Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Micro frontend architektura

Diplomová práce

Autor: Bc. David Šprla
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Pavel Kříž, Ph.D.

Odborný konzultant: Daniel Kopřiva

Hradec Králové listopad 2021

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 14.11.2021 Bc. David Šprla

Poděkování:

Rád bych poděkoval vedoucímu diplomové práce Ing. Pavlu Křížovi, Ph.D. za metodické vedení práce a odbornému konzultantu Danielu Kopřivovi za věcné připomínky a věnovaný čas při vypracování této diplomové práce.

Anotace

Zkoumaným tématem této diplomové práce je micro frontend architektura. Jedná se o přístup k vývoji frontendové části. Hlavní myšlenkou této architektury je rozdělení monolitických frontendů na malé nezávislé aplikace, které jsou spravovány jednotlivými vývojovými týmy. V současné době jej využívají některé velké společnosti, mezi které patří například IKEA, Starbucks, Spotify, DAZN, Skyscanner, Zalando, SoundCloud, OpenTable, Upwork a mnoho dalších. Cílem práce je seznámit čtenáře s konceptem micro frontend architektury, jakým způsobem funguje, s dostupnými možnostmi integrace, komunikací mezi micro frontendy, otázkou stylování, verzování a nasazení do produkčního prostředí. V práci jsou uvedeny výhody i nevýhody micro frontend architektury a také případy vhodného i nevhodného použití. Micro frontend architektura není univerzálním přístupem pro všechny typy projektů. Aby se zjistilo, zda je použití této architektury na konkrétním projektu vhodné, je nutné předem provést důkladnou analýzu a porovnat, zda budou výhody převažovat nad nevýhodami. Vhodným případem použití micro frontend architektury je především u středních a velkých projektů, kde jsou monolitické frontendy obtížně spravovatelné.

Annotation

Title: Micro frontend architecture

The researched topic of this diploma thesis is micro frontend architecture. This is an approach to the development of the frontend part. The main idea of this architecture is to divide the monolithic frontends into small independent applications, which are managed by individual development teams. It is currently used by some large companies such as IKEA, Starbucks, Spotify, DAZN, Skyscanner, Zalando, SoundCloud, OpenTable, Upwork and many more. The aim of the thesis is to acquaint the reader with the concept of micro frontend architecture, how it works, with available integration options, communication between micro

frontends, the issue of styling, versioning and deployment in a production environment. The thesis presents the advantages and disadvantages of micro frontend architecture as well as cases of suitable and inappropriate use. The micro frontend architecture is not a universal approach for all types of projects. In order to determine whether the use of this architecture on a particular project is appropriate, it is necessary to carry out a thorough analysis in advance and compare whether the advantages outweigh the disadvantages. A suitable case of using the micro frontend architecture is especially for medium and large projects, where monolithic frontends are difficult to manage.

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

SPA	Single Page Application
SSR	Server Side Rendering
HTML	HyperText Markup Language
API	Application Programming Interface

DOM	Document Object Model
SEO	Search Engine Optimization
CSS	Cascading Style Sheets
CDN	Content Delivery Network
DDD	Domain-Driven Design
RPS	Requests per second
ESI	Edge Side Includes
SSI	Server Side Includes
XML	Extensible Markup Language
SEO	Search Engine Optimization
URL	Uniform Resource Locator
BEM	Block, Element, Modifier
CI	Continuous Integration
IoT	Internet of Things
UX	User Experience
LCP	Largest Contentful Paint
FID	First Input Delay
CLS	Cumulative Layout Shift

Ukázky kódu

1 Úvod

Zkoumaným tématem této diplomové práce je micro frontend architektura. Jedná se o přístup k vývoji frontendové části aplikací, který v současné době nabírá na popularitě. Hlavní myšlenkou této architektury je rozdělení monolitických frontendů na malé nezávislé aplikace, které jsou spravovány jednotlivými vývojovými týmy. V současné době jej využívají některé velké společnosti, mezi které patří například IKEA, Starbucks, Spotify, DAZN, Skyscanner, Zalando, SoundCloud, OpenTable, Upwork a mnoho dalších.

Během výzkumu tohoto tématu jsem se zabýval hledáním patřičných odpovědí na předem připravené otázky, které by měly čtenáře dostatečným způsobem seznámit s micro frontend architekturou.

Mezi tyto otázky patří úvodní seznámení, popis, k čemu architektura slouží, jaké výhody dokáže nabídnout, zda má případně i nějaké nevýhody a kde lze nalézt její uplatnění. Dále je nutné seznámit čtenáře s různými variantami implementace, možnými způsoby rozdělení aplikace, postupy pro kompozici jednotlivých micro frontendů a jejich routování. Pokud mezi sebou potřebují micro frontendy komunikovat, je dobré znát vhodné přístupy ke komunikaci. Další zkoumanou otázkou jsou možnosti izolovaného stylování aplikace a globální stavy (state). Zajímavou oblastí je použití více frameworků (React, Vue, Angular) v projektu. Dále bylo nutné zjistit, jakým způsobem lze micro frontendy verzovat a také vyřešit otázku procesu nasazení do produkčního prostředí.

Druhá část této práce se věnuje ukázkové micro frontendové aplikaci. Primárním účelem této aplikace bylo praktické otestování zkoumané architektury a ověření některých uvedených výhod a nevýhod.

2 Úvod do vývoje frontendu

Vývoj prezentační vrstvy moderní webové aplikace prošel za poslední řadu let velkým pokrokem. Vliv na tuto skutečnost mají především neustále rostoucí nároky na aplikaci. Dnešní vyvíjené webové aplikace musí splňovat rychlé načítání, možnost spuštění na široké škále zařízení a poskytovat okamžitou reakci na interakci ze strany uživatelů.

Moderní webové aplikace jsou mnoha firmami používány jako klíčové interaktivní prostředí pro jejich uživatele, proto je na inovace v oblasti vývoje prezentační vrstvy kladen velký důraz a pozornost [1]. Vývojové týmy prakticky neustále hledají nové způsoby, jak efektivně vyvíjet, nasazovat a udržovat aplikace, aby tyto firmy mohly poskytovat hodnoty svým zákazníkům rychle a efektivně.

Na trh jsou neustále uváděny nové frontendové frameworky, díky kterým si mohou vývojáři vybrat z široké nabídky možností pro vytváření robustních a funkčně bohatých aplikací, jako jsou single-page aplikace (SPA), aplikace s vykreslením na straně serveru (SSR) nebo statické HTML soubory kombinované do webové stránky. U těchto aplikací však velmi často nastává situace, že skončí jako frontendové monolity. Tím aplikace na straně klienta narůstá a dostává se do problému s obtížným škálováním při vývoji, zejména pokud nastane situace, kdy několik odlišných týmů potřebuje upravovat totožnou frontendovou aplikaci současně [2]. Pokud je vyvíjena malá webová aplikace s menším počtem vývojářů, nebude takový vývoj aplikace nijak obtížným úkolem. Nicméně, při vylepšování a obohacování komplexní webové aplikace o nové funkce, se veškeré kapacity menšího počtu vývojářů rychle vyčerpají. Právě takové situace pomáhá řešit micro frontendová architektura [1].

2.1 Současné architektury pro vývoj frontendu

Současná doba nabízí vývojářům široký výběr architektur pro návrh a vývoj frontendových aplikací. Je důležité zmínit, že zatím žádná architektura nefunguje jako perfektní nástroj pro všechny typy projektů, proto je důležité při výběru architektury porozumět všem výzvám, které se naskytnou během vývoje [2,9].

2.1.1 Single-page aplikace

Single-page aplikace pravděpodobně patří mezi nejpoužívanější implementace. Webové SPA načítají pouze jeden webový dokument a poté dochází k aktualizaci obsahu těla tohoto jediného dokumentu skrze rozhraní API v JavaScriptu, jako je XMLHttpRequest a Fetch, pokud má být zobrazen jiný obsah. Díky tomu umožňuje uživatelům používat webové stránky bez nutnosti načtení úplně nových stránek ze serveru. Existuje velké množství frameworků a knihoven (React, Vue, Angular), které jsou navrženy tak, aby vytvořily vrstvu abstrakce pro manipulaci s DOMem. SPA mají mnoho výhod. Stažení kódu aplikace na straně klienta proběhne pouze jednou, na začátku životního cyklu, a celá logika aplikace je poté k dispozici pro celou relaci uživatele. Vyhýbají se vícenásobným dotazům na server, pro načtení další aplikační logiky, a okamžitě vykreslují všechny pohledy během životního cyklu aplikace, což může vést k dynamičtějšímu zážitku.

SPA mají i své nevýhody pro určité typy aplikací. Dochází k delší době prvního načtení než u jiných architektur, protože je stahována celá aplikace místo toho, co potřebuje uživatel vidět. Další nevýhodou je optimalizace pro vyhledávače (SEO), jelikož je obtížné indexovat veškerý obsah SPA, pokud nejsou připraveny alternativní způsoby. Problémem pro SPA může být i nedostatečný hardware zařízení nebo také organizační stránka. Pokud jsou spravovány velké SPA distribuovanými nebo společnými týmy na stejné kódové základně, různé oblasti totožné aplikace mohou skončit ve směsi přístupů a rozhodnutí, což může vyústit ve zmatek mezi členy týmu [9,11].

2.1.2 JAMstack architektura

JAMstack je zkratkou pro JavaScript, API a Markup. Jedná se o úspěšnou a stále oblíbenější frontendovou architekturu v posledních letech, sloužící pro vytváření moderních webových aplikací založených na pokročilých nástrojích a pracovních postupech. Mezi jeho výhody patří vyšší bezpečnost, levnější infrastruktura a lepší výkon předem vykreslených statických stránek s dynamickými možnostmi prostřednictvím JavaScriptu a API, obsluhované bez webových serverů. Výstupem je statický artefakt složený z HTML, CSS a JavaScriptu. Artefakt může být obsloužen přímo CDN, protože aplikace nevyžaduje žádnou technologii na straně serveru

ke svému fungování. Díky tomu se mohou frontendoví vývojáři zaměřit pouze na vývoj a ladění frontendu. Zatímco micro frontend architektura umožňuje rozdělení frontendového monolitu, JAMstack se soustředí na úplné oddělení frontendu a backendu [2,9,12].

2.1.3 Izomorfní/univerzální aplikace

Izomorfní nebo univerzální aplikace jsou webové aplikace, v rámci kterých je kód mezi serverem a klientem sdílen a může běžet v obou kontextech. Při správném použití přináší tato technika určité výhody. Hodí se obzvlášť pro aplikace, kde je potřeba řešit dobu do interakce, A/B testování nebo SEO, díky možnosti generování stránky na straně serveru. Poté může prohledávač rychleji indexovat finální stránku. Protože webová aplikace sdílí kód mezi serverem a klientem, může server provést část vykreslování stránky požadované prohlížečem, načíst potřebná data k zobrazení z databáze nebo z jednoho či více rozhraní API, zkompilovat data dohromady a poté jej předběžně vykreslit pomocí šablonovacího systému, který se používá pro generování pohledu. Stránka se klientovi zobrazí, aniž by potřebovala další síťová volání pro vyžádání dalších dat k zobrazení. Vzhledem k tomu, že je požadovaná stránka předem vykreslena na serveru a je plně nebo částečně interpretována na backendu, prodlouží se tím doba do interakce.

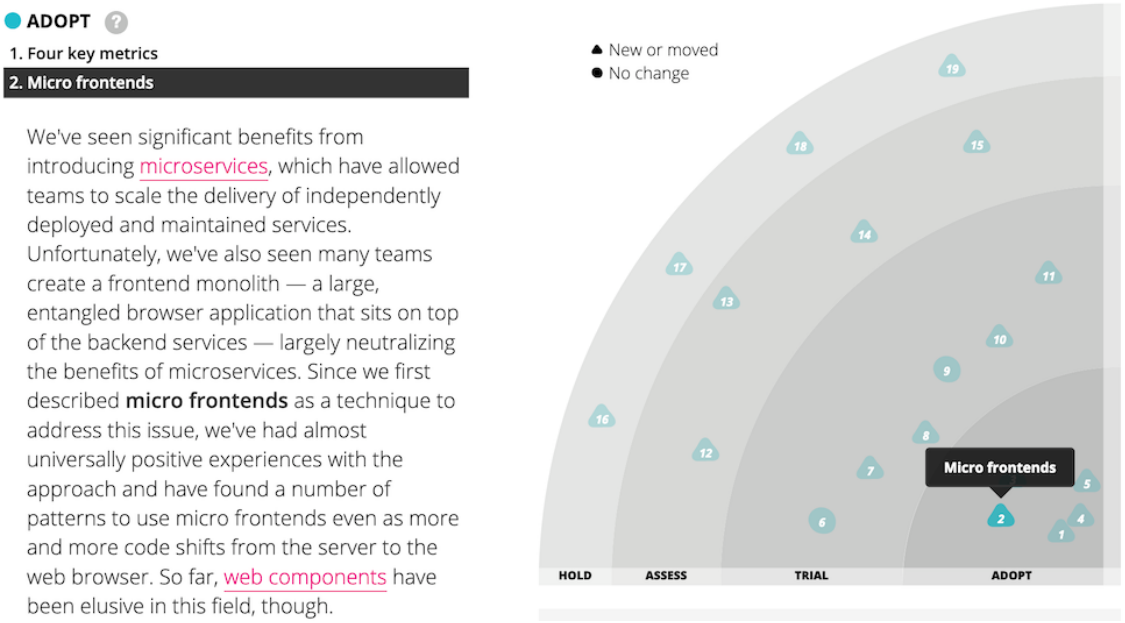
Mezi nevýhody izomorfních aplikací patří organizační problémy podobně jako u SPA. Dalším problémem může být škálovatelnost, pokud návštěvnost aplikace dosahuje miliónů uživatelů. V tomto případě je potřeba zvolit správnou strategii ukládání do mezipaměti, aby se minimalizovaly dopady na servery, protože se generují stránky, které jsou předem vykresleny na serveru [2,9,13].

2.1.4 Statické webové stránky

Statické webové stránky jsou užitečné pro rychle implementovatelné weby, které nejsou určeny k tomu, aby byly dostupné po dlouhou dobu. Takovým případem je inzerce konkrétních produktů nebo služeb. Pokaždé, když uživatel klikne na odkaz, načte se nová statická stránka [9].

3 Seznámení s micro frontend architekturou

S termínem micro frontend bylo možné se poprvé setkat v technologickém radaru (obrázek 1) společnosti ThoughtWorks na konci roku 2016 [3]. Společnost si byla vědoma výhod plynoucích ze zavedení architektury mikroslužeb, díky kterým bylo vývojovým týmům umožněno škálování vývoje nezávisle udržovaných a nasazovaných služeb. Tyto týmy se často snažily předejít vytvoření frontendových monolitů stejně jako na straně backendu s pomocí mikroslužeb. Zde se dostává do popředí termín micro frontend [4].



Obr. 1 Technologický radar ThoughtWorks
Zdroj: https://martinfowler.com/articles/micro-frontends.html

Micro frontend architektura sdílí hlavní principy, výhody a problémy architektury mikroslužeb. Umožňuje rozdělení frontendu na jednotlivé polonezávislé frontendy, oddělení obchodní logiky od frontendu a vytvoření navzájem nezávislých interagujících služeb [2].

Myšlenka micro frontend architektury však není nová. Mezi předchozí používaná pojmenování konceptu patří například název Frontend Integration for Verticalised Systems nebo Self-contained Systems. Oproti těmto předchůdcům je micro frontend mnohem srozumitelnější a přesnější termín [5].

Benefitů, plynoucích z micro frontendové architektury, si všimly i některé společnosti průmyslového odvětví. V současné době se micro frontendy úspěšně používají ve velkých společnostech, mezi které patří například IKEA, Starbucks, Spotify, DAZN, Skyscanner, Zalando, SoundCloud, OpenTable, Upwork a mnoho dalších [2,6,7].

3.1 K čemu slouží micro frontend architektura a co dokáže řešit

Společnost Thoughtworks definuje techniku micro frontendu jako: „*architektonický styl, kde jsou nezávisle dodávané frontendové aplikace skládány do většího celku*“ [3].

Přistupuje k vývojovému procesu frontendu pomocí konceptu a myšlenky architektury mikroslužeb. Dokáže vyřešit některé problémy frontendových monolitů díky rozdělení monolitické kódové báze (codebase) na několik nezávislých funkcí. Takto rozdělené části webové aplikace jsou vyvíjeny různými autonomními týmy a poté jsou výsledné funkce sloučeny do homogenního rozhraní [8,9]. Týmy v micro frontend architektuře jsou mezifunkční a vyvíjí komplexní (end-to-end) funkce, od uživatelského rozhraní až po databázi. Díky tomu je možné frontendovým vývojářům poskytnout stejnou úroveň flexibility, testovatelnosti a rychlosti, jakou disponují vývojáři backendu s architekturou mikroslužeb [5,8,9,10].

3.2 Klíčové koncepty micro frontend architektury

Všestranný a flexibilní vývojový proces, který umožňuje micro frontend architektura, spočívá v následujících klíčových myšlenkách [8].

3.2.1 Technologická nezávislost

Každý tým, pracující na různých komponentách aplikačního frontendu, by měl mít možnost si vybrat a vylepšovat svůj stack bez jakékoliv konzultace nebo koordinace s ostatními týmy. Použití vlastních elementů je efektivní způsob, jak toho dosáhnout, protože umožňují jednotlivým týmům skrýt podrobnosti implementace při vzájemném sdílení neutrálního rozhraní [5,8,10].

3.2.2 Izolace týmových kódů

Klíčem k vytváření nezávislých aplikací je zajistit, aby byly samostatné. Pro dosažení samostatnosti je nutné izolovat týmový kód tím, že nedojde ke sdílení běhového prostředí, a to i přesto, že všechny týmy používají pro vývoj stejný framework [8].

3.2.3 Vytvoření týmových prefixů

Pokud někde není možné dosáhnout izolace, je vhodné použít konvenci pojmenování, aby se předešlo kolizím a ujasnilo vlastnictví. Konvenci pojmenování lze použít například u názvů CSS, Eventů, Local Storage a Cookies [5,10].

3.2.4 Upřednostnit nativní funkce prohlížeče před vlastními API

Pokud je opravdu nutné použít vlastní přizpůsobené API rozhraní před nativními funkcemi prohlížeče, je nezbytné zachovat co nejjednodušší řešení. Díky tomu lze zefektivnit operativnost při vytváření API rozhraní pro více týmů [8].

3.2.5 Tvorba odolných aplikací

Odolné aplikace spočívají na užitečných funkcích, které jsou schopny určitým způsobem obstát i v takových případech, kdy dojde k selhání JavaScriptu nebo jeho prodlevě při spuštění. Mezi dva hlavní způsoby, jak toho dosáhnout, patří univerzální vykreslování a progresivní vylepšování [5,8].

4 Implementace micro frontend architektury

Existují různé přístupy k architektuře micro frontendové aplikace. Před samotnou volbou nejlepšího možného přístupu pro daný projekt je nutné porozumět kontextu působnosti. Některá architektonická rozhodnutí budou muset být učiněna předem, jelikož se od těchto rozhodnutí budou odvíjet ta budoucí. Mezi taková rozhodnutí patří například definice micro frontendu z technického hlediska, organizace pohledů, jakým způsobem bude sestaven konečný pohled pro uživatele, komunikace mezi micro frontendy a otázka sdílení dat.

Tyto typy rozhodnutí se nazývají jako micro-frontends decisions framework a skládají se ze čtyř klíčových oblastí [2,9,14].

1. Definice
2. Kompozice
3. Routování
4. Komunikace

Každá z těchto oblastí hraje zásadní roli k dosažení dobrých výsledků v micro frontendovém projektu. V průběhu vývoje bude potřeba učinit další rozhodnutí, nicméně díky těmto čtyřem pilířům by mělo být možné kaskádovat drtivou většinu ostatních rozhodnutí, aniž by bylo nutné znovu provést posudek architektury [14].

Celkové shrnutí micro-frontends decisions frameworku je možné vidět v tabulce níže. Pro každou definici micro frontendu jsou uvedeny možnosti kompozice, routování a komunikace.

Tabulka 1 Shrnutí micro-frontends decisions frameworku

Definice	Kompozice	Routování	Komunikace
Horizontální	Client-side Edge-side Server-side	Client-side Edge-side Server-side	Event emitter Custom events Web storage Query strings
Vertikální	Client-side Server-side	Client-side Edge-side Server-side	Web storage Query strings

Zdroj: Mezzalira, Luca. Building Micro-Frontends. O'Reilly Media, Inc., 2021.

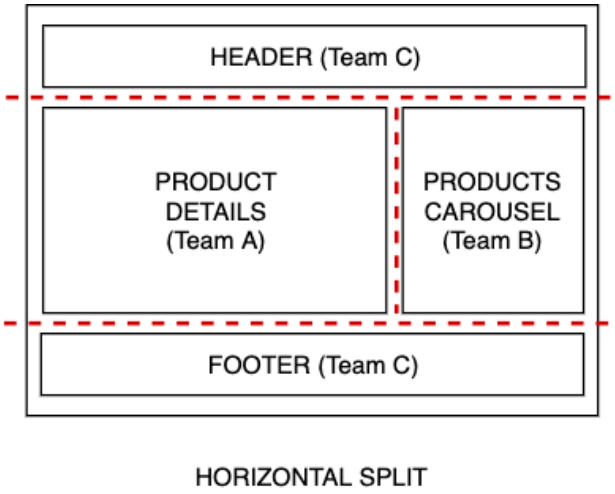
4.1 Definice micro frontendů

Na začátku je nutné rozhodnout, jakým způsobem bude rozdělen aplikační frontend. Primárně existují dva způsoby, jakými lze aplikační frontend rozdělit.

1. Horizontální rozdělení
2. Vertikální rozdělení

4.1.1 Horizontální rozdělení

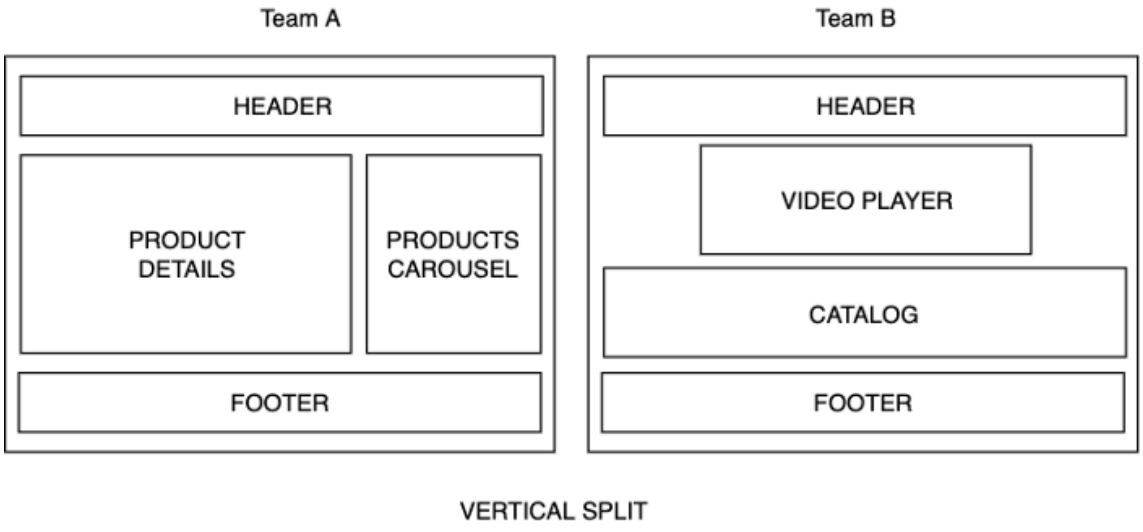
S horizontálním rozdělením (obrázek 2) bude v jediném pohledu načteno několik micro frontendů. Za tyto části pohledu bude odpovědných několik týmů, přičemž bude nutné zachovat určitou soudržnost ve vývoji, například jednotný vzhled. Horizontální rozdělení nabízí větší flexibilitu, jelikož je možné použít některé micro frontendy opakovaně v různých pohledech. Nevýhodou tohoto přístupu je horší škálovatelnost. Používá se především u obsahově náročných webů [9,15,16].



Obr. 2 Horizontální rozdělení
Zdroj: https://medium.com/@lucamezzalira/micro-frontends-decisions-framework-ebcd22256513

4.1.2 Vertikální rozdělení

V případě vertikálního rozdělení (obrázek 3) se využívá multifunkčních týmů k vývoji micro frontendů, které vyžadují pouze znalost jedné subdomény. Hlavní výhodou tohoto přístupu je možnost rozdělení problémové domény na menší části, přesně dle potřeby vývojového týmu. Kvůli tomu se některé stránky skládají z několika mikro rozhraní. Primární využití vertikálního rozdělení je u velkých webových aplikací a webových portálů [2,15,16].



Obr. 3 Vertikální rozdělení
Zdroj: https://medium.com/@lucamezzalira/micro-frontends-decisions-framework-ebcd22256513

4.2 Kompozice micro frontendů

Jeden z hlavních principů micro frontendové architektury spočívá v rozdělení webové aplikace do různých segmentů. Pro kompozici jednotlivých micro frontendových aplikací lze použít tři různé přístupy [2,8].

1. Client-side (horizontální i vertikální rozdělení)
2. Edge-side (horizontální rozdělení)
3. Server-side (horizontální rozdělení)

Tyto přístupy ke kompozici micro frontendových aplikací se odvíjejí od rozdělení aplikačního frontendu. V případě horizontálního rozdělení lze použít všechny tři přístupy. Vertikální rozdělení nabízí především kompozici na straně klienta [9].

Co se týče použití více frameworků v projektu, jedná se o kontroverzní rozhodnutí. Technicky je implementace více UI frameworků možná. Vytváří však problémy s výkonem, protože je nutné stáhnout větší množství dat, a navíc může dojít k potencionálním kolizím závislostí. Proto se nedoporučuje používat více frameworků v micro frontendové architektuře, stejně jako v jiných frontendových architekturách, jako je například SPA. Existují však případy použití, kde výhody tohoto přístupu převažují nad nevýhodami, například při migraci starší aplikace na novou. Následkem čehož jsou micro frontendy vydávány spíše iterativně, než aby byly vydány všechny najednou [9].

Aby bylo možné usnadnit výběr vhodného přístupu kompozice, bude v následujících kapitolách uvedeno souhrnné hodnocení každého přístupu. Mezi hodnocené vlastnosti pro každou implementaci patří:

- Nasaditelnost – spolehlivost a jednoduchost nasazení micro frontendů v prostředí.
- Modularita – snadné přidání nebo odebrání jednotlivých micro frontendů a snadná integrace se sdílenými komponentami, které jsou hostovány micro frontendy.
- Jednoduchost – jednoduchost pochopení nebo realizace.

- Testovatelnost – míra podpory testování softwarového artefaktu. Pokud je míra testovatelnosti vysoká, pak je hledání chyb v systému pomocí testování jednodušší.
- Výkon – ukazatel toho, do jaké míry micro frontend odpovídá kvalitě UX (User Experience) popsaného webovými vitálními prvky, které lze považovat za základní metriky pro zdravý web. Mezi tyto prvky patří například LCP (Largest Contentful Paint), FID (First Input Delay), CLS (Cumulative Layout Shift) [46].
- Vývojářské zkušenosti – zkušenosti, kterým jsou vývojáři vystaveni. Patří sem klientské knihovny, SDK sady, frameworky, open source kód, nástroje, API, technologie nebo služby.
- Škálovatelnost – schopnost procesu, sítě, softwaru nebo organizace růst a řídit zvýšenou poptávku.
- Koordinace – sjednocení nebo synchronizace úsilí členů, za účelem zajištění jednotného jednání při následování společných cílů.

Všechny tyto vlastnosti jsou hodnoceny na pětibodové stupnici, přičemž jedním bodem jsou označeny špatné vlastnosti a pěti body ty nejsilnější vlastnosti daného přístupu [9].

4.2.1 Client-side kompozice

V případě client-side kompozice, kde aplikační prostředí načítá jednotlivé micro frontendy uvnitř sebe, by tyto micro frontendy měly mít jako vstupní bod soubor HTML nebo JavaScript, aby mohlo aplikační prostředí dynamicky připojit DOM uzly v případě HTML souboru nebo inicializovat javascriptovou aplikaci, pokud je vstupem soubor JavaScriptu. Dále je možné použít kombinaci iframů k načtení různých micro frontendů nebo transkluzivní mechanismus (zahrnutí části nebo celého dokumentu do jiných dokumentů) na straně klienta prostřednictvím techniky client-side include [2,9,47].

4.2.1.1 Module Federation

Nový nativní plugin Module Federation se objevil s vydáním Webpacku 5. Umožňuje načítání bloků javascriptového kódu synchronně nebo asynchronně, což znamená, že může více vývojářů nebo týmů pracovat izolovaně a použít lazy-loading pro načítání různých javascriptových bloků na pozadí za běhu.

Skládá se ze dvou hlavních částí. Z hostitele, který představuje kontejner jednoho nebo více načtených micro frontendů nebo knihoven a ovladače, který představuje micro frontend nebo knihovnu, která bude do hostitele načtena za běhu.

Module Federation dokáže jednoduše vystavovat různé micro frontendy nebo dokonce sdílené knihovny, což umožňuje jednoduchou asynchronní integraci. Díky tomu je možné importovat vzdálené micro frontendy a sestavit pohled tak, jak je potřeba. Dalším rysem je možnost sdílení externí knihovny napříč několika micro frontendy, a to bez obav z potencionálních konfliktů za běhu. Pokud jsou knihovny sdíleny mezi více micro frontendy, Module Federation načte pouze jednu verzi pro všechny micro frontendy, které knihovny používají.

Může být použit, pokud je potřeba provozovat aplikaci plně na klientské straně, tak i v případě použití vykreslování na straně serveru.

Velká jednoduchost sdílení kódu mezi projekty je také nejslabším místem pluginu. Při práci v nedisciplinovaném týmu může mít sdílení knihoven, snippetů a micro frontendů napříč více pohledy za následek velmi komplikovanou architekturu pro údržbu, a to díky bezproblémové integraci [9,21,22].

Tabulka 2 Hodnocení kompozice s použitím Module Federation

Charakteristika	Skóre (1 = nejnižší, 5 = nejvyšší)
Nasaditelnost	4/5
Modularita	4/5
Jednoduchost	5/5
Testovatelnost	4/5
Výkon	4/5
Vývojářské zkušenosti	5/5
Škálovatelnost	5/5
Koordinace	3/5

Zdroj: Mezzalira, Luca. Building Micro-Frontends. O'Reilly Media, Inc., 2021.

4.2.1.2 Webové komponenty

Webové komponenty jsou sadou API rozhraní webové platformy, která umožňuje vytvářet vlastní opakovaně použitelné a zapouzdřené HTML tagy pro použití na webových stránkách nebo aplikacích. Mají zajímavé vlastnosti, díky kterým jsou webové komponenty vhodným řešením pro budování micro frontendové architektury.

Umožňují zapouzdření stylů do webových komponent a vytváření sdílených knihoven pro micro frontendové projekty, které používají stejný nebo odlišný UI framework. Všechny majoritní UI frameworky, jako je React, Vue a Angular, webové komponenty podporují.

Webové komponenty se skládají ze tří hlavních technologií, které lze společně použít k vytvoření vlastních elementů se zapouzdřenými funkcemi, které lze bez kolizí opakovaně použít na libovolných místech. Mezi tři hlavní technologie patří custom elements, shadow DOM a HTML šablony.

Custom elements jsou rozšířením HTML komponent, které lze použít jako kontejnery micro frontendů, což umožňuje komunikaci s vnějším světem, například prostřednictvím callbacků nebo událostí.

Shadow DOM je sada API rozhraní JavaScriptu pro připojení zapouzdřeného stínového DOMu k elementu, který je vykreslen odděleně od hlavního DOMu. Díky tomu lze zachovat všechny funkce elementu soukromé, takže je vyloučena kolize s jinými částmi dokumentu.

HTML šablony umožňují psát šablony, které lze opakovaně použít jako základ struktury vlastních elementů.

```
<html>
  <body>
    <div id="app"></div>

    <script src="https://list.domain.com/bundle.js"></script>
    <script src="https://detail.domain.com/bundle.js"></script>

    <script type="text/javascript">
      const webComponentsByRoute = {
        '/': 'micro-frontend-list',
        '/detail': 'micro-frontend-detail'
      };
      const webComponentType =
webComponentsByRoute[window.location.pathname];

      const app = document.getElementById('app');
      const webComponent = document.createElement(webComponentType);
      app.appendChild(webComponent);
    </script>
  </body>
</html>
```

Ukázka kódu 1 Příklad použití webových komponent

Webové komponenty jsou pro micro frontendovou architekturu užitečné hlavně díky vlastním elementům, které se používají jako kontejner micro frontendu, zatímco shadow DOM umožňuje zapouzdření stylů micro frontendu, aniž by přepsaly styly jiného micro frontendu [9,25,26].

Tabulka 3 Hodnocení webových komponent

Charakteristika	Skóre (1 = nejnižší, 5 = nejvyšší)
Nasaditelnost	4/5
Modularita	3/5
Jednoduchost	4/5
Testovatelnost	4/5
Výkon	4/5
Vývojářské zkušenosti	4/5
Škálovatelnost	5/5
Koordinace	3/5

Zdroj: Mezzalira, Luca. Building Micro-Frontends. O'Reilly Media, Inc., 2021.

4.2.1.3 Iframy

Jedním z nejjednodušších přístupů ke kompozici aplikací v prohlížeči je právě iframe. Poskytují izolaci mezi micro frontendy, kterou žádné jiné řešení nabídnout nemůže.

Iframe je vložený rámec, který se používá uvnitř webové stránky k načtení jiného dokumentu HTML. Pokud je potřeba reprezentovat micro frontend jako zcela nezávislý artefakt od zbytku aplikace, iframy poskytují jednu z nejsilnějších izolací, kterou v prohlížeči lze mít.

Rámec iframe poskytuje podrobnou kontrolu nad tím, co v něm může běžet. Méně privilegovaná implementace pomocí atributu sandbox (ukázka kódu 2) zabraňuje spuštění jakékoliv logiky JavaScriptu nebo odeslání formulářů.

```
<iframe sandbox src="https://domain.com/component"/>
```

Ukázka kódu 2 Příklad použití iframu s atributem sandbox

Iframy využívá například micro frontendový framework Luigi pro vytváření intranetových aplikací.

Použití iframů se u spotřebitelských webů důrazně nedoporučuje, protože iframy jsou z hlediska výkonu opravdu špatnou volbou. Jsou velmi náročné na procesor, zvláště pokud je v jednom pohledu použito více iframů [3,9,23,24].

Tabulka 4 Hodnocení iframů (horizontální rozdělení)

Charakteristika	Skóre (1 = nejnižší, 5 = nejvyšší)
Nasaditelnost	5/5
Modularita	3/5
Jednoduchost	3/5
Testovatelnost	3/5
Výkon	2/5
Vývojářské zkušenosti	3/5
Škálovatelnost	5/5
Koordinace	3/5

Zdroj: Mezzalira, Luca. Building Micro-Frontends. O'Reilly Media, Inc., 2021.

4.2.2 Edge-side kompozice

V případě edge-side kompozici je pohled sestaven na úrovni CDN. Mnoho poskytovatelů CDN poskytuje možnost použití značkovacího jazyka Edge Side Include (ESI), který je založen na XML. Používá se pro sestavení různých HTML fragmentů do HTML stránky a doručení finálního výsledku klientovi. ESI umožňuje škálování webové infrastruktury za účelem využití velkého počtu přístupových bodů, které CDN síť poskytuje ve srovnání s omezeným množstvím kapacity datového centra, na kterém je běžně hostována většina softwaru.

Nevýhodou ESI je její různorodý způsob implementace poskytovateli CDN. Strategie multi-CDN, stejně jako přenos kódu od jednoho poskytovatele k druhému, by proto mohla vést k mnoha refaktorům a nové logice k implementaci [9,17].

```
<html>
 <body>
  <esi:include src="https://www.domain.com/MFE_1.html"/>
  <esi:include src="https://www.domain.com/MFE_2.html"/>
 </body>
</html>
```

Ukázka kódu 3 Příklad použití ESI

Implementaci ESI lze vidět ve zdrojovém kódu výše. Prohlížeč tag esi nikdy neuvidí, protože tyto tagy budou nahrazeny obsahem z dané URL adresy [18].

Tabulka 5 Hodnocení edge-side kompozice (horizontální rozdělení)

Charakteristika	Skóre (1 = nejnižší, 5 = nejvyšší)
Nasaditelnost	3/5
Modularita	4/5
Jednoduchost	2/5
Testovatelnost	3/5
Výkon	3/5
Vývojářské zkušenosti	2/5
Škálovatelnost	4/5
Koordinace	3/5

Zdroj: Mezzalira, Luca. Building Micro-Frontends. O'Reilly Media, Inc., 2021.

4.2.3 Server-side kompozice

Horizontálně rozdělené architektury se server-side kompozicí jsou nejflexibilnějšími a nejvýkonnějšími řešeními dostupnými v micro frontendovém ekosystému, a to díky cloudu, který poskytuje dokonalé prostředí pro vývojáře, kteří se zaměřují více na tok hodnot než na provoz infrastruktury.

Ke kompozici může dojít za běhu nebo při kompilaci. V tomto případě původní server skládá pohled načtením všech různých micro frontendů a sestavením finální stránky.

Server-side kompozice vytváří konečný pohled na straně serveru, kde je možné řídit rychlost konečného výstupu pomocí technik, jako je ukládání do mezipaměti v různých vrstvách, ve službě, v paměti nebo CDN.

Volí se většinou v případě, jestliže je kladen velký důraz na SEO, protože tato technika zrychluje dobu načítání stránky, která je plně vykreslena bez potřeby logiky JavaScriptu. Lepším výsledkům ve vyhledávači pomáhá taktéž server-side rendering (SSR), který odesílá plně vykreslenou stránku klientovi, čímž se značně zrychlí prvotní načtení stránky oproti vykreslení na straně klienta. Rychlost načtení stránky je jeden z aspektů, které vyhledávače berou v potaz [2,9,19].

```
<html>
  <body>
    <!--# include virtual="/header.html" -->
    <!--# include virtual="/footer.html" -->
  </body>
</html>
```

Ukázka kódu 4 Příklad použití SSI

Zdrojový kód výše znázorňuje implementaci server-side kompozice s použitím server-side includes (SSI).

```
<html>
  <body>
    <header>
      Hlavička vykreslená na straně serveru
    </header>
    <footer>
      Patička vykreslená na straně serveru
    </footer>
  </body>
</html>
```

Ukázka kódu 5 Příklad použití SSI a SSR

Pokud je použito vykreslení na straně serveru, nahradí server SSI komentář za obsah, na který se odkazoval (ukázka kódu 5). Poté je takto vykreslená stránka předána klientovi [2].

Vzhledem k tomu, že jsou různé frontendy sloučeny do jedné stránky, neexistuje způsob, jak zajistit správnou izolaci. V důsledku toho mohou být skripty a šablony stylů ve vzájemném konfliktu. Výskyt konfliktů může být častější v případě nedostatku vhodných nástrojů pro ladění. Právě pro tyto účely byly zavedeny frameworky, jako je Podium, Mosaic a Puzzle.js. Mosaic je pravděpodobně jeden z nejznámějších, protože byl jeden z prvních open source frameworků, které přijaly tento styl architektury [9,20].

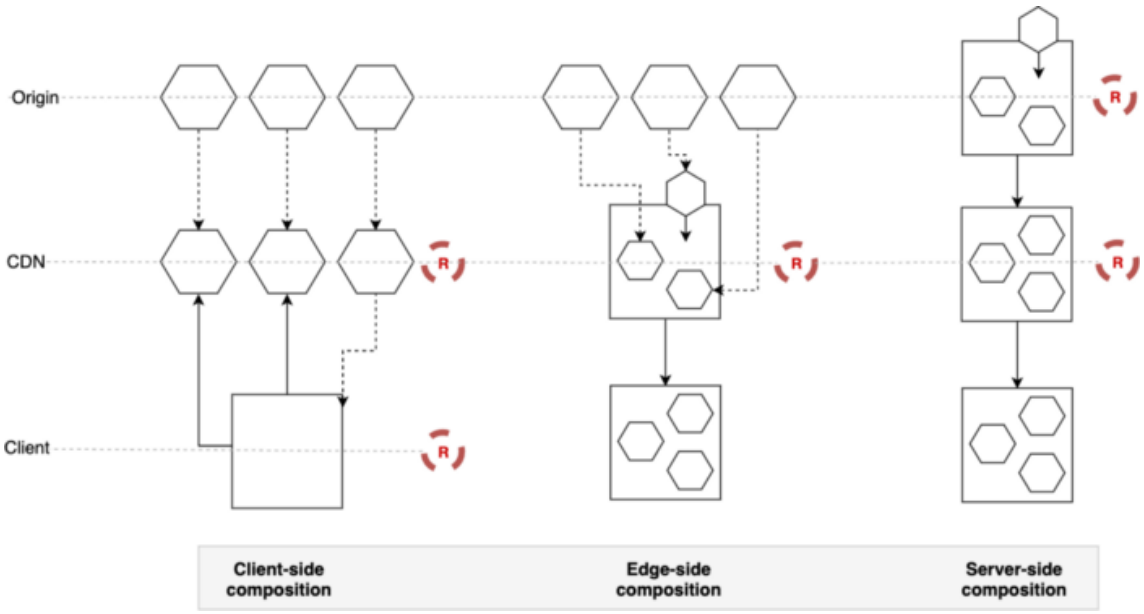
Tabulka 6 Hodnocení server-side kompozice (horizontální rozdělení)

Charakteristika	Skóre (1 = nejnižší, 5 = nejvyšší)
Nasaditelnost	4/5
Modularita	5/5
Jednoduchost	3/5
Testovatelnost	4/5
Výkon	5/5
Vývojářské zkušenosti	3/5
Škálovatelnost	3/5
Koordinace	3/5

Zdroj: Mezzalira, Luca. Building Micro-Frontends. O'Reilly Media, Inc., 2021.

4.3 Routování micro frontendů

Volba způsobu routování aplikace je striktně spojena s volbou přístupu ke kompozici micro frontendů v projektu. Stejně jako kompozice, tak i routování může probíhat client-side, edge-side nebo server-side způsobem (obrázek 4) [9,14].



Obr. 4 Routování micro frontendů dle kompozice
Zdroj: https://medium.com/@lucamezzalira/micro-frontends-decisions-framework-ebcd22256513

4.3.1 Client-side routování

V případě routování na straně klienta jsou načítány micro frontendy podle stavu uživatele. Příkladem může být načtení ověřené oblasti aplikace, pokud byl uživatel již autentizován nebo načtení pouze vstupní stránky, jedná-li se o první návštěvu aplikace ze strany uživatele.

Pokud je použito aplikační prostředí, které načte micro frontend jako SPA, je tak aplikační prostředí zodpovědné za vlastnictví logiky směrování. Aplikační prostředí tedy nejdříve načte konfiguraci routování a poté rozhodne, který micro frontend bude nahrán.

Perfektní případ použití je u složitého routování, například pokud jsou micro frontendy založeny na ověřování, geolokalizaci nebo jakékoliv jiné sofistikované

logice. U vícestránkových webů mohou být micro frontendy načteny prostřednictvím transkluze na straně klienta [9,14].

4.3.2 Edge-side routování

Edge-side routování je založeno na URL adrese stránky, zatímco CDN má na starosti obsluhu stránky požadované sestavením micro frontendů pomocí transkluze. Důležitým faktem při výběru této varianty je to, že v tomto případě není mnoho prostoru pro vytváření chytrého směrování [14].

4.3.3 Server-side routování

Pokud je zvoleno routování na straně serveru, leží celá aplikační logika na aplikačních serverech. Je nutné vzít v úvahu, že škálování infrastruktury může být netriviální, obzvlášť pokud je řízen nárazový provoz s velkým množstvím požadavků za sekundu (RPS). Servery musí být schopny držet krok se všemi požadavky a velmi rychle horizontálně škálovat. Každý aplikační server pak musí být schopen načíst micro frontendy pro skládající se stránku, která má být zobrazena. Tento problém může být zmírněn s pomocí CDN. V tomto případě není možné se ve velké míře spoléhat na CDN, pokud se pracuje s dynamickými nebo personalizovanými daty, protože by tato data byla zastaralá nebo by nebyla personalizovaná [9,14].

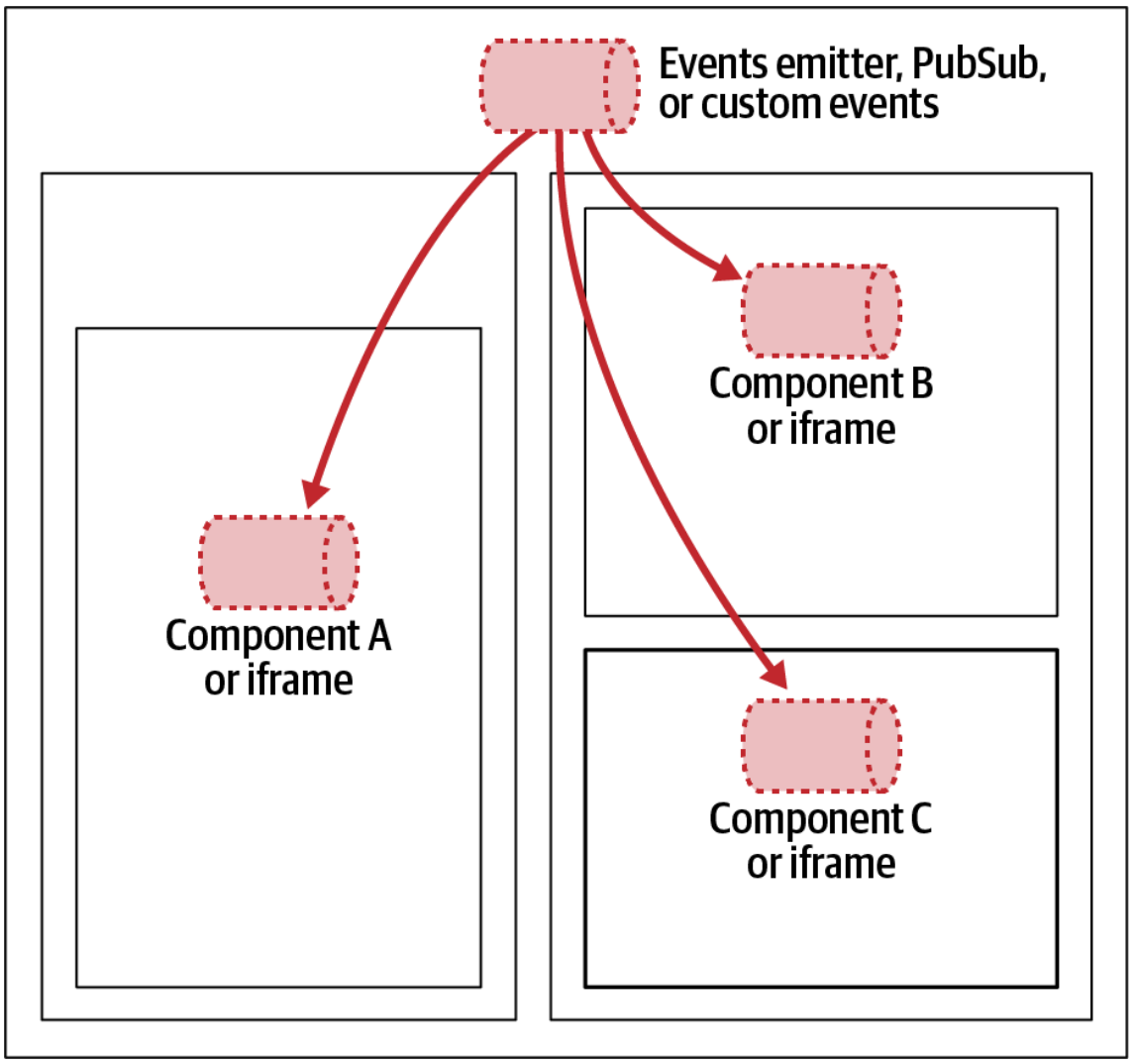
4.4 Komunikace mezi micro frontendy

Komunikace mezi micro frontendy by v ideálním případě nemusela vůbec existovat, protože by jednotlivé micro frontendy byly zcela oddělené. Pokud je na jedné stránce více micro frontendů, nemusí být snadné spravovat konzistentní uživatelské rozhraní. To platí také u komunikace mezi micro frontendy, které jsou vlastněny různými týmy. Aby nebyl porušen princip nezávislého nasazování, neměly by o sobě jednotlivé micro frontendy vůbec vědět.

Existuje několik možností, jak informovat ostatní micro frontendy, že došlo k nějaké události. Je však doporučeno, aby komunikace mezi micro frontendy probíhala formou zpráv nebo událostí a předešlo se jakémukoliv sdílnému stavu. Stejně jako sdílení databáze napříč mikroslužbami, jakmile jsou sdíleny datové

struktury a doménové modely, vytváří se obrovské množství propojení a je poté velmi obtížné provádět změny.

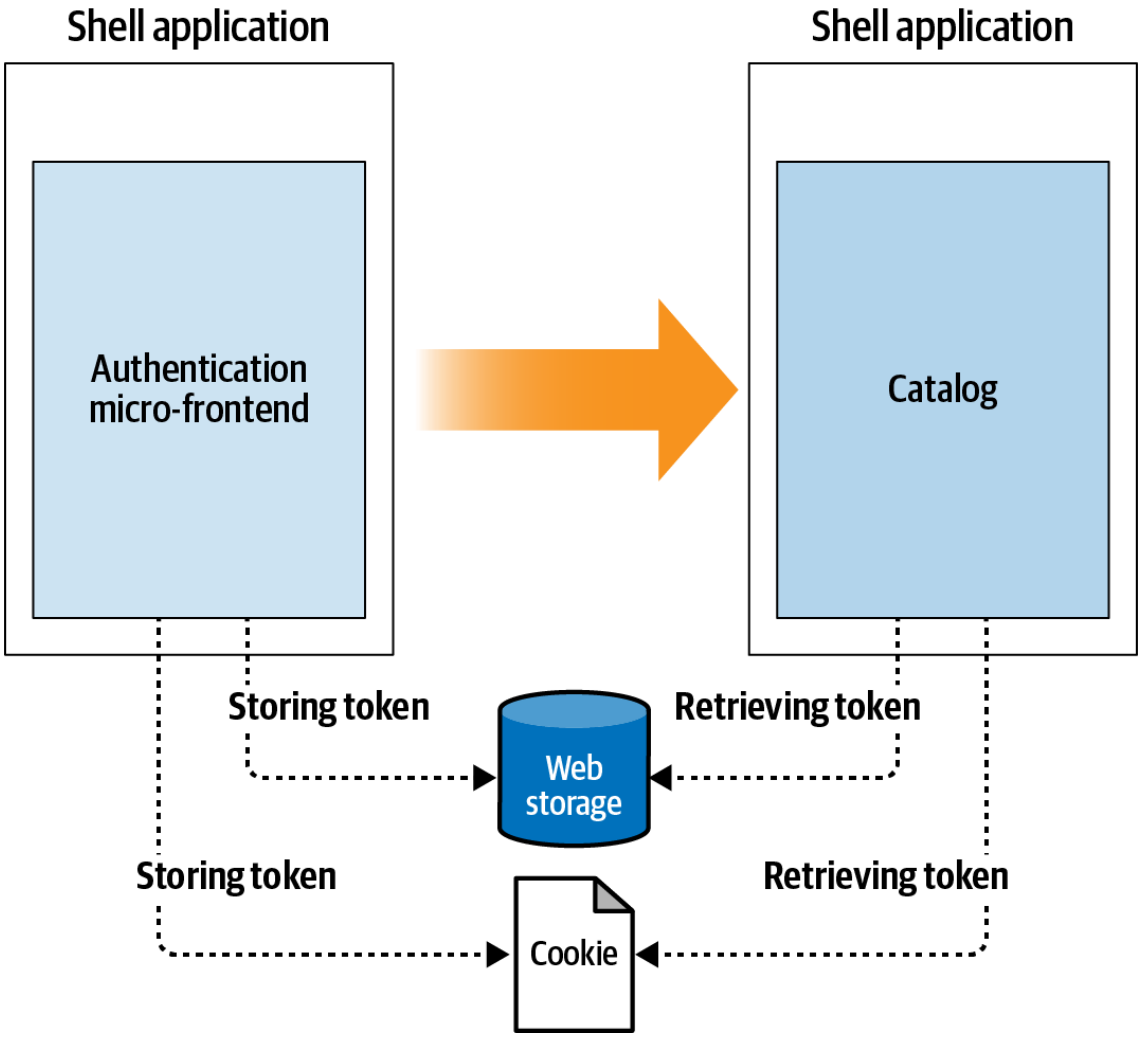
Prvním způsobem komunikace je sběrnice událostí, která umožňuje odděleným komponentám komunikovat mezi sebou prostřednictvím událostí odeslaných přes sběrnici do každého micro frontendu a oznámit událost každému zvlášť. Pokud se některé micro frontendy o odeslanou událost zajímají, mohou ji poslouchat a reagovat na ni. Pro vložení sběrnice událostí je potřeba kontejner, který vytvoří instanci sběrnice a vloží ji do všech micro frontendů na stránce (obrázek 5) [3,9,14].



Obr. 5 Komunikace přes event emmiter a custom events
Zdroj: Mezzalira, Luca. Building Micro-Frontends. O'Reilly Media, Inc., 2021.

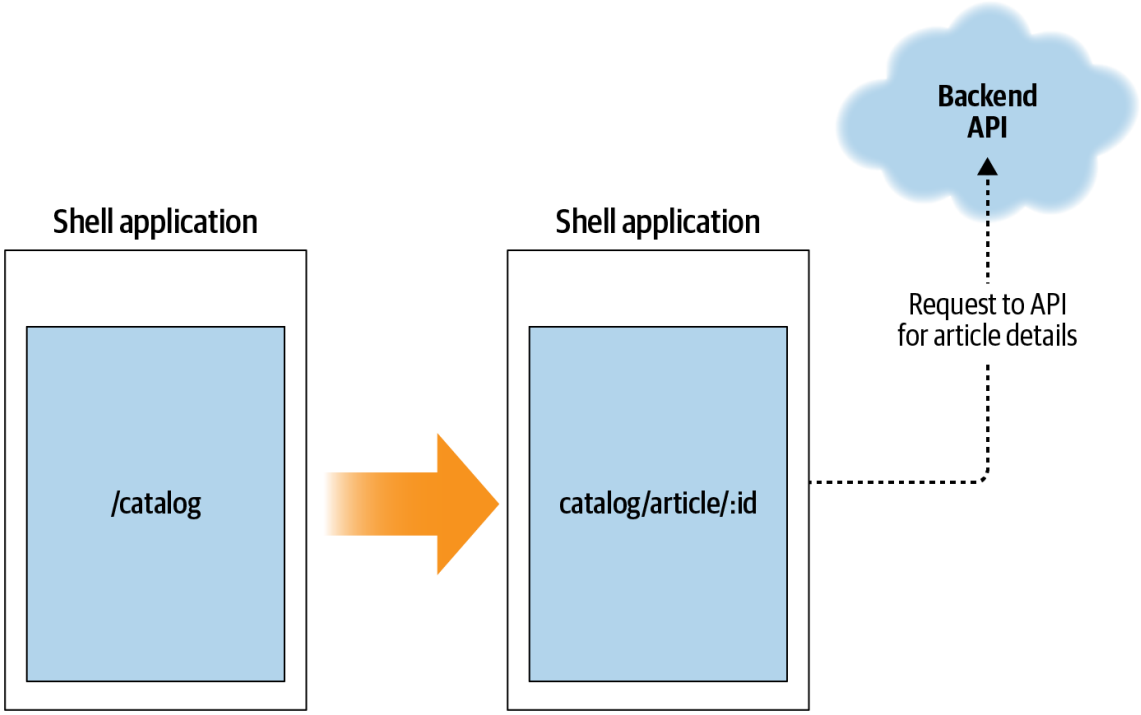
Dalším řešením jsou Custom Events. Jsou to normální události s vlastním tělem, takže je možné definovat vlastní řetězec, který událost identifikuje. Custom Events by měly být odesílány prostřednictvím objektu, který je dostupný všem micro frontendům, jako je například window objekt. Pokud jsou micro frontendy implementovány pomocí iframů, může se použitím sběrnice událostí předejít problémům s výběrem správného window objektu, protože každý iframe má svůj vlastní window objekt [9].

Lze také použít úložiště na straně klienta (obrázek 6). Sem spadá například Cookies, Local Storage a Session Storage [9,29].



Obr. 6 Komunikace přes web storage a cookies
Zdroj: Mezzalira, Luca. Building Micro-Frontends. O'Reilly Media, Inc., 2021.

Ke komunikaci lze použít také adresní řádek, přes který lze předávat data pomocí query strings (obrázek 7). Tento způsob není vhodný pro předávání citlivých dat, jako jsou hesla a dalších osobních údajů o uživateli [9].



Obr. 7 Komunikace přes query strings
Zdroj: Mezzalira, Luca. Building Micro-Frontends. O'Reilly Media, Inc., 2021.

Dalším způsobem komunikace je použití stavového kontejneru Redux, který umožňuje mít jediné globální sdílené úložiště pro celou aplikaci. Pokud však má být každý micro frontend samostatnou aplikací, pak by měl mít svůj vlastní Redux store. Dokumentace Reduxu uvádí, že je možné mít oprávněně více storů, pokud je Redux aplikace izolována jako komponenta ve větší aplikaci [3,27,28].

4.5 Stylování

Jazyk CSS je ze své podstaty globální, využívá dědičnosti a kaskáduje. U micro frontendových aplikací proto nastávají nemalé potíže s ovládnutím stylů napříč všemi komponentami. Pokud by měl první vývojový tým ve svých stylech nadefinovaný nadpis h1 s bílou barvou textu a druhý tým by měl stejný nadpis

nastavený na černou barvu a oba micro frontendy těchto týmů byly sestaveny do jedné stránky, dojde ke kolizi stylů [3].

V průběhu několika let bylo představeno několik přístupů, aby byly CSS více pod kontrolou vývojářů. Mezi takové přístupy patří například konvence pojmenování zvaná BEM. Ten by měl zajistit, aby se selektory použily pouze na místech, kde jsou zamýšleny [3].

```
.bem {

}
.bem__element {

}
.bem--modifier {

}
```

Ukázka kódu 6 BEM zápis CSS

Další variantou je využití možnosti vnoření selektorů u CSS preprocesorů (Less, Sass), nicméně tento postup není úplně ideální z hlediska udržitelnosti a možnosti úprav [3].

Novějším přístupem je aplikace všech stylů programově pomocí CSS modulů nebo jedné z různých CSS-in-JS knihoven, což zajišťuje, že jsou styly přímo aplikovány pouze na místech, která vývojář zamýšlel. CSS-in-JS knihovny ve většině případů podporují například scoped styly, které generují unikátní názvy tříd stylů. Všechny styly jsou díky tomu zapouzdřeny bez možnosti ovlivnění jakéhokoliv stylu definovaného mimo komponentu [3,35].

```
<style scoped>
  .scoped-example {

  }
</style>

/* CSS s vygenerovaným unikátním názvem */
.scoped-example[data-v-a1b2c3d] {

}
```

Ukázka kódu 7 Scoped CSS ve frameworku Vue

4.6 Micro frontend architektura v praxi

Tato kapitola popisuje různé možnosti implementace micro frontendové architektury v několika velkých organizacích, jako je Spotify, IKEA, Skyscanner, OpenTable a Zalando.

4.6.1 Spotify

Spotify je online streamovací mediální služba, která používá ve své desktopové aplikaci micro frontendy, které využívají iframy pro kompozici různých částí stejného pohledu. Komunikace mezi těmito iframy probíhá prostřednictvím sběrnice událostí, která skvěle odděluje jednotlivé části aplikace [30,31].

4.6.2 IKEA

IKEA, nábytkářská společnost, implementovala micro frontendy s použitím edge-side includes společně s client-side includes. Klíčem k úspěšnému používání micro frontend architektury ve společnosti IKEA je udržet týmy v malém počtu lidí. Webový architekt pro společnost IKEA, Gustaf Nilsson Kotte, věří, že týmy s počtem deseti až dvanácti lidí spolu fungují velmi dobře. Pokud je počet členů týmu větší, ztrácí tým na efektivitě [30,31,32].

4.6.3 Skyscanner a OpenTable

Společnosti Skyscanner a OpenTable používají micro frontendový framework OpenComponents. Ten používá registr, který shromažďuje všechny dostupné komponenty zapouzdřující data a uživatelské rozhraní. Výstupem je HTML fragment, který je možné zapouzdřit do libovolné HTML šablony [9,31,33].

4.6.4 Zalando

Zalando je e-commerce společnost, která také stojí v čele projektu Mosaic. Jedná se o sadu služeb, knihoven a specifikací, které definují, jak se mezi sebou budou komponenty ovlivňovat, aby podpořily micro frontend architekturu pro velké weby [20,30,34].

4.7 Výhody

V této kapitole je uvedeno shrnutí výhod, které micro frontend architektura dokáže nabídnout.

4.7.1 Postupné vylepšování

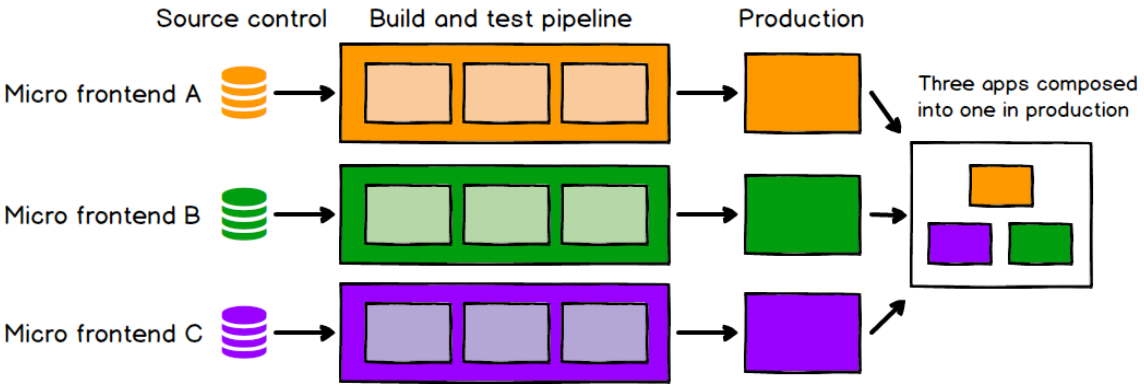
Možnost postupného vylepšování aplikace je jedna z hlavních motivací pro mnoho organizací, aby tuto architekturu implementovali i pro své projekty. Monolity nejde neustále vylepšovat a obohacovat o nové funkce, aniž by se vývoj nedostal do stádia, kdy je vhodné provést celkový přepis. U micro frontendu jsou moduly rozděleny mezi týmy a každý z nich má na starost end-to-end správu i vývoj svých modulů, a to zcela nezávisle na ostatních týmech. Díky tomu se nabízí větší flexibilita, efektivita a kratší doba pro vykonání implementace změn a aktualizací [3,8].

4.7.2 Jednoduché a oddělené zdrojové kódy

Zdrojový kód jednoho monolitického frontendu je přirozeně mnohem větší než zdrojový kód každého jednotlivého micro frontendu. Tyto jednotlivé zdrojové kódy jsou pak mnohem čistší, jednodušší a mnohem lépe se s nimi pracuje. Redukuje riziko neúmyslného a problematického propojení komponent, které by o sobě navzájem neměly vědět. Micro frontendová architektura se také snaží vývojáře tlačit k tomu, aby se více zajímali o správné chování komponent a datovou komunikaci mezi nimi [3,8].

4.7.3 Autonomní nasazení do produkce

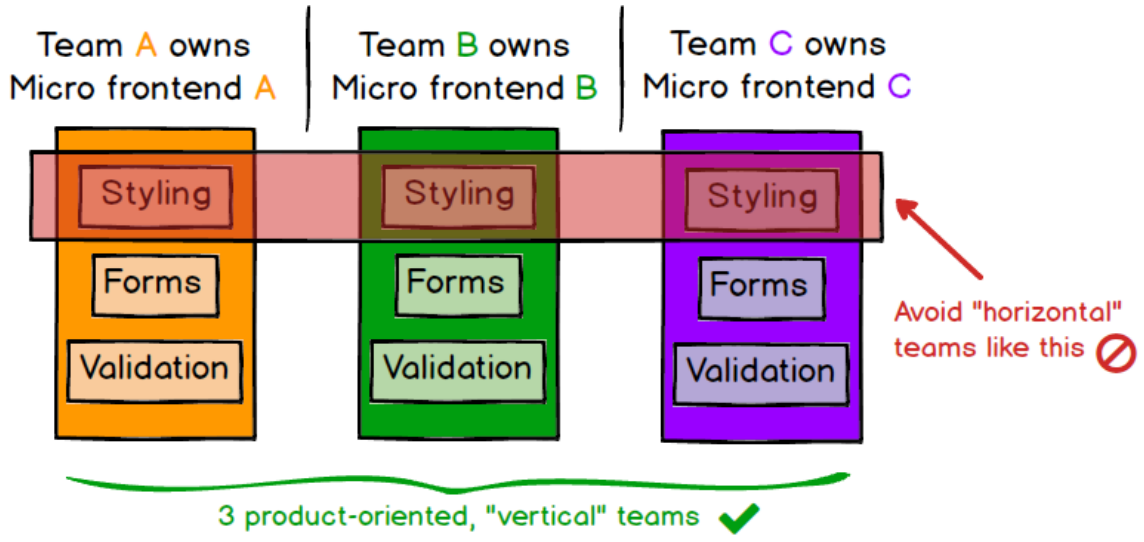
Další klíčovou výhodou micro frontendové architektury je nezávislé nasazení do produkčního prostředí. Jelikož je snížen rozsah jednotlivého nasazení do produkce, dojde také ke snížení souvisejících rizik spojených s procesem. Bez ohledu na to, jak a kde je zdrojový kód hostován, každý micro frontend by měl mít svůj vlastní nezávislý CI kanál, který zajistí sestavení, testování a nasazení do produkce (obrázek 8) [3,8].



Obr. 8 Autonomní nasazení do produkce
Zdroj: https://martinfowler.com/articles/micro-frontends.html

4.7.4 Nezávislé týmy

Jestliže je možné oddělit jednotlivé zdrojové kódy aplikace i cykly nasazení do produkce, dostáváme se k plně nezávislým týmům, které defacto mohou vlastnit část produktu, a to od návrhu až po produkci. Tyto jednotlivé týmy mají k dispozici vše k tomu, aby dokázali rychle a efektivně splnit požadavky klienta. Důležitou podmínkou je rozdělení týmů podle svislých segmentů (obrázek 9), jinak by nebylo možné zajistit end-to-end pokrytí vývoje jedním týmem [3].



Obr. 9 Rozdělení týmů
Zdroj: https://martinfowler.com/articles/micro-frontends.html

4.7.5 Univerzální design

Vzhledem k tomu, že jsou micro frontendy rozdělené do nezávislých týmů, které pracují na různých komponentách, je možné docílit mnohem univerzálnějšího designu než u monolitického frontendu. Každý tým tak může využít plně svůj potenciál a možnosti, aby dokázal nabídnout vždy maximální kvalitu [8].

4.8 Nevýhody

Přechozí kapitola, popisující výhody micro frontendů, jasně definovala, že micro frontend je o tom, aby se dokázaly vytvořit plně nezávislé týmy, které mají k dispozici vše potřebné k vývoji nových funkcí pro zákazníky. Taková nezávislost je opravdu mocná věc, která má však i svá úskalí [1].

4.8.1 Redundance

Redundance se může objevit v každém systému a micro frontend architektura není výjimkou. Vzhledem k tomu, že v micro frontend architektuře vedle sebe pracuje více týmů, které mají svůj vlastní technologický stack, dochází tak k duplicitám a nadbytečnostem.

Další příčina vzniku redundance je nastavení a správa vlastního aplikačního serveru, kanál pro CI a případně CSS a JavaScript. Vývojáři by se proto měli snažit o její minimalizaci, což povede ke zvýšení efektivity a konzistence celé aplikace.

Pokud nastane nějaká kritická chyba v používané knihovně, musí ji každý tým opravit zvlášť, což s sebou přináší duplicitní kód a výrazné časové zatížení, jelikož nelze chybu opravit na jednom centrální místě [1].

4.8.2 Konzistence

Aby bylo možné označit vývojové týmy za plně nezávislé, bylo by nutné použít vlastní databázi pro každý tým. Může však nastat situace, kdy jeden tým potřebuje data z druhého týmu. Tuto situaci lze řešit například sběrnicí událostí nebo pomocí feedu.

První tým vlastní data a ostatní týmy tato data pravidelně replikují. V případě nějaké havárie jednoho z týmů mohou ostatní týmy fungovat bez problému se stálým přístupem k lokálním datům. Replikace dat však vyžadují svůj

čas a mají určitou latenci, proto mohou být změny v datech na krátkou dobu nekonzistentní [1].

4.8.3 Heterogenita

Jestliže má každý tým vlastní technologický stack, komplikuje to případný přechod vývojářů z jednoho týmu do druhého nebo sdílení osvědčených postupů ve vývoji. Není však nutné, aby každý tým měl odlišný technologický stack. I kdyby všechny vývojové týmy používaly úplně stejné technologie, všechny výhody micro frontendu by zůstaly stále platné [1].

4.8.4 Více kódu na straně frontendu

Tento nedostatek lze logicky odvodit z principu fungování micro frontend architektury. Vývoj jednotlivých micro frontendů, které mohou běžet izolovaně, přináší značné množství kódu navíc. Z toho důvodu je nutné sledovat výkonnostní stránku webu už od samostatného začátku vývoje [1].

4.9 Případy vhodného použití

Stejně jako u jiných možných přístupů, tak ani micro frontend architektura není univerzálním řešením pro všechny možné situace. Při výběru je důležité porozumět výhodám a nevýhodám, které tento přístup nabízí. Při správném rozhodnutí dokáže micro frontend architektura přinést do vývoje pořádek, efektivitu, organizaci a zjednodušení. V opačném případě dokáže udělat vývoj mnohem složitějším [1].

4.9.1 Střední až velké projekty

Pokud na aplikaci pracuje malá skupina lidí, není škálování aplikace žádný velký problém, a to hlavně díky micro frontend architektuře, která škálování projektů velmi usnadňuje. Čím většího počtu členů bude skupina nabývat, tím více se komplikuje rozhodování a také se přestává zvládat režie komunikace mezi členy týmu. Ideální počet členů týmu by měl být přibližně mezi pěti až deseti lidmi. Jakmile se dostane přes maximální hodnotu, mělo by se zvážit, zda se nevyplatí tým rozdělit [1].

4.9.2 Nejlépe funguje na webu

Myšlenky, na kterých je postavena micro frontend architektura, nejsou nijak omezeny na konkrétní platformu. Fungují však nejlépe na webu, a to především díky jeho otevřenosti [1].

4.9.2.1 Nativní monolitické aplikace

Nativní aplikace pro platformy jako je Android nebo iOS jsou od návrhu monolitické, takže není možné vytvářet a nahrazovat funkce za běhu. Pokud je třeba aktualizovat nativní aplikaci, musí se vytvořit jeden balík aplikace, který je odeslán do procesu kontroly společnosti Google nebo Apple. Tomuto postupu je možné se vyhnout, a to načtením částí aplikace z webu. Obecně se dá říct, že pokud je cílovou platformou web, micro frontend architektura se zde může skvěle hodit. Jestliže je cílovou platformou nativní aplikace, bude nutné přijmout určitá omezení [1].

4.9.2.2 Několik frontendů na tým

Neplatí zde omezení jednoho vývojového týmu na jeden frontend. Běžně lze najít případy, kdy jsou vyvíjeny dva frontendy pro aplikaci, front-office a back-office [1].

4.10 Případy nevhodného použití

Micro frontend architektura umožňuje efektivní a škálovatelný vývoj. Pokud je k dispozici jen malá skupina vývojářů a komunikace funguje bez problému, zavedení micro frontendů nebude takovým přínosem, jak by se očekávalo.

Další důležitou věcí je správné vertikální rozdělení celé aplikace mezi vývojové týmy. Jakmile není zřejmé, který tým je zodpovědný za implementaci určité funkce, nastává mezi týmy nejistota a dlouhé diskuze. Při správném vertikálním rozdělení by nemělo docházet k nejasným nebo překrývajícím se týmovým úlohám.

Nelehkým úkolem pro jeden tým může být i vývoj spousty různých aplikací a nativních uživatelských rozhraní [1].

5 Ukázková aplikace

Zkoumání tématu micro frontend architektury bylo velmi zajímavé, ale v určitých oblastech také složité na porozumění. Z toho důvodu bylo vhodné vyvinout jednoduchou ukázkovou aplikaci postavenou na micro frontend architektuře. Během vývoje aplikace bylo hlavním cílem blíže porozumět metodikám micro frontend architektury a vyzkoušet si na praktickém příkladě tento nastávající moderní trend vývoje komplexních aplikací.

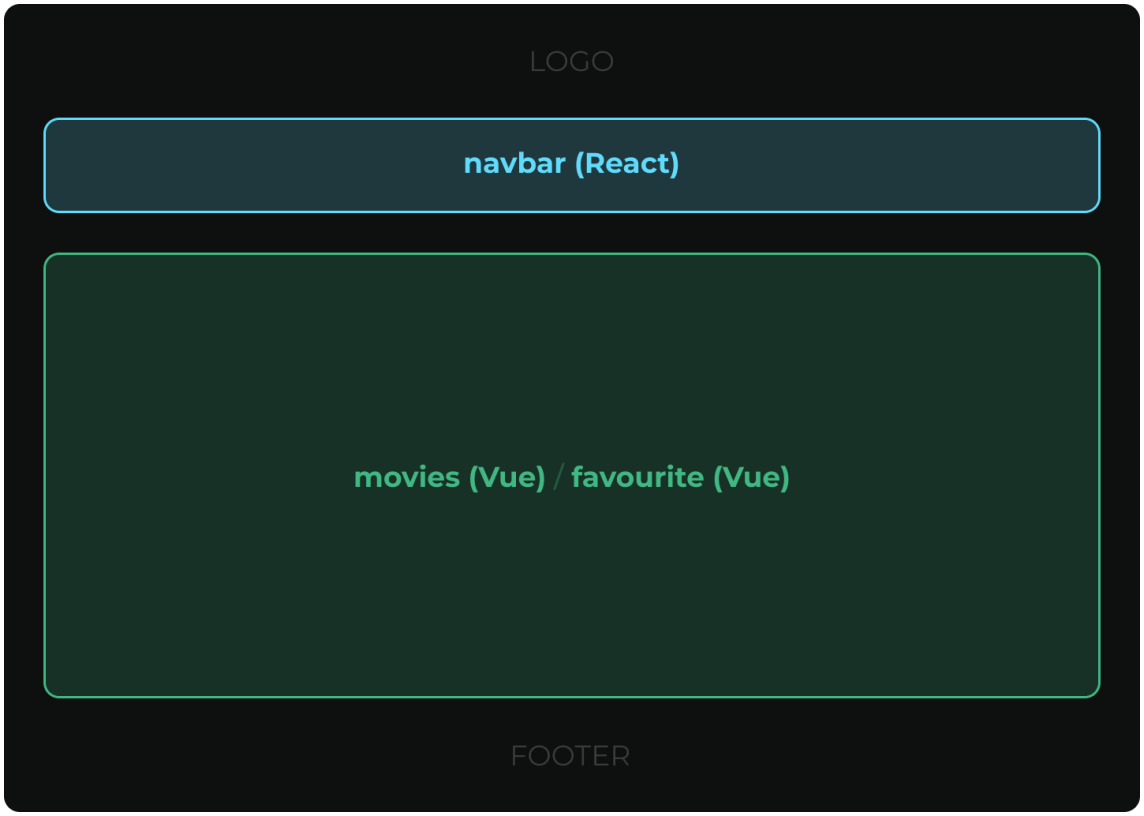
Jedná se o velmi jednoduchou aplikaci, která nesplňuje adekvátní předpoklady pro použití micro frontend architektury. Sloužila jako nástroj k procvičení a lepšímu porozumění během vypracování této práce. Postupy a způsoby implementace technologií nemusí být vždy korektní.

V následujících kapitolách bude podrobně popsáno, o čem aplikace pojednává, co obsahuje, jaké byly použity technologie a jakým způsobem funguje. Dále bude vysvětlena komunikace mezi micro frontendy, stylování, způsob verzování a procesy nasazení do produkce.

5.1 O aplikaci

Ukázková aplikace nese název MFTV (Micro Frontend TV). Jedná se o filmovou knihovnu, ve které je možné přidat uvedené filmy do oblíbených. Ty jsou poté k naleznutí v patřičné sekci.

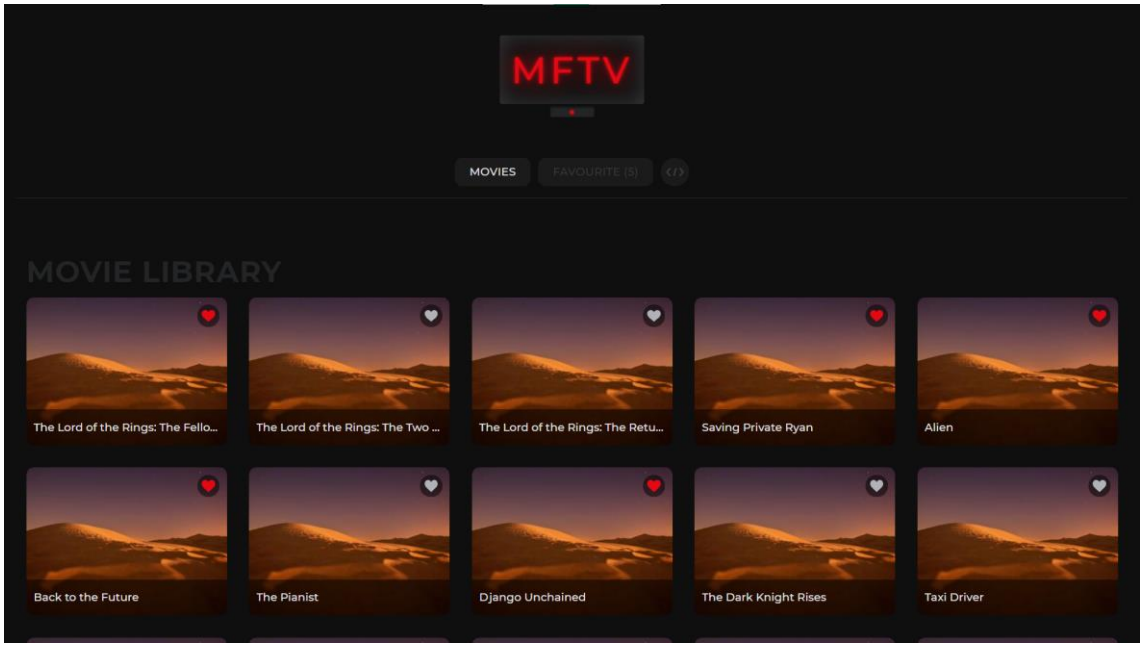
Celá aplikace je složena z pěti částí. První částí je kontejner, který celou aplikaci obaluje a obsahuje veškeré důležité nastavení pro chod projektu včetně indexové souboru. Přechod mezi jednotlivými stránkami zajišťuje micro frontend *navbar* napsaný v React frameworku. Hlavní obsah aplikace zajišťují dva micro frontendy *movies* a *favourite*, které jsou napsané ve frameworku Vue. Poslední část tvoří komponenta *styleguide*, která v sobě uchovává globální CSS pro celou aplikaci. Schéma rozložení aplikace můžete vidět na následujícím obrázku.



Obr. 10 Schéma aplikace MFTV
Zdroj: Vlastní tvorba

Protože se v této aplikaci používají dva různé frameworky, byla použita knihovna Single-spa. Díky této knihovně lze snadno ovládat zapojení a odpojení micro frontendů. Bližším informacím k této knihovně bude vyhrazen prostor v následujících kapitolách.

Aplikace obsahuje pouze dvě možné stránky – domovskou stránku a favourite. *Navbar* micro frontend je zobrazen permanentně, zatímco micro frontendy *movies* a *favourite* se vykreslují v závislosti na aktuální URL adrese. Z toho vyplývá, že vždy běží současně dva micro frontendy napsané v odlišném frameworku – Reactu a Vue.



Obr. 11 Domovská stránka MFTV aplikace
Zdroj: Vlastní tvorba

Na domovské stránce (obrázek 11) je možné přidat nebo odebrat konkrétní film do nebo z oblíbených, takto přidané filmy lze poté nalézt na stránce favourite. Současně je aktuální počet oblíbených filmů zobrazen u navigačního tlačítka favourite.

Posledním prvkem v navigačním menu je inspector micro frontendu. Ten po aktivaci zvýrazní jednotlivé micro frontendy a zobrazí jejich název včetně frameworku, ve kterém je napsán.

5.2 Technologický stack

Vývoj ukázkové aplikace je postaven na velkém množství použitých technologií. Nejdůležitější použité technologie budou popsány v následujících kapitolách.

Výběr technologií pro vývoj této ukázkové aplikace nemá za úkol charakterizovat nejlepší a univerzální stack pro všechny micro frontendové aplikace. Jedná se čistě o osobní preferenci technologií, které byly nějakým způsobem zajímavé, zažité nebo se jednalo o nejlevnější variantu z nabízených možností. Některé technologie byly doporučeny několika různými zdroji, a proto byly zahrnuty do vývojového stacku této aplikace. Především je důležité

připomenout různorodost projektů, z toho důvodu není doporučeno tento výběr aplikovat na všechny typy projektů, jelikož každý projekt má své specifické požadavky na implementaci.

5.2.1 Single-spa

Single-spa je javascriptový framework, který dokáže sestavit, testovat a nasadit několik nezávislých micro frontendů v jedné frontendové aplikaci. Značně usnadňuje použití několika odlišných frameworků v single-page aplikaci. Poskytuje lazy loading aplikací, takže zbytečně nenačítá ostatní aplikace, dokud nejsou potřeba [36].

Tato knihovna byla zvolena především kvůli snadné implementaci různých frameworků na jedné stránce a dalším benefitům, které už pro potřeby projektu nebyly tak důležité. Podle několika různých zdrojů se jednalo o velmi často doporučovanou knihovnu ve spojení s implementací odlišných frameworků, které běží současně na jedné stránce. Stejný problém nastává i v případě této aplikace, na základě čehož bylo zvoleno použití této knihovny.

5.2.2 Webpack

Webpack je ve své podstatě module bundler statických modulů pro moderní javascriptové aplikace. Zpracovává především javascriptové soubory, ale je také schopen různě transformovat nebo balíkovat jakýkoliv zdroj nebo asset. Jedná se o velmi mocný nástroj, nicméně jeho nakonfigurování je poměrně náročný úkol [37].

5.2.3 SystemJS

SystemJS slouží jako dynamický zavaděč modulů. Dokáže importovat moduly za běhu v jakémkoliv známém formátu, který se v dnešní době používá (ES6, AMD, UMD, CommonJS). Je dostatečně chytrý na to, aby dokázal detekovat používaný formát a zacházel s ním odpovídajícím způsobem [38].

Tuto knihovnu ve svém základu používá i single-spa framework.

5.2.4 React

Prvním použitým javascriptovým frameworkem pro vývoj uživatelského rozhraní je React. Umožňuje kompozici uživatelského rozhraní z malých a izolovaných komponent [39].

5.2.5 Vue

Druhým použitým progresivním frameworkem pro vývoj uživatelského rozhraní je Vue. Na rozdíl od jiných monolitických frameworků, jako je například Angular, je Vue navrženo tak, aby se dalo postupně adoptovat. Hlavní knihovna frameworku se totiž zaměřuje pouze na pohledovou vrstvu a lze ji tak snadno vyjmout a integrovat do jiných knihoven nebo existujících projektů. Hodí se také skvěle pro single page aplikace, pokud se použije ve spolupráci s podpůrnými knihovnami a jinými moderními nástroji [40].

Výběr Vue frameworku pro micro frontendy *movies* a *favourite* měl svůj podstatný důvod. Autor práce s tímto frameworkem má největší zkušenosti a také je zde podpora scoped stylů. Scoped styly se zde skvěle uplatní, jelikož jsou výše zmíněné micro frontendy nejnáročnější a nejobjemnější na stylování. Navíc je možné zachovat třídy stylů jednoho micro frontendu a použít je v druhém, protože jsou téměř identické.

5.2.6 GitHub

GitHub je zisková společnost, která nabízí bezplatnou službu hostování pro open source projekty. Disponuje příjemným a přehledným uživatelským rozhraním, takže je vhodný i pro začínající vývojáře. Nabízí řadu užitečných funkcí pro spolupráci. Mezi ty patří například sledování chyb, správa úloh nebo wiki pro každý projekt [41].

Na GitHubu je uloženo všech pět částí aplikace, každá ve svém repozitáři.

5.2.7 Travis CI

Pojmem CI se rozumí praktika častého slučování po malých změnách v kódu. Tento postup vývoje a testování, po menších inkrementacích, pomáhá udržovat aplikaci neustále v dobré kondici.

Veškerý proces CI ukázkové aplikace je řízen platformou Travis CI. Ten podporuje vývojový proces automatickým buildováním, testováním změn v kódu a poskytuje okamžitou zpětnou vazbu o výsledcích daných změn. Další oblastí, kde lze Travis využít, je automatizace dalších částí vývojových procesů, a to díky správě nasazování a notifikací. To znamená, že je možné mít na sobě závislé úlohy, které se provedou na základě aktuální fázi buildu [42].

5.2.8 Amazon S3

Služba Amazon S3 (Simple Storage Service) slouží pro ukládání objektů. Nabízí výbornou škálovatelnost, datovou dostupnost, zabezpečení a výkon. Dokáže pokrýt celou řadu možných případů použití od webových stránek, mobilních aplikací, zálohování a obnovení, archivování, podnikových aplikací, až po IoT zařízení. Poskytuje funkce pro správu, takže je možné optimalizovat, organizovat a konfigurovat přístup k datům dle individuálních obchodních a organizačních požadavků [43].

Nutno podotknout, že se jedná o placenou službu. Pro potřeby ukázkové aplikace bylo možné využít roční bezplatné verze, která je určitým způsobem limitována.

5.2.9 Heroku

Heroku je populární platforma cloudových služeb. Umožňuje vytváření, dodání, monitoring a škálování aplikací. Použití této platformy je velice snadné, takže je vhodnou volbou pro mnoho vývojových projektů.

Protože si Heroku spravuje hardware i servery, mohou se zákazníci zaměřit čistě na vývoj svých aplikací, a ne na infrastrukturu, která je pohání [44,45].

5.3 Jak aplikace funguje

Přecházející kapitoly byly obecným shrnutím o čem aplikace pojednává, co je jejím obsahem, z jakých micro frontendů se skládá, jaké jsou použité technologie a jaké možné interakce aplikace nabízí.

Následující kapitoly nabídnou podrobný popis micro frontendů, jak aplikace funguje, detailní rozbor kódu určitých funkcí a zaměření na konkrétní procesy.

5.4 Root config (kontejnerová aplikace) – app

Funkci kontejneru hlavní stránky, vykreslení HTML stránky a JavaScriptu s registrovanými aplikacemi, veškerou koordinaci připojování a odpojování jednotlivých micro frontendových aplikací má na starosti právě kontejnerová aplikace zvaná *app*.

5.4.1 Registrace micro frontendů

```
// DavidSprla-app.js
import { registerApplication, start } from "single-spa";
import * as location from "./locations";

registerApplication({
  name: "@DavidSprla/navbar",
  app: () => System.import("@DavidSprla/navbar"),
  activeWhen: location.navbar,
  customProps: {
    domElement: document.getElementById('navbar')
  }
});

registerApplication({
  name: "@DavidSprla/movies",
  app: () => System.import("@DavidSprla/movies"),
  activeWhen: location.movies,
  customProps: {
    domElement: document.getElementById('movies')
  }
});

registerApplication({
  name: "@DavidSprla/favourite",
  app: () => System.import("@DavidSprla/favourite"),
  activeWhen: location.favourite,
  customProps: {
    domElement: document.getElementById('favourite')
  }
});
```

Ukázka kódu 8 Registrace micro frontendů

Základním předpokladem k tomu, aby Single-spa knihovna mohla pracovat s ostatními aplikacemi, je potřeba tyto aplikace jednotlivě registrovat. K tomu slouží funkce registerApplication, která přijímá minimálně tři argumenty. Mezi ty patří

název aplikace, metoda pro načtení aplikace a funkce aktivity, která slouží především pro určení, kdy je daná aplikace aktivní nebo neaktivní. Posledním volitelným argumentem jsou vlastní parametry pro předání.

V případě této ukázkové aplikace se přes vlastní parametry předává DOM element podle ID dané aplikace. Díky tomu se budou jednotlivé aplikace vykreslovat na předem určených místech, pokud dojde k aktualizaci stránky prohlížeče.

5.4.2 Router registrovaných aplikací

Jak již bylo výše zmíněno, jedním z povinných argumentů funkce registerApplication je takzvaná funkce aktivity, která jednoduše vrací logickou hodnotu true nebo false pro každou aplikaci individuálně.

```
// locations.js
export function prefix(location, ...prefixes) {
  return prefixes.some(
    prefix => location.href.indexOf(`${location.origin}/${prefix}`) !== -
1
  );
}

export function navbar() {
  return true;
}

export function movies(location) {
  return !prefix(location, 'favourite');
}

export function favourite(location) {
  return prefix(location, 'favourite');
}
```

Ukázka kódu 9 Router registrovaných aplikací

Pro ukázkovou aplikaci byl speciálně vytvořen javascriptový soubor, který s aktuální lokací pracuje komplexněji. Ve výše zobrazeném zdrojovém kódu je možno vidět 4 funkce. Navbar funkce vrací vždy true, protože navigace webové aplikace má být vždy zobrazena. Dále je zde funkce movies, která je zobrazena pouze v případě, pokud prefix URL adresy (https://mftv.herokuapp.com/prefix) nezačíná

slovem favourite. Z toho důvodu je zobrazena při spuštění aplikace na domovské stránce. Funkce favourite funguje opačným způsobem než funkce movies. Zavolá se pouze v případě, pokud prefix začíná slovem favourite.

Takto vyhodnocené lokace jsou naimportovány do hlavního konfiguračního souboru s registracemi aplikací a předány do argumentu activeWhen.

5.4.3 Index soubor

Posledním důležitým prvkem v kontejnerové aplikaci je indexový HTML soubor. Ten kromě běžných metadat v hlavičce obsahuje i odkaz na favicony pro celou aplikaci, logo a patičku. Dále je v něm obsažen i odkaz na Single-spa a SystemJS knihovnu.

Velmi důležitým prvkem, který se v index souboru vyskytuje, jsou import mapy. Import mapa je obyčejný JSON, který obsahuje informace o tom, kde lze nalézt každý javascriptový bundle. Ukázková aplikace obsahuje dva druhy import map – pro lokální vývoj a pro produkční prostředí. Pokud aplikace běží v lokálním prostředí, tak je produkční import mapa přepsána lokální verzí.

```
<!-- index.ejs -->
<!-- Import mapa pro produkční prostředí -->
  <script type="systemjs-importmap" src="//mftv.s3.eu-central-
1.amazonaws.com/%40DavidSprla/importmap.json"></script>

<!-- Import mapa pro lokální prostředí -->
  <% if (isLocal) { %>
  <script type="systemjs-importmap">
    {
      "imports": {
        "@DavidSprla/app": "//localhost:9000/DavidSprla-app.js",
        "@DavidSprla/navbar": "//localhost:9001/DavidSprla-navbar.js",
        "@DavidSprla/movies": "//localhost:9002/js/app.js",
        "@DavidSprla/favourite": "//localhost:9003/js/app.js",
        "@DavidSprla/styleguide": "//localhost:9004/DavidSprla-
styleguide.js"
      }
    }
  </script>
  <% } %>
```

Ukázka kódu 10 Indexový HTML soubor

Import mapy obsahují také importy sdílených závislostí, do kterých spadá knihovna Single-spa, React, React-dom a Vue. Díky sdíleným závislostem nemusí aplikace stahovat více kopií jednotlivých knihoven, ale stačí pouze jediná.

Produkční verze import mapy je uložena na cloudovém objektovém úložišti S3 od Amazonu. Během procesu nasazení jednotlivých micro frontendů na produkční prostředí se import mapa automaticky aktualizuje. Tento proces bude podrobněji popsán v následujících kapitolách.

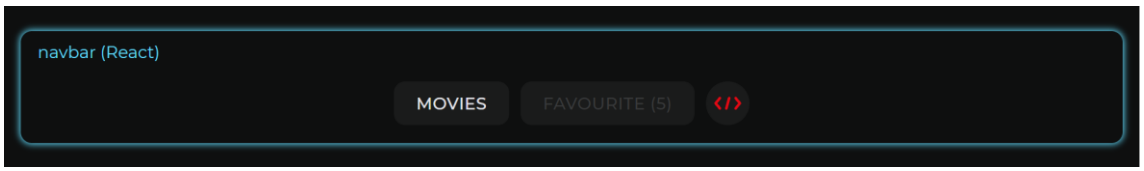
5.5 Micro frontend – navbar

První popisovanou micro frontendovou aplikací je *navbar* (obrázek 12). Jeho hlavním úkolem je umožnění navigace v aplikaci.

Aplikace je napsána v React frameworku a využívá i React Router pro výše zmiňovanou navigaci. Skládá se ze tří interaktivních tlačítek.

První dvě (movies a favourite) používají NavLink od React Routeru pro navigaci. NavLink oproti klasickému Linku umožňuje stylování vykresleného elementu, pokud se shoduje se současnou URL adresou. Vlastní třídu pro aktivní prvek lze nastavit přes activeClassName.

Poslední tlačítko slouží pro zapnutí nebo vypnutí funkce framework inspektora. Jak funguje aktivace framework inspektora je vysvětleno v následujících kapitolách.



Obr. 12 Micro frontend – navbar
Zdroj: Vlastní tvorba

```
// root.component.js
export default function Root() {
...
  return (
    <BrowserRouter>
      <nav className={`navbar ${inspect ? 'navbar--inspect' : ''}`}>
        <div className="navbar__item">
          <NavLink exact to="/" className="navbar__link"
activeClassName="navbar__link--active">
            Movies
          </NavLink>
        </div>
        <div className="navbar__item">
          <NavLink to="/favourite" className="navbar__link"
activeClassName="navbar__link--active">
            Favourite ({favourite})
          </NavLink>
        </div>
        <div className="navbar__item navbar__item--inspect">
          <button type="button" title="Framework inspector"
className={`navbar__inspect ${inspect ? 'navbar__inspect--active' : ''}`}
onClick={handleInspectBtn}>
          </button>
        </div>
      </nav>
    </BrowserRouter>
  );
}
```

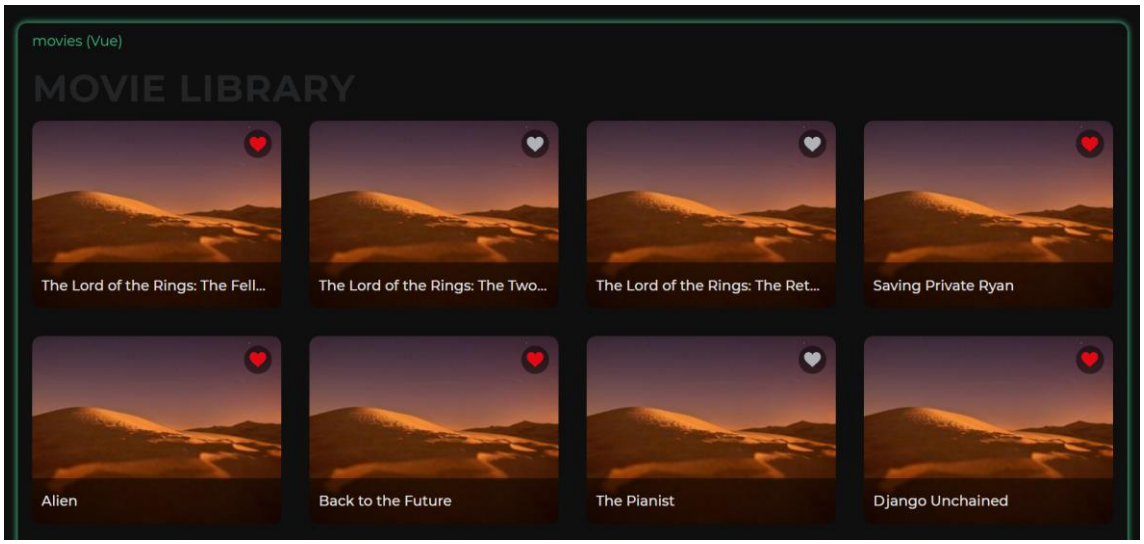
Ukázka kódu 11 Micro frontend – navbar

5.6 Micro frontend – movies

Hlavní obsah celé aplikace tvoří micro frontend *movies* (obrázek 13). Obsahuje seznam všech filmů, které jsou v aplikaci k dispozici. Dále je možné přidat konkrétní filmy do oblíbených, ty jsou poté k vidění v sekci favourite. Proces přidání a odebrání z oblíbených je taktéž popsán v následujících kapitolách.

Používá se zde framework Vue, a to především z jednoho hlavního důvodu, a tím je možnost jednoduchého použití scoped stylů. Vzhledem k tomu, že se v micro

frontendu favourite používá téměř totožná HTML struktura i třídy stylů, je to velmi vítaná možnost.



Obr. 13 Micro frontend – movies
Zdroj: Vlastní tvorba

5.7 Micro frontend – favourite

Podobnou micro frontend aplikací je *favourite*. Obsahuje seznam všech oblíbených filmů, které byly přidány skrze micro frontend *movies*. Pokud je seznam prázdný, uživatel je na tuto skutečnost upozorněn informační hláškou.

Základem tohoto micro frontendu je stejně jako v předchozím případě framework Vue.

5.8 Micro frontend utilita – styleguide

Poslední částí ukázkové aplikace je *styleguide*. Jedná se o modul v prohlížeči, který umožňuje export proměnných nebo funkcí. Ty si mohou poté jednotlivé micro frontendy naimportovat a používat. Slouží především pro sdílení běžné logiky, kterou by si jinak každá micro frontendová aplikace musela implementovat zvlášť.

Sdílený obsah v této ukázkové aplikaci tvoří styly. Mezi ty patří například import používaného fontu, základní stylování dokumentu, nadefinování proměnných. Mimo jiné sem patří i styly pro logo a patičku, jelikož nejsou součástí žádného z micro frontendů.

```css
/* global.css */
@import
url('https://fonts.googleapis.com/css2?family=Montserrat:wght@400;500;700
&display=swap');

:root{
  --vue: #41B883;
  --react: #61DBFB;
  --black: #0e0f0f;
  --dark-gray: #242526;
  --gray: #3a3b3c;
  --light-gray: #b0b3b8;
  --white: #e4e6eb;
  --red: #E50914;

  --transition: all .238s ease;

  font-size: 10px;
}

*{
  box-sizing:  border-box;
  -ms-overflow-style: none;
}

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

body{
  display: flex;
  flex-direction: column;
  font-family: "Montserrat", sans-serif;
  font-size: 1.5rem;
  background-color: var(--black);
}
...
```

Ukázka kódu 12 Globální styly

Tento soubor se styly je importován do hlavního souboru *styleguide* utility. Poté je proveden build, který je nakonec vložen pomocí import mapy do aplikace.

```js
// DavidSprla-styleguide.js
import "./global.css";
```

Ukázka kódu 13 Import souboru s globálními styly

5.9 Komunikace mezi micro frontendy a globální state

Vzhledem ke konceptu micro frontend architektury je doporučeno, aby ke komunikaci mezi jednotlivými micro frontendy docházelo v co nejmenší možné míře. Pouze v tomto případě bude zajištěna vzájemná nezávislost. Jestliže spolu micro frontendy komunikují příliš, nastává otázka, zda jednotlivé micro frontendy nesloučit.

Ukázková aplikace obsahuje dva případy komunikace. Prvním případem je přidání filmu do oblíbených. Druhým případem je přepínání framework inspektora.

Co se týče globálního úložiště, platí podobné doporučení, jako v případě komunikace. Zde je jako globální úložiště použit Local Storage, kde je uchováno pole oblíbených filmů a indikátor stavu aktivity framework inspektora.

5.9.1 Přidání filmu do oblíbených

Film je možné přidat do oblíbených přes tlačítko u každého filmu na stránce movies. Kliknutím na tlačítko se spustí funkce favouriteMovie a jako parametr se předá celý objekt filmu.

```
// Movies.vue
{
  id: 1,
  title: 'The Lord of the Rings: The Fellowship of the Ring',
  favourite: false
}
```

Ukázka kódu 14 Objekt filmu

Funkce vyhledá v Local Storage prvek movies, který reprezentuje pole oblíbených filmů a vyhledá v něm vybraný film. Pokud film nebyl nalezen, přidá se do pole, v opačném případě se z pole odstraní. Následně se prvky v poli seřadí podle ID filmu. Jakmile je pole seřazeno, vloží se do prvku movies v Local Storage.

```
// Movies.vue
const moviesStorage = JSON.parse(localStorage.getItem('movies')) || []
let movieIndex = this.movies.findIndex(movieItem => movieItem.id ===
movie.id)

if (!moviesStorage.find(movieStorage => movieStorage.id === movie.id)) {
  moviesStorage.push(movie)
  this.movies[movieIndex].favourite = true
} else {
  moviesStorage.splice(moviesStorage.indexOf(moviesStorage.find(movieStor
age => movieStorage.id === movie.id)), 1)
  this.movies[movieIndex].favourite = false
}

moviesStorage.sort((a, b) => {
  return a.id - b.id
})

localStorage.setItem('movies', JSON.stringify(moviesStorage))
```

Ukázka kódu 15 Přidání filmu do oblíbených

Pokud je v aplikaci zobrazena stránka favourite, během mounted hooku se z Local Storage načtou oblíbené filmy a následně uloží do reaktivní proměnné.

Proces přidání filmu do oblíbených však obsahuje ještě poslední akci a tou je vytvoření Custom Eventu s názvem favouriteEvent, který v sobě nese informaci o aktuálním počtu oblíbených filmů.

```
// Movies.vue
const favouriteCustomEvent = new CustomEvent('favouriteEvent', {
  detail: moviesStorage.length
})
window.dispatchEvent(favouriteCustomEvent)
```

Ukázka kódu 16 Vytvoření Custom Eventu s názvem favouriteEvent

Tento Custom Event je odposloucháván v micro frontendu *navbar*, který událost zpracuje a přiřadí informaci z Custom Eventu do lokální proměnné. Hodnota proměnné je zobrazena vedle navigačního tlačítka favourite.

```
// root.component.js v micro frontendu navbar
const [favourite, setFavourite] = useState(0)

const handleFavourite = (event) => {
  setFavourite(event.detail)
}

window.addEventListener('favouriteEvent', handleFavourite)
```

Ukázka kódu 17 Event Listener na favouriteEvent

5.9.2 Přepínání framework inspektora

Druhým případem komunikace je přepínání framework inspektora. Ten lze přepnout v *navbar* aplikaci. Nabývá pouze logických hodnot true nebo false. Pokud je aktivní, přidá se třída stylů v micro frontendech *navbar*, *movies* a *favourite*, která zvýrazní jejich hranice působnosti, název a framework, který je použit.

Stejně jako v předchozím případě, i zde funguje komunikace na principu Custom Eventu a Event Listeneru. Custom Event je vytvořen v navbar aplikaci, která jako informaci pošle inverzní hodnotu aktuálního stavu inspektora. Nakonec se hodnota uloží do Local Storage. Odposlech na tento Custom Event je implementován v micro frontendech *movies* a *favourite*. Pokud dojde k přechodu na jinou stránku nebo znovunačtení aplikace, stáhne se během mounted hooku z Local Storage aktuální stav inspektora.

5.10 Stylování

Téma stylování ukázkové aplikace bylo již několikrát v práci zmíněno. Tato kapitola se blíže věnuje způsobu implementace a řešení určitých problémů.

Ukázková aplikace je stylována ze dvou zdrojů. Prvním zdrojem jsou globální styly, které leží v samostatné micro frontendové utilitě zvané *styleguide*. V globálních stylech jsou definovány CSS proměnné a stylování kostry webové aplikace včetně loga a patičky. CSS proměnné jsou skvělým příkladem, jak zachovat určitý standard stylování napříč celou aplikací. Příkladem mohou být použité barvy, fonty nebo jiné hodnoty. Vzhledem k principům micro frontend architektury je dobré udržet zdravou míru použitých globálních stylů, aby byla stále zajištěna nezávislost, ale i určitý standard pro aplikaci.

Druhým zdrojem jsou lokální styly jednotlivých micro frontendových aplikací, které by měly být správně izolovány od ostatních, aby nedošlo k nechtěnému přepsání.

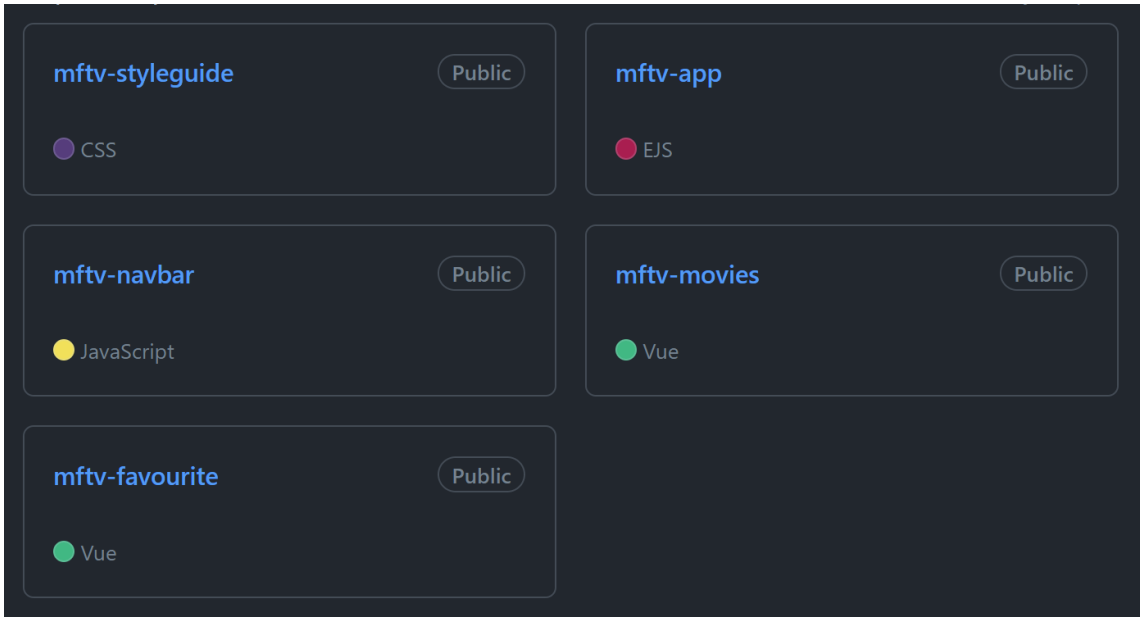
V případě micro frontendů *movies* a *favourite* jsou použity scoped styly, které perfektně izolují jednotlivé styly micro frontendů, takže je možné použít stejné názvy tříd stylů pro stejné komponenty. Velkou nevýhodou je zbytečná duplikace kódu.

Micro frontend *navbar* implementuje styly formou CSS modulů. Takto importované styly nejsou izolovány a lze je snadno přepsat. Řešením této situace by bylo například použití nějakého prefixu názvu tříd podle daného micro frontendu.

Jak je možno vidět na tomto projektu, používají se zde různé implementace stylů. Pokud by se pro micro frontend *navbar* použil prefix názvů tříd, nastal by mírný zmatek v použitých stylech. Doporučeným východiskem by bylo jednotné použití určitého způsobu, například výše zmíněné prefixy tříd stylů dle jednotlivých micro frontendů (nav-classname, mov-classname, fav-classname). Takový způsob je pak velmi jednoduchý pro případný debugging, protože je možné rychle zjistit původ elementu podle názvu třídy. Velmi snadný a bezproblémový způsob je použití scoped stylů, který by se dal považovat za nejlepší variantu.

5.11 Verzování

Aby bylo možné docílit end-to-end řízení jednotlivých micro frontendů, je nutné dosáhnout jejich absolutní nezávislosti. Jedním z těchto kroků je i nezávislé verzování. Protože je ukázková aplikace složena z pěti různých aplikací, má každá z nich svůj vlastní veřejný repozitář na GitHubu (obrázek 14). Díky tomu má každý tým plnou kontrolu nad verzováním své aplikace.



Obr. 14 GitHub repozitáře MFTV
Zdroj: Vlastní tvorba

5.12 Nasazení do produkce

Způsob nasazení do produkčního prostředí je posledním problémem k řešení. Jak již bylo zmíněno v přechozích kapitolách, ukázková aplikace je složena z pěti částí, které umožňují nezávislost od vývoje až po nasazení do produkce. Proto je nutné každému týmu jednotlivě nastavit celý proces nasazení.

Proces nasazení jednotlivých micro frontendových aplikací probíhá následujícím způsobem. Jakmile je jednotlivým týmem proveden příkaz commit a následně push nového kódu do master větve, je touto akcí spuštěna úloha Travisu CI.

Travis zahájí svou činnost podle konfiguračního souboru, který je specifický pro každou aplikaci. V případě ukázkové aplikace je spuštěn build, který se poté nahraje do úložiště Amazon S3. Z hlediska bezpečnosti jsou pro konfiguraci citlivých údajů AWS S3 použita proměnná prostředí, která lze nakonfigurovat přes webové rozhraní Travisu.

```yaml
# travis.yml
language: node_js
node_js:
  - node
env:
  global:
    - PATH=$HOME/.local/bin:$PATH
before_install:
  - pyenv global 3.7.1
  - pip install -U pip
  - pip install awscli
script:
  - npm run build
  - echo "AWS COMMIT - $TRAVIS_COMMIT"
  - mkdir -p dist/@DavidSprla/navbar/$TRAVIS_COMMIT
  - mv dist/*.* dist/@DavidSprla/navbar/$TRAVIS_COMMIT/
deploy:
  provider: s3
  access_key_id: "$AWS_ACCESS_KEY_ID"
  secret_access_key: "$AWS_SECRET_ACCESS_KEY"
  bucket: "$AWS_BUCKET"
  skip_cleanup: true
  acl: public_read
  region: eu-central-1
  local_dir: dist
  cache-control: "max-age=31536000"
  on:
    branch: master
```

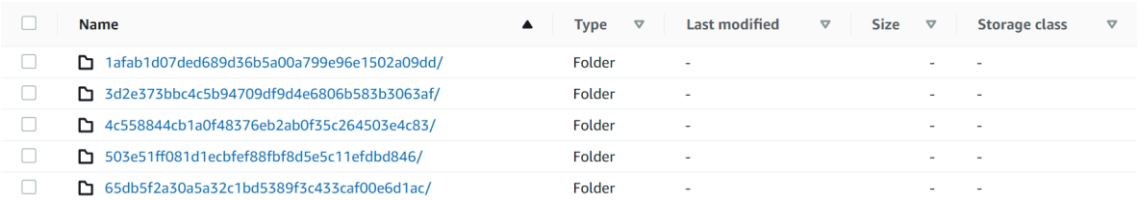
Ukázka kódu 18 Konfigurační soubor pro navbar

Nahrané soubory jsou poté k vidění skrze AWS konzoli, kde je každá aplikace uložena jako jeden objekt. Všechny tyto objekty spadají pod jediný zastřešující bucket (obrázek 15).

☐	app/	Folder	-	-	-
☐	favourite/	Folder	-	-	-
☐	importmap.json	json	October 2, 2021, 22:23:07 (UTC+02:00)	1.1 KB	Standard
☐	movies/	Folder	-	-	-
☐	navbar/	Folder	-	-	-
☐	styleguide/	Folder	-	-	-

Obr. 15 Seznam objektů v úložišti Amazon S3
Zdroj: Vlastní tvorba

Jednotlivé buildy aplikací jsou v S3 uloženy pod číslem commitu, takže je možné dohledat jejich jednotlivé verze (obrázek 16).



Name	Type	Last modified	Size	Storage class
1afab1d07ded689d36b5a00a799e96e1502a09dd/	Folder	-	-	-
3d2e373bbc4c5b94709df9d4e6806b583b3063af/	Folder	-	-	-
4c558844cb1a0f48376eb2ab0f35c264503e4c83/	Folder	-	-	-
503e51ff081d1ecbfef88fbf8d5e5c11efdbd846/	Folder	-	-	-
65db5f2a30a5a32c1bd5389f3c433caf00e6d1ac/	Folder	-	-	-

Obr. 16 Jednotlivé buildy micro frontendu navbar
Zdroj: Vlastní tvorba

Jakmile proběhne kompletní proces nasazení na S3, nově nahrané změny se však v kontejnerové aplikaci zatím neprojeví. Kontejnerová aplikace se skrze import mapy odkazuje na jednotlivé buildy aplikací. Jestliže nedojde k aktualizaci import mapy, bude stále odkazovat na starší buildy. Proto je nutné po dokončení nasazení spustit akci, která automaticky aktualizuje import mapu s novým odkazem na build aplikace. Travis má pro tuto situaci speciální krok after_deploy, který spustí vlastní skript.

```
# travis.yml
after_deploy:
  - chmod +x after_deploy.sh
  - "./after_deploy.sh"
```

Ukázka kódu 19 Konfigurační soubor pro navbar (after_deploy)

Tento skript stáhne aktuální import mapu z AWS S3, spustí soubor update-importmap.mjs, který provede aktualizaci import mapy a následně ji nahraje zpět.

```
# after_deploy.sh
echo "Downloading importmap.json from AWS S3 bucket"
aws s3 cp s3://mftv/@DavidSprla/importmap.json importmap.json
echo "Updating importmap..."
node update-importmap.mjs
echo "Uploading importmap.json to AWS S3 bucket"
aws s3 cp importmap.json s3://mftv/@DavidSprla/importmap.json --cache-
control 'public, must-revalidate, max-age=0' --acl 'public-read'
echo "Updating importmap was successful"
```

Ukázka kódu 20 Obsah souboru after_deploy.sh

V tuto chvíli se již kontejnerová aplikace odkazuje na správné buildy aplikací a tím pádem se projeví i nově provedené změny v kódu.

6 Shrnutí výsledků

Hlavním cílem této diplomové práce bylo seznámit čtenáře s micro frontend architekturou. Mezi otázky, na které jsem během zkoumání tohoto tématu hledal patřičné odpovědi, patří popis konceptu micro frontend architektury, jakým způsobem funguje, dostupné možnosti integrace, výhody a nevýhody, případy vhodného a nevhodného použití této architektury.

Micro frontend architekturu, stejně jako ostatní architektury, nelze považovat za vhodný přístup pro všechny typy projektů. Vhodné použití, pro střední a velké projekty, potvrzují úspěšná nasazení ve známých společnostech, které byly v této práci popsány.

Možností pro implementaci micro frontend architektury existuje mnoho. Volba závisí například na potřebách projektu, dostupném týmu vývojářů, jejich zkušenostech a dalších faktorech. V teoretické části této práce je uvedeno několik variant implementace včetně souhrnné tabulky s hodnocením jednotlivých vlastností dané možnosti, které mohou pomoci při výběru vhodné varianty.

Poslední částí při zkoumání tohoto tématu byl vývoj vlastní ukázkové micro frontendové aplikace. Smyslem této aplikace bylo prozkoumat micro frontend architekturu z praktického hlediska a ověřit si některé výhody a nevýhody popsané v teoretické části. Prvotní zkušeností při vývoji aplikace bylo zdlouhavé nastavování, instalace potřebných balíčků a rozběhnutí vývoje v lokálním prostředí. Vzhledem k tomu, že se aplikace skládá z pěti částí (tři micro frontendové aplikace a dvě globální), měl by být vývoj v ideálním případě rozdělen mezi tři týmy, které by měly na starost vývoj jednotlivých částí aplikace. Protože však byly všechny tyto týmy zastoupeny mnou, byl počáteční vývoj mírně frustrující, jelikož pro dosažení prvních viditelných výsledků bylo nutné investovat velké množství času. S postupem času se prvotní negativa vytratila a poté již bylo možné čerpat z výhod micro frontend architektury, například nezávislého nasazení do produkce.

7 Závěry a doporučení

Micro frontend architektura není univerzálním přístupem pro všechny typy projektů. Aby se zjistilo, zda je použití této architektury na konkrétním projektu vhodné, je nutné předem provést důkladnou analýzu a porovnat, zda budou výhody převažovat nad nevýhodami. Vhodný případ použití micro frontend architektury je především u středních a velkých projektů, kde jsou monolitické frontendy obtížně spravovatelné. S micro frontend přístupem se monolit rozdělí na menší nezávislé části aplikace, které spravují jednotlivé týmy [2,3,10].

Zde bych rád navázal na alternativní možnosti zkoumání micro frontend architektury. Vzhledem k tomu, že není možné sestavit perfektní rozhodující tabulku pro všechny typy projektů, bylo by vhodnější provést výzkum a analýzu na konkrétních reálných projektech.

V případě výzkumu na projektech, které jsou již postaveny na micro frontend architektuře, by bylo vhodné porovnat jednotlivé charakteristiky původního řešení (pokud existuje) a nového řešení. Mezi charakteristiky lze zařadit například efektivitu vývoje nebo koordinaci.

Pokud by se jednalo o nový projekt, zaměřil by se výzkum na výběr vhodné varianty implementace micro frontend architektury podle specifických požadavků projektu.

8 Seznam použité literatury

[1] Geers, Michael. Micro Frontends in Action. Manning Publications Co., 2020.

[2] Peltonen, Severi, et al. “Information and Software Technology.” Elsevier, vol. 136, no. Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review, Mar. 2021, p. 18, https://www.sciencedirect.com/science/article/pii/S0950584921000549.

[3] Jackson, Cam. „Micro Frontends". Micro Frontends, 19. červen 2019, https://martinfowler.com/articles/micro-frontends.html.

[4] Thoughtworks, Inc. „Micro frontends". Micro frontends | Technology Radar | Thoughtworks, 19. květen 2020, https://www.thoughtworks.com/radar/techniques/micro-frontends.

[5] Geers, Michael. „Micro Frontends". Micro Frontends - extending the microservice idea to frontend development, https://micro-frontends.org/.

[6] Entando. „7 Successful Companies Using Micro Frontends". 7 Successful Companies Using Micro Frontends - Entando Home, 4. únor 2020, https://www.entando.com/page/en/7_successful_companies_using_micro_frontends_en_1?contentId=BLG2303&modelId=25.

[7] Spezzano, Luca. „Micro Frontends Architecture". Micro Frontends Architecture. The future of frontend development | by Luca Spezzano | NotOnlyCSS | Medium, 10. prosinec 2019, https://medium.com/notonlycss/micro-frontends-architecture-1407092403d5.

[8] Tapan, Vora. „Micro Frontends – Revolutionizing Front-end Development with Microservices". Micro Frontends – Revolutionizing Front-end Development with Microservices - Cuelogic Technologies Pvt. Ltd., 23. listopad 2020, https://www.cuelogic.com/blog/micro-frontends-part1.

[9] Mezzalira, Luca. Building Micro-Frontends. O’Reilly Media, Inc., 2021.

[10] Singh Gill, Navdeep. „Micro Frontend Architecture and Best Practices". Micro Frontend Architecture and Best Practices, 26. květen 2021, https://www.xenonstack.com/insights/micro-frontend-architecture.

[11] MDN contributors. „SPA (Single-page application)". SPA (Single-page application) - MDN Web Docs Glossary: Definitions of Web-related terms | MDN, 8. říjen 2021, https://developer.mozilla.org/en-US/docs/Glossary/SPA.

[12] Netlify. „The modern way to build Websites and Apps that delivers better performance". For fast and secure sites | Jamstack, 2021, https://jamstack.org/.

[13] Gordon, Elyse Kolker. Isomorphic Web Applications. Manning Publications Co, 2018.

[14] Mezzalira, Luca. „Micro-frontends decisions framework". Micro-frontends decisions framework | by Luca Mezzalira | Medium, 22. prosinec 2019, https://medium.com/@lucamezzalira/micro-frontends-decisions-framework-ebcd22256513.

[15] Dhaduk, Hiren. „Micro Frontend Architecture: The Newest Approach To Building Scalable Frontend". Micro-frontend architecture: Principles, implementations and challenges, 22. říjen 2021, https://www.simform.com/blog/micro-frontend-architecture/.

[16] Rappl, Florian, a Lothar Schottner. The Art of Micro Frontends. Packt Publishing, 2021.

[17] proinity LLC. „What Is Edge Side Includes (ESI)?" What Is Edge Side Includes (ESI)? - KeyCDN Support, 4. říjen 2018, https://www.keycdn.com/support/edge-side-includes.

[18] Wistow, Simon. „Using ESI, Part 1: Simple Edge-Side Include". Using ESI, Part 1: Simple Edge-Side Include | Fastly, 27. srpen 2014, https://www.fastly.com/blog/using-esi-part-1-simple-edge-side-include.

[19] Edpresso Team. „What is server-side rendering?" What is server-side rendering?, 2021, https://www.educative.io/edpresso/what-is-server-side-rendering.

[20] Zalando Tech. „Project Mosaic". Project Mosaic—Frontend Microservices, 2016, https://www.mosaic9.org/.

[21] webpack contributors. „Module Federation". Module Federation | webpack, https://webpack.js.org/concepts/module-federation/.

[22] Grini, Helene. „Micro frontends with Module Federation". Micro frontends with Module Federation | by Helene Grini | Bredvid, 22. březen 2021, https://blog.bredvid.no/micro-frontends-with-module-federation-e4ed75fcc328.

[23] Refsnes Data. HTML <iframe> Tag. 2021, https://www.w3schools.com/tags/tag_iframe.ASP.

[24] The Luigi project authors. „The Enterprise-Ready Micro Frontend Framework". Luigi - The Enterprise-Ready Micro Frontend Framework, 2021, https://luigi-project.io/.

[25] MDN contributors. „Using custom elements". Using custom elements - Web Components | MDN, 14. říjen 2021, https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_custom_elements.

[26] MDN contributors. „Using shadow DOM". Using shadow DOM - Web Components | MDN, 14. říjen 2021, https://developer.mozilla.org/en-US/docs/Web/Web_Components/Using_shadow_DOM.

[27] Abramov, Dan, a Redux documentation authors. „Redux". Redux - A predictable state container for JavaScript apps. | Redux, 2021, https://redux.js.org/.

[28] Abramov, Dan, a Redux documentation authors. „Redux FAQ: Store Setup". Store Setup | Redux, 2021, https://redux.js.org/faq/store-setup.

[29] MDN contributors. „Web Storage API". Web Storage API - Web APIs | MDN, 14. září 2021, https://developer.mozilla.org/en-US/docs/Web/API/Web_Storage_API.

[30] Entando. „7 Successful Companies Using Micro Frontends". 7 Successful Companies Using Micro Frontends - Entando Home, 4. únor 2020, https://www.entando.com/page/en/7_successful_companies_using_micro_frontends_en_1?contentId=BLG2303&modelId=25.

[31] lucamezzalira. „Adopting a Micro-frontends architecture". Adopting a Micro-frontends architecture, 8. duben 2019, https://lucamezzalira.com/2019/04/08/adopting-a-micro-frontends-architecture/.

[32] Stenberg, Jan. „Experiences Using Micro Frontends at IKEA". Experiences Using Micro Frontends at IKEA, 7. srpen 2018, https://www.infoq.com/news/2018/08/experiences-micro-frontends/.

[33] OpenComponents. „OpenComponents". OpenComponents, https://opencomponents.github.io/.

[34] Brockmeyer, Jan. „Micro Frontends: from Fragments to Renderers (Part 1)". Micro Frontends: from Fragments to Renderers (Part 1), 11. březen 2021, https://engineering.zalando.com/posts/2021/03/micro-frontends-part1.html.

[35] Pfeiffer, Andrei. „A Thorough Analysis of CSS-in-JS". A Thorough Analysis of CSS-in-JS | CSS-Tricks, 26. květen 2021, https://css-tricks.com/a-thorough-analysis-of-css-in-js/.

[36] single-spa. „single-spa". single-spa | single-spa, 2021, https://single-spa.js.org/.

[37] „Concepts". Concepts | webpack, 2021, https://webpack.js.org/concepts/.

[38] Kiran, Rabi. „Modular JavaScript: A Beginners Guide to SystemJS & jspm". Modular JavaScript: A Beginners Guide to SystemJS & jspm - SitePoint, 11. květen 2016, https://www.sitepoint.com/modular-javascript-systemjs-jspm/.

[39] Facebook Inc. „React". React – A JavaScript library for building user interfaces, 2021, https://reactjs.org/.

[40] You, Evan. „Introduction". Introduction — Vue.js, 21. říjen 2020, https://vuejs.org/v2/guide/.

[41] Kinsta Inc. „What Is GitHub? A Beginner’s Introduction to GitHub". What Is GitHub? A Beginner’s Introduction to GitHub, 28. květen 2021, https://kinsta.com/knowledgebase/what-is-github/.

[42] „Core Concepts for Beginners". Core Concepts for Beginners - Travis CI, https://docs.travis-ci.com/user/for-beginners/.

[43] Amazon Web Services, Inc. „What is Amazon S3?" What is Amazon S3? - Amazon Simple Storage Service, 2021, https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html.

[44] Salesforce.com. „Heroku is a cloud platform that lets companies build, deliver, monitor and scale apps — we’re the fastest way to go from idea to URL, bypassing all those infrastructure headaches." What is Heroku | Heroku, 2021, https://www.heroku.com/what.

[45] Rusev, Konstantin. „What is Heroku and What is it Used For?" What is Heroku and What is it Used For? - MentorMate, 31. prosinec 2020, https://mentormate.com/blog/what-is-heroku-used-for-cloud-development/.

[46] Google Developers. „Web Vitals". Web Vitals, https://web.dev/learn-web-vitals/.

[47] Tatum, Malcolm. „What is Transclusion?" What is Transclusion?, 2021, https://www.easytechjunkie.com/what-is-transclusion.htm.

9 Obsah elektronické přílohy

1. mftv-app (kontejner aplikace)
2. mftv-navbar (micro frontend – navbar)
3. mftv-movies (micro frontend – movies)
4. mftv-favourite (micro frontend – favourite)
5. mftv-styleguide (utilita s globálními styly)

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**David Šprla**
Studium:	I1900553
Studijní program:	N1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název diplomové práce:	**Micro frontend architektura**
Název diplomové práce AJ:	Micro frontend architecture

Cíl, metody, literatura, předpoklady:

Cíl: Popsat k čemu slouží microfrontend architektura, co řeší. Jak řešit problémy nasazování, verzovaní, komunikace mezi komponentami, globální stav (state), lokální stylování. Zda je možné využít SSR. Koexistence více jazyků/frameworků (react, angular, vue apod.)

Osnova:

1. Úvod
2. Cíl práce
3. Současný stav webových aplikací
4. Microfrontend architektura
5. Ukázková aplikace
6. Výsledky a závěr

1. Geers, Michael. Micro Frontends in Action. Manning Publications, 2020
2. Jackson, Cam. "Micro Frontends". Micro Frontends, 19. červen 2019, https://martinfowler.com/articles/micro-frontends.html

Garantující pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	Ing. Pavel Kříž, Ph.D.
Datum zadání závěrečné práce:	9.9.2021