Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Vývoj webové platformy pro přípravu dat vhodných ke strojovému učení

Martin Choutka

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Martin Choutka

Informatika

Název práce

Vývoj webové platformy pro přípravu dat vhodných ke strojovému učení

Název anglicky

Development of a web platform for the preparation of data suitable for machine learning

---

Cíle práce

Hlavním cílem práce je vytvořit webovou platformu pro přípravu dat vhodných ke strojovému učení na základě označování dat, dle zvolených metodik a technologií.

Vedlejším cílem je zhodnotit tyto metodiky, technologie a shrnout průběh vývoje.

Metodika

Na základě analýzy odborné literatury týkající se vývoje softwaru, programovacích jazyků a technologií s nimi spojených bude provedena syntéza získaných informací. Tyto informace poslouží ke zpracování teoretického východiska.

Praktická část práce spočívá ve využití nastudované metodiky vývoje softwaru. Dle vybrané a vhodné metodiky provedeme analýzu požadavků, které by měla platforma splňovat. Po analýze vybereme vhodné technologie, pomocí kterých bychom měli být schopni platformu vyvinout. Vývoj bude probíhat dle metodiky. Platforma bude postupně implementována.

Po implementaci bude následovat verifikace, platforma bude otestována a zpřístupněna široké veřejnosti, od které získaná odezva projekt zhodnotí.

Pomocí logické indukce vyvodíme obecný závěr ze získaných poznatků při vývoji platformy, navrhneme, jakým směrem by se mohla na základě uživatelské odezvy ubírat a popíšeme, kdo by ji mohl v budoucnu využívat.

Doporučený rozsah práce

40-50

Klíčová slova

JavaScript, MongoDB, Next.js, webová aplikace, Node.js, data

Doporučené zdroje informací

ATTARDI, Joe. Modern CSS: Master the Key Concepts of CSS for Modern Web Development. 1. New York, New York, USA: Apress, 2020. ISBN 978-1-4842-6293-1.

BIERER, Doug. MongoDB 4 Quick Start Guide. 1. Birmingham, UK: Packt Publishing, 2018. ISBN 978-1-78934-353-3.

COULSON, Lewis, Brett JEPHSON a Marian ZBURLEA. The HTML and CSS Workshop. 1. Birmingham, UK: Packt Publishing, 2019. ISBN 978-1838824532.

HORSTMANN, Cay. Modern JavaScript for the Impatient. 1. Boston, Massachusetts, USA: Addison-Wesley Professional, 2020. ISBN 0-13-650214-8.

SANTACROCE, Ferdinando. Git Essentials – Second Edition. 1. Birmingham, UK: Packt Publishing, 2017. ISBN 9781787120723.

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

Ing. Tomáš Vokoun

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 17. 8. 2021

doc. Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 5. 10. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 13. 03. 2022

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Vývoj webové platformy pro přípravu dat vhodných ke strojovému učení" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15. 3. 2022 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Tomáši Vokounovi za odborné vedení práce, jeho vstřícné rady a cenné poznámky.

Vývoj webové platformy pro přípravu dat vhodných ke strojovému učení

Abstrakt

Práce se zaměřuje na tréninková data a tvorbu platformy, která přípravu těchto dat umožňuje. Teoretická část pojednává o základních procesech vývoje softwaru, které jsou v praxi používány, a které praktická část využívá. Práce v teoretické části popisuje jednotlivé technologie, které se používají na straně klienta, či na straně serveru. Popsána jsou také řešení, která umožňují verzování, či způsoby, kterými lze vyvinuté aplikace provozovat. V neposlední řadě rozebírá tréninková data, způsoby označování či jaká jsou úskalí těchto dat. Následně jsou analyzovány platformy, které označování dat umožňují.

V praktické části je provedena analýza dostupných technologií, kdy byl jako programovací jazyk zvolen JavaScript, který se používá na straně klienta, tak na straně serveru, a zaštiťuje tak celou aplikaci. Analýza byla také provedena pro frontend knihovny, služby umožňující hosting či pro ukládání dat. V praktické části jsou tyto technologie využity k vytvoření platformy, umožňující podobné funkcionality, jako platformy popsané v části teoretické. Samotná platforma byla úspěšně nasazena, a v neposlední řadě dochází k analýze vzniklé platformy s těmi již existujícími.

Klíčová slova: JavaScript, MongoDB, Next.js, webová aplikace, Node.js, data, strojové učení, AWS, serverless, tréninková data

Development of a web platform for the preparation of data suitable for machine learning

Abstract

The bachelor thesis focuses on training data and the creation of a platform that allows the preparation of this data. The theoretical part deals with the software development processes that are used in practice and which the practical part uses. Describes the various technologies, systems that allow versioning or hosting providers. Finally, it analyses training data, methods of data labelling and its pitfalls. Finally, the platforms that allow data labelling are compared.

In the practical part, an analysis of available technologies is performed, where JavaScript is chosen as the programming language, which is used on the client-side and the server-side. The analysis was also performed for frontend libraries, hosting services or data storage services. In the practical part, these technologies are used to create a platform that allows similar functionality as the platforms described in the theoretical part. The platform itself has been successfully deployed. The resulting platform is analysed with those that already exist.

Keywords: JavaScript, MongoDB, Next.js, web application, Node.js, data, machine learning, AWS, serverless, training data

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

HTML	HyperText Markup Language
XHTML	Extensible Hypertext Markup Language
W3C	World Wide Web Consortium
WHATWG	Web Hypertext Application Technology Working Group
CSS	Cascading Style Sheets
DOM	Document Object Model
API	Application Programming Interface
SASS	Syntactically Awesome Style Sheets
AMD	Asynchronous Module Definition
NPM	Node.js Package Manager
SEO	Search Engine Optimization
MVC	Model-View-Controller
CDN	Content Delivery Network
SPA	Single Page Application
SQL	Structured Query Language
AWS	Amazon Web Services
PHP	Hypertext Preprocessor
VPS	Virtual Private Server
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
JSON	JavaScript Object Notation
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
UI	User Interface
SVG	Scalable Vector Graphics
JWT	JSON Web Token
SDK	Software Development Kit
CORS	Cross-Origin Resource Sharing
SMTP	Simple Mail Transfer Protocol
SES	Simple Email Service
S3	Simple Storage Service

1 Úvod

Automatizace je nezaměnitelnou součástí našeho života. Setkáváme se s ní na každém kroku. V bankovnictví, kde probíhá automatické skenování dat z karet, ve výrobě, kde se automatizace používá pro detekci vadných výrobků až po automobilový průmysl, ve kterém dochází za posledních pár let k neuvěřitelnému rozmachu. Je používána z jednoduchých důvodů – dokáže zefektivnit či zlevnit proces. Firem, které na automatizaci spoléhají, je celá řada. Od výrobce autonomních automobilů Tesla, až po výrobce grafických čipů NVIDIA. Tyto společnosti využívají modelu strojového učení.

Strojové učení je proces, při kterém se využívá matematických modelů dat, pomocí kterých se počítač učí bez instrukcí uživatele. Ten by měl v budoucnu rozeznávat situace, které se dříve naučil, a ty které jsou těmto situacím podobné. Tento proces ale vyžaduje obrovské množství dat. Tato data musí být specializovaným nástrojem anotována dle přání společnosti, aby mohla být pro strojové učení využita. Ve zjednodušeném kontextu jde o označování objektů na fotografiích či videích. Objekty lze označovat spoustou způsobů. Může se jednat o jednoduché obdélníky, mnohoúhelníky, body či přímky. Tyto objekty musejí být převedeny do takových dat, kterým porozumí počítač samotný. Výstupem jsou proto data, obsahující jednotlivé objekty, s přesnými souřadnicemi.

Open-source projektů, řešících tuto tématiku není mnoho, rozhodl jsem se proto problematiku vývoje shrnout, nástroj vyvinout, zpřístupnit jako open-source a spustit pro širokou veřejnost.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem práce je vytvořit webovou platformu pro přípravu dat vhodných ke strojovému učení na základě označování dat, dle zvolených metodik a technologií. Vedlejším cílem je popsat tyto metodiky, technologie a shrnout průběh vývoje.

2.2 Metodika

Na základě analýzy odborné literatury týkající se vývoje softwaru, programovacích jazyků a technologií s nimi spojených bude provedena syntéza získaných informací. Tyto informace poslouží ke zpracování teoretického východiska.

Praktická část práce spočívá ve využití nastudované metodiky vývoje softwaru. Dle vybrané a vhodné metodiky provedeme analýzu požadavků, které by měla platforma splňovat. Po analýze vybereme vhodné technologie, pomocí kterých bychom měli být schopni platformu vyvinout. Vývoj bude probíhat dle metodiky. Platforma bude postupně implementována.

Po implementaci bude následovat verifikace, platforma bude otestována a zpřístupněna široké veřejnosti, od které získaná odezva projekt zhodnotí.

Pomocí logické indukce vyvodíme obecný závěr ze získaných poznatků při vývoji platformy, navrhneme, jakým směrem by se mohla na základě uživatelské odezvy ubírat a popíšeme, kdo by ji mohl v budoucnu využívat.

3 Teoretická východiska

3.1 Proces vývoje softwaru

3.1.1 Model vodopádu

Model vodopádu znázorňuje posloupnost na sebe navazujících jednotlivých etap. Jeho dodržování v praxi je velmi složité, a proto je jeho význam spíše teoretický. (1) Jednotlivé etapy jsou:

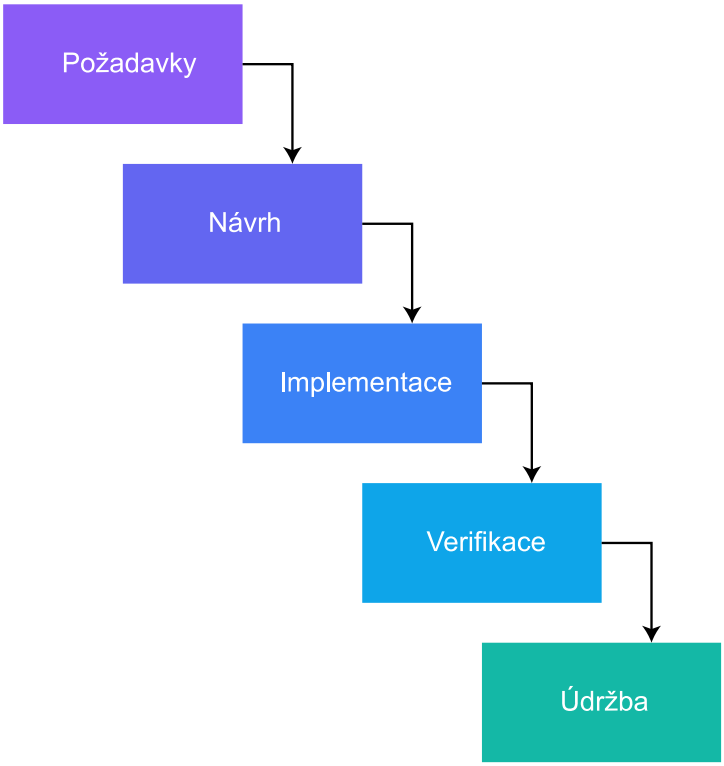
- Požadavky
- Návrh
- Implementace
- Verifikace
- Údržba

Výhody

- Jednoduchý
- Při stálých požadavcích nejlepší struktura výsledného produktu

Nevýhody

- Zákazník musí stanovat všechny své požadavky předem
- Při změnách požadavků má model dlouhou dobu realizace
- Zákazník vidí spustitelnou verzi až v moment spuštění



Obrázek 1: Vodopádový model (2)

3.1.2 Spirálový model

Spirálový model velmi dobře pokrývá nedostatky vodopádového modelu. Patří do skupiny přístupů řízených riziky. To znamená, že postup do další fáze záleží na důsledně provedené analýze a hodnocení. Je založen na iterativním přístupu, zavádí tedy opakovanou analýzu všech rizik, je proto vhodný pro větší projekty. Definoval ho poprvé Barry Boehm v roce 1986. (1) Jeho cyklus je rozdělen do čtyř částí:

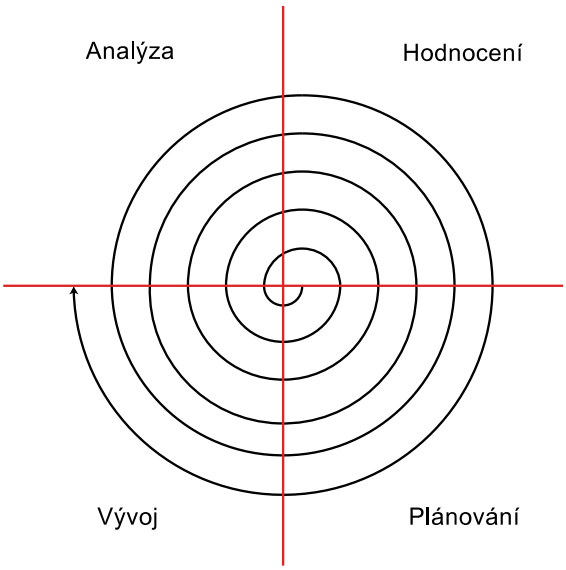
- Analýza
- Hodnocení
- Vývoj
- Plánování

Výhody

- Vytváří prostředí pro vývoj znovupoužitelných komponent
- Model je komplexní – vhodný pro složité projekty
- Dá se pomocí něj vyloučit nevhodná řešení

Nevýhody

- Nepružný pro internetové aplikace
- Komplikovanost
- Silný důraz na důvěru ve vývojáře
- Spustitelná verze až v moment spuštění



Obrázek 2: Spirálový model (3)

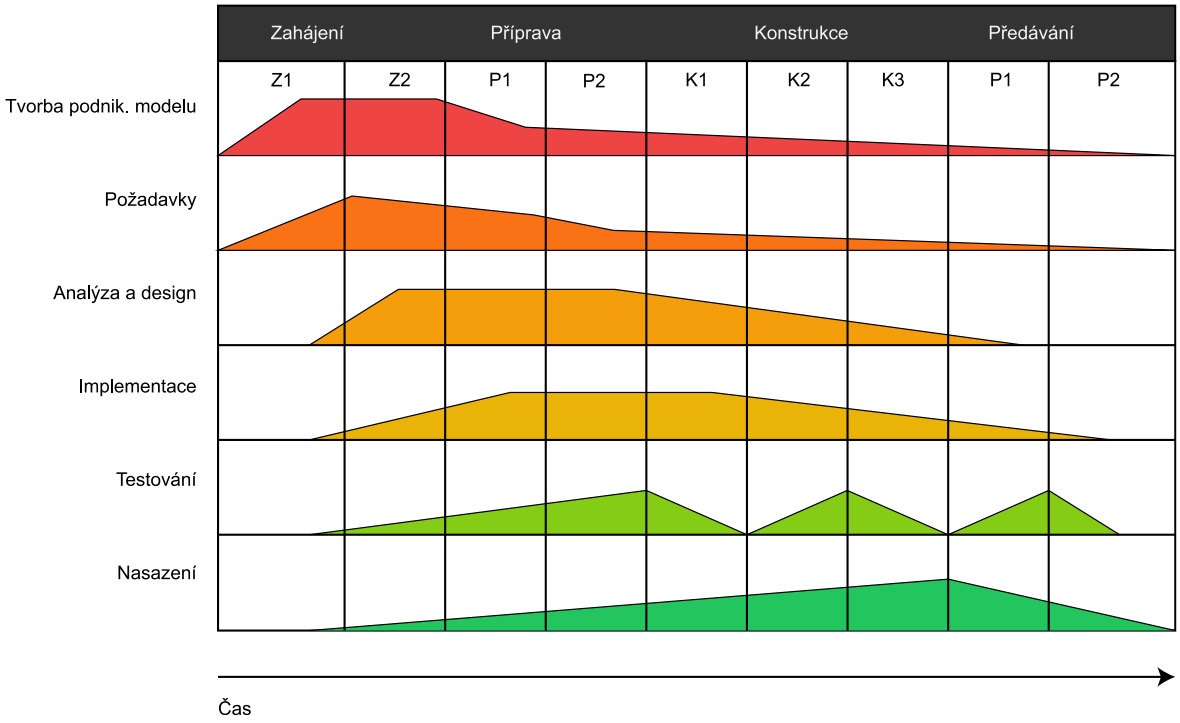
3.1.3 Metodika RUP

Jedná se o objektově orientovaný přístup k životnímu cyklu software. Náleží do skupiny přístupů řízených případy použití. Znamená to tedy, že základním elementem je chápan případ použití. Pro modelování procesu se využívá UML. (1)

Obsahuje čtyři základní fáze:

- Zahájení
- Příprava
- Konstrukce
- Předávání

Každá fáze obsahuje několik dalších iterací. Před započetím nové iterace musí být vždy splněny kritéria iterace předchozí. Zahájení definuje účel, rozsah a obchodní kontext projektu. Ve fázi přípravy je analýza potřeb zákazníka a projektu, příp. definování architektury. Konstrukční fáze je nejdelší, zde se píše zdrojový kód. V poslední fází předávání se projekt předá zákazníkovi nebo postoupí do dalšího cyklu. (1)



Obrázek 3: RUP model (4)

3.1.4 Agilní přístup k tvorbě softwaru

Agilní přístupy se snaží oprostit od klasických náročných postupů za účelem zrychlení vývoje. Jsou vhodné pro menší systémy, webové aplikace apod. Pro tyto projekty jsou stávající metodiky moc složité a zbytečně brzdí vývoj. Zadání je mnohdy nejasné, často se mění, a proto je vhodné agilní přístup využívat. (1) Agilní přístup má tři základní principy:

- Přírůstkový vývoj s velmi krátkými iteracemi
- Důraz na komunikaci mezi zákazníkem a vývojářem
- Přísné automatizované testování

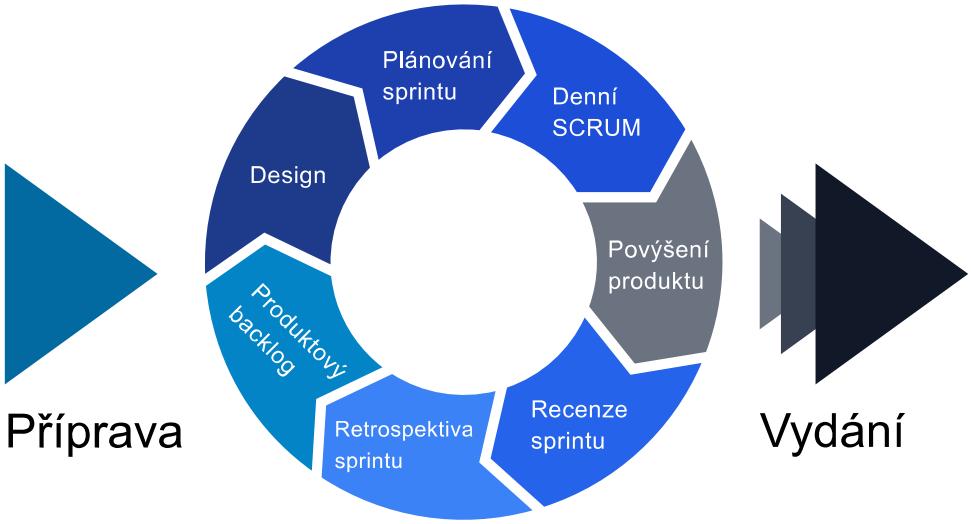
Agilní přístupy mají několik dalších metodik:

- SCRUM
- Lean Software Development
- Adaptive Software Development
- Feature-driven Development
- Extreme Programming
- Crystal metodiky

3.1.4.1 SCRUM

SCRUM je inkrementální iterativní způsob řízení vývoje. Je určený pro týmy o počtu tří až devíti pracovníků, kteří si práci rozdělí do činností, které mohou být dokončeny během vývojových cyklů s pevnou dobou trvání (tzv. sprinty). Týmy sledují vývoj a provádějí pro přizpůsobování plánů projektu 15minutové schůzky, kterým se říká stand-up meeting. Funkční částí jsou dodávány vždy na konci sprintu. (1) Klíčovým principem je:

- Zákazníci mění své pohledy na to, co potřebují a chtějí
- Vznik nepředvídatelných situací, pro které není vhodný plánovaný přístup



Obrázek 4: SCRUM způsob (5)

3.1.4.2 Lean Software Development

Jedná se o velmi populární způsob, podobný způsobu SCRUM – tedy jedná se o dynamickou stabilitu, resp. kontrolovaný chaos. Má za cíl postavit jednoduché řešení, předložit ho zákazníkovi, a na základě zákazníkovy odezvy projekt inkrementálně budovat. (1)

3.1.4.3 Adaptive Software Development

Oproti standardnímu vývoji projektu si adaptivní vývoj staví na třech etapách:

- Spekulace
- Spolupráce
- Učení

Spekulace zahrnuje stanovení počtu iterací, termínu ukončení každé iteraci, určení obsahu jednotlivých iterací a určení termínu ukončení termínu. Spolupráce zahrnuje vývoj samostatného projektu a jeho komponent. Důraz se klade na interní komunikaci členů týmu. Učení slouží ke zlepšení procesu vývoje a zhodnocení každé ukončené iterace (hodnocení kvality projektu, jeho stavu a práce týmu). Úspěšně se využívá u menších i rozsáhlých projektů. Je vhodná, u projektů, které nelze dodávat po částech. (1)

3.1.4.4 Feature-driven development

Jedná se o způsob řízení, založený na dvoutýdenních iteracích řízených vlastnostmi produktu. Vlastnostmi se myslí dílčí výsledek, užitečný z pohledu zákazníka. Vlastnosti jsou v tomto způsobu řízení přesně definovány. Vede vývojáře k vytváření fungujících přírůstků každé dva týdny. Určuje přesný termín dokončení vývoje, na rozdíl od metodiky SCRUM. Definuje vlastnictví tříd, tedy každý vývojář odpovídá za určitou třídu. Metodika je jinak chápána jako konvenční a konzervativní, blíží se více konvenčním přístupům než agilním. (1)

3.1.4.5 Extreme Programming

Způsob řízení pro malé až středně velké týmy, u projektů, které často mění zadání, nebo jejich zadání není jasné. Je velmi striktní, co se týče dodržování pravidel. Název je získaný díky extrémnímu využití metod i z jiných způsobů řízení. U extrémního programování se nepíše dokumentace, je proto nutné zdrojový kód psát podle konvencí. (1) Využívá:

- Nepřetržitou vazbu díky krátkým iteračním cyklům
- Přírůstkového přístupu
- Automatizované testy, na kterých se podílí jak zákazník, tak vývojáři
- Programátoři spolu úzce spolupracují, tzn. využívají párového programování

3.1.4.6 Crystal metodiky

Crystal metodika staví na tom, že nemůžou metodiky vyhovovat všem požadavkům projektu. Definoval ji Alistair Cockburn, dle trojrozměrné matice. (1) Tu tvoří:

- Počet lidí zúčastněných na projektu
- Míra důležitosti projektu pro zákazníka

- Prioritu projektu

Dle těchto tří bodů se vybere metodika, která se dále přizpůsobí konkrétnímu projektu.

3.2 Technologie na straně klienta

3.2.1 HTML

HTML je značkovací jazyk, který definuje strukturu webové stránky. První verze vyšla v roce 1993 a do dnešní doby si tento jazyk prošel několika změnami. V roce 2004 nebyl Ian Hickson spokojen s navrhovanou změnou W3C přejít na nový formát XHTML. Důvodem byly stále se zvyšující potřeby aplikací a vývojářů. Chtěl HTML pouze rozšířit, nikoliv navrhovat něco nového, zvláště v momentě, kdy nebyl v jazyku významný posun. Vytvořil uskupení WHATWG, ve kterém bylo několik programátorů ze společností Apple, Mozilla Foundation a Opera Software, kteří měli podobné myšlenky. HTML tak začali spravovat. Existovaly proto dvě vývojové větve, W3C a WHATWG. Prohlížeče v roce 2007 začaly od XHTML ustupovat. W3C v roce 2007 oznámilo, že znovu zformují skupinu, která začne HTML vyvíjet. Editorem v této skupině byl zvolen Ian Hickson. A tak XHTML pomalu zaniklo, W3C fungovalo na sémantických verzích, zatímco WHATWG bylo neustále v pohybu. (6) (7) V roce 2019 W3C oznámilo, že WHATWG budou jedinými vydavateli HTML standardu. HTML dokument je textový soubor, který obsahuje speciální značky, které prohlížeči definují, jak by měl být dokument vykreslen. Pokud uživatel navštíví webovou stránku, prohlížeč obdrží HTML dokument, který zobrazí.

3.2.1.1 Syntaxe a struktura

Syntaxe HTML se skládá z elementů. Element obsahuje tagy. Tag se skládá z tagu počátečního, a koncového. Existují také tagy samouzavírací, ty nepotřebují tag koncový. Každý tag obsahuje špičaté závorky a název elementu. Koncový tag obsahuje lomítko. Název elementu (například *p, img, h1*) společně s atributy popisuje prohlížeči, jak by měl být jeho obsah vykreslen. Počáteční tag může obsahovat jakýkoli počet atributů, ale atributy musí u konkrétního elementu existovat. Například u elementu *anchor* lze přidat do *href* atributu jakoukoli webovou adresu. Element je poté v prohlížeči klikatelný, a odkáže uživatele na webovou adresu specifikovanou právě v onom atributu. (8)

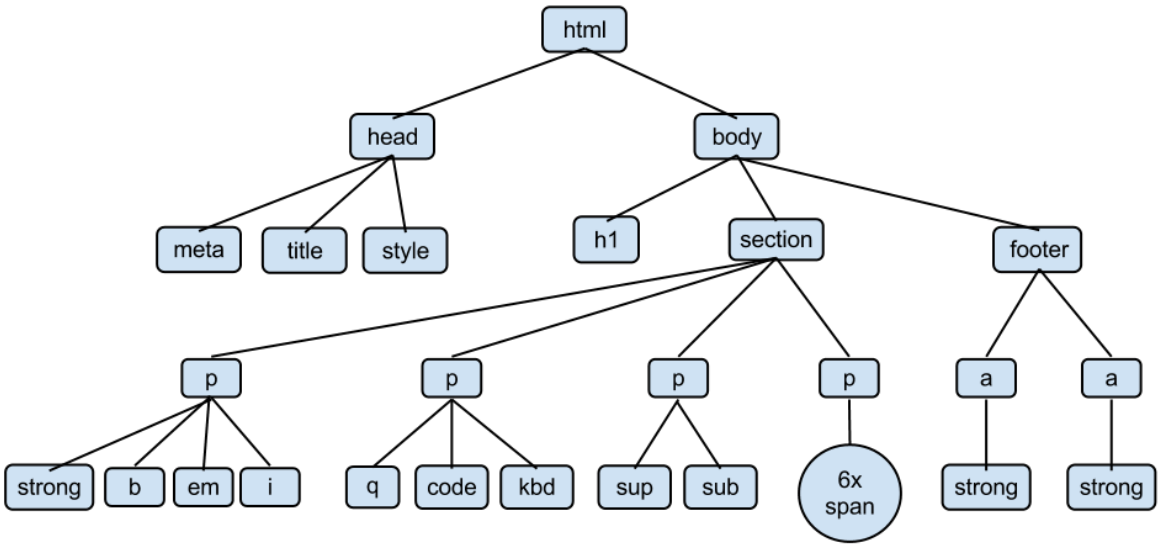
Struktura dokumentu se skládá ze základního elementu, kterým je *html*. Uvnitř *html* jsou dva elementy, *head* a *body*. Do prvního jmenovaného se uvádějí elementy, které se

týkají metadat. To znamená, že je zde uveden název dokumentu, popis dokumentu, nebo je zde využit element *link*, díky kterému lze propojit dokument s CSS. Skoro vše, co je uvedeno v hlavičce, uživatel na stránce nevidí. Výjimkou je název dokumentu, který prohlížeče zobrazují na kartě prohlížeče, případně i favikon. (8)

Pro popis kódu se využívají komentáře, které prohlížeč ignoruje a nevykresluje. Komentáře se zapisují: <!-- *Toto je komentář* -->. V dnešní době není standardem, že by se HTML kód popisoval. Komentáře jsou spíše využívány k tomu, aby se zabránilo vykreslení některého elementu, který bychom v budoucnu potřebovali, a nechceme ho z kódu odstranit.

3.2.1.2 DOM

DOM je znázornění dokumentu v paměti. Jedná se o strom objektů, který je složen ze skupiny uzlů. Uzel je asociován s objektem, ve kterém jsou uchovávány vlastnosti, metody a události HTML elementu. (8) Při využití jazyka JavaScript je možné změnit text některého ze zvolených elementů. Není nutné text měnit v HTML dokumentu. Stačí využít DOM API, pomocí kterého lze text dynamicky měnit a se stromem objektů manipulovat.



Obrázek 5: DOM strom (9)

3.2.2 CSS

První standard kaskádových stylů vydal v roce 1996 World Wide Web Consortium. (10) Jedná se o jazyk, který specifikuje, jak by se měl HTML dokument zobrazit. Bez kaskádových stylů by webové stránky byly velmi jednoduché a bez designu. Kaskádové styly nám umožňují definovat rozvržení stránky, pozicovat elementy, či používat animace.

Název byl zvolen kaskádové styly, jelikož lze na jeden element aplikovat více stylů. Styly jsou ale aplikovány podle pravidel, kdy specifičtější styl vyhrává. Tak jak by se dala znázornit kaskáda.

3.2.2.1 Syntaxe a zápis

Kaskádové styly se skládají z pravidel. Pravidla cílí na HTML elementy pomocí selektorů, které udávají, na který konkrétní element bude pravidlo aplikováno. Pravidla mohou cílit na elementy několika způsoby. Pravidlo se skládá ze selektoru a vlastností, které jsou uvnitř složených závorek. Vlastnosti se skládají z názvu vlastnosti, dvojtečky, hodnoty a středníku. Selektor se může skládat z názvu elementu, tečky a hodnoty atributu třídy či hashtagu a hodnoty atributu *id*. (10)

Pravidlo se selektorem skládajícím se z názvu elementu cílí na všechny elementy v dokumentu se stejným názvem. Pravidlo se selektorem skládajícím se z tečky a hodnoty atributu třídy cílí na všechny elementy, které mají stejnou hodnotu v atributu třídy. Pravidlo se selektorem skládajícím se z hashtagu a hodnoty atributu *id* cílí na všechny elementy, které mají stejnou hodnotu v atributu *id*. (10)

Nutno podotknout, že je využit již zmíněný kaskádový efekt. Pravidla se selektorem na *id* přebijí pravidla se selektorem na třídu a název elementu. Pravidla se selektorem na třídu přebijí selektor na název elementu. Podobný princip existuje u vlastností. Pokud definujeme v jednom pravidle dvě stejné vlastnosti s rozdílnými hodnotami, aplikuje se ta hodnota, která je definována později. Pravidla kaskádových stylů jsou z mnoha četných důvodů neuvěřitelně komplexní, ve stručnosti se nedají popsat, a proto jsou popsány pouze nutné základy k pochopení toho, jak kaskádové styly na webových stránkách fungují. (10)

3.2.2.2 Použití a podpora prohlížečů

Kaskádové styly je možné používat třemi způsoby. Jedním ze způsobů jsou takzvané inline styly, kdy jsou vlastnosti vepsány v atributu cílového elementu. Z principu není možné používat pravidla se selektory, které jsme si popsali výše. Druhým způsobem je *style* element, který se umísťuje do hlavičky dokumentu. Do tohoto elementu je možné vepisovat pravidla se selektory, není ale možná tato pravidla znovu aplikovat u jiného dokumentu. Třetí způsob je nejpoužívanější a doporučovaný. Jedná se o vytvoření separátního souboru

kaskádových stylů, kdy se soubor propojí s dokumentem. Soubor můžeme propojit s více dokumenty, a tak pravidla používat na vícero stránkách. (10)

Prohlížeče nemusí nutně podporovat všechny vlastnosti, které jsou v kaskádových stylech k dispozici. V dnešní době používáme pro tvorbu webových stránek spoustu užitečných vlastností, které nám pomáhají lépe rozvrhnout stránku. Jednou takovou je *grid*, kterou podporuje většina prohlížečů. Internet Explorer, který se v dnešní době používá spíše v korporátech má s takovými vlastnostmi ale problém. Pokud bychom si takovou stránku v tomto prohlížeči otevřeli, nebude se zobrazovat správně. Proto se v prohlížečích používá takzvaná částečná podpora, kdy je podporována starší verze vlastnosti. Které vlastnosti jsou v prohlížečích podporovány se lze dozvědět z webových stránek https://caniuse.com/.

3.2.2.3 Preprocesory

Jedná se o skriptovací jazyky, které se kompilují do kaskádových stylů. Pomocí preprocesorů lze využívat možnosti, které v kaskádových stylech nejsou k dispozici. Zápis pravidel v těchto preprocesorech je podobný jako v kaskádových stylech. Můžeme využívat vnořená pravidla, proměnné či mixiny. Mezi známé preprocesory patří SASS, Less či Stylus. (10)

3.2.3 Tailwind CSS

Tailwind CSS je vysoce přizpůsobitelný CSS framework. Jedná se o souhrn CSS tříd, které lze využít u HTML elementů. Na rozdíl od tradičního zápisu tříd, kdy každá třída obsahuje několik vlastností a souhrnně se tak definuje vzhled nějakého objektu, využívá takzvaný atomický způsob. Jedná se o princip, kdy má jedna vlastnost jednu třídu. Tím se liší od konkurenční knihovny Bootstrap. Kromě základních vlastností podporuje také tmavý režim, animace, přechody a jiné pokročilé CSS vlastnosti. Tailwind má skvělou podporu v React aplikacích. Zároveň podporuje *tree shaking* a atomický zápis umožňuje detailní stylování. Z hlediska použití je Bootstrap lepší pro začínající vývojáře, či pro projekty, které se nechtějí designem zaobírat. Tailwind je doporučován tam, kde je využito JavaScript frameworku, a kde je kladen větší důraz na design. (11)

```
<!-- Nadpis bude tučný, velké velikosti se spodním odsazením -->
<h2 class="font-bold text-2xl mb-4">Nadpis druhé úrovně</h2>
```

Obrázek 6: Ukázka využití Tailwind CSS tříd

3.2.4 JavaScript

JavaScript je objektově orientovaný programovací jazyk. Poprvé byl představen v prohlížeči Netscape Navigator v květnu 1995. Za jeho zrodem stál Brendan Eich, který ve společnosti Netscape pracoval. Hlavním cílem jazyka bylo tehdy komplementovat Javu, jazyk, který se v té době hojně využíval při tvorbě webových stránek. JavaScript měl zlepšit interaktivitu HTML. Nutno podotknout, že JavaScript a Java spolu nemají skoro nic společného. Název byl zvolen čistě jako marketingový tah, jelikož byla v tehdejší době Java velmi rozšířená. Poté, co se JavaScript rozšířil mimo Netscape Navigator, byl napsán standard ECMA-262, který popisoval, jak by se měl jazyk chovat. Tento standard byl napsán neziskovou společností Ecma International. Druhý standard ISO/IEC 16262 je šířený Mezinárodní organizací pro normalizaci. (12)

Jazyk popisovaný těmito standardy se nazývá ECMAScript, nikoliv JavaScript. Důvodem byla ochranná známka na název „JavaScript“ společnosti Sun (nyní Oracle). JavaScript je proto jazyk, zatímco ECMAScript definuje jazykový standard verze jazyka. Nejnovější jazykový standard je nyní ECMAScript 8, vydaný v červnu 2017. (12)

3.2.4.1 Proměnné

V JavaScriptu je možné deklarovat proměnné několika způsoby. Proměnné nemají typ. Znamená to tedy, že v nich lze uchovávat jakékoliv hodnoty, ať už se jedná o čísla, či řetězce. Jedná se o jedno z úskalí tohoto jazyka. Z tohoto důvodu vznikl TypeScript, který již typovaný je. Jedná se ale pouze o syntaktický doplněk a musí se kompilovat do JavaScriptu. (13)

Pokud chceme proměnnou později měnit, měli bychom využít deklaraci pomocí *let*. *Let* nám umožňuje hodnotu později přepsat. Proměnná deklarovaná bez hodnoty nabývá hodnoty *undefined*. Pokud proměnnou později měnit nechceme, měli bychom využít deklaraci přes *const*. Pokud bychom chtěli hodnotu změnit, program skončí chybou. Deklarace přes *var* je také možná, hodnota se může měnit i později. Tento způsob deklarace ale není doporučován, jelikož je zastaralý. Navíc na rozdíl od *let* není zaměřený pouze na funkci, ale je k dispozici globálně. (13)

Konvence deklarování proměnných je dobré dodržovat. Názvy proměnných se musejí skládat pouze z písmen, čísel, nebo znaků dolaru ($) či podtržítka (_). V praxi se

používá pro pojmenovávání *camelCase,* princip, kdy název začíná písmenem malým a každé další slovo proměnné písmenem velkým. (13)

3.2.4.2 Objekty

Objekty v JavaScriptu jsou trochu jiné než od těch, které lze nalézt v jazycích založených na třídách, například Java či Python. Objekt je skupina klíčů a hodnot. Má veřejná data a nejedná se o instanci žádné třídy. Nejedná se tedy o standardní objekt v tradičním orientovaném programování. Objekty je možné ukládat do proměnné, a jakmile je objekt uložený, lze u proměnné využívat tečkovou notaci, stejně tak můžeme měnit jednotlivé hodnoty objektu. Lze používat *delete* operátor, který z proměnné umožňuje smazat klíč. V objektu je možné ponechat koncovou čárku, kterou není nutno mazat, jelikož s ní JavaScript umí pracovat. (13)

3.2.4.3 Pole

Pole je objekt, jehož název vlastností jsou čísla v řetězci. Do pole lze ukládat libovolný počet hodnot. Zajímavostí je, že pole obsahuje vlastnost *length*, do které se ukládá počet hodnot v poli. Stejně jako u objektů není ponechaná koncová čárka problém, v případě, kdy se předpokládá vepsání další hodnoty. Jelikož je pole objekt, dá se poli přiřadit jakákoli vlastnost, ačkoliv se nejedná o běžný standard. V případě, kdy je potřeba zkontrolovat, zda je proměnná pole, lze použít *Array.isArray()* funkci. Operátor *typeof* u pole vrací hodnotu *object*, jak popsáno výše. (13)

3.2.4.4 Podmínky

Podmínky vykonávají či přeskakují tvrzení na základě toho, zda došlo ke splnění podmínky či nikoliv. Základní podmínkou je podmínka *if/else,* a také složitější podmínka *switch*. *If* podmínka funguje podobně jako v ostatních programovacích jazycích. Samotný výraz, vůči kterému je podmínka prováděna je umístěn do jednoduchých závorek. V závorce je možné vyhodnocovat několik výrazů a podmínky tak kombinovat. Podmínku *if* je možné používat samotnou, kdy dojde k vykonání tvrzení ve složených závorkách při splnění podmínky. Pokud je chtěno vykonat tvrzení, je možné jej uvést ve složené závorce za definicí *else*. Tato definice se provede v případě nesplnění podmínky. Ty je možné zanořovat.

Switch umožňuje provádět podmínku, kdy je kontrolována hodnota proměnné vůči jednotlivým definovaným *case* hodnotám. Pokud se hodnota proměnné rovná té hodnotě definované u *case,* dojde k vykonání tvrzení definovaných u onoho *case*. (13)

3.2.4.5 Cykly

While cyklus, jak známe z ostatních jazyků, plní příkaz, dokud je podmínka splněna. Podmínku je možné kontrolovat na začátku či na konci. Tomu se říká *do* cyklus. *For* cyklus slouží k iteraci mezi elementy. Důležitou součástí je zde proměnná (iterátor), která po každém cyklu zvyšuje svoji hodnotu o 1 do doby, než je podmínka splněna. *For* cyklus má dvě varianty, *for of* a *for in* cykly. (13)

3.2.4.6 Funkce

Funkce se v JavaScriptu deklarují způsobem, kdy je deklarován nejdříve název funkce, v případě potřeby jsou uvedené parametry a následně tělo funkce. Lze také využít takzvané arrow funkce. Ty se deklarují způsobem, kdy je využit operátor =>, který je nazýván šipka. (13) Není potřeba definovat typy parametrů či výsledku funkce.

3.2.4.7 Asynchronní programování

Webové aplikace či programy mohou spouštět své úlohy buď synchronně, či asynchronně. Souběžné úlohy v jazycích jako je C++ používají několik vláken, u kterých je nutné hlídat, aby nedošlo ke korupci dat, když je hodnota aktualizována několika vlákny najednou. JavaScript běží na jednom vlákně, kdy lze spustit v jeden okamžik pouze jednu úlohu. Pro spuštění další úlohy je nutné počkat až se tato úloha dokončí. Teprve pak lze spustit další. Proto jsou úlohy, u kterých není třeba čekat na jejich dokončení asynchronní. Jednoduše lze definovat, co je od funkce očekáváno a kterou následující funkci by měla spustit, až se dokončí. (13)

3.2.4.8 Moduly

Moduly jsou soubory obsahující kód ve formě tříd, funkcí, či jiných hodnot, které je plánováno využívat na více místech. Lze je využívat jak v prohlížeči, tak na straně serveru. Je možné využívat jak moduly, které si napíše sám vývojář, tak ty, které vyvinuli ostatní vývojáři. Existuje několik definicí modulů. Prvním z nich je Common.js. Jedná se

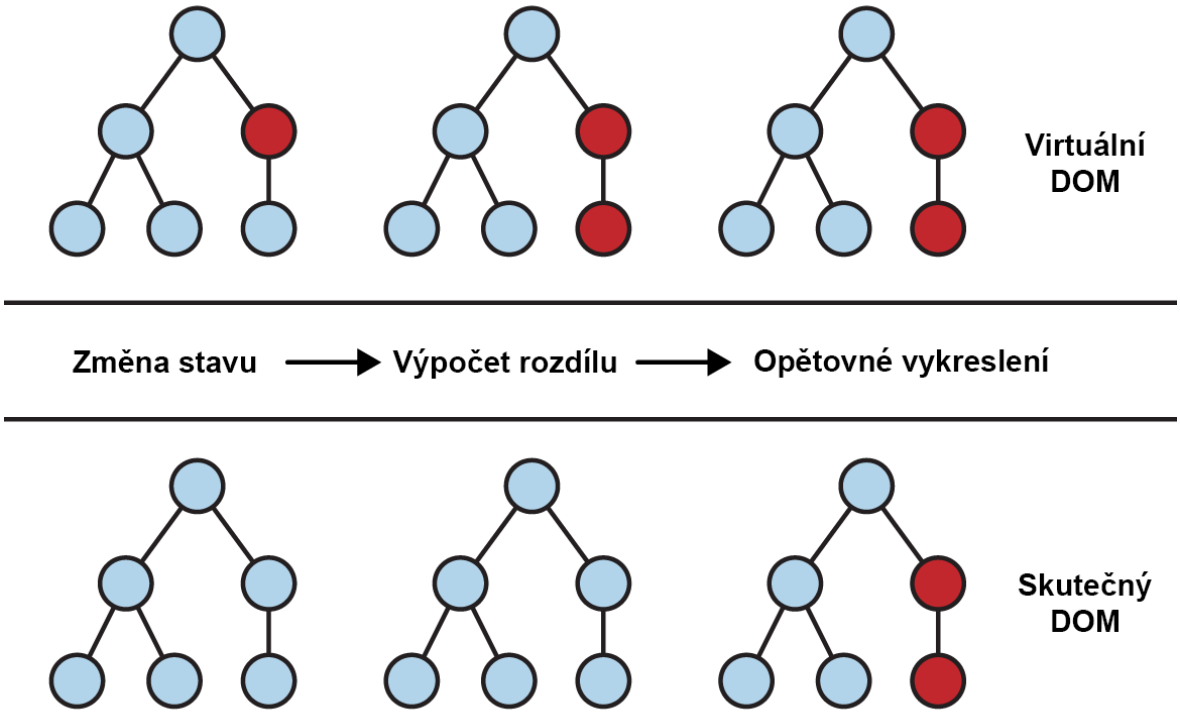
o modulový systém, využívaný na straně serveru, tedy u Node.js. Funkci lze vyexportovat pomocí objektu *export*. V případě potřeby využití funkce v jiném souboru lze funkci importovat pomocí funkce *import*. Důvod, proč je tento systém využívaný na straně serveru je ten, že se moduly načítají synchronně, tedy postupně za sebou. AMD standard definuje systém pro načítání modulů asynchronně, což je vhodné u aplikací na straně klienta. ECMAScript moduly na těchto systémech staví s rozdílem, že jsou analyzovány tak, aby u nich byly zjištěny závislosti a exporty co nejdříve, což způsobuje, že těla modulů nejsou v první fázi načítána. To umožňuje asynchronní načítání modulů a jejich kruhovou závislost. (13)

3.2.5 React

React je deklarativní JavaScript knihovna pro budování uživatelských rozhraní. První koncept vyvinul v roce 2011 Jordan Walke, který v té době pracoval ve společnosti Facebook. Netrvalo dlouho a projekt dokončil v roce 2012 pod názvem React. Facebook a Instagram se rozhodli ve stejném roce React zakomponovat do svých aplikací. V roce 2013 se React stal open-source knihovnou, což značně urychlilo vývoj. V roce 2015 proto vznikla samostatná knihovna React Native, založená na Reactu, určená pro budování mobilních aplikací. Hlavní výhodou je možnost načítání a aktualizace dat na webové stránce bez nutnosti opětovného načtení. (14)

3.2.5.1 Virtual DOM

Virtual DOM je programátorský koncept, kdy je virtuální podoba uživatelského rozhraní uložena v paměti a poté synchronizována se skutečnou podobou. Tomuto procesu se říká rekonciliace. Ten funguje na principu, kdy je uživatelské rozhraní v určitém stavu, které je potřeba pozměnit. React pozmění skutečný DOM tak, aby se shodoval s tím virtuálním. Vývojář se o samotnou změnu nestará. (15)



Obrázek 7: Princip Virtual DOM (16)

3.2.5.2 JSX

JSX je rozšíření jazyku JavaScript, které má pomoci jednodušeji popsat, jak by měl virtuální DOM vypadat a fungovat. Do takového souboru lze psát jak JavaScript, tak HTML či CSS. V jednom souboru je vše a uživatelské rozhraní proto nemusí být rozděleno do několika souborů. Nutno podotknout, že zápis HTML se lehce liší, například atribut *tabindex* se zapíše *tabIndex*. Důvodem je používání camelCase zápisu kódu. (17)

3.2.5.3 Komponenty

Jedním z hlavním stavebních kamenů jsou komponenty. Komponenty umožňují rozdělit uživatelské rozhraní do několika znovu použitelných částí. React umožňuje definovat komponenty jako třídy či funkce. Komponenta může přijmout (stejně jako jakákoli funkce) vstupy, kterým se říká *props* (properties). Funkce vrací element, který popisuje, co by se mělo zobrazit na obrazovce. Pokud dojde ke změně v komponentě, a ta již byla vykreslena ve skutečném DOM, nezbývá nic jiného než synchronizovat virtuální a skutečný DOM. Komponentu ve skutečném DOM nelze přímo měnit, musí se odebrat a znovu přidat. (18)

3.3 Technologie na straně serveru

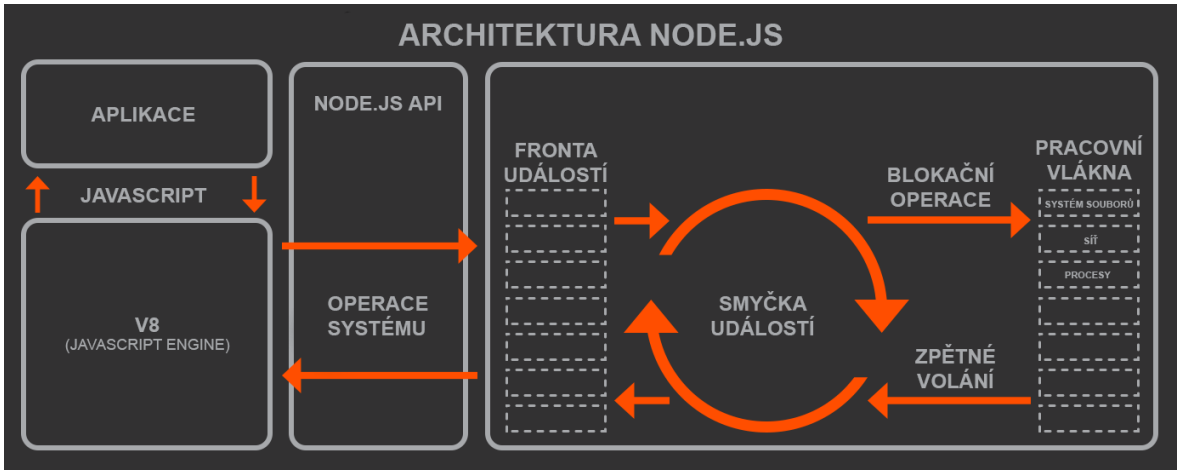
3.3.1 JSON

JSON je otevřený formát pro výměnu dat. Byl zpřístupněn široké veřejnosti v dubnu 2001 Douglasem Crockfordem. Je jednoduché ho číst a pracovat s ním. Je založen na části programovacího jazyka JavaScript na základě standardu ECMA-262 z prosince 1999. Jedná se o formát textový, který je nezávislý na jazyce. Lze ho proto používat s jazykem JavaScript, C# či PHP. Využívá konvence, které jsou známy programátorům nezávisle na jazyce. Skládá se z kolekce párů klíč/hodnota. V jiných jazycích se jedná o objekty, záznamy či struktury. (19) (20)

3.3.2 Node.js

Node.js je runtime, který vyvinul v roce 2009 Ryan Dahl. Je multiplatformní, s otevřeným zdrojovým kódem, určený pro psaní aplikací na straně serveru. Node.js aplikace jsou psány v jazyce JavaScript a mohou být puštěny na OS X, Microsoft Windows a Linuxu. Pokud chce vývojář napsat aplikaci na straně serveru má na výběr několik možností, například programovací jazyky PHP či C#. Pokud ho chce napsat v jazyce JavaScript, musí využít Node.js, který napsaný JavaScript kompiluje. (21)

Je asynchronní, což znamená, že je neblokující. Server, který je postavený na Node.js, nikdy nečeká, až se asynchronní operace dokončí. Může mezitím provádět další operace. Je postavený na JavaScript V8, enginu, který používá Google Chrome. Díky jeho designu je velmi rychlý, používá ho i mimo jiné např. Deno, alternativa k Node.js. Je doporučováno používat Node.js u IO operací. Zásadně se nedoporučuje tam, kde probíhají operace náročné na procesor. Hlavní podstatou Node.js je událostmi řízená architektura, kdy využívá pouze jednoho vlákna procesoru. Typickým příkladem jazyka, který umožňuje využití více vláken je PHP. (21)



Obrázek 8: Architektura Node.js (22)

3.3.3 NPM

Jedná se o správce balíčků pro runtime Node.js. Je automaticky nainstalován při instalaci Node.js. Obsahuje velké množství balíčků, které si může vývojář stáhnout a přidat do svého projektu a zrychlit tak vývoj. NPM využívá soubor *package.json*, ve kterém jsou definovány všechny balíčky, které jsou v projektu potřeba. (23)

Každý balíček neboli závislost, má specifikovanou verzi platných verzí, dle sémantického verzování. To zabraňuje tomu, aby nebylo možné nainstalovat verzi, která by mohla vnést do projektu nechtěné změny. Zároveň je velmi jednoduché závislosti aktualizovat na nové verze či odebírat. Většina programátorů si také může myslet, že se NPM používá pouze u aplikací na straně serveru. To ale není pravda, jelikož se také používá u frameworků na straně klienta. Na NPM proto najdeme React, Vue či Angular. (23)

3.3.4 Next.js

Next.js je JavaScript framework postavený na knihovně React. Vznikl v roce 2016. Autorem je Guillermo Rauch, CEO společnosti Vercel. Obsahuje spoustu vylepšení, které React nemá a řeší jeho hlavní nedostatky, jedním z největších je SEO.

Při tvorbě webové aplikace na architektuře MVC s použitím frameworku Express se musí pro každou stránku vytvořit odpovídající *route*, pro kterou je vykreslen příslušný *view*. V Next.js je ale možné mít ve složce *pages* soubor, který, pokud bude správně vyexportován, lze využít jako stránku. (24)

Každou stránku lze vykreslit statickou generací, či vykreslením na straně serveru. Statická generace se využívá v případě, kdy je stránka stejná pro všechny uživatele

a nepotřebuje být vykreslena při požadavku. Stránka ale může využít načtení dat z databáze, či jiného serveru v okamžik sestavení, kdy je HTML generováno. Statickou generací získáme rychlejší vykreslení obsahu, lepší SEO a možnost cachování obsahu pomocí CDN. (24) (25)

Vykreslení na straně serveru se využívá v případě, kdy by měla být zobrazena stránka, která by měla být specifická pro uživatele, nebo obsahuje data, která se velmi často mění. Stránka může využít načtení dat z databáze, či jiného serveru stejně jako u statické generace. Bohužel dojde ke ztrátě možnosti cachovat celé stránky a vykreslení trvá zpravidla déle. To je uzpůsobeno tím, že se při požadavku musí navázat spojení s instancí databáze, provedou se patřičné funkce a teprve poté se stránka může uživateli vykreslit. (26)

Next.js umožňuje kombinování těchto dvou způsobů v jedné aplikaci. Při sestavení automaticky vyhodnotí, která stránka obsahuje funkci pro vykreslení na straně serveru či statickou generaci, a dle toho stránku patřičně sestaví. Od vývojáře není potřeba žádné nastavování. V Next.js pro statickou generaci existují dvě funkce, jedna pro stránky, které mají statickou cestu (tedy předem nadefinovanou) a dynamickou (kterou může uživatel změnit). Dynamické cesty jsou skvělou příležitostí pro redakční systémy, kdy lze z redakčního systému získat data o všech článcích. Tak lze vygenerovat statické stránky, které nemusí načítat data z databáze při požadavku uživatele. Pokud se v redakčním systému přidá článek, lze nakonfigurovat *webhooks*, které znovu spustí sestavení celé aplikace, a tím se přidá nový článek. V případě několika tisíc článků by ale docházelo k sestavení celého projektu při přidání nového článku. To by zabralo několik vteřin, ne-li minut. Proto je možné do funkce přidat vlastnost *revalidate*, která sestaví pouze ty stránky, ve kterých došlo ke změně. (24)

3.3.4.1 Vestavěná podpora CSS

Next.js má vestavěnou podporu CSS. Globální CSS se musí přidávat v souboru *_app.js*. Styly, které by se měly vztahovat pouze k určité komponentě umožňuje Next.js přidávat také, musí se ale dodržovat konvence pojmenování *[název].module.css*. Zjevná je i podpora SASS, stejně tak jako inline styly, které se ale z podstaty Reactu píšou jinak než v HTML.

3.3.4.2 Rozložení

Jelikož by se logické části stránky měly rozdělovat do komponent, jak je z podstaty knihovny React známo, měly by se některé komponenty načítat na každé stránce. Většinou to je záhlaví a zápatí. Pokud jsou vytvořeny tyto dvě komponenty, musely by se definovat na každé stránce, což není vhodné. Next.js proto umožňuje využít komponentu, které se říká *Layout*, která bude obsahovat záhlaví, zápatí a jako vlastnost bude přejímat potomky, tedy veškerý ostatní obsah konkrétní stránky.

3.3.4.3 API

Jednou z nejvíce užitečných věcí je vestavěná podpora API cest. Ta umožňuje sestavení vlastního API bez potřeby statického serveru jako je Express. Důvodem, proč není potřeba využívat statického serveru je princip, kdy jsou jednotlivé API cesty sestaveny jako serverless lambda funkce, které podporuje například AWS. Ty musejí být v aplikaci umístěné ve složce */api*, která je zanořena do složky */pages*. Jednou z nevýhod je samotný princip. Jelikož se jedná o serverless funkce, které nevyžadují neustále běžící server, nelze jednoduše ukládat data na onen server samotný. Samozřejmá je i podpora parametrů v URL, kdy je lze v samotné API cestě získat. Velkou výhodou je velká škálovatelnost, rychlost a absence potřeby starat se o server samotný. V API cestě lze provádět připojení na databázi či jiné operace, u kterých není žádoucí, aby byly veřejné. (27)

3.3.4.4 Další komponenty

Image komponenta je rozšíření HTML *img* elementu. Zajišťuje optimalizaci obrázků. Škáluje obrázky dle velikosti obrazovky, ať se jedná o obrázek hostovaný na stejné doméně, nebo o obrázek z redakčního systému. Ve výchozím nastavení je na obrázcích nastavený lazy loading, tedy načítání až tehdy, kdy jsou potřeba. To zkracuje čas potřebný k vykreslení webové stránky. Obrázky jsou zároveň cachovány. (28)

Link komponenta je rozšíření HTML elementu *anchor*. Komponenta je velmi podobná té, kterou využívá React Router. Pokud je použito *anchor* elementu k navigaci v Next.js aplikaci, dochází k tomu, že se při přechodu na novou stránku celý DOM načítá znova, a to z principu není vítané. Proto tato komponenta značku nahrazuje, a umožňuje tak klientskou SPA navigaci. (29)

Komponenta *Head* připojuje svůj obsah (např. *title* tag) do HTML elementu *head*. Standardně totiž není možné do stránek vepisovat *head* element. Tato komponenta to umožňuje.

3.3.5 MongoDB

MongoDB je objektově orientovaná, jednoduchá a dobře škálovatelná nerelační databáze. Byla poprvé vydána v roce 2007 společností 10gen. V roce 2009 se z projektu stal open-source. Z hlediska rychlosti je na tom v některých případech lépe než klasické relační databáze. Samozřejmě záleží na objemu prohledávaných dat a na typu operace. (30) (31)

Nerelační databáze je ta, která nevyužívá jazyka SQL. Nemá žádné pevné schéma, žádné tabulky, sloupce či řádky. Je vhodná pro big data. NoSQL databáze jsou velmi dobře škálovatelné a distribuovatelné. Této distribuci se říká *sharding*. Jedná se o distribuci dat na cluster serverů. MongoDB se řadí mezi dokumentové databáze. (31)

3.3.5.1 Struktura

MongoDB využívá kolekce. Jedná se o skupinu dokumentů stejného typu. V relační databázi by se jednalo o tabulku. Dokument značí jeden záznam. Jeden dokument je roven řádku v relační databázi. Dokumenty se skládají z páru klíč – hodnota. Každý dokument může obsahovat několik klíčů. Klíč značí sloupce v relační databázi. (31)

Kolekce je možné mezi sebou propojovat referencemi, aby byl ustálen v projektu normalizovaný datový model. V dokumentu kolekce by se nacházel klíč, do kterého by se ukládal jedinečný identifikátor cizího dokumentu, a informace o tom, v jaké kolekci se cizí dokument nachází. Tento způsob je velmi podobný SQL řešení. (31)

Druhým způsobem, jak databázi modelovat, by bylo využití subdokumentů – resp. vložených dokumentů. Jde o způsob, kdy je v jednom dokumentu uložený další. Podstatnou výhodou je to, že lze jedním dotazem získat data o obou subjektech, zatímco při referenčním způsobu bychom se museli databáze dotazovat dvakrát. (31)

3.4 Systém pro správu verzí Git

Git je systém správy verzí softwaru. Byl poprvé vydán v roce 2005 Junioem Hamanem a Linus Torvaldsem pro vývoj jádra Linuxu. (32) V době, kdy bylo vyvíjeno jádro Linuxu, se změny uchovávaly jako záplaty a archivované soubory. V roce 2002 začal projekt

využívat distribuovaný systém správy verzí, zkráceně DVCS, pojmenovaný BitKeeper. V roce 2005 došlo k ochladnutí vztahů mezi firmou vyvíjející BitKeeper a projektem jádra Linuxu. To mělo za následek zrušení bezplatné licence, a přimění vývojářů jádra Linuxu k vyvinutí vlastního nástroje pro správu verzí. Jejich cílem bylo vyvinout systém, dle těchto požadavků:

- Rychlost
- Jednoduchý design
- Podpora pro spoustu paralelních větví
- Schopný uřídit velké projekty účinně

Od roku 2005 se Git velmi zlepšil, dospěl a zároveň stále dodržuje základní požadavky. Proto se v dnešní době používá v drtivé většině projektů. Ostatně, na systému Git staví GitHub, GitLab či BitBucket. Jedná se o webové služby, do kterých se repositáře ukládají. Git se od ostatních systémů liší jeho strukturou a architekturou. Ostatní systémy využívají principu, kdy uchovávají soubory a každou změnu souboru. Git využívá principu, kdy se při změně projektu vyfotí aktuální stav všech souborů, a uloží se reference k tomuto stavu. Pokud se některé soubory nezmění, Git neukládá tento soubor znova, ale jen odkaz k předešlé identické verzi. (32)

Důležitým poznatkem je fakt, že Git samotný nevyužívá žádný externí server. Všechny změny a historie změn se ukládají v lokální databázi. Pro kontrolu je využito kontrolního součtu, který je prováděn předtím, než se cokoliv uloží. Pro referenci, je poté tohoto kontrolního součtu využito. Projekty, které jsou vytvořeny, a využívají Git, jsou značeny jako repositáře. Všechny soubory, či složky v repositáři Git hlídá. Pokud je soubor upraven, bude ve stavu *modified*. Pokud označen k nahrání, bude ve stavu *staged*. Pokud bude nahrán, bude ve stavu *committed*. (32)

3.5 Nasazení aplikací

3.5.1 Webhosting

Webhosting je služba, kdy si lze pronajmout prostor pro webovou stránku na cizím serveru. Mezi společnosti, které webhosting provozují na českém území, patří WEDOS, FORPSI či Active24. Tyto společnosti nabízí možnost pronajmutí webové prostoru, společně s podporou pro PHP a SQL (MySQL, MariaDB). Při pronajmutí webového prostoru není možné měnit konfiguraci serveru na kterém se prostor nachází. Většina

nastavení je řízena vlastníkem serveru. Webhosting je sdílený, to znamená, že se na jednom serveru nachází více zákazníků. Důležité je podotknout, že na českém území působí velmi málo poskytovatelů, kteří provozují webhosting podporující programovací jazyky JavaScript či C#. (33)

3.5.2 VPS

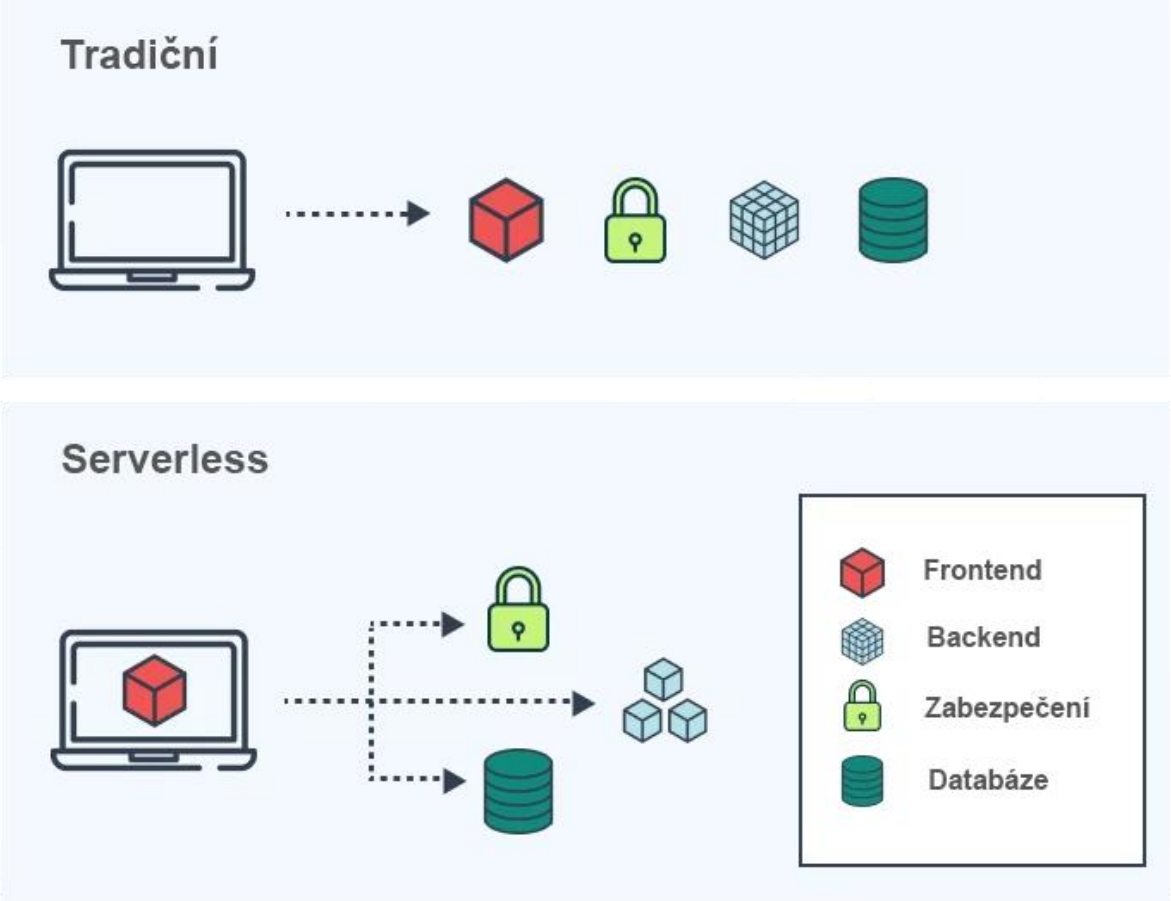
Pokročilejší verzí je využití virtuálního serveru. Jde o server, běžící ve virtualizovaném prostředí. Na tomto prostředí lze provozovat libovolný operační systém (Linux, Windows), s libovolným programovacím jazykem. Na server lze nainstalovat PHP, Node.js a jiné. Zákazník si virtuální server spravuje sám, musí tedy znát věci spojené s provozem vlastního serveru. Virtuální server sdílí stejně jako v případě webhostingu prostředky s ostatními zákazníky. Na jednom fyzickém serveru může běžet několik virtuálních serverů. Výkonnost virtuálních serverů je ale oproti webhostingu vyšší. (33)

3.5.3 Dedikovaný server

Dedikované servery jsou fyzické servery, které jsou plně k dispozici uživateli. Stejně jako v případě VPS, může uživatel provozovat cokoliv, s tím rozdílem, že má celý výkon serveru pro sebe. V některých případech si také může volit konfiguraci serveru samotného – procesor, paměť či disky. (33)

3.5.4 Serverless

Serverless je jeden z principů, jak vyvíjet a hostovat webové stránky či aplikace. U tohoto principu jsou stále zastoupeny servery, ale uživatel se nestará o jejich chod, nasazení, verze či hardware. Je to podobné jako v případě normálního webhostingu. Jakmile je stránka či aplikace nasazena, instance se dokáže škálovat dle potřeby. Největším rozdílem je způsob placení za služby. U normálního webhostingu platíme měsíčně stejnou částku. Serverless instance se ale účtují dle využitého výpočetního výkonu. Pokud tedy aplikace nevyužívá žádné prostředky, resp. pokud aplikaci nikdo nevyužívá, bude účtovaná částka nulová. Tento princip se hodí u aplikací, které proběhnou jednou za čas, a využijí větší množství výkonu. Mezi společnosti, které serverless podporují patří Amazon, Google, CloudFlare, Vercel či Microsoft. (34)



Obrázek 9: Tradiční vs. serverless architektura (35)

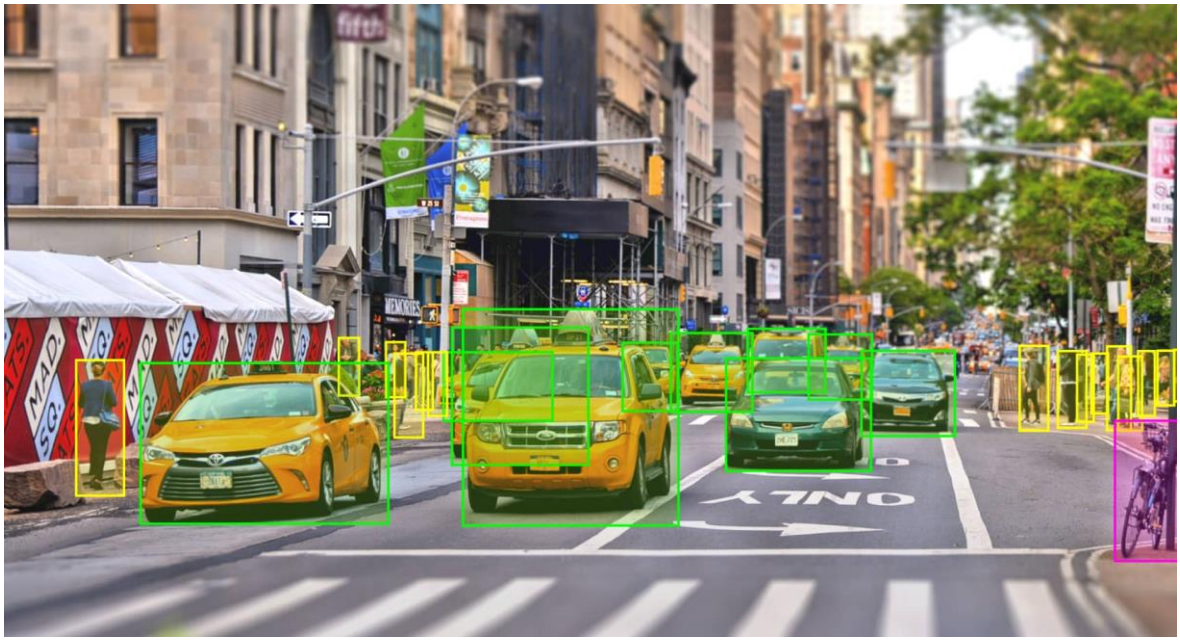
3.6 Tréninková data

Tréninková data jsou sady dat (fotografie, videa, sada textových či zvukových souborů) s přiřazenými relevantními značkami, které se používají k vytváření a zpřesňování pravidel datového modelu. Tento datový model je používán konkrétním strojem. Tréninková data učí model, jak by měl datový výstup vypadat. Model analyzuje tyto sady dat opakovaně, aby pochopil její charakteristiku a zpřesnil své uvažování. (36) (37) (38) Tréninková data rozdělujeme na dvě skupiny:

- Označená data
- Neoznačená data

Označená data jsou sadou dat, označených jednou nebo více značkami. Jsou používána v učení se s učitelem. To umožňuje modelům strojového učení naučit se charakteristiku spojenou s konkrétními značkami, což může být využito ke klasifikaci novějších sad. To znamená, že model může použít označená data k pochopení vlastností konkrétních značek, a nové sady správně klasifikovat. Označování dat je časově náročný

proces, jelikož anotaci provádí lidé, a samotná kolekce těchto dat je drahá. Uchovávání označených dat není tak snadné, jako dat neoznačených. Neoznačená data jsou opakem. Jedná se o nezpracována data, která nebyla nijak označena. Používají se v učení bez učitele. Modely strojového učení musejí najít vzory a podobnosti mezi daty ke stanovení závěru. Například, fotka s jablky, banány a pomeranči by nebyla označená. Model by tak musel každou fotku projít, a jednotlivé objekty označit sám – na základě barvy, velikosti či tvaru. Model by byl schopen jednotlivé objekty vytřídit do jednotlivých skupin, netušil by ale, že sada konkrétních objektů jsou pomeranče, například. (36) (37) (38)



Obrázek 10: Anotace několika objektů (39)

Algoritmy, které jsou známy z programování, vezmou vstup a vrátí přesný výstup, který se postupem času nemění. Tyto algoritmy historii nezohledňují, a postupem času se nezlepšují. To nelze říct o algoritmech strojového učení, které na historii staví, a dle historických dat provádějí rozhodnutí. Tréninková data jsou používána k trénování modelu. Testovací data evaluují kvalitu a přesnost modelu. V rámci co nejlepší kvality modelu je potřeba, aby byla tréninková a testovací data rozdílná. Nelze použít stejný vzorek dat pro obojí, protože by model věděl, co očekávat. Model nelze natrénovat dokonale. S největší pravděpodobností v budoucnu narazí na nový vzorek, se kterým se nikdy nesetkal. Validační data jsou sady dat, které se používají k průběžné a časté evaluaci během tréninkové fáze. Model se z validačních dat neučí, data jako samotná slouží k tomu, aby nebyl model veden špatným směrem. tzn. kontroluje se průběžná kvalita modelu. Je důležité, aby byla testovací

data co nejkvalitnější, jinak bude model velmi špatně odhadovat budoucí situace a dělat chyby. Ty stojí peníze. (36) (37) (38)

Data musí být relevantní, přímo specifická tomu, co od modelu očekáváme. Pokud chceme, aby model dokázal v budoucnu detekovat auta, je nutné mu podsouvat data, na kterých jsou auta označená. Nelze mu předsouvat obrázky potravin. Zároveň musí být různorodá. Model se nemůže zaměřovat pouze na jednu barvu, či na jednu značku automobilu. Pokud by byl model trénován pouze na bílá auta, nebude mít u černého auta v budoucnu takovou přesnost. To stejné platí i pro národnosti a etniky. Data musí být rovnoměrná. Měla by obsahovat stejné množství informací. Například, aby model dokázal hodnotit kvalitu potravin dle složení, měl by každý snímek obsahovat jak složení samotné, tak nutriční hodnoty. Pokud by některé snímky obsahovaly pouze nutriční hodnoty, je jasné, že by model nebyl tak přesný. Měla by být také obsáhlá, nemůžou obsahovat vše, ale je nutné mít takovou sadu dat, aby bylo pokryto co největší množství variant ze skutečného světa. Neoznačená data se dají získat formou open-source. Takovým zdrojem může být Google Dataset Search, případně lze také spoustu dat získat z kamer, senzorů a různých IoT zařízení. (36) (37) (38)

3.6.1 Způsoby označování

Při označování dat se lze setkat s těmito typy:

- Obdélník
- Mnohoúhelník
- Elipsa či kruh
- Kvádr
- Čáry či křivky
- Body

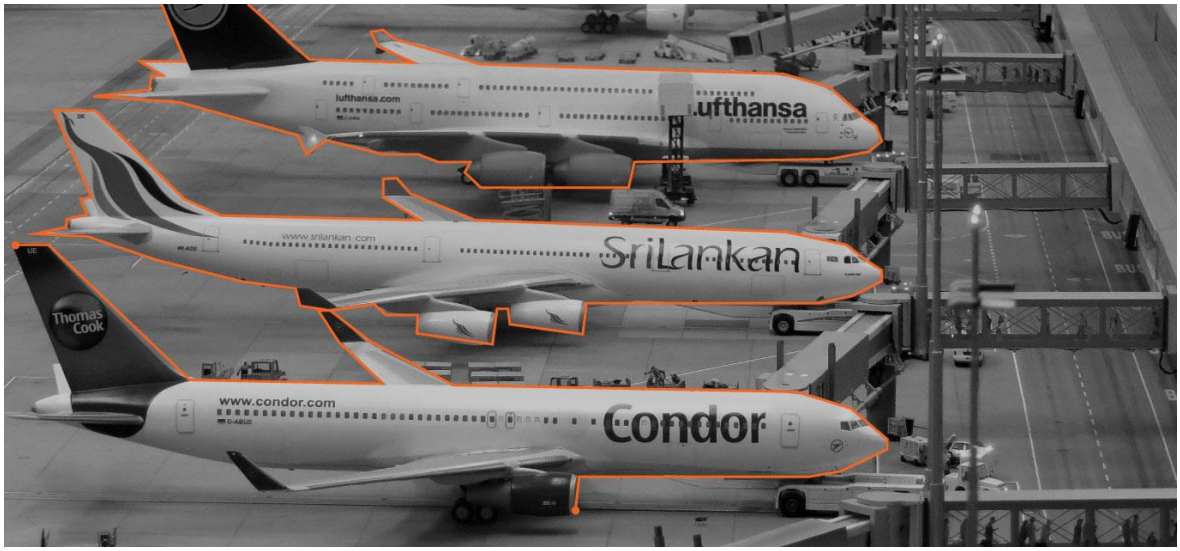
Obdélník je základním typem, se kterým se lze setkat při označování. Jeho zaznamenávání je jednoduché, stačí totiž uchovávat pouze výchozí souřadnice a jeho velikost (šířka a výška). Běžně se jedná o souřadnice levého horního rohu. (36)

Mnohoúhelník je pokročilejším typem, který se používá v případě, kdy je nutné daný objekt velmi přesně anotovat. U tohoto typu dochází k zaznamenávání bodů na okraji anotovaného objektu. Počet bodů není omezený, pro vytvoření mnohoúhelníku je ale potřeba zaznamenat alespoň dva body. Uchovávání dat se skládá ze souřadnic každého bodu, kdy je nově zaznamenaný bod vytvořený a uchovaný ihned za bodem předchozím. (36)

Elipsa či kruh je typem, který je zaznamenán dvěma informacemi. V první řadě se jedná o souřadnice středu a rádiusu. Je využíván podobně jako obdélník a umožňuje rotaci. Používá se pro anotaci objektů, jejichž tvar je podobný těmto typům. (36)

Obrázek 11: Využití kvádru (obdélník v prostoru) (40)

Anotovat lze několik objektů na fotografii. Z tohoto důvodu je praktické, aby byly pro anotované objekty vybrané vhodné typy anotací. Je velmi nepraktické pro objekt, kterým může být například auto, zvolit elipsu či kruh. Je také důležité brát v potaz čas anotace. Čím složitější typy budeme při anotaci vyžadovat, tím déle bude anotace trvat. Anotovat objekt pomocí obdélníku je mnohem rychlejší než u mnohoúhelníku, u kterého je vhodné kopírovat daný objekt. Používat typy ale není vždy nutné. Je možné také obrázky pouze klasifikovat. Klasifikace nevyžaduje používání typů, dochází totiž pouze k výpisu toho, co se na obrázku nachází. Čím vhodnější typy jsou ale použity, tím přesnější data jsou získány. Například, k označení automobilu je nejvhodnějším typem mnohoúhelník, pomocí kterého lze získat data s nejvyšší přesností. Použití obdélníku by nemuselo být vhodné, automobily by při jeho použití nemusely zahrnovat kola. Stejně tak může dojít při jeho použití k anotaci automobilu druhého, který se překrývá s námi anotovaným. Toto vše lze s pomocí mnohoúhelníku vyřešit. (36)

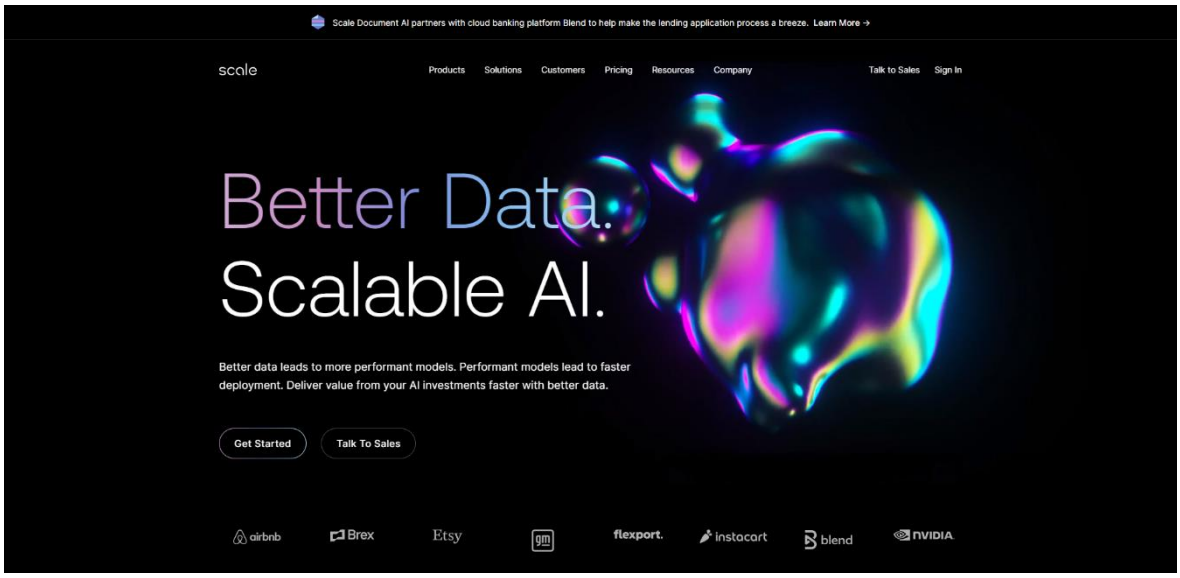


Obrázek 12: Ukázkový příklad využití mnohoúhelníku v praxi (41)

3.6.2 Datové platformy

Datové platformy jsou webové aplikace, jejichž cílem je označování a kategorizace dat. Může se jednat o fotografie, text – například recenze, odborné texty či zvukové nahrávky. Označování obrázků může probíhat tak, že zákazník zašle větší množství obrázků, které chce zpracovat. V případě fotky ulice by chtěl označit všechny automobily. Datová platforma fotku obdrží, zaměstnanci společně se speciálními nástroji označí na fotografii automobily, a fotku označí za dokončenou. Zákazník obdrží data o pozici automobilů na fotografii. Datová platforma poté všechna data uchová, a tím má do budoucna k dispozici ohromné množství dat, které může poskytovat dál. Nutno dodat, že takto označená data se využívají pro strojové učení. Typickým představitelem platformy, která by tato data využila, je TensorFlow. Existuje mnoho firem, které využívají datové platformy pro své strojové učení, např. Airbnb, NVIDIA, PayPal, SAP, Lyft a další. (36)

3.6.2.1 Scale AI

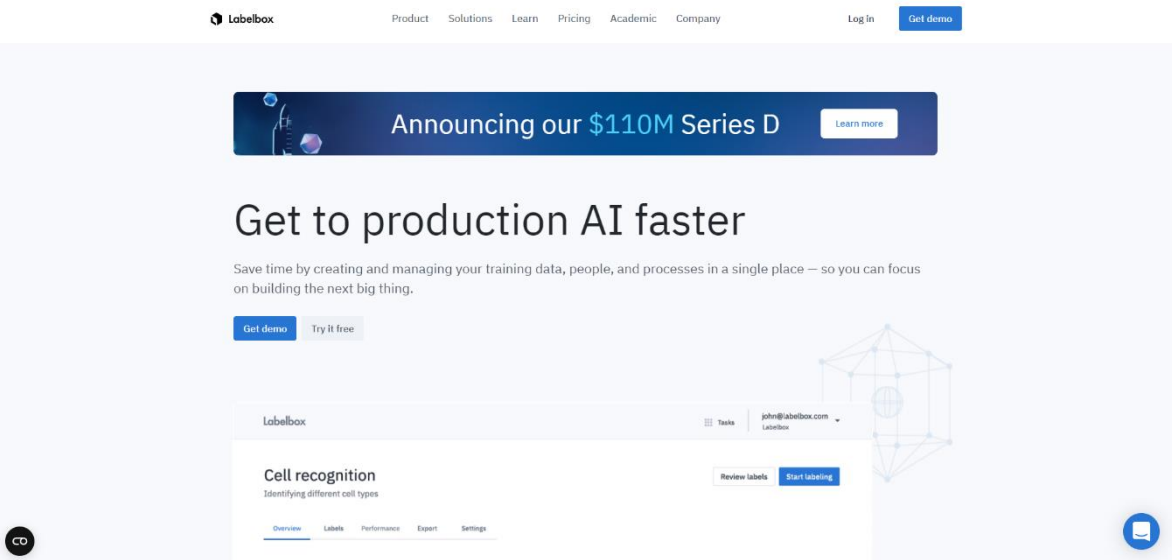


Obrázek 13: Úvodní stránka Scale.ai (42)

Scale AI založil v roce 2016 Alexander Wang. Jde o společnost, zabývající se označováním dat, a poskytování těchto označených dat třetím firmám. Mezi klienty patří například Airbnb, Samsung, SAP či NVIDIA. Momentálně nabízí hned několik produktů, největší část se týká anotací, například Scale Rapid či Scale Image. Scale dokáže data zpracovat na základě strojového učení předběžným označením a lidskou revizí. Scale se nejvíce soustředí na práci s API, proto je možné každý postup provést prostřednictvím API. (42)

API podpora je dostatečná, lze využít dvou momentálně podporovaných SDK – pro Python a JavaScript (Node.js). Pokud chceme využít jakýkoli jiný jazyk, je to možné provedením běžného HTTP požadavku. Například takto lze provést volání API pomocí JavaScript na straně klienta. Tvorba projektů přes volání API je jednoduchá, srozumitelná. Projekt lze volat, a provádět s ním některé operace, jako je nahrávání obrázků k označení aj. Nutno podotknout, že se nahrané obrázky neukládají na servery Scale, ukládají se pouze jejich odkazy. To sebou nese výhody pro Scale, ale může být negativem pro zákazníka. Scale dokáže po dokončení úloh zavolat uživatelem definovanou URL, případně odešle e-mail. (42)

3.6.2.2 Labelbox



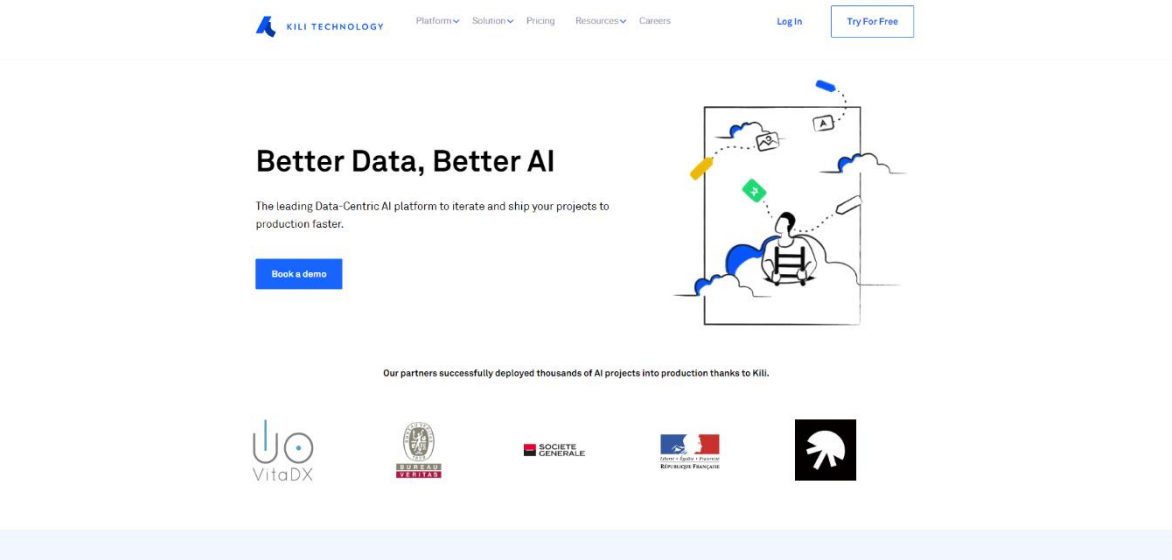
Obrázek 14: Úvodní stránka Labelbox (43)

Labelbox je populární službou pro označování dat, založená v roce 2017. Stejně jako Scale.ai, umožňuje tvorbu a anotaci hned několika projektů, jako je například anotace obrázků, textů, videí či zvukové stopy. Dokáže také predikovat anotaci dat na základě již uživatelem naučeného AI modelu. Výslednou anotaci je ale potřeba vždy potvrdit uživatelem. Data je možné importovat přes aplikaci, pomocí API či s užitím SDK. Labelbox poskytuje SDK pouze ve verzi pro programovací jazyk Python. Umožňuje také kolaboraci v rámci týmu pomocí komentářů. Platforma jako taková poskytuje také různé balíčky, pomocí kterých může společnost dostat přiděleného projektového manažera, který se stará o správné zpracování nahraných dat. Mezi společnosti využívající Labelbox patří například televizní síť WarnerBros, producent oceli ArcelorMittal či agrochemická společnost Bayer. (43)

Platforma samotná umožňuje nahrávání souborů na servery platformy, v případě, kdy je chtěno, lze soubory nahrát na vlastní datové úložiště. Podporovaná datová úložiště jsou AWS S3, Google Cloud Storage či Azure Cloud Storage. Samozřejmostí je definice taxonomie projektu, a v případě, kdy je nutné, lze vytvářet své vlastní editory, které mohou být oproti výchozímu pozměněny. Podporované objekty jsou obdélníky, mnohoúhelníky či body. V editoru je možné měnit efekty obrázku, tedy nastavovat jas či kontrast. Objekty samotné lze duplikovat, přesouvat či skrývat. Editor je jinak strohý. Mimo editor poskytuje analytiku k projektu, tedy základní informace o počtu položek hotových, či v procesu.

Šikovnou funkcí jsou také webhooky, tedy mechanismy, které po určité změně v procesu anotace (dokončení procesu, dokončení jedné položky) dokáží upozornit definovanou webovou adresu. Platforma je dostupná ve třech úrovních: zdarma, pro verze či verze pro korporace. Většina funkcionalit ve verzi zdarma je omezená, podpora je dostupná pro většinu dat, bohužel je zde maximální limit provedených anotací nastavených na 10 tisíc, a stejně tak je omezen API přístup. Zbylé dvě úrovně vyžadují kontaktování společnosti samotné – není zde proto dostupný ceník. Dostupná je podpora interního týmu společnosti, který s anotací dat, dokáže pomoci. (43)

3.6.2.3 Kili Technology



Obrázek 15: Úvodní stránka Kili Technology (44)

Kili Technology je francouzská firma, založená v roce 2018. Poskytuje stejné služby jako platformy předchozí. S její pomocí lze provádět anotaci obrázků, videí, PDF souborů či textů a zvukových stop. Lze rychle anotovat, případně, pokud je potřeba, pouze klasifikovat. Anotaci je možné provádět nástroji jako je obdélník, mnohoúhelník, či čára. Tvorbu projektů a některé další operace, lze samozřejmě provádět také pomocí API, kdy je k dispozici Python SDK. Co se týče ceníku, Kili poskytuje tři úrovně, jednu zdarma, dvě placené – jedná se o podobný případ jako u platformy Labelbox. Úroveň poskytovaná zdarma limituje počet uživatelů na 5, v případě, kdy je potřeba nástroj používat v týmu. Poskytuje, ale neomezený počet označených položek či rolí. V případě, kdy je požadována anotace zvukové stopy, PDF či podpora automatizace, je nutné vybrat jeden ze zpoplatněných plánů. Projekty a označené položky je možné exportovat ve formátu JSON.

Dostupné jsou také různé prvky pro kontrolu kvality označených dat. Nahrávání dat je umožněno přes samotnou aplikaci, pomocí API, případně lze využívat zadávání URL adresy položek samotných. V platformě je zakomponováno samostatné strojové učení, kdy je jako u platformy Labelbox možné postupným označováním dat rovnou model strojového učení učit. Tento model poté pomáhá při označování dalších dat. (44)

4 Vlastní práce

4.1 Volba technologií

Pro tvorbu aplikace samotné bylo potřeba vybrat vhodné programovací jazyky a technologie. Výběr byl proveden vícekriteriální analýzou. Jednotlivé výběry bylo potřeba rozlišit na několik kategorií. Každá kategorie má podstatnou hodnotu při tvorbě aplikace.

- Programovací jazyky (k tvorbě serveru)
- Frontend knihovnu
- Hosting
- Datové úložiště

4.1.1 Programovací jazyk

U výběru programovacích jazyků došlo k výběru vhodného jazyka. Jedná o jazyky, které se běžně používají k psaní aplikací na straně serveru. Nejpoužívanějšími jazyky jsou PHP, Python, Java, C# či JavaScript (Node.js). (45) Byla stanovena následující kritéria:

- Uzpůsobilost k serverless provozu
- Zkušenosti s jazykem
- Možnosti hostingu
- Rychlost

Tato kritéria byla do vícekriteriální analýza zavedena. Kritériím byly stanovené váhy, kdy nejvyšší váhu měla zkušenost vývojáře s jazykem, poté možnosti hostingu, tedy, kolik společností poskytuje pro tento programovací jazyk hosting, uzpůsobenost k serverless provozu, který se vyplatí v případě, kdy nechceme za server platit fixní měsíční částky, ačkoliv by se v aplikaci nic nedělo. Rychlost je kritérium s nejnižší váhou. V podobné aplikaci nezáleží na rychlosti, a rychlostní rozdíly jsou zanedbatelné. Nejlépe z analýzy vyšel programovací jazyk JavaScript (Node.js), se kterým je autor velmi dobře seznámen. Zároveň je velmi rychlý a podporuje serverless.

Kritéria	Typ	Stupnice	Váha
Zkušenosti s jazykem	MAX	POM	0,40
Možnosti hostingu	MAX	POM	0,25
Serverless	MAX	POM	0,25
Rychlost	MAX	POM	0,10
Celkem			**1**

Tabulka 1: Kritéria a váhy pro analýzu programovacích jazyků

Varianty	Zkušenosti s jazykem	Možnosti hostingu	Serverless	Rychlost
PHP	2	9	9	6
Python	1	4	9	5
Ruby	0	4	9	4
C#	4	7	9	7
JavaScript (Node.js)	10	4	10	10

Tabulka 2: Matice programovacích jazyků

Varianty	Skóre	Pořadí
JavaScript	1,75	**1**
C#	2,38	**2**
PHP	2,63	**3**
Python	3,88	**4**
Ruby	4,38	**5**

Tabulka 3: Skóre a pořadí programovacích jazyků

4.1.2 Frontend knihovna/framework

V případě, kdy byl zvolen programovací jazyk, bylo také potřeba zvolit frontend knihovnu či framework. Ta zajišťuje jednoduchou práci s DOM. Těch je několik. Nejdůležitější je ale React, Angular, Svelte a Vue. (45) Byla stanovena čtyři kritéria:

- Zkušenost s knihovnou/frameworkem
- Popularita
- Velikost
- Dokumentace

Nejvyšší váha byla stanovena zkušenostem, kdy je potřeba, aby vývojář dané knihovně či frameworku rozuměl. Knihovny jsou v mnoha věcech podobné, záleží tedy spíše na zkušenostech daného vývojáře a schopnostech. Druhým kritériem s nejvyšší váhou je popularita, která se odvíjí od počtu vývojářů, kteří knihovnu či framework za poslední rok využili. (45) Čím populárnější knihovna je, tím více bude různých návodů, které usnadní

samotný vývoj. Třetím nejdůležitějším kritériem je velikost knihovny. Při frontend vývoji se snažíme udržovat kód co nejmenší, aby došlo k přenosu co nejméně dat směrem k uživateli a vykreslení obsahu bylo co nejrychlejší. Posledním kritériem je dokumentace. Některé knihovny mají rychlý a progresivní vývoj, což bohužel snižuje kvalitu dokumentace, která se velmi často mění a nestíhá být aktualizována.

Nejlépe z analýzy vyšla frontend knihovna React, která byla zakomponována do projektu. Nejdůležitějším kritériem v tomto případě byla zkušenost s React knihovnou a její popularita. Ke knihovně React byl také zvolen framework Next.js, který staví na knihovně React a doplňuje ji o podstatné funkcionality.

Kritéria	Typ	Stupnice	Váha
Zkušenost	MAX	POM	0,40
Popularita	MAX	POM	0,30
Velikost	MAX	POM	0,20
Dokumentace	MAX	POM	0,10
Celkem			**1**

Tabulka 4: Kritéria a váhy pro analýzu frontend knihoven

Varianty	Zkušenost	Popularita	Velikost	Dokumentace
React	10	40	6	8
Angular	0	23	5	7
Svelte	2	19	10	6
Vue	1	3	8	10

Tabulka 5: Matice frontend knihoven

Varianty	Skóre	Pořadí
React	1,50	**1**
Svelte	2,30	**2**
Vue	2,90	**3**
Angular	3,30	**4**

Tabulka 6: Skóre a pořadí frontend knihoven

4.1.3 Hosting

Aplikaci je nutno hostovat. Existuje několik možností hostování, které se dělí na dvě hlavní kategorie, a to je hostování na statickém serveru, či jako serverless funkce. Z tohoto důvodu byla stanovena následující kritéria:

- Cena
- Podpora CI/CD

- Edge funkce
- Podpora HTTPS
- Maximální počet minut sestavení
- Maximální přenos dat
- Možnost vlastní domény

Bylo vybráno 5 platforem, které umožňují hostování aplikace. Všechny platformy podporují serverless funkce. Jedná se o platformy Vercel, AWS Amplify, Digital Ocean, Heroku či Netlify. Nejvyšší váhu má cena, CI/CD z Git repositáře či podpora vlastních domén. Důležitá je také automatická podpora HTTPS, tedy automatické vygenerování certifikátu pro aplikaci – bez nutnosti další konfigurace, i v případě vlastní domény. Stejnou váhu má přenos dat. Jedná se o maximální počet dat, která je za období jednoho měsíce možné převést. Nejnižší váhu má kritérium edge funkcí, tedy principu, kdy dojde k distribuci částí kódu na straně serveru na mnoho míst po celém světě. Stejnou váhu má maximální počet minut sestavení, který udává, jak dlouho a často lze v období jednoho měsíce aplikaci sestavovat ze zdrojového kódu do funkční aplikace. (46)

Nejlépe vyšla platforma Vercel, která exceluje ve všech kritériích. Byla proto zvolena jako platforma, na které bude aplikace provozována.

Kritéria	Typ	Stupnice	Váha
Cena	MAX	POM	0,19
CI/CD	MAX	POM	0,19
Edge funkce	MAX	POM	0,13
HTTPS	MAX	POM	0,08
Minut sestavení	MAX	POM	0,13
Přenos dat	MAX	POM	0,08
Vlastní domény	MAX	POM	0,19
Celkem			**1**

Tabulka 7: Kritéria a váhy pro analýzu poskytovatelů hostingu

Varianty	Cena	CI/CD	Edge funkce	HTTPS	Minut sestavení	Přenos dat	Vlastní domény
Vercel	10	1	1	10	6000	100	1
AWS Amplify	6	1	0	10	1000	15	1
Digital Ocean	6	1	0	10	100	1	1
Heroku	9	1	0	1	N/A	N/A	1
Netlify	10	1	0	10	300	100	1

Tabulka 8: Matice poskytovatelů hostingu

Varianty	Skóre	Pořadí
Vercel	2,01	**1**
Netlify	2,59	**2**
AWS Amplify	3,15	**3**
Digital Ocean	3,49	**4**
Heroku	3,62	**5**

Tabulka 9: Skóre a pořadí poskytovatelů hostingu

4.1.4 Datové úložiště

Z principu výše vybrané platformy pro hosting Vercel a frameworku Next.js je třeba uchovávat data ve službě třetí strany. Data nelze uchovávat v aplikaci. Vercel doporučuje využívání datových úložišť AWS S3, Google Cloud Storage či Microsoft Azure. (47) Vzhledem k tomu, že se datová úložiště chovají velmi podobně, a jejich jediným rozdílem je ceník (48) (49) (50), bude porovnáván především on. Bylo proto stanoveno pět kritérií:

- Cena za GB uložených dat
- Cena za operace typu „A“
- Cena za operace typu „B“
- Přenos dat do úložiště
- Přenos dat z úložiště

Všechna kritéria mají stejnou váhu. V tomto případě vyšlo nejlépe datové úložiště Google Cloud, následované AWS S3. Jelikož nejsou mezi úložišti propastné rozdíly, bylo nakonec zvoleno řešení od AWS, které poskytuje cenovou úlevu, v případě, kdy je aplikace i datové úložiště provozováno na AWS.

Kritéria	Typ	Stupnice	Váha
Cena za GB	MIN	POM	0,20
Cena za "A" operace	MIN	POM	0,20
Cena za "B" operace	MIN	POM	0,20
Přenos dat do úložiště	MIN	POM	0,20
Přenos dat z úložiště	MIN	POM	0,20
Celkem			**1**

Tabulka 10: Kritéria a váhy pro analýzu datových úložišť

Varianty	Cena za GB	Cena za "A" operace	Cena za "B" operace	Přenos dat do úložiště	Přenos dat z úložiště
AWS S3	0,0245	0,0000054	0,00000043	1	0,02
Google Cloud	0,023	0,000005	0,0000004	1	0,01
Microsoft Azure	0,0196	0,00000702	0,00000056	1	0,02

Tabulka 11: Matice datových úložišť

Varianty	Skóre	Pořadí
Google Cloud	1,40	**1**
AWS S3	2,30	**2**
Microsoft Azure	2,30	**3**

Tabulka 12: Skóre a pořadí datových úložišť

4.2 Aplikace

V dnešní době se vše co nejvíce automatizuje. Shledáváme se s tím u automobilů, ve výrobě, v armádě a jiných odvětvích. Takováto automatizace probíhá tak, že se za pomoci označených dat učí model strojového učení. Jelikož se o tuto problematiku autor zajímá, a myslí si, že jde o zajímavé odvětví, rozhodl se vytvořit aplikaci, která označování dat umožňuje. Aplikace se skládá z následujícího:

- Frontend
- Backend (serverless funkce)
- Databáze
- Datového úložiště
- Služby pro zasílání e-mailů

Frontend je v aplikaci vše, co je zobrazeno uživateli. Standardně se jedná o soubory umístěné ve složce *pages* a *components*. Backend je část aplikace, která je uživateli skrytá. Komunikuje především s databází. V případě Next.js se jedná o serverless funkce, které se nahrají na servery AWS, a vykonávají určité operace pouze tehdy, kdy jsou aplikací zavolány. Tyto funkce zajišťují přihlašování, registraci či tvorbu projektů. Standardně se jedná o soubory umístěné v *pages/api*. Databáze slouží k ukládání dat, ať už se jedná o vytvořené projekty, či o založené uživatelské účty. Datové úložiště uchovává všechna data, které uživatel nahrál. Jedná se o obrázky, které chce označit. Ověřovací e-maily příp. e-maily s odkazem k obnovení hesla zasílá služba pro zasílání e-mailů. Tyto čtyři body společně tvoří celek, jehož výstupem je aplikace, zajišťující kompletní proces označování dat.

4.3 Struktura projektu

Struktura projektu je u aplikací postavených v Next.js poměrně rigidní. Je důležité udržovat základní pravidla, pokud chceme, aby naše aplikace fungovala bezchybně. Byla proto zvolena struktura, kdy se v kořenové složce projektu nachází složka *pages*. Tato složka

obsahuje všechny stránky a cesty k API, které se v aplikaci nachází. API se nachází ve složce *api*, a všechny soubory, které se v této složce nacházejí, jsou sestaveny jako serverless funkce. Ostatní soubory a složky nacházející se ve složce *pages* se týkají stránek samotných.

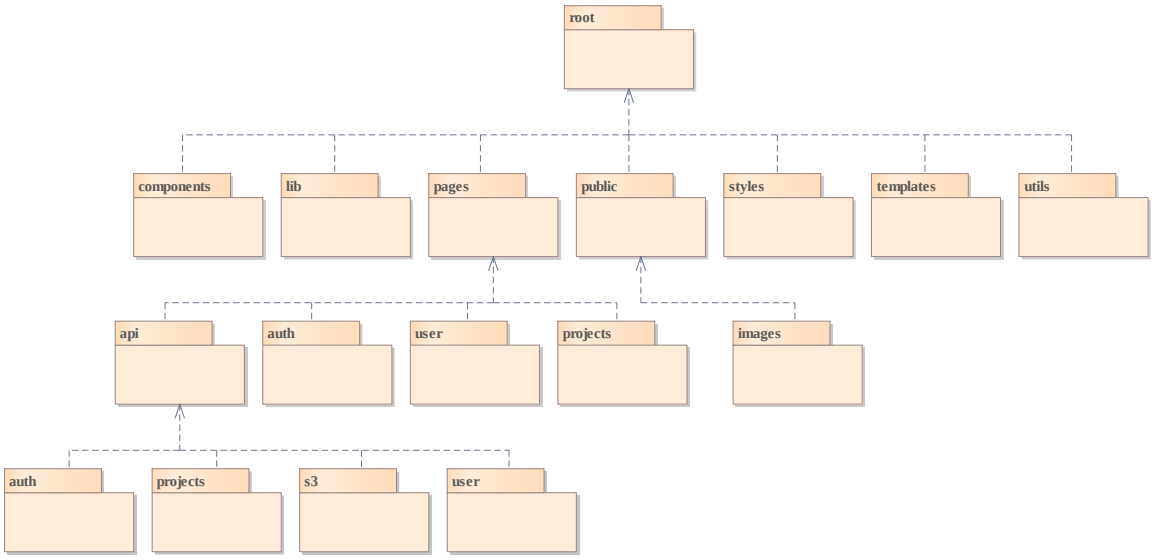
Další důležitá složka je *components*, obsahující všechny React komponenty, které jsou používány v aplikaci. Tato složka má několik souborů či podsložek, pomocí kterých jsou jednotlivé komponenty strukturovány. Dle pravidel má každá komponenta jeden výchozí export na soubor, soubory jsou označovány velkým písmenem tzv. *PascalCase*. Je zvykem komponenty uspořádávat do několika struktur dle potřeb projektu, jako je například atomický design.

Složka *lib* obsahuje malé knihovny, soubory, které mají jasnou funkcionalitu. Jde například o soubory a funkce, které připojují databázi, či inicializují nějakého klienta.

Složka *public* obsahuje veřejné soubory, které jsou přístupné komukoliv. Jedná se většinou o obrázky, ikony či favikon. Složka *styles* obsahuje CSS, které v případě této aplikace nabývá malé velikosti. Je standardem, že se do této složky umisťuje hned několik stylů, které jsou přidružené jednotlivým stránkám či komponentám. Do hlavního CSS byly importovány Tailwind styly.

Složka *utils* obsahuje jednoduché funkce, příp. objekty, které se používají napříč aplikací. Příkladem je funkce, která formátuje velikost souboru v Bytech, a formátuje je do člověkem čitelné podoby, či funkce, která kombinuje více tříd do jednoho řetězce.

Složka *templates* obsahuje e-mailové šablony, které využívá služba AWS SES. Tyto šablony jsou ve formátu JSON. Níže je vidět základní rozvržení projektu, které obsahuje složky výše popsané.



Obrázek 16: Struktura projektu

4.4 Tvorba projektů

Nejdůležitější součástí aplikace je umožnění tvorby projektů. Celý proces by měl být jednoduchý a intuitivní, s možností rozšíření o jakékoliv další funkcionality. Bylo proto navrženo proces rozdělit na tři části:

- Založení projektu
- Nahrání dat
- Určení taxonomie

Při vymýšlení procesu byly navrženy dvě možnosti, jak by mohl proces probíhat. První možností bylo vše řešit na jedné stránce, kdy by po každé uživatelem hotové části byla načítána jiná komponenta. Uložení do databáze by probíhalo až v posledním kroku, kdy uživatel dokončuje nastavování projektu. Druhou možností bylo pro každou část vytvořit vlastní stránku, kdy se na první stránce projekt vytvoří, databáze vrátí jeho ID, a to je používáno v parametru adresy URL jako jedinečný identifikátor pro další stránky. Z hlediska jednoduchosti byla zvolena varianta druhá. Při první variantě by jedna stránka řídila všechny tři prvky, což by znamenalo zbytečnou složitost kódu. Při variantě druhé je navíc projekt ukládán postupně.

Založení projektu je první částí, se kterou se uživatel setká. Na této obrazovce může vyplnit název projektu, popis a typ. Na výběr byl definován jeden typ, a to anotace obrázků. Po vyplnění těchto polí, a stisknutí tlačítka, jsou data odeslána API, která projekt v databázi

založí. Projekt je spojený s konkrétním uživatelem, kdy dochází k uložení ID uživatele k projektu.

Druhou částí je nahrávání dat. Ta umožňuje uživateli k projektu nahrát data, která by chtěl označit. Vybírat může uživatel ze svého zařízení obrázky formátu JPEG či PNG. Výběr souborů byl vymyšlen tak, aby bylo možné zvolit jeden či více souborů. Aplikace také skrývá elementy, na základě, zda došlo ke splnění podmínky – které se vyskytují po celé aplikaci. V tomto konkrétním případě byla vytvořena podmínka, která skrývá tlačítko pro nahrání souborů, pokud nebyl vybrán ani jeden soubor. Druhá podmínka se týká již nahraných dat. Ta skrývá tabulku nahraných souborů, pokud nebyl nahrán ani jediný soubor.

```
{filesSelected.length != 0 && (
  <Button type="button" onClick={handleSubmission}>
    Upload
  </Button>
)}
```

Obrázek 17: Skrývání tlačítka pomocí podmínky

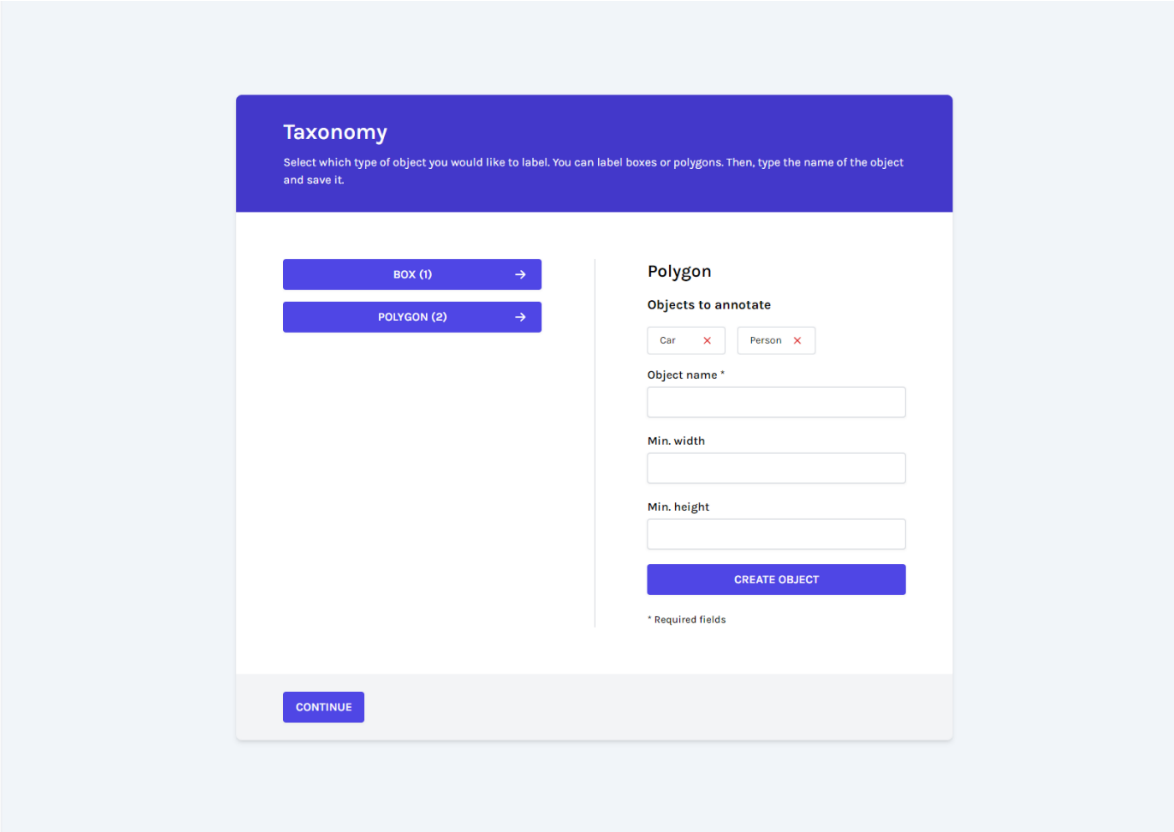
Po nahrání souborů uživatelem může nastat úspěch či chyba. Při úspěchu dojde k zobrazení nahraných souborů, kdy tabulka obsahuje název nahraného souboru, a jeho velikost, která je pomocnou funkcí převedena do uživatelem čitelné podoby. Pokud dojde k chybě, tato část nebude vůbec vykreslena, a dojde k naplnění stavové proměnné error příslušnou hláškou ze strany API. To způsobí nenulovou hodnotu této stavové proměnné, a vykreslení chybové hlášky – komponenty. Uživatel může provést nahrání nespočetně, není nutné nahrát vše poprvé.

```
{error && <AlertError title={error} className="mb-6" />}
```

Obrázek 18: Podmínka chybové hlášky

Třetí část se věnuje taxonomii. Každý uživatel by měl mít možnost určit co chce označovat a jakým způsobem. Bylo proto implementováno následující. Na levé straně má uživatel možnost vybrat, kterým nástrojem chce objekty označovat. Dostupné jsou nástroje obdélník a mnohoúhelník. Pravá část obsahuje formulář pro vyplnění názvu objektu, a jeho minimální rozměry. Ta ale zůstává skrytá, do té doby, než uživatel vybere, jaký nástroj chce použít, a dle volby nástroje se mění. Po vytvoření objektu je objekt uložen do stavové proměnné, a zobrazen v seznamu vytvořených objektů nad formulářem. Ty lze kdykoliv smazat, kdy dochází k odebrání objektu ze stavové proměnné. Po vytvoření všech objektů

dochází k nahrání těchto objektů pomocí API do databáze, a pokud je vše úspěšné, uživatel je přesměrován na stránku s projekty.



Obrázek 19: Definování taxonomie projektu

4.5 Anotace

Pro účely anotace a označování dat byla vytvořena komponenta, která označování dat umožňuje. Komponenta umožňuje uživateli označovat libovolný objekt na obrázku, dle definované taxonomie. Objekty lze označovat jako obdélníky či mnohoúhelníky. Každý nový objekt, který chce uživatel anotovat, dostane náhodně vygenerované ID, které se ukládá do stavové proměnné. Pomocí tohoto stavu komponenta ví, vůči kterému objektu v budoucnu provádět operace. Po jakémkoliv kliknutí do obrázku je zavolána funkce, která nastaví relativní pozici kliku vůči obrázku. Na základě zvoleného nástroje, je zavolána funkce, která odpovídá za zbytek anotace pro konkrétní nástroj. V případě, kdy bychom chtěli označit objekt jako polygon, funkce zkontroluje, zda se již v poli nachází objekt s totožným ID. Pokud ano, funkce vytvoří nový objekt se souřadnicemi kliku, a tento objekt vloží do pole již stávajícího objektu. Pokud ne, funkce vytvoří nový objekt s ID, názvem,

typem a souřadnicemi kliku. Tento objekt je poté vložen do stavové proměnné, která obsahuje všechny objekty.

Objekty, konkrétně tedy mnohoúhelníky, jsou poté mapovány tak, že se každému objektu, jehož typ je *polygon* namapuje HTML značka *polygon*, kdy atribut *points* obsahuje všechny kliky a jejich souřadnice, které uchováváme ve stavové proměnné.

```
{annotations
  .filter((item) => item.type === "polygon")
  .map((item) => {
    return (
      <polygon
        key={item.id}
        points={getPositionString(item)}
        className="polygon"
      />
    );
  })}
```

Obrázek 20: Mapování HTML elementu polygon

Hodnotu atributu je ale potřeba řádně namapovat, proto byla vytvořena funkce *getPositionString*, která jako parametr přebírá konkrétní objekt.

```
function getPositionString(item) {
  const position = item.data.map((coordinate) => {
    return `${coordinate.x}/${coordinate.y} `;
  });

  const positionString = position
    .toString()
    .replaceAll(",", " ")
    .replaceAll("/", ",");
  return positionString;
}
```

Obrázek 21: Funkce getPositionString

Všechny souřadnice z konkrétního objektu se namapují, a jsou zformátovány tak, aby výsledný formát vypadal takto.

```
551.390625,404  540.390625,413
```

Obrázek 22: Formát souřadnic mnohoúhelníku

Anotace je náročný proces, při kterém je potřeba, aby byl objekt označen co možná nejpřesněji. Z tohoto důvodu byla anotace rozšířena o možnost přiblížení či oddálení

obrázku. Kontejneru, který obaluje obrázek, byl přidružen event listener. Tento event listener je spuštěn tehdy, pokud uživatel nad kontejnerem použije kolečko myši. Event listener vyvolává funkci *handleWheel*, která celý proces obstarává. Uživatel může použít kolečko myši, když chce obrázek posouvat, v případě, kdy je obrázek přiblížený, a nevešel by se do plátna. Byla proto navržena varianta, kdy dochází k přiblížení tehdy, kdy je zmáčknuta klávesa Shift. Samotný *event* objekt tuto hodnotu obsahuje, nebylo proto problém podmínku vymyslet. Objekt *event* obsahuje také hodnotu *deltaY*, která nám udává hodnotu kladnou či zápornou. Z této hodnoty lze vyčíst, zda uživatel posouvá kolečkem nahoru, či dolů. Přibližování znamená zápornou hodnotu, oddalování kladnou. Byla proto sestavena podmínka, která toto rozlišuje. V obou případech dochází k nastavení stavové proměnné *multiplier*, nynější hodnotou přiblížení, která je násobená hodnotou 1,25 v případě přiblížení, či naopak 0,75 při oddálení.

```
function handleWheel(event) {
  if (event.shiftKey === true) {
    if (event.deltaY < 0) {
      if (multiplier <= 32) {
        setMultiplier(multiplier * 1.25);
      }
    } else if (event.deltaY > 0) {
      if (multiplier > 1) {
        setMultiplier(multiplier * 0.75);
      }
    }
  }
}
```

Obrázek 23: Funkce handleWheel

Stavová proměnná *multiplier* nám tedy udává hodnotu, o kterou chceme obrázek přiblížit. V moment, kdy je, jakkoliv změněna, dochází k zavolání funkce *useEffect*, která je součástí knihovny React. Funkce *useEffect* obsahuje jako druhý parametr onu stavovou proměnnou *multiplier*, v kontextu to znamená, že má být obsah *useEffect* spuštěn jen a pouze tehdy, kdy dojde ke změně této stavové proměnné. Dojde proto k zavolání funkce *updateDimensions*.

```
useEffect(() => {
  updateDimensions();
}, [multiplier]);
```

Obrázek 24: Použití useEffect při změně stavové proměnné multiplier

Obrázek 25: Anotace v případě 6x přiblížení

Tato funkce nastavuje pouze šířku a výšku obrázku na základě jeho původní velikosti, která je násobená stavovou proměnnou *multiplier*. Při zvětšení obrázku by ale docházelo k chybám, kdy by souřadnice výsledných objektů byly na jiném místě, než je očekáváno. Při zisku souřadnic kliknutí je proto nutno souřadnice kliknutí vydělit hodnotou stavové proměnné *multiplier*. Pro správnou funkcionalitu přiblížení či souřadnic objektů je ale potřeba mnohem více. Byly proto potřeba nastavit správné velikosti jednotlivých elementů, relativní či absolutní pozicování, správné *overflow*, či správné hodnoty v atributu *viewBox* u elementu *svg*.

```
function updateDimensions() {
  setImageHeight(multiplier * objectHeight);
  setImageWidth(multiplier * objectWidth);
}
```

Obrázek 26: Funkce updateDimensions

Obrázek 27: Rozsáhlá anotace aut na fotografii

4.6 Komponenty

Pro jednotnost UI a snadnou úpravu bylo vytvořeno 40 komponent, které jsou využívány v několika částech aplikace. Každá má nějakou funkci. Používají se například jako kontejnery (obalují nějaký obsah), tlačítka či jiné funkční komponenty (volají nějakou funkci, provádí operaci). V neposlední řadě slouží k rozvržení aplikace. Jednou z komponent, která byla vyvinuta, je funkční komponenta *Card*, která se používá jako karta obsahující formuláře, či jiné elementy. Jedná se o funkci, která vrací JSX kód, který se při kompilaci transformuje do HTML.

4.7 Pomocné funkce

Pro pomocné účely byly vytvořeny funkce, které mají za cíl usnadnit některé operace v aplikaci. Tyto funkce se využívají v několika částech aplikace. Jedna z nich řeší problém, kdy je potřeba spojit více definovaných tříd k sobě. Třídy, které vstupují jako parametr musí být řetězce, je ale možné, do této funkce zakomponovat i podmínky, kdy se vstupující třídy na jejichž základě mění.

```
/**
 * A helper function for returning a string containing classes separated by
spaces
 * @param  {...any} classes A list of classes
 * @returns Classes joined together
 */
export function joinClassNames(...classes) {
  return classes.filter(Boolean).join(" ");
}
```

Obrázek 28: Pomocná funkce kombinující vícero tříd

4.8 Návrh databáze

Bylo rozhodováno mezi databázemi relačními a nerelačními. Volba padla na databáze nerelační. Jedná se konkrétně o databázi MongoDB, která je již popsaná v teoretické části. Tato databáze umožňuje používat JavaScript a JSON v celém projektu, což usnadňuje práci a programování samotné. Pro vývojáře, který se zaobírá jazykem JavaScript, je tak aplikace jednodušší k pochopení. Volbu byla provedena z důvodu infrastruktury a zvolených technologií aplikace na základě následujících podmínek a potřeb:

- Lze ji provozovat oddělené od aplikace.
- Lze ji provozovat jako serverless.
- Jednoduchá na pochopení pro vývojáře zabývajícím se jazykem JavaScript.
- Škálovatelná a dostatečně rychlá.
- Cenově výhodná, s možností ceníku za počet operací, příp. zdarma.

Databázi je potřeba provozovat odděleně od aplikace, z důvodu, kdy aplikace samotná běží na službě, na které je možné provozovat pouze aplikace typu Next.js, nikoliv databáze samotné. Oddělení databáze od aplikace nám zaručí možnost ji škálovat, či přecházet na placené plány, v případě, kdy by byla databáze vytížená. Pokud bychom chtěli platit pouze za operace provedené na databázi, můžeme využít služby serverless. Z tohoto důvodu byla zvolena služba MongoDB Atlas. Jedná se o výše popsané, kdy je databáze provozována v cloudu.

Databáze byla vytvořena na serverech AWS tak, aby se všechny služby nacházely v jednom regionu, a byla tak zajištěna rychlá operace mezi nimi. Z tohoto důvodu se databáze nachází v regionu Frankfurt (*eu-central-1*). Jelikož byla aplikace vytvořena stylem serverless, nevíme, na jaké IP adrese se nachází – během měsíce se může změnit několikrát. Databázi musel být, proto povolen přístup z jakékoliv adresy, kdy se aplikace vůči databázi

autorizuje jménem a heslem. U databáze dochází k replikaci, kdy je primární node replikován na dva sekundární. Replikace zajišťuje stabilitu databáze. V případě, kdy by hlavní node selhal, zvolí se jeden z nodů sekundárních.

Připojení na databázi z aplikace je jednoduché. V aplikaci byla vytvořena proměnná prostředí *MONGODB_URI*, která obsahuje řetězec, který v sobě obsahuje adresu serveru databáze, přihlašovací údaje k serveru a databázi, ke které se má připojit. V aplikaci bylo potřeba vymyslet jednoduchý způsob, jak vytvořit modely, které by reprezentovaly kolekce. Proto byla zvolena knihovna Mongoose. Ta umožňuje modely vytvořit, tak jak bychom je znaly z objektově orientovaného programování. Připojení provádí soubor níže uvedený, který připojení provádí přes tuto knihovnu, aby bylo možné využití modelů. Soubor nejdříve importuje knihovnu Mongoose, podívá se do systémových proměnných, a zkontroluje, zda se zde tato proměnná nachází. Pokud ano, obsah systémové proměnné se uloží do proměnné v rámci tohoto souboru. Důležité je uchovávat připojení stále nacachované, z důvodu, kdy se při vývoji aplikace soubory kompilují při jakékoliv změně. Každá nová kompilace by v tomto případě vytvořila nové připojení k databázi, až do doby, kdy by bylo vytvořeno takové množství, které by databáze nebyla schopna pojmout. Tento soubor je poté exportován jako modul, a je potřeba ho využít tam, kde očekáváme připojení k databázi.

```
import mongoose from "mongoose";

const MONGODB_URI = process.env.MONGODB_URI;

if (!MONGODB_URI) {
  throw new Error(
    "Please define the MONGODB_URI environment variable inside
.env.local"
  );
}

let cached = global.mongoose;

if (!cached) {
  cached = global.mongoose = { conn: null, promise: null };
}

async function dbConnect() {
  if (cached.conn) {
    return cached.conn;
  }

  if (!cached.promise) {
    const opts = {
      bufferCommands: false,
    };

    cached.promise = mongoose.connect(MONGODB_URI,
opts).then((mongoose) => {
      return mongoose;
    });
  }
  cached.conn = await cached.promise;
  return cached.conn;
}

export default dbConnect;
```

Obrázek 29: Připojení aplikace k databázi

Modely byly vytvořeny ve složce models, kde se nachází dva modely – *User* a *Project*. *User* se týká uživatelů, kteří si založí v aplikaci účet. Schéma modelu musí být podobné schématu, které používá knihovna NextAuth.js. Ta obsahuje svoje vlastní modely, přes jejichž schéma je schopna uživatelské účty zakládat nezávisle na modelech námi definovaných. Schéma v základu obsahuje e-mailovou adresu, jméno, profilový obrázek a hodnotu, zda je e-mail uživatele ověřen. V aplikaci jsou tato pole rozšířena o heslo,

ověřovací a token obnovení. Heslo se používá při registraci uživatele přes e-mail, registrace přes OAuth toto pole nevyužívá. Ověřovací token slouží k ověření e-mailové adresy při zakládání účtu. Token obsahuje hodnotu tokenu, datum a čas kdy expiruje a také hodnotu, zda byl již použit, aby nebylo možné uživatele registrovat dvakrát. Podobné je u tokenu obnovení, který slouží k obnově hesla uživatele. Schéma je poté exportováno jako model, kdy dochází k namapování modelu na databázové připojení.

Obrázek 30: ER model databáze

4.9 Autorizace a autentifikace uživatelů

V aplikaci bylo třeba vymyslet jednoduchou autorizaci uživatelů, s možností registrace přes e-mailovou adresu, či přes OAuth protokol. Byla zvolena knihovna NextAuth.js, která dokáže uživatele autorizovat přes služby, které si uživatel zvolí. V aplikaci byla zvolena možnost autorizace přes Facebook, Google, Twitter či GitHub. Pro

funkčnost OAuth autorizace bylo potřeba aplikaci zaregistrovat u každé výše zmíněné služby, a vyžádat si API klíč. Tyto API klíče jsou uloženy jako systémové proměnné. Knihovna funguje na jednom koncovém API. To obsahuje importované jednotlivé OAuth moduly služeb, kdy je každému modulu přiřazen příslušný API klíč. Ty komunikují s konkrétními službami. Pro přihlašování přes e-mailovou adresu a heslo je využito příslušné služby, které má nadefinovanou metodu *authorize*, která se volá při každém zavolání příslušné funkce ze strany klienta. Tato metoda nejdříve zavolá připojení databáze, získá salt a uživatelské údaje, které se do metody dostávají jako parametr ze strany klienta. Následuje hashovaní uživatelem zadaného hesla s pomocí soli. Tento hash se používá v případě, kdy se dle uživatelsky zadané e-mailové adresy vyhledá v databázi shodný uživatel. Hash tohoto uživatele se porovná s hashem uživatelem zadaným. Pokud jsou přihlašovací údaje správné, do JWT tokenu a session objektu se údaje o uživateli uloží a dojde k přihlášení.

Knihovna musí s databází komunikovat pomocí svého vlastního adaptéru, který vykonává předem definované funkce, potřebné pro základní úkony s uživatelskými účty. Jelikož neexistuje žádný adaptér pro Mongoose, bylo třeba vyvinout vlastní. Adaptér musí splňovat několik základních testů, které ověřují funkčnost adaptéru. Inspiroval jsem se již existujícím adaptérem pro MongoDB. Testování probíhalo na lokální MongoDB instanci, která běžela v Docker kontejneru. Po úspěšných testech byl adaptér nahrán do NextAuth.js repositáře. Adaptér může využít kdokoli, používající Mongoose ve svém projektu. Adaptér a diskusi k němu lze shlédnout na adrese www.github.com/nextauthjs/adapters/pull/355.

4.10 Nahrávání dat do datového úložiště

Pro potřeby označování dat bylo potřeba data uchovávat. Tato data nelze uchovávat na stejném místě, kde běží aplikace samotná. Zvoleno bylo proto objektové úložiště S3 od AWS. Toto úložiště v sobě neobsahuje adresářovou strukturu, soubory jsou uloženy jako objekty. Místu, kam se data uchovávají se říká bucket. Aplikace přistupuje k bucketu pomocí klienta, který je součástí balíčku, resp. SDK. Pro úspěšnou komunikace mezi klientem a samotným úložištěm byl vytvořen na stránkách S3 uživatel, který bude mít potřebná oprávnění. Tato oprávnění budou aplikaci opravňovat k přístupu do S3. Vytvořenému uživateli bylo přiřazeno následující oprávnění. Toto oprávnění je definované AWS, a opravňuje uživatele, k přístupu do jakýchkoliv S3 bucketů. Na těchto bucketech umožňuje provádět jakékoliv operace, ať už se jedná o požadavky GET či POST.

Název oprávnění	Typ oprávnění
AmazonS3FullAccess	AWS managed policy

Tabulka 13: Nastavení oprávnění pro službu AWS S3

Bucket bylo potřeba nakonfigurovat. Z důvodu, kdy potřebujeme z bucketu načítat data jako obrázky v *img* značce, je nutné, aby byly všechny objekty ukládané do bucketu veřejné. To bylo provedeno jednoduchým pravidlem, které povolí komukoliv provádět operace typu *GetObject* a *GetObjectVersion*. Pravidla jsou definována JSON zápisem.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": ["s3:GetObject", "s3:GetObjectVersion"],
      "Resource": "arn:aws:s3:::datapick/*"
    }
  ]
}
```

Obrázek 31: Definování pravidel bucketu

Pro správnou funkčnost byl nakonfigurován CORS bucketu. V případě, kdy by CORS nebylo nakonfigurováno správně, bylo by možné objekty stahovat z prohlížeče přímým odkazem, nebylo by ale možné tyto objekty zobrazit v aplikaci, která funguje pod jinou doménou než samotný bucket. CORS bylo nakonfigurováno tak, aby umožňovalo zobrazování objektů v naší aplikaci. Povoleny byly metody GET, POST, PUT a DELETE. Konfigurace je definována JSON zápisem.

```
[
  {
    "AllowedHeaders": ["*"],
    "AllowedMethods": ["GET", "PUT", "POST", "DELETE"],
    "AllowedOrigins": ["*"],
    "ExposeHeaders": []
  }
]
```

Obrázek 32: Konfigurace CORS bucketu

Proces inicializace klienta v aplikaci je jednoduchý. Klient komunikuje pomocí API klíče, který byl získán při vytvoření uživatele. API klíč vstupuje do klienta jako parametr, stejně tak region, ke kterému má klient provádět požadavky.

```
import { S3Client } from "@aws-sdk/client-s3";

const credentials = {
  accessKeyId: process.env.AWS_SDK_ACCESS_KEY_ID,
  secretAccessKey: process.env.AWS_SDK_SECRET_ACCESS_KEY,
};

const s3Client = new S3Client({
  region: process.env.AWS_SDK_REGION,
  credentials: credentials,
});

export { s3Client };
```

Obrázek 33: Inicializace S3 klienta

Nahrávání souborů pomocí klienta je složitější. Pokud chceme nahrát soubory do bucketu, musíme je nahrávat pomocí POST požadavku, který v sobě musí obsahovat soubory samotné a jejich metadata. POST požadavek je nutno provádět vůči vygenerované URL. Tu generuje klient pro každý soubor zvlášť. V rámci aplikace je také nutno soubory uchovávat v pomyslných složkách. Byl proto implementován systém, který vezme ID uživatele, ID nově založeného projektu, název souboru a sestaví z nich řetězec, kdy jsou jednotlivé hodnoty odděleny lomítkem. Níže je vidět ukázka, jak se soubor v bucketu uchovává. Níže vytvořený soubor by byl proto zanořen do dvou složek. Pokud se data úspěšně nahrají do S3, proběhne zavolání API aplikace, která soubory uloží ve formě metadat (název, cesta k souboru či velikost) do databáze k příslušnému projektu.

```
61e210f9bbc456050253a7c6/61e2275c1031bd5d8cf662ce/image.jpg
```

Obrázek 34: Jedinečný klíč souboru

4.11 Odesílání e-mailů uživatelům

Jelikož je potřeba odesílat ověřovací odkazy uživatelům, či odkazy k nastavení nového hesla, je potřeba využít služby, která toto umožňuje. Jelikož platforma, na které je aplikace hostována, doporučuje využít e-mailové služby, místo SMTP připojení, bylo vybíráno mezi třemi službami. První službou je SendGrid, který umožňuje vývojáři odesílat

omezené množství e-mailů zdarma, má jednoduché rozhraní a přehlednou dokumentaci. Při potřebě odesílat větší množství e-mailů je ale služba drahá, a ačkoliv obsahuje robustní systém vytváření šablon, je zbytečně složitý. AWS SES umožňuje odesílat omezené množství e-mailů zdarma, má jednoduché rozhraní a přehlednou dokumentaci. Je ale levný, pokud překročíme množství zdarma a umožňuje používání šablon. Poslední službou je MailChimp, který je podobný službě SendGrid. Je bohužel zbytečně složitý. Po důkladném zvážení bylo zvoleno řešení AWS SES, které je levné, a splňuje požadavky. Pro zprovoznění této služby bylo potřeba nastavit uživateli, definovanému již u AWS S3 další oprávnění, které ho opravňují k používání AWS SES. Jedná se o oprávnění také definované AWS. Po přidání oprávnění byly oprávnění uživatele následující.

Název oprávnění	Typ oprávnění
AmazonS3FullAccess	AWS managed policy
AmazonSESFullAccess	AWS managed policy

Tabulka 14: Nastavení oprávnění pro služby AWS

Inicializace klienta je obdobná jako u S3. Odesílání specifických e-mailů zajišťují dvě funkce, které se nachází v separátním souboru. Tyto funkce připraví parametry pro příkaz odeslání e-mailu. Do funkce vstupuje jako parametr e-mail příjemce a odkaz pro ověření e-mailové adresy či obnovení hesla. Jako parametr je tedy uváděn odesílatel e-mailu, jeho příjemce, šablona a data šablony. Klient odesílá e-maily jako šablony, ve kterých jsou proměnné, které se před odesláním naplní konkrétními hodnotami.

```
import { SendTemplatedEmailCommand } from "@aws-sdk/client-ses";
import { sesClient } from "/lib/sesClient.js";

/**
 * Sends an email with a verification link to the recipient
 * @param {*} email An email address of the user/recipient of the email
 * @param {*} verificationLink A verification link of the user
 * @returns Data from the SES client
 */
const sendVerificationEmail = async (email, verificationLink) => {
  const params = {
    Source: process.env.EMAIL_SENDER,
    Template: "SendVerificationEmail",
    Destination: {
      ToAddresses: [email],
    },
    TemplateData: `{ "email": "${email}", "verificationLink":
"${verificationLink}" }`,
  };

  try {
    const data = await sesClient.send(new SendTemplatedEmailCommand(params));
    return data;
  } catch (err) {
    throw err;
  }
};
```

Obrázek 35: Funkce pro zasílání e-mailu

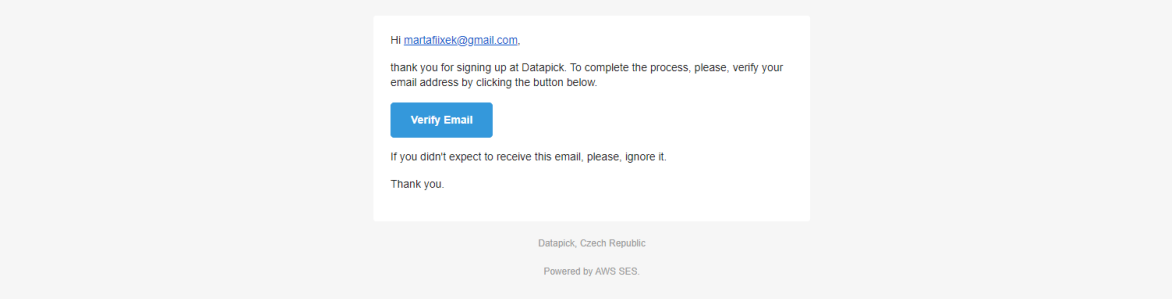
Byly vytvořeny dvě šablony. Tyto dvě šablony jsou ve formátu JSON dle schématu níže. Každá šablona musí obsahovat jedinečný název, předmět e-mailu, HTML obsah a textový obsah. Šablony byly vytvořeny pomocí níže uvedeného příkazu. Tento příkaz šablonu uloží k uživatelskému účtu uživatele, samotný JSON soubor proto není potřeba dále uchovávat.

```
aws ses create-template --cli-input-json file://template.json
```

Obrázek 36: CLI příkaz pro založení šablony

```
{
  "Template": {
    "TemplateName": "SendRecoveryEmail",
    "SubjectPart": "Set a new password",
    "HtmlPart": "<html><body>Email</body></html>",
    "TextPart": "Hi {{email}}, it seems like you have requested to change your
forgotten password. To complete the process, please, click this link
{{recoveryLink}}. If you didn't expect to receive this email, please, ignore
it. Thank you."
  }
}
```

Obrázek 37: Struktura JSON šablony



Obrázek 38: E-mail odesílaný při registraci

4.12 Nasazení

Aplikace byla v průběhu vývoje implementována ve dvou větvích. Vývojová větev slouží pro nasazování změn, a testování funkčnosti. V případě, kdy nasazené změny byly ve vývojové větvi otestovány úspěšně, byly překlopeny do větve produkční. Ta obsahuje aplikaci ve stabilní, funkční podobě. Nutno také dodat, že jsou databáze pro větve odděleny. Není proto možné narazit ve vývojové větvi na produkční data a naopak. Vývoj probíhal ve veřejném repositáři na platformě GitHub. Jakékoliv změny, které nastaly ve vývojové větvi tohoto repositáře spustily automatické sestavení aplikace na platformě Vercel. Po dokončení automatického sestavení byla vždy platformou vygenerována jedinečná URL, vedoucí na aplikaci ve stavu konkrétní změny. To umožňovalo rychlé a průběžné řešení chyb během vývoje. Produkční aplikace se sestavuje pouze v případě, kdy dojde ke změně v produkční větvi. To umožňuje podsouvat uživatelům pouze kvalitní, a testem dříve ověřené změny.

Pro lepší branding byla zaregistrována doména *datapick.tech*. Tato doména byla zaregistrována na autorovu osobu. Pro správné fungování domény a aplikace dohromady, bylo potřeba, aby doména odkazovala na aplikaci. Toto lze zajistit změnou DNS záznamů

na servery Vercel. DNS záznamy byly nastaveny následovně. Typ záznamu A byl nastaven tak, aby při zadání datapick.tech do vyhledávače byl prohlížeč odkázán na server s IP adresou 76.76.21.21. V případě, kdy uživatel zadá www jako prefix, je třeba uživatele opět přesměrovat na správnou IP adresu – toto řeší typ CNAME.

Typ záznamu	Název	Hodnota
A	datapick.tech	76.76.21.21
CNAME	www.datapick.tech	cname.vercel-dns.com

Tabulka 15: Nastavení DNS záznamů

4.13 Analýza dostupných aplikací

Byla provedena analýza dostupných aplikací, popsaných v teoretické části, a aplikace vytvořené. Analýza byla provedena vícekriteriální analýzou, kdy byla vybrána následující kritéria. Prvním kritériem bylo, zda se jedná o open-source aplikaci. Jako druhé kritérium bylo zvoleno, kolik formátů lze označovat (obrázky, zvukové stopy). Třetí kritérium je jednoduchost platformy při používání. Čtvrtým kritériem je možnost predikce dle modelu strojového učení. Pátým kritériem je možnost uchovávat data na svých vlastních datových uložištích. Posledním kritériem je ceník platformy, tedy zda je zpoplatněná či nikoli.

Váhy byly přiřazeny tak, kdy je nejdůležitější, aby aplikace byla s otevřeným zdrojovým kódem a byla rozšiřitelná. Stejné váhy mají kritéria 2, 3 a 6. Nejméně důležitým kritériem je podpora predikce dle modelu strojového učení. Byly vybrány čtyři testovací subjekty, Scale AI, Labelbox, Kili Technology a vytvořená aplikace. Tyto subjekty obdržely body k jednotlivým kritériím. Po výpočtu analýzy dle metody váženého součtu bylo zjištěno, že by měla být upřednostňována platforma Labelbox, následovaná platformou Datapick. Skutečné užití se ale může lišit. Upřednostnění aplikace před jinými konkurenty je předpokládáno u malých týmů či jednotlivců. Velké týmy by přiřadily kritériím jiné váhy dle jejich potřeb.

Kritéria	Typ	Stupnice	Váha
Open source	MAX	POM	0,29
Počet formátů	MAX	POM	0,19
Jednoduchost	MAX	POM	0,19
AI predikce	MAX	POM	0,05
Uchovávání dat	MAX	POM	0,10
Ceník	MAX	POM	0,19
Celkem			**1**

Tabulka 16: Kritéria a váhy pro analýzu platforem

Varianty	Open source	Počet formátů	Jednoduchost	AI predikce	Uchovávání dat	Ceník
Scale AI	1	4	6	8	10	5
Labelbox	5	4	10	8	10	5
Kili Technology	5	4	8	8	8	5
Datapick	10	1	8	0	6	10

Tabulka 17: Matice platforem

Varianty	Skóre	Pořadí
Labelbox	2,11	**1**
Datapick	2,32	**2**
Kili Technology	2,55	**3**
Scale AI	3,12	**4**

Tabulka 18: Skóre a pořadí platforem

5 Závěr

V teoretické části byly popsány základní metodiky vývoje softwaru, které se v praxi používají. Jedná se o již dříve uvedené klasické či agilní procesy. Ty byly z části využity v průběhu praktické části. Byly popsány základní technologie na straně klienta, použité v aplikaci, na základě analýzy variant. Samozřejmostí je HTML, CSS či JavaScript. Zvolena byla ale také knihovna React či framework Tailwind CSS. Technologie na straně serveru představují například Node.js, Next.js či MongoDB, které byly opět zvoleny na základě analýzy variant. U všech technologií bylo popsáno jejich fungování, přínos v rámci aplikace a jejich teorie. Teoretická část také popisuje verzování či samotné nasazení aplikací, resp. jejich hostování, kde je probráno, které varianty existují. Poslední částí v rámci teorie je popis tréninkových dat v rámci strojového učení, jejich typy, způsoby označování aj. Následuje také popis hlavních tří platforem pro označování tréninkových dat.

Praktická část se soustředí na aplikaci samotnou. Ta obsahuje analýzu programovacích jazyků, frontend knihoven/frameworků, hostingů, datových úložišť či služeb pro zasílání e-mailů. Na základě této analýzy byly zvoleny patřičné technologie. Jako programovací jazyk byl zvolen JavaScript, frontend řeší framework Next.js, hosting poskytuje platforma Vercel, datové úložiště a službu pro zasílání e-mailů zaštituje AWS. Aplikace umožňuje uživatelům vytvářet účty, projekty, definovat taxonomii projektů, nahrávat data, označovat jednotlivá data a tyto data od aplikace získávat. Praktická část také zahrnuje analýzu dostupných aplikací s vytvořenou. Ta rozebírá, v jakých případech by byla vytvořená aplikace vhodná a pro koho.

Při tvorbě aplikace nastalo několik problémů, kdy došlo k vyvinutí separátního balíčku pro open-source projekt Next Auth. Tento balíček byl komunitou vřele přijat, a po schválení se plánuje s poskytnutím široké veřejnosti. Aplikace byla nasazena na adrese https://www.datapick.tech/ a je volně přístupná komukoliv, kdokoliv ji může modifikovat či provozovat na vlastních serverech.

6 Seznam použitých zdrojů

(1) MARTINŮ, Jiří a Petr ČERMÁK. *Metodiky vývoje software*. Olomouc, 2018. Dostupné také z: https://dl1.cuni.cz/pluginfile.php/886974/mod_resource/content/1/Metodiky-v%C3%BDvoje-software-studijn%C3%AD-text.pdf

(2) HOADLEY, Paul. Waterfall model. In: *Wikimedia Commons* [online]. San Francisco: Wikimedia Foundation, 2005 [cit. 2021-08-07]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/5/51/Waterfall_model.png

(3) Spiral model. In: *TechTarget* [online]. 2019 [cit. 2022-02-06]. Dostupné z: https://cdn.ttgtmedia.com/rms/onlineimages/whatis-spiral_model.png

(4) KRARUP, Jakob. *Unified Process Model for Iterative Development* [online]. In: . 2020 [cit. 2022-03-13]. Dostupné z: https://upload.wikimedia.org/wikipedia/commons/c/c2/Unified_Process_Model_for_Iterative_Development.svg

(5) *Scrum Methodology* [online]. In: . Covetus LLC, 2018 [cit. 2022-03-08]. Dostupné z: https://www.covetus.com/uploads/topics/15591212335259.jpg

(6) HOFFMANN, Jay. A Tale of Two Standards. *The History of the Web* [online]. 2017 [cit. 2021-08-08]. Dostupné z: https://thehistoryoftheweb.com/when-standards-divide/

(7) GOLDSTEIN, Alexis, Louis LAZARIS a Estelle WEYL. *HTML5 & CSS3 for the Real World*. 2. Melbourne: SitePoint, 2015. ISBN 9780987467485.

(8) COULSON, Lewis, Brett JEPHSON a Marian ZBURLEA. *The HTML and CSS Workshop*. 1. Birmingham, UK: Packt Publishing, 2019. ISBN 978-1838824532.

(9) *Getting Down with CSS* [online]. In: . [cit. 2022-03-08]. Dostupné z: http://www.openbookproject.net/tutorials/getdown/css/images/lesson4/HTMLDOMTree.png

(10) ATTARDI, Joe. *Modern CSS: Master the Key Concepts of CSS for Modern Web Development*. 1. New York, New York, USA: Apress, 2020. ISBN 978-1-4842-6293-1.

(11) MICHÁLEK, Martin. Tailwind CSS: další evoluční krok pro CSS frameworky. *Vzhůru dolů* [online]. Martin Michálek, 2021 [cit. 2021-08-10]. Dostupné z: https://www.vzhurudolu.cz/prirucka/tailwind-css

(12) *History and evolution of JavaScript* [online]. [cit. 2021-08-10]. Dostupné z: https://exploringjs.com/impatient-js/ch_history.html

(13) HORSTMANN, Cay. *Modern JavaScript for the Impatient*. 1. Boston, Massachusetts, USA: Addison-Wesley Professional, 2020. ISBN 0-13-650214-8.

(14) The History of React.js on a Timeline. *RisingStack* [online]. Budapešť: RisingStack, 2021 [cit. 2022-10-21]. Dostupné z: https://blog.risingstack.com/the-history-of-react-js-on-a-timeline/

(15) *Virtual DOM and Internals* [online]. San Francisco: Meta Open Source, 2019 [cit. 2021-10-22]. Dostupné z: https://reactjs.org/docs/faq-internals.html

(16) How React.js Virtual DOM works?. In: *Hoang Trinh* [online]. 2020 [cit. 2022-03-13]. Dostupné z: https://hoangtrinhj.com/static/cb7b5d849c23ea9f8ed4dfbf6f1ea296/03914/virtual-dom-visualization.png

(17) *Introducing JSX* [online]. San Francisco: Meta Open Source, 2022 [cit. 2021-10-22]. Dostupné z: https://reactjs.org/docs/introducing-jsx.html

(18) *Components and Props* [online]. San Francisco: Meta Open Source, 2022 [cit. 2021-10-22]. Dostupné z: https://reactjs.org/docs/components-and-props.html

(19) *JSON* [online]. San Francisco: MDN, 2022 [cit. 2021-10-22]. Dostupné z: https://developer.mozilla.org/en-US/docs/Glossary/JSON

(20) *Working with JSON* [online]. San Francisco: MDN, 2022 [cit. 2021-10-22]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

(21) *Introduction to Node.js* [online]. San Francisco: OpenJS Foundation, 2022 [cit. 2022-08-13]. Dostupné z: https://nodejs.dev/learn/introduction-to-nodejs

(22) The Secret Life of Event Loop - Meetup Overview. In: *Symphony* [online]. San Francisco: Symphony, 2019 [cit. 2022-03-08]. Dostupné z: https://symphony.is/blog/secret-life-event-loop-meetup-overview

(23) *About npm* [online]. Oakland: npm, 2022 [cit. 2022-08-13]. Dostupné z: https://docs.npmjs.com/about-npm

(24) *Pages* [online]. San Francisco: Vercel, 2022 [cit. 2022-08-14]. Dostupné z: https://nextjs.org/docs/basic-features/pages

(25) *GetStaticPaths* [online]. San Francisco: Vercel, 2022 [cit. 2022-08-14]. Dostupné z: https://nextjs.org/docs/basic-features/data-fetching/get-static-paths

(26) *GetServerSideProps* [online]. San Francisco: Vercel, 2022 [cit. 2022-08-14]. Dostupné z: https://nextjs.org/docs/basic-features/data-fetching/get-server-side-props

(27) *API Routes* [online]. San Francisco: Vercel, 2022 [cit. 2022-08-14]. Dostupné z: https://nextjs.org/docs/api-routes/introduction

(28) *Next/image* [online]. San Francisco: Vercel, 2022 [cit. 2022-08-15]. Dostupné z: https://nextjs.org/docs/api-reference/next/image

(29) *Next/link* [online]. San Francisco: Vercel, 2022 [cit. 2022-08-15]. Dostupné z: https://nextjs.org/docs/api-reference/next/link

(30) KESHAVARZ, Samira. *Analyzing Performance Differences Between MySQL and MongoDB* [online]. Kolín nad Rýnem, 2021 [cit. 2021-12-01]. Dostupné z: https://www.researchgate.net/publication/349764039. Výzkumný projekt. Technische Hochschule Köln.

(31) BIERER, Doug. *MongoDB 4 Quick Start Guide*. 1. Birmingham, UK: Packt Publishing, 2018. ISBN 978-1-78934-353-3.

(32) SANTACROCE, Ferdinando. *Git Essentials - Second Edition*. 1. Birmingham, UK: Packt Publishing, 2017. ISBN 9781787120723.

(33) *Webhosting, VPS nebo dedikovaný server?* [online]. Hluboká nad Vltavou: WEDOS, 2020 [cit. 2021-11-15]. Dostupné z: https://www.wedos.cz/hosting-srovnani

(34) DABIT, Nader. *Full Stack Serverless* [online]. 2. Sebastopol: O'Reilly Media, Inc., 2020 [cit. 2021-11-17]. ISBN 978-1-492-05989-9. Dostupné z: https://learning.oreilly.com/library/view/full-stack-serverless/9781492059882

(35) Serverless vs. Microservices Architecture: What Does The Future of Business Computing Look. In: *ByteAnt* [online]. Lviv: ByteAnt, 2020 [cit. 2022-03-08]. Dostupné z: https://www.byteant.com/media/b3pfcb2q/traditional-vs-serverless-architecture.png

(36) SARKIS, Anthony. *Training Data for Machine Learning* [online]. 2. Sebastopol: O'Reilly Media, Inc., 2021 [cit. 2021-11-16]. ISBN 978-1-492-09445-6. Dostupné z: https://learning.oreilly.com/library/view/training-data-for/9781492094517

(37) SYDORENKO, Iryna. Machine Learning and Training Data: What You Need to Know. *Label Your Data* [online]. 2021 [cit. 2021-11-17]. Dostupné z: https://labelyourdata.com/articles/machine-learning-and-training-data

(38) JOBY, Amal. *What Is Training Data? How It's Used in Machine Learning* [online]. 2021 [cit. 2021-11-17]. Dostupné z: https://learn.g2.com/training-data

(39) Interesting Uses of Image Recognition Technology in AI. In: *Datafloq* [online]. Haag: Datafloq, 2021 [cit. 2022-03-04]. Dostupné z: https://datafloq.com/wp-content/uploads/2021/12/blog_pictures2Fimage_annotations.jpeg

(40) In: *Stardust AI* [online]. [cit. 2022-03-13]. Dostupné z: https://stardust.ai/static/d176e02d98a384a5df5ac08d27f91b85/36530/TechSolution_CuboidLocalization_2.png

(41) Computer Vision Annotation: Tools, Types, and Resources. In: *TELUS International* [online]. Vancouver: TELUS International, 2021 [cit. 2022-03-03]. Dostupné z: https://images.ctfassets.net/3viuren4us1n/1wVRkllWhgwfTDjlvU4dLN/6c474dfd882a1797383c00e1ad828509/2020-01_behind-computer-vision.webp

(42) *Scale AI* [online]. San Francisco: Scale AI, 2022 [cit. 2022-01-07]. Dostupné z: https://scale.com/

(43) *Labelbox* [online]. San Francisco, California, United States of America: Labelbox, 2022 [cit. 2022-01-07]. Dostupné z: https://labelbox.com/

(44) *Kili Technology* [online]. Paříž: Kili Technology, 2022 [cit. 2022-01-07]. Dostupné z: https://cloud.kili-technology.com/docs

(45) 2021 Developer Survey. *2021 Developer Survey* [online]. New York: Stack Exchange, 2021 [cit. 2022-01-09].

(46) *Compare Hosting & Deployment Platforms* [online]. Wroclaw: Benjamas, 2021 [cit. 2022-02-15]. Dostupné z: https://bejamas.io/compare/

(47) *File Storage & Uploads* [online]. San Francisco: Vercel, 2021 [cit. 2022-01-09]. Dostupné z: https://vercel.com/docs/concepts/solutions/file-storage

(48) *Cloud Storage pricing* [online]. Kali: California, 2022 [cit. 2022-01-16]. Dostupné z: https://cloud.google.com/storage/pricing

(49) *Amazon S3 pricing* [online]. Seattle: Amazon, 2022 [cit. 2022-01-16]. Dostupné z: https://aws.amazon.com/s3/pricing/

(50) *Azure Blob Storage pricing* [online]. Redmond: Microsoft, 2022 [cit. 2022-01-16]. Dostupné z: https://azure.microsoft.com/en-us/pricing/details/storage/blobs/

Přílohy

Příloha A CD se zdrojovým kódem