

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 28. února 2024

.....
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 Angular	3
2.1.1 Komponenty	3
2.1.2 Správa stavů	3
2.1.3 Předávání vlastností	3
2.1.4 Servisy a direktivy	3
2.1.5 Životní cyklus	3
2.1.6 State management	3
2.1.7 Routování	3
2.1.8 Ekosystém	3
2.2 React	3
2.2.1 Komponenty	4
2.2.2 JSX	5
2.2.3 Správa stavů	5
2.2.4 Hooky	6
2.2.5 Životní cyklus	6
2.2.6 State management	7
2.2.7 Routování	7
2.2.8 Ekosystém	7
2.3 Svelte	8
2.3.1 Komponenty	8
2.3.2 Reaktivita	9
2.3.3 Předávání vlastností	10
2.3.4 Eventy	10
2.3.5 Životní cyklus	11
2.3.6 State management	12
2.3.7 Routování	12
2.3.8 Ekosystém	13
2.4 Vue	13
2.4.1 Single-File Components	14
2.4.2 Reaktivita	14
2.4.3 Předávání vlastností	15
2.4.4 Direktivy a eventy	16

2.4.5	Životní cyklus	17
2.4.6	State management	17
2.4.7	Routování	18
2.4.8	Ekosystém	18
2.5	Porovnání	18
3	Testování frameworků	19
3.1	Analýza a návrh testových úloh	19
3.2	Demonstrační aplikace	19
3.2.1	Angular	19
3.2.2	React	24
3.2.3	Svelte	25
3.3	Testování aplikací a výsledky	25
4	Ukázková kapitola	24
4.1	Struktura a formát	24
4.1.1	Jak strukturovat práci	24
4.2	Obrázky a tabulky	24
4.2.1	Vkládání ukázkového kódu	25
4.3	Vyznačování pojmů v textu	26
4.4	Odrážky, číslování, pojmenované odstavce	26
5	Práce se zdroji	28
5.1	Seznam použité literatury	28
5.2	Citace	28
5.3	Parafráze	29
	Závěr	31
	Seznam použité literatury	32
	Seznam obrázků	36
	Seznam tabulek	37
	Seznam zkratk	38
	Přílohy	39

3 Testování frameworků

- proč a co je obsahem kapitoly?

3.1 Analýza a návrh testových úloh

- co a proč porovnávám,
- v návrhu - jak, jaké testové úlohy?
- (dokumentace - možná nahoře, syntax, výkonnostní testy, velikosti bundlů, účel aplikace, rychlost, srozumitelnost, ...)

3.2 Demonstrační aplikace

V této kapitole srovnáme implementaci stejných funkcionalit ve třech vybraných frameworkcích.

3.2.1 Angular

Instalace projektu

- Node.js + NPM
- `npm init @angular@latest NAZEV_APLIKACE`
- <https://www.npmjs.com/package/@angular/create>
- <https://tailwindcss.com/docs/guides/angular>

Správa stavů

Pro implementaci jednoduchého counteru nejprve vytvoříme counter komponentu. Můžeme začít se strukturou HTML značek pro hlavní komponentu. Protože chceme opakovaně použít logiku jednotlivých tlačítek, vytvoříme komponentu counter-button. Ta může přijímat například nějaké CSS styly nebo přes EventEmitter (output) posílat informaci o kliknutí na tlačítko směrem nahoru ve stromě komponent. Funkci emit() našeho EventEmitteru zavoláme na tlačítku v counter-buttonu právě tehdy, když uživatel klikne na tlačítko – použijeme listener ve formě (click). K

propsání textu či jiných elementů nebo komponent mezi párovými tagy `<counter-button></counter-button>` pak poslouží párový či nepárový element `<ng-content />`.

Následně v counter komponentě musíme importovat třídu `CounterButtonComponent` a do všech elementů `counter-button` předat jejich vstupy a výstupy. Námi defikovanovanému outputu `buttonClicked` předáme v šabloně metodu, která se vykoná po emitu (kliknutí na tlačítko ve vnořené komponentě) a metodu zavoláme pomocí kulatých závorek. V rámci counter komponenty pak definujeme stav jako vlastnost `count` na třídě. Vlastnost pak můžeme modifikovat skrze metody třídy, které voláme v outputu `buttonClicked`.

- šablony + logika komponenty
- správa stavů (reaktivita)
- body k vypíchnutí: boilerplate frameworku

Interakce v uživatelském prostředí

Při vytváření jakékoli UI komponenty můžeme začít šablonou, nebo definovat funkční stránku. My začneme s tvorbou šablony. V případě vlastního dropdown samotným tlačítkem a seznamem možností. Otevření možností zajistíme tak, že na tlačítko přidáme `click listener`. Funkčnost pak zajistíme díky modifikaci stavu `isOpen`, který se provede při volání metody `toggleDropdown`. V rámci této metody je třeba zavolat i `event.stopPropagation()`. Předejdeme tak potenciální chybě ve formě tzv. `event bubblingu` – spuštění událostí na prvcích odlišných od cílového.

Podmíněně pak můžeme vypsát list možností, které získáme v jednom z inputů. Pro vypsání všech možností použijeme blok `@for`. K vybraní konkrétní možnosti použijeme zase `(click)` a do obslužné metody pošleme aktuální prvek v poli – `option`. Metoda `handleOptionClick` pak zajistí uložení aktuálně vybrané možnosti, zavření dropdownu a vyemitování vybrané možnosti do rodičovské komponenty.

V případě, že máme dropdown otevřen a chceme jej po kliknutí mimo tentýž dropdown bezpečně zavřít, nehledě na počet vykreslených dropdown komponent na stránce, budeme postupovat následovně. Pro každou komponentu vytvoříme unikátní vlastnost ve formě ID. To pak dynamicky umístíme na kořenový element dro-

dropdownu. V komponentě pak budeme naslouchat na události v DOM pomocí dekorátoru `@HostListener`. Dekorátor přijímá DOM událost, na který má poslouchat – `document:pointerdown`, případně další argumenty nebo také formu vypublikované události. Pod dekorátorem pak definujeme obslužnou metodu, která se volá při emitu specifikované události. V rámci metody pak zajistíme uzavření aktuálně otevřeného dropdownu.

Dropdown pak může mít různé inputy, které povedou k lepší znovupoužitelnosti. Hodnotu inputu (konkrétně např. `defaultValue`) v komponentě získáme v metodě životního cyklu `OnInit`. Kupříkladu v konstruktoru bychom dostali pouze `undefined`. Styly ve formě JavaScriptových hodnot do šablony přidáme pomocí `ngClass`. Když těchto hodnot chceme na elementu více, musíme je zřetězit pomocí JavaScriptu, nebo sloučit již dříve.

- body k vypíchnutí: dynamické stylování, logika v template
- problémy: zavírání posledně otevřeného dropdownu před otevřením dalšího D.
- výhody frameworku: podle bodů nahoře..., tvorba typů ve Svelte

Předávání vlastností, získávání dat z API

Pro ukázkou předávání vlastností a získávání dat z API můžeme vytvořit komponentu, která bude překládat zadaný text do vybraného jazyka. Začneme tedy vytvořením rodičovské komponenty, která při změně vlastností (zadaného textu uživatelem a výstupního jazyka) zavolá API, které vrátí přeložený text. V rámci této komponenty vytvoříme vnořené komponenty, které budou sloužit k zadání vstupního textu, výběru jazyka a zobrazení výsledku.

`LanguageDropdownComponent` umožní uživateli vybrat jazyk, do kterého chce přeložit text. Přes `EventEmitter` aktualizujeme výstupní jazyk v rodičovské komponentě. V rámci obslužné metody `handleLanguageChange` pak také aktualizujeme hodnotu vlastnosti `inputValuesChanges$`. Tato vlastnost je `Subject`, speciální typ `observable`, z knihovny `RxJS`. Později dovolí na základě změny hodnoty poslat dotaz na server ve správný moment. Podobným způsobem poté můžeme registrovat událost změny vstupního textu – naslouchat na změnu vstupního textu.

Zadání vstupního textu pak může řešit komponenta `TranslationInputComponent`, která obdobným způsobem aktualizuje hodnotu vstupního textu v rodičovské

komponentě. Aktuální hodnotu formulářového prvku nastavíme pomocí `[ngModel]`. Pro naslouchání na změnu hodnoty formulářového prvku zase využijeme (`ngModelChange`). V případě, že chceme aktualizovat výšku textového pole na základě jeho obsahu, můžeme využít vlastní direktivu `AutosizeTextAreaDirective`. V konstruktoru direktivy získáme `element`, na který přidáme tuto direktivu. Dále budeme potřebovat třídu `Renderer2`, která umožňuje manipulovat s DOM. V direktivě budeme naslouchat na změnu hodnoty textového pole pomocí dekorátoru `@HostListener` a události `input`. Následně v rámci obslužné metody zajistíme aktualizaci výšky.

Změny hodnoty vlastnosti `inputValuesChanges$` začneme odebírat pomocí `subscribe`. Abychom předešli dotazování serveru ihned po změně hodnoty vlastnosti `inputValuesChanges$`, použijeme operátor `debounceTime`. Ten povolí poslat dotaz na server až po uplynutí určité doby od poslední změny, kterou můžeme nastavit. `Subscribe` zavolá veřejnou metodu služby (`getTranslation`), která vrací přeložený text. Nakonec, aby dotazování serveru fungovalo, je třeba metodu `setUpInputChangeSubscription` zavolat v konstruktoru nebo hooku `OnInit`.

Se službou `TranslationService` a veřejnou metodou pro vykonání dotazu nám pomůže třída `HttpClient`. Ta je dostupná přímo v základních modulech Angularu. Službu `HttpClient` získáme v konstruktoru, kde ji pomocí klíčového slova `private` přiřadíme do vlastností třídy. Následně na HTTP klientovi zavoláme metodu `post` vůči API, které vrátí přeložený text ze serveru. Pokud úspěšná odpověď ze serveru obsahuje složitější strukturu, ze které potřebujeme získat jen nějakou část, pak s konverzí odpovědi pomůže RxJS operátor `map()`. Metoda `getTranslation` vrací observable, v translator komponentě proto hodnoty odebíráme pomocí metody `subscribe`.

V momentě, kdy obdržíme odpověď ze serveru, zobrazíme přeložený text uživateli. K tomu poslouží `TranslationOutputComponent`, které na vstupu předáme výstupní text spolu s dalšími vstupními vlastnostmi. V rámci šablony pak podmíněně vykreslíme přeložený text, chybu nebo načítání.

Při zarovnání vstupního a výstupního pole v UI si musíme dát pozor na to, že šířku je potřeba nastavit již v prvním potomku `div` elementu, na kterém nastavíme flexbox. Důvod spočívá v tom, že Angular vytváří v DOMu element pro každou komponentu.

- předávání vlastností nahoru a dolů

- fetchování dat
- body k vypíchnutí: velice odlišné reakce na změny, stylování komponent nebo elementů, update textarea (hodnoty), jiné řešení modularity (update stylů textarea)
- problémy:
- výhody frameworku: předávání vlastností má nej Svelte

Tvorba formulářů

Angular je flexibilní z pohledu možností tvorby formulářů. My použijeme reaktivní formuláře, jelikož jsou flexibilnější a umožní nám jednodušší reakce na změny prvků. Vytvoříme komponentu zaměřenou na jednoduché investiční kalkulace. Bude obsahovat dvě vnořené komponenty: formulář pro zadání vstupních dat a komponentu výsledku kalkulace, která se zobrazí po potvrzení formuláře.

Můžeme začít přímo s tvorbou reaktivního formuláře. Typ `InvestForm` popisuje strukturu formulářových prvků, které bude formulář obsahovat. Protože formulářových prvků budeme mít více (a souvisí spolu), vytvoříme formulářovou skupinu, ve které definujeme samotné formulářové prvky. Formulářová skupina `investForm` bude vlastnost třídy, která umožní přístup k hodnotám formuláře a jeho validaci. Vlastnost `investForm` nastavíme v rámci `OnInit` hooku, protože potřebujeme nastavit počáteční hodnoty formuláře na základě vstupní vlastnosti `defaultValues`. Důvodem je, že hodnoty vstupních vlastností jsou v komponentě dostupné nejdříve v rámci hooku `OnInit`.

Zde narazíme na problém s nenastavením počáteční hodnoty vlastnosti přímo nebo v konstruktoru. Můžeme ho vyřešit za pomoci vykřičníku – řekneme tak TypeScriptu, že obsah proměnné je nenulový. Další možností je vypnout pravidlo `strictPropertyInitialization` v souboru `tsconfig.json`. Po opravě chyby se vrátíme k formuláři, který dle mého názoru nejsnadněji vytvoříme pomocí třídy `FormBuilder` ze základního balíčku `@angular/forms`. Formulář vytvoříme v samostatné metodě `initializeInvestForm`, jejíž výsledek přiřadíme v hooku `OnInit` do vlastnosti `investForm`. Samotná metoda `initializeInvestForm` bude vracet instanci třídy `FormGroup`. Argumentem pro metodu `group` pak je objekt, který popisuje strukturu formuláře.

Vlastnosti objektu budou klíče formulářových prvků a jejich hodnoty pole, kde první prvek bude počáteční hodnota a druhý prvek pole validátorů.

V šabloně následně propojíme formulářovou skupinu s formulářem. K tomu poslouží direktiva `[formGroup]` a její hodnotu nastavíme na vlastnost `investForm`. V rámci formuláře pak vytvoříme formulářové prvky, které propojíme direktivou `formControlName`. Hodnota pak musí odpovídat klíči prvku ve formulářové skupině. Pro zajištění efektivní obsluhy chyb formuláře můžeme využít getter metody, které vrátí konkrétní formulářový prvek.

Dále vytvoříme tlačítko s typem `submit`, přes které uživatel formulář potvrdí. Na form značku přidáme (`ngSubmit`), který vyemituje událost při potvrzení formuláře. Obslužná metoda pak prostřednictvím výstupové vlastnosti publikuje aktuální hodnotu reaktivního formuláře do rodičovské komponenty.

V rámci rodičovské komponenty tedy vykreslíme samotný formulář a při jakémkoli potvrzení formuláře získáme aktuální hodnoty z formuláře díky `outputu`. Hodnoty formuláře pak dostaneme v obslužné metodě `handleFormChanged`. Pomocí služby `FutureValuesCalculatorService` tyto hodnoty transformujeme do požadovaného formátu. Výsledek uložíme do vlastnosti `futureValues`.

Když jsou hodnoty vypočteny, vykreslíme je na stránce prostřednictvím komponent `future-values-info` a `future-value-info`. První z komponent slouží k rozložení výsledků do požadovaného formátu a vytvoření komponent pro jednotlivé výsledky. Komponenta `future-value-info` pak přijímá vstupní vlastnost, kterou v šabloně před vykreslením v DOM přetransformujeme díky rouře (`LocalizedNumberPipe`). Stejného výsledku bychom mohli dosáhnout i přes metodu na třídě. Tento přístup Angular nedoporučuje, jelikož metody se v rámci šablony spouští opakovaně a mohou způsobit problémy s výkonem. Oproti tomu roura umožní lepší znovupoužitelnost a přehlednost.

Country guesser

Layout a routování

3.2.2 React

Instalace projektu

Správa stavů

Interakce v uživatelském prostředí

Předávání vlastností, získávání dat z API

3.2.3 Svelte

Instalace projektu

Správa stavů

Interakce v uživatelském prostředí

Předávání vlastností, získávání dat z API

3.3 Testování aplikací a výsledky

- výsledky a průběh z 3.1

Seznam obrázků

1	Ukázka vložení titulku s označením zdroje	25
---	---	----

Seznam tabulek

1	Ukázka tabulky	25
---	--------------------------	----

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

-