VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

VYUŽITÍ SLACK API PRO HEADLESSCMS

USE OF SLACK API FOR HEADLESSCMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JOZEF HRUŠKA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Hruška Jozef**
Program: Informační technologie
Název: **Využití Slack API pro HeadlessCMS**
Use of Slack API for HeadlessCMS
Kategorie: Web

Zadání:

1. Prostudujte a seznamte se s principem fungování existujících nástrojů HeadlessCMS a dokumentací k Slack API.
2. Na základě získaných znalostí navrhněte vlastní řešení, které by využívalo Slack k propojení, transformaci a zobrazení obsahu v HeadlessCMS místo použití konvenční DB, která je využívána současnými řešeními. Návrh konzultujte s vedoucím.
3. Navržené řešení implementujte ve zvoleném prostředí.
4. Proveďte testování vytvořené aplikace a jejího propojení s aplikací Slack.
5. Práci zhodnoťte, srovnejte se stávajícími HeadlessCMS nástroji a navrhněte další možná budoucí vylepšení.

Literatura:

- SIMPSON, K.: *You don't know JS: ES6 and beyond*. Beijing: O'Reilly, 2016. ISBN 978-1-491-90424-4.
- STURGEON, P.: *Build APIs you won't hate*. Philip J. Sturgeon, 2015. ISBN 978-0-692-23269-9.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1-2.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Bartík Vladimír, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 14. května 2020
Datum schválení: 16. října 2019

Abstrakt

Práca si dáva za cieľ vytvoriť redakčný systém s otvoreným aplikačným rozhraním a možnosťou správy obsahu v prostredí aplikácie Slack. Inštalácia a následné používanie systému nevyžaduje žiadnu konfiguráciu zo strany uživateľa. Otvorené (verejné) ako aj uzatvorené (skryté) aplikačné rozhranie je vybudované podľa špecifikácie GraphQL. Uzatvorené rozhranie slúži výhradne k čítaniu dát, tzn. že nie je možné dáta akokoľvek vkladať či modifikovať použitím tohto rozhrania. Výstupom práce je plne funkčný prototyp systému, ktorého súčasti boli implementované pomocou nástrojov React a Node.js s dôrazom na prívetivosť uživateľského rozhrania.

Abstract

The aim of this thesis is to create a content management system with open application interface and capability of content management inside Slack application. Installation and use of system does not require any configuration by user. Open (public) and closed (hidden) application interface is built following GraphQL specification. The closed interface is read-only which means it is not possible to insert or modify data through this interface. The output of this work is a fully functional prototype of system and its which were implemented with tools as React and Node.js with focus on user-friendly interface.

Kľúčové slová

RS, Slack, React, Prisma, Apollo, Node.js, Next.js, Redux, Bolt, GraphQL, TypeScript, PostgreSQL

Keywords

CMS, Slack, React, Prisma, Apollo, Node.js, Next.js, Redux, Bolt, GraphQL, TypeScript, PostgreSQL

Citácia

HRUŠKA, Jozef. *Využití Slack API pro HeadlessCMS*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Využití Slack API pro HeadlessCMS

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Vladimíra Bartíka, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Jozef Hruška
7. apríla 2020

Poďakovanie

Rád by som poďakoval Ing. Vladimírovi Bartíkovi, Ph.D. za odborné vedenia a čas venovaný mojej bakalárskej práci. Ďaľej by som rád poďakoval Jánovi Vorčákovi, M.Sc., za inšpiráciu vedúcu k téme tejto práce, za jeho cenné rady a konzultácie.

Obsah

Kapitola 1

Úvod

Tradičné redakčné systémy pre správu obsahu sú obvykle zostavené z dvoch hlavných súčastí – administračného a verejného webového rozhrania. Administračné rozhranie slúži pre vytváranie a úpravu obsahu, webové na jeho nasledné zobrazenie. Webové rozhranie je obvykle jednotné pre všetky platformy a zariadenia na ktorých je využívané, tzn. že jeho používanie je často neoptimálne. Pre tento dôvod sa začali využívať systémy bez webového rozhrania disponujúce klasickým administračným rozhraním a otvoreným aplikačným rozhraním (API[1]), umožnujúcim získavanie obsahu na požadované platformy. Tento obsah je následne možné optimalizovať individuálne podľa potreby. Súčasné riešenia redakčných systémom s otvoreným rozhraním sú často robustné systémy, ktoré však vyžadujú komplexnú konfiguráciu predtým ako ich je možné začať využívať. Niektoré takéto systémy vyžadujú aj vlastnú infraštruktúru pre ich nasadenie. Tieto skutočnosti otvárajú priestor pre redakčný systém, ktorý by pomohol vyriešiť tieto prekážky. Takýto systém je cieľom tejto práce.

Navrhovaný systém poskytuje možnosť správy obsahu bez nutnosti úvodnej konfigurácie a infraštruktúry. Obsah je možné spravovať pomocou užívateľského rozhrania v aplikácii Slack alebo webového rozhrania. Plnú funkcionalitu však užívateľ môže využiť bez nutnosti používania webového rozhrania.

Obsah kapitol

Prvá kapitola, Teoretická časť, sa skladá zo štyroch sekcií venovaným postupne existujúcim riešeniam podobným tejto práci, vývoji serverových aplikácii, klientských aplikácii a špecifikácii GraphQL.

[1] API – Application programming interface

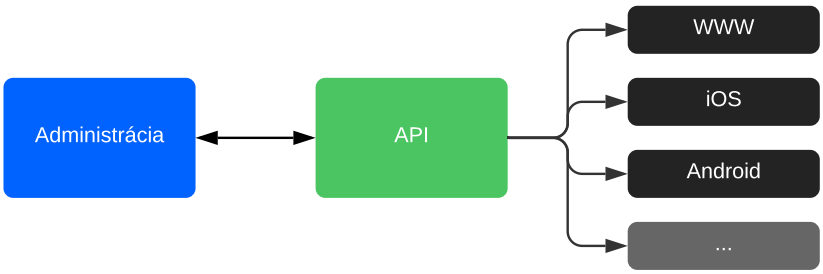
Kapitola 2

Teoretická časť

Správa obsahu a jeho doručenie konzumentom. Táto kapitola popisuje základné princípy, techniky a technológie, ktoré sú nutné pre zostavenie a pochopenie princípu fungovania tejto práce. V prvej sekcií 2.1 sú bližšie priblízené redakčné systémy s otvoreným aplikačným rozhraním, v druhej sekcií 2.3 vývoj serverových aplikácií, tretia sekcia 2.2 popisuje vývoj klientských aplikácii, štvrtá sekcia 2.4 vysveluje špecifikáciu GraphQL. Posledná sekcia 2.5 je venovaná otvorenej API aplikácie Slack, jej možnostiam a obmedzeniam.

2.1 Redakčné systémy s otvoreným aplikačným rozhraním

Redakčné systémy disponujúce iba skrytou administračnou časťou a verejným aplikačným rozhraním sú typicky nazývané *headless*[1] redakčné systémy. Tieto riešenia štandardne disponujú rozhraním REST[2] alebo GraphQL, ktoré implementuje aj táto práca. Headless redakčný systém nerieši zobrazenie samotného obsahu. Jediný spôsob ako obsah získať je využiť niektoré z dostupných rozhraní poskytované konkrétnym riešením. Výhodou oproti tradičným redakčným systémom je možnosť získané dáta optimalizovane zobraziť na rôznych zariadeniach.



Obr. 2.1: Ilustračná schéma generického headless redakčného systému.

Členenie obsahu v takýchto redakčných systémoch je typicky v dvoch vrstvách – kategórie a obsahové typy (komponenty). Kategórie sú zoznamy združujúce jednotlivé komponenty, môžu byť homogénne (všetky prvky zoznamu sú jedného typu) alebo heterogénne (prvky zoznamu sú typicky iných typov). Komponenty sú atomickými prvkami headless redakčných

[1] headless – ang. bez hlavy (bez webového rozhrania)
[2] REST – Representational State Transfer

systémov. Môžu nadobúdať rôznych typov, ktoré určujú ich vnútornú dátovú štruktúru. Typický príklad často používaných typov komponentov sú napríklad `prostý text` alebo `odkaz`. Niektoré headless redakčné systémy umožňujú vytvárať aj vlastné typy komponentov a tak si prospôsobiť dáta vlastným špecifickým potrebám.

2.1.1 Strapi

Najpopulárnejší Headless redakčný systém. Disponuje administračným panelom zostaveným na mieru, REST aj GraphQL rozhraním, systémom uživateľských práv a mnohými inými vlastnosťami. Tento redakčný systém má aj obchod s aplikáciami, ktoré si môžu používatelia pridať a tým rozšíriť funkcionalitu.

> Strapi je flexibilný, open-source[3] Headless CMS[4], ktorý dáva vývojárom slobodu voľby ich obľúbených nástrojov a zároveň dovoľuje editorom jednoducho spravovať a distribuovať ich obsah. (Dokumentácia Strapi.io [5])

V prípade, že by užívateľ Strapi mal nakonfigurovanú kolekciu s názvom „restaurants“, získanie počtu týchto kolekcií by mohlo vyzerať takto:

```
1 GET "http://localhost:1337/restaurants/count" // Odpoved: 1
```

Výpis 2.1: Príklad HTTP požiadavku na REST rozhranie Strapi.

Pre použitie Strapi je nutné systém spustiť na vlastnej infraštruktúre, pripojiť k predom vytvorenej relačnej databáze a celý systém nakonfigurovať.

2.1.2 Netlify CMS

Netlify na rozdiel od väčšiny headless redakčných systémov nevyužíva pre ukladanie svojich dát relačnú databázu. Pre uloženie celého obsahu webovej aplikácie využíva repozitáre vytvorené v prostredí `git`.

> Jadro Netlify CMS tvorí React aplikácia ktorá využíva rozhranie pre prácu s GitHub[5], GitLab[6] alebo Bitbucket[7] API. (Dokumentácia Netlify CMS [4])

Netlify sa využíva väčšinou pre menšie stránky ako sú napríklad dokumentácie alebo produktové stránky, pretože umožňuje udržovať obsah relevantný k danej verzií produktu.

2.2 Vývoj klientských aplikácii

Sekcia popisuje teoretické znalosti nutné pre návrh a implementáciu klientských aplikácií (v prípade tejto práce webových stránok). Webové technológie sa vyvíjajú vysokou rýchlosťou a s nimi aj nároky užívateľov na rýchlosť, použiteľnosť, ale aj vzhľad takýchto aplikácií.

[3] open-source – otvorený kód, zväčša vyvíjaný komunitou
[4] CMS – ang. content management system (redakčný systém)
[5] https://developer.github.com/v3/
[6] https://docs.gitlab.com/ee/api/
[7] https://confluence.atlassian.com/bitbucket/

2.2.1 User Experience (UX)

Prí návrhu uživateľského rozhrania je jedným z najdôležiteľších bodov zaistenie čo najlepšej uživateľskej skúsenosti. Ide o proces, kedy sa dizajnér daného grafického rozhrania snaží vnímať svoj návrh zo strany koncového užívateľa. Vo svete neexistuje jednotná definícia úkonov, ktoré vedú k dokonalému uživateľskému zážitku. Výsledná aplikácia musí byť čo najlepšie použiteľná a zároveň intuitívna pre užívateľov.

Porozumenie potrebám užívateľa

Dizajnér sa snaží najprv porozumieť potrebám a problémom užívateľov, ktorí budú využívať produkt, ktorý navrhuje. Po porozumení sa snaží prísť s riešeniami pre tieto problémy. Svoje riešenia postupne vkladá to návrhu, pričom sa zároveň pokúša o čo najväčšiu originalitu.

Použiteľnosť

Výsledný produkt musí dbať aj na jednoduchosť použitia – snaží sa nevytvárať užívateľovi nové prekážky. Veľmi jednoducho dosiahnuteľná, ale často zanedbávaná vlastnosť je dobrá použiteľnosť pre ľudí so zdravotnými znevýhodneniami. Dizajnér musí zaistiť, aby farebné pozadia jednotlivých prvkov mali dostatočný kontrast od ich obsahu alebo zvýrazniť prvok v prípade, že je užívateľom používaný (túto vlastnosť majú v určitej podobe vstavané aj niektoré webové prehliadače, avšak nie vždy optimálne).

2.2.2 HTML

Hypertext Markup Language je *značkovací jazyk*, pomocou ktorého je možné popísať štruktúru webových stránok. Pre popis jednotlivých častí webovej stránky HTML využíva *elementy (značky)*.

> HTML je nejzákladnejším stavebným kameňom webu. „Hyperlink“, v názve referuje k možnosti využitia odkazov, ktorými je možné prepojiť webové stránky. (MDN [8])

HTML štruktúra jednoduchého dokumentu by mohla vyzerať takto:

```
<html lang="sk">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dokument</title>
  </head>
  <body>
    <p>Text</p>
  </body>
</html>
```

Výpis 2.2: Príklad jednoduchej HTML štruktúry.

Elementy

HTML element je oddelený od zbytku textu v dokumente „`tagmi`“, ktoré pozostávajú z názvu elementu medzi znakmi „`<`“ a „`>`“. Názov elementu vo vnútri tagu je *case insensitive*, tzn. že nezáleží či je písaný veľkými alebo malými písmenami. Napríklad tag

`<title>` môže byť napísaný aj ako `<Title>` alebo `<TITLE>`. Všetky varianty týchto zápisov sú valídne. [8]

Tagy sa klasifikujú na dve skupiny – *párové* a *nepárové*. Párové tagy sú také, ktoré obsah elementu ohraničujú otváracím (`<title>`) a uzatváracím (`</title>`) tagom. Nepárové tagy sú také, ktoré nemajú svoj uzatvárací tag, napríklad obrázok (`<img />`).

Zoznam niektorých najpoužívanejších elementov:

- `head` – Obsahuje strojom čítateľné informácie (metadáta) o dokumente ako napríklad titulok, skripty alebo štýly. [8]
- `body` – Reprezentuje obsah HTML dokumentu, pričom sa v jednom dokumente môže nachádzať maximálne raz. [8]
- `title` – Definuje titulok dokumentu, ktorý je zobrazený vo webovom prehliadači. [8]
- `button` – Reprezentuje klikateľné tlačidlo, použiteľné napríklad pre potvrdzovanie formulárov alebo kdekoľvek inde v HTML dokumente ako štandardné tlačidlo. Tlačidlá sú v štýle jednotnom s platformou na ktorej sú zobrazované, ak nie sú priložené štýly, ktoré by ich upravovali. [8]

2.2.3 CSS

Cascading Style Sheets (CSS) je deklaratívny jazyk, ktorý dokáže kontrolovať ako sa webový stránky zobrazujú vo webových prehliadačoch. Prehliadače aplikujú CSS štýly priamo na elementy nimi upravené a potom ich zobrazia. Deklarácie štýlov obsahujú *vlastnosti* a ich *hodnoty*, ktoré určujú ako má webová stránka vyzerať. [8]

CSS je možné pridať tromi spôsobmi:

- Importovaním externého CSS súboru v hlavičke dokumentu.
- Vložením medzi element `<style>` do hlavičky dokumentu.
- Vložením jednotlivých vlastností a ich hodnôt do tagov jednotlivých elementov cez parameter `style`.

Jednotlivé vlastnosti sa elementom priraďujú použitím *CSS selektorov*. Existujú aj selektory alebo kombinátory, ktoré umožňujú zvoliť rodičovské alebo vedľajšie elementy. [8]

```
1  .button {
2    color: red;
3    text-transform: uppercase;
4  }
```

Výpis 2.3: Príklad zápisu v jazyku CSS.

2.2.4 JavaScript

JavaScript je populárny *interpretovaný* programovací jazyk. Napriek tomu, že je známy predovšetkým ako skriptovací jazyk pre webové aplikácie, dnes je využívaný mnohými prostrediami mimo webových prehliadačov, ako napríklad Node.js. [8]

Štandardom pre JavaScript je *ECMAScript*[8]. Od roku 2012 všetky moderné webové prehliadače podporujú ECMAScript verzie 5.1. Staršie prehliadače podporujú aspoň ECMAScript 3. V roku 2015 bola vydaná verzia ECMAScript 2015 (známa aj ako ECMAScript 6 alebo ES6). Odvtedy je štandard ECMAScript na cykle ročných vydaní. [8]

2.2.5 TypeScript

TypeScript je rozšírenie programovacieho jazyka JavaScript. Jedná sa o silne typovaný, objektovo orientovaný a kompilovaný programovací jazyk [7]. TypeScript je obvykle nutné skompilovať do natívneho JavaScriptu pre zachovanie kompatibility. Využívanie TypeScriptu nie je nutné, avšak vďaka vlastnosti silného typovania umožní vývojárovi predísť chybám ešte pred kompiláciou.

> Vzťah medzi TypeScriptom a JavaScriptom je unikátny medzi modernými programovacími jazykmi. TypeScript existuje ako vrstva nad JavaScriptom; ponúka vlastnosti JavaScriptu a pridáva svoju vlastnú vrstvu navrch. Táto vrstva je nazývaná *typovací systém TypeScript*. (Dokumentácia TypeScript [7])

```
1  /* Definicia typu */
2  type Person = {
3    meno: string;
4    priezvisko: string;
5  }
6
7  /* Priradenie typu k objektu */
8  const Osoba: Person = {
9    meno: "Jan";
10   priezvisko: "Novak";
11 }
```

Výpis 2.4: Príklad zápisu v programovacom jazyku TypeScript.

Využitie programovacieho jazyka TypeScript nie je limitované pre vývoj klientských aplikácii. Rovnako ako pri JavaScripte sa jedná o univerzálny programovací jazyk, ktorý je vďaka nástrojom ako Node.js možné využiť napríklad aj na tvorbu serverových aplikácií.

2.2.6 React

Populárna JavaScriptová knižnica pre *budovanie užívateľských rozhraní*. React je deklaratívny, efektívny a flexibilný. Dovoľuje vytvárať užívateľské rozhrania zložené z malých izolovaných častí kódu, nazývaných *komponenty* (Komponenty). [2]

JSX

Syntaktické rozšírenie JavaScriptu inšpirované značkovacími jazykmi, avšak s možnosťou využívať plné možnosti JavaScriptu. JSX vzniklo, pretože v moderných webových aplikáciach bolo čoraz častejšie spájať vykreslovaciu logiku a logiku užívateľských rozhraní. [2]

[8] https://www.ecma-international.org/

```
const meno = "Jan Novak";
const element = <h1>Ahoj, {meno}</h1>;

ReactDOM.render(
  element,
  document.getElementById("root")
);
```

Výpis 2.5: Príklad využitia JSX v React aplikácií. [2]

Atribúty JSX tagov môžu prijímať textové reťazce (`<div className='block'>`) alebo JavaScriptové výrazy (`<img src={item.image} />`), ktoré sa neskôr vyhodnotia.

Elementy JSX sú kompilované do volaní `React.createElement()`, ktoré vrátia obyčajné JavaScriptové objekty nazvané „React elementy“. [2]

```
const element = {
  type: 'h1',
  props: {
    className: 'pozdrav',
    children: 'Ahoj svet!'
  }
};
```

Výpis 2.6: Príklad jednoduchého React elementu po kompilácií. [2]

Tieto objekty slúžia ako „popis“ pre zobrazenie. React tieto objekty využíva pre zostavenie a urdržovanie aktuálnosti DOM[9]. [2]

Komponenty

Komponenty umožňujú vývojárovi rozdeliť uživateľské rozhranie do samostatných, *znovu použiteľných* častí [2]. Komponenty sa delia na *funkcionálne* a *triedne*.

Najjednoduchší spôsob ako definovať komponentu je obyčajná JavaScriptová funkcia:

```
const Titulok: React.FC = (props) => {
  return <h1>Vitajte na {props.nazov}</h1>;
}
```

Výpis 2.7: Príklad definície funkcionálnej komponenty.

Avšak pre definíciu komponenty možeme použiť aj ES6 triedu:

```
class Titulok extends React.Component {
  render {
    return <h1>Vitajte na {props.nazov}</h1>;
  };
}
```

Výpis 2.8: Príklad definície triednej komponenty.

[9] DOM – document object model

> Konceptuálne sú komponenty ako obyčajné JavaScriptové funkcie. Prijímajú vstupy nazývané *props* a vracajú React elementy popisujúce zobrazenie na obrazovke. (Dokumentácia React [2])

2.2.7 Next.js

Framework[10] umožňujúci render (vykresľovanie) React aplikácie na serveri.

Najväčším problémom moderných JavaScript aplikácií je fakt, že samotné vykresľovanie obsahu na rozdiel od napríklad PHP prebieha na strane klienta. To zapríčiňuje, že aplikácie sú nie len *náročné na výpočetný výkon*, ale aj ich prvotné načítanie *trvá dlhšiu dobu*, čo môže negatívne ovplyvniť SEO.

Next.js rieši tieto problémy vykreslením stránok vopred. HTML je generované pri každej požiadavke na stránku už na serveri s priloženým minimálnym JavaScript kódom. Po obdržaní tejto stránky webovým prehliadačom klienta sa JavaScriptový kód inicializuje a zaistí plnú interaktivitu. Tento proces sa nazýva „hydratácia“. [15]

Ďalšími benefitmi Next.js sú:

- `Statický export` – Stránky, ktoré nevyžadujú žiadne dynamické dáta je možné vygenerovať už pri nasadení aplikácie. [15]
- `Nulová konfigurácia` – Žiadna vstupná konfigurácia nie je potrebná. Všetky funkcionality Next.js sú dostupné ihneď po inštalácií. [15]
- `Optimalizácia` – Automatické optimalizovanie balíčkov, ktoré sú vyžadované, čo umoňuje rýchlejšiu kompiláciu. [15]

2.3 Vývoj serverových aplikácii

`Server` – centrálny počítač z ktorého ostatné počítače získavajú informácie. [1]

Serverová aplikácia je proces spustený na centrálne dostupnom zariadení. Obvykle slúži ako zdroj informácií pre ostatné zariadenia, typicky v počítačovej sieti. Tento proces očakáva požiadavky a odpovedá na ne vopred určenou reakciou. Serverové aplikácie je možné implementovať v rôznych programovacích jazykoch, no medzi najpoužívanejšie patrí PHP, Java, Python alebo JavaScript (popr. rozšírenie TypeScript). Aplikácia má ďalej určený komunikačný protokol, pomocou ktorého prijíma požiadavky a odosiela odpovede. Pri bežných aplikáciach ide najčastejšie protokol HTTP.

2.3.1 HTTP

Protokol HTTP je využívaný ako generický protokol pre prenos dát (správ) medzi klientom a serverom, napríklad HTML dokumentov. Komunikácia je zahájená klientom, typicky webovým prehliadačom.

[10] framework – kompletný set nástrojov

Hypertext Transfer Protocol (HTTP) je protokol na aplikačnej úrovni pre distribuované a spolupracujúce informačné systémy. Protokol HTTP je využívaný iniciatívou World Wide Web od roku 1990. (RFC 2616 [3])

Terminológia

- `Požiadavka (Request)` – HTTP správa odoslaná od klienta, adresovaná pre server.
- `Odpoveď (Response)` – HTTP správa odoslaná od servera, adresovaná pre klienta. Odosiela sa po prijatí a spracovaní HTTP požiadavku od klienta.
- `Metóda (Method)` – Pole v hlavičke HTTP požiadavku. Definuje operáciu, ktorú má server vykonať po prijatí daného HTTP požiadavku.

Metódy

- `OPTIONS` – Reprezentuje požiadavku na popis komunikačných schopností príjemcu. Umožňuje klientovi vopred zistiť komunikačné možnosti bez nutnosti odosielania konkrétneho HTTP požiadavku na daný zdroj dát. [8]
- `GET` – Vyžiadanie si konkrétnych dát. Požiadavky s metódou GET by dáta mali výhradne získavať, nie ich odosielať. [8]
- `POST` – Odoslanie dát na server. Dáta sú priložené v tele správy. Typ odosielaných dát určuje hlavička *Content-Type*. [8]
- `PUT` – Vytvorenie nových alebo úprava existujúcich dát pomocou dát priložených v tele správy. [8]
- `DELETE` – Odstránenie dát. [8]

Typy metód, ktoré nesúvisia s touto prácou boli zámerne neuvedené.

```
fetch("http://localhost:5000/uzivatelia", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
  },
  body: JSON.STRINGIFY({
    meno: "Jan",
    priezvisko: "Novak"
  })
});
```

Výpis 2.9: Príklad odoslania HTTP POST metódy v prostredí TypeScript.

2.3.2 Node.js

Asynchrónny udalosťami riadený Javascriptový runtime[11] vytvorený pre budovanie škálovateľných sieťových aplikácii [10]. Node.js umožňuje spustiť JavaScriptový kód mimo prostredia webových prehliadačov a poskytuje rozšírené funkcionality ako napríklad prístup k súborovému systému alebo kryptografické metódy.

[11] runtime – behové prostredie programu

Node Package Manager - NPM

Najväčší softvérový register na svete. NPM umožňuje vývojárom open-source softvéru zdieľať JavaScriptové balíčky verejne alebo si vytvoriť súkromný repozitár a balíčky zdieľať iba v organizácií. [9]

NPM je rozdelené do troch hlavných častí:

- `Webová stránka` – Vyhľadávanie balíčkov, správu profilov a organizácií. [9]
- `Command Line Interface (CLI)` – Konzolové rozhranie pre interakciu s NPM. [9]
- `Register` – Verejná databáza JavaScriptového softvéru a metadát. [9]

Hlavnou časťou je pre vývojára práve `CLI`, ktoré mu umožňuje spravovať balíčky vo svojej aplikácií. Nainštalované balíčky ale ich závislosti sú inštalované do zložky v koreňovom adresári projektu `node_modules`. Nastavenia, informácie o projekte a zoznam nainštalovaných balíčkov sa uchováva v súbore `package.json`, taktiež umiestenom v koreňovom adresári projektu.

2.3.3 Relačné databázy

V dnešnej dobe štandardný typ databázy pre ukladanie a prístup k dátam vo vzájomných väzbách. Dáta sú rozdelené do tabuliek, kde každý riadok reprezentuje jednu entitu. Entita je vždy identifikovateľná svojim primárnym kľúčom, ktorý musí byť v danej tabuľke unikátny. Každý stĺpec tabuľky má vopred definovaný dátový typ a nesie hodnoty pre každý element v tabuľke.

SQL (*Structured Query Language*) je štandardizovaný dotazovací jazyk pre prístup a manipuláciu s dátami v relačných databázach.

```
1 SELECT * from Users;
```

Výpis 2.10: Príklad jednoduchého dotazu v jazyku SQL.

PostgreSQL

Plnohodnotný open-source objektovo-orientovaný databázový systém, ktorý je v aktívnom vývoji už viac ako 30 rokov. PostgreSQL je kompatibilný so všetkými hlavnými platformami ako Linux, MacOS, Solaris a Windows. Tento systém si obľúbili mnohé veľké spoločnosti po celom svete. [14]

Prisma

Prisma je open-source *sada nástrojov pre správu databázy*. Nahradzuje tradičné ORM[12] a uľahčuje prístup k databáze pomocou automaticky generovaného klienta pre zostavovanie dotazov. Prisma je implementovaná v jazyku TypeScript, čo znamená, že *podporuje striktnú typovú kontrolu* a zmenšuje tým pravdepodobnosť chyby vo vývoji. [11]

[12]ORM – Object-relational mapping

```
await prisma.users.create({
    data: {
    firstName: "Alice",
    email: "alice@prisma.io",
    active: true,
  }
})
```

Výpis 2.11: Príklad tvorby užívateľa pomocou nástroja Prisma. [11]

2.4 GraphQL

Dotazovací (query) jazyk pre API a runtime na strane servera pre vykonávanie dotazov pomocou *vopred definovaného* typového systému. GraphQL nie je viazané žiadnou špecifickou technológou ukladania dát, naopak pracuje ako backend pre už existujúci kód a dáta. [13] GraphQL nie je implementácia, ale iba špecifikácia. Existujú mnohé implementácie GraphQL implementované v mnohých programovacích jazykoch, na rôznych platformách.

GraphQL služba je vytvorená definovaním typov a ich jednotlivých polí. Každému polu každého typu je následne priradená funkcia. [13] Tieto funkcie sú nazývané „*resolvers*“. Tieto funkcie vyhodnocujú dáta jednotlivých polí pri prichádzajúcej požiadavke.

```
type Query {
  me: User
}

type User {
  id: ID
  name: String
}
```

Výpis 2.12: Príklad jednoduchej GraphQL schémy. [13]

Po spustení GraphQL služby (typicky URL adresa dostupná cez HTTPS) očakáva GraphQL dotazy, ktoré spracováva a vyhodnocuje. Obdržané dotazy najskôr skontroluje, či požadujú iba typy a polia, ktoré sú definované. V prípade, že sú dotazy v poriadku, Graphql spustí požadované resolvery a vracia výsledok. [13]

GraphQL rozlišuje tri typy operácií s dátami. Prvý typ operácie je *query* a slúži výhradne na čítanie dát. Druhým typ operácie sa nazýva *mutation* a slúži na úpravu, vkladanie alebo odstránenie dát. Tretím a posledným typom operácie je *subscription*. Ak GraphQL služba spracuje operáciu typu *subscription*, pošle vyžiadané dáta klientovi pri každej ich zmene.

Pre ukážku, *query* dotaz na GraphQL službu so schémou ukázanou vo výpise 2.12:

```
{
  me {
    name
  }
}
```

Výpis 2.13: Príklad *query* dotazu na GraphQL službu. [13]

... by po spracovaní GraphQL službou mohol vrátiť JSON[13] objekt:

```
{
  "me": {
    "name": "Luke Skywalker"
  }
}
```

Výpis 2.14: Príklad dát obdržaných z GraphQL služby. [13]

2.4.1 Apollo Client

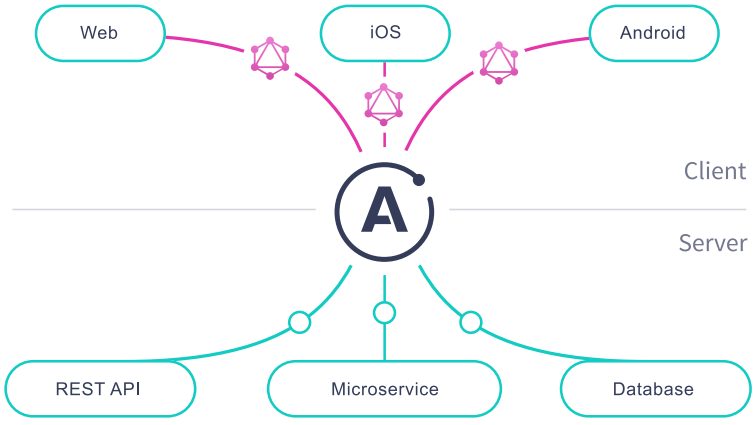
Implementácia GraphQL špecifikácie pre klientské aplikácie. Apollo Client je priamo prepojený s knižnicou React a poskytuje nástroje umožňujúce rýchly vývoj webových aplikácií, ktoré získavajú dáta z GraphQL služby.

> Apollo Client je knižnica umožňujúca plnohodnotnú správu stavu dát v JavaScriptových aplikáciach. Umožňuje vývojárovi len napísať požadovaný GraphQL dotaz a Apollo Client sa už postará o získanie, uloženie dát do cache a obnovenie uživateľského rozhrania. Využívanie knižnice Apollo Client vedie vývojára ku správnej štruktúre kódu. (Dokumentácia Apollo GraphQL [6])

Apollo GraphQL umožňuje využiť svoju cache aj pre ukladanie lokálneho stavu aplikácie. Vďaka tomu sa pre danú aplikáciu stáva jediným zdrojom dát.

2.4.2 Apollo Server

Serverová časť GraphQL implementácie, kompatibilná so všetkými GraphQL kientami, vrátane Apollo Client. Apollo Server dokáže pracovať ako samostatný GraphQL server alebo ako súčasť už existujúcej aplikácie ako napríklad Express[14] server. [6]



Obr. 2.2: Diagram architektúry Apollo Server. [6]

[13] JSON - javascript object notation
[14] https://expressjs.com/

Apollo Server dokáže pôsobiť ako *jednotná brána* pre všetky klientské aplikácie. Brána má predom definovanú schému, tzn. že klient presne vie aké operácie môže vykonávať a aké dáta môže očakávať späť.

2.5 Slack API

Aplikácia pre tímovú a pracovnú komunikáciu Slack umožňuje vývojárom vytvoriť vlastné aplikácie a prepojiť ich so Slackom pomocou ich *otvorenej API*. [12]

Aplikácie môžu vykonávať niektoré akcie bežných užívateľov:

- *Odosielať správy* do rôznych koverzácií. [12]
- *Čítať správy a konverzácie*. [12]
- *Vytvoriť, archivovať a spravovať konverzácie*. [12]
- *Reagovať na označenia* od užívateľov. [12]

... no API aplikáciam umožňuje aj akcie mimo uživateľských práv:

- *Otvoriť dialógové okná* pre získanie alebo zobrazenie extra informácií. [12]
- *Zostaviť interaktívne komponenty* na ktoré môzu reagovať [12]
- *Vytvoriť a aktualizovať „Home tab“ (domovskú obrazovku)* kde môže užívateľ interagovať s aplikáciou. [12]
- *Definovať skratky* vďaka ktorým užívateľ môže rýchlo vykonať akcie v aplikácií. [12]

2.5.1 Block Kit

Framework pre tvorbu uživateľského rozhrania pre Slack aplikácie, ktorý ponúka kompromis kontroly a flexibility pri budovaní rozhraní v správach. [12] Uživateľské rozhranie sa skladá pomocou blokov, ktoré sú vo forme JSON objekov prenášané do Slack aplikácie. Bloky sa delia na dve hlavné kategórie – *interactive* a *layout* bloky.

```
{
  "type": "section",
  "text": {
    "type": "mrkdwn",
    "text": "Hello, *World*!"
  }
},
```

Výpis 2.15: Príklad jednoduchého bloku v Slack aplikácií.

Bloky sa vkladajú do lineárneho zoznamu nazvanom *view*, ktorý je následne odoslaný na príslušnú časť API, aby bol spracovaný. Niektoré bloky podporujú formátovanie pomocou modifikovanej verzie jazyka *markdown*.

Kapitola 3

Návrh riešenia

Kapitola popisuje jednotlivé etapy vzniku návrhu služby, ktorá je predmetom tejto práce – pod názvom „**Slackify**“. Výsledný návrh vznikol po nadobudnutí požadovaných znalostí a analýz požiadavkov kladených na výsledný produkt.

3.1 Zadanie práce

Zadanie, ktorého celé znenie je priložené na strane 2, popisuje výsledné riešenie práce ako *headless* redakčný systém. Súhrn hlavných požiadaviek na systém je nasledovný:

- Užívateľ systému by mal byť schopný spravovať obsah v *plnej miere* a bez obmedzení v prostredí aplikácie Slack.
- Užívateľ by mal byť schopný začať využívať službu *okamžite* po nainštalovaní do svojho pracovného prostredia v aplikácií Slack. To znamená, že žiadna konfigurácia alebo nasadenie na vlastnú infraštruktúru by nemalo byť požadované.
- Všetky vytvorené dáta by mali byť dostupné cez verejné API.
- Systém by mal disponovať webovým rozhraním, odkiaľ by malo byť možné spravovať obsah a nastavenia.

Hlavnou výsadou práce nie je čo najväčšia flexibilita, ale jednoduchosť v použití a možnosť rýchleho nasadenia.

3.1.1 Odchýlky návrhu od pôvodného zadania

Slack API umožňuje vytvoriť uživateľské rozhranie zložené z maximálneho počtu 100 blokov. Táto skutočnosť znamená, že nie je možné zobrazovať napríklad dlhé zoznamy prvkov. Pre tento dôvod je maximálny počet zobrazených prvkov v zoznamoch Slack aplikácie limitovaný. Prvky zoznamov nad určený limit sú skryté a uzívateľ ich môže spravovať výhradne vo webovom prostredí.

3.2 Návrh architektúry

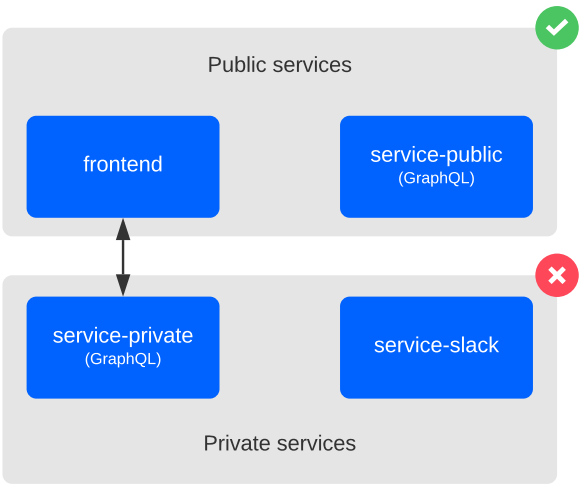
Na začiatkoch navrhovania architektúry bola služba rozdelená do dvoch hlavných častí podľa vzoru *backend* a *frontend*. Frontend bola webová služba a backend poskytovala GraphQL bránu pre prístup k dátam a logike pre frontend, ale slúžila aj ako prístupový bod pre

koncových užívateľov headless redakčného systému a tiež ako bod pre komunikáciu so Slack API. Dáta boli ukladané v relačnej databáze.

Aj keď v prvopočiatkoch tvorby návrhu sa táto architekúra zdala ako vyhovujúca, neskôr sa ukázalo ako lepšia možnosť nasledujúce rozdelenie do štyroch menších služieb:

- `frontend` – Webová aplikácia poskytujúca prístup k obsahu a nastaveniam headless redakčného systému. Táto časť zostala nezmenená.
- `service-slack` – Serverová aplikácia zodpovedná za komunikáciu so Slack API. Udalosti a akcie prichádzajúce zo Slack aplikácie smerujú na túto službu, kde sú spracované a vyhodnotené.
- `service-private` – Serverová GraphQL aplikácia poskytujúca logiku a prístup k dátam „frontend“ službe.
- `service-public` – Serverová GraphQL aplikácia poskytujúca prístup k obsahu redakčného systému koncovým užívateľom.
- `database` – Relačná databáza obsahujúca uživateľské dáta, nastavenia a obsah od koncových užívateľov.

Takáto architektúra sa oproti jednej veľkej backend službe ukázala ako výhodná z dvoch hlavných dôvodov. Prvým dôvodom je *omnoho jednoduchšia údržba* viacerých menších služieb (napríklad udržovanie logickej súborovej štruktúry bolo pri jednej monolitickej službe obtiažne). Druhým významným dôvodom je možnosť jednotlivé služby *verziovať a vydávať samostatne*, pretože fungujú nezávisle jedna na druhej.



Obr. 3.1: Vizualizácia architektúry podľa návrhu.

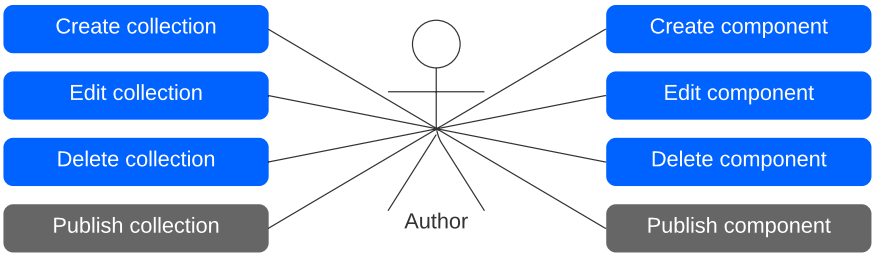
Jednotlivé služby sú ešte rozdelené do dvoch skupín. `Frontend` a `service-public` sú služby dostupné pre verejnosť (*public services*), `service-private` a `service-slack` dostupné pre verejnosť nie sú (*private services*).

3.3 Diagram prípadov použitia

Priložené diagramy prípadov použitia zobrazujú chovanie systému z pohľadu autora a zariadenia, ktoré sa snaží získať obsah z redakčného systému cez verejné aplikačné rozhranie `service-public` (charakterizované mobilným telefónom).

3.3.1 Autor

Po nainštalovaní aplikácie **Slackify** sa stáva každý člen pracovného prostredia s prístupom ku aplikácií *Autorom*.



Obr. 3.2: Diagram prípadov použitia - *Autor*

.

Autor môže v prostredí **Slackify** vykonávať nasledujúce akcie:

- `Vytvoriť kolekciu/komponentu` – Po zadaní a potvrdení vstupných údajov je nová kolekcia alebo komponenta pridaná do databázy. Po vytvorení majú východiskový stav *nepublikovaná*.
- `Upraviť kolekciu/komponentu` – Kolekcie alebo komponety je možné upraviť v ľubovoľný čas. Jediná hodnota, ktorú nie je možné modifikovať je typ kolekcie alebo komponenty.
- `Zmazať kolekciu/komponentu` – Kolekcie alebo komponety je možné zmazať v ľubovoľný čas. Pri zmazaní kolekcie sa zmazú aj všetky komponenty k nej priradené.
- `Publikovať kolekciu/komponentu` – Publikovaná kolekcia alebo komponenta je okamžite dostupná verejne. *Author* rovnako môže kolekcie alebo komponenty skryť.

Literatúra

[1] Cambridge University. *Cambridge Dictionary* [online]. [cit. 2020-04-03]. Dostupné z: https://dictionary.cambridge.org/.

[2] Facebook Inc.. *React* [online]. [cit. 2020-04-05]. Dostupné z: https://reactjs.org/.

[3] Fielding, R., UC Irvine, Gettys, J., Compaq/W3C, J. Mogul et al. *Hypertext Transfer Protocol – HTTP/1.1* [Internet Requests for Comments]. RFC. 1997.

[4] Komunita Netlify CMS. *Dokumentácia Netlify CMS* [online]. [cit. 2020-04-03]. Dostupné z: https://strapi.io/documentation/.

[5] Komunita Strapi.io. *Dokumentácia Strapi.io* [online]. [cit. 2020-04-03]. Dostupné z: https://strapi.io/documentation/.

[6] Meteor Development Group Inc.. *Dokumentácia Apollo GraphQL* [online]. [cit. 2020-04-05]. Dostupné z: https://www.apollographql.com/docs/.

[7] Microsoft Corporation. *TypeScript Language* [online]. [cit. 2020-04-04]. Dostupné z: https://www.typescriptlang.org.

[8] Mozilla and individual contributors. *Mozilla MDN web docs* [online]. [cit. 2020-04-04]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/.

[9] npm, Inc.. *Node Package Manager* [online]. [cit. 2020-04-04]. Dostupné z: https://www.npmjs.com/.

[10] OpenJS Foundation a Joyent. *Node.js* [online]. [cit. 2020-04-04]. Dostupné z: https://nodejs.org/en/.

[11] Prisma. *Prisma* [online]. [cit. 2020-04-05]. Dostupné z: https://www.prisma.io/.

[12] Slack. *Slack API* [online]. [cit. 2020-04-05]. Dostupné z: https://api.slack.com/.

[13] The GraphQL Foundation. *GraphQL* [online]. [cit. 2020-04-05]. Dostupné z: https://graphql.org/.

[14] The PostgreSQL Global Development Group. *PostgreSQL* [online]. [cit. 2020-04-05]. Dostupné z: https://www.postgresql.org/.

[15] ZEIT, Inc.. *Next.js* [online]. [cit. 2020-04-05]. Dostupné z: https://nextjs.org/.