Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Mikroslužby pro získávání zdravotnických dat

Bakalářská práce

Mykyta Belimenko

Školitel: PhDr. Miloš Prokýšek, Ph.D.

České Budějovice 2020

Belimenko, M., 2020: Mikroslužby pro získávání zdravotnických dat. [Microservices for Obtaining Health Data. Bc. Thesis, In Czech.] – 35 p., Faculty of Science, University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Cílem práce je vytvořit sadu mikroslužeb pro potřeby získávání dat v průběhu administrativních procesů Ministerstva zdravotnictví ČR. Tato práce zahrnuje analýzu, BPMN diagram, dokumentaci a vzorky zdrojového kódu.

Abstract

The aim of the thesis is to create a set of microservices for the purposes of obtaining data during the administrative processes of Ministry of Health of the Czech Republic. This thesis contains an analysis, documentation and samples of source code.

Prohlášení

Prohlašuji, že svoji bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne 06. 12. 2020 ……..

Mykyta Belimenko

Cíle práce:

Cílem práce je vytvořit sadu mikroslužeb pro potřeby získávání dat v průběhu administrativních procesů Ministerstva zdravotnictví ČR. Úkolem studenta je popsat příslušné procesy (BPMN), ve kterých budou mikroservisy nasazeny. Následně definovat požadavky na mikroslužeb kladené a popsat jejich vzájemná rozhraní (UML). Řešitel provede rešerši stávajícího řešení a zhodnotí výhody a nevýhody nasazení mikroslužeb. Součástí práce budou bezpodmínečně následující dokumenty:

- Zadávací dokumentace
- Analýza funkčních a nefunkčních požadavků
- Dokumentace kódu
- Testovací dokumentace
- Uživatelská dokumentace

Poděkování

Chtěl bych poděkovat panu PhDr. Miloši Prokýškovi, PhD., za odborné vedení práce, cenné rady a vstřícnost při konzultacích.

Obsah

1 Úvod

Téma této bakalářské práce je vytvořit sadu mikroslužeb pro potřeby získávání dat v průběhu administrativních procesů Ministerstva zdravotnictví České republiky (MZČR). Především se jedná o mikroslužby pro nahrávání souborů k žádostem, mikroslužby pro platební bránu a mikroslužbu zajišťující vizualizaci dat a řízení procesů.

Zákon č. 95/2004 Sb.[1] vyžaduje, aby zařízení, která organizují vzdělávací programy ve zdravotnictví, byly akreditovány. Rovněž i vzdělávací programy pořádané těmito zařízeními musí být akreditovány, aby mohly být uznány jako způsobilé vzdělávací programy k výkonu povolání ve zdravotnictví.

O akreditaci se žádá pomocí žádosti. §14 zákona č. 95/2004 Sb. uvádí „Žádost se předkládá ministerstvu v listinné podobě; přílohou listinného podání je i elektronická podoba žádosti na nosiči dat."

Ministerstvo zdravotnictví dnes provozuje izolované a uzavřené aplikační řešení, které není propojeno na ekosystém ministerstva a bylo vytvořeno pomocí interních zdrojů ministerstva. Tato aplikace je postavena na skriptovacím jazyku PHP, je provozována bez databázového úložiště a slouží pouze k sestavení žádosti o akreditaci ve formátu PDF, která je dále zpracována fyzicky. Aplikace je provozována na Linuxové platformě mimo vnitřní infrastrukturu ministerstva. Aktuálně je tato aplikace dostupná na https://akrform.mzcr.cz.

Hlavním účelem stávající aplikace je provést uživatele vyplněním formuláře tak, aby se minimalizovala pravděpodobnost formálních chyb při vyplňování. Ministerstvo zdravotnictví v současnosti nenabízí jinou variantu vyplnění formulářů (např. prostřednictvím vzorů ke stažení ve Wordu apod.).

Nedostatek současného řešení rovněž spočívá v nedostatečné podpoře celého procesu podání žádosti o akreditaci. Rovněž chybí možnost elektronického podání s možností uhrazení správního poplatku. Dále se nabízí i možnost vytvoření interního systému, který by jednotlivé žádosti evidoval a administroval.

[1]Zákon č. 95/2004 Sb. o podmínkách získávání a uznávání odborné způsobilosti a specializované způsobilosti k výkonu zdravotnického povolání lékaře, zubního lékaře a farmaceuta

Výsledným stavem má být plnohodnotné podávání žádostí skrze prostředky elektronické komunikace, včetně úhrady správního poplatku skrze online platební metody (např. platba kartou).

Další motivací je zvýšení procenta úspěšných žádostí bez nutnosti přerušovat řízení na základě chyb, kterého se žadatelé ve svých žádostech dopouštějí. S vyšší mírou bezchybných žádostí se rovněž sníží pracovní nároky na práci ministerských úředníků (přerušování řízení, archivace papírových spisů, strukturované informace).

Struktura výsledného formuláře, bude odpovídat logickému členění povahy nahrávaných dokumentů. To uživatele intuitivně povede k eliminaci situací, kdy žádost je podána nekompletní. V neposlední řadě je elektronické podání šetrnější k životnímu prostředí a je zpracováno rychleji.

Vzhledem k problematickému zdravotnickému prostředí, je tato práce orientována na realizaci řešení pomocí mikroslužeb, které budou opakovatelně využitelné i v jiných aplikacích Ministerstva zdravotnictví a zároveň umožní postupný rozvoj řešení. Příkladem je mikroslužba platební brány, která se přímo nabízí pro využití ve více správních agendách.

Využití mikroslužeb je dle sdělení konzultanta, se kterým byl celý koncept práce řešen, i v souladu se záměrem Ministerstva zdravotnictví, které plánuje výstavbu mikroslužeb, které budou navzájem komunikovat přes definovaná rozhraní. Rozdělení jednotlivých částí procesu žádostí o akreditaci do mikroslužeb je plánováno z těchto důvodů:

- Rychlejší nasazování jednotlivých komponent, neboť není třeba čekat na dokončení jedné velké aplikace, ale lze postupně zapojovat dílčí části.
- Využitelnost mikroslužeb i pro účely jiných správních agend (např. již zmíněné využití platební brány).
- Měnící se požadavky ze strany MZČR a dosavadní zkušenost s monolitickými aplikacemi

Přístup ke službám bude realizován pomocí webové aplikace a optimalizován pro běžně dostupná zařízení (počítač, tablet, mobil atp.).

Cíle této práce je vytvoření funkční soustavy mikroslužeb, které dají poskytovatelům zdravotních služeb možnost plnohodnotného podávání žádostí skrze prostředky elektronické komunikace.

První část práce je zaměřena na analýzu relevantních mikroslužeb, formulaci logického rámce realizace, specifikaci požadavků a návrhem architektury aplikace. Tato část práce slouží zároveň jako zadávací dokumentace pro druhou, implementační, část práce.

Ve druhé části práce je popsán proces vývoje aplikace od kódování, testování až po nasazení do produkčního provozu.

Výběr technologií pro implementaci řešení je založen na zkušenostech autora práce a jejich relevance je zároveň podpořena v přiměřené míře v první části práce. Vzhledem k rozsahu řešení a výslednou implementaci provedenou pouze jedním vývojářem byla metodika vývoje částečně odvozena od metodiky SCRUM, se kterou má autor praktické zkušenosti.

Zuzana Šochová [1] tvrdí, že „SCRUM je proces postavený na týmové spolupráci, získávání časté zpětné vazby a transparentní komunikaci v rámci týmu a firmy, ale i směrem k zákazníkovi.“. To znamená, že na vývoji a nasazení produktu participuje několik vývojářů, testeru, včetně vlastníka produktu a tzv. scrum mastera. Vývoj je členěn do řadu kratších, jasně definovaných úseku, které se označují jako sprint.

Z této metodiky autor převzal především úzké spojení s konzultantem ze strany MZČR a krátkou iterativní formu vývojových cyklů, se získáním okamžité zpětné vazby.

2 Analýza a specifikace

2.1 Logický rámec

Tabulka 1 Logický rámec projektu

Popis projektu (Cil)	**Objektivně ověřitelné ukazatele**	**Zdroje k ověření**	**Předpoklady**
Zefektivnění zpracování správních podání pomocí mikroslužeb.	Dostupná a funkční aplikace na webu.	Provozní statistiky systému. Slovní vyjádření a odezva od pracovníků Ministerstva zdravotnictví ČR.	
Účel projektu	**Objektivně ověřitelné ukazatele**	**Zdroje k ověření**	**Předpoklady**
Integrace mikroslužeb do navazujícího procesu pro zpracování správních žádostí podávaných žadateli z řad veřejnosti	Reálné využití aplikace poskytovateli zdravotních služeb (pilotní běh aplikace).	Nahrávané dokumenty skrze nahrávací komponentu. Uhrazené transakce skrze platební bránu. Kontrola dat skrze vizualizační komponentu.	Stručně a jasně stanovené požadavky na systém.
Výstupy	**Objektivně ověřitelné ukazatele**	**Zdroje k ověření**	**Předpoklady**
Technická dokumentace Uživatelská dokumentace Nahrávací komponenta (webová aplikace) pro nahrávání/podání formulářů.	Funkční systém na kveten 2021 Pochopitelní testovací a uživatelská dokumentaci.	Testování aplikace. Dostupná uživatelská dokumentace na webu. Dostupná technická dokumentace v projektu.	Zájem a kladné hodnocení projektu ze strany ministerstva zdravotnictví a uživatelů.

„Savage“ komponenta (webová aplikace) pro správu uživatelů.			
Klíčové činnosti	**Prostředky / Vstupy**	**Harmonogram**	**Předpoklady**
1. Testování kódu 2. Analýza funkčních a nefunkčních požadavků 3. Návrh a implementace nahrávací mikroslužby 4. Návrh a implementace mikroslužby pro správu uživatelů. 5. Návrh použiti platební brány. 6. Zprovoznění OwnCloudu, MongoDB a RabbitMq	Owncloud úložiště Comgate vývojové a testovací prostředí. IDEA Intellij Reálné produkční data ve formátu JSON Server Architektura běžného systému	Březen 2019 – Návrh BPMN a UML duben Květen–červen 2019: Analýza a návrh vhodných technologií a návrh jednotlivých mikroslužeb. Srpen–leden 2019: vývoj mikroslužeb Leden 2020: dokumentace technická (Readme.md) uživatelská (video) a testování	Dobře navržena architektura projektu. Zajištění běhového prostředí – serveru. Poskytnuti informace o běžném systému. Poskytnuti produkčních dat. Poskytnuti testovacího prostředí platební brány.

Zdroj: Vlastní zpracování

2.2 Funkční požadavky

Nahrávací komponenta „Upload“

Tato komponenta (mikroslužba) přijímá na vstupu data formátu JSON[2] ve stanovené struktuře přes REST API[4]. Mikroslužba aktivuje vygenerování nahrávacího formuláře pro podání více souborů najednou. Současně zobrazí výši správního poplatku vážícího se na danou agendu, kontaktní údaje apod. Příklad je uveden v kapitole „3.8 Databázový model“.

Před odesláním dat probíhá validace na stanovená pravidla, např. omezení na velikost souboru, nebo kontrola povinných souborů atd.

Po odeslání formuláře je očekáván následující stav:

- Dokumenty jsou nahrány do cloudového úložiště Owncloud.
- Změna aktuálního stavu procesu uživatele v systému.
- Uživatel je přesměrován na platební bránu s daty.

Komponenta „Savage“

Pracovník Ministerstva zdravotnictví může zkontrolovat stav každého uživatele a jeho procesů pomocí mikroslužby „Savage“. Mikroslužba „Savage“ na webu zobrazuje aktuální stav a informace o uživateli na základě informací o něm vedených v databázi.

Pracovník Ministerstva zdravotnictví v roli administrátora procesu může též na webu manuálně upravit všechny výše uvedené údaje.

2.3 Nefunkční požadavky

Platební brána – po nahrání formulářů může být uživatel přesměrováván na platební bránu ComGate, kde se provede online platba kartou nebo převodem. Po výstupu z platební brány ComGate uživatel obdrží potvrzení o zaplacení.

Dostupnost – jednotlivé mikroslužby budou zapouzdřeny do tzv. docker kontejneru a následně nahrány na server mající internetovou konektivitu.

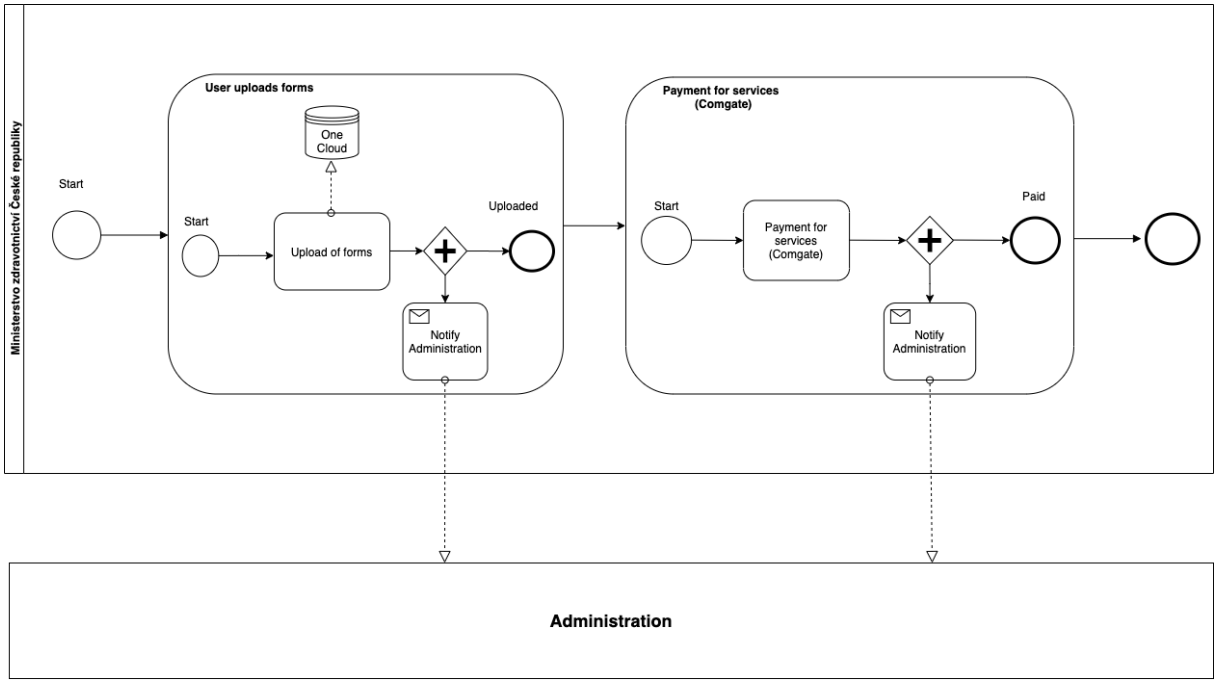
[2] JSON - textový formát pro výměnu dat založený na JavaScriptu.
[4] Representational State Transfer / Application programming interface – je architektura rozhraní aplikací, navržená pro distribuované prostředí.

2.4 Business Process Model and Notation

Pro popis implementovaných procesů byl zvolen procesní diagram v standardu BPMN 2.0, který se používá se zejména pro detailní modelování, analýzu a redesign procesů podnikové architektury, která je známá také jako Enterprise Architecture (EA). [2]

Hlavním cílem BPMN diagramu je jednoduše popsat všechny implementované procesy, a tak aby snadno srozumitelné celému týmu, tj. od business analytiků, kteří vytváří prvotní návrhy procesů a konče vývojářem, který bude systém implementovat.

Obrázek 1 Scénář použití aplikace BPMN diagram



Zdroj: Vlastní zpracování

3 Návrh systému

Před zahájením vývoje aplikace bylo nutné zvolit vhodné technologie a správnou architekturu, která umožní dosáhnout maximální úrovně interaktivity a adaptability. Za dvě základní architektonická paradigmata lze považovat tzv. „Monolitics Architecture“ a „Microservice Architecture“. Níže jsou obě popsána a jsou uvedeny hlavní rozdíly, které vedli k přijmutí architektury mikroslužeb, jako vhodného paradigmatu pro tuto implementaci.

3.1 Monolithic vs. Microservices Architecture

Monolitická architektura představuje jednostupňovou softwarovou aplikaci, ve které se různé komponenty kombinují do jednoho programu, z jedné platformy. Komponenty mohou být:

- Prezentační služby / UI– zodpovídá např. za zpracování požadavků HTTP[7] a odpověď buď pomocí HTML[8], nebo JSON / XML[9] (pro API webových služeb).
- Business logika – Vlastní logika aplikace a logika dat, často též v kombinaci s prezentační logikou
- Data Access Layer – objekty přístupu k datům odpovědné za přístup k databázi a perzistenci dat.

Tabulka 2 Výhody a nevýhody monolitické architektury

Výhody	Nevýhody
Snadná implementace	Obtížná udržitelnost velkého monolitického bloku
E2E testy	Nutnost opakovatelného sestavování i po sebemenší změně
Snadné nasazení	Dlouhé spouštění v případě, že projekt je rozsáhlý
Lepší výkon	Obtížná migrace na novou technologii

[7] Hypertext Transfer Protocol - internetový protokol určený pro komunikaci s WWW servery.
[8] HyperText Markup Language – textový značkovací jazyk.
[9] eXtensible Markup Language - rozšiřitelný značkovací jazyk.

Vojtěch Hordějčuk [3] tvrdí, že „Architektura založená na mikroslužbách (microservice architecture) popisuje návrh složitého systému pomocí malých, jednoduchých a nezávisle běžících komponent, nazývaných mikroslužby. Tyto komponenty spolu mohou vzájemně komunikovat, nejčastěji pomocí rozhraní postavených nad HTTP.“ Používá rozdělení jednoho velkého projektu na moduly. Každá komponenta podporuje konkrétní „business cíl“. Mikroslužby mají následující charakteristiky:

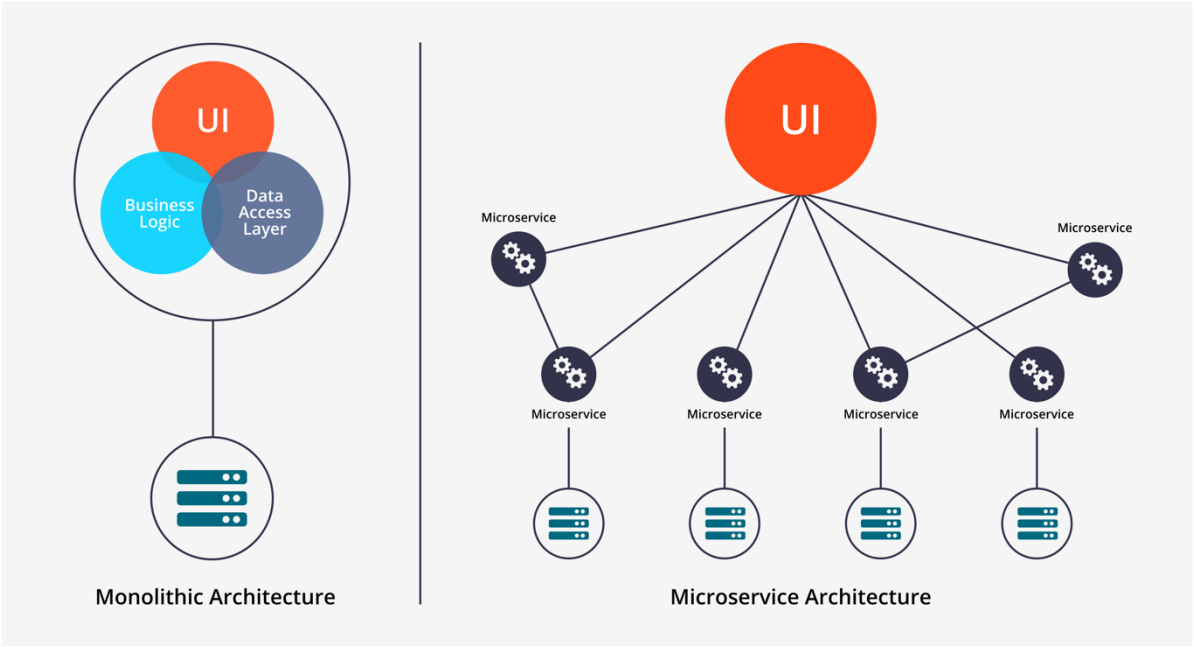
- Jsou malé.
- Lze je nasadit nezávisle na sobě.
- Jsou zaměřeny na jednotlivé business funkčnosti.
- Jsou volně spojené.
- Mohou být implementovány pomocí různých programovacích jazyků.

Místo sdílení jedné databáze jako v monolitické aplikaci může mít každá mikroslužba vlastní databázi, a tedy zvolit nejlepší DB pro každou z nich. Tedy v závislosti na vykonávané činnosti např. zvolit mezi SQL, NoSQL či dalšími tipy databází.

Tabulka 3 Výhody a nevýhody mikroslužeb

Výhody	Nevýhody
Snadná podpora	E2E testy
Správný nástroj pro každý úkol	Obtížné nasazování
Distribuce práce mezi týmy	Zvýšená spotřeba prostředků
Horizontální škálování a odolnost proti chybám	Komunikace mezi ostatní moduly (API)
Schopnost opravit, vyměnit a vylepšit modul	Obtížné udržování závislostí verzí mezi mikroslužbami
Snížení času pro nasazení a schopnost to udělat paralelně	

Obrázek 2 Monolitická architektura proti Mikroslužeb



Zdroj: [Medium [https://medium.com/startlovingyourself/microservices-vs-monolithic-architecture-c8df91f16bb4]](https://medium.com/startlovingyourself/microservices-vs-monolithic-architecture-c8df91f16bb4)

3.2 Výběr vhodného frameworku

Framework je sbírka mnoha funkcí (ideálně ale tříd a objektů), které programátorovi šetří práci. [4] Pro hodnocení a výběr byla využita data z komunitního webu Stack Overflow [5] a zvažovány byly aktuální nejpopulárnější opensource frameworky [https://existek.com/blog/top-front-end-frameworks-2020/], a to:

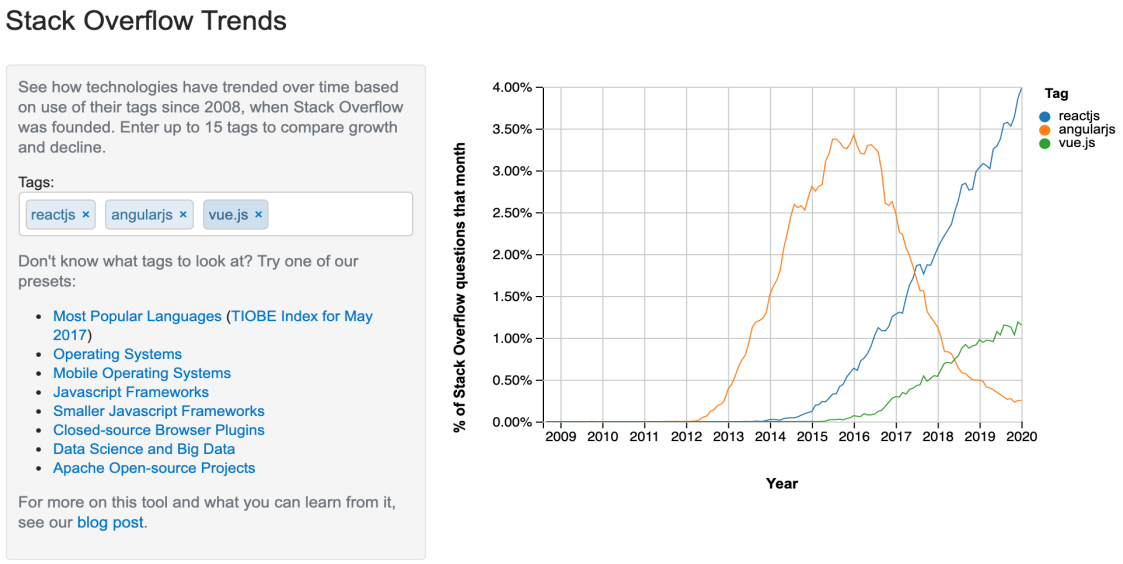
- React.js
- Angluar.js
- Vue.js

Jako kritéria výběru byly zvoleny následující ukazatele:

- Podpora komunity vývojářů.
- Výkon.
- Velikost.

Podpora komunity vývojářů je podle webu Stack Overflow je uvedena obrázku 3. Z obrázku je patrné, že Vue.js a React.js jsou podle tohoto webu perspektivní

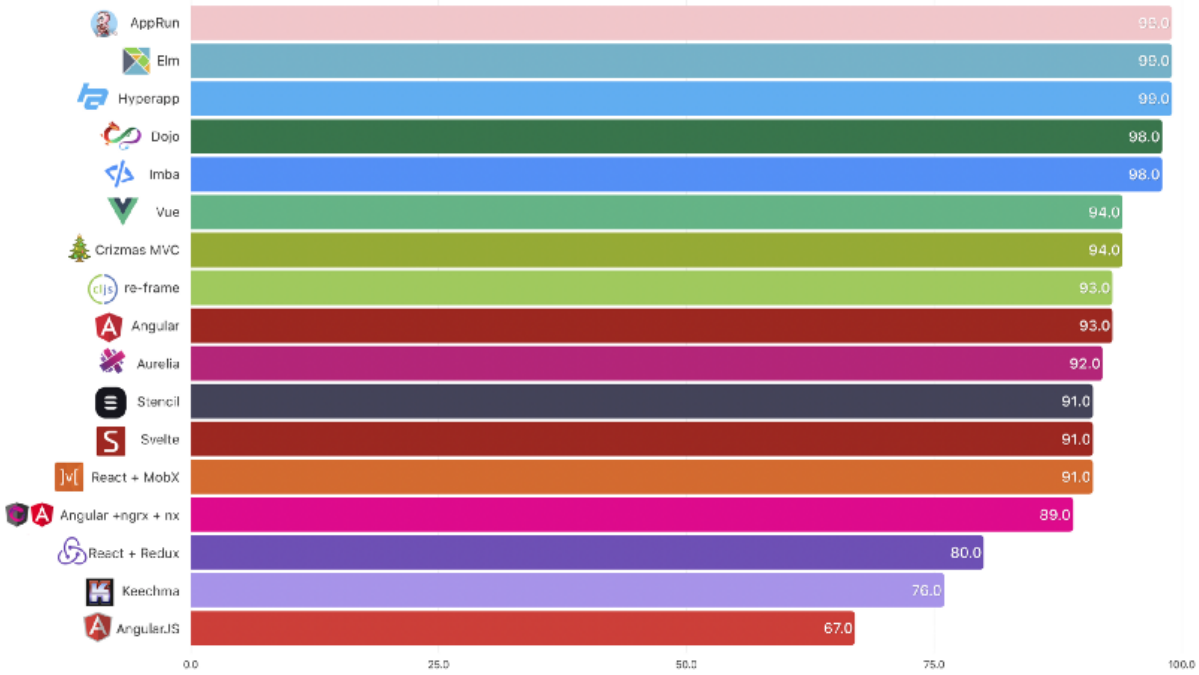
Obrázek 3 Trendy frameworků



Zdroj: Web Stack Overflow [https://insights.stackoverflow.com/trends?tags=reactjs%2Cangularjs%2Cvue.js]

Hodnocení **výkonu** je na obrázku 4. Zde můžete vidět statistiky výkonu všech frontendových frameworků (použité hodnocení je na škále 0-100, větší číslo znamená lepší výkon):

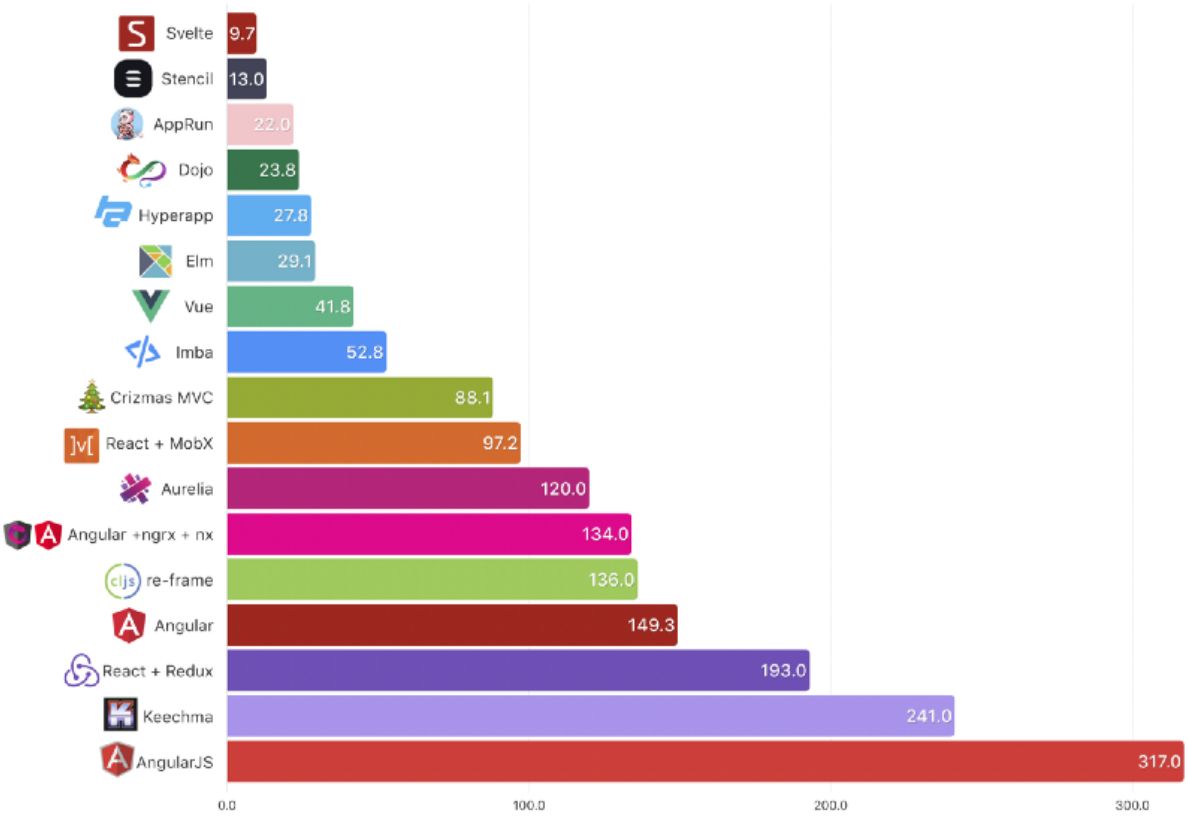
Obrázek 4 Výkon frameworků



Zdroj: Frameworks performance [https://www.freecodecamp.org/news/a-realworld-comparison-of-front-end-frameworks-with-benchmarks-2019-update-4be0d3c78075/]

Na obrázku číslo 5, je zobrazeno porovnání **velikosti** každého frontendového frameworku. Hodnocení čím menší, tím lepší.

Obrázek 5 Velikosti frameworků



Zdroj: Frameworks size [https://www.freecodecamp.org/news/a-realworld-comparison-of-front-end-frameworks-with-benchmarks-2019-update-4be0d3c78075/]

Jako pomocné kritérium hodnocení autor subjektivně přidává **dokumentaci** frameworku, která je dle jeho názoru nejkvalitnější u Vue.js. Z výše uvedeného vyplývá, že všechny frameworky mají dobrou pozici v komunitě a jsou výkonově srovnatelné. Z hlediska velikosti dopadá lépe Vue.js. **Na základě tohoto hodnocení si autor práce pro implementaci řešení zvolil framework Vue.js**

3.3 Přehled vybraných technologií

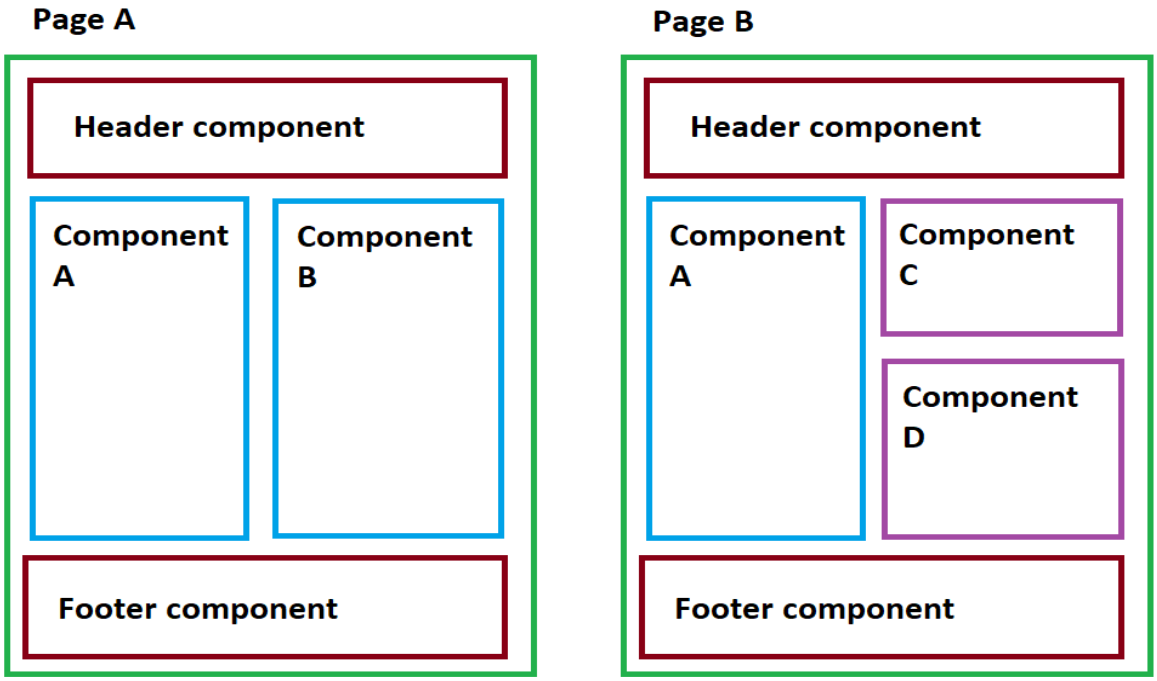
Pro implementaci řešení byl použit systém vzájemně provázaných technologií. Níže je uvedená jejich stručná charakteristika, jejímž účelem je seznámit čtenáře z jejich hlavními rysy.

Vue.js

Vue.js (též označovaný jako View) framework je využit především pro oblast prezentační. Je snadné jej integrovat do jiných knihoven a projektů. Vue.js je díky svým vlastnostem řazen mezi progresivní frameworky, což znamená, že umožňuje webovou aplikaci rozdělit na několik částí a ty pak vyvíjet nezávisle na sobě. Pokud tak chcete Vue použít v rámci již existující aplikace, nemusíte nutně znovu kódovat všechny stránky, případně komponenty aplikace. Jedná se o univerzální a výkonné řešení, jež napomáhá udržování pořádku v kódu. [6]

Centrální koncept Vue.js je koncept komponent, tj. malých částí uživatelského rozhraní, které lze znovu použít Samotná aplikace se tedy skládá z komponent. Jedna komponenta může zahrnovat několik dalších komponent. Komponenty tvoří strom (obrázek 6).

Obrázek 6 Vue.js komponenty



Zdroj: Vue.js components [https://i.stack.imgur.com/WRZip.png]

Node.js

Node.js je prostředí umožňující spouštět JavaScript kód mimo webový prohlížeč.“. Předtím, Javascript používali jen na straně klienta. Pomocí Node.js bylo možné na serveru spustit kód

Javascript. S Node.js můžete psát plnohodnotné aplikace. Node.js může pracovat s externími knihovnami – npm[10], vyvolávat příkazy z kódu JavaScript a fungovat jako webový server. [7]

Výhody Node.js:

- Node.js umožňuje běh JavaScriptu na základě modulu JavaScript V8 z prohlížeče Chrome.
- Node.js používá událostně řízený, neblokující I/O[11] model, který usnadňuje a zefektivňuje.
- Ekosystém balíčku Node.js, npm, je největší ekosystém knihoven s otevřeným zdrojovým kódem na světě.

Jak už bylo zmíněno, Node.js je schopen v případě rozšíření o další knihovny vystupovat v roli webového serveru. Pro tento projekt byla k tomuto účelu využita nadstavba Express.js (viz dále).

Express.js

Express.js je v současné době nejpopulárnější nadstavbou Node.js (dle NPM). Tento framework běží na http protokolu, který je zabudován do Node.js. Express.js obaluje vestavěný http protokol v Node.js s jeho shellem a poskytuje další a snadnější funkce pro vývoj REST API, tím umožňuje mnohem jednodušším a čistším způsobem rozdělovat robustní API a webové servery. Instalace Express.js je velmi jednoduchá. Jednoduše jej nainstalujte pomocí npm stejně jako u jakéhokoli jiného balíčku.

Mongoose a MongoDB

MongoDB [8] je dokumentová databáze - NoSQL[12]. Dokumentové databáze ukládají data jako ucelené dokumenty, často ve formátech XML, JSON či BSON[13]. Tyto dokumenty jsou sebe–popisující stromové datové struktury, které mohou sestávat z map, kolekcí a skalárních hodnot.

[10] Node.js package manager - správce balíčků pro JavaScript
[11] I/O - Vstup/Výstup
[12] NoSQL je databáze, které nejsou založené na relačním datovém modelu
[13] BSON je binární datový formát.

Příkladem dokumentu ve formátu JSON může např. být:

Obrázek 7 Struktura JSON

```
[
   {"name":"Ram", "email":"Ram@gmail.com"},
   {"name":"Bob", "email":"bob32@gmail.com"}
]
```

Zdroj: JSON Example [https://www.javatpoint.com/json-example]

Jednou z hlavních vlastností MongoDB je flexibilita jeho datové struktury. To odlišuje MongoDB od databází SQL[14], například MySQL nebo Microsoft SQL Server, ve kterém je pro každý objekt uložený v databázi zapotřebí pevné schéma.

Výhody MongoDB:

- Flexibilní struktura dat.
- Sharding[15]
- Vysoká rychlost.
- Vysoká dostupnost.
- Škálovatelnost.
- Snadné nastavení prostředí.
- Plná technická podpora.

Mongoose je ODM[16], který umožňuje provádět CRUD operace nad MongoDB, resp. transformovat dokumenty uložené v databázi do objektové reprezentace. Mongoose poskytuje rozsáhlou sadu funkcí pro vytváření a práci se schématy. V tuto chvíli Mongoose obsahuje osm datových typů, tzv. SchemaTypes (datové typy schématu). Jedná se o tyto typy:

- String.
- Number.
- Date.
- Buffer.
- Boolean.

[14] Structured Query Language (strukturovaný dotazovací jazyk)
[15] princip návrhu databáze, kdy řádky databázové tabulky drženy odděleně, místo aby byly rozděleny do sloupců
[16] ODM - Object Data Modeling

- Mixed.
- ObjectId[17]
- Array.

Výhody Mongoose:

- Flexibilní směrování.
- Clustering.
- Podpora velkého množství klientů.
- Grafické rozhraní pro správu a trasování.
- Velká komunita.

RabbitMQ

Při komunikaci modulů mezi sebou na serverové straně byla původně zvažována přímá komunikace mezi moduly pomocí REST API. Při hlubší analýze byl však tento způsob shledán jako nesprávný, resp. uváděný jako anti-pattern. Především se jedná o úzkou závislost jednotlivých služeb, jejich časovou synchronizaci a řešení chybových stavů (např. pád služby apod.)

RabbitMQ je aplikace pro práci s frontami zpráv (message-queueing), které se také nazývají zprostředkovatel zpráv (message broker) nebo správce front (queue manager). Jednoduše lze říci, že se jedná o software, ve kterém lze definovat fronty, k nimž se mohou různé aplikace připojit a odesílat/přijímat zprávy (metoda Publisher/subscriber).

Posílání a přijímání zpráv probíhá asynchronně, to znamená, že klienti mohou s sebou navzájem komunikovat bez blokovacích operací, což samozřejmě zrychluje zpracování požadavku.

Zpráva může obsahovat jakékoli informace. Máme například informace o procesu/ úkolu, který by měla spustit/zpracovat jiná aplikace (nebo jiný server), nebo to může být jednoduchá textová zpráva. Správce front – aplikace, která ukládá zprávu, dokud se jiná aplikace (server), ke které je zpráva adresována, nepřipojí a nevybere (nepřijme) zprávu z fronty. Poté cílový přijímač zprávu zpracuje tak, jak potřebuje.

[17] unikátní identifikátor objektu, primary key, _id

Docker-compose a docker

Pro zjednodušení vývoje projektu byly využity tzv. Docker containers. Docker je technologie, která slouží k virtualizaci aplikací, operačních systémů atd. Jakýkoli obraz (image), může být zabalen do kontejneru a dále je s ním nakládáno jako se samostatnou funkční jednotkou. Například lze kopie stejného kontejneru nasadit paralelně vedle sebe, instanciovat kontejner na základě množství požadavků apod [9]. Docker umožňuje:

- Izolované spouštění aplikací v kontejnerech.
- Zjednodušení vývoje, testování a nasazení aplikací.
- Není nutné konfigurovat prostředí pro spuštění.
- Zjednodušuje škálovatelnost a správu aplikací pomocí systémů orchestrace kontejnerů.
- Můžete parametrizovat každou krabičku zvlášť.

Docker Compose je pomocný nástroj dodávaný s aplikací Docker. Je určen ke správě docker kontejnerů a k řešení problémů spojených s nasazováním projektů.

3.4 Architektura systému

Při vývoji aplikace byl využit tzv. MEVN stack. MEVN zahrnuje čtyři komponenty s otevřeným zdrojovým kódem: MongoDB, Express, Vue a Node.js. Tyto komponenty poskytují vývojářům komplexní vývojové prostředí.

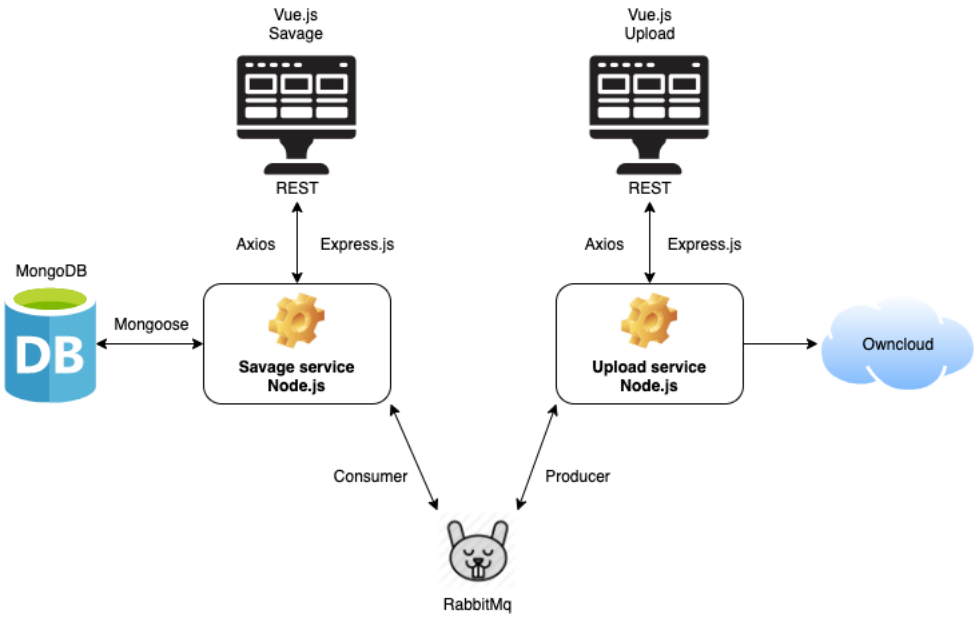
Hlavní výhoda pro vývojáře používající stack MEVN je, že každý řádek kódu je napsán v JavaScriptu, který je možné použít pro kód jak na straně klienta, tak na straně serveru.

Existuje velké množství takových technologických stacků. Příklady takových jsou MERN a MEAN. Jediný rozdíl v těchto hromádkách je použití jiných frontendových frameworku (Angular, React). S příchodem Vue.js se tedy objevil stack – MEVN.

Tyto stacky jsou velmi oblíbené, protože použití MEVN umožňuje vývojářům vytvářet vysoce výkonné webové aplikace.

Také autor práce v návaznosti na trendy využil jeden z nejnovějších přístupů pro vývoj aplikací, a to „Microservice architecture", která se nejčastěji používá na straně serveru, nicméně je možné nalézt i příklady, kdy byla tato architektura využita i na straně klienta [https://micro-frontends.org/].

Obrázek 8 Architektura systému



Zdroj: Vlastní zpracování

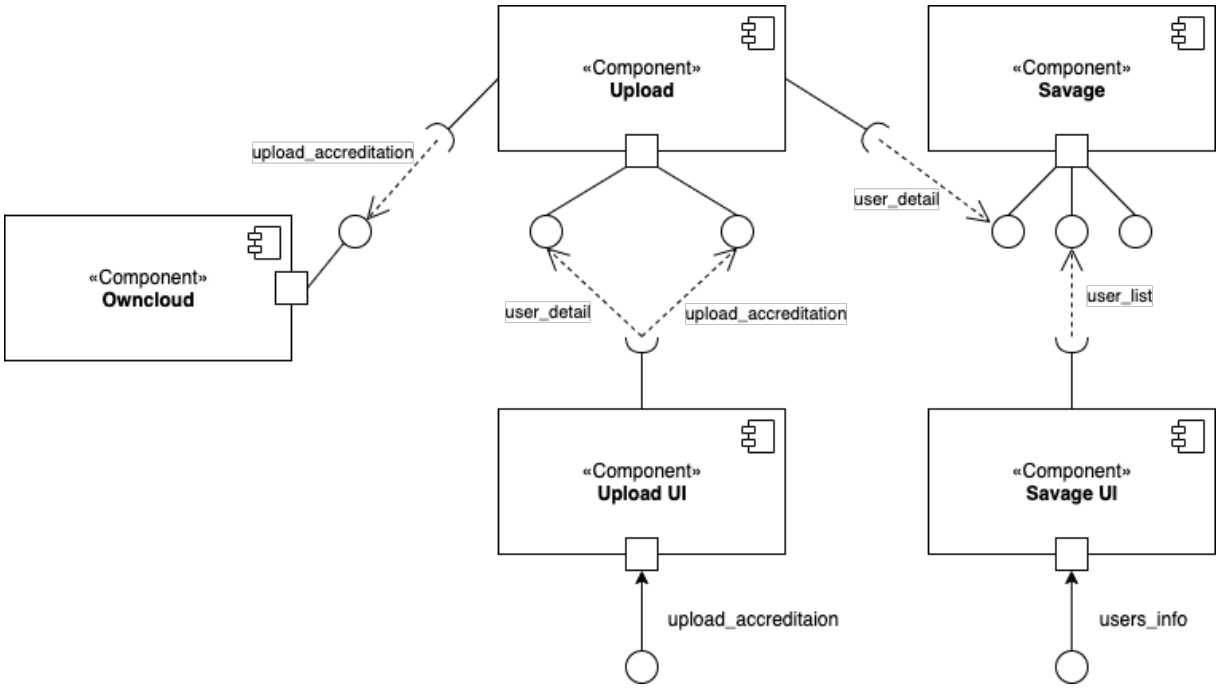
3.5 Component diagram UML

UML[18] vizuálně reprezentuje chování a strukturu systému nebo procesu a tím pomáhá odhalovat dopředu potenciální chyby, který mohou vzniknut v aplikačních strukturách, chování systémů a dalších podnikových procesech.[9]

Hlavně pro znázornění vazeb mezi jednotlivými komponentami byl využit UML diagram komponent, který patří mezi diagramy struktur protože nám může pomoci zejména tam, kde používáme vývoj založený na komponentách a kde struktura systému je založena na komponentách.[10]. UML diagram popisuje organizaci a zapojení fyzických komponent v systému.

[18] UML (Unified Modeling Language), využit standard 2.5

Obrázek 9 UML diagram



Zdroj: Vlastní zpracování

3.6 Serverová část

Savage komponenta

Tato komponenta je navržen tak, aby přijímal informace o uživatelích ve struktuře JSON a poskytoval je dalším modulům. Vedlejším cílem tohoto modulu je také správa uživatelů. Pomocí Mangoose je připojen k MongoDB a dokáže provádět CRUD operace.

Savage modul rovněž konzumuje zprávy od Upload modulu pomoci RabbitMQ, které poskytují informace o tom, zda uživatel úspěšné nahrál formuláře a pokročil na další krok procesu.

Upload komponenta

Upload komponenta zajišťuje nahrávání uživatelských souborů do Owncloud. Informace o uživatelích zjišťuje pomocí Savage module, který poskytuje tuto informace pomocí message brokeru – RabbitMq.

3.7 Klientská část

Savage komponenta

Tento modul byl vytvořen pomocí Vue.js proto, aby administrátorům Ministerstva zdravotnictví umožnil správu uživatelů.

Správce tak může provádět akce jako editace, prohlížení a mazání uživatelů ze systému. Tento modul lze přirovnat k aplikacím typu CRM (řízení vztahů se zákazníky).

Upload komponenta

Tato klientská část Upload komponenty je postaven rovněž na Vue.js a zobrazuje uživatelům rozhraní, které umožňuje podat (nahrát) žádost o akreditaci, nebo vzdělávací plán prostřednictvím webové stránky. Po úspěšném nahrání formulářů, upload modul pošle tyto formuláře na server.

3.8 Databázový model

Jak již bylo uvedeno výše v technické části, autor práce si pro ukládání dat zvolil databázi MongoDB. Důvodem bylo, že uživatelská data přicházejí ve struktuře JSON.

V době dokončení této práce měla databáze MongoDB pouze jednu kolekci (collection) – userAccreditations, která obsahuje informace o uživatelích a jejich aktuální stav. Dále obsahuje informace o souborech, které se budou ukládat do cloudového úložiště – Owncloud.

Vzhledem k tomu, že databáze nebyla Ministerstvem zdravotnictví pro práci na projektu poskytnuta, autor se rozhodl použít pro ukládání dat omezenou verzi – MLab a docílil tak plně funkční aplikace.

Níže je uveden vzorek akreditačních dat (obsah hlavičky „form" je generován dynamicky):


```
[
    {
      "info": {
        "name": "Jorshua Agastins",
        "requestId": "requestedId",
        "fee": 1000,
        "email": "uzivatel@example.com",
        "redirect": "/url-kam-budu-presmerovan",
        "folderName": "Fakultni_nemocnice_Motol-21.5.2019_1415",
        "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9"
```

```
      },
      "form": [
        {
          "groupName": "\u017d\u00e1dost o akreditaci",
          "items": [
            {
              "id": "zadost-o-akreditaci"
            },
            {
              "id": "vzdelavaci-plan",
              "label": "Vzd\u011bl\u00e1vac\u00ed pl\u00e1n",
              "maxSize": 2048,
              "description": "Zde p\u0159ilo\u017ete \u017e\u00e1dost o
akreditaci.",
              "acceptedFormats": ".doc,.pdf,.docx",
              "required": true,
            }
          ]
        },
        {
          "groupName": "\u017divotopisy l\u00e9ka\u0159\u016f",
          "items": [
            {
              "id": "zivotopis-fd65fd3",
              "label": "\u017divotopis: MUDr. Jan Nov\u00e1k",
              "acceptedFormats": "",
              "required": false,
              "tags": [
                "zivotopis"
              ],
            },
            {
              "id": "zivotopis-tu9zjhz3",
              "label": "\u017divotopis: MUDr. Jan Svoboda",
              "acceptedFormats": ".doc,.pdf,.docx",
              "required": false,
              "tags": [
                "zivotopis"
              ],
            }
          ]
        }
      ],
      "state": {
        "mz": false,
        "upload": false,
        "payment": false
      }
    }
  ]
```

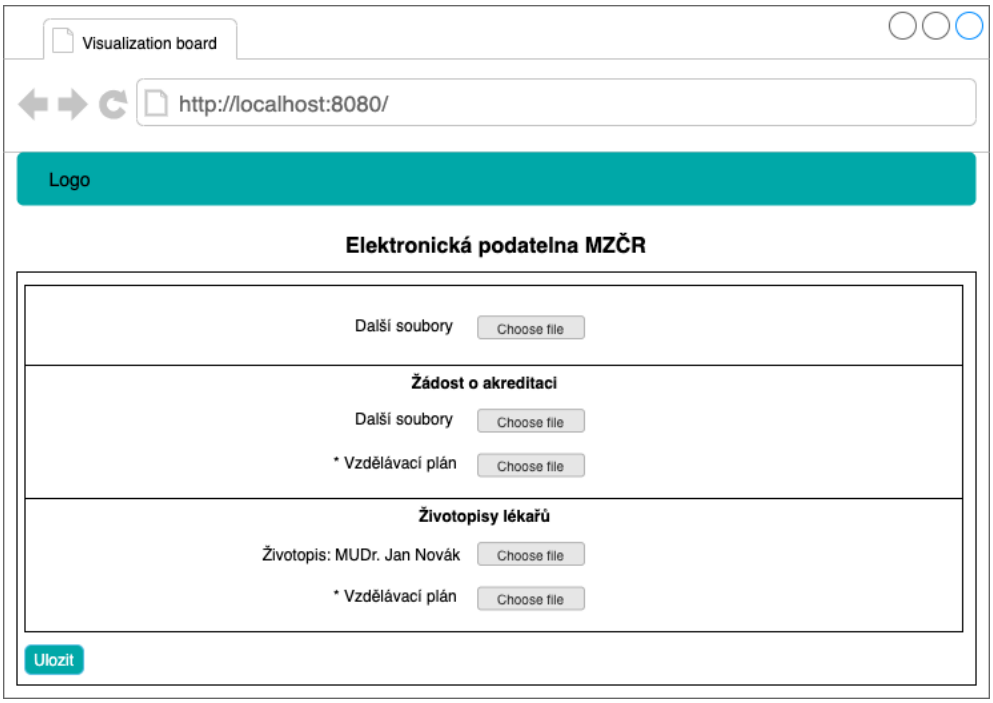
3.9 Uživatelské rozhraní

Existuje mnoho různých konceptů pro uživatelské rozhraní, například: material design, metro, skeuomorphism, bootstrap, quasar, ZURB Foundation atd. Výběr uživatelského rozhraní byl koordinován s Ministerstvem zdravotnictví. Nakonec bylo vybráno rozhraní Bootstrap.

Bootstrap – je jedním z nejpopulárnějších front-end frameworků pro vývoj responzivních a mobilních webů. Jak říká, Vitalij Petráš [11] "Jednou z největších výhod bootstrap frameworku je jeho dokumentace. Když píšete kód v souladu s bootstrapem a jeho funkcemi je snadno pochopitelný i pro ostatní. Je rozhodně příjemnější se zorientovat v udržovaném frameworku."

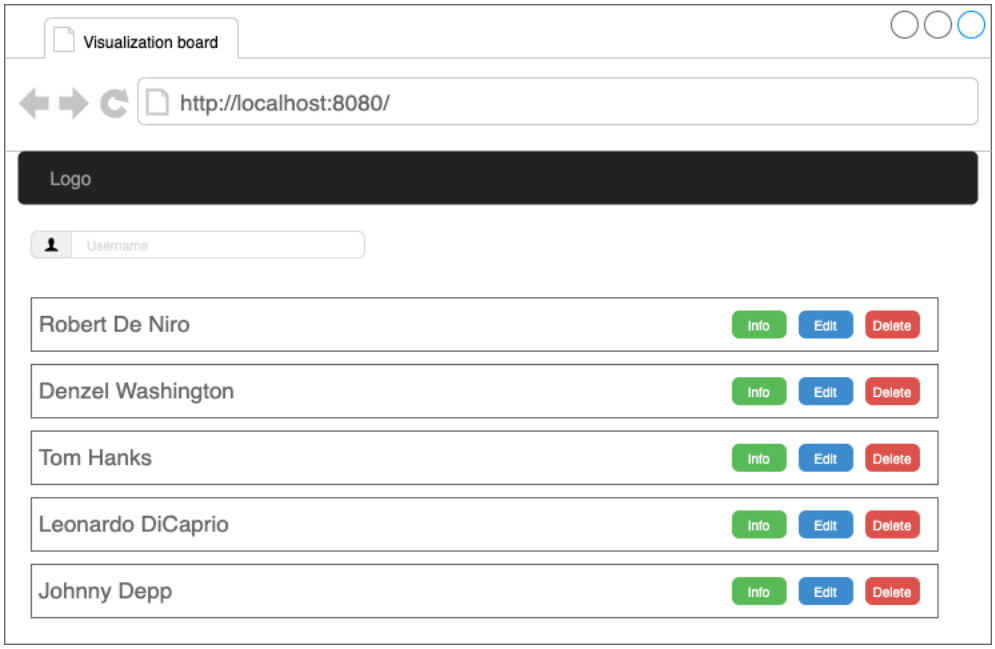
Pro vlastní design byl použit nástroj draw.io. Po četných jednáních a změnách byl schválen následující návrh:

Obrázek 10 Nahrávací komponenta – Uživatelské rozhraní



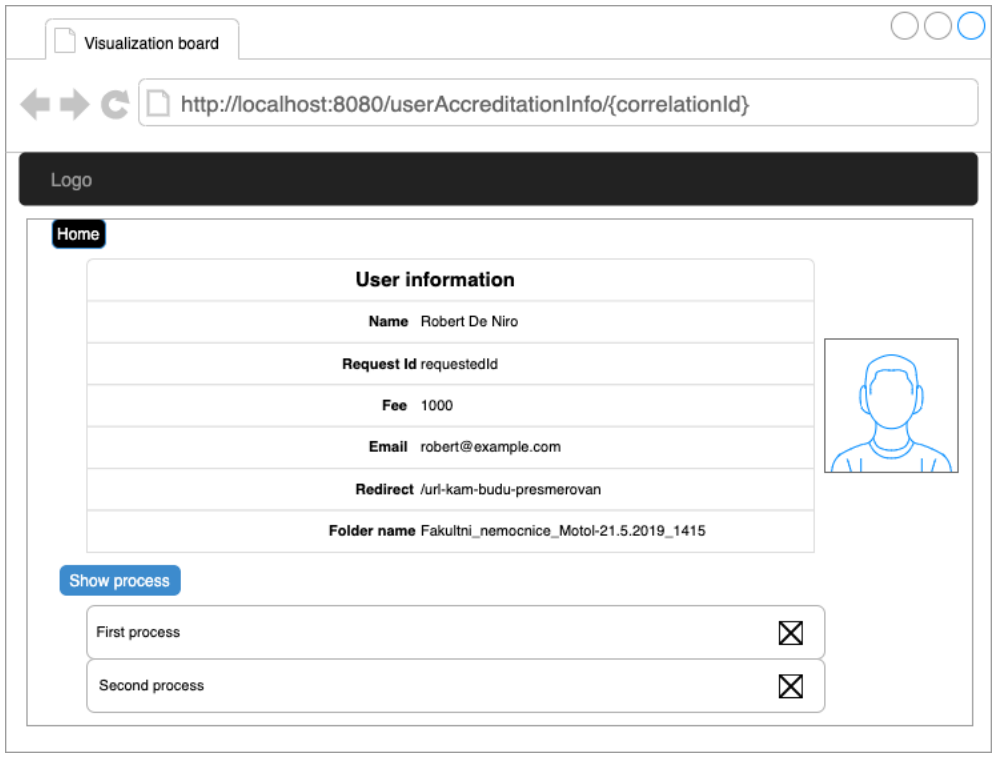
Zdroj: Vlastní zpracování

Obrázek 11 Seznam uživatelů– Uživatelské rozhraní



Zdroj: Vlastní zpracování

Obrázek 12 Informace uživatele – Uživatelské rozhraní



Zdroj: Vlastní zpracování

4 Implementace

Vzhledem k tomu, že se jedná o rozsáhlý projekt, tak v této kapitole není uveden celý kód, ale jen jeho stěžejní části, které jsou důležité pro popis fungování a architektury celého řešení.

Kopie kódu je k dispozici online pomocí služby bitbucket, což usnadňuje postupný vývoj projektu a zároveň zachovává soukromý přístup. V kapitole „7 Závěr“ je uveden odkaz na zdrojový kód a přístupové údaje.

4.1 Implementace Savage

Databázový server

Pro maximální komfort, bylo použito cloudové úložiště – MLab, jehož značnou výhodou je snadné ovládání. Pro ukládání dat se nemusí instalovat žádné další balíčky ani aplikace lokálně. Stačí kliknout na odkaz https://mlab.com a vytvořit si účet. Poté může uživatel bez znalosti příkazů ručně nakonfigurovat databázi pomocí uživatelského rozhraní.

Data access layer

Mongoose byl používán ke komunikaci mezi MongoDB (MLab) a aplikacemi. Tento nástroj je nutné nainstalovat pomocí následujícího příkazu:

```
npm i mongoose
```

Následně po instalaci je třeba importovat do aplikace:

```
const mongoose = require('mongoose');
```

Dále je třeba nakonfigurovat připojení ke vzdálenému cloudovému úložišti MLab

```
const DB_URL =
‘mongodb://<dbuser>:<dbpassword>@ds149676.mlab.com:49676/mz';

process.env.MONGODB = DB_URL;

mongoose.connect (process.env.MONGODB, {useNewUrlParser:
true});

mongoose.connection.on ('error', function () {
```

```
console.log('MongoDB Connection Error. Please make sure that
MongoDB is running.');

proces.exit(1); });
```

Následující kód vytvoří kolekci: UserAccreditaion

```
const userAccreditationSchema = mongoose.Schema ({

   correlationId: String,

   info: [],

   form: [],

   state: {},

   upload: {}

});

let userAccreditation = mongoose.model ('Accreditaion',
userAccreditationSchema);
```

Pokud chceme najít všechny dokumenty pro tento model, použijeme funkci .find()

```
userAccreditation.find();
```

Pokud však chceme najít konkrétního uživatele podle některých parametrů, můžeme použít *.findOne(),* pokud chceme provést změny v modelu *.findOneAndUpdate().* V naší aplikaci hledáme dokumenty pomocí correlationId parametru, který je unikátní pro každý dokument:

```
userAccreditation.findOneAndUpdate ({"correlationId":
correlationId}, body);
```

Chceme-li odstranit, použijeme funkci *.findOneAndDelete().* Pokud chceme vytvořit nové schéma, použijeme *.save().*

Application layer

Aby si frontend a backend mohli mezi sebou vyměňovat data, autor práce se rozhodl použít REST API rozhraní (na rozdíl od komunikace mezi komponentami).

Tato komunikace je založena na využití nadstavby Express.js (viz výše). Dále je uveden postup instalace a konfigurace.

Instalace:

```
npm install express –save
```

Nastavení serveru:

```
let app = express();

let server = require('http').createServer(app);

server = app.listen(3000, "127.0.0.1", () => {

console.log("Server listening on port:%s",
server.address().port);

});
```

Nyní, když máme server spuštěný na localhost: 3000, musíme přidat výše uvedené operace.

```
//receive accreditation object from MZ....

app.post('/userAccreditation', async (req, res) => {

   const result = await
userAccreditationService.createNewUser(req.body[0]);

   res.status(200).send(result);

});

//update user by id

app.put('/userAccreditation/id/:correlationId', async (req,
res) => {

   const body = req.body;

   const correlationId = req.params.correlationId;

   const result = await
userAccreditationService.findUserByCorrelationIdAndUpdate(cor
relationId, body);

   res.status(200).send(result); });
```

```
//delete user by id

app.delete('/userAccreditation/id/:correlationId', async(req,
res) => {
const body = req.body;|
```

```
const correlationId = req.params.correlationId;

   const result = await
userAccreditationService.deleteUserByCorrelationId(correlation
Id, body);

   res.status(200).send(result);

});

//get all users

app.get('/userAccreditationList', async (req, res) => {

   const result = await
userAccreditationService.getAllUsers();

   res.status(200).send(result);

});
```

Pro vzájemnou komunikaci mikroslužeb autor použil návrh RabbitMQ Publish/ Subscribe.

Publisher:

Instalace**:**

```
npm install amqplib
```

Nastavení message brokeru:

```
const amqp = require('amqplib');

const EventEmitter = require('events');
```

Definujeme frontu, na kterou očekáváme odpověď:

```
const REPLY_QUEUE = 'amq.rabbitmq.reply-to';
```

Definujeme frontu, na kterou posíláme zprávu:

```
const QUEUE_ACCREDITATION = 'direct_accreditation';
```

Žádáme o informace:

```
channel.responseEmitter.once("userAccreditation", resolve);
channel.sendToQueue("direct_accreditation", Buffer.from(
```

```
JSON.stringify(message)), {
message, replyTo: REPLY_QUEUE}
);
```

Definujeme kanál, kde budeme přijímat zprávy:

```
const createClient = () => amqp.connect('amqp://localhost')

    .then(conn => conn.createChannel()) // create channel

    .then(channel => {

        channel.responseEmitter = new EventEmitter();

        channel.responseEmitter.setMaxListeners(0);

        channel.consume(REPLY_QUEUE,

            msg =>
channel.responseEmitter.emit(QUEUE_ACCREDITATION,
msg.content),

            {noAck: true});

        return channel; });
```

4.2 Implementace Savage

Na straně „Savage" modulu nás zajímá implementace Subscriber a prezentační vrstvy.

Subscriber:

Instalace a nastavení je totožné „Upload" modulu.

Definujeme frontu, na kterou očekáváme zprávu, stejné jako v Publisher:

```
const QUEUE_ACCREDITATION = 'direct_accreditation';
```

Přijímáme zprávu:

```
amqp.connect('amqp://localhost')
    .then(connection => {
        return connection.createChannel();
    })
    .then(channel => {
        channel.assertQueue(queueAccreditation, {durable:
```

```
false});
        channel.prefetch(1);
        channel.consume(queueAccreditation, msg => {

            console.log("savage-server: Got msg from upload-
server: " + msg.content);)});
```

Posíláme odpověď:

```
channel.sendToQueue(msg.properties.replyTo,Buffer.from(JSON.s
tringify(result)));
```

4.3 Implementace Upload-UI

Pro dynamické generování položek autor práce vymyslel vlastní logiku,

```
<div v-for="form in forms" :key="form.id">
    <b-card>
        <br>
        <div class="row">
            <div class="col-sm">
                <h3 class="text-
center">{{form.groupName}}</h3>
            </div>
        </div>
        <div v-for="item in form.items" :key="item.id">
            <UploadForm :key="item.id"
                        :label="item.label"
                        :required="item.required"
                        :multiple="false"
                        :acceptFormats="item.acceptFormats"

@getFileContent="getFileFromChildComponent"
            />
        </div>
    </b-card>
</div>
```

kde pomocí cyklu procházíme pole „forms“ a vytváříme dynamické položky. Toto pole zároveň definuje strukturu dat. Další výhodou je, když Vue.js aktualizuje seznam prvků v poli vykresleném pomocí „v-for“, standardně používá strategii „in-place patch“ (opravy na místě). Pokud by se pořadí datových položek změnilo, namísto přesunu prvků DOM[20] tak, aby odpovídalo pořadí položek, Vue.js opraví každý prvek na místě a ujistí se, že reflektuje to, co

[20] Document Object Model - objektově orientovaná reprezentace XML nebo HTML dokumentu.

by mělo být vykresleno v dané pozici bez obnovení cely stránky. Toto se dá dosáhnout pomocí klíčového atributu „:key“, který by měl být vždy unikátní.

Řešeni CORS:

Pro práce s owncloud bylo třeba vyřešit problém s CORS. CORS[21] ”je technika využívající hlaviček protokolu HTTP, která umožňuje aplikacím běžícím ve webovém prohlížeči přistoupit ke zdroji (datům), který leží na jiné doméně než na které běží ona aplikace.je technika využívající hlaviček protokolu HTTP, která umožňuje aplikacím běžícím ve webovém prohlížeči přistoupit ke zdroji (datům), který leží na jiné doméně než na které běží ona aplikace.” [12] V projektu, autor práce používá REST API na komunikaci mezi modulem Upload a Owncloud uložištěm. Povolit CORS da se i ze strany serveru, a i ze strany klientu taky. Vue.js nabízí nejednoduší řešeni jak povolit CORS – stačí přidat soubor: ‚vue.config.js‘ do hlavni složky modulu. Po dalším startu aplikaci, Vue.js detekuje novou konfiguraci a zprovozni ji. Obsah toho souboru je:

```
module.exports = {
    devServer: {
        proxy: 'http://localhost:80'
    }
};
```

Čímž jsem pomoci proxy konfigurace povolil CORS pro uložiště Owncloud, které bezi na portu 80.

[21] CORS (Cross-Origin Resource Sharing)

5 Testování

Backend

Na testování backendu, autor práce zvolil testovací framework „Mocha“, který běží na Node.js a umožňuje paralelní testování.

Instalace:

```
npm install --save-dev mocha
```

Konfigurace:

```
let chai = require('chai');
let chaiHttp = require('chai-http');
let should = chai.should();

chai.use(chaiHttp);
```

Také je důležité v app.js přidat na testování další řádek:

```
module.exports = server; // for testing
```

Vzorek testování kontrolérů ve „savage“ modulu:

```
describe.only('Add userAccreditation, then check if it
saved', () => {
    it('it should POST userAccreditation model with name
"Jorshua Agastins"', (done) => {
        chai.request(server)
            .post('/userAccreditation')
            .send([
                {
                    "info": {
                        "name": "Jorshua Agastins",
                    },
                    "form": [],
                    "state ": {},
                    "upload ": {},
                }
            ])

.end((err, res) => {
                res.should.have.status(200);
```

```
          done(); });
});

it('it should GET all user accreditaions and has specific
userAccreaditaion with name "Jorshua Agastins"', (done) => {
        chai.request(server)
            .get('/userAccreditationList')
            .end((err, res) => {
                res.should.have.status(200);

res.body[0].info[0].name.should.include("Jorshua Agastins");
                done();
            });
    });
});
```

každý „it“ je soukromý test. Pokud chceme propojit několik testů do jediného scénáře, musíme přidat do „describe“ – „.only“

Ruční testování probíhalo rovněž během vývoje pomocí nástroje „Postman“, a to především při kontrole REST API rozhraní.

Frontend

Pro testování frontendu bylo využito E2E testů. End-to-end testování je technika, která testuje celý softwarový produkt od začátku do konce, aby bylo zajištěno, že se aplikace chová podle očekávání. Definuje systémové závislosti a zajišťuje, aby všechny integrované součásti fungovaly podle očekávání.

Hlavním účelem testování typu End-to-end (E2E) je testování ze zkušeností koncového uživatele simulováním scénáře skutečného uživatele a ověřením testovaného systému a jeho komponent pro integraci a integritu dat.

Softwarové systémy jsou dnes složité a propojené s mnoha subsystémy. Pokud některý ze subsystémů selže, může dojít k selhání celého softwarového systému. Jedná se o hlavní riziko, kterému se lze vyhnout komplexním testováním.

Instalace:

```
vue add e2e-cypress
```

nebo pomoci npm:

```
npm install cypress --save-dev
```

Pro zobrazení všech e2e testu byl použit příkaz:

```
npm run cypress:open
```

Vzorek e2e testu ve „savage“ modulu:

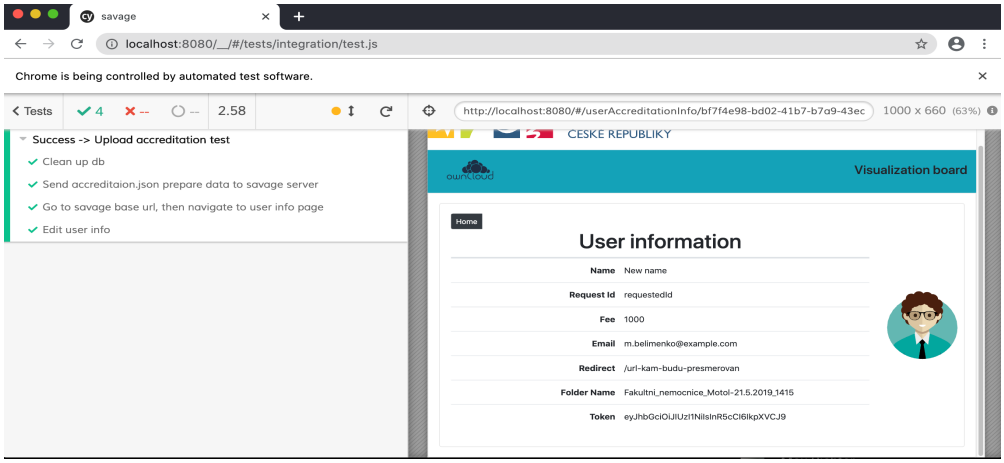
```
it('Go to savage base url, then navigate to user info page',
() => {
    cy.visit('http://localhost:8080')
    cy.get('.edit-info-button').click();
    cy.url().should('include', '/edit');
})

it('Edit user info', () => {
    cy.get('.name').clear().type('New name');
    cy.get('.submit').click();
    cy.url().should('not.include', '/edit');
    cy.get('table').contains('New name');
})
```

Jedná se o e2e test. Element ‚cy‘ je module z Cypress knihovny, který umožňuje přistupovat do jednotlivých elementu DOM.

Na začátku test přesměruje uživatele na hlavní stránku Savage modulu. Pak po zvolení tlačítka editace uživatele se následně zkontroluje, jestli se změnilo url. Zadá se nove jméno uživatele, uloží se, a zkontroluje, zda změny se projevili.

Na obrázku můžete vidět výstupy malického e2e testu:

Obrázek 13 Cypress testovací vzorek



Zdroj: Vlastní zpracování

6 Možnosti dalšího vývoje

Jako další možnosti zlepšení navrhuji:

Implementovat podporu mobilních zařízení

Trh s mobilními aplikacemi roste mílovými kroky. Toto obrovské odvětví se každým dnem rozšiřuje a ještě se nezastaví. Mobilní aplikace rozšiřují své funkce a možnosti, nyní je možné je synchronizovat s jinými zařízeními. Proto jsou stále více žádané.

Vytvoření robota (bota)

Chatbot je software určený ke komunikaci s uživateli. Chatboti mohou provádět informační, referenční funkce. To usnadní podaní formulářů.

Vytvoření funkce chatu mezi uživatelem a zaměstnancem MZ

Další možnost je vytvořit online chat, mezi klientem a administrátorem, který bude podporován např. v režimu 8x5.

Zasílání notifikace na e-mail uživatele

Po každém zásadním kroku procesu, jako je například generování formulářů, podání, nebo platba, by mohl být uživatel informován i pomocí asynchronní zprávy, tedy například emailem.

Přidání platebního systému

Přidat možnost úhrady za službu prostřednictvím elektronické platební brány. Pro tyto účely je momentálně zvažováno využití hotového řešení ComGate.

Vytvoření rolí pro správu systému

Role zabezpečení definuje přístupová práva k různým typům záznamů pro určitou kategorii uživatelů. Chcete-li řídit přístup k datům, můžete upravit existující role zabezpečení, vytvořit nové a přiřadit uživatele k jiným rolím. Každý uživatel může mít několik rolí zabezpečení.

7 Závěr

Výsledkem této bakalářské práce je webová aplikace, postavená na architektuře mikroslužeb, která umožňuje plnohodnotné podávání žádostí o akreditace a vzdělávací plány skrze prostředky elektronické komunikace a následně je ukládat do cloudového úložiště. Dále vznikla možnost editovat, vyhledávat a kontrolovat stav každého uživatele v systému. V průběhu vývoje bylo od původního záměru vyvinout vlastní platební bránu ustoupeno a momentálně je úmyslem zadavatele použít existující řešení, a to platební bránu Comgate. V rámci práce byl v průběhu analýzy definován příslušný proces formou BPMN diagramu a popsáno vzájemné rozhraní mikroslužeb formou diagramu komponent z jazyka UML. Byla provedena analýza existujících javascript frameworků a výběr technologie pro implementaci řešení. Pro front end byl použit jeden z nejvýznamnějších responzivních front-end frameworků Bootstrap. Bylo navrženo a schváleno uživatelské rozhraní. Pro ukládání dat byla vybrána vhodná NoSql databáze (MongoDB). Pro celé řešení byl zvolen moderní způsob distribuce formou kontejnerů (Docker), včetně message brokeru, OwnCloudu i samotných komponent výsledného řešení. Moderní architektura založená na mikroslužbách umožňuje snadno do budoucna rozšiřovat funkcionalitu, a to v jakémkoliv jazyce, nebo odebírat už existující, bez velkého vlivu na celou aplikaci.

Vzhledem k situaci na straně cílové organizace, nebylo doposud možné nasadit aplikaci do pilotního provozu. Práce tak zůstává do jisté míry ve formě proof-of-concept, se kterým se zástupci cílové organizace teprve učí pracovat. Díky kontejnerizaci je však celé řešení vstřícné pro nasazení na vlastní infrastruktuře organizace, či na některém veřejném cloudu.

Uživatelská dokumentace: https://drive.google.com/file/d/19OxFCc8HzlTIV-34JnTcxfFZw5zZI2Ne/view?usp=sharing

Zdrojový kód: https://bitbucket.org/mkyoung01/mz/src/master/

Přístupové údaje ke zdrojovému kódu

email: ministerstvo.zdravotnictvi.jcu@gmail.com
heslo: PrfJcu2020

8 Citovaná literatura

[1] Z. Šochová, „SCRUM,“ [Online]. Available: https://sochova.cz/co-je-scrum-proces.htm.

[2] „BPMN,“ [Online]. Available: https://www.tx.cz/bpmn/co-je-bpmn .

[3] „Mikrosluzba,“ [Online]. Available: http://voho.eu/wiki/mikrosluzba/.

[4] „Framework,“ [Online]. Available: https://php.baraja.cz/proc-pouzivat-frameworky#co-je-a-co-dela-framework.

[5] „Stack Overflow,“ [Online]. Available: https://stackoverflow.com/.

[6] B. Koďousková, „Vue.js,“ [Online]. Available: https://www.rascasone.com/cs/blog/co-je-framework-vuejs.

[7] „Node.js,“ [Online]. Available: https://www.itnetwork.cz/javascript/nodejs/uvod-do-nodejs.

[8] „MongoDB,“ [Online]. Available: https://cs.bccrwp.org/compare/what-is-mongodb-and-how-is-it-different-compared-to-a-sql-database-such-as-postgresql-cfb9ac/.

[9] M. GRYGAŘÍKOVÁ, „Master,“ 14 08 2019. [Online]. Available: https://www.master.cz/blog/docker-kubernetes-kontejnery-jak-funguji-proc-je-chtit/.

[10] UML, „Microsoft,“ 24 09 2019. [Online]. Available: https://www.microsoft.com/cs-cz/microsoft-365/business-insights-ideas/resources/guide-to-uml-diagramming-and-database-modeling.

[11] „Component diagram,“ [Online]. Available: http://mpavus.wz.cz/uml/uml-s-component-3-3-2.php.

[12] „Bootstrap,“ 26 05 2019. [Online]. Available: https://www.html-factory.cz/clanek/proc-pouzivat-bootstrap/.

[13] „CORS,“ [Online]. Available: https://opendata.gov.cz/%C5%A1patn%C3%A1-praxe:chyb%C4%9Bj%C3%ADc%C3%AD-cors.

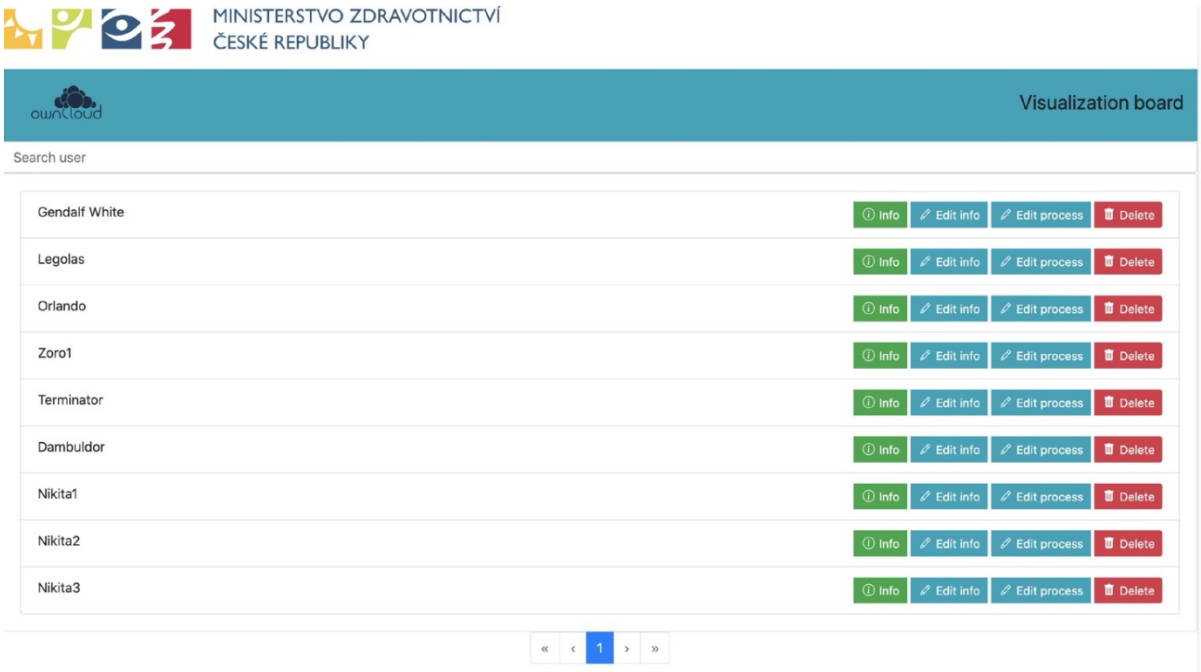
9 Seznam obrázků a tabulek

Seznam obrázků

Tabulky

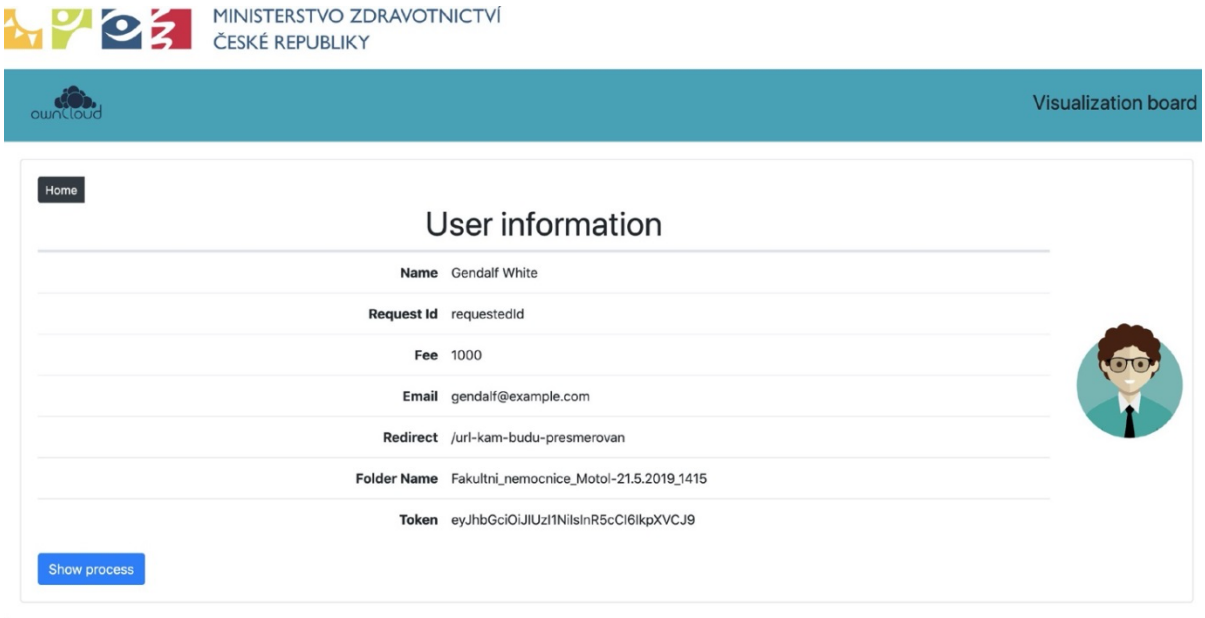
10 Příloha A Grafický vzhled aplikace

Obrázek 14 Seznam uživatelů - Uživatelské rozhraní



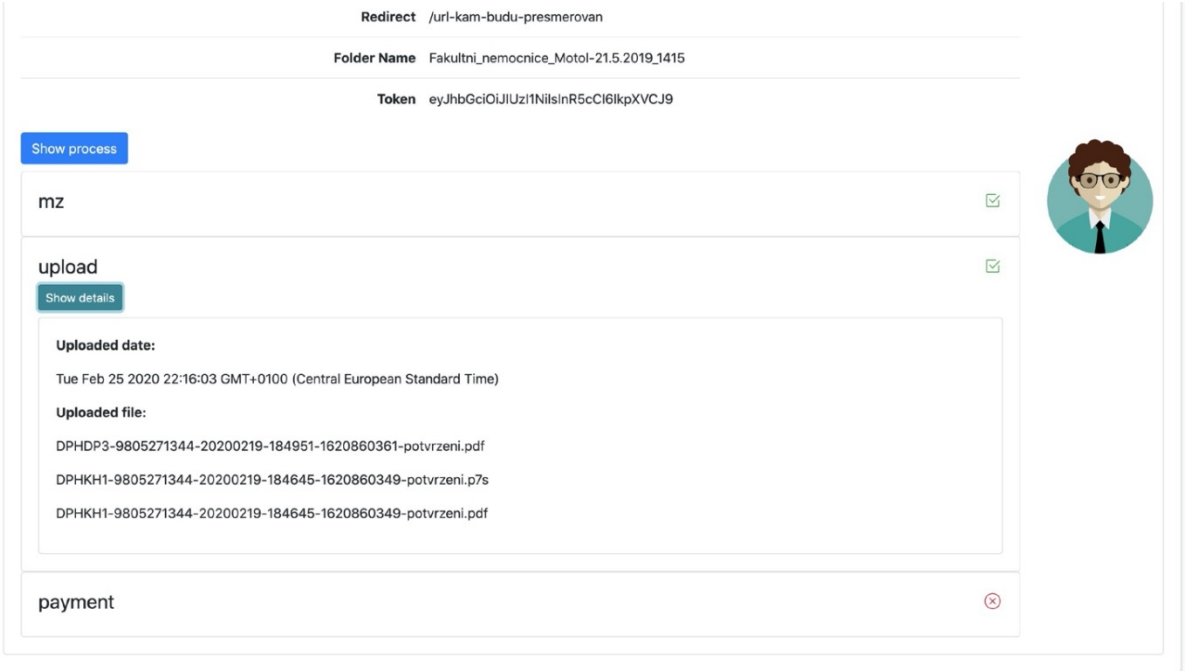
Zdroj: Vlastní zpracování

Obrázek 15 Informace uživatele - Uživatelské rozhraní



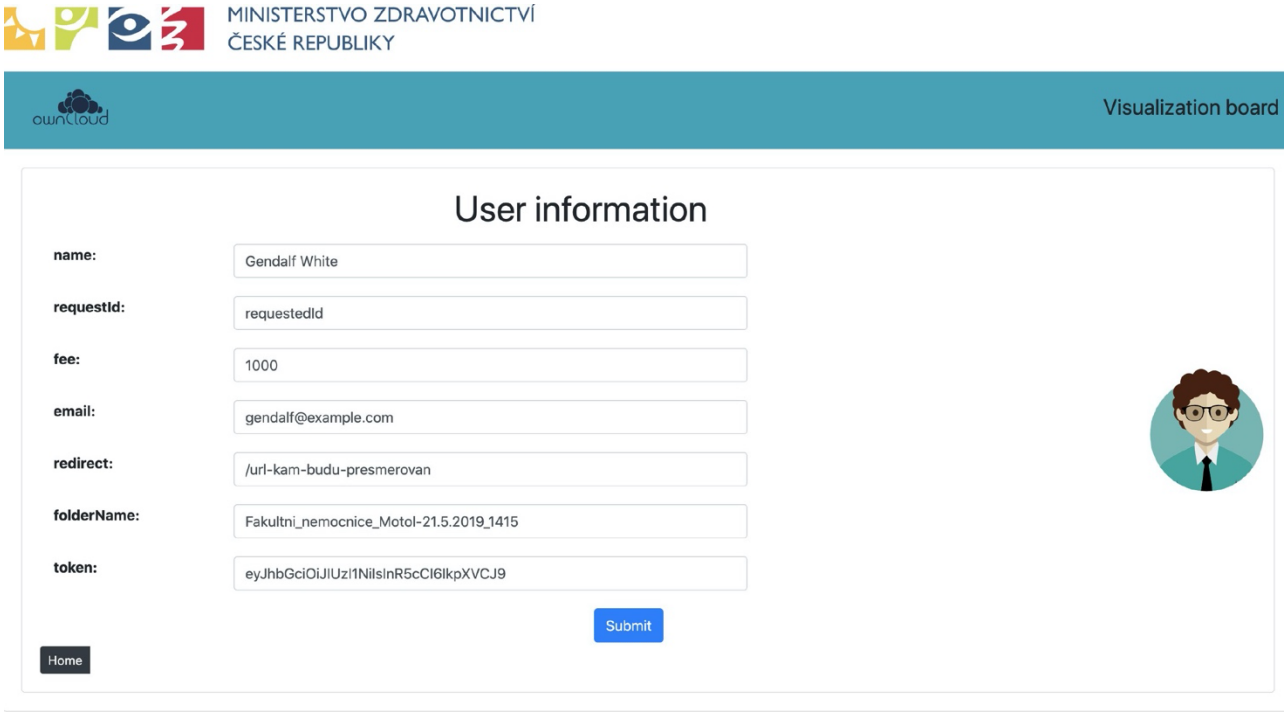
Zdroj: Vlastní zpracování

Obrázek 16 Procesy uživatele - Uživatelské rozhraní



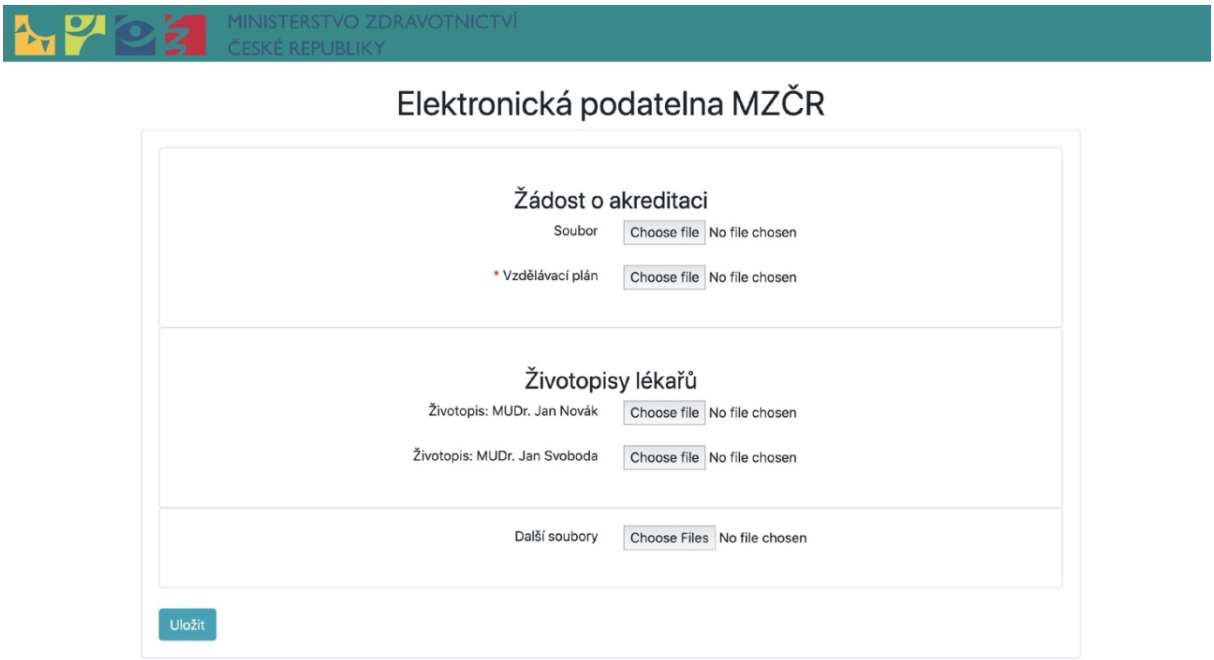
Zdroj: Vlastní zpracování

Obrázek 17 Editace uživatele - Uživatelské rozhraní



Zdroj: Vlastní zpracování

Obrázek 18 Nahrávací komponenta - Uživatelské rozhraní



Zdroj: Vlastní zpracování