



STŘEDNÍ PRŮMYSLOVÁ ŠKOLA
OBCHODNÍ AKADEMIE
JAZYKOVÁ ŠKOLA
FRÝDEK-MÍSTEK

Příspěvková organizace
Moravskoslezského kraje



MATURITNÍ ZKOUŠKA

PRAKTICKÁ ZKOUŠKA Z ODBORNÝCH PŘEDMĚTŮ

Téma č.3

Fotografický web a kompletní administrační systém

Obor vzdělání:

18 – 20 – M/01 Informační technologie

Třída:

4. IT

Autor práce:

Jiří Vala

Školní rok:

2021/22

Vedoucí učitel práce:

Ing. Jiří Sumbal

Prohlášení autora

„Prohlašuji, že jsem tuto práci vypracoval samostatně a použil jsem literárních pramenů a informací, které cituji a uvádím v seznamu použité literatury a dalších zdrojů informací.“

Ve Frýdku-Místku, dne: podpis

Zadání práce

1. Návrh a koncepce webu - shrnutí požadavků zákazníka a návrh řešení
2. Použité platformy, technologie a software
3. Vývoj webové stránky
4. Vývoj administračního systému
5. Hosting
6. Vytvoření vlastního webového serveru a virtuálního webu na bázi virtuálního počítače

Anotace

Tato práce se zabývá průběhem vývoje webové stránky pro amatérského fotografa ve formě virtuální galerie a jejího administračního systému. Popisuje použité technologie a postupy ve vývoji spolu s konfigurací jednotlivých služeb klíčových ke spuštění celého projektu na produkční server a jeho správné a dlouhodobě nezávadné fungování. Cílem práce bylo vytvořit funkční produkt v podobě webu a jeho administračního systému se všemi aspekty komerčního projektu pro koncového zákazníka, který jej bude používat jako své virtuální portfolio.

Anotation

This thesis focuses on the development process of a website for an amateur photographer in a form of virtual gallery and it's content management system. Describes technologies and procedures used in the development as well as configuration of individual services mandatory for deployment of the whole project on a production server and it's seamless and longterm functioning. The goal of this work was to create a fully functioning product in a form of website and it's content management system with all aspects of commercial project for a end customer, which will use it as his virtual portfolio.

Klíčová slova

webová stránka; webová aplikace; administrační systém; webová prezentace; webová galerie; vývoj; javascript; react; linux; webhosting

Keywords

website; web application; content management system; web presentation; web gallery; development; javascript; react; linux; webhosting;

Obsah

1	Vymezení pojmu	6
1.1	Seznam použitých zkratek	10
2	Úvod	11
3	Klíčové vlastnosti a požadavky projektu	12
3.1	Serverless architektura	13
3.2	Moderní technologie	14
3.2.1	Frontend a UI/UX	14
3.2.2	Backend	17
3.3	Hosting	20
4	Struktura projektu	21
4.1	Komerční část	21
4.2	Maturitní část	22
5	Průběh vývoje	23
5.1	Komunikace se zákazníkem	24
5.2	Organizace úkolů a změn k implementaci	24
6	Webová prezentace	25
6.1	Vanilla Javascript routing	27
7	Administrační systém	29
7.1	Využití frameworku React.js	32
7.1.1	React.js vs Vanilla Javascript	32
8	VPS - Virtual Private Server	33
8.1	VPS provideři	33

8.2 Konfigurace VPS	34
8.2.1 Operační systém GNU/Linux	34
8.2.2 Vytvoření developer uživatele	35
8.2.3 Konfigurace prostředí	36
8.3 Konfigurace CI/CD	38
8.3.1 Naklonování projektu na lokální úložiště	40
8.3.2 Nginx konfigurace pro web i administrační systém	41
9 Seznam obrázků	45
10 Seznam tabulek	46
11 Seznam použitého softwaru	47
12 Seznam použité literatury	48
13 Citace	50

1 Vymezení pojmu

1. **Adobe Flash player** - Bývalý počítačový program určený pro prohlížení multimedialního obsahu vytvořeného na zastaralé softwarové platformě Adobe Flash.
2. **Algoritmus** - Proces jednotlivých příkazů za účelem splnění zadaného úkolu.
3. **API** - API je soubor procedur, funkcí a protokolů zajišťující komunikaci mezi dvěma platformami, které si vzájemně vyměňují data (například dvě rozdílné aplikace). K public API má přístup kdokoliv, k private API jen autentifikovaní uživatelé například pomocí unikátního ID tokenu.
4. **Auth** - Autentizace.
5. **Backend** - Logická část aplikace nebo webu. Požadavky, které uživatel aplikace provede (například přejmenování záznamu v databázi) a data (například všechny záznamy), které si vyžádá, tento server nejdříve zpracuje, případně vytáhne z databáze a pak je odešle uživateli aplikace, která je následně zobrazí, či případně upraví v databázi.
6. **Bundler** - Program, který vygeneruje jeden soubor scriptů (bundle) ze všech scriptů používaných v projektu.
7. **Cache** - Část paměti uchovávající data, která se používají v aktuální iteraci programu.
8. **CI/CD pipeline** - Představuje sérii kroků, které je nutné provést pro správné dodání nové verze softwaru.
9. **Cloud Computing Provider** - Poskytovatel clouдовých služeb a infrastruktury (serverů).
10. **Cloud Storage** - Úložný prostor v cloudu. Nejčastěji se používá pro ukládání dokumentů a fotografií. Nejznámější je Google Drive.
11. **Commit** - Potvrzení změn a jejich uložení do lokálního repozitáře verzovacího systému, který ji zařadí do historie změn.
12. **Content Management System** - Administrační systém, je systém, který umožňuje manipulaci s daty a obsahem webové stránky či webové (ale i desktopové) aplikace.

13. **Continuous Delivery** - Proces, který umožnuje nepřetržité dodávání aktuální verze projektu k testování či produkčnímu spuštění. V praxi to znamená, že pokud tým vývojářů provede změnu a uloží ji do remote repositáře (místo pro ukládání zdrojových kódů v cloudu), tato změna se automaticky projeví (deployne) na produkční server (server, na kterém je spuštěna aktuální verze projektu a je poskytována uživatelům z venčí).
14. **DDOS útok** - Distributed Denial Of Service - je útok jehož cílem je pomocí sítě mnoha virtuálních počítačů, které v jednu chvíli budou provádět požadavky na daný server, vyřadit tento server z provozu, jelikož nezvládne nápor takového počtu požadavků a zhroutí se.
15. **Deploy** - Nasazení softwaru je v informatice souhrn všech činností, které připravují softwarový systém k použití.
16. **Developer Experience** - Popisuje jednoduchost a intuitivitu, kterou vývojáři zažívají během práce s určitým softwarem nebo nástrojem.
17. **Devops** - Složenina anglických výrazů Development a Operations. Je to přístup k vývoji software, který zdůrazňuje komunikaci, spolupráci a integraci mezi vývojářem a odborníky na informační technologie z provozu.
18. **Element (HTML dokumentu)** - HTML dokument se skládá z elementů, jako jsou například *div* a nebo *section*.
19. **Endpoint** - Koncový bod.
20. **Fetching** - Proces během kterého se "stahuje" data z databáze. Tento anglický výraz doslova znamená "přinést".
21. **Framework** - Jedná se o podpůrnou softwarovou strukturu, která zjednoduší vývoj a organizaci projektu v daném jazyce. Může obsahovat další funkce, podpůrné programy nebo vzory a doporučené postupy ve vývoji.
22. **Frontend** - Jedná se o grafické uživatelské rozhraní, v rámci webových aplikací a stránek většinou nejčastěji pomocí značkovacího jazyka *HTML*, stylovacího jazyka *CSS* a scriptovacího jazyka *Javascript*.

23. **Hosting (Webhosting)** - Znamená pronájem prostoru na cizím webovém serveru za účelem spuštění webové stránky či aplikace.
24. **Knihovna** - Jedná se souhrn procedur a funkcí, často však také konstant a datových typů, které usnadňují vývojáři práci tím, že může použít již hotový kód.
25. **Load Balancer** - Softwarový (ale existuje i hardwarový) nástroj k vyvažování náporu na část infrastruktury projektu, například webového serveru.
26. **Open Source** - Jedná se o projekt s otevřeným zdrojovým kódem, což znamená, že na jeho vývoji se může podílet každý a každý si jej může stáhnout. Popularita open-source projektů v posledních letech velmi roste.
27. **Out-of-the-box** - Předem připraveno.
28. **Pagination** - Jedná se o proces rozdělování obsahu na jednotlivé stránky a podstránky. Nejčastěji ji vidíme ve formě čísel na konci webové stránky.
29. **Performance** - Výkon, výkonnost označuje měřítko pro určování funkčnosti webové stránky nebo aplikace.
30. **Plugin** - Modul pro daný software.
31. **Preprocessor** - Počítačový program, který zpracovává vstup tak, aby jej dále mohl zpracovat jiný program.
32. **Pull** - Příkaz verzovacího systému git, díky kterému se stáhnou nejnovější změny z *remote* repozitáře do *lokálního repozitáře*.
33. **Push** - Příkaz verzovacího systému git, díky kterému se nahrají nejnovější změny z *lokálního repozitáře* do *remote* repozitáře.
34. **Push Notifikace** - Notifikace, které se zobrazují uživateli na mobilním zařízení v notifikační listě nebo na zamykací obrazovce.
35. **Repozitář** - Softwarový repozitář je v informatice označení pro místo, odkud mohou být staženy a nainstalovány softwarové balíčky.
36. **Route** - Označení pro *endpoint* (podstránku) React aplikace.

37. **Routing** - Proces, pomocí kterého React aplikace rozhodne, na který *endpoint* má uživatele poslat.
38. **Runtime** - Běhové prostředí je v informatice skupina software, určená na podporu realizace počítačových programů napsaných v některém z programovacích jazyků.
39. **Screenshot** - Snímek obrazovky.
40. **Script** - Soubor kroků a příkazů, které *shell* po spuštění *scriptu* vykoná.
41. **Shell** - Jedná se o vrstvu operačního systému, která provádí příkazy a spouští programy.
42. **Single Page Application** - Jednostránková aplikace je webová aplikace nebo webová stránka, která komunikuje s uživatelem dynamickým přepisováním aktuální webové stránky novými daty z webového serveru, namísto výchozí metody webového prohlížeče načítání celých nových stránek.
43. **Symlink** - Symbolický link odkazuje na jiný soubor nebo adresář. Odkaz je realizován jako malý speciální soubor, který obsahuje absolutní nebo relativní cestu k cílovému souboru.

1.1 Seznam použitých zkratek

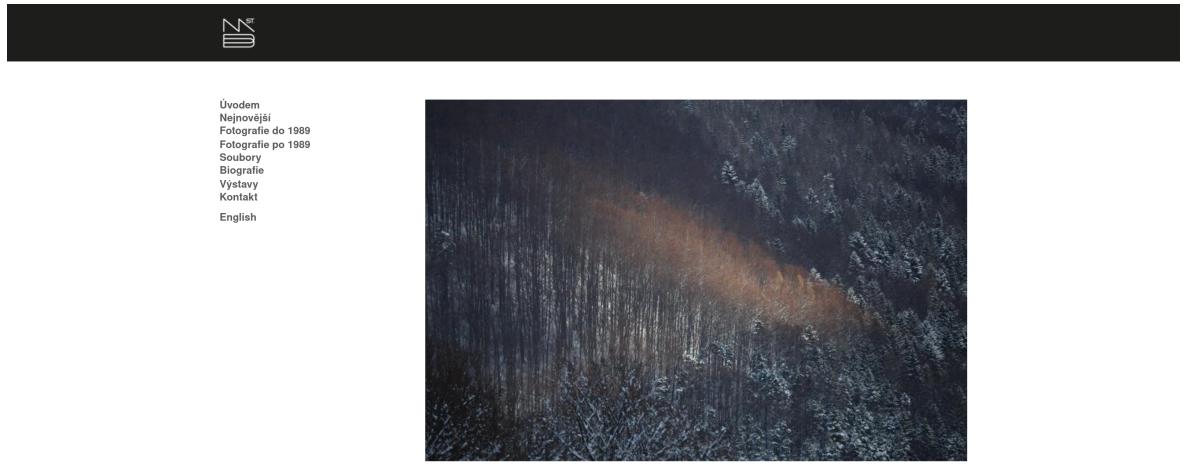
1. **API** - Application programming interface
2. **CD** - Continuous deployment/delivery
3. **CI** - Continuous integratiaon
4. **CLI** - Command line interface
5. **CMS** - Content management system
6. **CSS** - Cascading style sheet
7. **DDOS** - Distributed denial of service
8. **DMP** - Dlouhodobá maturitní práce
9. **DOM** - Document object model
10. **GNU** - GNU's not unix
11. **GUI** - Graphical user interface
12. **HTML** - Hypertext markup language
13. **ISO** - Archive of contents
14. **SSH** - Secure shell
15. **SSL** - Secure socket layers
16. **UI/UX** - User interface/User experience
17. **URL** - Uniform resource locator
18. **VM** - Virtual machine
19. **VPS** - Virtual private server
20. **YAML** - YAML ain't markup language

2 Úvod

Pan Milan Bureš st. (dále jen fotograf nebo zákazník) je amatérský fotograf, který si již před pár lety nechal na zakázku vytvořit webovou stránku, na které by prezentoval své fotografie, zachycující náhodné okamžiky jeho života. Nicméně postupem času se jeho web i administrační systém stali zastaralými a bylo potřeba celý systém zmodernizovat i vzhledem k faktu, že jeho webová stránka fungovala pomocí *Adobe Flash Player*, kterému skončila podpora a není již podporovaný moderními prohlížeči. Proto mě v rámci dlouhodobé maturitní práce zprostředkované mým garantem kontaktoval a požádal, zda-li bych jeho webovou prezentaci mohl zmodernizovat. Mým úkolem bylo vytvořit web, který bude sloužit jako virtuální galerie jeho fotografií a bude zobrazitelná a funkční na všech zařízeních a také administrační systém (dále jen CMS), který bude umožňovat nahrávání, správu a manipulaci fotografií či alb, ve kterých se jednotlivé fotografie seskupují. V rámci mé maturitní práce bylo potřeba, aby projekt obsahoval i část jiného předmětu, než jen programování (celý systém jsem naprogramoval) a databáze (data a fotografie, které fotograf nahraje/uloží). Rozhodl jsem se tedy, že si vytvořím vlastní server běžící na operačním systému Linux, na kterém budu hostovat vlastní instanci celého systému a bude obsahovat funkcionality tzv. *Continuous Deployment*, což přesně kopíruje funkcionality komerční instance projektu určené jen pro účely zákazníka.

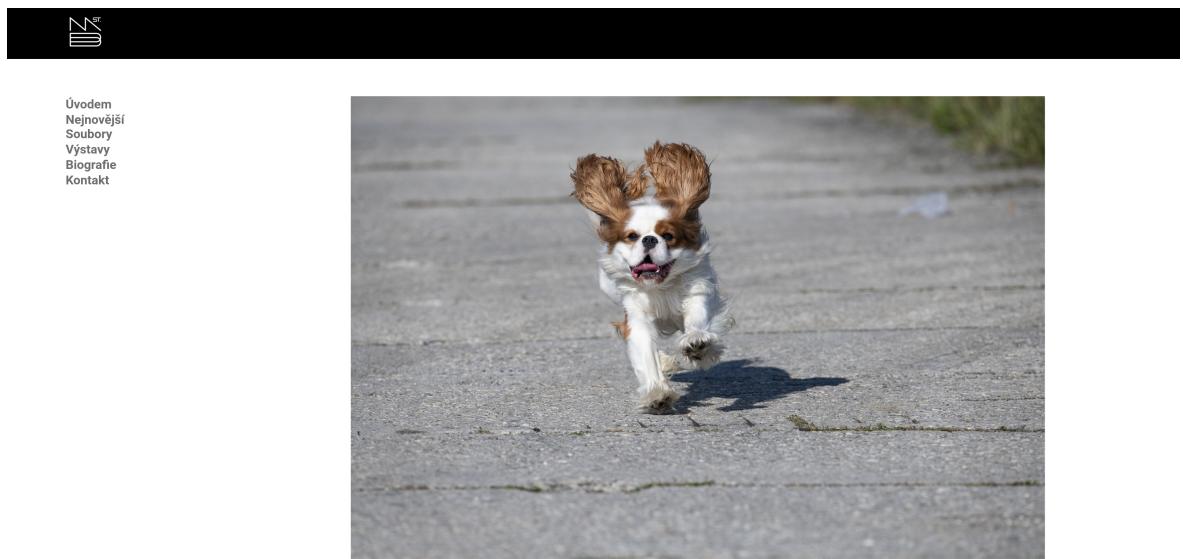
3 Klíčové vlastnosti a požadavky projektu

Během vývoje webové prezentace byl kladen velký důraz na to, aby se co nejvíce podobala stávající verzi. Neveškeré elementy webu byly replikovatelné a také jsem se chtěl co nejvíce řídit pravidly dobrého webu. Pro porovnání jsou zde screenshots jednotlivých webových stránek.



Obr.1 - Původní webová stránka

(Zdroj: Autor práce)



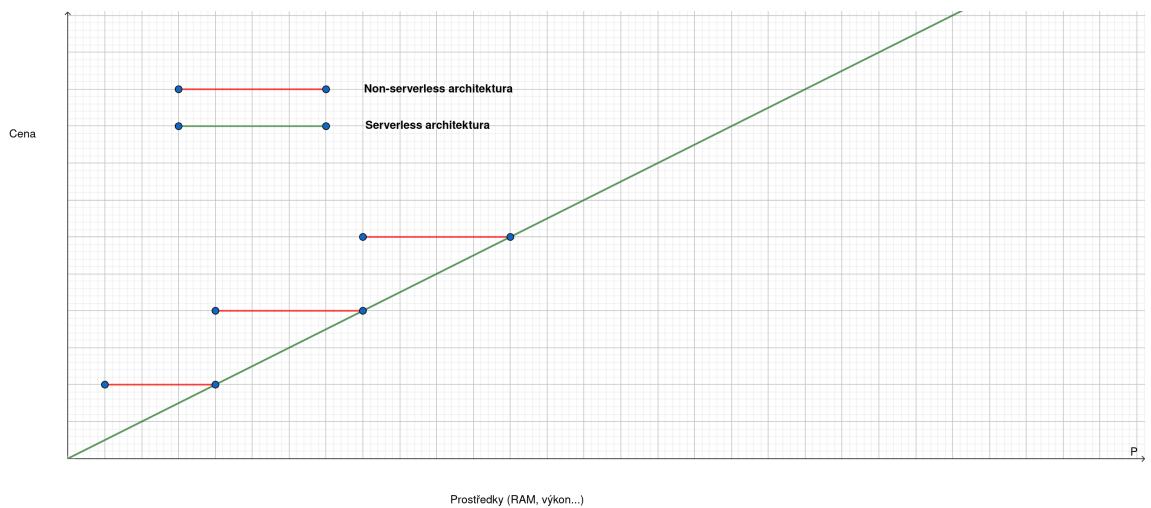
2021 © Mgr. Milan Bureš st.
Made with ❤ by Orexin.

Obr.2 - Mnou vytvořená webová stránka
(Zdroj: Autor práce)

3.1 Serverless architektura

Ačkoliv je samozřejmostí, že je jak webová prezentace, tak CMS hostovaná na nějakém fyzickém serveru, k požadavkům jako *fetching* dat z databáze nepoužívám žádný další (backend) server. To znamená, že jsem sice odkázaný kompletně na svého *cloud computing providera*, což je v mé případě Firebase od Google, ale na druhou stranu se nemusím (a ani zákazník) starat o údržbu vlastního serveru a ani jeho první konfiguraci, vše je tzv. *out-of-the-box* připravené k použití. Díky serverless architektuře jsem nemusel vytvářet žádný *backend server s private API*, pomocí kterého bych dostával data z databáze a *cloud storage* do webové prezentace nebo administračního systému. Serverless architektura má mnoho výhod, jedna z těch hlavních a největších je samozřejmě absence potřeby konfigurovat a později spravovat vlastní server, jelikož to kompletně zprostředkovává daný *cloud computing provider*, ale také celková cena "pronájmu", jelikož ta se počítá podle počtu požadavků na danou službu (například fetchování dat z databáze). Další je například bezpečnost, jelikož jsou tyto architektury zabezpečené například proti DDOS útokům, ale také celková škálova-

telnost celého systému. *Cloud computing provider* automaticky škáluje prostředky (úložistě, výpočetní výkon apod.), které jsou k dispozici pro daný systém.



Obr.3 - Graf popisující škálovatelnost jednotlivých architektur
(Zdroj: Autor práce)

Na grafu jde vidět, že pokud si zvolíme cestu non-serverless architektury, musíme s narůstajícími požadavky výkon zvyšovat, což ale v praxi znamená dočasné odstavení systému od provozu a dodatečnou správu hardwaru a tento výpočetní výkon se nemění, pokud se opět nevylepší, zatímco serverless řešení se automaticky škáluje.

3.2 Moderní technologie

Jelikož byl původní systém postaven na poměrně zastaralých technologiích, bylo potřeba modernizovat a proto jsem během vývoje používal jen ty nejmodernější a mezi vývojáři velmi rozšířené technologie.

3.2.1 Frontend a UI/UX

Jedná se o grafické uživatelské rozhraní, v rámci webových aplikací nejčastěji vytvořené pomocí značkovacího jazyka *HTML*, stylovacího jazyka *CSS* a scriptovacího jazyka *Javascript*. Nicméně v dnešní době na tyto technologie, které existují delší dobu, existuje mnoho *frameworků* a *knihoven*, které nejen zjednodušují práci vývojářům, ale také celkově zlepšují

performance aplikace či webu, které jsou díky tomu rychlejší, intuitivnější a použitelnější, což ve výsledku znamená, že například návštěvník webové stránky neztratí zájem, jelikož se stránka nebude dlouze načítat. UI (user interface - uživatelské prostředí) a UX (user experience - v hrubém překladu uživatelská zkušenost) je sada pravidel a procedur, které mají za úkol zjednodušit uživatelskou interakci se systémem tak, aby se uživateli produkt používal co nejsnáze a byl co nejvíce intuitivní. Během vývoje administračního systému jsem se inspiroval návrhem, který jsem našel na designovém webu Dribble.com.

3.2.1.1 HTML,CSS a Javascript

Jedná se o základní technologie, se kterými se dnes vyvíjejí webové stránky či aplikace a všechny ostatní technologie si z nich něco vzaly. Základním stavebním kamenem webové stránky i aplikace je *HTML* dokument, který obsahuje všechny *elementy*, které má web zobrazovat. Tyto elementy nastyluje stylovací jazyk *CSS*, což znamená, že v *CSS* dokumentu vybereme jednotlivý *HTML* element pomocí *CSS* selektoru a nastavíme mu například jinou barvu, šířku či například nastavíme animaci, která se spustí po tom, co uživatel přes tento *element* přejede myší. Pokud má mít web nějakou funkcionality, například po zadání dvou čísel vypočítat součet, použijeme *Javascript*. Ten je vlastně logickým mozkem celého webu a stará se o početní operace, získávání dat z databáze a jejich následné vložení do struktury *HTML* dokumentu (DOM) nebo interaktivitu s uživatelem.

3.2.1.2 React.js

React.js je *Javascriptový framework* vyvíjený firmou Meta (Facebook) a *open-source* komunitou vývojářů po celém světě a k dnešnímu dni se jedná o nejvíce používaný *framework* na světě pro jeho jednoduchost a modularitu. Je optimální pro tvorby *SPA* (single page aplikace), jako je například administrační systém, protože velmi dobře pracuje s rychle měnícími se daty. Během vývoje *React* aplikace se celá aplikace rozdělí na jednotlivé komponenty, což má za následek jednoduchý a přehledný zdrojový kód a celou strukturu projektu, což je potřebné u velkého a komplexního projektu a já sám jsem během vývoje pocítil výrazný rozdíl v jednoduchosti vývoje oproti samotnému *vanilla Javascriptu* (bez použití *frameworku*)

čí knihovny.)

```
// Component - tento blok kódu mohu použít kolikrát, kolikrát potřebuji
import React from "react";
import "../style/components/SummaryWidget.css";
class SummaryWidget extends React.Component {
  constructor(props) {
    super(props);
  }
  render() {
    return (
      <div className="summary-widget">
        <div className="icon">
          <>{this.props.icon}</>
        </div>
        <div className="data">
          <p id="name">{this.props.data_name}</p>
          <p id="value">{this.props.data_value}</p>
        </div>
      </div>
    );
  }
}

export default SummaryWidget;
```

Obr.4 - Ukázka vytvoření jednoduché komponenty pomocí frameworku React.js

(Zdroj: Autor práce)

3.2.1.3 SASS/SCSS

SASS je *preprocessor* stylovacího jazyka CSS, což znamená, že styly psané v SCSS (sass) zpracuje a překompiluje do klasického CSS.

SCSS je další z mnoha stylovacích jazyků a jedná se o přímé rozšíření klasického CSS, přidává například proměnné a nebo nesting. To znamená, že jednotlivé selektory můžeme psát přímo hierarchicky do sebe a pod sebe, díky čemuž jsou jednotlivé bloky stylů přehlednější. Dále tzv. mixins, což jsou vlastně bloky stylů, které se dají použít vícekrát, a díky tomu

nemusí vývojář psát enormní počet nových stylů.

```
.element > p {  
    color: #b30e2c  
}  
.element > button {  
    border: solid #b30e2c 1px;  
    background-color: #b30e2c;  
    color: white;  
}
```

Obr.5 - Ukázka stylování v CSS

(Zdroj: Autor práce)

```
.element {  
    p {  
        color: $mainRed; // Promenna  
    }  
    button {  
        @include submit-btn; // Pouziti mixin pomocí jeho unikatniho jmena  
    }  
}
```

Obr.6 - Ukázka stylování v SCSS

(Zdroj: Autor práce)

3.2.2 Backend

3.2.2.1 Node.js

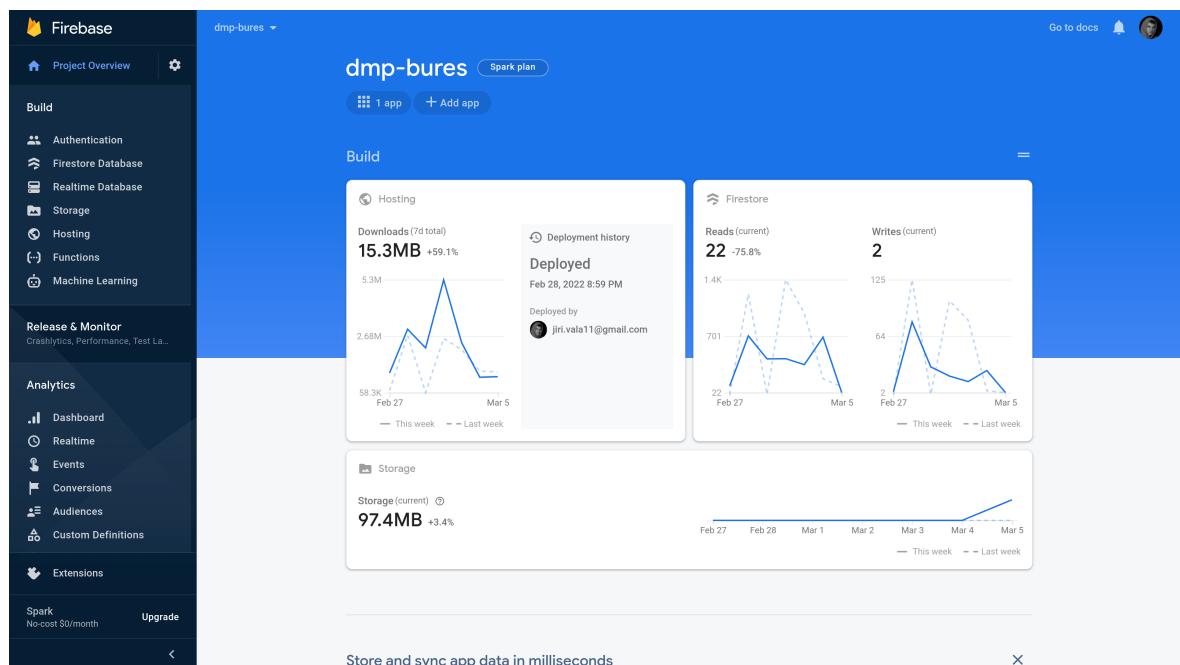
Node.js je tzv. *runtime* pro Javascript určený k vytváření vysoce škálovatelných *backend* aplikací a webových serverů. Toto má společné např. s jazykem PHP, který má stejné zaměření. Javascript se tedy díky tomuto prostředí dá používat i na serveru a ne jen na druhém konci, u klienta. Avšak na rozdíl např. od zmíněného PHP je v Node.js kladen důraz na vyšokou škálovatelnost, tzn. schopnost obsloužit mnoho připojených klientů naráz. Pro tuto vlastnost a vysokou výkonnost je dnes Node.js velmi oblíbený pro tvorbu tzv. API serverů pro klientské single page aplikace rovněž v Javascriptu.

3.2.2.2 Shell scripty

Shell scripty jsou sada příkazů, které po spuštění zpracovává a provede *shell*. Význam těchto scriptů v rámci tohoto projektu jsou *CI/CD* a *DevOps Operace*, které jsou detailněji popsány v bodu 8.3.

3.2.2.3 Firebase

Firebase je *cloud computing provider* nabízející služby jako *hosting* webových stránek, *hosting* databází, ale mnoho dalších jako je například machine learning, které však pro mě, vzhledem k povaze projektu, nejsou důležité. V tomto projektu zajišťuje všechny tyto zmíněné služby a nebýt Firebase, musel bych je všechny řešit sám, například pronájmem dalšího serveru, který by sloužil jako databázový apod. Databáze, které v rámci Firebase využívám, jsou tzv. *Firestore* a *Cloud Storage*.



Obr.7 - Nástěnka projektu ve Firebase Console

(Zdroj: Autor práce)

Firestore je NO-SQL databáze, což znamená, že jednotlivé záznamy uchovává v datovém typu JSON, což je key-value datový typ, který slouží pro jednoduché popisování parametru daného předmětu, například člověka. Pro mé účely (ukládání záznamů o jednotlivých albech

a fotografiích) je naprosto dostačující, ale jedinou nevýhodou je, že NO-SQL databáze nemůží vytvářet relace mezi jednotlivými tabulkami, tudíž, když jsem chtěl vytvořit relaci mezi albem a jeho fotografiemi, musel jsem v albu vytvořit pole s názvem `connectedImages` a v něm uchovávat *ID* všech těchto fotografií.

Obr.8 - Firebase Firestore záznamy jednotlivých alb

(Zdroj: Autor práce)

Cloud Storage je cloudové úložiště pro všechny typy souborů. Vzhledem k tomu, že by měl pan fotograf nahrávat jen fotografie, ve *scriptu*, který řídí nahrávání nových fotografií, je implementováno jednoduché ověření, zda-li soubor, který se snaží nahrát, je doopravdy typu obrázku/fotky. Nicméně kdyby si to zákazník přál, jsem schopen implementovat i nahrávání souborů jiných, než fotografií.

```
export const Images = {
  Meta: {
    /**
     * @param {*} File File that is uploaded to `input` element
     * @returns {bool} `true` if the file is type of image
     */
    isImage: (file) => {
      if (file["type"].split("/")[0] === "image") {
        return true;
      } else return false;
    },
    // ... další helper funkce
  };
};
```

Obr.9 - Ukázka funkce, která zjišťuje, zda je nahraný soubor typu image

(Zdroj: Autor práce)

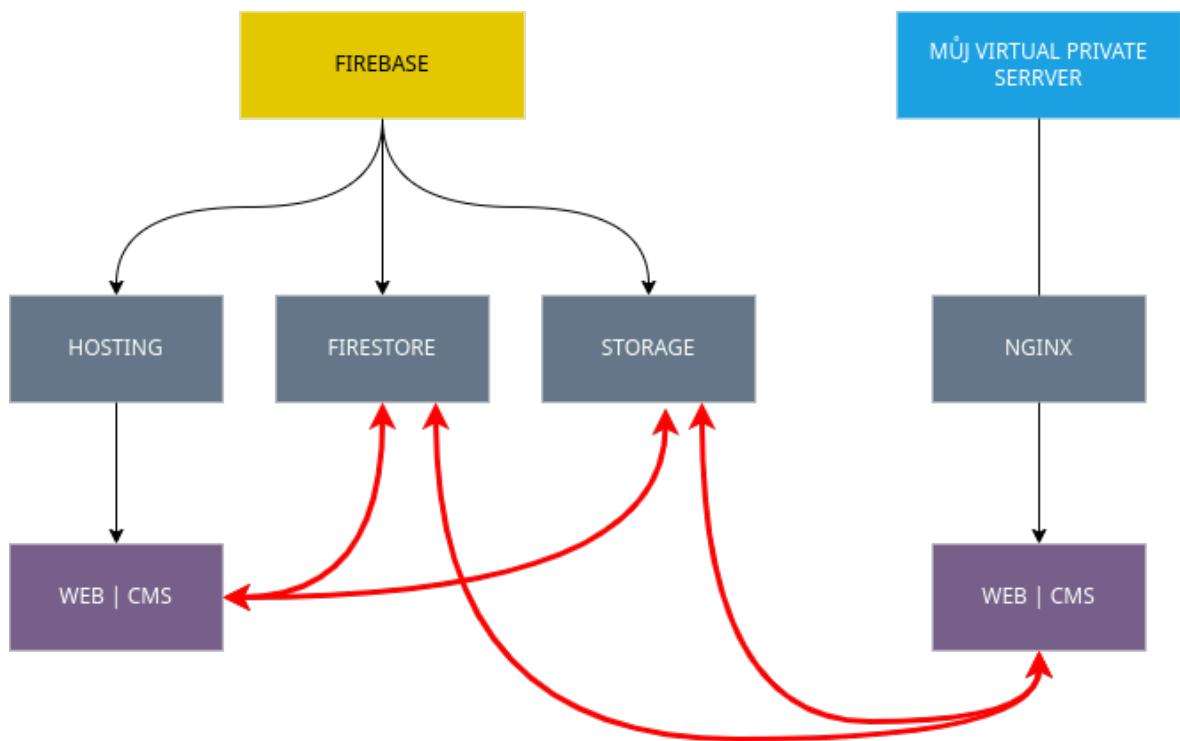
3.3 Hosting

Hosting je zřízen pomocí Firebase Hosting nejen pro jednoduchou integraci s Github repozitářem a předem připravenými CI/CD pipelines, ale také díky jeho ceně. V základním tarifu, jsou všechny služby Firebase (včetně *hostingu*), zdarma. V rámci tohoto tarifu je však *hosting* limitován na 10GB úložiště, tedy projekt nesmí přesáhnout stanovenou kvótu spolu s maximálním trafficem 360MB/den. Samozřejmostí je však SSL certifikát a možnost hostovat více než jednu stránku v rámci projektu.

Nastavení *hostingu* je u Firebase velmi jednoduché a díky CLI programu Firebase Tools je vývojář schopen deploynout projekt jedním jednoduchým příkazem, nebo vytvořit CI/CD script, který projekt deployne po pushnutí změn do repozitáře.

4 Struktura projektu

Projekt je rozdělen na dvě částí - komerční a maturitní (DMP). Co se týče části komerční, všechny služby okolo *hostingu* a databází bude zprostředkovávat Firebase, jelikož je to v rámci dlouhodobého hlediska nejjednodušší a hlavně nejlevnější řešení. Maturitní část (DMP) budu zcela spravovat já, mám proto pronajatý vlastní server, na kterém budu zprostředkovávat *hosting* a *backend* služby, nicméně všechna data budou poskytována a požadována z databází komerční části.



Obr.10 - Diagram popisující strukturu projektu. Vlevo je část komerční, vpravo maturitní
(Zdroj: Autor práce)

4.1 Komerční část

Jedná se o část projektu, která ke svému fungování využívá kompletně služeb Firebase. Jde o ostrou verzi systému, která bude přístupná pod doménou zákazníka pro uživatele z

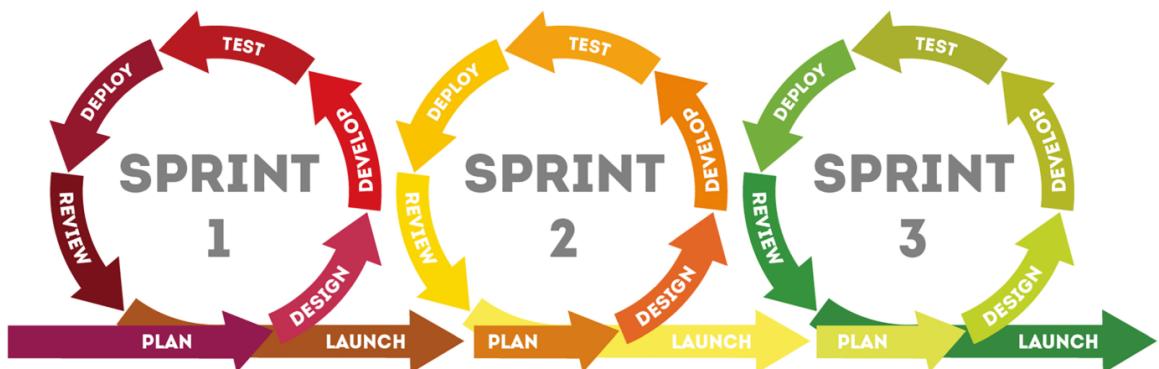
vnější sítě, kteří budou moci se stránkou interagovat a prohlížet si fotografie, které pan fotograf nahraje a uloží.

4.2 Maturitní část

Ačkoliv je tato část spuštěná na mém privátním virtuálním serveru, ani tak není zcela nezávislá na Firebase, jelikož využívá data z Firebase Firestore. To by se dalo vyřešit například spuštěním instance No-SQL databáze MongoDB, která funguje na podobném principu jako Firestore, a fotografie, které by uživatel této instance nahrál na server, by se jednoduše ukládaly do předem připraveného adresáře na lokálním úložišti. Nicméně pro jednodušší nastavování a z důvodu nutnosti přepracovat celý kód jak webu tak i CMS, jsem se rozhodl zůstat u Firebase co se dat týče. Téměř veškeré funkcionality, které obsahuje komerční instance projektu, obsahuje i tato maturitní část, jediná věc co chybí, je SSL certifikát a vlastní doména. I když by bylo poměrně jednoduché obojí přidat, domény jsou placené a já bohužel žádnou doménu zatím nevlastním.

5 Průběh vývoje

Vývoj obou systémů, který započal v dubnu roku 2021 jsem se snažil provádět co nejvíce agilně. To znamená, že se odehrával v takzvaných sprintech, kde jeden sprint je časový úsek pro implementaci a vyladění požadované změny. Nicméně častokrát se stalo, že jsme spolu se zákazníkem označili právě probíhající sprint za dokončený (já jsem implementoval změnu a zákazník ji označil za odpovídající), ale za pár dní mi přišel email s tím, že se zákazníkovi tato změna nakonec nelibí a chtěl by ji změnit. Nicméně i když se jedná o lehce kontraproduktivní chování, je to zcela běžné i během vývoje mnohem větších a komplexnějších projektů a musí se s tím počítat. Proto je dobré, že je celý projekt modulární a není až tak složité implementovat změnu.



Obr.11 - Princip fungování agilního vývoje, který se odehrává v takzvaných sprintech
(Zdroj: netmagnet.cz)

Vývoj obou systémů začal nejprve ve vanilla Javascriptu, bez použití jakéhokoliv frameworku, což se jevilo jako nejjednodušší řešení, jelikož jsem tehdy neuměl používat žádný framework. Čistý Javascript je pro projekt, jako je webová stránka více než dostačující, nicméně vzhledem k modulárnosti a mnoha dalším vylepšením, které framework jako například Gatsby.js přidává, bych jej v případě, že bych měl vytvořit webovou stránku, použil. Nicméně se změnami a složitějšími funkcionalitami, které jsem postupně přidával do administračního systému, se zdrojový kód začal jevit poměrně nepřehledným, a proto jsem se rozhodl, že jej refaktorují do Reactu, který nejen zvýší celkovou rychlosť a performance systému, ale také

přehlednost struktury souborů a kódu, což se nakonec vyplatilo, protože nyní mám jasný přehled o tom, kde co je a jakým způsobem to funguje.

5.1 Komunikace se zákazníkem

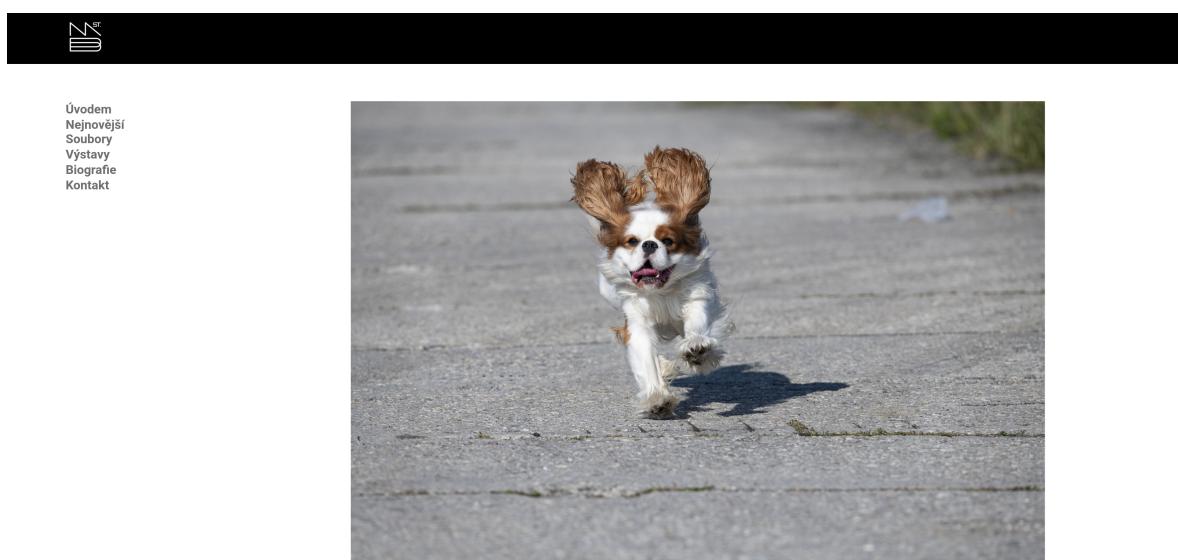
Komunikace se zákazníkem probíhala primárně a nejvíce prostřednictvím mailů, což se mi během vývoje velmi osvědčilo, jelikož všechny informace byly sepsány přehledně na jednom místě a bylo jasné, na čem jsme se domluvili. Proběhlo i několik telefonních hovorů, od kterých jsme ale nakonec pro jejich nepřehlednost upustili.

5.2 Organizace úkolů a změn k implementaci

Jakmile jsem dostal od zákazníka nějaký požadavek, ať už ve formě e-mailu nebo telefonátu, okamžitě jsem si jej zapsal a následně přidal do úkolníčku. Těch jsem během vývoje vyzkoušel několik, ale nejvíce mi osvědčil Todoist pro jeho přehlednost a jednoduché ovládaní. Dá se v něm jednoduše nastavit priorita daného úkolu a ty se dají rozdělit do projektu, což je velké plus a uživatel tak nemá na hlavní stránce změť úkolů, které například ani nesouvisejí s projektem, na kterém zrovna pracuje. Jediná nevýhoda, kterou Todoist má, je absence upozornění na dobu nastaveného zpracování úkolu.

6 Webová prezentace

Webová prezentace bude panu fotografovi sloužit jako virtuální galerie fotek, aby lidé, které jeho tvorba zajímá, měli všechny jeho fotografie na "dosah ruky" a vzhledem k tehdejší koronavirové situaci je více než vhodné, aby nemuseli chodit přímo na výstavu. Samotná webová prezentace obsahuje přesně to, co obsahovala ta stávající, tedy, hlavní stránku, na které je jen galerie z alba "Hlavní", jelikož se jedná o fotografie, které se panu fotografovi líbí nejvíce a o kterých si myslí, že by mohly oslovit nejvíce lidí. Po kliknutí na náhledovou fotografiu se spustí galerie, která avšak nedovoluje rozkliknout informační box s více informacemi o dané fotografii. Data, které web používá ke svému fungování nejsou dynamická, jelikož by to z principu fungování statické webové stránky nemělo smysl. Data se stahují z databáze ve chvíli, kdy uživatel přijde na stránku poprvé. To znamená, že pokud uživatel nahraje novou fotografiu, změny na webu se projeví až ve chvíli, kdy uživatel stránku znova načte.

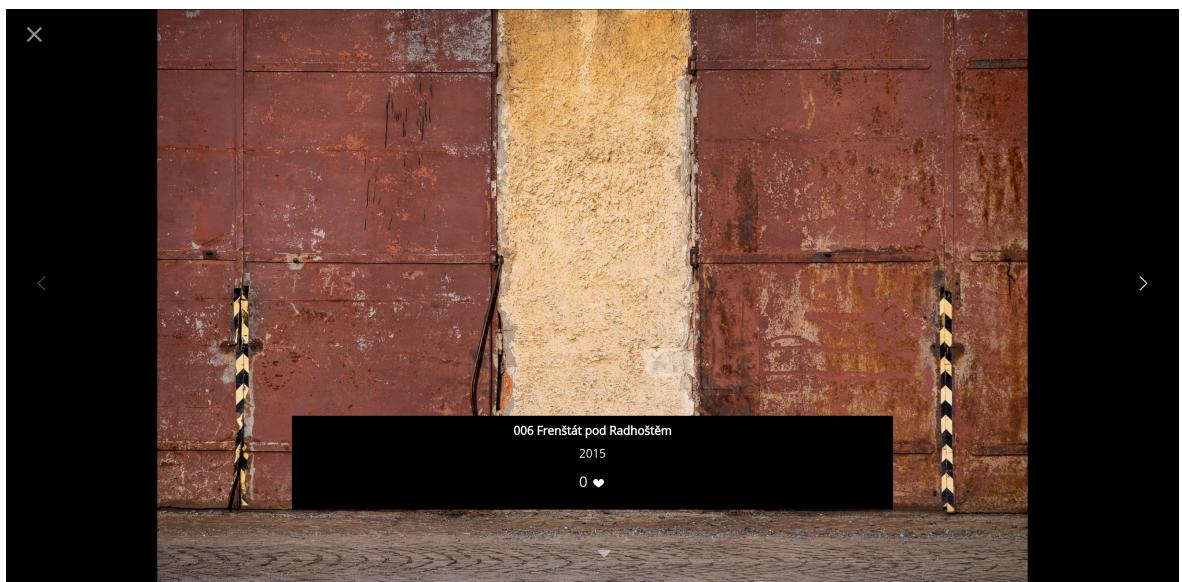


Obr.12 - Hlavní stránka webové prezentace
(Zdroj: Autor práce)

Dále stránku "Úvodem", sloužící jako předmluva pro celý web, který má simuloval umělecké dílo spolu s jeho uměleckými fotografiemi. Tento text je samozřejmě editovatelný v

administračním systému.

Odkaz na stránku alba "Nejnovější". Jak by se mohlo z názvu zdát, nejedná se o album s fotografiemi, které byly nahrány nedávno, ale o fotografie, které byly nedávno vyfoceny. Ostatní alba totiž mohou obsahovat i fotografie, které byly vyfoceny před rokem 2000. To znamená, že se jedná o výběr fotografií, které pan fotograf nedávno vyfotil a rozhodl se, že je bude sdílet. Při vývoji této podstránky jsem musel vymyslet systém, jakým budu fotografie na webu zobrazovat, jelikož si zákazník přál, aby místo galerie, která je použita u všech ostatních alb, byla použita tabulka s náhledy fotografií. Abych tohoto docílil se správnou *pagination*, musel jsem pole se všemi fotografiemi rozdělit na části po 6 fotografiích a poté je v těchto částech zobrazit na webu. Po kliknutí na náhled fotografie se však otevře galerie, jako u všech ostatních alb.



Obr.13 - Galerie po rozkliknutí náhledové fotografie

(Zdroj: Autor práce)

Dále stránku "Výstavy", která obsahuje seznam všech výstav, na kterých pan fotograf kdy vystavoval své fotografie. Samozřejmě se jedná o editovatelný text.

"Biografie" popisuje dosavadní život pana fotografa společně s jeho fotografií.

V poslední řadě "Kontakt", kde lze najít veškeré kontaktní možnosti. Formulář, který podstránka obsahuje, bude pomocí Firebase Functions posílat email a *push notifikaci* panu

fotografovi vždy, když mu někdo napíše email.

The screenshot shows a website layout. On the left, there's a vertical sidebar with a dark header containing a logo and navigation links: 'Úvodem', 'Nejnovější', 'Soubory', 'Výstavy', 'Biografie', and 'Kontakt'. Below this is a 'Kontakt' section with input fields for 'Vaše jméno a příjmení' (Jan Novák), 'Váš E-mail' (jan-novak@priklad.cz), and 'Vaše telefonní číslo' (+420 123 456 789). There's also a placeholder 'O co jde?' and a 'ODESLAT' button. To the right, a large text area says 'Odesláním souhlasíte se zpracováním Vašich osobních údajů.' At the bottom, there are four social media icons with corresponding contact information: a phone icon for '+420 123 456 789', an envelope icon for 'info@info.cz', an Instagram icon for '@milanbures', and a Facebook icon for 'Milan Bureš'.

2021 © Mgr. Milan Bureš st.
Made with ❤ by Orexin.

Obr.14 - Stránka s kontaktním formulářem

(Zdroj: Autor práce)

Jelikož je webová prezentace napsána ve vanilla Javascriptu, v budoucnu mám v plánu ji přepsat pomocí frameworku Gatsby.js, který slouží pro generování statických HTML stránek za použití React.js, což bude mít za následek nejen mnohonásobně jednodušší budoucí vývoj, ale také rychlejší načítací časy.

6.1 Vanilla Javascript routing

Jelikož jsou alba dynamickými prvky a v databázi se jejich počet mění, bylo nutné implementovat systém, který umožní, aby si uživatel mohl zobrazit všechny vytvořená alba. Nejhorším řešením, které by mohlo být, je pokaždé, když uživatel administračního systému vytvoří nové album, ručně vytvořit nový HTML dokument, který bude obsahovat galerii s obsahem daného alba. Implementoval jsem tedy *algoritmus*, který na základě *id* obsažené

v URL linku alba vytáhne z databáze správný záznam o albu právě pomocí daného *id*.

```
// This file manipulates with URL query params using which
// we show correct album

// Imports
import { async } from "regenerator-runtime";
import "../css/collections.css";
import { Collections, Images, Storage } from "./core";
import { Gallery } from "../components/gallery";

// Variables
let collectionId = new
URL(document.location).searchParams.get("collectionId");
let collectionObject = Storage.getSpecific(collectionId);
const galleryWrapper = document.querySelector("#gallery-wrapper");

document.querySelector(".collectionName").innerText = collectionObject.name;
galleryWrapper.appendChild(new Gallery(collectionObject.images));
```

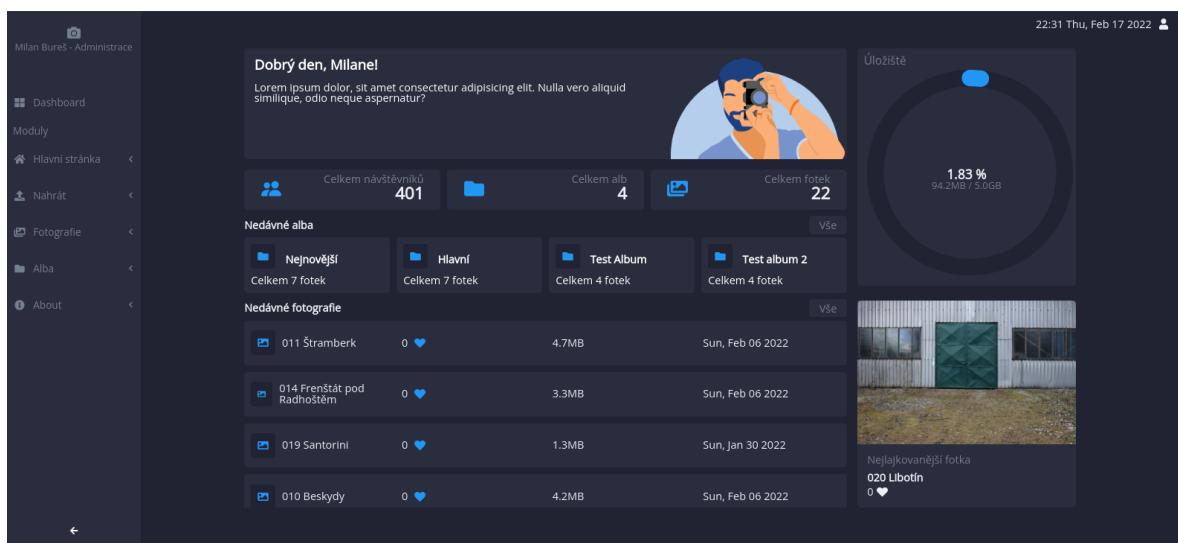
Obr.15 - Ukázka algoritmu získávání správného alba

(Zdroj: Autor práce)

Funguje to tak, že uživatel webové stránky klikne v menu na odkaz alba, který obsahuje *id* alba, na které chce uživatel přejít. Tyto data, název a odkaz na album, jsou stáhnuty z databáze ve chvíli, kdy uživatel přijde na stránku poprvé. Po kliknutí je přesměrován na podstránku *collections*, která podle *id* obsaženého v URL linku zjistí přesně které album chce uživatel vidět a jeho data vyrenderuje v podobě galerie do dokumentu. Ve své podstatě se jedná o podobný princip, jakým funguje React Router.

7 Administrační systém

Administrační systém je systém, který zprostředkovává manipulaci s daty, které se objevují na dané webové stránce. Jednoduše řečeno, díky administračnímu systému je uživatel schopen své fotografie nahrávat, mazat, upravovat jejich název a popis, stejně tak uvidí, která z nich je nejvíce oblíbená na základě lajků a podobně. Dále administrační systém obsahuje funkcionality pro manipulaci s alby, do kterých bude uživatel své fotografie rozdělovat. Ty samozřejmě může obdobně vytvářet, mazat a upravovat jejich fotografický obsah. V neposlední řadě systém obsahuje i informační nástěnku, kde je velmi dobře vidět například kolik fotografií je v danou chvíli nahraných na webu a v jakých albech, kolik úložného prostoru jeho fotografie zabírají a nebo třeba seznam posledních nahraných fotografií, aby je nemusel hledat.



Obr.16 - Ukázka nástěnky administračního systému

(Zdroj: Autor práce)

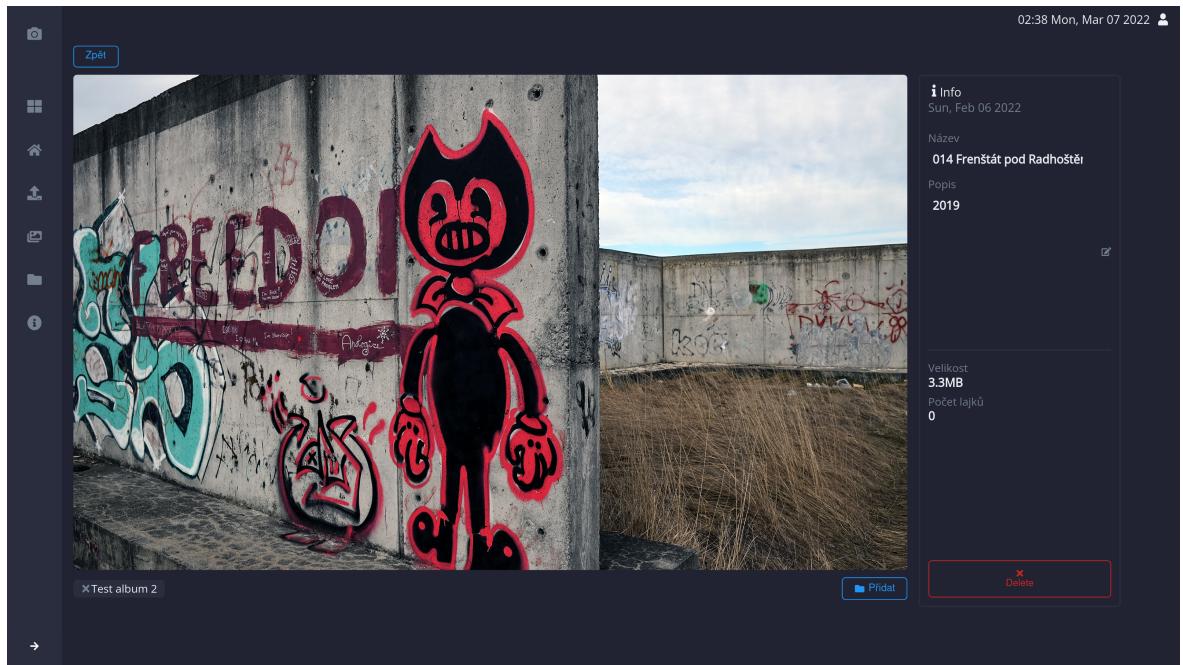
Všechny data, se kterými uživatel pracuje, jsou zobrazována v reálném čase, což znamená, že pokud, například upraví název fotografie, tato změna se ihned projeví jak v samotném administračním systému, tak v databázi a tím pádem i na webové stránce. Díky Firebase Auth se do systému nedostane uživatel, který není autentifikovaný, tedy přihlášený. Prozatím

celý autentifikační systém funguje za pomocí emailu a hesla, jelikož web spravuje jeden člověk a není potřeba, abych implementoval například autentifikaci pomocí Google či jiného již existujícího účtu. Celá aplikace je "obalená" v *auth elementu*, který po úspěšném přihlášení (ověří se, zda uživatel existuje na Firebase Auth serveru) uchovává informace o přihlášeném uživateli a pokud záznam o tomto uživateli existuje, *auth element* jej pustí dál, jinak bude přesměrován zpět na přihlašovací obrazovku. Administrační systém obsahuje veškeré nástroje k tomu, aby uživatel mohl manipulovat s fotografiemi a alby, které nahraje a vytvoří. Po nahrání jedné či více fotografií se tyto fotografie objeví na podstránce "Fotografie" v přehledném seznamu. Kliknutím na jednotlivé fotografie se uživatel dostane na stránku pro manipulaci s jednotlivou fotografií, kde ji může přejmenovat, přidat popis, přiřadit do alba nebo ji z alba odstranit, a nebo fotografií odstranit úplně. Také lze vidět metadata fotografie, jako například kdy byla nahrána, či případně kolikrát ji návštěvníci webové prezentace udělili "lajk".

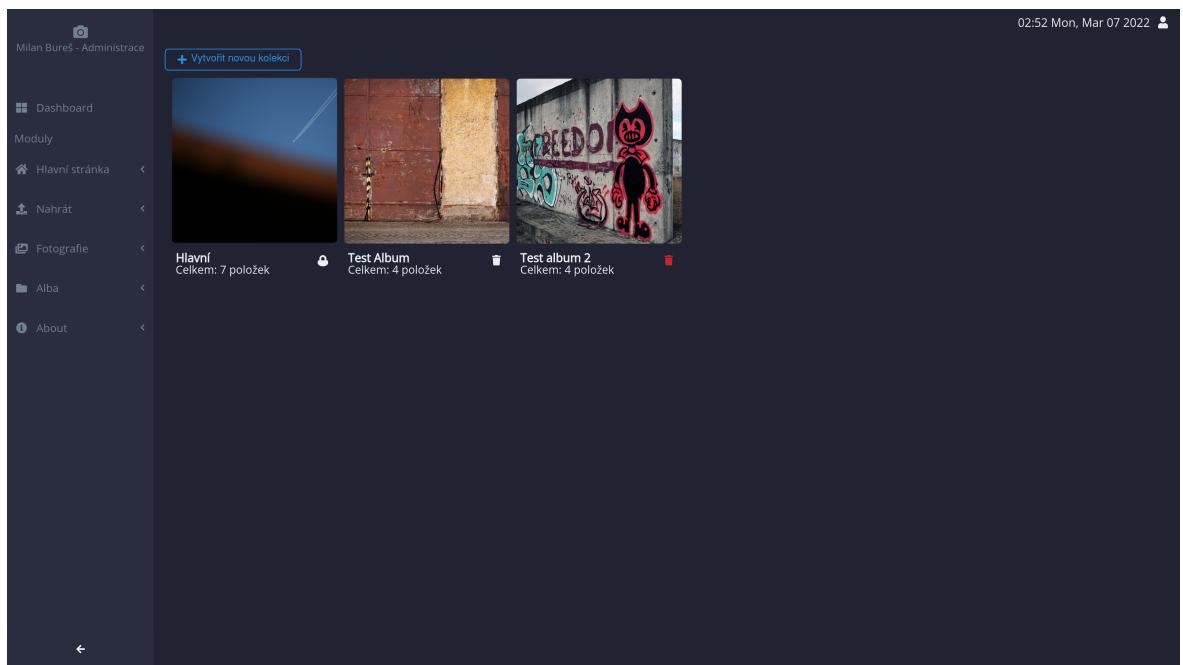
Všechny fotografie				
	Název	Počet lajků	Velikost	Datum nahrání
	011 Štramberk	0	4.7MB	Sun, Feb 06 2022
	014 Frenštát pod Radhoštěm	0	3.3MB	Sun, Feb 06 2022
	019 Santorini	0	1.3MB	Sun, Jan 30 2022
	010 Beskydy	0	4.2MB	Sun, Feb 06 2022
	022 Frenštát pod Radhoštěm	0	4.4MB	Tue, Feb 08 2022
	008 Věžkovice	0	6.4MB	Sun, Feb 06 2022
	002 Praha	0	7.0MB	Sun, Feb 06 2022
	001 Rhodos	0	3.0MB	Sun, Jan 30 2022
	003 Tichá	0	9.8MB	Wed, Jan 26 2022
	004 Lubina	0	5.7MB	Thu, Jan 27 2022

Obr.17 - Stránka se seznamem všech nahraných fotografií

(Zdroj: Autor práce)



Obr.18 - Stránka pro manipulaci s jednotlivou fotografií
(Zdroj: Autor práce)



Obr.19 - Stránka se všemi aktuálně vytvořenými alby
(Zdroj: Autor práce)

7.1 Využití frameworku React.js

Administrační systém byl v raných fázích vývoje vyvíjen jen za pomocí čistého Javascriptu a jelikož tento přístup postupem času a komplexnosti projektu začal snižovat přehlednost i celkovou jednoduchost vývoje, celá aplikace byla přepsána a refaktorována za pomocí frameworku React.js, ale například *algoritmy* pro ukládání nebo získávání dat z databáze se nezměnily.

Díky tomu se celkové načítací časy zmenšily, protože pokud například uživatel změnil *endpoint*, celá aplikace se nemusela načítat znovu, jelikož z principu fungování Reactu již načtená celá byla a tudíž to nebylo potřeba. Dále také načítací časy obrázků zaznamenaly pokles, protože React pracuje velmi dobře s rychle měnícími se daty a například stahování, *cacheování* a následné zobrazování obrázků umí rychleji, než uměla moje vanilla implementace.

7.1.1 React.js vs Vanilla Javascript

Není tajemstvím, že co se celkové *performance* týče, bude React vždy pomalejší, než čistý vanilla Javascript, už jen z toho důvodu, že je React svým způsobem nadstavbou nad Javascriptem a během *runtime* provádí mnohem více operací, než čistý Javascript. Nicméně co se tzv. *developer experience* týče, vyvíjení webových aplikací ale i webů je díky Reactu mnohem jednodušší, než jen za pomocí čistého Javascriptu, jelikož celý kód je rozdělen do takzvaných komponent, které fungují jako stavební kameny aplikace a jednoduše se do nich vkládají a vývojář má tak přesný a jasný přehled o struktuře projektu.

Dalším základním rozdílem je způsob, jakým React oproti projektu používající jen vanilla Javascript funguje. Projekt s vanilla Javascriptem je rozdělen na více *HTML* dokumentů (například podle počtu podstránek) a každý z nich má svůj vlastní Javascript soubor, který se spustí ve chvíli, kdy se daný *HTML* dokument načte. Nicméně React na druhou stranu má jen jeden *HTML* dokument, který obsahuje jeden Javascriptový soubor a podle toho, který *endpoint* uživatel vyžádá (navštíví) se obsah *HTML* dokumentu změní na danou komponentu typu *route*. Na tomto principu funguje React Router, díky kterému se obsah stránky nemusí načítat pokaždé, kdy uživatel danou stránku vyžádá, nýbrž se jednoduše "překreslí".

8 VPS - Virtual Private Server

Pro potřeby práce jsem si pronajal a nakonfiguroval vlastní server, který kopíruje veškeré funkcionality Firebase. Zajišťuje *hosting* pomocí softwarového webového serveru nginx, díky čemuž si na adrese VPS může zobrazit webovou stránku i modifikovat její obsah přes administrační systém. Kromě toho na VPS běží i Node.js *backend server*, díky kterému bude moct uživatel této instance projektu odesílat emaily panu fotografu bez použití email serveru třetí strany. De-facto se jedná o druhou instanci komerčního projektu pro pana fotografa.

8.1 VPS provideři

Na trhu je mnoho *cloud computing providerů*, nabízejících virtuální počítače v nejrůznějších početně výkonnostních kategoriích. Mezi nejznámější patří Digital Ocean, Linode, Hostinger nebo Amazon Cloud, kterému jsem se chtěl ale zámerně vyhnout, jelikož se mi vůbec nelíbí prostředí, ve kterém uživatel své VMs spravuje a jejich ceník je poměrně nepřehledný.

Provider	Cena/měsíc	Disk	RAM	Transfer (\$/GB)
Digital Ocean	\$5	25GB	1GB	\$0.01
Hostinger	\$3.95	20GB	1GB	Do 1TB/měsíc zdarma
Linode	\$5	25GB	1GB	Do 1TB/měsíc zdarma
A2	\$8.99	150GB	1GB	Do 2TB/měsíc zdarma

Tabulka popisující rozdíly mezi nabídkami jednotlivých *cloud computing providerů*
(Zdroj: Autor práce)

Já jsem se rozhodl pro DigitalOcean, jelikož s nimi mám již předešlou zkušenost a jejich prostředí pro správu virtuálních počítačů mi ze všech konkurenčních providerů přijde nejvíce in-

tuitivní a přehledné, spolu s ceníkem, který neobsahuje žádné lsti. Jejich cena není ze všech uvedených na listu nejlevnější, ale jedná se o kvalitní službu, která je této ceně adekvátní.

8.2 Konfigurace VPS

8.2.1 Operační systém GNU/Linux

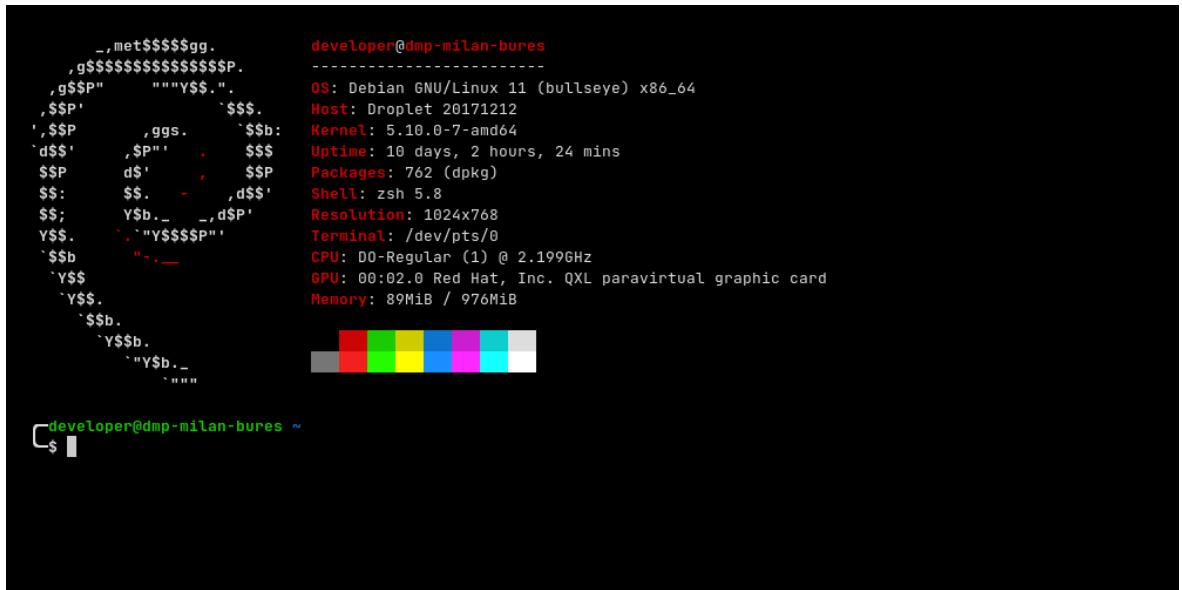
V dnešní době je jako serverový operační systém nejvíce rozšířený GNU/Linux pro jeho minimalističnost a poměrně jednoduchou konfiguraci.

Podle hostingtribunal.com až 96% serverů z 1 milionu celosvětově nejpoužívanějších serverů používá Linux jako svůj operační systém a proto jsem se i já rozhodl použít Linux a nejen z toho důvodu, že jej každodenně používám.

8.2.1.1 Debian 11

Původně jsem na svůj VPS chtěl nainstalovat Arch Linux, jelikož se jedná o Linuxovou distribuci, kterou každodenně používám, nicméně má jednu velkou nevýhodu - nemá vlastní instalátor (i když nyní již existuje instalační *script*), což mě osobně při instalaci v domácích lokálních podmínkách vůbec nevadilo, ale v případě instalování Arche na VPS by to znamenalo, že bych musel mít vlastnoručně přizpůsobené *ISO*, které bych použil k instalaci, jelikož DigitalOcean nenabízí Arch k nainstalování na jejich VPS, které nazývají Droplets. Volba tedy padla na Debian 11, jelikož je v jeho minimalistické instalaci poměrně "čistý" a tudíž neobsahuje zbytečné programy a podobně (bloatware). Vzhledem k tomu, že tento "počítač" bude sloužit jako server, není potřeba a je až zbytečné instalovat jakékoli desktopové prostředí *GUI*, jelikož by to mělo za následek zvýšení požadavků na výpočetní výkon serveru, což je nežádoucí, jelikož chceme, aby operační systém serveru zabíral co nejméně prostoru a prostředků, a zároveň celá konfigurace systému je proveditelná jen za pomoci příkazové řádky *CLI*. Na obrázku podle lze vidět, že *CLI-only* (bez *GUI*) Debian v idle stavu (stavu, kdy neprovádí

žádné operace) zabírá na paměti RAM jen 89MB z dostupných 1GB, tedy necelých 10%.



```
met$$$$$gg.      developer@dmp-milan-bures
$$$$$$$$$$$$$P.
,$$P"    ""Y$$.".
,$$P'          '$$.
'$,$P      ,ggs. '$$b: -----
OS: Debian GNU/Linux 11 (bullseye) x86_64
Host: Droplet 20171212
Kernel: 5.10.0-7-amd64
Uptime: 10 days, 2 hours, 24 mins
Packages: 762 (dpkg)
Shell: zsh 5.8
Resolution: 1024x768
Terminal: /dev/pts/0
CPU: D0-Regular (1) @ 2.199GHz
GPU: 00:02.0 Red Hat, Inc. QXL paravirtual graphic card
Memory: 89MiB / 976MiB

developer@dmp-milan-bures ~
```

Obr.20 - Diagnostický nástroj Neofetch zobrazující základní informace o systému po tom, co se na něj připojíme pomocí protokolu SSH
(Zdroj: Autor práce)

8.2.2 Vytvoření developer uživatele

Tento uživatel bude využíván jako správce celého vývojového i produkčního prostředí a bude mít skoro stejná práva jako root (administrátor systému).

```
useradd -m developer
passwd developer
```

Aby developer mohl naplno využívat systém, je potřeba, aby mohl používat sudo , které bylo díky Digital Ocean předinstalováno. Přidáme jej tedy do sudoers skupiny.

```
usermod -aG sudo developer
```

A ověříme, že vše správně dopadlo:

```
su developer
sudo -l
[sudo] :
User developer may run the following commands on dmp-milan-bures:
(ALL : ALL) ALL
```

8.2.3 Konfigurace prostředí

8.2.3.1 Shell

Vzhledem k tomu, že během práce s VPS používám git, který funguje na takzvaných větvích, je dobré vědět, se kterou větví právě pracuji. Proto nainstaluji jiný *shell*, který mi dovolí si jednoduše upravit příkazovou řádku.

```
sudo apt-get install zsh
```

Zsh neboli Z-shell je mocný příkazový *shell* který odemyká mnoho možností ať už po stránce přizpůsobení vzhedu ale také vylepšení funkcí. V kombinaci s *open-source* projektem oh-my-zsh jsem schopen změnit nejen vzhled celé příkazové řádky, ale také přidat *plugins* jako například git, který mi, pokud se v aktuálním adresáři nachází .git soubor(soubor, který inicializuje složku jako git repozitář), zobrazí aktuálně zvolenou větev. Oh-my-zsh nainstaluji pomocí nástroje wget přímo z oh-my-zsh repozitáře na GitHubu, který poté spustí instalaci *script*.

```
wget https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh
sh install.sh
```

8.2.3.2 Git a verzovací systém

Git je *open-source* nástroj pro správu verzí zdrojového kódu anebo jeho součástí vytvořený Linusem Torvaldsem. Jednoduše řečeno jedná se o software, který sleduje změny v souborech, aby pak měl uživatel/vývojář větší přehled o tom, jaké změny provedl. V současnosti

se git nejvíce používá pro kolaboraci mezi vývojáři na *open-source* ale i *closed-source* projekty, jelikož git samotný je *open-source*, tudíž jej může kdokoliv využívat na svém interním serveru. Vzhledem k tomu, že z *VPS* nebudu pushovat žádné změny do remote repositáře, nebylo potřeba provádět žádnou pokročilejší konfiguraci a git stačí jen nainstalovat.

```
sudo apt-get install git
```

Jediné, co jsem nakonfiguroval musel je zachování přihlašovacích údajů tak, aby se git nemusel pokaždé při každém *pullu* dotazovat na přihlašovací údaje k *remote repository* serveru, v méém případě ke Githubu.

```
git config --global credential.helper store
```

8.2.3.3 Node.js a NPM

Javascript samotný je spustitelný jen v prohlížeči, kde většinou plní funkci logického "mozku" webové stránky a provádí logické operace. Díky tomu avšak nemá moc jiného využití, což Node.js mění. Node.js je Javascript *runtime*, díky kterému můžeme spustit Javascriptový kód i mimo prohlížeč, jako například v příkazové řádce, podobně jako ostatní scriptovací ale i programovací jazyky, jako například Python nebo Java.

Umožňuje nám vyvíjet API servery pro webové stránky i webové aplikace. Jelikož je webová prezentace i administrační systém *serverless*, není sám o sobě Node.js potřeba pro správnou funkčnost systému, ale vzhledem k faktu, že používám third-party knihovny a pluginy jako například webpack, musím Node.js nainstalovat. Tyto knihovny a další podpůrné programy lze nainstalovat pomocí takzvaného Node Package Manageru (Správce balíčků pro Node.js), neboli NPM.

```
sudo apt-get install nodejs npm
```

Zjistíme verzi Node.js:

```
node -v  
[node]: v12.22.5
```

A jelikož verze 12 není nejnovější, musím aktualizovat na nejnovější stable verzi, jinak by se totiž mohlo stát, že nějaká z *dependencies*, které v projektu používám, nebude se verzí 12 spolupracovat.

```
sudo npm i -g node  
sudo npm i -g npm
```

A opět zkontrolujeme verzi Node.js:

```
node -v  
[node]: v16.14.0
```

8.2.3.4 Nginx

Nginx je *open-source* softwarový webový server a *load balancer*, který se zaměřuje hlavně na co nejmenší konsumpci paměti a výkonu a skvěle zvládá velký nápor připojených zařízení. Nainstalujeme jej z Debian repozitářů.

```
sudo apt-get install nginx
```

A spustím pomocí systemctl.

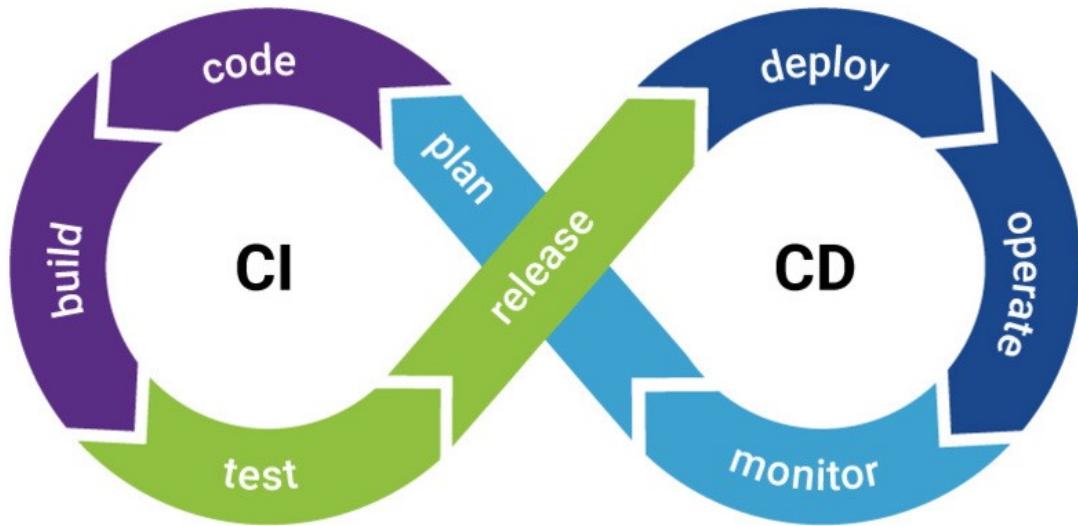
```
sudo systemctl start nginx
```

Nyní po úspěšném zapnutí nginx se VPS chová jako webový server a pokud v prohlížeči přejdu na adresu <http://159.65.112.226/> (IP adresa VPS), zobrazí se přivítací obrazovka nginx, což indikuje, že je všechno správně nakonfigurováno a funkční.

8.3 Konfigurace CI/CD

CI (Continuous Integration) a *CD* (Continuous Delivery) je proces, který umožňuje ne-přetržité dodávání aktuální verze projektu k testování či produkčnímu spuštění. V praxi to znamená, že pokud tým vývojářů provede změnu a uloží ji do *remote repozitáře* (místo pro ukládání zdrojových kódů v *cloudu*), tato změna se automaticky projeví (*deployne*) na produkční server (server, na kterém je spuštěna aktuální verze projektu a je poskytována uživatelům z venčí). Poté, co provedu změnu v zdrojovém kódu webové prezentace nebo adminis-

tračního systému a tuto změnu nahraji pomocí verzovacího systému git do Github *repositoria*, je potřeba, aby se tato změna projevila na produkčním serveru. K tomu využijí *shell scripty* a Github actions *hooky*.

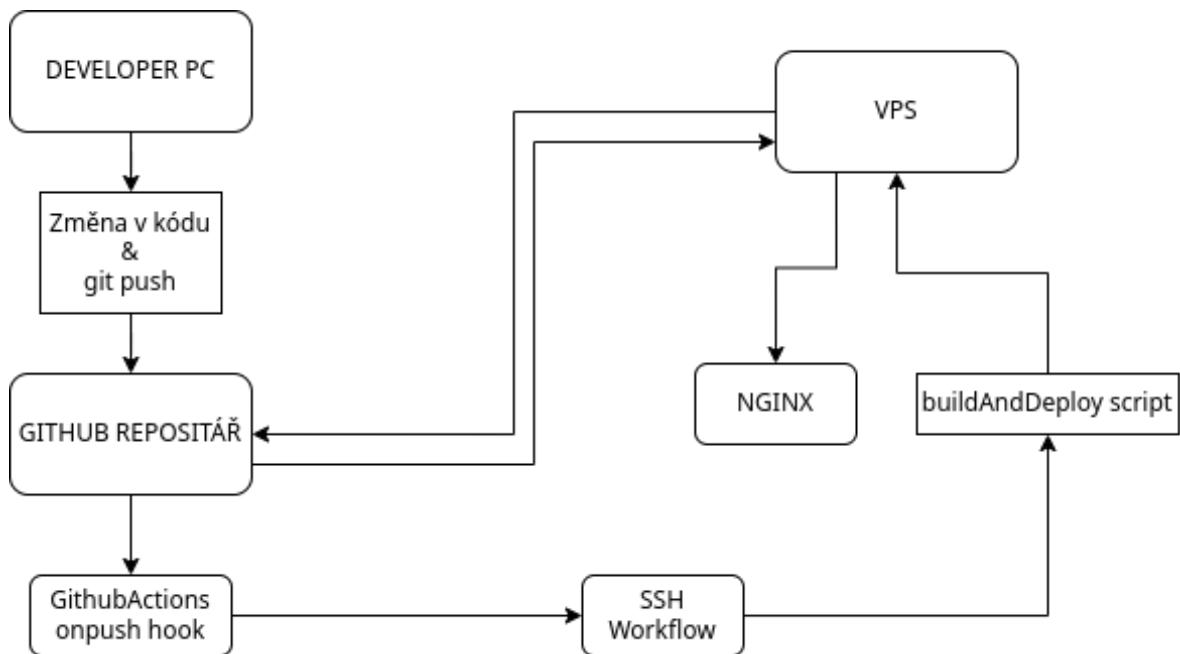


Obr.21 - Diagram znázorňující princip fungování CI/CD

(Zdroj: synopsys.com)

Celá operace probíhá tak, že jakmile vývojář (v tomto případě já) provede změny ve zdrojovém kódu a *pushne* je do *remote repositoria*, Github Action detekují *push* a spustí interní *script* napsaný v jazyce *YAML*, který se pomocí protokolu *SSH* připojí s předem připravenými přihlašovacími údaji na VPS, kde spustí *buildAndDeploy script*. Ten pomocí gitu *pullne* změny na lokální úložiště a spustí *build scripty*, což znamená, že *bundler* překompiluje zdrojové soubory projektu tak, aby byly optimalizované pro použití v produkčním prostředí a aby

je dokázal softwarový webový server nabízet uživatelům z vnější sítě.



Obr.22 - Diagram zobrazující princip stahování poslední změny na lokální úložiště VPS

(Zdroj: Autor práce)

8.3.1 Naklonování projektu na lokální úložiště

Jelikož není nejlepším řešením mít inicializovaný lokální klon git *repositoria* přímo ve složce, ze které nabízím webové stránky do vnější sítě, vytvořím složku, ve které se bude klon *repositoria* nacházet a projekt do ní naklonuji.

```
mkdir ~/files  
cd files  
git clone https://github.com/[repository].git
```

Nyní mám na systému přesnou kopii projektu a proto vytvořím *symlink*, který uložím do složky, kterou nginx defaultně používá pro soubory nabízených webových stránek.

```
ln -s ~/files/dmp-bures/website/dist /var/www/dmpbures.cz  
ln -s ~/files/dmp-bures/dashboard/build /var/www/admin.dmpbures.cz
```

8.3.2 Nginx konfigurace pro web i administrační systém

Aby byl web i administrační systém přístupný z vnější sítě, je pro každý z těchto "webů" nutné vytvořit vlastní konfigurační soubor aby nginx věděl, kde se nacházejí jednotlivé zdrojové soubory a pod jakou adresou mají být přístupné. Pro web i administrační systém vytvořím konfigurační soubor v adresáři "/etc/nginx/sites-available".

/etc/nginx/sites-available/dmpbures.cz

```
server {  
    listen 80;  
    listen [::]:80;  
    root /var/www/dmpbures.cz;  
    index index.html;  
    server_name dmpbures.cz www.dmpbures.cz;  
    location / {  
        try_files $uri $uri/ =404;  
    }  
}
```

/etc/nginx/sites-available/admin.dmpbures.cz

```
server {  
    listen 81;  
    listen [::]:81;  
    root /var/www/admin.dmpbures.cz;  
    index index.html;  
    server_name admin.dmpbures.cz www.admin.dmpbures.cz;  
    location / {  
        try_files $uri $uri/ =404;  
    }  
}
```

Struktura konfiguračních souborů je poměrně jednoduchá a jasná:

1. **listen** - Na jakém portu web poběží. Všimněte si prosím, že se porty webu a CMS liší, aby se navzájem nekryly a uživatel tak mohl přistoupit k oboum webům.
2. **root** - Složka ve které se nachází zdrojové soubory dané stránky
3. **index** - Soubor který slouží jako vstupní *HTML* dokument (zobrazí se jako první)

4. **server name** - Název serveru, pod jakým je web přístupný. V lokální síti by to bylo pod specifikovaným názvem, ve vnější v případě absence DNS záznamu pod IP VPS.

5. **location** - Udává lokaci dalších souborů, jako například 404 stránky (která se zobrazuje, pokud vyhledávaná stránka neexistuje)

Ve složce `/etc/nginx/sites-available` se však nachází jen konfigurační soubory webů, které jsou podle názvu složky k dispozici, nicméně nejsou aktivně nabízeny uživatelům z vnější sítě. Abych toho dosáhl, musím vytvořit kopii konfiguračních souborů ve složce `/etc/nginx/sites-enabled`, například pomocí symlinků.

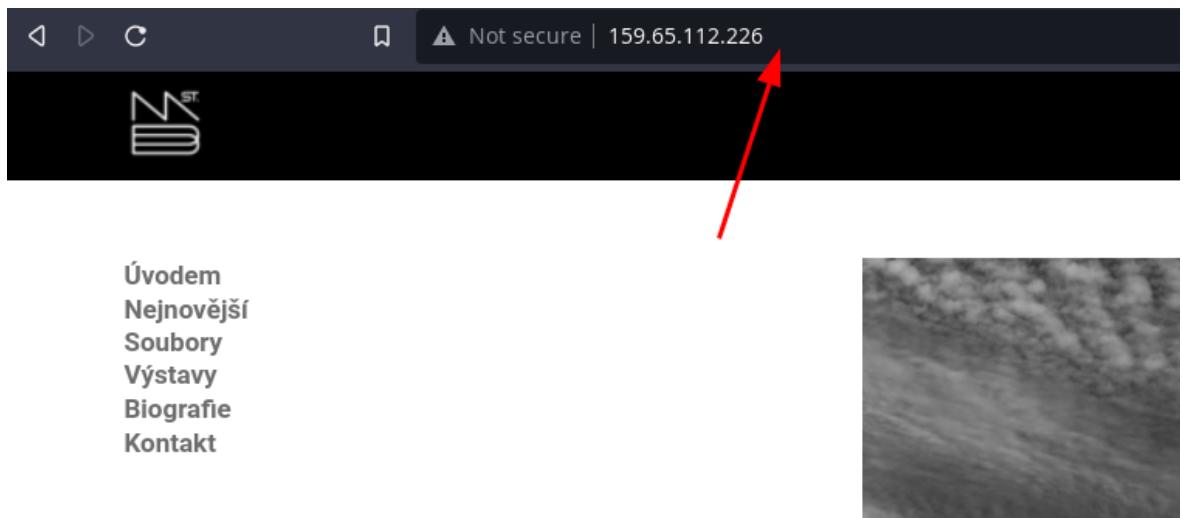
```
ln -s /etc/nginx/sites-available/dmpbures.cz  
/etc/nginx/sites-enabled/  
ln -s /etc/nginx/sites-available/admin.dmpbures.cz  
/etc/nginx/sites-enabled/
```

Nyní mám nakonfigurovaný nginx i weby, které má nabízet uživatelům z vnější sítě. Aby vše začalo fungovat, nginx restartuji a ověřím konfiguraci.

```
nginx -t  
sudo systemctl restart nginx  
systemctl status nginx
```

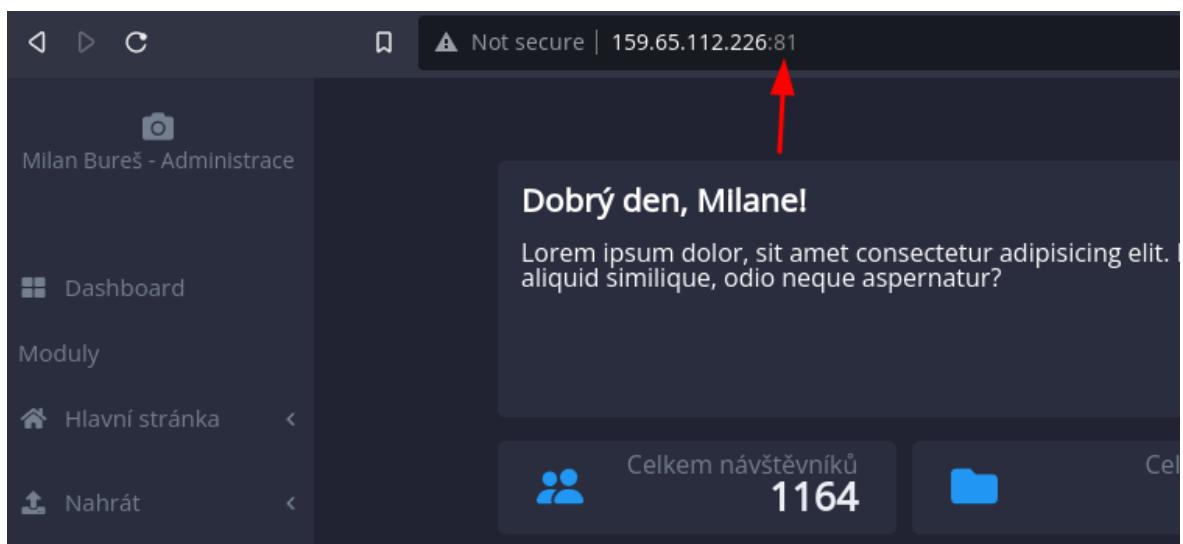
Po spuštění prohlížeče a vyhledání adresy `http://159.65.112.226` provede prohlížeč požadavek na můj nyní nakonfigurovaný webový server, kde jej nginx zachytí, přečte si adresu, kterou prohlížeč požaduje a pošle zpět odpovídající HTML soubor(y), v tomto případě, jelikož je defaultní port 80, webovou prezentací. Kdyby uživatel vyhledal adresu `http://159.65.112.226:81`, webový server by odeslal zpět soubory administračního systému a uživatel by tak mohl edi-

tovat obsah.



Obr.23 - Visualizace adresy, na které běží web díky VPS

(Zdroj: Autor práce)



Obr.24 - Visualizace adresy, na které běží administrační systém díky VPS

(Zdroj: Autor práce)

Závěr

Během vývoje tohoto projektu jsem se naučil mnoho nových věcí, které věřím, že mě v mé profesionálním životě posunuly dál, a jsem rád, že jsem si jako dlouhodobou matuřitní práci vybral právě tento projekt. Mrzí mne však, že jsem začal pozdě s vývojem pomocí frameworku React.js, jelikož bych měl vývoj snazší a neztratil bych čas vyvíjením administračního systému ve vanilla Javascriptu a tím pádem bych se mohl soustředit na důležitější věci, než je opětovné vyvíjení systému, který jsem již jednou vytvořil.

Summary

During the development process of this project, I have learnt a lots of new things, which I firmly believe, have leveled up my professional skills and I am very glad for this opportunity to have this kind of project as my maturita project. However, I am a little bit sad, that I did not start using React.js sooner, because the development would have been a lot easier and I would not have lost time developing content management system using vanilla Javascript and thus I would have more time for a lot more important things than creating a system, that I have already created.

9 Seznam obrázků

Obr.1 - Původní webová stránka	12
Obr.2 - Mnou vytvořená webová stránka	13
Obr.3 - Graf škálovatelnosti	14
Obr.4 - Ukázka React.js komponenty	16
Obr.5 - Ukázka stylování v CSS	17
Obr.6 - Ukázka stylování v SCSS	17
Obr.7 - Firebase Console nástěnka	18
Obr.8 - Firebse Firestore	19
Obr.9 - IsImage funkce	20
Obr.10 - Diagram struktury projektu	21
Obr.11 - Princip agilního vývoje	23
Obr.12 - Hlavní stránka webové prezentace	25
Obr.13 - Galerie	26
Obr.14 - Kontakt	27
Obr.15 - Vanilla Javascript routing	28
Obr.16 - CMS nástěnka	29
Obr.17 - CMS všechny nahrané fotografie	30
Obr.18 - CMS manipulace s daty fotografie	31
Obr.19 - CMS všechny alba	31
Obr.20 - VPS SSH přivítací obrazovka	36
Obr.21 - Princip fungování CI/CD	40
Obr.22 - Princip fungování CI/CD na VPS	41
Obr.23 - IP adresa webu na portu 80	44
Obr.24 - IP adresa CMS na portu 81	44

10 Seznam tabulek

01 - Tabulka popisující rozdíl mezi cloud computing providery 34

11 Seznam použitého softwaru

1. **Microsoft Visual Studio Code** - IDE použité k vývoji celého projektu.
2. **LaTeX** - Markup jazyk pro psaní prací ve velké typografické kvalitě pomocí profesionály předdefinovaných typografických stylů.
3. **Gimp** - Open-source rastrový editor pro editaci grafických objektů.
4. **Typora** - Textový editor pro textové soubory typu Markdown pro jednodušší práci s nimi.
5. **Git** - Verzovací systém.
6. **Github** - Server pro ukládání remote git repozitářů.

12 Seznam použité literatury

1. YAML – Wikipedie. [online].
Dostupné z: <https://cs.wikipedia.org/wiki/YAML>
2. URL - Wikipedia. [online]. Dostupné z: <https://en.wikipedia.org/wiki/URL>
3. Secure Sockets Layer – Wikipedie. [online].
Dostupné z: https://cs.wikipedia.org/wiki/Secure_Sockets_Layer
4. What is the Document Object Model?. World Wide Web Consortium (W3C) [online].
Dostupné z: <https://www.w3.org/TR/WD-DOM/introduction.html>
5. Symbolický odkaz – Wikipedie. [online].
Dostupné z: https://cs.wikipedia.org/wiki/Symbolický_odkaz
6. Single-page application - Wikipedia. [online].
Dostupné z: https://en.wikipedia.org/wiki/Single-page_application
7. Běhové prosředí – Wikipedie. [online].
Dostupné z: https://cs.wikipedia.org/wiki/Běhové_prosředí
8. Softwarový repozitář – Wikipedie. [online].
Dostupné z: https://cs.wikipedia.org/wiki/Softwarový_repozitář
9. Preprocesor – Wikipedie. [online].
Dostupné z: <https://cs.wikipedia.org/wiki/Preprocesor>
10. Vyvažování zátěže – Wikipedie. [online].
Dostupné z: https://cs.wikipedia.org/wiki/Vyvažování_zátěže
11. DevOps – Wikipedie. [online].
Dostupné z: <https://cs.wikipedia.org/wiki/DevOps>
12. Nasazení softwaru – Wikipedie. [online].
Dostupné z: https://cs.wikipedia.org/wiki/Nasazení_softwaru

13. Commit (verzování) – Wikipedie. [online].

Dostupné z: [https://cs.wikipedia.org/wiki/Commit_\(verzování](https://cs.wikipedia.org/wiki/Commit_(verzování)

14. CI/CD Pipeline – Wikipedie. [online].

Dostupné z: https://cs.wikipedia.org/wiki/CI/CD_Pipeline

15. Cache – Wikipedie. [online].

Dostupné z: <https://cs.wikipedia.org/wiki/Cache>

13 Citace

1. What Is CI/CD and How Does It Work? | Synopsys. Synopsys | EDA Tools, Semiconductor IP and Application Security Solutions [online]. Copyright ©2022 Synopsys, Inc. All Rights Reserved [cit. 22.04.2022].

Dostupné z: <https://www.synopsys.com/glossary/what-is-cicd.html>

2. 3 události v Net Magnetu a 3 rady pro podnikatele | Net Magnet. Kreativní internetová agentura Net Magnet - weby & marketing | Net Magnet [online]. Copyright ©2022 [cit. 22.04.2022].

Dostupné z: <https://www.netmagnet.cz/blog/3-udalosti-3-rady-2019/>