**Univerzita Hradec Králové**
**Fakulta informatiky a managementu**
**Katedra informačních technologií**

# Jednostránkové webové aplikace

Bakalářská práce

Autor: Dan Forejtek
Studijní obor: Informační management

Vedoucí práce: Mgr. Daniela Ponce, Ph.D.

Hradec Králové Listopad 2020

Prohlášení:

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

V Hradci Králové dne 14. 11. 2020 Dan Forejtek

Poděkování:
Děkuji Mgr. Daniele Ponce, Ph.D. za její vstřícnost a trpělivost při odborném vedení této práce.

# Anotace

Tato bakalářská práce se zabývá problematikou Jednostránkových webových aplikací psaných v jazyce JavaScript a jejich možné využití s frameworky pracujícími na straně klienta i serveru. Tím se JavaScript stává jediným programovacím jazykem, který je možné spouštět jak na straně klienta, tak i serveru. V první části teoretické sekce jsou rozebrány rozdíly mezi jednostránkovými a klasickými webovými aplikacemi, dále jsou zde představeny front-endové frameworky, případně knihovny a nástroje pro podporu jejich vývoje. V následující kapitole poté univerzální frameworky, které rozšiřují jejich pole působnosti také na stranu back-endu. Praktickým výstupem této práce je aplikace agregující data pro projektové řízení napsána pomocí frameworku, který byl v této práci vyhodnocen jako nejvýhodnější pro daný projekt. Součástí je také popis náležitostí daného projektu, který vedl k rozhodnutí, kteký z frameworků bude použit.

# Anotation


### Title: Single page web applications

This bachelor thesis deals with the issue of Single page web applications written in JavaScript and their possible use with frameworks operating on the client and server side. This makes JavaScript the only programming language that can be run on both client and server side. In the first part of the theoretical section, the differences between one-page and classic web applications are discussed, as well as front-end frameworks or libraries and tools to support their development. In the next chapter, then, universal frameworks that extend their scope to the back-end side. The practical output of this work is an application aggregating data for project management written using a framework, which was evaluated in this work as the most advantageous for the considered project. It also includes a description of the requirements of the project, which led to the decision of which of the frameworks will be used.

# Obsah

# 1 Úvod

*"Any application that can be written in JavaScript, will eventually be written in JavaScript."*

— Jeff Atwood, spoluzakladatel StackOverflow

Webové aplikace začaly nabývat na popularitě v 90. letech minulého století. Tehdy ve formě statických HTML stránek. Za účelem přidání dynamiky vznikl v roce 1995 skriptovací jazyk JavaScript, jehož autorem je Brendan Eich, spoluzakladatel projektu Mozilla. Hlavní posun od statických stránek k dynamickým přišel až v roce 2005, kdy byl publikován přístup Asynchronous JavaScript and XML (dále jen AJAX), který umožňoval změny v aplikaci bez nutnosti přenačtení stránky.

Pojem Single page applications (česky jednostránkové aplikace) vyznačuje moderní způsob tvorby webových stránek pomocí jazyku JavaScript. Principem těchto aplikací je pouze jedna stránka, která po prvotním načtení komunikuje se serverem pomocí AJAX dotazů. Server vrací data klientu obvykle ve formátu JSON, aplikace data zpracuje dynamicky na straně klienta jazykem JavaScript, který provede změny Document Object Modelu (DOM). Jelikož k renderování HTML nedochází na straně serveru, jak tomu je u klasických webových aplikací, uživateli se nepřenačítá stránka. Tímto způsobem se zvyšuje uživatelská přívětivost díky rychlosti, a zároveň se snižují nároky na výpočetní výkon serveru.

# 2 Cíl práce

Teoretická část práce si klade za cíl zanalyzování moderních frameworků pro tvorbu Single page aplikací, česky Jednostránkových webových aplikací. V práci jsou představeny základní stavební kameny všech moderních webových aplikací. Součástí práce je výběr vhodného frameworku pro vývoj webové aplikace pro agregování nástrojů pro projektové řízení, který bude interně používán v rámci společnosti Mercedes-Benz Financial Services s. r. o., pro kterou autor v době vzniku této práce pracuje.

V praktické části bude popsána vyvíjená aplikace a její funkční nároky. Výstupem teoretické části bude popisovaná aplikace pro agregování nástrojů pro projektové řízení.

Cíle práce se dají shrnout v těchto bodech:

- představit technologie používané pro vývoj jednostránkových webových aplikací
- vybrat a zanalyzovat frameworky založené na těchto technologiích
- vybrat vhodný framework pro zmíněnou aplikaci
- zhodnotit a odůvodnit výběr
- obohacení čtenáře i autora o informace zjištěné při tvorbě práce ohledně technologií používaných k tvorbě webových aplikací

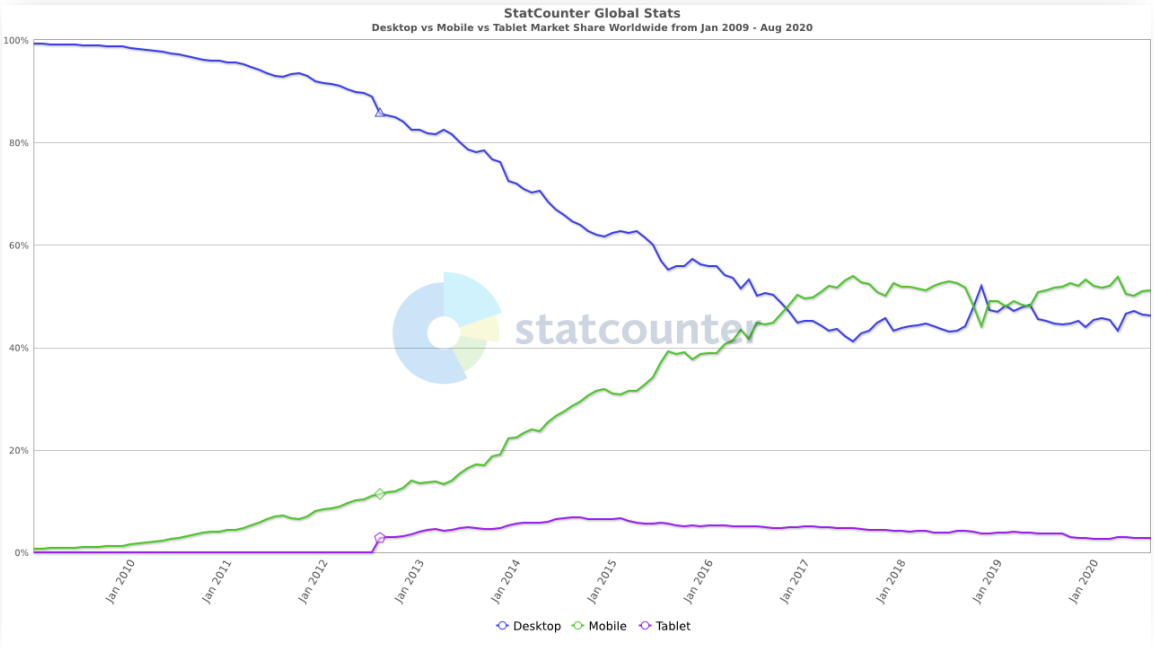
# 3 Webové aplikace

Webová aplikace (často také Web app) je počítačový program poskytovaný z webového serveru prostřednictvím internetové či intranetové sítě. Uživatel k takové aplikaci přistupuje pomocí tenkého klienta – webového prohlížeče. Aplikace je dostupná na všech platformách pod podmínkou, že obsahují webový prohlížeč. Dostupnost takových aplikací je také jejich hlavní výhodou a důvodem k jejich stále rostoucí popularitě v řadách vývojářů, ale i v řadách běžných uživatelů.

V případě klasických desktopových aplikací je na straně klienta nutná instalace dané aplikace, ovšem v případě webových aplikací tato nutnost odpadá, jelikož jeden webový prohlížeč může zobrazit několik různých webových aplikací. To šetří místo na disku a další prostředky výpočetní techniky. Pro vývojáře odpadá nutnost vývoje aplikace pro různé platformy a operační systémy, což běžně obnáší i použití různých programovacích jazyků jako například u nativních mobilních aplikací na iOS jazyk Swift a v případě Androidu Java. To má za následek nemalou finanční a také časovou zátěž.

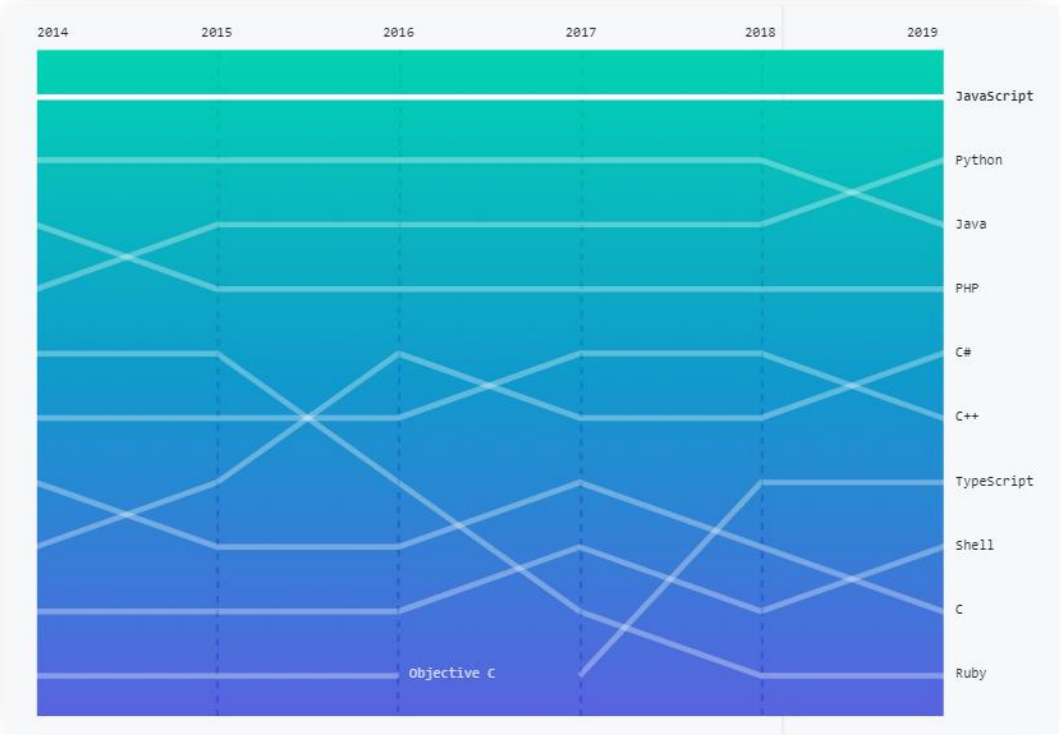
Existují zde i nevýhody, jako například rozdílná implementace webových standardů tvůrci webových prohlížečů. V kontextu nekompatibility nebo celkově rozdílné implementace standardů je nechvalně známý například Internet Explorer, jež je celosvětově neoblíbený většinou webových vývojářů. Další nevýhodou je paradoxně rychlost, jakou se webové technologie a přístupy jejich vývoje mění. Obecně informační zdroje (ať už blogové příspěvky, či odpovědi na StackOverflow) starší než jeden rok jsou v kontextu webového vývoje, a tím spíše v jazyce JavaScript, v mnoha případech zastaralé.

Podle World Advertising Research Center (2019) přistupuje k webovým stránkám z telefonu až 51 % uživatelů, na toto číslo tedy připadají 2 miliardy z celkových 3,9 miliard uživatelů internetu. WARC dále uvádí stoupající trend v počtu přístupu k internetu z mobilního zařízení a odhadují, že v roce 2025 by toto číslo mohlo dosáhnout 72,6 %. (CNBC 2019)



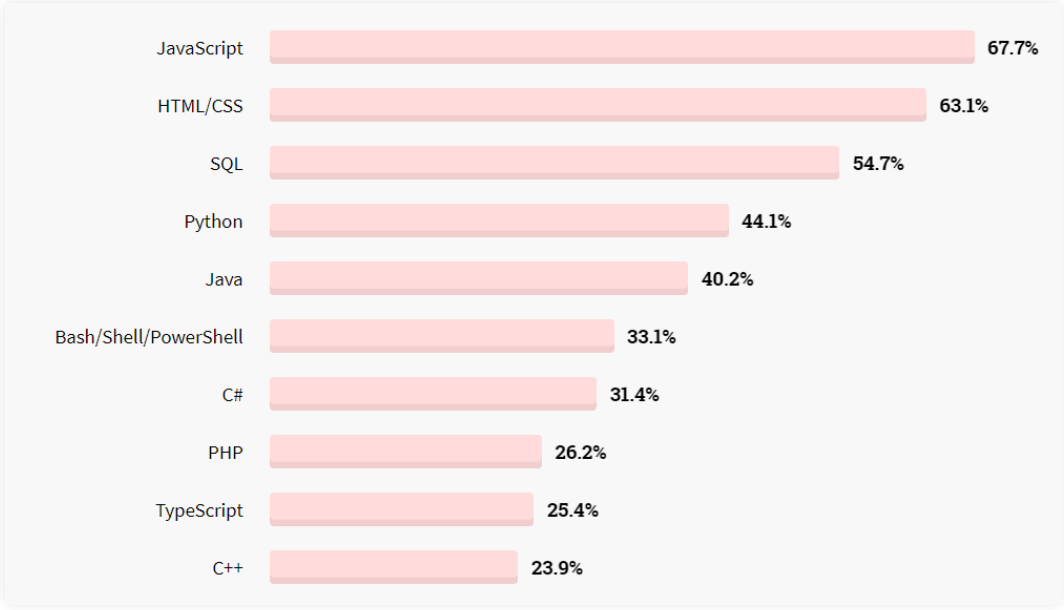

*Graf 1 - uživatelé internetu dle typu zařízení (Statcounter 2020)*

Podle každoročního průzkumu serveru GitHub.com, který je zároveň největším GIT repositářem na světě, je již několik let v řadě nejpoužívanějším jazykem JavaScript. Další zajímavostí je, že v první desítce tohoto žebříčku je také TypeScript, tedy jazyk, jež je nadstavbou JavaScriptu. (GitHub 2020)




*Graf 2 - nejpoužívanější prog. jazyky posledních let (GitHub 2019)*

Tento trend podkládá i každoroční průzkum serveru StackOverflow. Podle dat serveru je již osm let v řadě nejčastěji používaným jazykem mezi profesionálními vývojáři JavaScript. V roce 2020 JavaScript používalo 69,7 % z 47 184 respondentů. (StackOverflow 2020)




*Graf 3 - nejpopulárnější prog. jazyky dle průzkumu Stack Overflow Developer Survey 2020 (StackOverflow 2020)*

JavaScript není pouze programovací jazyk, ale spíše platforma, která nabízí mnohé výhody oproti jiným platformám. (Wagner 2017)

Výhody JavaScriptu oproti jiným jazykům v kontextu tvorby webových aplikací:

- Je to jediný jazyk nativně podporovaný webovými prohlížeči
- Je to jediný univerzální jazyk, který umožňuje tvorbu webových aplikací za pomoci pouze jediného programovacího jazyku – všechny ostatní jazyky jako C# nebo Java mohou být použity pouze na back-endu a musí být na front-endu kombinovány právě s JavaScriptem. Je tedy vyžadováno, aby vývojář ovládal alespoň dva programovací jazyky.
- Stejný kód může být spuštěn na front-endu, tak i na back-endu
- Kombinuje objektově orientované programování s funkcionálním

# 4 Jednostránkové webové aplikace

Client-server je v dnešní době jedna z nejvíce rozšířených architektur v kontextu vývoje large-scale softwaru. To umožňuje jasné vymezení rolí, o které se stará server a o které klient. Server se stará o business logiku, ukládání dat, přístup k datům ze služeb třetích stran a tak dále. Strana klienta má na starost pouze prezentaci dat uživateli. Tato architektura zároveň umožňuje napojení několika klientů, na jeden back-end a zároveň jednoduchou škálovatelnost. (Konshin 2018)

SPA je implementací této architektury na straně klienta. JavaScriptová aplikace je spuštěna z webové stránky a poté běží kompletně v prohlížeči. Veškeré vizuální změny na stránce se odehrávají jako reakce na akce vyvolané uživatelem a dat stažených ze vzdálených API.

Název jednostránkové webové aplikace vychází právě z principu, jakým tyto stránky fungují. Server slouží pro prvotní stažení HTML struktury, CSS a JavaScriptu. Právě JavaScript se, na straně klienta, po stažení stará o vykreslování obsahu a navigaci pomocí prohlížečového History API, které dynamicky mění obsah stránky a URL adresu v adresním řádku prohlížeče.

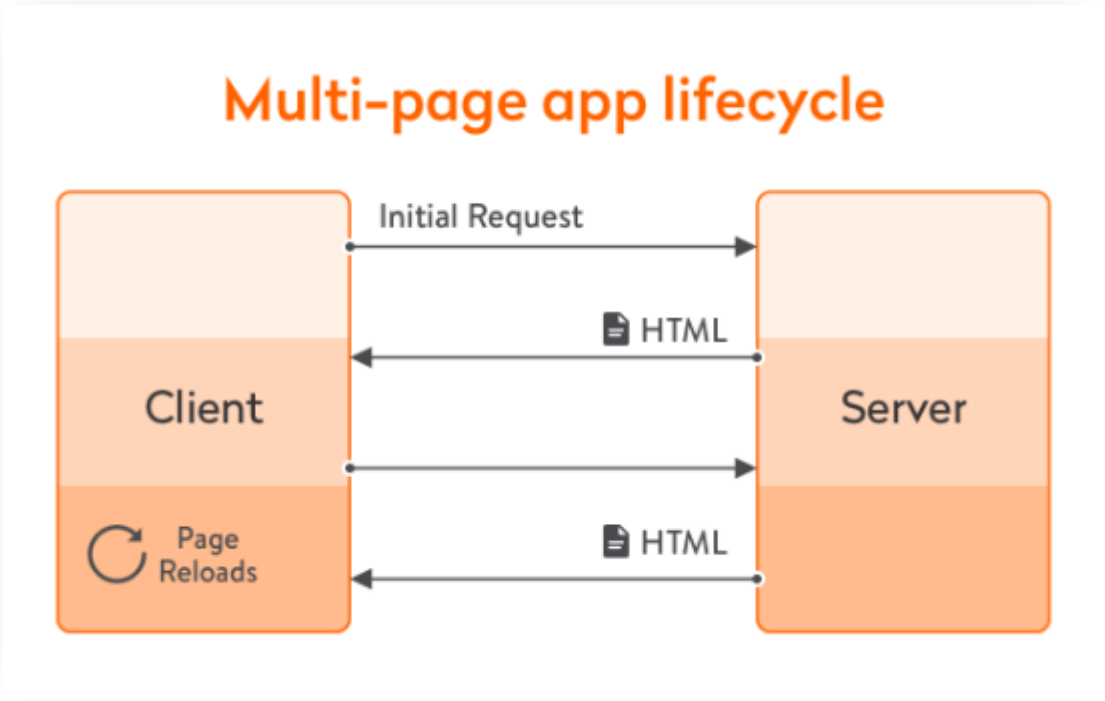
## 4.1 Historie

Nehledě na popularitu SPA, před příchodem konceptu Web 2.0 v roce 2000 trpěly webové aplikace nedostatkem uživatelské interakce a responzivity. To se změnilo s příchodem techniky zkráceně známé jako AJAX. Ta umožňuje webovým aplikacím asynchronně stahovat data z API a měnit webovou stránku bez nutnosti obnovení. (Mesbah, van Deursen 2007)
Právě tato technologie později umožnila vznik nového typu webových aplikací, které se nazývají jednostránkové.

**jQuery** by se dalo označit jako předchůdce moderních frameworků, přestože s nimi má jen málo společných rysů. jQuery sloužilo k ulehčení práce vývojáře a zjednodušení JS kódu. Dodnes je hojně využíván na většině webových stránkách, na kterých je JS použit. Pokud jste někde viděli date picker, nebo obrázkovou galerii, s největší pravděpodobností se jedná o jQuery. Jedna z nejvýznamnějších funkcí této knihovny je AJAX. Jako další funkcionalitu, která významně ulehčovala práci s DOM se dají označit jQuery selektory.

## 4.2 Princip fungování MPA

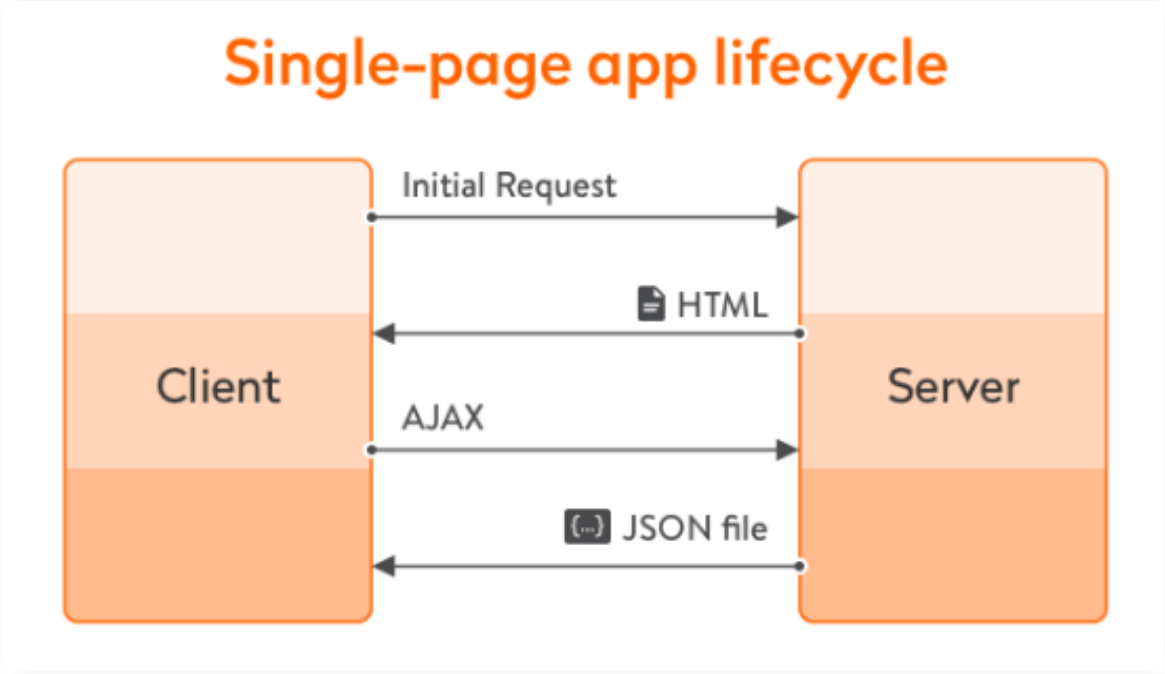
MPA funguje klasickým způsobem, to znamená, že každá změna stránky je provedena odesláním požadavku na server a stažením stránky vykreslené na straně serveru. Zobrazení nově stažené stránky nastane po obnovení okna prohlížeče. Tento proces se opakuje s každým dotazem na server. Nevýhodou je tedy nízká interaktivita se vstupy uživatele a obecně nízká uživatelská přívětivost, stejně jako vysoké nároky na výpočetní techniku na straně serveru. MPA jsou dobrou volbou v případě e-shopů, kvůli lepšímu Search engine optimization známé spíše pod zkratkou SEO. Příkladem použití MPA jsou e-shopy jako Amazon.com nebo eBay.com, které obsahují obrovské množství produktů a celkově obsahu. Nevýhodami jsou pomalé načítání, protože při každé interakci dojde ke stažení obsahu ze serveru a jeho zobrazení až po obnovení okna prohlížeče.




*Obrázek 1 - schéma životního cyklu MPA (Yalantis 2020)*

## 4.3 Princip fungování SPA

Jak již bylo zmíněno v úvodu sekce, po prvotním dotazu na server dojde ke stažení HTML struktury, CSS a JavaScriptu pouze za účelem prvního načtení aplikace. Poté již veškerá komunikace se serverem probíhá asynchronními dotazy na API, které zpravidla vrací data ve formátu JSON, případně jakémkoliv jiném formátu dat. Překreslena je pouze část stránky, ve které ke změně došlo. Nedochází tedy k obnovení celého okna prohlížeče. Výpočetní nároky jsou v tomto případě z části přeneseny na stranu klienta. SPA jsou vhodné pro weby, kde dochází k častým interakcím uživatele s aplikací, které jsou vykonávány asynchronně, takže není nutné obnovovat okno při odpovědi ze serveru.




*Obrázek 2 - schéma životního cyklu SPA (Yalantis 2020)*

Příkladem mohou být aplikace jako je Trello, nebo Gmail. Jako hlavní nevýhodu SPA bylo nejčastěji označované SEO. Bylo tomu tak z důvodu, že při navštívení webu crawlerem nebyl dostupný žádný obsah, protože o vykreslování obsahu se staral JavaScript, proto se nedoporučovalo SPA používat pro blogy nebo e-shopy, které jsou na SEO silně závislé. Stejně rychle jako se vyvíjejí webové technologie se však vyvíjí i crawlery, takže toto už neplatí zhruba od roku 2015, kdy Google oznámil, že problémy s indexací SPA jsou již minulostí. (Google 2015)

Další nevýhodou je složitější zajištění bezpečnosti aplikace. Celý zdrojový kód je dostupný na straně klienta, není možné např. skrýt API klíč. Řešením tohoto problému může být například JSON Web Token (JWT). K tomuto řešení je však stále zapotřebí server, oproti kterému by se uživatel musel autentizovat. Další možností jsou univerzální JavaScriptové aplikace, které umožňují vykonávání kódu na straně serveru. Více o těchto aplikacích je pojednáváno v 5. kapitole.

## 4.4 Nástroje pro vývoj a podporu vývoje SPA

Vývoj SPA je podpořen celou řadou nástrojů, které tvoří ucelený ekosystém postavený kolem jazyku JavaScript. Tyto nástroje nejsou pro vývoj nutností, avšak jsou tím, co dělá vývoj v JS tak atraktivním. V následujících odstavcích jsou vyjmenovány nejpodstatnější nástroje pro vývoj takových aplikací.

### 4.4.1 Node.js

Node.js je multiplatformní běhové prostředí (runtime environment) pro JavaScript, umožňující spouštění JavaScriptu mimo prohlížeč – na serveru. Tím JS nabyl nový rozměr a stal se jediným programovacím jazykem běžícím v prohlížeči i na serveru. Díky neblokujícímu (non-blocking) I/O modelu, může být Node.js rychlejší ve zpracovávání requestů než ostatní webové servery. Právě limity Apache HTTP serveru byly motivací pro vývoj Node.js.

### 4.4.2 npm

Zkratka npm znamená Node Package Manager. Jedná se o největší softwarový registr na světě. (npm 2020)

npm se skládá ze 3 částí:

- webová stránka – obsahuje seznam všech balíčku, včetně jejich popisů a často i ukázkou použití
- CLI – uživatelské rozhraní v příkazovém řádku sloužící pro správu a instalaci jednotlivých balíčků
- registry – veřejná databáze JavaScript knihoven

### 4.4.3 Yarn

Yarn je alternativou k npm CLI a využívá stejný zdroj dat, tedy npm repository. Stojí za ním několik v oboru známých korporací, jmenovitě Facebook, Exponent, Google a Tilde. Výhodami Yarnu je cachování balíčků a instalace balíčku paralelně, což má za

následek znatelné zrychlení. Jednou z dalších předností Yarnu je řešení problému monorepozitářů, tedy existence několika package.json souborů v rámci jednoho projektu, použitím nástroje **Workspaces**. (Yarn 2020)

### 4.4.4 Babel

Babel je JavaScriptový transpiler, který umožňuje používání nových ECMAScript standardů v době, kdy ještě neproběhla jejich implementace do prohlížečů a Node.js. Zpravidla se nedá nikdy říct, kdy a který prohlížeč (nebo Node.js) provede implementaci nového standardu. Z tohoto důvodu vznikl Babel, který transpiluje (překládá) nové ES verze (ES2015, ES2016, ES2017 atd.) do obecně podporované verze ES5.

### 4.4.5 webpack

Webpack je nástroj pro vytváření JavaSciptových balíčků z modulárně psaného kódu. Funguje tak, že shromáždí modulárně psaný kód, ze kterého následně vytvoří jeden balíček, který je poté importován do stránky. Webpack není omezený pouze na JS, ale podporuje také další typy jazyky, a tedy i souborové formáty. Může se jednat například o jsx, css, sass, less, obrázkové formáty jako png, jpeg a svg a v neposlední řadě datové formáty typu json, yaml nebo xml. Webpack obsahuje nástroje pro minifikaci a uglifikaci kódu. V případě minifikace se jedná o odebrání white-spaces a odstranění nepotřebných částí jako jsou komentáře, uglifikace udělá kód pro lidské oko nečitelný tím, že přejmenuje všechny proměnné a funkce na pouhá písmena. Uglifikace je na rozdíl od minifikace nezvratitelný proces.

## 4.5 API

Zkratka API náleží rozhraní pro komunikaci (výměnu dat) mezi aplikacemi, celým názvem Application Programming Interface. Existují mnohé standardy API, ale v kontextu JavaScriptových aplikací uvažujeme především standard REST a GraphQL.

### 4.5.1 REST

Celým názvem Representational state transfer je architektura definující způsob přenosu dat pomocí HTTP volání. V této architektuře je každá URI (adresa zdroje) přístupovým bodem pro různé zdroje dat. Tyto přístupové body naslouchají určitým akcím, které jsou definované HTTP metodou. Na základě této akce vrací server požadovaná data.

Základními HTTP metodami jsou:

- GET (dotaz na jeden či více záznamů)
- POST (vytvoření záznamu)
- PUT (úprava existujícího záznamu)
- DELETE (odstranění záznamu)

Zjednodušený příklad volání RESTful API:

- [GET] http://localhost:3000/api/users - vrací seznam všech uživatelů
- [POST] http://localhost:3000/api/users - založení nového uživatele
- [GET] http://localhost:3000/api/users/1 - vrací konkrétního uživatele s ID 1
- [DELETE] http://localhost:3000/api/users/1 - smaže uživatele s ID 1

### 4.5.2 GraphQL

GraphQL byl do roku 2015 proprietárním projektem společnosti Facebook, který je nyní open-source. Jedná se tedy o poměrně nový a moderní dotazovací jazyk, který se těší velké popularitě a je zároveň považován za alternativu a nástupce REST API. GraphQL využívá pouze jeden přístupový bod. Dotazy na API se provádí popisem struktury dat (query), které očekáváme, že nám API vrátí. Odpověď z API poté přesně odpovídá struktuře poskytnuté v dotazu. To nám umožňuje pracovat pouze s daty, které v aplikaci potřebujeme a také nám to umožňuje snížit počet API dorazů na nutné minimum. (Rascasone 2020)

Samotný API server stále pracuje s množinou všech dostupných dat (schema), nazpět však posílá pouze data požadovaná v dotazu. To znamená, že je nutné stále brát na vědomí výpočetní náročnost těchto dotazů.

GraphQL je silně typový, dotazy jsou validovány před odesláním na server, což snižuje četnost chyb v syntaxi nebo jiných chyb, které mohou dotaz činit neplatným. Zároveň je introspektivní, což znamená, že vývojáři se mohou dotazovat na dostupné typy. (Rascasone 2020)

GraphQL podporuje čtení i zapisování (mutování) dat. Je také možné v reálném čase sledovat změny dat.

```
{
  repositoryOwner(login: "mbfs-3nt") {
    repositories (first: 1) {
      nodes {
        id
        name
        issues (first: 4) {
          nodes {
            id
            number
            title
          }
        }
      }
    }
  }
}
```

```
{
  "data": {
    "repositoryOwner": {
      "repositories": {
        "nodes": [
          {
            "id": "MDEwOlJlcG9zaXRvcnk0NjkxNw==",
            "name": "easy-project",
            "issues": {
              "nodes": [
                {
                  "id": "MDU6SXNzdWUyMTY1NjM=",
                  "number": 1,
                  "title": "Bug Fixes after upgrade"
                },
                {
                  "id": "MDU6SXNzdWUyMTY1NjQ=",
                  "number": 2,
                  "title": "Prepare Oracle DB Server"
                },
                {
                  "id": "MDU6SXNzdWUyMjMzMTE=",
                  "number": 3,
                  "title": "Test before upgrade"
                },
                {
                  "id": "MDU6SXNzdWUyNTIwMjk=",
                  "number": 4,
                  "title": "Update Apollo"
                }
              ]
            }
          }
        ]
      }
    }
  }
}
```

*Ukázka kódu 1 - příklad GraphQL dotazu a odpovědi. (autor)*

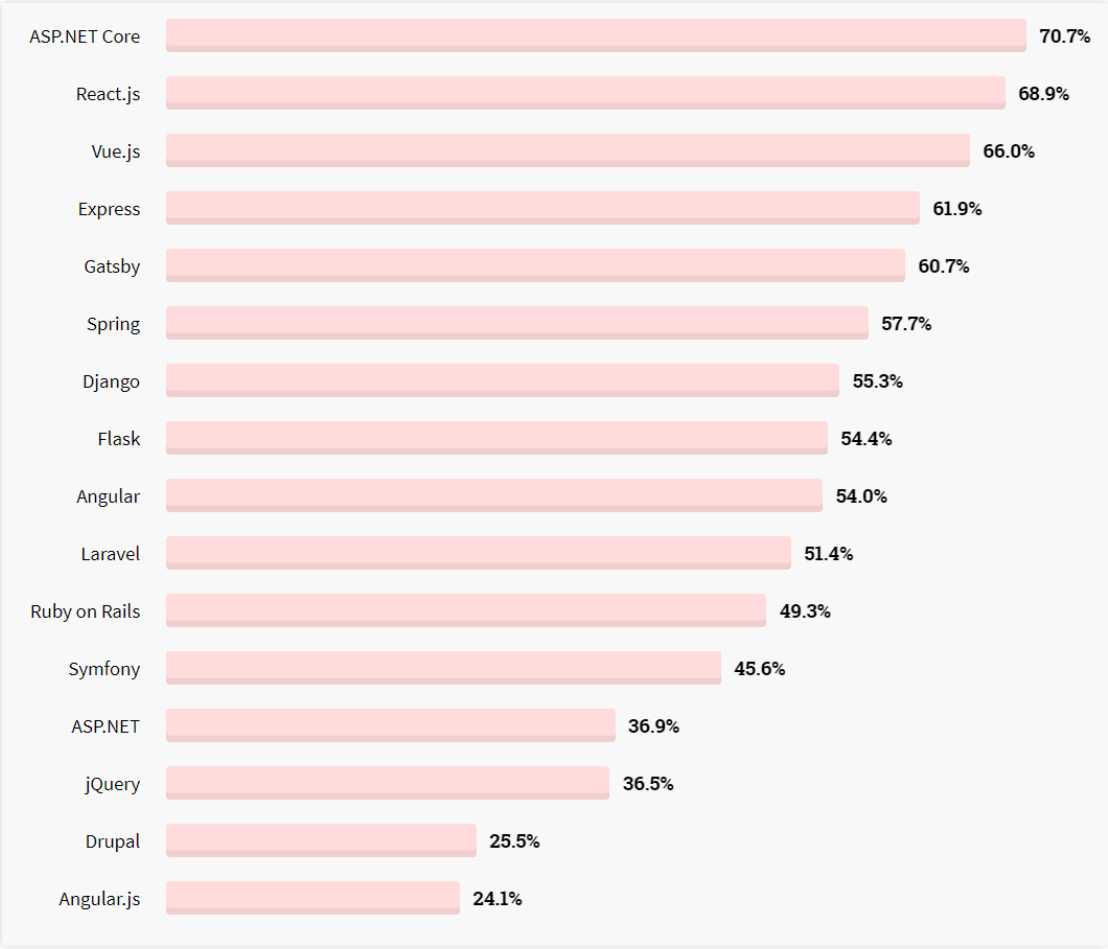
## 4.6 Frameworky pro vývoj SPA

Základní definice frameworku by se dala vyjádřit takto: JavaScriptový framework je kolekce knihoven, které vývojáři poskytují předpřipravený kód pro rutinní programování – rámec pro vývoj webových aplikací. (Skillcrush 2020)

Důležité je také vymezení rozdílu mezi knihovnou a frameworkem. V případě knihovny, vývojář rozhoduje, kdy knihovnu použít, respektive kdy zavolá funkci, kterou knihovna obsahuje. Má tedy pod kontrolou běh aplikace. V případě frameworku má běh aplikace pod kontrolou framework a vývojáři pouze poskytuje prostor pro implementaci jeho vlastního kódu. Kód je ovšem volán frameworkem.

Aktuálně existuje velké množství front-endových JavaScipt frameworků, je však jednoduché určit tři nejpoužívanější a také nejznámější z nich.

Jedním z kritérií je každoroční průzkum mezi vývojáři organizovaný serverem StackOverflow.com. Aktuální průzkum v roce 2020 přináší tyto data.




*Graf 4 - milované webové frameworky (StackOverflow 2020)*

Na obrázku je znázorněn graf označovaný jako „Loved Web Frameworks“ v překladu „Milované Webové Frameworky“. Z něhož vyplývá, že pokud uvažujeme pouze JS frameworky, tak na první příčce je React.js, na druhé Vue.js a na třetí Angular. Za zmínku stojí i fakt, že 7 z 16 frameworků je napsaných v jazyce JavaScript. Krom výše

již zmíněných se jedná ještě o Express, Gatsby, který bude rozepsán v další kapitole, jQuery jako představitel historie JS frameworků a posledním je Angular.js – starší verze Angularu.

Jako jeden z dalších ukazatelů může posloužit počet stažení, nebo počet hvězd na serveru GitHub.

### 4.6.1 Společné rysy frameworků

Hlavním cílem frameworků je, aby ušetřily práci vývojáři a tím zkrátily nutnou dobu pro vývoj aplikace. Způsob, jakým ušetřit čas a prostředky je udržování stavu, tedy mít synchronizované UI s daty. Jak říká Alberto Gimeno (2018) „The biggest, by far, improvement these frameworks provide is having the ability to implement UIs that are guaranteed to be in sync with the internal state of the application“.

Frameworky používají dva způsoby, jak toho dosáhnout:

- **Překreslit celou komponentu** – **React**. V momentě, kdy se stav komponenty změní, vykreslí virtuální DOM a porovná ho se snapshotem virtuálního DOM, poté vyhodnotí změny a ty promítne do reálného DOM. Tomuto procesu se říká rekonciliace.
- **Hlídat změny pomocí pozorovatelů** – **Angular a Vue**. Stavové proměnné jsou pozorovány a v případě, že se změní, pouze ty elementy DOM, ve kterých se změnily hodnoty při renderování jsou aktualizovány.

Poměrně významnou podobností je i skutečnost, že ke všem zmíněným frameworkům existují knihovny, které umožňují jejich použití jakožto nativní mobilní aplikace pro různé platformy jako je iOS nebo Android.

- React – React Native
- Angulare – NativeScript
- Vue – Weex

### 4.6.2 React

React ve skutečnosti není framework, ale knihovna pro tvorbu uživatelských rozhraní, přesto je důležité ho zmínit. Proto aby byl React plnohodnotným frameworkem jsou zapotřebí další knihovny, například pro globální state management nebo routování. React je open-source a aplikuje funkcionální paradigma. Vyvíjí ho společnost Facebook již od roku 2013 a jeho původním autorem je Jordan Walke, zaměstnanec Facebooku. Co se týká popularity, je obecně nejoblíbenější knihovnou. Vzhledem k tomu, jak je React rozšířený, je pro vývojáře snadné najít nespočet různých zdrojů, které mohou pomoct s osvojením této knihovny. Pokud přijdete k Reactu jako nováček, můžete si být téměř jisti, že na problematiku, na kterou hledáte řešení, narazil někdo jiný dávno před vámi. Právě komunita kolem daného frameworku/knihovny by se dala označit za jeden z nejdůležitější rysů při jeho výběru.

#### 4.6.2.1 Komponenty

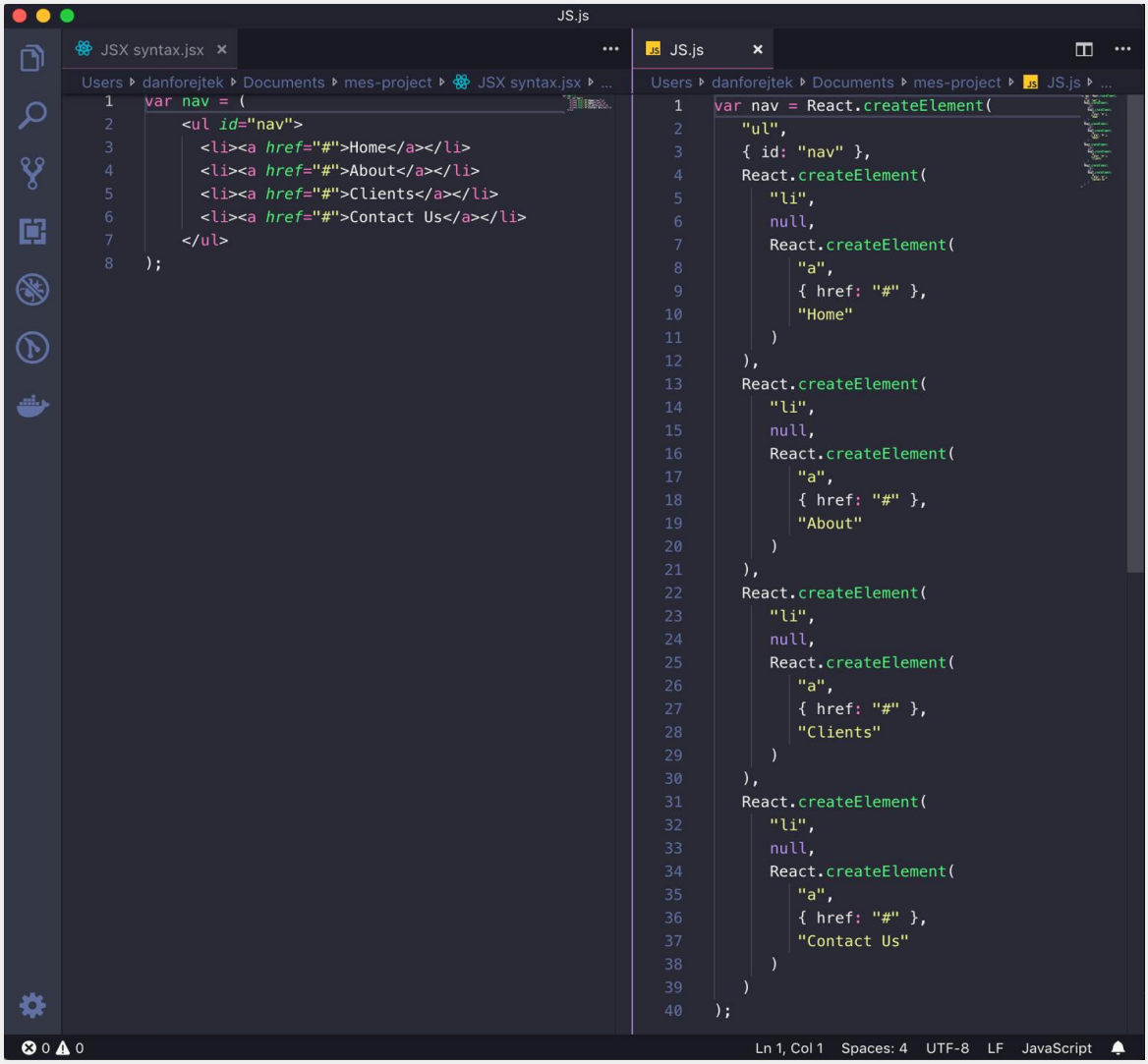
Základním principem Reactu je kompozice komponent. Existují dva druhy komponent, první je class (stateful) komponenta, před příchodem React Hooks ve verzi 16.8 bylo možné spravovat state pouze v této komponentě. Druhým typem je functional (dříve také označována jako stateless) komponenta. Výhodou komponent je jejich znovu-použitelnost v rámci projetu, kdy odpadá nutnost zbytečné duplikace kódu. Komponenty je možné libovolně zanořovat. (Copes 2019)

React aplikuje **jednosměrný tok dat** (Unidirectional Data Flow). V kontextu této knihovny to znamená, že existuje pouze jeden způsob, jakým se mohou data přesouvat částmi aplikace. Každá komponenta má svůj *state (stav)*, ten je možný přenášet na *child (potomka)* komponenty pomocí *props (vlastností)*. *State* je možné měnit pouze na základě *akce*. (Copes 2019)

#### 4.6.2.2 JSX

JSX (JavaScript Syntax Extension) je sice syntaxí podobný HTML, ale jedná se o syntaktické rozšíření JavaScriptu, které se kompiluje do React elementů. Použití JSX není nutností, avšak je v kombinaci s React žádoucí (nedává příliš smysl používat React bez JSX). Kód psaný v JSX je velice dobře čitelný a díky syntaxi podobné HTML je

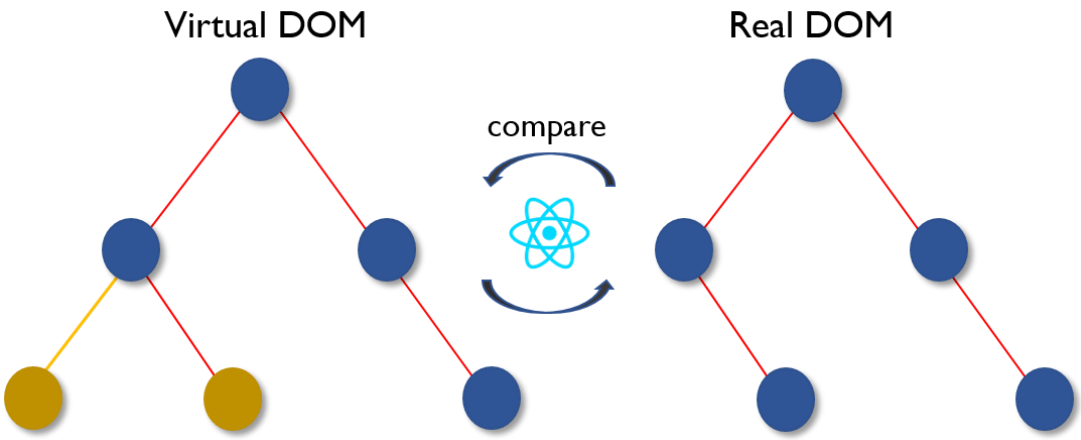
jednoduchý na naučení, drobné změny přichází až v názvech atributů. Pravidlem je, že každá komponenta musí vracet právě jeden element, který však může obalovat další elementy, jak můžeme vidět na obrázku, kde element <ul> obaluje <li> elementy.




*Ukázka kódu 2 - zápis v JSX (na levé straně) a transpilovaný JSX kód (na pravé straně) (autor)*

#### 4.6.2.3 Virtual DOM

Virtual DOM optimalizuje interakci s prohlížečem. React si drží v paměti prohlížeče zjednodušenou kopii DOM objektu – Virtual DOM. Každá změna *state* (dat) se nejprve provede v rychlejším a jednodušším Virtual DOM, poté React takzvaným *diffem* vyhodnotí všechny změny aplikované na Virtial DOM, v poslední fázi React přenese zjištěné změny do skutečného DOM objektu. Tím dojde k překreslení pouze těch elementů, které byly ovlivněny změnou *state*. (Codecademy 2020)




*Obrázek 3 - vizualizace porovnávání Virtual DOM s reálným DOM (Pngitem 2020)*

### 4.6.3 Vue

Framework Vue je inspirován z části Angularem a z části Reactem. Po svém prvním vydání na GitHub začalo Vue získávat obrovskou popularitu mezi vývojářskou komunitou, aktuálně je nejlépe hodnocenou knihovnou na GitHub, co do počtu hvězd (způsob, kterým ostatní uživatelé označují své oblíbené knihovny). Křivka učení je označována jako strmá, proto je Vue vhodnou volbou pro začínající vývojáře bez předchozích znalostí jiných frameworků. Vue je open-source a aplikuje patern Model-view-viewmodel, hlavní knihovna se soustřeďuje na zobrazovací vrstvu. Routování a state management je rozdělen do samostatných oficiálních knihoven, které jsou vyvíjeny a spravovány týmem stojícím za Vue. (Vue 2020)

#### 4.6.3.1 Templates

Jednoduchost Vue spočívá v jeho syntaxi založené na HTML. Všechny Vue templates (šablony) jsou validním HTML. Single-file šablona může obsahovat HTML obalený tagem <template>, JavaScript obalený tagem <script> a také CSS styly obalené ve <style>. V případě stylů se dá atributem měnit, zda se jedná o globální nebo pouze lokální (scoped) styly. Vue však podporuje i množství dalších možností, jak definovat template, například JSX. Oproti Reactu a Angularu má tu výhodu, že není třeba se učit další superset JavaScriptu jak tomu je v případě Reactu (JSX) a Angularu (TypeScript), což by se dalo označit jako hlavní důvod proč je učící křivka Vue strmější než u ostatních zmíněných frameworků. (RubyGarage 2020)


*Ukázka kódu 3 - příklad Vue komponenty (autor)*

#### 4.6.3.2 Vue direktivy

Vue template je validní HTML, k poskytnutí dynamiky používá takzvané direktivy. Příkladem direktiv jsou

- v-if / v-else – podmíněné příkazy
- v-once – komponenta, která se nepřekresluje
- v-for – for cyklus
- v-on – event listener
- v-html – vložení HTML

```HTML
<div v-if="Math.random() > 0.5">
  Now you see me
</div>
<div v-else>
  Now you don't
</div>
```

*Ukázka kódu 4 - příklad použití Vue direktiv (Vue 2020)*

Direktivy se přidávají jako atributy elementů. Je také možné vytvářet vlastní direktivy, případně je stahovat jako npm balíčky.

### 4.6.4 Angular

Angular je nástupcem AngularJS a je plnohodnotným frameworkem, který obsahuje nástroje pro načítání dat, správu stavu, vlastní vývojový jazyk TypeScript a další vývojářské nástroje. TypeScript usnadňuje přechod těm, kteří dříve vyvíjeli v objektově orientovaných jazycích jako je C# nebo Java, kterými je inspirován. Cílovou skupinou Angularu je především podniková sféra s komplexními projekty. Angular není vhodný pro začínající vývojáře, jeho učící křivka je pozvolná. Jako hlavní příčina se dá označit TypeScript, který přidává na obtížnosti. Na druhém místě by se dalo označit programovací paradigma OOP, které je složitější na naučení oproti funkcionálnímu. Samotná popularita v konkurenci Reactu a Vue je také na ústupu. Přesto co se týká obsazení na trhu práce má Angular velký podíl především protože jsou v něm napsané velké projekty velkých korporací. Za zmínku stojí i nevhodnost Angularu pro menší projekty, a to kvůli své robustnosti.

Angular se skládá z těchto knihoven:

- @angular/common
- @angular/compiler
- @angular/core
- @angular/forms
- @angular/http

- @angular/platform-browser
- @angular/platform-browser-dynamic
- @angular/router
- @angular/upgrade

## 4.7 Závěr

SPA mají bezpochyby vyšší uživatelskou přívětivost, stejně tak i stále rostoucí uživatelskou základnu mezi vývojáři a společnostmi, které na tuto technologii přechází. U SPA jsou však i obecně známé nedostatky jako SEO. Hlavním důvodem je, že webové crawlery známé také jako roboty, prochází webové stránky za účelem jejich indexace. Data získaná robotem jsou následně vyhodnocována a jejich výstupem je hodnocení webu, které se promítne v pořadí, v jakém je stránka zobrazena uživateli vyhledávače. Samozřejmě čím výše web je, tím se úměrně zvyšuje pravděpodobnost, že ji uživatel navštíví – jedná se tedy o velmi důležitý parametr. Princip, na kterém SPA fungují však zamezuje robotům procházet obsah webu, protože obsah není dostupný okamžitě po navštívení webu, je stahován a vykreslen až po načtení všech skriptů. Navíc roboty nemohou najít další linky na stránky, které by mohly navštívit a ve výsledku zaindexují pouze úvodní stranu. Poté zašlou stránku indexeru, který ji vykreslí, získá z ní odkazy na další stránky, které opět pošle indexeru a tak stále dokola. Crawler tento proces velice zpomaluje a dělá ho neefektivním. Tento problém řeší *univerzální JavaScriptové aplikac*e, o kterých pojednává následující kapitola.

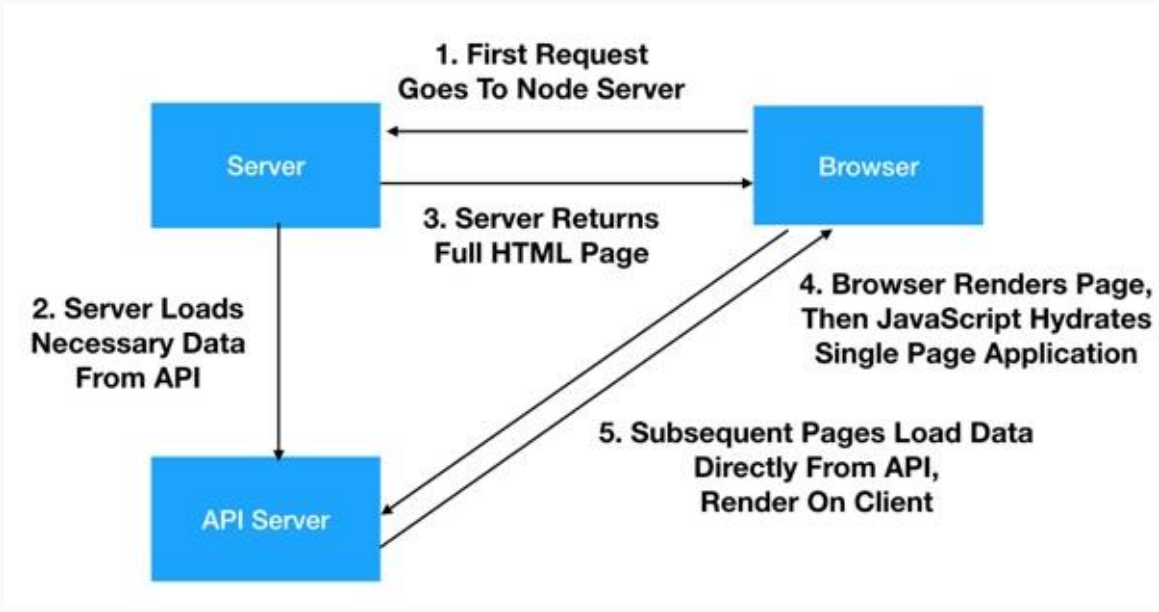
# 5 Univerzální JavaScriptové aplikace

Jedná se o moderní přístup k tvorbě jednostránkových webových aplikací v jazyce JavaScript. Některé zdroje označují tento typ aplikací jako *izomorfní*. Myšlenkou za tímto názvem je, že *aplikace mají stejný formát (ἴσος: isos), nebo tvar (μορφή: morphe) nehledě v jakém prostředí běží, ať je to prohlížeč či server.* (Oreilly 2015)

Tento druh aplikace používá JS jak na straně klienta, tak i na straně serveru. To napomáhá k vyšší rychlosti prvního načtení stránky. Po prvním načtení se aplikace

chová prakticky stejně jako klasické SPA a aktualizuje obsah AJAX dotazy na pozadí. (Mardan 2016)

Aby byl JavaScript kód izomorfní, je třeba ho zbavit vlastností (atributů, proměnných, funkcí), které jsou specifické pro dané prostředí. Jako příklad může posloužit *window* objekt, který je dostupný pouze na straně prohlížeče, či *request*, který je naopak dostupný pouze na straně serveru. V některých případech nám s touto problematikou mohou pomoci nástroje jako je například webpack. Aktuálně je možné si všimnout trendu, kterým jsou psány různé knihovny, příkladem je knihovna isomophic-fetch, která si zachovává svou funkcionalitu nezávisle na běhovém prostředí.

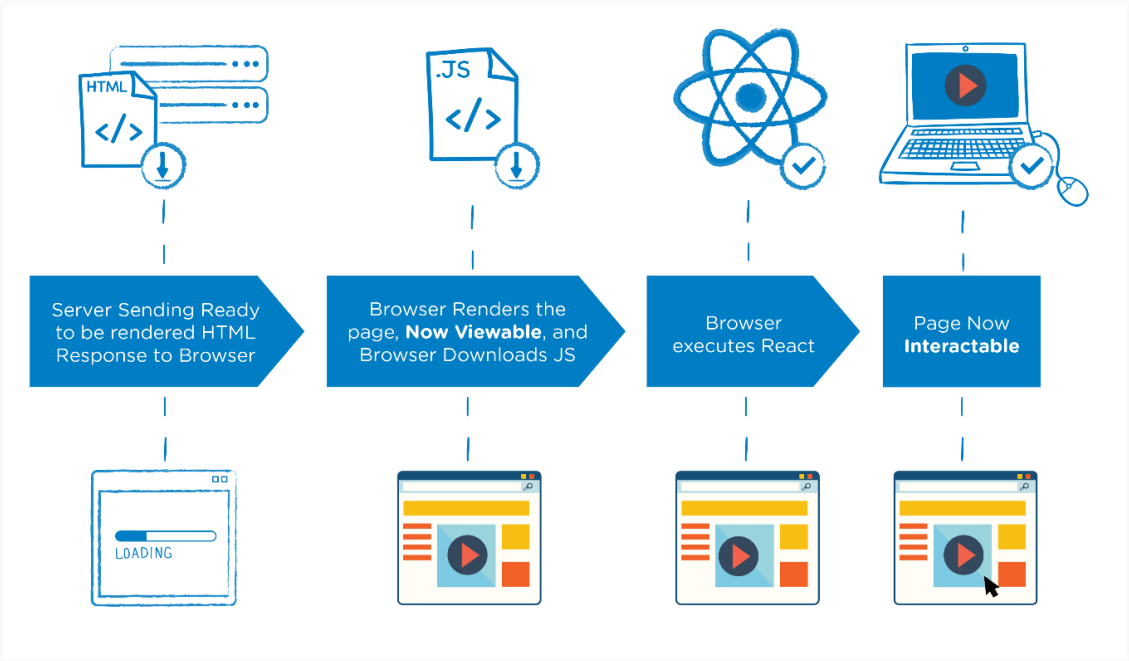



*Obrázek 4 - posloupnost komunikace univerzálních aplikací (ZenDev 2018)*

## 5.1 Server-side rendering

Server-side rendering umožňuje vykreslení obsahu stránky na straně serveru. Klientovi je odesláno vygenerované HTML, stránka je tedy plnohodnotně zobrazena dříve, než se stáhne JavaScriptová aplikace, která by v případě vykreslování na straně klienta vykreslila HTML až po svém kompletním stažení. Tímto přístupem však roste náročnost logiky aplikace a klade větší nároky na vývojáře. Je také třeba si uvědomit, že mezi klienty existují určité rozdíly, se kterými je při vývoji třeba počítat. Výslednými přínosy server-side renderingu je lepší SEO, zkrácení doby do prvního načtení „meaningful“ obsahu a obecně rychlost celého webu. (Magnolia 2020)

**Hydratace (hydratation)** je proces přenesení stavu ze strany serveru na stranu klienta, tím dochází k přeměně statické HTML stránky na single page aplikaci. (Gatsby.js 2020)



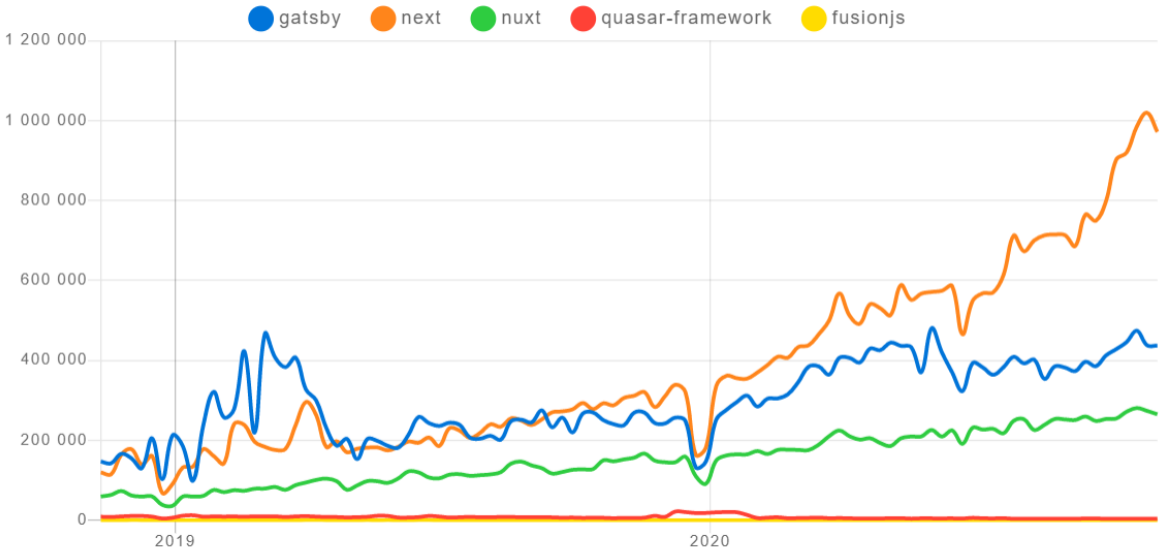

*Obrázek 5 - popis fungování vykreslování na straně serveru (Walmart Labs 2017)*

## 5.2 Frameworky pro vývoj univerzálních JavaScriptových aplikací

Univerzální frameworky si dávají za úkol potlačit nevýhody Single Page aplikací, tím že poskytují také back-end, na kterém je například možné pre-renderování či generování statických stránek. Code-splitting a další optimalizace podávání zdrojů webovému prohlížeči. Při požadavku je tedy pro uživatele vykreslena stránka na straně serveru, takže dostane statický obsah, který je poté aktualizován (rehydratován) pomocí JavaScriptu. Tím je aplikace přívětivá jak pro uživatele, který nemusí čekat na načtení po stažení aplikace, především ale i pro SEO, které je dlouhodobě označované jako největší nevýhoda SPA.

Těchto frameworků není příliš mnoho. Obecně jsou v komunitě vývojářů proti sobě srovnávány 3 frameworky a to Next.js, Nuxt.js a Gatsby. To že si mezi sebou konkurují

převážně tyto frameworky dokazuje i srovnávací stránka vytvořená týmem stojícím za Gatsby. Dalšími méně užívanými, a tedy i známými jsou například Quasar nebo Fusion.




*Graf 5 - četnost stažení za poslední 2 roky (npmrends 2020)*

| | stars | issues | updated | created |
|---|---|---|---|---|
| gatsby | 47 762 | 599 | Nov 11, 2020 | May 22, 2015 |
| next | 56 760 | 825 | Nov 11, 2020 | Oct 6, 2016 |
| nuxt | 31 266 | 407 | Nov 11, 2020 | Oct 26, 2016 |
| quasar-framework | 16 734 | 303 | Nov 11, 2020 | Oct 5, 2015 |
| fusionjs | | | Nov 22, 2017 | Feb 25, 2017 |

*Obrázek 6 - srovnání popularity frameworků (npmtrends 2020)*

### 5.2.1 Next.js

Next.js je open-source framework postavený na knihovnách React, Babel a Webpack. Je vyvíjen společností Vercel (dříve ZEIT), která zároveň poskytuje Server-less hosting uzpůsobený tomuto frameworku. Toto složení umožňuje vývojáři velice snadno založit nový projekt s prakticky nulovou nutností počáteční konfigurace, jelikož je postaráno o bundlování pomocí webpacku, transpilace novějších verzí JavaScript na starší verze

široce podporovány prohlížeči pomocí Babel. Next se také stará o optimalizaci ve formě rozdělení kódu (code splitting) podle jednotlivých stránek, nikdy se tedy nestahuje najednou celý kód aplikace. Next.js bere svou inspiraci z PHP, například v případě routeru, který se konfiguruje pomocí souborů či složek vytvořených ve složce „pages“. Next umožňuje také psaní API endpointů, stačí pouze napsat handler funkci, které jsou předány parametry request a result. Bez nutné konfigurace umožňuje také psaní CSS v JavaScript souboru díky systému *styled-javascript*. To s sebou přináší optimalizaci, kdy se ze serveru stahuje pouze na stránce používané CSS, a ne celý bundle, jak je tomu u klasických webových aplikací. (Next.js 2020)

#### 5.2.1.1 Routing

Routování je založené na souborovém systému. Každý soubor exportující funkci ve složce *pages* je automaticky dostupný jako cesta. Router podporuje také vnořené cesty, kdy stačí založit složku a do ní vložit soubor s exportovanou funkcí.

Příklad index a vnořené cesty:

- pages/index.js → /
- pages/example.js → /example
- pages/example/index.js → /example
- pages/example/first.js → /example/first

Next.js podporuje také dynamické routování, které je možné vytvořit hranatými závorkami v názvu souboru nebo složky.

Příklad dynamické cesty:

- pages/blog/[slug].js → /blog/slug-example
- pages/[username]/settings.js → /dan/settings

#### 5.2.1.2 Data fetching

Next.js si nevynucuje žádný specifický způsob jakým mají být data zpracována. Je možné používat jakékoliv knihovny pro získávání dat jak na front-endu tak na back-endu. Next nabízí několik metod, které jsou spouštěny v různých fázích životního cyklu dotazu. Každá z těchto funkcí vrací data komponentám jako props.

**getStaticProps** umožňuje zpracování dat v build time (před requestem). Tato funkce je vhodná pro stahování dat z headless CMS. Tato data by neměla být specifická dle uživatele. Použití je vhodné pro stránky, na kterých je kladen důraz na CEO.

**getStaticPaths** pro účely generování statických stránek.

**getServerSideProps** pro účely vykreslování na straně serveru.

#### 5.2.1.3 Nasazení

Next.js zjednodušuje také nasazení (deployment) aplikace, která může být spuštěna na jakémkoliv prostředí, které podporuje Node.js, díky poskytnutým příkazům *next build*, který vytvoří optimalizované build aplikace a následně příkaz *next start* ji spustí. Doplňkovým příkazem je *next export*, který vyexportuje stránky do statického HTML. Tento ekosystém doplňuje CLI aplikace Now, která provede nasazení projektu na produkční prostředí server-less služby Vercel.

Next.js je zároveň možné spouštět na jiných Node.js server frameworcích, jako je například Express nebo Fastify.

#### 5.2.1.4 Závěr

Framework Next.js je všestranným nástrojem pro tvorbu webových aplikací. Umožňuje jak vykreslování na straně serveru, tak i automatickou statickou optimalizaci. Navíc dokáže sloužit i jako generátor statických stránek.

Routování a obecně tvorba stránek je velice intuitivní, a to včetně tvorby dynamických rout.

Důležitými vlastnostmi je přístup, ke stahování dat, který je v případě tohoto frameworku zcela v rukou vývojáře. Velkým bonusem jsou i API routy, které nahrazují nutnost další aplikace starající se o poskytování dat.

Tým stojící za frameworkem stojící je také velkým příslibem jeho budoucího rozvoje.

### 5.2.2 Gatsby

Gatsby je open-source framework pro tvorbu statických stránek za pomocí knihovny React a dotazovacího jazyka GraphQL, jehož hlavním zaměřením je rychlost. Gatsby

funguje na jednoduchém principu, kdy kompiluje React aplikaci do statických HTML stránek, které jsou extrémně rychlé. Přináší s sebou také celý ekosystém jako jsou pluginy, grafická témata nebo starter projekty, které mohou posloužit jako vzor pro budoucí aplikace. Všechna data nutná pro vygenerování stránek, jako například seznam článků, musí být zpracována aplikací a za tímto účelem je nutné je předávat pomocí GraphQL. Gatsby umožňuje tvorbu aplikaci podobných stránek, díky hydrataci na straně klienta, čímž se statická stránka může proměnit na React aplikaci, která se následně stará o renderování obsahu.

#### 5.2.2.1 GraphQL datová vrstva

Gatsby je specifický svou datovou vrstvou, která je postavená kolem GraphQL. V dokumentaci označují získávání dat jako plugin-driven. V praxi to znamená, že pro každý zdroj dat (markdown, CMS) je třeba nainstalovat plugin. Tato data jsou poté v aplikaci dostupná jako tvz. uzly (nodes). Tyto uzly je také možné založit i bez pluginu a to metodou createNode(). (Gatsby 2020)

gatsby-node.js

JS

```
exports.sourceNodes = ({ actions, createNodeId, createContentDigest }) => {
  const pokemons = [
    { name: "Pikachu", type: "electric" },
    { name: "Squirtle", type: "water" },
  ]

  pokemons.forEach(pokemon => {
    const node = {
      name: pokemon.name,
      type: pokemon.type,
      id: createNodeId(`Pokemon-${pokemon.name}`),
      internal: {
        type: "Pokemon",
        contentDigest: createContentDigest(pokemon),
      },
    }
    actions.createNode(node)
  })
}
```

*Ukázka kódu 5 - tvorba datových uzlů (Gatsby 2020)*

#### 5.2.2.2 Pluginy

Framework je velice modulární. Jednotlivé funkcionality jsou přidávány pomocí pluginů. Rozlišuje se zde mezi oficiálními pluginy vytvořenými autory frameworku a komunitní, které tvoří samotní uživatelé. Gatsby poskytuje přehlednou webovou knihovnu všech dostupných pluginů.

Příkladem typů dostupných pluginů jsou například

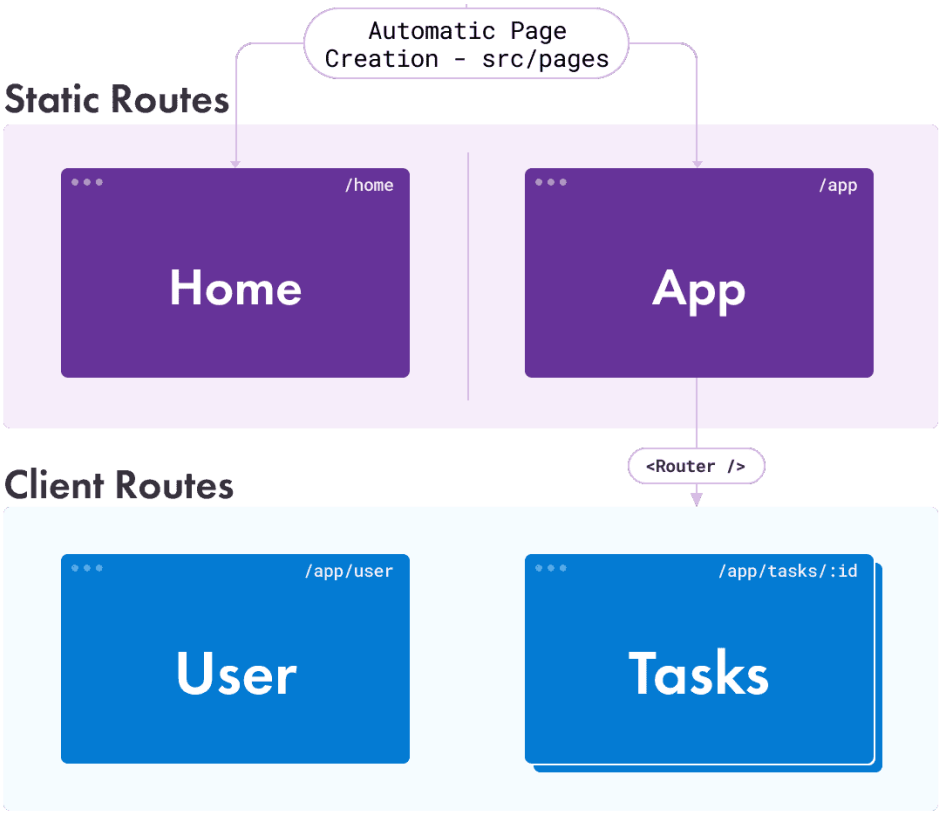
- Napojení externích dat a obsahu do datové vrstvy z CMS, statických souborů nebo REST API
- Transformace různých datových formátů např. z Markdown, YAML nebo CSV do JSON
- Optimalizace obrázků
- Přidání služeb třetích stran jako jsou Google Analytics nebo Instagram
- Přidání předkonfigurovaných funkcionalit jako jsou například témata (themes)

Pluginy jsou instalovány jako npm balíčky, které je následně třeba přidat do konfiguračního souboru **gatsby-config.js**.

#### 5.2.2.3 Routování (tvorba stránek)

Stránky je možné tvořit podobně jako je tomu u frameworku Next.js. Každý soubor exportující React komponentu ve složce **src/pages** reprezentuje routu.

Gastby nenabízí možnost tvorby dynamických cest. Tato funkcionalita je řešena routami na straně klienta pomocí komponenty **<Router />**. Dokumentace tento přístup nazývá **Hybrid app pages** (stránky hybridní aplikace).




*Obrázek 7 - schéma routování v Gatsby (Gatsby 2020)*

#### 5.2.2.4 Závěr

Gatsby je převážně generátor statických stránek, v čemž je také jeho největší síla. Pokud máme předem daná data, ze kterých chceme vytvořit stránky, je Gatsby správnou volbou. Použitelnost najdeme především u stránek, které jsou silně závislé na SEO a rychlosti. Hodí se proto skvěle pro tvorbu blogů, stránek dokumentace, nebo produktové stránky. Každou z těchto stránek je možné hydratací proměnit na React aplikaci. Tím je například umožněno tvořit dynamické routy na straně klienta.

Na co se Gatsby naopak nehodí, jsou aplikace pracující s často se měnícími daty. Také zde přicházíme o možnost používat globální stav. GraphQL datová vrstva je další položkou, kterou musí vývojář ovládat, aby mohl pracovat s Gatsby. Také je nutné zmínit, že tento framework neposkytuje back-endovou vrstvu, ta slouží pouze pro generování stránek v *build-time*.

Silnou stránkou je velké množství pluginů různých účelů, ale také *starter* projekty a tzv. *themes*, které přinášejí předpřipravený rámec aplikace, který stačí pouze upravit potřebám daného projektu.

Gastby využívá pro prezentaci své dokumentace například React, Apollo nebo DigitalOcean.

### 5.2.3 Nuxt.js

Nuxt.js se podobá Next.js nejenom svým názvem, ale také stejné funkcionality ať už se jedná o vykreslování na straně severu nebo generování statických stránek. Systém routování je prakticky stejný, nabízí také stejné možnosti týkající se nasazení.

Je tedy možné tento framework označit za plnohodnotnou alternativu k Next.js s rozdílným frameworkem pro tvorbu uživatelských rozhraní, kterým je v tomto případě Vue.js.

Mezi známými společnostmi využívající Nuxt.js je například Nespresso, todois, GitLab nebo také Ubisoft.

## 5.3 Závěr

Univerzální JavaScriptové aplikace vznikly za účelem smazání obecně známých nedostatků SPA aplikací. Zvyšují bezpečnost aplikace, ale také SEO, přesto jsou zapisovány pomocí stejného univerzálního jazyku, který vývojáři SPA již znají. Renderování na straně serveru výrazně zrychluje first meaningful render, který je důležitým parametrem pro uživatele, který web navštěvuje poprvé. U zmíněných frameworků existují jak společné rysy, tak i odlišnosti, které je předurčují k použití na jiných typech projektů.

# 6 Analýza vyvíjené aplikace

Aplikace má sloužit pro agregování a správu projektů, které jsou ve společnosti plánovány. K těmto projektům jsou přiřazovány jednotlivé milníky, zodpovědné osoby a další údaje. Každý projekt obsahuje dílčí úkoly, které je třeba splnit za účelem dokončení celého projektu. Jednotlivé úkoly mají určený termín dokončení (deadline) a své řešitele.

Zdrojem dat pro aplikaci je korporátní GitHub server, kde jsou jednotlivé projekty zakládány v rámci GitHub organizace. Původně se mělo jednat o GitLab, který je mnohem lépe vybaven projektovými nástroji, než tomu je v případě GitHubu, čímž se celý vývoj aplikace do jisté míry zkomplikoval. Původně se taky počítalo s použitím REST API (v3), ale to je u GitHubu poměrně omezené. Z tohoto důvodu bylo rozhodnuto o použití GraphQL API (v4).

Oproti GitLab GitHub neumožňuje stanovení deadline pro úkoly, stejně tak projekty mají zcela jinou strukturu a účel v rámci aplikace atd. Z tohoto důvody již nebyl GIT repozitář dostačujícím zdrojem dat a bylo nutné přidat vlastní databázi, která bude uschovávat dodatečná data, která není možné zadávat v GitHub.

K autentizaci uživatele nám slouží korporátní LDAP server. Jelikož se jedná o interní aplikaci, existuje předpoklad, že každý zaměstnanec má existující účet na tomto LDAP serveru. Naše aplikace tedy nespravuje přihlašovací údaje a z tohoto důvodu není třeba starat se o registraci nových uživatelů.

## 6.1 Výběr vhodného frameworku

Při výběru vhodného frameworku pro daný projekt je třeba vycházet z požadavků na výslednou aplikaci. Výběr frameworku je u projektů klíčový. Vhodně vybraný framework může vývojáři ušetřit čas a zpříjemnit vývoj. Frameworky se starají o složité části aplikace jako je transpilace, code splitting, server-side rendering, routing a další optimalizace nutné pro rychlý a bezproblémový běh JavaScriptové aplikace. Framework by měl mít také jednoduchou, srozumitelnou, a přesto dostatečně

konkrétní dokumentaci, která seznámí vývojáře se všemi funkcionalitami. Důležité jsou také praktické ukázky kódu, ze kterých je možné čerpat best-practices.

V neposlední řadě s ohledem na to, jak rychle se technologie posouvají, je třeba dát zřetel na vývojářskou základnu daného framework a předpoklad, že u daného frameworku bude docházet k dalšímu vývoji a údržbě.

### 6.1.1 Výčet funkcionalit aplikace

V našem případě se nejednalo o aplikaci, která bude vystavena na veřejném internetu. Aplikace slouží k interním účelům, a proto je dostupná jen v korporátním intranetu. Z tohoto důvodu pro nás nemá žádný význam SEO. Není tedy nezbytně nutné používat server-side rendering.

Naše aplikace konzumuje data z několika různých zdrojů. Hlavním zdrojem je GitHub GraphQL API. Pro ukládání dodatečných dat byla zvolena NoSQL databáze, konkrétně MongoDB. Tato databáze je hostována na našich lokálních Windows serverech.

K tomu abychom zpřístupnili data z databáze naší aplikaci, bylo za potřebí připravit API. Vzhledem k jednoduchosti databázové struktury bylo zbytečné se zabývat vývojem GraphQL API, které pro nás v tu dobu bylo poměrně neznámé. Proto bylo zvoleno jednoduché REST API.

Aplikace se také stará o rozesílání upozornění a oznámení pomocí emailu. Máme zde emaily rozesílané na základě akce uživatele, ale také pravidelná upozornění vyhodnocována každý den, pro tento případ bylo nutné použít CRON (plánovač úloh).

## 6.2 Popis kritérií pro rozhodování

Jelikož se v našem týmu většina kolegů nevěnuje webovému vývoji, bylo rozhodování založena především na mých zkušenostech.

Vzhledem k tomu, že jsem povětšinou používal React a měl jsem s ním předchozí zkušenosti, byla zvolena právě tato knihovna pro tvorbu uživatelského rozhraní. Tím nám z výběru odpadl framework Nuxt, který používá Vue pro tvorbu uživatelských

rozhraní. Je však třeba konstatovat že Nuxt by našim účelům posloužil stejně dobře jako Next.

Data v naší aplikaci jsou často aktualizována, proto použití static site generátoru nedává v tomto ohledu smysl. Je pro nás výhodnější používat client-side rendering a stejně tak stahování dat na straně klienta. To však nutně nevylučuje použití frameworku Gatsby.

Uvažovaná aplikace musí konzumovat naše vlastní API, u kterého bylo rozhodnuto, že bude REST, což by bylo v případě použití Gatsby komplikované, protože Gatsby vyžaduje, aby veškerá data byla procesována skrze vlastní GraphQL API v buildtime, to by zbytečně protahovalo a komplikovalo vývoj naší aplikace, především tedy routování.

V případě použití Gatsby by také nebylo možné mít CRON a obecně ani logiku starající se o rozesílání emailů v rámci jedné aplikace. Naopak v případě Next.js je možné použít vlastní server, který iniciuje Next.js. To nám umožňuje přidat CRON funkcionalitu do této jediné aplikace.

Dalším kritérium, které bylo nutné zvážit je tvorba REST API pro naše data v MongoDB. V tomto případě je velice příhodné, že Next.js bez jakékoli nutnosti konfigurace poskytuje vlastní API cesty stejně jednoduchým způsobem, jakým se řeší routování jednotlivých stránek. Nepřináší to tedy s sebou žádné další komplikace, které by bylo nutné řešit.

Považuji také za důležité zmínit, že tento projekt bude sloužit jako Proof of Concept pro budoucí projekty, u nichž se uvažuje o mnohem větším rozsahu a také nárocích. V tomto smyslu bylo také brána v potaz obecná použitelnost frameworku na projekty, které jsou uvažovány do budoucna.

Pro tato jednotlivá kritéria tedy přichází v úvahu pouze jeden framework a tím je **Next.js**, tento framework má nespočet výhod oproti Gatsby, mezi nejdůležitější bych si dovolil uvést:

- Server-side rendering, Static site generation a možnost jejich kombinování
- Hot code reloading
- Automatické routování

- Podporu API cest (API routes)
- Možnost použití vlastního serveru (např. Express nebo Fastify)
- Generování statických stránek (tuto funkcionalitu přináší také Gatsby)
- Možnost libovolného přístupu ke stahování dat (data fetching)
- Dokumentace, example projekty a uživatelská báze
- Tým společností a vývojářů, kteří za frameworkem stojí

## 6.3 Vývoj webové aplikace ve frameworku Next.js

V následujících kapitolách bude popsán vývoj aplikace ve frameworku Next.js. Založení projektu probíhá pomocí npx nebo yarn create příkazu (avšak není to nutností, závislosti i strukturu je možné samozřejmě vytvořit „ručně“). K tomu, abychom mohli začít, potřebujeme nejprve nainstalovat běhové prostředí, tedy Node.js se kterým bude automaticky nainstalována i CLI aplikace npm. Příprava běhového prostředí nebude součástí této práce.

### 6.3.1 Založení nového projektu

Samotné založení projektu je poměrně jednoduchým procesem díky existenci CLI aplikace pro tvorbu Next.js aplikací. Mimo jiné zde odpadá i nutnost konfigurace, o vše se stará samotný framework.

```
> yarn create next-app
```

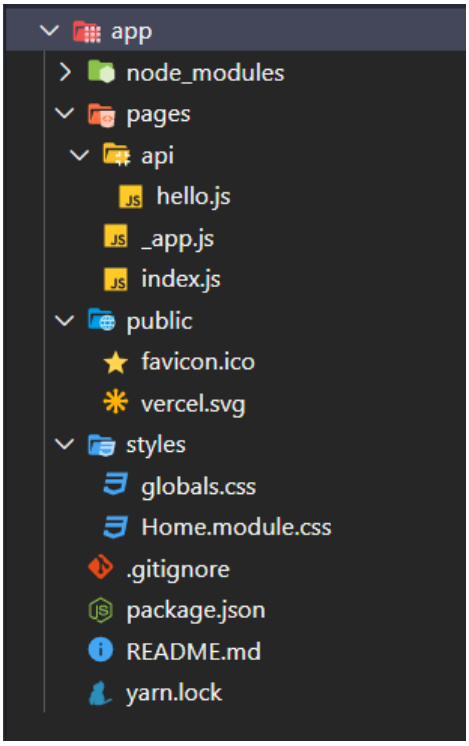
Následně budete vyzváni k zadání názvu aplikace. Poté nástroj automaticky stáhne všechny dependence z npm repozitáře a založí základní strukturu aplikace. Součástí hlášky o úspěšné instalaci je také výpis příkazů, které budete potřebovat pro vývoj aplikace.

```
Success! Created app at /home/dan/thesis/app
Inside that directory, you can run several commands:

  yarn dev
    Starts the development server.

  yarn build
    Builds the app for production.

  yarn start
    Runs the built app in production mode.

We suggest that you begin by typing:

  cd app
  yarn dev
```

*Ukázka kódu 6 - výstup z CLI po vytvoření projektu (autor)*

Výsledná stromová struktura aplikace vypadá následovně:

- **pages**: zde najdeme všechny naše stránky, které jsou zároveň routy (cesty)
- **pages/api:** složka api vnořená do složky pages slouží pro naše API routy (cesty), každý .js soubor v této složce reprezentuje náš API endpoint
- **public:** složka public slouží pro statické soubory, především tedy obrázky, ikonky, favicon.ico, robots.txt, ale také například lokalizační soubory atd.
- **styles:** tato složka obsahuje naše CSS/SCSS/SASS soubory, použití této složky není frameworkem vyloženě vynucováno, ale v případě použití např. Sass preprocesoru je třeba přizpůsobit konfiguraci Babelu a přidat složku, kterou má transpilovat




*Ukázka kódu 7 - automaticky vygenerovaná stromová struktura aplikace (autor)*

V této fázi je již možné spustit vývojový server Next.js příkazem

```
> yarn dev
```

jehož výstupem je informace o úspěšném spouštění serveru na adrese localhost a portu 3000.

```
➜ yarn dev
yarn run v1.22.5
$ next dev
ready - started server on http://localhost:3000
```

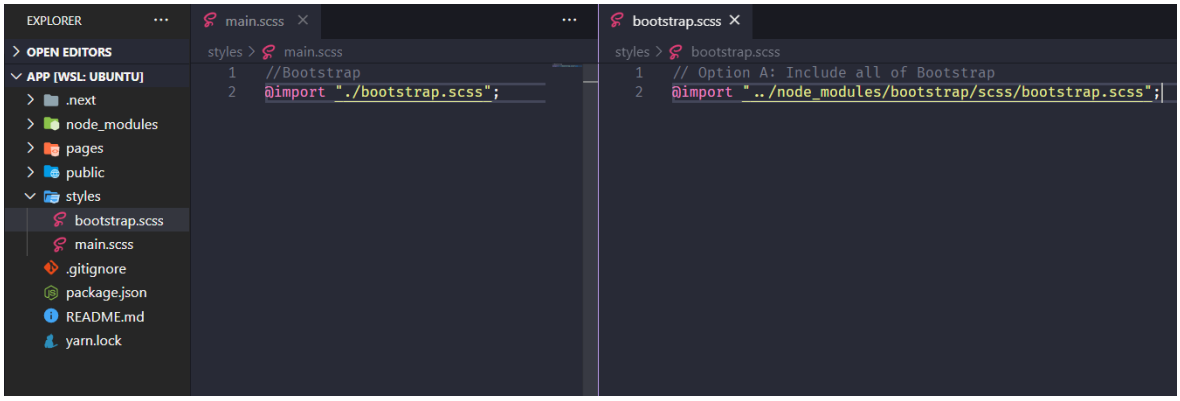
*Ukázka kódu 8 - výstup z konzole po spuštění vývojového prostředí (autor)*

### 6.3.2 Konfigurace Sass a Bootstrap

Projekt využívá front-endový toolkit Bootstrap 4. Tento toolkit používá CSS preprocesor Sass, pro který má Next.js podporu bez nutnosti další konfigurace, stačí pouze nainstalovat z npm repozitáře, stejným způsobem instalujeme také Bootstrap.

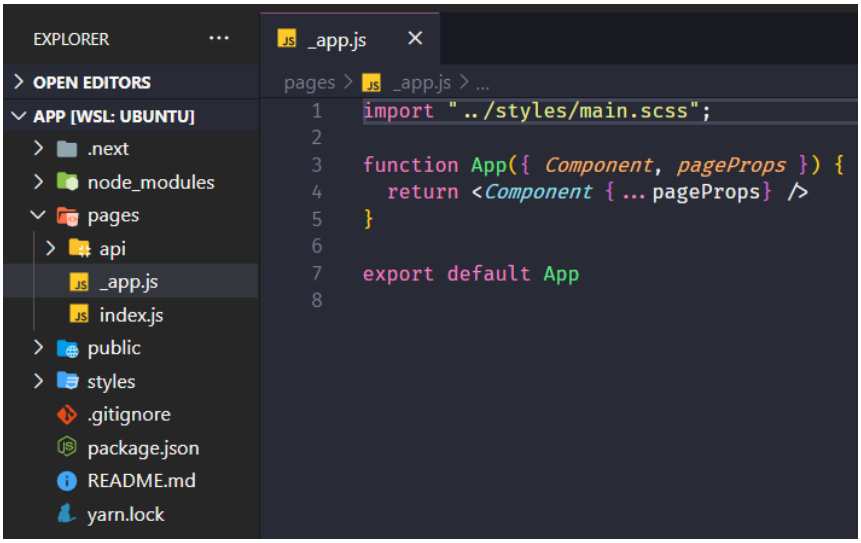
Sass umožňuje importování částí CSS kódu, je tedy dobré mít jeden SCSS soubor, který bude importovat všechny naše CSS komponenty. Tento soubor pojmenujeme

**main.scss** a budeme v něm importovat všechny ostatní Sass komponenty. Komponenty, které Bootstrap obsahuje je možné importovat jednotlivě, nebo jako celek bez ohledu na to, zda jsou v projektu používány. Aby byl náš kód přehlednější, vytvořil jsem další soubor **bootstrap.scss**, ve kterém naimportuji všechny komponenty toolkitu.




*Ukázka kódu 9 - import stylů SCSS (autor)*

Všechny globální styly je u Next.js nutné importovat v komponentě App, která se nachází v souboru **_app.js** (./pages/_app.js), tato komponenta inicializuje všechny stránky.




*Ukázka kódu 10 - import SCSS stylů v _app.js (autor)*

### 6.3.3 Přihlášení uživatele

V aplikaci mohou uživatelé přidávat komentáře k jednotlivým projektům či úkolům, nejen z tohoto důvodu je nutné je autentizovat. Jak již bylo řečeno, identita uživatele je

uložena v korporátním LDAP serveru, odpadá proto nutnost ukládat hesla uživatelů v naší databázi. Ke správě procesu přihlášení a uložení přihlášeného uživatele v JWT použijeme knihovnu **next-auth**. Tato knihovna umožňuje několik způsobů autentizace. Pro naše účely však použijeme bezestavovou autentizaci pomocí vlastního back-endu (LDAP).

O veškerou logiku se stará knihovna. Jediné, co musíme udělat je ověřit zadané přihlašovací údaje, které jsou kombinací uživatelského jména a heslo oproti našemu LDAP serveru. Pokud přihlašovací údaje odpovídají, vrátíme knihovně data o uživateli, pokud ne vrátíme hodnotu false.

K ověření uživatele oproti LDAP použijeme knihovnu s názvem **activedirectory2**.

```
// ...
const ad = new ActiveDirectory(config);

const authenticateUser = (username, password) => {
  const promise = new Promise((resolve) => {
    const user = username;
    const pass = password;
    ad.authenticate(user, pass, function(err, auth) {
      if (err) {
        console.log(JSON.stringify(err))
        resolve(false)
      }

      if (auth) {
        ad.findUser(username, function(err, user) {
          if (err) {
            console.log('ERROR: ' +JSON.stringify(err));
            return;
          }

          if (!user) console.log('User: ' + sAMAccountName + ' not found.');
          if (user) {
            const name = user.givenName;
            const surname = user.sn;
            const email = user.mail;
            resolve({"login_status": true, name, surname, email});
          }
        });
      } else {
        resolve(false);
      }
      });
  })
  return promise;
}
// ...
```

*Ukázka kódu 11 - ověření uživatele oproti LDAP pomocí activedirectory2 (autor)*

Ukázka kódu výše zobrazuje kromě samotné autentizace i způsob, kterým jsou z LDAP získána dodatečná data o uživateli, která jsou posléze předána knihovně next-auth.

```
// ...
const options = {
  site: process.env.NEXT_PUBLIC_SITE,
  providers: [
    Providers.Credentials({
      // The name to display on the sign in form (e.g. 'Sign in with ...')
      name: 'Credentials',
      // The credentials is used to generate a suitable form on the sign in page.
      // You can specify whatever fields you are expecting to be submitted.
      // e.g. domain, username, password, 2FA token, etc.
      credentials: {
        username: { label: "Username", type: "text", placeholder: "" },
        password: { label: "Password", type: "password" }
      },
      authorize: async (credentials) => {
        const username = credentials.username;
        const password = credentials.password;
        // Add logic here to look up the user from the credentials supplied
        let user = await authenticateUser(username, password);
        // Any object returned will be saved in `user` property of the JWT
        return Promise.resolve(user)

      }
    })
  ],
// ...
```

*Ukázka kódu 12 - konfigurace a předání objektu uživatele knihovně next-auth (autor)*

Po úspěšném přihlášení je možné získat údaje o přihlášeném uživateli pomocí hooku **useSession()**, pro použití tohoto způsobu přístupu k datům je třeba použít komponentu **Provider**, která je dostupná z balíčku next-auth. Touto komponentou je nutné obalit komponenty obsažené v _app.js. Na pozadí používá komponenta Provider React Context, což je metoda pro předávání dat bez nutnosti používání props na jednotlivé komponenty. Hook useSession() není dostupný na straně serveru. Na serveru je možné používat metodu **getSession()**. Na pozadí tato metoda provolává API endpoint /api/auth/session, který vrací právě údaje o přihlášeném uživateli.

### 6.3.4 API cesty

Jak již bylo řečeno Next.js podporuje také API cesty, a to bez nutnosti konfigurace. Tato skutečnost nám umožňuje vytvořit API pro získávání dat z naší databáze v rámci jedné aplikace.

API cesty fungují stejným způsobem jako funguje routování v případě stránek. Tyto endpointy vytváříme ve složce api, která je vnořena ve složce pages.

U endpointu je nutné vytvořit *handler* funkci, která obdrží parametry **req** a **res**, kdy req je instancí http.IncommingMessage a res instancí http.ServerResponse. Oba tyto parametry Next.js obohacuje o dodatečné funkce a middleware.

Vzhledem ke skutečnosti, že GitHub pro nás není dostatečným zdrojem dat, vytvoříme prostor pro ukládání těchto dat ve vlastní NoSQL databázi MongoDB, jejíž data budeme poskytovat vlastním REST API.

Pro modelování databázových schémat a připojování do databáze použijeme ODM knihovnu mongoose. Tato knihovna za nás bude mimo jiné spravovat relace, řešit validaci schématu a také samotné připojení do DB.

```
export default async function handler(req, res) {
  const {
    body: { project },
    method
  } = req

  await dbConnect()

  switch (method) {
    case 'GET': {
      Project.find({})
      .populate([
        {
          path: "assignees",
          populate: {
            path: "user",
            select: "username name surname"
          }
        }
      ])
      .exec((error, project) => {
        if (error) res.status(500).json({ error: error });
        if (!error) res.status(200).json(project);
      })
      break;
    }
    case "POST": {
      Project.create(project, (error, project) => {
        // ...
      })
      break;
    }
    default: {
      res.status(405).json({ message: "Unsupported method" })
    }
  }
}
```

*Ukázka kódu 13 - API endpoind pro dodatečná data k projektům (autor)*

### 6.3.5 Stahování dat (Data Fetching)

Podstatná část dat pro naši aplikaci je uložena na GitHub. Tato data mimo jiné určují strukturu a hierarchii stránek. V GitHub je pro nás hlavní uzlem organizace, pod tímto uzlem jsou zakládány projekty a projekty obsahují množinu úkolů Každý úkol může nabývat několika stavů, které jsou určené kanban tabulí obsahující 3 sloupce (stavy), To do, In progress a Done.

Pro získání dat používáme GraphQL API. Podle zmíněné struktury jsme schopni zapsat GraphQL dotaz, který bude vypadat následovně:

```
{
  organization(login: "mbfs-3nt") {
    projects(first: 100, states: OPEN, orderBy: {field: CREATED_AT, direction: DESC}) {
      projects: nodes {
        id
        body
        databaseId
        number
        name
        state
        columns(first: 3) {
          columns: nodes {
            id
            name
            cards {
              totalCount
            }
          }
        }
      }
    }
  }
}
```

*Ukázka kódu 14 - zápis GraphQL query (autor)*

Pro dotazování na REST API budeme používat knihovnu **Axios**, která zjednodušuje práci s dotazy. Její výhodou je, že mimo jiné vrací Promise a automaticky transformuje data dotazů i odpovědí.

Pro dotazování na GraphQL API je pro nás výhodnější (určitě to však není nutností) použít jinou knihovnu než Axios. V tomto projektu použijeme knihovnu **graphql-request**. Ta nám umožňuje snazší zápis a předávání proměnných.

Další velmi důležitou funkcionalitou je cachování a revalidace dat. Za tímto účelem budeme používat knihovnu zvanou SWR. Tato knihovna je vyvíjena samotnými autory frameworku Next.js. Použitím této knihovny jsme poměrně jednoduše schopni aplikovat cachování na naše data, což je vnímáno jako jedna z nejtěžších disciplín webového vývoje.

SWR podporuje jakýkoliv nástroj pro stahování dat, proto si vytvoříme znovupoužitelný React Hook pro stahování a cachování dat.

```
import { request } from 'graphql-request';
import useSWR from "swr";

export default function useQuery(query, variables = {}, { initialData, ...config } = {}) {
  const defaultConfig = { ...config, revalidateOnFocus: false, refreshInterval: 180000 };
  const endpoint = process.env.NEXT_PUBLIC_SITE+'/api/proxy-graphql';
  const queryVal = Array.isArray(query) ? query[0] : query;
  const { data, error, mutate } = useSWR(() =>
    query,
    () => request(endpoint, queryVal, variables),
    {
      ...defaultConfig,
      initialData: initialData && {
        status: 200,
        statusText: 'InitialData',
        headers: {},
        data: initialData
      }
    }
  )

  return {
    data,
    loading: !error && !data,
    error,
    mutate
  }
}
```

*Ukázka kódu 15 - React Hook pro stahování a cachování dat z GraphQL API (autor)*

Data je možné také stahovat na straně serveru, tak aby byla dostupná v době dotazu na server. K tomuto účelu slouží metoda **getServerSideProps()** . Je důležité zmínit, že tato metoda je dostupná pouze ve stránkách, tedy komponentách typu *page*. V této metodě je možné importovat další npm balíčky i vlastní metody. Jsou zde také dostupné instance **req** a **res**. Příkladem použití v našem projektu může být například zabezpečená stránka dostupná pouze po přihlášení. Toto ověření může například probíhat skrze šifrované cookies za použití knihovny next-iron-session jak je znázorněno v ukázce níže. Je zde také vidět způsob, kterým je možné aplikovat lokalizace. Následně jsou data získaná touto metodou vrácena *page* komponentně jako *props*.

```
// ...
export const getServerSideProps = withIronSession(
  async ({ req, res }) => {
    const user = req.session.get("__user");

    if (!user) {
      res.statusCode = 302;
      res.setHeader('Location', `/login`)
      return { props: {} };
    }

    return {
      props: {
        user,
        namespacesRequired: ["common"]
      },
    };
  },
  {
    cookieName: "__user",
    cookieOptions: {
      secure: process.env.NODE_ENV === "production" ? true : false
    },
    password: process.env.ACCESS_SESSION_SECRET
  }
);

export default withTranslation("common")(Dashboard);
```

*Ukázka kódu 16 - použití metody getServerSideProps() (autor)*

V případě, že uživatel není přihlášen (neexistuje cookies, nebo není validní), je přesměrován na přihlašovací stránku, v opačné případě jsou komponentě vrácena data o uživateli formou props.

Pole působnosti je tedy opravdu rozsáhlé, ale jak již bylo zmíněno, my v tomto projektu využíváme především vykreslování na straně klienta.

### 6.3.6 Routování (tvorba stránek)

Jak již bylo řečeno v teoretické části práce, routování v Next.js je velmi intuitivní. Routování je založené na souborovém systému, kdy strom složek a souborů reprezentuje jednotlivé stránky. V hierarchii stránek je v případě naší aplikace výčet projektů. Proto je tento přehled v kořenu struktury a je tedy dostupný na relativní adrese „/“ (http://localhost:3000/). Tato adresa je ve složce pages reprezentována souborem index.js.

Každá stránka musí exportovat funkci, jejíž návratovou hodnotou je JSX.

```
const PROJECTS = gql`
  {
    organization(login: "mbfs-3nt") {
      projects(first: 100, states: OPEN, orderBy: {field: CREATED_AT, direction: DESC}) {
        ...
      }
    }
  }
`;

export default function Projects() {

  const {
    loading,
    error,
    data: { organization: { projects: { projects } = {} } = {} } = {} } = useQuery(PROJECTS, {});

  return (
    <div>
      {error && <p>Something went wrong ... </p>}
      {loading && <p>Loading ... </p>}
      {projects && projects.map((project) => {
        return <Project key={project.databaseId} projectData={project} />;
      })}
    </div>
  )
}
```

*Ukázka kódu 17 - stažení dat pomocí hooku useQuery() a výpis projektů (autor)*

Každý jednotlivý projekt odkazuje na stránku s detailem projektu. Pro odkazování v rámci aplikace slouží komponenta Link, která je součástí frameworku. Tato komponenta umožňuje odkazování na dynamické cesty. Ve výchozím nastavení tato komponenta také na pozadí prefetchuje odkazované stránky.

Pro zobrazení detailu projektu vytvoříme ve složce pages novou složku project. V této složce poté vytvoříme soubor s názvem [projectNumber].js. Jak bylo zmíněno v teoretické části, název uvnitř hranatých závorek reprezentuje proměnnou, jejíž hodnota je následně předána komponentě pomocí props. Tímto způsobem tedy definujeme naše dynamické cesty. Taková cesta pak vypadá například takto *„/project/1“*.

Po navštívení této dynamické cesty můžeme podle této proměnné vytvořit dotaz na API s konkrétním identifikátorem projektu.

### 6.3.7 Vlastní server

Použití vlastního serveru není v dokumentaci obecně doporučováno, avšak tato možnost zde existuje pro případy, kdy výchozí server není dostačující. Při použitím vlastního serveru není možné aplikaci nasadit na Vercel, automatickou optimalizaci statických stránek, nebo serverless funkce. Ani jedna z těchto skutečností pro nás však nehraje v případě tohoto projektu roli. Proto jsme se rozhodli pro funkcionalitu pravidelného spouštění funkcí pro rozesílání emailových upozornění implementovat právě ve vlastním server.js souboru. Použita byla knihovna **node-cron**, která umožňuje použití syntaxe GNU crontab.

```
// ...
const port = parseInt(process.env.PORT, 10) || 3000
const dev = process.env.NODE_ENV !== 'production'
const app = next({ dev })
const handle = app.getRequestHandler()

if (!dev) {
  if (process.env.NODE_APP_INSTANCE === '0') {
    cron.schedule("0 3 * * *", function() {
      console.log("---------------------");
      console.log("Running send deadline mail job");
      deadlineMail.createEmail()
    });
  }
}
app.prepare().then(() => {
  const server = express()

  // security headers middleware
  server.use(helmet());

  server.all('*', (req, res) => {
    return handle(req, res)
  })

  server.listen(port, (err) => {
    if (err) throw err
    console.log(`> Ready on http://localhost:${port}`)
  })
})
```

*Ukázka kódu 18 - vlastní server (autor)*

Pro spravování procesu jsme použili aplikaci PM2, která umožňuje jednoduché monitorování procesu, konzumaci pamětí a CPU.

# 7 Vyhodnocení

Všechny cíle vytyčené v úvodu práce se podařilo zdárně splnit. V první části práce byly zanalyzovány frameworky pro vývoj moderních webových aplikací, a to jak pro tvorbu jednostránkových, tak i univerzálních aplikací. Tyto jednotlivé frameworky jsou v práci popsány, a především vyzdvihnuty jejich rozdílnosti. Samotný výběr zmíněných frameworků proběhl především na základě jejich popularity, tedy četnosti stažení z npm. Jako další parametr nám posloužil průzkum mezi samotnými vývojáři.

Jak bylo v práci zmíněno, v tuto chvíli neexistuje velké množství univerzálních frameworků, které by bylo možné mezi sebou přímo srovnávat. Pro aplikaci vyvíjenou v praktické části byly uvažovány celkem tři frameworky, kdy dva byly založeny na React a jeden na Vue. Především kvůli autorově předchozí zkušenosti s React byl z úvah vyřazen Nuxt.js, který je založen na Vue. Hlavním faktorem pro výběr Next.js byla jeho flexibilnost co se týká stahování dat, ale také to, že Gatsby není vhodný pro aplikace, jejichž data jsou často aktualizována.

Aplikace, která byla výsledkem této práce rozšířila autorovi znalosti o další framework a také o zkušenosti s vývojem webového back-endu, a především zdárně slouží svému účelu ve společnosti, kde je autor zaměstnán.

# 8 Závěr a doporučení

V práci jsem shrnul, jaké jsou aktuálně nejpopulárnější a nejpoužívanější frameworky pro tvorbu uživatelských rozhraní. Postupem času k těmto frameworkům vznikly nástroje a technologie pro podporu a usnadnění jejich vývoje. Tyto nástroje rozšiřují schopnosti zmíněných frameworků a zároveň se snaží potírat jejich nevýhody. Univerzální frameworky, které byly v této práci zmíněny ulehčují vývojářům jejich práci, kdy z jejich strany kompletně odpadá nutnost konfigurace podpůrných nástrojů a optimalizují napsaný kód, čímž zvyšují výkon samotné aplikace. Mimo jiné přinášejí i funkcionality známé z klasických, vícestránkových webových aplikací jako je například vykreslování na straně serveru, nebo generování statických stránek.

Všechny vývojářské týmy, ale také jednotlivci řeší otázku, které technologie pro svůj projekt zvolit. Je to možná ta nejtěžší volba, pokud jde o vývoj software. Kritérií, na které je třeba brát zřetel je mnoho, ať už jde o zkušenosti, nebo o požadavky projektu. Nové a neznámé technologie je nejlepší zkoušet na menších projektech, na kterých není závislá hlavní činnost společnosti a u kterých nebude případné přepsání do jiného frameworku či dokonce jazyka příliš nákladné.

Cílem této práce bylo najít vhodný framework právě pro takovýto menší projekt, který slouží také jako proof of concept pro naše budoucí projekty. Proto při jeho výběru byl brán velký zřetel na to, aby byl dostatečně univerzální a bylo možné jej použít ať už na velké či malé projekty. Naše projekty mají obvykle dlouhou životnost, a proto je důležité vybrat framework, který budeme rádi používat ještě dlouhá léta.

Next.js je právě takovým frameworkem. Tým a společnosti, které za ním stojí jsou velkým příslibem pro budoucí vývoj. To také dokazují již existující projekty postavené na tomto frameworku. Stále častěji si ho pro své projekty vybírají nadnárodní korporace.

# 9 Seznam zdrojů


1. HANDLEY, Lucy. Nearly three quarters of the world will use just their smartphones to access the internet by 2025. *CNBC* [online]. 24. 1. 2019 [cit. 2020-07-04]. Dostupné z: https://www.cnbc.com/2019/01/24/smartphones-72percent-of-people-will-use-only-mobile-for-internet-by-2025.html

2. Desktop vs Mobile vs Tablet Market Share Worldwide. *Statcounter* [online]. [cit. 2020-11-11]. Dostupné z: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/

3. The State of the OCTOVERSE. GitHub [online]. [cit. 2020-07-06]. Dostupné z: https://octoverse.github.com/

4. GitHub [online]. [cit. 2020-07-11]. Dostupné z: https://octoverse.github.com

5. Stack Overflow Developer Survey 2020. StackOverflow [online]. [cit. 2020-07-06]. Dostupné z: https://insights.stackoverflow.com/survey/2020

6. WAGNER, Gerd a Mircea DIACONESCU. *Web Applications with Javascript or Java*. Berlín: Walter de Gruyter, 2017. ISBN 978-3-11-049993-3.

7. KONSHIN, Kirill. Next.js Quick Start Guide: Server-side rendering done right [online]. Birmingham: Packt Publishing, 2018 [cit. 2020-07-06]. ISBN 978-1-78899-366-1. Dostupné z: https://github.com/PacktPublishing/Next.js-Quick-Start-Guide

8. MESBAH A., van DEURSEN A., "Migrating Multi-page Web Applications to Single-page AJAX Interfaces," 11th European Conference on Software Maintenance and Reengineering (CSMR'07), Amsterdam, 2007, pp. 181-190, doi: 10.1109/CSMR.2007.33. Dostupné z: https://ieeexplore.ieee.org/document/4145036

9. Single-Page Apps vs Multiple-Page Web Apps: What to Choose for Web Development. *Yalantis* [online]. [cit. 2020-11-11]. Dostupné z: https://yalantis.com/uploads/ckeditor/pictures/4791/spa-vs-mpa.png

10. Deprecating our AJAX crawling scheme. In: *Google: Webmaster Central Blog* [online]. 14. 10. 2015 [cit. 2020-07-15]. Dostupné z: https://webmasters.googleblog.com/2015/10/deprecating-our-ajax-crawling-scheme.html

11. npm [online]. [cit. 2020-07-06]. Dostupné z: https://docs.npmjs.com/

12. Yarn [online]. [cit. 2020-07-06]. Dostupné z: https://classic.yarnpkg.com/lang/en/

13. CO JE GRAPHQL A V ČEM JE LEPŠÍ NEŽ REST [online]. 2020 [cit. 2020-10-31]. Dostupné z: https://www.rascasone.com/cs/blog/co-je-graphql-versus-rest

14. GraphQL [online]. [cit. 2020-10-31]. Dostupné z: https://graphql.org/

15. *TECH 101: WHAT IS A JAVASCRIPT FRAMEWORK? HERE'S EVERYTHING YOU NEED TO KNOW* [online]. Skillcrush [cit. 2020-07-11]. Dostupné z: https://skillcrush.com/blog/what-is-a-javascript-framework/

16. GIMENO, Alberto. The deepest reason why modern JavaScript frameworks exist. In: *Medium: Daily JS* [online]. 24. 1. 2019 [cit. 2020-11-11]. Dostupné z: https://medium.com/dailyjs/the-deepest-reason-why-modern-javascript-frameworks-exist-933b86ebc445

17. COPES, Flavio. The React Handbook. *FreeCodeCamp* [online]. 8. 1. 2019 [cit. 2020-11-10]. Dostupné z: https://www.freecodecamp.org/news/the-react-handbook-b71c27b0a795

18. React: The Virtual DOM: Fighting Wasteful DOM Manipulation. *Codecademy* [online]. [cit. 2020-11-10]. Dostupné z: https://www.codecademy.com/articles/react-virtual-dom

19. React virtual DOM. *Pngitem* [online]. [cit. 2020-11-11]. Dostupné z: https://www.pngitem.com/middle/hmRRRim_react-virtual-dom-png-download-virtual-dom-in/

20. Vue.js [online]. [cit. 2020-07-11]. Dostupné z: https://vuejs.org/

21. The Best JS Frameworks for Front End. Rubygarage.org [online]. 2020 [cit. 2020-07-27]. Dostupné z: https://rubygarage.org/blog/best-javascript-frameworks-for-front-end

22. Renaming isomorphic JavaScript? [online]. oreilly.com [cit. 2020-07-16]. Dostupné z: https://www.oreilly.com/content/renaming-isomorphic-javascript/

23. MARDAN, Azat. Why Everyone is Talking About Isomorphic / Universal JavaScript and Why it Matters. *Medium* [online]. 21. 5. 2016 [cit. 2020-11-10]. Dostupné z: https://medium.com/capital-one-tech/why-everyone-is-talking-about-isomorphic-universal-javascript-and-why-it-matters-38c07c87905

24. 7 Frontend Architecture Lessons From Nuxt.js. *ZenDev* [online]. [cit. 2020-11-11]. Dostupné z: https://zendev.com/2018/09/17/frontend-architecture-lessons-from-nuxt-js.html

25. The Benefits of Server Side Rendering Over Client Side Rendering. *Medium: Walmar Global Tech* [online]. [cit. 2020-11-11]. Dostupné z: https://medium.com/walmartglobaltech/the-benefits-of-server-side-rendering-over-client-side-rendering-5d07ff2cefe8

26. STARYGA, Bartosz. SPA SEO: Mission Impossible? In: *Magnolia* [online]. 23. 5. 2020 [cit. 2020-07-15]. Dostupné z: https://www.magnolia-cms.com/blog/spa-seo-mission-impossible.html

27. Gatsby vs next vs nuxt vs quasar-framework vs fusionjs. *npm trends* [online]. [cit. 2020-11-11]. Dostupné z: https://www.npmtrends.com/gatsby-vs-next-vs-nuxt-vs-quasar-framework-vs-fusionjs

28. Gatsby.js [online]. [cit. 2020-07-11]. Dostupné z: https://gatsbyjs.com/

29. Next.js [online]. [cit. 2020-07-11]. Dostupné z: https://nextjs.org/

30. Node.js [online]. [cit. 2020-07-11]. Dostupné z: https://nodejs.org/

31. Webpack [online]. [cit. 2020-07-07]. Dostupné z: https://webpack.js.org/

32. *React vs. Angular vs. Vue.js: Which one should I learn in 2019* [online]. Medium.com [cit. 2020-07-15]. Dostupné z: https://medium.com/@rahul77349/react-vs-angular-vs-vue-js-which-one-should-i-learn-in-2019-23ec05a49f78

33. *LEARNING JAVASCRIPT FRAMEWORKS WILL BOOST YOUR JOB PROSPECTS—BUT WHICH ONE SHOULD YOU LEARN FIRST?* [online]. skillcrush.com [cit. 2020-07-15]. Dostupné z: https://skillcrush.com/blog/which-javascript-framework-should-you-learn-first/

34. Isomorphic (Universal) JavaScript. Medium.com [online]. [cit. 2020-07-19]. Dostupné z: https://medium.com/commencis/isomorphic-universal-javascript-496dc8c4341a

# 10 Seznam obrázků

# 11 Seznam grafů

# 12 Seznam ukázek kódu

# 13 Přílohy

*Příloha 1 - Zdrojové kódy vygenerované struktury aplikace s částmi kódu zmíněnými v této práci*


# Zadání bakalářské práce

| | |
|---|---|
| **Autor:** | **Dan Forejtek** |
| Studium: | I1500285 |
| Studijní program: | B6209 Systémové inženýrství a informatika |
| Studijní obor: | Informační management |
| **Název bakalářské práce:** | **Jednostránkové aplikace** |
| Název bakalářské práce AJ: | Single-page applications |

**Cíl, metody, literatura, předpoklady:**

Cílem práce je představit technologii single page aplikací a demonstrovat postup vývoje za použití JS frameworku vhodného vzhledem k určení aplikace. Tento cíl v sobě zahrnuje i úkol vyhodnotit, jaký framework je vhodný pro danout aplikaci.

Osnova práce:

1. Úvod

2. Technologie používané pro vývoj webových aplikací

3. Principy a vlastnosti SPA

4. Analýza vyvíjené aplikace

5. Výběr a představení vhodných frameworků

6. Ukázková implementace

7. Zhodnocení vhodnosti frameworků

8. Závěr a doporučení

[1] FLANAGAN, D. JavaScript: The Definitive Guide. 6th ed. Sebastopol, CA: O?Reilly, 2011, xvi, 1078 p. ISBN 05-968-0552-7.

[2] SIMPSON, Kyle, et al. You Don?t Know JS: ES6 {\& Beyond. "O?Reilly Media, Inc.", 2015

[3] KLAUZINSKI, Philip a John MOORE. Mastering JavaScript Single Page Application Development. Birmingham: Packt Publishing, 2016. ISBN 978-1785881640.

[4] Getting Started | Next.js. Next.js by Vercel - The React Framework [online]. Dostupné z: https://nextjs.org/docs

[5] MDN web docs: JavaScript. MDN web docs [online]. online: Mozilla, 2020. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[6] Getting Started ? React. React ? A JavaScript library for building user interfaces [online]. Copyright ? 2020 Facebook Inc. [cit. 17.06.2020]. Dostupné z: https://reactjs.org/docs/getting-started.html

| | |
|---|---|
| Garantující pracoviště: | Katedra informačních technologií,<br>Fakulta informatiky a managementu |
| Vedoucí práce: | Mgr. Daniela Ponce, Ph.D. |
| Datum zadání závěrečné práce: | 15.10.2018 |