

SLEZSKÁ UNIVERZITA V OPAVĚ  
Filozoficko-přírodovědecká fakulta v Opavě

## BAKALÁŘSKÁ PRÁCE

SLEZSKÁ UNIVERZITA V OPAVĚ  
Filozoficko-přírodovědecká fakulta v Opavě

Lukáš Sukeník

Studijní program: Moderní informatika  
Specializace: Informační a komunikační technologie

**Porovnání SPA frontend frameworků**

**Comparison of SPA frontend frameworks**

Bakalářská práce

Opava 2024

Vedoucí bakalářské práce:  
doc. RNDr. Lucie Cíencialová, Ph.D.

*Kopie podkladu zadání práce  
z IS, podepsaná*

## **Abstrakt**

Text abstraktu v češtině. Rozsah by měl být 50 až 100 slov. Abstrakt není cíl práce, zde stručně popište, co čtenář má na následujících stránkách očekávat. Typické formulace: „V práci se zabýváme...“, „Tato bakalářská práce pojednává o...“, „součástí je“, „je provedena analýza“, „praktickou částí práce je aplikace xxx“ ... Prostě napište stručný souhrn či charakteristiku obsahu práce.

## **Klíčová slova**

Napište 5–8 klíčových slov v českém jazyce (v jednotném čísle, první pád atd.), měla by vystihovat téma práce. Slova odděľujte čárkou. Snažte se vystihnout nejdůležitější pojmy vystihující práci.

## **Abstract**

Anglická verze abstraktu by měla odpovídat české verzi, třebaže nemusí být úplně doslova. Když nutně potřebujete automatický překlad, použijte raději <https://www.deepl.com/cs/translator>, je lepší než Google Translator. Není nutno překládat doslova.

## **Keywords**

Anglická obdoba českého seznamu klíčových slov.

### **Čestné prohlášení**

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškerou literaturu a další zdroje, z nichž jsem při zpracování čerpal, v práci řádně cituji a jsou uvedeny v seznamu použité literatury.

V Opavě dne 9. listopadu 2023

.....  
Lukáš Sukeník

## **Poděkování**

Rád bych poděkoval za odborné vedení, rady a cenné poznatky k danému tématu vedoucímu práce ..... Také bych rád poděkoval mé rodině a přátelům za podporu a pomoc během mého studia.

# Obsah

Úvod	1
1 Webové aplikace	2
2 Analýza frameworků	3
2.1 React	3
2.1.1 Komponenty	4
2.1.2 JSX	4
2.1.3 Správa stavů	5
2.1.4 Hooky	5
2.1.5 Životní cyklus	6
2.1.6 State management	6
2.1.7 Routování	7
2.1.8 Ekosystém	7
2.2 Svelte	8
2.2.1 Komponenty	8
2.2.2 Reaktivita	9
2.2.3 Předávání vlastností	9
2.2.4 Eventy	10
2.2.5 Životní cyklus	11
2.2.6 State management	11
2.2.7 Routování	12
2.2.8 Ekosystém	13
2.3 Vue	13
2.3.1 Single-File Components	14
2.3.2 Reaktivita	14
2.3.3 Předávání vlastností	15
2.3.4 Direktivy a eventy	16
2.3.5 Životní cyklus	16
2.3.6 State management	17
2.3.7 Routování	17
2.3.8 Ekosystém	18
2.4 Porovnání	18
3 Testování frameworků	20

<b>4</b>	<b>Ukázková kapitola</b>	<b>21</b>
4.1	Struktura a formát . . . . .	21
4.1.1	Jak strukturovat práci . . . . .	21
4.2	Obrázky a tabulky . . . . .	21
4.2.1	Vkládání ukázkového kódu . . . . .	22
4.3	Vyznačování pojmů v textu . . . . .	23
4.4	Odrážky, číslování, pojmenované odstavce . . . . .	23
<b>5</b>	<b>Práce se zdroji</b>	<b>25</b>
5.1	Seznam použité literatury . . . . .	25
5.2	Citace . . . . .	25
5.3	Parafráze . . . . .	26
	<b>Závěr</b>	<b>28</b>
	<b>Seznam použité literatury</b>	<b>29</b>
	<b>Seznam obrázků</b>	<b>33</b>
	<b>Seznam tabulek</b>	<b>34</b>
	<b>Seznam zkratk</b>	<b>35</b>
	<b>Přílohy</b>	<b>36</b>



## 2 Analýza frameworků

Tato část práce se zabývá analýzou frameworků React, Svelte a Vue. Nejdříve je analyzován každý framework samostatně. Analyzovány jsou klíčové koncepty frameworků, stejně tak jako určité podobnosti těchto technologií. Kapitulu zakončuje teoretické srovnání frameworků.

Výběr frameworků byl proveden na základě analýzy výsledků z Developer Survey 2023 od Stack overflow. Rozhodující byla sekce *Web frameworks and technologies*, která hodnotila webové technologie dle jejich žádanosti na trhu a obdivem mezi vývojářskou komunitou. Tato strategie zajistila, že vybrané frameworky odpovídají současným trendům a potřebám webového vývoje, ale také vycházejí z uznání a podpory komunity.

- srovnání frameworků dle:
  1. DOM
  2. Kompilace zdrojových souborů

### 2.1 React

Pod pojmem React rozumíme open-source JavaScript framework, který vyvinula a dále vyvíjí společnost Meta (dříve Facebook). Podle [5] jde spíše o knihovnu funkcí, než-li o komplexní nástroj pro tvorbu webových aplikací (framework). Tato technologie se používá pro vývoj interaktivních uživatelských rozhraní a webových aplikací.[13]

První kořeny Reactu sahají až do roku 2010, kdy tehdejší společnost Facebook přidala novou technologii XHP do PHP. Jde o možnost znovu použít určitý blok kódu, stejného principu posléze využívá i React. Následně Jordan Walke vytvořil FaxJS, jenž byl prvním prototypem Reactu. O rok později byl přejmenován na React a začal jej využívat Facebook. V roce 2013 byl na konferenci JS ConfUS představen široké veřejnosti a stal se open-source.

Od roku 2014 vývojáři představují nespočet vylepšení samotné knihovny, stejně jako spoustu rozšíření pro zlepšení vývojových procesů. Kolem roku 2015 postupně React nabývá na popularitě i celkové stabilitě. Následně je představen

také React Native, což je framework pro vývoj nativních aplikací. V dnešní době je React využíván společnostmi každého rozsahu po celém světě. Z těch největších jde například o Metu, Uber, Twitter a Airbnb.[5, 12]

### 2.1.1 Komponenty

Hlavním stavebním kamenem Reactu jsou komponenty, jež představují nezávislé, vnořitelné a opakovaně použitelné bloky kódu. Komponentu v Reactu tvoří JavaScript funkce a HTML šablona. Validně seskládané komponenty poté tvoří webovou aplikaci. V Reactu se můžeme setkat s funkčními a třídními komponentami. Vytváření třídních komponent oficiální dokumentace nedoporučuje.

Pro komunikaci mezi komponentami se používá předávání vlastností (props), přes které je možné předávat hodnoty jakýchkoli datových typů. Výstup komponent tvoří elementy ve formě JSX. Tyto elementy obsahují informace o vzhledu a funkcionalitě dané komponenty.[5, 19]

```
import React from 'react';

function ParentComponent() {
  const someProps = {color: 'red'};

  return (
    <div>
      <ChildComponent color={'red'} />
      <ChildComponent color={someProps.color} />
      <ChildComponent {...someProps} />
    </div>
  );
}

function ChildComponent({color}) {
  return (
    <div class={color}></div>
  );
}
```

### 2.1.2 JSX

Název JSX kombinuje zkratku jazyka JavaScript – JS a počáteční písmeno ze zkratky XML. Konkrétně jde o syntaktické rozšíření, které vývojářům umožňuje tvořit React elementy pomocí hypertextového značkovacího jazyku přímo v JavaScriptu. V rámci JSX pak je možné dynamicky vykreslovat obsah na základě logiky definované po-

mocí JavaScriptových hodnot. Při kompilaci se JSX překládá do JavaScriptu pomocí nástroje Babel.[\[5, 19\]](#)

```
import React from 'react';
import ChildComponent from './ChildComponent';

function MyComponent() {
  const loaded = true;

  return (
    <div>
      {loaded ?
        <ChildComponent color={'red'} width={100} height={100} />
        : 'Loading ...'}
    </div>
  );
}
```

### 2.1.3 Správa stavů

Stav lze definovat jako lokální vnitřní vlastnost či proměnnou dané komponenty, jež představuje základní mechanismus pro uchovávání a aktualizaci dat. Pro aktualizaci komponenty je tedy nutné stav změnit. React pak na tuto skutečnost zareaguje a vyvolá tzv. re-render neboli překreslení komponenty s novými daty.

Za účelem ukládání stavu se využívá hook (funkce) `useState`. Ten poskytuje stavovou proměnnou, přes kterou se dostaneme k aktuálnímu stavu. Dále `useState` poskytuje state setter funkci, díky které můžeme stav aktualizovat. Jediný argument `useState` definuje počáteční hodnotu daného stavu.[\[16, 19\]](#)

```
import React, { useState } from 'react';

function App() {
  const [count, setCount] = useState(0);

  return (
    <button onClick={() => setCount(count + 1)}>
      You clicked this button {count} times.
    </button>
  );
}
```

### 2.1.4 Hooky

Specifickou funkcionalitou pro React jsou tzv. hooky, které byly do Reactu přidány až ve verzi 16.8.0.[\[20\]](#) Hook je definován jako funkce, která obohacuje komponenty

pomocí předdefinovaných funkcionalit. Jedním z nejpoužívanějších hooků je `useState`. Vývojáři mohou používat již zabudované hooky, nebo si vytvářet své vlastní s pomocí předdefinovaných hooků. Mezi zabudované hooky patří např. `useEffect`, `useMemo`, `useCallback`, `useRef`, `useContext`.<sup>[19]</sup>

```
import { useEffect } from 'react';

useEffect(() => {
  // code for setup

  return () => {
    // code for cleanup
  };
}, [
  // dependencies
]);
```

### 2.1.5 Životní cyklus

Životní cyklus komponenty je sekvence událostí, jež nastanou mezi vytvořením a zničením komponenty. Ve třídách komponent existovaly speciální metody, tzv. lifecycle metody, starající se o provedení určité části kódu při daném okamžiku v životě komponenty. Nyní React disponuje pár hooky, které umožňují provádět side-effects podobně jako lifecycle metody.

O momentu, kdy je komponenta přidána na obrazovku, mluvíme jako o namontování (mount) komponenty. Při změně stavu či obdržení nových parametrů hovoříme o aktualizaci (update) komponenty. A v neposlední řadě okamžik, kdy je komponenta odstraněna z obrazovky, nazýváme odmontování (unmount) komponenty.<sup>[17, 19]</sup>

### 2.1.6 State management

Základní práce se stavy spočívá v lokálních stavech komponent a následným předáváním stavu do potomků či rodičů. V případě, že potřebujeme sdílet stav mezi komponentami, měli bychom zvážit odlišné řešení. React sám o sobě disponuje pouze základním řešením, kterému říká Context API. Context umožňuje sdílet data celému podstromu dané komponenty. To se může hodit například při vytváření barevných módů aplikace, sdílení informace o přihlášeném uživateli, anebo routování.<sup>[19]</sup>

Správa stavů v komplexních aplikacích se stává výzvou. Problémy začínají při potřebě sdílení identických dat mezi větším množstvím konzumentů. Existuje však mnoho knihoven třetích stran, které vývojáři využívají pro usnadnění manipulace se stavy. Společné cíle state management knihoven spočívají v ukládání a získávání globálního stavu, jednodušší správě stavů a rozšiřitelnosti aplikace. Mezi tyto knihovny patří kupříkladu Redux, MobX, Recoil nebo Jotai.[2, 9]

### 2.1.7 Routování

React nemá žádný nativní standard pro routování. Podle [5] je React Router jedním z nejvíce populárních řešení pro React. Knihovna React Router umožňuje nastavení jednotlivých cest aplikace. Zajišťuje tedy routování na straně klienta. Klasické webové stránky při změně URL pokaždé žádají o nové dokumenty. Routování na straně klienta může provést aktualizaci stránky bez dalších duplikátních requestů. Při vyžádání dané cesty pak můžeme na stránce vykreslit požadovaný obsah a zažádat pouze o data potřebné pro vykreslení. Výstup činí rychlejší uživatelskou zkušenost, jelikož prohlížeč nevyžaduje nové dokumenty a nemusí vyhodnocovat kaskádové styly či JavaScript.[5, 21]

### 2.1.8 Ekosystém

Tato knihovna sama o sobě není úplně komplexním nástrojem. I přesto se stále vyvíjí a její ekosystém se neustále rozrůstá. Na druhou stranu pak existuje mnoho různých nástrojů třetích stran, funkcí, API, které mohou vývojáři při vývoji použít. Knihovny jsou velmi diverzifikované. Začíná to knihovnami zameřenými na stylování, či předpřipravenými komponentami pro uživatelské rozhraní. Samozřejmě najdeme i balíčky pro tvorbu tabulek, formulářů, grafů nebo grafických animací. Dále můžeme využít mnohých knihoven, které řeší správu stavů, routování, dotazování na API. Nechybí ani dokumentační knihovny, vývojářské rozšíření pro prohlížeče, striktní typování, překlady, testovací balíčky. V neposlední řadě pro React existují nadstavby ve formě frameworků, které pak poskytují lepší základy pro produkční aplikace.[3, 11, 19]

## 2.2 Svelte

Svelte je relativně novým open-source JavaScript frameworkem, za jejímž stvořením stojí vývojář Richard Harris. Framework kompiluje komponenty přímo do čistého nativního a vysoce optimalizovaného JavaScriptu bez potřeby runtime. To vše ještě před tím, než uživatel navštíví webovou aplikaci v prohlížeči. Tato metoda poskytuje výhodu hlavně co se týče rychlosti oproti klasickým deklarativním frameworkům jako jsou např. React, Vue nebo Angular. Stejně jako tyto frameworky je Svelte určen k vývoji rychlého a kompaktního uživatelského rozhraní pro webové aplikace.

První verze byla představena ke konci roku 2016. Verze 3, jež byla vydána v dubnu 2019, přinesla vylepšení týkající se zjednodušení tvorby komponent. Mimo jiné tato verze hlavně představila vylepšení ve smyslu reaktivity. Po této verzi framework nabral na popularitě díky jeho jednoduchosti. Verze 4 pak v roce 2023 představila pouze minimální změny, jež spočívají v údržbě a přípravách pro verzi nastávající.

Přestože Svelte nedisponuje rozsáhlým ekosystémem jako jiné JavaScriptové frameworky, získal si přízeň mnoha velkých společností. Mezi ně patří například firmy jako The New York Times, Avast, Rakuten a Razorpay.[10, 22, 26]

### 2.2.1 Komponenty

Podobně jako v Reactu, komponenty jsou základní stavební bloky Svelte. Komponentu tvoří HTML, CSS a JavaScript, kde vše patří do jednoho souboru s příponou .svelte. Všechny tři části komponenty jsou nepovinné. Logika komponenty musí být zapsána mezi párové script tagy. Následuje jedna nebo více značek pro definování šablony komponenty. V neposlední řadě kaskádové styly se zapisují mezi style tagy.

V rámci šablony Svelte umožňuje využívat logické bloky pro podmíněné vykreslování nebo také iterace přes pole hodnot (list). Zabudovaná je i podpora manipulace s asynchronním JavaScriptem - promises.[22]

```
<script>
let content = 'some-content';
</script>

<div>{content}</div>

<style>
```

```
div {
background-color: red;
}
</style>
```

### 2.2.2 Reaktivita

Srdcem Svelte jsou reaktivní stavy komponenty, které jednoduše definujeme jako proměnné v JavaScriptu. Jejich hodnotu aktualizuje JavaScript funkce pomocí přidělování nových hodnot. Kupříkladu stav o datovém typu pole tudíž nelze aktualizovat pouze pomocí metody push či splice. Je nutné využít jiné intuitivní řešení pomocí přidělení nové hodnoty. O všechno ostatní se pak ale postará sám Svelte v pozadí. Svelte aktualizuje DOM při každé změně stavu komponenty.

Mezi specifické funkce Svelte patří reaktivní deklarace, které se starají o aktualizaci stavů na základě stavů jiných. Další zabudovanou funkcí jsou tzv. reactive statements, jež umožní definovat akce, které se mají vykonat reaktivně – jako reakce na nějaký výrok.[\[7, 22\]](#)

```
<script>
  let count = 0;

  function increment()
  count++;

</script>

<button on:click=increment>
  You clicked this button count times.
</button>
```

### 2.2.3 Předávání vlastností

Pro komunikaci mezi komponentami slouží mechanismus předávání vlastností. V rodičovské komponentě je nutné komponentě říci, jakou hodnotu chceme předat a do jaké proměnné ji chceme uložit v child komponentě. Pak v child komponentě vytvoříme stejnojmennou vlastnost s klíčovým slovem export.

Pokud chceme předávat vlastnosti parent komponentě, je třeba vytvořit vlastnost již na parent komponentě. Následně ji předat child komponentě a v rodičovské komponentě přidat před předání vlastnosti do komponenty klíčové slovo bind.[\[22\]](#)

```
// Parent.svelte
<script>
```

```
import ChildComponent from './Child.svelte';

const someProps = {color: 'red'};
</script>

<ChildComponent color='red' />
<ChildComponent color={someProps.color} />
<ChildComponent {...someProps} />

// Child.svelte
<script>
export let color;
</script>

<div class={color}></div>
```

### 2.2.4 Eventy

Svelte má velice jednoduché API pro práci s DOM eventy. Stačí použít direktivu `on` na HTML elementu, která vyžaduje název eventu a callback funkci.

```
<script>
let count = 0
</script>

<button on:click={() => count++}>
  You clicked this button {count} times.
</button>
```

Vývojáři také přišli s možností odesílání a přijímání eventů pro komponenty. V child komponentě je třeba mít nějaký DOM event handler, na který chceme reagovat v parent komponentě. Poté je nutné využít zabudovanou metodu `createEventDispatcher`, které předáme potřebné parametry – náš libovolný název pro event komponenty a hodnotu. V rodičovské komponentě pak reagujeme na event pomocí klíčového slova `on` a našeho libovolného názvu pro event. Naši hodnotu poté získáme v callback funkci.[\[7, 22\]](#)

```
// Parent.svelte
<script>
import Child from './Child.svelte';

function handleMessage(event)
alert(event.detail.text);

</script>

<Child on:message=handleMessage />

// Child.svelte
```



```

<script>
import createEventDispatcher from 'svelte';

const dispatch = createEventDispatcher();

function sayHello()
dispatch('message',
text: 'Hello!'
);

</script>

<button on:click=sayHello>Click to say hello</button>

```

Zdroj zdrojového kódu: [22]

### 2.2.5 Životní cyklus

Komponenty ve Svelte disponují životním cyklem, který začíná v momentě vytvoření komponenty a končí jejím zničením. Funkce `onMount` tvoří callback, který je zavolán po přidání komponenty do DOMu. Pokud chceme vykonat určité akce při zničení komponenty, můžeme toho dosáhnout dvěma způsoby. Prvním způsobem je vrácení callback funkce v rámci `onMount` funkce. Druhou možností představuje využití funkce `onDestroy`, která v argumentu přijímá callback funkci.

Pro práci převážně s imperativními akcemi slouží zabudované funkce `beforeUpdate` a `afterUpdate`. V případě `beforeUpdate` funkce jde o callback, který se volá před aktualizací komponenty, tj. před prvním voláním `onMount` nebo po každé změně stavu. Oproti tomu, `afterUpdate` je callbackem, jenž Svelte vykoná po prvním zavolání `onMount` nebo po každé aktualizaci komponenty.[7, 22]

### 2.2.6 State management

Svelte poskytuje pestrou škálu API pro správu stavů aplikace v závislosti na rozsahu a složitosti ukládaných dat. Základním přístupem pro správu stavů je ukládání a manipulace se stavy v rámci stromu komponent. To zahrnuje tvorbu reaktivních stavů a jejich distribuci ve stromě pomocí předávání vlastností, bindování či eventů.

Další možností state managementu představuje využití Context API, které umožňuje jednorázové uložení jakékoli hodnoty. Nasledně je možné získat tuto hodnotu i v rámci neincidentních komponent. Ukládání a získávání contextu umožňují funkce `setContext` a `getContext`.

Pro sofistikovanější práci se stavy slouží tzv. stores. V podstatě se jedná o globální úložiště stavů, které umožňuje uchovávat a získávat data. Store je jednoduše objekt s metodou `subscribe`, která umožní konzumentu dostat aktualizovaná data. Jednodušší variantu pro získání aktuálních dat představuje použití znaku `$` před názvem proměnné. Svelte nám poskytuje hned několik podob storu. To jsou jednak `writable` a `readable` stores, kde jediný rozdíl spočívá v možnosti aktualizace dat. Pro stavy, které jsou odvozeny z jiných stores, existuje tzv. `derived` store. V neposlední řadě nám Svelte povoluje vytvořit i vlastní store.

Již zabudované globální úložiště můžeme jednoduše vytvořit pomocí metod `writable`, `readable` a `derived`. `Writable` požaduje jako argument počáteční hodnotu. `Readable` navíc jako druhý argument může přijímat funkci `start`, jež implementuje callbacky volající se při prvním a posledním `subscribe`.[\[7, 22, 25\]](#)

### 2.2.7 Routování

Svelte nemá přímou podporu routování v aplikacích. Oficiální dokumentace uvádí jako oficiální knihovnu pro routování `SvelteKit`. Ve skutečnosti se jedná o framework nad Svelte, který poskytuje i další možnosti rozšíření webové aplikace. Dokumentace však doporučuje i jiné knihovny pro routování na základě odlišných přístupů. Konkrétně knihovny `page.js`, `svelte-routing`, `svelte-navigator`, `svelte-spa-router` nebo `routify`.[\[22, 24\]](#)

Routování ve `SvelteKit` je implementováno pomocí file systému. Komponenta s názvem `+page.svelte` definuje stránku aplikace. Framework umožňuje pro opakující se uživatelská prostředí využít tzv. `layouts`. Jde o soubor, který aplikuje určité elementy (duplicitní kód) pro aktuální adresář komponent. Pro vykreslení obsahu na základě samotných komponent se využívá `element slot`. `SvelteKit` také umožňuje vytváření dynamických parametrů přímo v souborovém systému. Díky takovým cestám je možné tvořit např. individuální příspěvky na blogu. Pomocí `+server.js` můžeme definovat API routy (endpoints) aplikace. Chybové stránky vytváříme pomocí `+error.svelte` souborů.[\[22, 23\]](#)

### 2.2.8 Ekosystém

I přesto, že Svelte používá stále více vývojářů, framework nedisponuje příliš rozsáhlým ekosystémem. Hlavní rozšíření spočívá v použití rozšiřujícího frameworku SvelteKit a jazyka TypeScript. Vztah Svelte a SvelteKit můžeme definovat jako sourozenecký, kdy SvelteKit poskytuje adaptivní prostředí pro vývoj aplikace jakéhokoli rozsahu.

Dle [14] neexistuje mnoho specifických knihoven přímo pro Svelte. Na druhou stranu je možné využít rozsáhlého ekosystému JavaScriptu, jelikož Svelte poskytuje přímou kontrolu nad DOM. V porovnání se specifickými knihovnami tento přístup však obvykle vyžaduje práci navíc. Problematické bývá využití knihoven, jež využívají API prohlížeče.[8, 14, 27]

## 2.3 Vue

Vue dostalo svůj název díky anglickému slovu view. Jedná se o deklarativní JavaScriptový open-source framework. Je určen efektivní tvorbě jak jednoduchých, tak i komplexních uživatelských rozhraní na webu. Framework je v současné době jedním z nejpobulárnějších frameworků pro tvorbu webových aplikací.[15, 28]

Evan You, tvůrce Vue.js, se inspiroval určitými částmi frameworku AngularJS, který však měl velmi strmou křivku učení. Vue tedy mělo být lehké, přizpůsobivé a snadné k naučení. Bylo vytvořeno roku 2013, uvolněno do světa až o rok později. Od té doby byly vydány pouze 3 majoritní verze, avšak ty přinesly mnoho změn.[1, 29]

Vue nabízí svobodnou volbu při tvorbě komponent ve formě dvou hlavních API – Options a Composition API. Options API můžeme přirovnat k objektovému přístupu, v porovnání s Composition API, jež využívá funkcionální přístup. Podle [28] Composition API přináší větší flexibilitu a umožňuje efektivnější návrhové vzory pro organizaci a znovupoužitelnost kódu. Analýza bude probíhat pomocí Composition API.

Framework klade důraz na progresivitu, což znamená, že roste s vývojářem a přizpůsobuje se jeho potřebám. Díky tomu si Vue oblíbily společnosti jako Xiaomi, Adobe, Gitlab, Trivago, BMW.[6, 28]

### 2.3.1 Single-File Components

Základní funkcí Vue jsou tzv. Single-File Components (SFC). Jedná se o hlavní stavební blok frameworku, který reprezentuje část webové stránky. Komponenta se skládá z šablony, dat komponenty, funkcí a kaskádových stylů. Hlavní výhodou tohoto přístupu představuje znovupoužitelnost. JavaScriptové funkce musí být zapísány mezi párové značky script s atributem setup, šablona do template tagů a styly do style bloku.[15, 28]

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

  const content = ref('some-content');
</script>

<template>
  <div>{{ content }}</div>
</template>

<style scoped>
  div {
    background-color: red;
  }
</style>
```

### 2.3.2 Reaktivita

V komponentě můžeme uchovávat informace pomocí reaktivních stavů. Oficiální dokumentace doporučuje používat funkci ref, která vyžaduje počáteční hodnotu. K hodnotě stavu pak v rámci skriptu přistupujeme pomocí klíčového slova value. V šabloně nám stačí pouze název stavu. Modifikaci stavu lze provést pomocí přiřazení nové hodnoty. Při změně jakéhokoli stavu komponenty pak Vue automaticky aktualizuje DOM s novými daty.[28]

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

  const count = ref(0);

  function increment() {
    count.value++;
  }
</script>

<template>
  <button v-on:click="increment">
    You clicked this button {{ count }} times.
```

```

    </button>
  </template>

```

### 2.3.3 Předávání vlastností

Komponenty spolu komunikují pomocí předávání vlastností. V parent komponentě je třeba předat požadovanou hodnotu do proměnné child komponenty. V child komponentě pak definujeme props vlastnost, kterou vytvoříme pomocí funkce `defineProps`. `DefineProps` funkce vyžaduje objekt s názvem a datovým typem předávané vlastnosti.

Pro předání vlastnosti z child to parent komponenty se využívá tzv. emitování eventů. V potomku vytvoříme vlastnost `emit`, v níž nadeklarujeme pole emitovaných hodnot pomocí funkce `defineEmits`. Dále je třeba definovat jednotlivé emity. První argumentem je název emitu, další argumenty jsou již předávané hodnoty. Rodičovská komponenta musí naslouchat na emitované eventy. Toho lze docílit pomocí `@response` direktivy, která typicky v callback funkci přeukládá argumenty na lokální stavy.[\[15, 28\]](#)

```

// Parent.vue
<script setup>
  import { ref } from 'vue';
  import ChildComponent from './Child.vue';

  const someProps = ref({color: 'red'});
</script>

<template>
  <ChildComponent :color="'red'" />
  <ChildComponent :color="someProps.color" />
</template>

// Child.vue
<script setup>
  const props = defineProps({
    color: String
  });
</script>

<template>
  <div :class="color"></div>
</template>

```

### 2.3.4 Direktivy a eventy

Framework disponuje mnoha direktivami, které umožňují přidávat do šablony různé funkce. Logiku vykreslování umožňují direktivy `v-if`, `v-else-if` a `v-else`. Pro iteraci přes pole hodnot slouží `v-for`. Mezi další užitečné direktivy patří `v-bind` a `v-model`. Díky `v-bind` je možné přidat jakémukoli elementu dynamickou hodnotu atributu, direktivu můžeme zkrátit i pomocí dvojtečky. Direktiva `v-model` zase zajistí obousměrné propojení pro formulářové prvky.

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

  const text = ref('');
</script>

<template>
  <input v-model="text" placeholder="Write something">
  <p v-if="text.length > 3">{{ text }}</p>
  <p v-else>You have to enter more than 3 chars</p>
</template>
```

Vue také umožňuje naslouchat na DOM eventy pomocí direktivy `v-on`. Ta pak vyžaduje libovolný DOM event a callback funkci, jež se vykoná při daném DOM eventu. Můžeme také zvolit ekvivalentní zápis s `@`. Další možnost představuje využití modifikátorů eventů, které se postarají například o vypnutí výchozího chování prvku.[\[15, 28\]](#)

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

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

<template>
  <button @click="() => count++">
    You clicked this button {{ count }} times.
  </button>
</template>
```

### 2.3.5 Životní cyklus

Každá instance Vue komponenty má daný životní cyklus. Ten můžeme rozdělit na 3 části – inicializační část, část, při které se mění data a část, kdy komponenta zaniká. Pro zajištění kontroly životního cyklu slouží tzv. hooks, které se vyznačují tím, že vždy přes svým názvem mají předponu `on`.

Při inicializaci komponenty můžeme využít hook akce `beforeMount` či `mount`. `BeforeMount` se volá ještě před tím, než se komponenta přidá na stránku. `Mount` až poté, kdy je vytvořen element komponenty – komponenta však ještě nemusí být v DOM. Před chystanou změnou dat v DOM se volá `beforeUpdate`, po vykonání změny je zavoláno `updated`. Před samotným zánikem komponenty Vue volá `beforeUnmount`. Po dokončení zničení se pak volá `unmounted`.[\[15, 28\]](#)

### 2.3.6 State management

Framework Vue v sobě nemá implementovaný žádný sofistikovaný state management. K základní práci se stavy aplikace poslouží reaktivní stavy jednotlivých komponent a jejich sdílení ve stromě komponent.

Pro komplexnější aplikace je třeba využít některou z knihoven třetích stran. Dokumentace doporučuje knihovnu Pinia, o kterou se starají vývojáři samotného Vue. Pinia je inspirována Vuex a využívá principu Composition API. Tato knihovna nabízí jednoduché API pro správu stavů v aplikaci. Hlavním prvkem jsou tzv. stores, do kterých ukládáme globální stavy aplikace. Store vytvoříme pomocí funkce `defineStore`, která požaduje identifikátor daného store a callback funkci. Ta by měla vracet samotný state definovaný pomocí `ref()` a akce (funkce, jež mění stavy) nad storem, případně computed state – jiný stav, který je typicky odvozen od původního stavu.[\[18, 28\]](#)

### 2.3.7 Routování

Samotný Vue framework neposkytuje zabudovanou podporu pro routování. Oficiální dokumentace doporučuje využití knihovny Vue Router, která umožňuje routování na straně klienta. Základní využití routeru spočívá ve vytvoření instance routeru. Tuto instanci inicializujeme polem požadovaných cest aplikace s konkrétními komponentami pro vykreslení. V rámci šablony pak můžeme použít `router-link` element, jež představuje odkaz na jinou cestu. Pro vykreslení obsahu po změně cesty slouží `router-view`. V podstatě se jedná o wrapper pro vykreslení komponenty, která je spojena s uživatelem vybranou cestou.

Vue Router disponuje také dynamickým routingem, který umožňuje definovat parametry cesty a vykreslit tedy stránku s dynamickými daty. Mezi další funkce patří např. zanořené routování a routování dle pojmenovaných cest aplikace.[28, 30]

### 2.3.8 Ekosystém

Vue framework sice nabízí solidní základ pro vývoj webových aplikací, sám o sobě ale není komplexním nástrojem pro vývoj aplikací většího rozsahu. Většina vývojářů využívá kromě frameworku i další knihovny třetích stran, které zvyšují produktivitu a zjednodušují vývoj požadovaných funkcí. V rámci ekosystému lze využít například knihovny pro práci s UI komponenty, routováním, state managementem. Nechybí ani knihovny pro typování, testování, statické generování či formátování kódu.

Při rostoucí komplexitě webových aplikací můžeme zvážit použití pokročilého frameworku Nuxt. Nuxt je postaven na základech Vue.js a poskytuje production-ready nástroje. Konkrétně řeší například pokročilou správu stavů, routing, hydrataci stránek či Server Side Rendering.[4, 28]

## 2.4 Porovnání

- co zjistím na první pohled, platforma,
- jako bych si četl reklamu ...,
- prvotní srovnání.



## Seznam obrázků

1	Ukázka vložení titulku s označením zdroje . . . . .	22
---	---	----

## Seznam tabulek

1	Ukázka tabulky . . . . .	22
---	--------------------------	----

# PŘÍLOHY

Do tohoto seznamu napište přílohy vložené přímo do této práce a také seznam elektronických příloh, které se vkládají přímo do archivu závěrečné práce v informačním systému zároveň se souborem závěrečné práce. Elektronickými přílohami mohou být například soubory zdrojového kódu aplikace či webových stránek, předpřipravený produkt (spustitelný soubor, kontejner apod.), vytvořená metodická příručka, tutoriál... (tento text odstraňte)

- Přílohy v souboru závěrečné práce:

- Příloha A    xxxx

- 

- Elektronické přílohy:

- Příloha A    xxxx

-