SLEZSKÁ UNIVERZITA V OPAVĚ
Filozoficko-přírodovědecká fakulta v Opavě

# BAKALÁŘSKÁ PRÁCE

Opava 2024 Lukáš Sukeník

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 Ciencialová, Ph.D.

**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 oddělujte 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 27. 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

# 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 frameworcích.

### 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> nám 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 tvorbě jakékoli UI komponenty můžeme začít jak šablonou, tak i logikou. My začneme s tvorbou šablony. V případě vlastního dropdown samotným tlačítkem a seznamem možností. Otevření možností zajístíme tak, že na tlačítko přidáme click listerner. 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 vypsat list možností, které získáme v jednom z inputů. Pro vypsání všech možností použijeme blok @for. K vybraní konkrétní možnosti použijeme zase (click) a do metody pošleme konkrétní možnost pole – 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 dropdownu. V komponentě pak budeme naslouchat na DOM eventy pomocí dekorátoru @HostListener. Přijímá DOM event, na který má poslouchat – document:pointerdown,

případně další argumenty nebo také formu vypublikovaného eventu. Pod dekorátorem pak definujeme obslužnou metodu, která se volá při emitu specifikovaného eventu. 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 až v lifecycle hooku 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 ukázku předávání vlastností a získávání dat z API můžeme vytvořit komponentu, která bude překládat zadaný text do vybraného jazyka. Začneme tedy vytvořením rodičovské komponenty, která nám při změně vlastností (zadaného textu uživatelem a výstupního jazyka) zavolá API, které nám 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 nám na základě změny hodnoty dovolí poslat dotaz na server ve správný moment. Podobným způsobem poté můžeme naslouchat na změny 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 nám umožní 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$ musíme začít odebírat pomocí subscribe. Abychom předešli dotazování serveru ihned po změně hodnoty vlastnosti inputValuesChanges$, použijeme operátor debounceTime. Ten nám povolí poslat dotaz na server až po uplynutí určité doby od poslední změny. Subscribe zavolá věřejnou metodu služby (getTranslation), která nám vrátí přeložený text. Nakonec aby nám 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 nám vrátí přeložený text. Pokud nám v úspěšné odpovědi ze serveru přijde nějaká složitější struktura, z které chceme vrátit jen nějakou část, pak nám s konverzí odpovědi pomůže RxJS operátor map(). Metoda getTranslation vrací observable, proto musíme v translator komponentě hodnoty odebírat pomocí metody subscribe.

V momentě, kdy dostaneme odpověď ze serveru, chceme zobrazit přeložený text uživateli. K tomu nám 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 musíme nastavit již v prvním potomku div elementu, na kterém nastavíme flexbox. Důvod je zřejmý – Angular v DOM vytvoří speciálně element pro každou komponentu.

- předávání vlastností nahoru a dolů
- fetchování dat

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

**Tvorba formulářů**

Angular nám umožňuje vytvářet formuláře několika způsoby. My použijeme reaktivní formuláře, které nám umožní vytvářet formuláře programaticky. Komponenta, kterou vytvoříme, se bude týkat jednoduché investiční kalkulačky. Bude obsahovat dvě vnořené komponenty pro formulář a výsledek při potvrzení formuláře.

Můžeme začít přímo s tvorbou reaktivního formuláře. Typ InvestForm popisuje strukturu formulářových prvků, které budeme chtít. Protože prvků formuláře chceme mít více (a souvisí spolu), vytvoříme formulářovou skupinu, ve které budou samotné formulářové prvky. Formulářová skupina bude vlastnost třídy, kterou lze nastavit ihned, v konstruktoru, případně některém v hooku OnInit. Jelikož chceme nastavit počáteční hodnoty fomuláře na základě vstupní vlastnosti, musíme vlastnost investForm nastavit až v hooku OnInit.

Zde narazíme na problém s nenastavením počáteční hodnoty vlastnosti přímo nebo v konstruktoru. Můžeme ho vyřešit pomocí vikričníhu – řekneme tak TypeScriptu, že vlastnost je nenulová. Další možností je přenastavení pravidla strictPropertyInitialization v souboru tsconfig.json. Po opravě chyby se vrátíme k formuláři, který nejjednodušeji 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 nám poslouží direktiva [formGroup] a její hodnotu nastavíme na vlastnost investForm. V rámci formuláře pak vytvoříme formulářové prvky, které propojíme direk-

tivou formControlName. Hodnota pak musí odpovídat klíči prvku ve formulářové skupině. Při obsluze chyb formuláře si můžeme pomoct pomocí getter metod, které nám vrátí konkrétní formulářový prvek.

Dále vytvoříme tlačítko, přes které uživatel formulář potvrdí. Na form značku přidáme (ngSubmit), který vyemituje událost při potvrzení formuláře. V obslužné metodě pak pomocí výstupové vlastnosti vypublikujeme aktuální hodnotu reaktivního formuláře do rodičovské komponenty.

Rodičovská komponenta tedy vykreslí samotný formulář, díky výstupní vlastnosti formuláře získá aktuální hodnotu formuláře. Hodnotu formuláře pak pomocí služby transformuje do požadovaného formátu, který následně vykreslí na stránce. V obslužné metodě handleFormChanged následně provedeme výpočet budoucí hodnoty investice pomocí služby FutureValuesCalculatorService a výsledek uložíme do vlastnosti futureValues.

Pokud máme hodnoty vypočteny, vykreslíme je na stránce pomocí 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 pomocí roury (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 výkonové problémy. Oproti tomu roura nám 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

# Seznam tabulek

# 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
  -