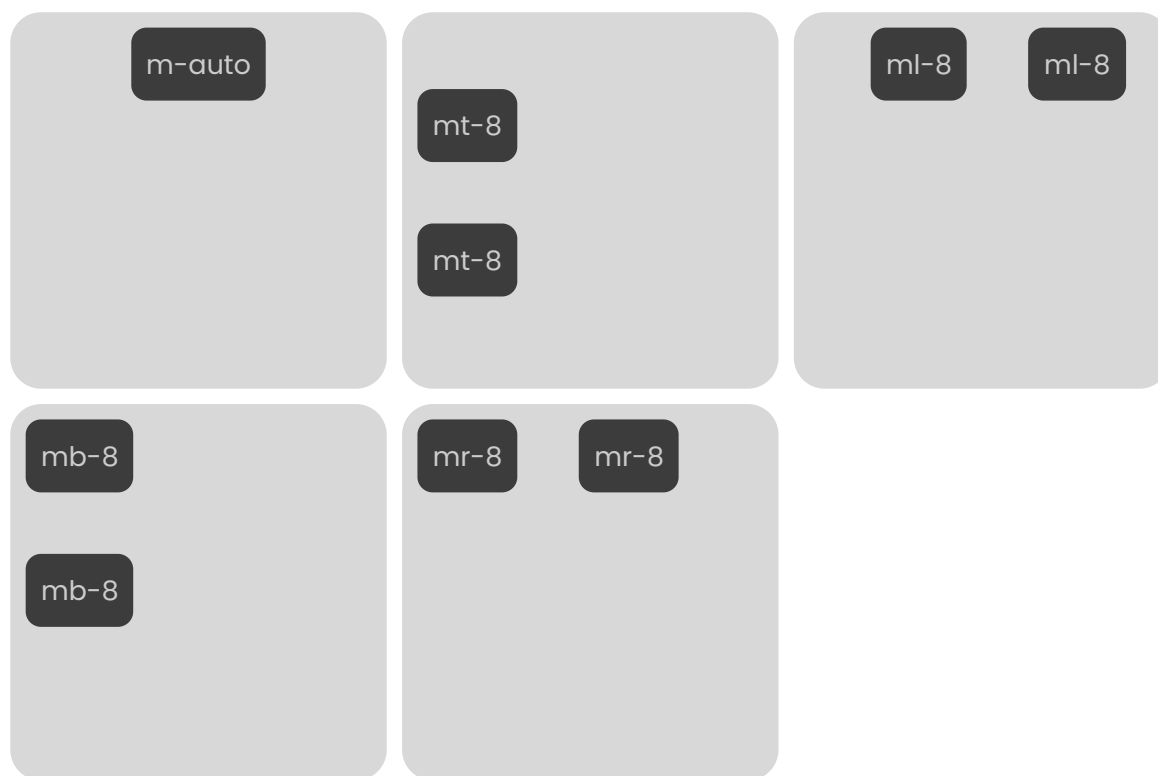
Primjena razmaka:

Klasa	Opis
<code>m-0 p-0</code>	uklanja vanjski/unutarnji razmak (0px)
<code>m-auto</code>	automatski centrirá element
<code>m-4 p-4</code>	postavlja vanjski/unutarnji razmak na 16px (4 * 4px)
<code>m-[20px]</code>	postavlja prilagođen vanjski/unutarnji razmak na 20px

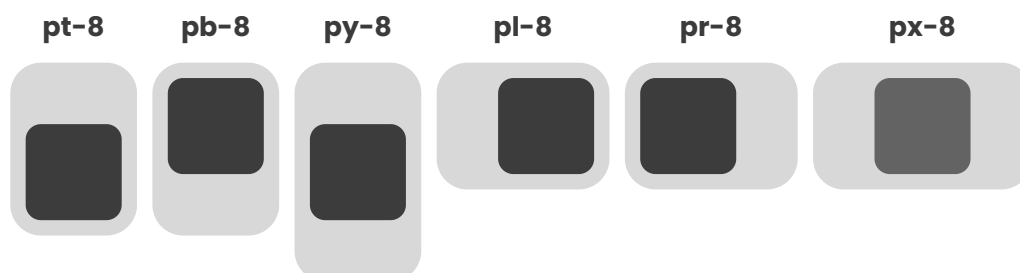
Razmaci mogu biti primijenjeni na specifične strane:

Klasa	Opis
<code>mt</code>	gore (<i>top</i>)
<code>ml</code>	lijevo (<i>left</i>)
<code>mb</code>	dolje (<i>bottom</i>)
<code>mr</code>	desno (<i>right</i>)
<code>my</code>	gore i dolje (<i>top & bottom</i>)
<code>mx</code>	lijevo i desno (<i>left & right</i>)

Primjer vanjskog razmaka:



Primjer unutarnjeg razmaka:



Flex

Flexbox je moćan alat za raspoređivanje i poravnavanje elemenata. U Tailwindu, fleksibilni raspored se kontrolira pomoću klase `flex`:

- `flex` – označava da je element fleksibilan
- `flex-row` – postavlja smjer elemenata u horizontalno (*default*)
- `flex-col` – postavlja smjer elemenata u vertikalno

Razmak između elemenata unutar roditelja s flex klasom se definira pomoću klase `gap`:

- `gap-<number>` = number * 4px
- npr. `gap-2`, `gap-5` (razmak od 8px, 20px)

*Horizontalno raspoređeni elementi s razmakom od 8px (2 * 4px):*

```
<!-- "flex-row" nije potreban, horizontalna os je osnovna zadana vrijednost -->
<div class="flex gap-2 bg-zinc-500/50 p-2 rounded-lg w-fit">
  <div class="bg-zinc-500/50 px-4 rounded-sm w-fit"> 1 </div>
  <div class="bg-zinc-500/50 px-4 rounded-sm w-fit"> 2 </div>
  <div class="bg-zinc-500/50 px-4 rounded-sm w-fit"> 3 </div>
</div>
```

1 2 3

*Vertikalno raspoređeni elementi s razmakom od 8px (2 * 4px):*

```
<div class="flex flex-col gap-2 bg-zinc-500/50 p-2 rounded-lg w-fit">
  <div class="bg-zinc-500/50 px-4 rounded-sm w-fit"> 1 </div>
  <div class="bg-zinc-500/50 px-4 rounded-sm w-fit"> 2 </div>
  <div class="bg-zinc-500/50 px-4 rounded-sm w-fit"> 3 </div>
</div>
```

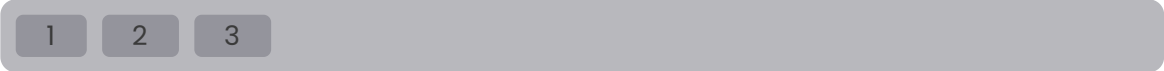
1

2

3

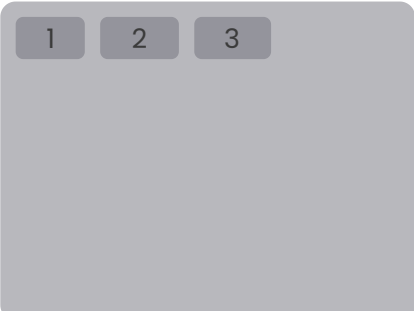
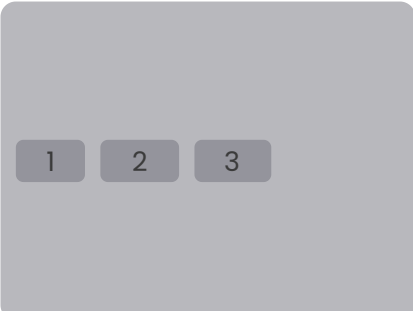
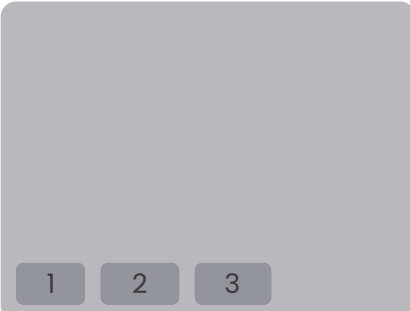
Poravnanje unutar flex elementa na glavnoj osi:

Klasa	Opis
<code>justify-start</code>	poravnava elemente na početak (lijevo za <code>flex-row</code> i gore za <code>flex-col</code>)
<code>justify-center</code>	poravnava elemente na centar
<code>justify-end</code>	poravnava elemente na kraj (desno za <code>flex-row</code> i dolje za <code>flex-col</code>)
<code>justify-between</code>	ravnomjerno raspoređuje elemente s maksimalnim razmakom između njih
<code>justify-around</code>	ravnomjerno raspoređuje elemente s jednakim prostorom oko svakog elementa
<code>justify-evenly</code>	ravnomjerno raspoređuje elemente s jednakim prostorom oko svakog elementa uzimajući u obzir zajednički prostor između dva elementa

justify-start	
justify-end	
justify-center	
justify-between	
justify-around	
justify-evenly	

Poravnanje unutar flex elementa na poprečnoj osi:

Klasa	Opis
<code>items-start</code>	poravnava elemente na početak poprečne osi
<code>items-center</code>	poravnava elemente na centar poprečne osi
<code>items-end</code>	poravnava elemente na kraj poprečne osi

items-start	items-center	items-end
		

Kada imamo previše elemenata unutar roditelja i oni počnu izlaziti izvan njegovih granica, možemo koristiti svojstvo `flex-wrap` kako bismo omogućili elementima da pređu u novi red umjesto da izlaze van roditelja.

`flex-nowrap`



`flex-wrap`



Kako centrirati div?

```
<div class="flex justify-center items-center">
  <div>
    Centriran div
  </div>
</div>
```



© Sarah Andersen

Samostalni zadatak za vježbu 1
