FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Portál pro organizování vědeckých konferencí (back-end)
Student:	Jakub Pečenka
Vedoucí:	Ing. Petr Pauš, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Informační systémy a management
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Cílem práce je vytvořit back-end a jednoduchý front-end pro webový portál pro pořádání vědeckých konferencí. Měly by být splněny následující požadavky. Back-end by měl podporovat ukládání dat o účastnících, abstraktech, samotné konferenci a programu do vhodné databáze pomocí vhodného API. Tyto údaje by měly být následně poskytovány pomocí vhodného API pro využití na front-endu.

Pokyny k vypracování:
1. Analyzujte vhodné technologie pro tvorbu back-endu webových aplikací s ohledem na možnosti použitého serveru.
2. Navrhněte databázové řešení a API, které bude data z databáze poskytovat.
3. Navrhněte jednoduchý front-end demonstrující použití API.
4. Řešení implementujte.
5. Proveďte vhodné testování.
6. Zhodnoťte ekonomický přínos prototypu.

Elektronicky schválil/a Ing. David Buchtela, Ph.D. dne 6. listopadu 2022 v Praze.

Bakalářská práce

PORTÁL PRO ORGANIZOVÁNÍ VĚDECKÝCH KONFERENCÍ (BACK-END)

Jakub Pečenka

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Petr Pauš, Ph.D.
11. května 2023

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Především bych chtěl poděkovat vedoucímu mé práce Ing. Petru Paušovi, PhD. za cenné rady a konzultace k této bakalářské práci. V neposlední řadě bych také rád poděkoval své rodině, která mě po celou dobu studia podporovala.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 11. května 2023 .....

Abstrakt

Tato bakalářská práce se věnuje revitalizaci již existujícího webového portálu pro pořádání vědeckých konferencí v rámci Fakulty jaderné a fyzikálně inženýrské Českého vysokého učení technického v Praze. Práce se v úvodní části zaměřuje na analýzu konceptů a nástrojů pro tvorbu webových aplikací. Následně je analyzován současný systém a jsou identifikovány požadavky, které je třeba zachovat při návrhu nového řešení. Tomu je věnována navazující část práce. Pro implementaci prototypu na základě stanoveného návrhu byl použit programovací jazyk Go a společně s ním webový framework Gin. Zachování dat aplikace je dosaženo pomocí NoSQL databáze MongoDB. Demonstrativní frontend je tvořen prostřednictvím JavaScriptové knihovny React.js. Zprostředkování komunikace mezi klientem a serverem je zajištěno implementací REST API.

Klíčová slova backend webové aplikace, vědecká konference, REST API, MongoDB, programovací jazyk Go

Abstract

This bachelor thesis is devoted to the revitalization of an existing web portal for organizing scientific conferences within the Faculty of Nuclear and Physical Engineering of the Czech Technical University in Prague. The thesis focuses in the introductory part on the analysis of concepts and tools for creating web applications. Subsequently, the current system is analyzed and the requirements that need to be maintained when designing a new solution are identified. Design of the new solution is the subject of the following part of the thesis. To implement the prototype based on the established design, the Go programming language was used along with the Gin web framework. The application's data persistence is achieved by using the NoSQL database MongoDB. The demonstrative frontend is created using the JavaScript library React.js. The mediation of communication between client and server is provided by the implementation of REST API.

Keywords web application backend, scientific conference, REST API, MongoDB, Go programming language

Seznam zkratek

FJFI	Fakulta jaderná a fyzikálně inženýrská
ČVUT	České vysoké učení technické
WWW	World Wide Web
OSI	Open Systems Interconnection
IP	Internet Protocol
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
URL	Uniform Resource Locator
API	Application Programming Interface
SOAP	Simple Object Access Protocol
REST	Representational state transfer
URI	Uniform Resource Identifiers
XML	Extensible Markup Language
JSON	JavaScript Object Notation
ORM	Objektově relační mapování

Úvod

Vysoké školy mají ve společnosti několik důležitých rolí. Pro tuto práci je tou klíčovou z nich podněcování debat o různorodých tématech a vzájemné sdílení a kultivace znalostí. Pro tuto činnost lze zvolit mnoho forem. Nejjednodušší možnou formou je uspořádat společenské setkání na kýžené téma, na kterém se sejdou lidé se společnými zájmy a sdílí mezi sebou nabyté vědomosti. Jedná se tedy o tak zvanou vědeckou konferenci. V současné době na Fakultě jaderné a fyzikálně inženýrské (FJFI) Českého vysokého učení technického (ČVUT) v Praze k tomuto účelu slouží webový portál, jehož pomocí lze tyto konference pořádat a organizovat. Pomocí tohoto portálu se mohou lidé se zájmem o účasti na aktuálně se konající konferenci registrovat nejen jako posluchači a návštěvníci, ale také jako přednášející na jejich vybrané téma.

Aktuální systém, který zprostředkovává potřebný provoz, je na popud jeho autorů třeba revitalizovat. Celý proces je pak rozdělen do dvou projektů. Jeden má za úkol se věnovat modernizaci frontendu portálu a ten druhý jeho backendu. Tato práce má za úkol věnovat se druhému z projektů.

Dané téma jsem si zvolil, jelikož jsem chtěl pracovat na reálném projektu, který po zhotovení najde své využití. Zároveň je mou motivací myšlenka, že mohu alespoň minimálním dílem přispět k pozitivnímu prostředí, které nezištně napomáhá rozvoji a rozšiřování společných vědomostí.

Tato práce samotná je pak určena všem budoucím správcům webového portálu, kteří jej budou udržovat v chodu nadcházející roky. Ale také studentům a mým kolegům, jenž budou na tuto práci navazovat a dále ji rozvíjet. Všem výše zmíněným by tedy měla posloužit mimo jiné jako dokumentace technických rozhodnutí, která byla učiněna během vývoje.

Primárním záměrem práce je vytvořit adekvátní náhradu za současný systém v požadovaném rozsahu. Z toho vyplývá, že cílem práce je připravit důkladný návrh a dokumentaci a následně zhotovit implementaci prototypu backendu pro webový portál a jeho snadnou údržbu. Důležitým prvkem práce je fakt, že navazuje na již existující systém a na jeho základních myšlenkách staví. Následně je cílem vytvořit jednoduché uživatelské rozhraní pro případnou demonstraci a prokázání funkčnosti zhotoveného backendu

Nejprve se v úvodní části zabývám definicí potřebných termínů a pojmů. Zároveň také popisuji technologie používané při tvorbě webových aplikací. Dále analyzuji stávající řešení, použité technologie a modely. V následující kapitole se věnuji analýze a návrhu nové aplikace. V první části kapitoly popisuji návrh části aplikace, která zajišťuje po-

skytování a persistenci potřebných dat. Dále navazuji návrhem rozhraní pro komunikaci s ostatními aplikacemi a popisem návrhu frontendu, který má sloužit pro demonstraci implementovaného prototypu řešení. Závěrem se věnuji samotné implementaci, jejímu testování, nasazení a také zhodnocení přínosu pro uživatele.

Kapitola 1

Architektura webových aplikací

V průběhu devadesátých let minulého století pronikla na svět průlomová technologie. World Wide Web (WWW) započal svou cestu k tomu stát se součástí mnoha životů. Z počátku se šířil pouze mezi vědeckou komunitou, pro kterou byl konec konců vyvinut. Jeho tvůrci ze Švýcarského CERNu měli za cíl vytvořit prostředek jehož pomocí mezi sebou mohou sdílet poznatky ze svých prací. Postupem času se WWW dostával mezi více a více uživatelů. Studie publikovaná v roce 2002 mapující začátky WWW uvádí, že téhož roku měla tato technologie celosvětově přes 500 milionů uživatelů. [1, 2]

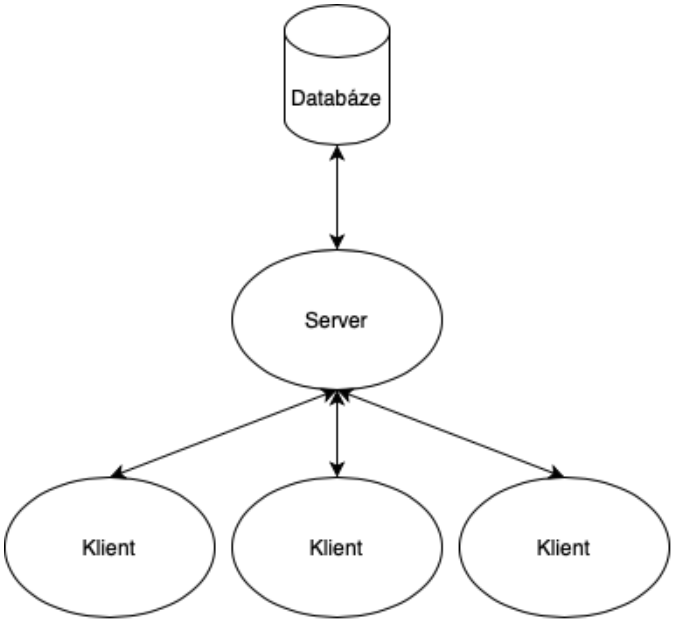
WWW se skládá z mnoha částí sdružených v jeden mocný celek. Pro komunikaci, jejíž pomocí si uživatelé mohou mezi sebou prostřednictvím internetu sdílet informace a znalosti, se využívá protokol aplikační vrstvy OSI modelu Hypertext Transfer Protocol (HTTP). Touto cestou uživatelé posílají a odebírají Webové dokumenty, které jsou zpracovávány ve formátu Hypertext Markup Language (HTML). Způsob, kterým se definuje jednoznačný přístup k těmto dokumentům, se nazývá Uniform Resource Locator (URL). Jeho součástí je deklarace použitého aplikačního protokolu, IP adresy cílového počítače a identifikace žádaného dokumentu. [1]

1.1 Klient-server

WWW zprostředkovává komunikaci pomocí protokolu HTTP. Ten ke svému fungování využívá takzvaný klient-server model. Jedná se o způsob, jakým lze definovat vztah mezi uživatelem, který si vyžádal kýžený dokument, a uživatelem, který daný dokument poskytuje prostřednictvím WWW. Využívání modelu klient-server není výhradou protokolu HTTP. Mezi rodinu sdružující se pod tento model patří například Simple Mail Transfer Protocol a File Transfer Protocol. [1, 3]

Celá architektura modelu klient-server stojí na dvou základních krocích. Klient v roli uživatele, který žádá o dokument, posílá požadavky serveru a ten v reakci na ně v roli uživatele, jenž vybraný dokument poskytuje, vrací klientovi odpovědi. Tento proces je znázorněn na diagramu v obrázku 1.1. V rámci jejich komunikace může probíhat výměna dat nejen od serveru směrem ke klientovi, ale také opačně. Důvodů proč vývojáři sahají po tomto modelu při tvorbě svých aplikací je celá řada. V první řadě lze docílit centralizace dat a zamezení jejich replikace napříč systémy klientů tím, že jsou dostupné pouze

na serveru. Dále je pak zajištěna jednoduchá komunikace mezi klientem a serverem tak, aby byla tato data a prostředky ihned k dispozici. Zároveň je výsledně dosaženo rozložení zátěže celé aplikace mezi více počítačů. [3]



Obrázek 1.1 Diagram modelu klient-server. Získáno z [3]

Způsobů, kterými se dá realizovat samotná implementaci modelu klient-server je mnoho. Mezi základní průmyslové návrhy se řadí dvouvrstvá a třívrstvá architektura. Dvouvrstvá architektura se skládá z klienta, který běží na zařízení uživatele, a serveru, jenž vystupuje v roli poskytovatele dat. V tomto režimu se server stará pouze o doručení dat klientovi. Z toho důvodu se využívá pouze takzvaný databázový server. Jedná se tedy pouze o datové skladiště, které uchovává samotná data a vazby mezi nimi. Na klienta jsou v tomto režimu kladeny vyšší nároky. Jelikož databázový server poskytuje pouze data, musí je klient sám zpracovávat. Zároveň také slouží k tomu, aby výsledek zobrazil uživateli. Pro jeho vysokou míru odpovědnosti je přezdíván jako tlustý klient. Naproti tomu stojí architektura třívrstvá, která je považována za populárnější. Celá architektura stojí na stejném principu jako dvouvrstvá. Nachází se zde klient a server. Rozdíl spočívá v tom, že klient již nekomunikuje se samotným datovým úložištěm, nýbrž s prostředníkem, který slouží pro jejich spojení. Server se skládá ze dvou částí jimiž jsou databázový server a oproti předešlému příkladu navíc aplikačnímu serveru. Ten zpracovává příchozí požadavky od klientů a získává potřebná data z databázového serveru. Klient v tomto případě neprovádí žádnou aplikační logiku a přezdívá se mu proto tenký klient. [3, 4]

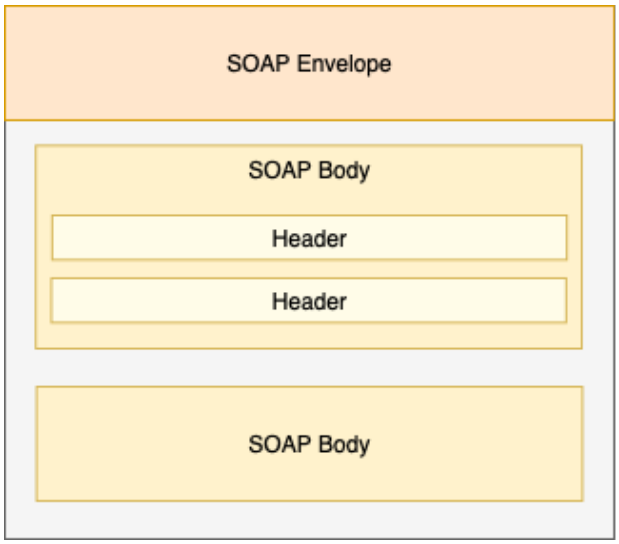
V pojetí návrhu aplikací, které slouží pro WWW, můžeme zavést pro zmiňované architektury nezávislé označení. Bez ohledu na způsob implementace a samotného výběru dvouvrstvé, třívrstvé nebo n-vrstvé architektury, je vhodné určit části aplikace a jejich zodpovědnosti. Ta část, která je zodpovědná za prezentaci dat klientovi, nazýváme frontend. Část zodpovědnou za zachování a zpracování dat nazýváme backend. [5]

1.2 Standardizace formátu přenosu dat mezi frontendem a backendem

Pro potřeby komunikace mezi frontendem a backendem se využívá protokol HTTP. Aby byla zajištěna vzájemná kompatibilita komunikace mezi různými aplikacemi, je třeba definovat společný formát přenosu dat. Jinými slovy je potřeba navrhnout architekturu takzvaného Application Programming Interface (API), jehož pomocí mezi sebou aplikace komunikují. V zájmu zachování soudržnosti s architekturou klient-server, si lze API představit jako komunikaci požadavků a odpovědí. K tomuto účelu bylo navrženo několik standardizovaných řešení. Mezi nejužívanější z nich se řadí Simple Object Access Protocol (SOAP) a Representational state transfer (REST). [6, 7]

1.2.1 SOAP

Standard aplikačního rozhraní SOAP byl vyvinut v roce 1998 a jeho první verze byla publikována o rok později. Na samotném vývoji se podílela například společnost Microsoft. Po svém uvedení se SOAP těšil velkému zájmu. Pro přenos dokumentů užívá formátu Extensible Markup Language (XML). Klient a server si mezi sebou vyměňují zprávy, takzvané SOAP Messages. Ty se skládají ze tří částí znázorněných na obrázku 1.2: [6, 7, 8]



Obrázek 1.2 Diagram SOAP Message. Získáno z [6]

- Envelope (obálka)
  - obsahuje informace o zprávě
  - identifikuje ji jako SOAP Message
  - jedná se o nepostradatelnou část zprávy
- Header (hlavička)

- není povinná
- slouží například k autorizaci
- může se objevit vícekrát a každá definovat jiné atributy viz 1.2

- Body (tělo)
  - obsahuje data v XML
  - sdružuje informace o požadavcích a odpovědích s daty

Samotný standard přináší při použití pro webové aplikace několik nevýhod. Mezi ně patří například pomalé parsování vstupních dat ve formátu XML, vyšší míra komplexnosti oproti ostatním řešení a řádově vyšší spotřeba paměti. Na výpisu kódu 1.1 je možné vidět způsob zápisu dat pomocí XML. [6]

Výpis kódu 1.1 Ukázka formátu XML. Získáno z [9]

```
<name>
    <first>John</first>
    <last>Smith</last>
</name>
```

1.2.2 REST

Architektonický styl REST byl navržen v roce 2000 a jeho autorem je Roy Fielding. Nejedná se tedy o klasický standard stejně jako v případě SOAP, ale pouze o souhrn jistých doporučení a omezení. Při jeho tvorbě bral autor v potaz nároky a požadavky, které na svůj vývoj a provoz kladou webové aplikace. Roy Fielding stanovil pro REST následující omezení: [6, 7, 10]

- Starting with the Null Style. Definuje, že vstupním bodem REST je prázdná sada omezení (Null Style).
- Client-Server. Rozdělení starostí mezi klienta a server. Klient má na za úkol zařídit uživatelské rozhraní a nezabývat se persistencí dat. Úkolem serveru je přesný opak. Výsledkem má být snadná škálovatelnost a nezávislý vývoj komponent.
- Stateless. Omezení stanovící bezstavovost komunikace. Požadavky zaslané klientem musí obsahovat veškeré potřebné informace pro server a nesmí se spoléhat na kontext předchozí komunikace. Veškerý stav aktuálních výměn požadavků a odpovědí je tak potřeba uchovávat u klienta.
- Cache. Každá odpověď na klientův požadavek musí být označena buďto jako cacheable, nebo non-cacheable. Všeříkající název indikuje povolení pro uložení obdržených dat do cache.
- Uniform Interface. Komponenty aplikace musí mít společné rozhraní. Jeho implementaci je pak možné vyvíjet nezávisle na ostatních komponentách. Dále je v REST definováno omezení na samotné rozhraní.
  - Musí být jednoznačně identifikován každý zdroj dat.

  - Každý takový zdroj dat je nutné mít možnost modifikovat.
  - Odpovědi na požadavky musí být samopopisné.
  - Stav aplikace by měl být poháněn hypermédii.

- Layered System. Aplikace by měla být členěna do vrstev. Zároveň by každá z vrstev měla mít přístup ke komunikaci pouze se svými sousedy. Důsledkem dodržení tohoto omezení sníží aplikace svou komplexitu. Nicméně se zvýší prodleva odezvy tím, že data musí procházet vícevrstvým systémem.

- Code-On-Demand. Klient by měl mít možnost stáhnout si dodatečný kód a rozšířit tak své dovednosti a vlastnosti. Cílem je zjednodušení klienta a zúžení nutně předem implementovaných vlastností. Toto omezení je v REST uloženo vývojářům jako nepovinné.

Pokud jsou během implementace návrhu API splněna tato omezení, lze pro něj pak užívat označení REST API. Takovou webovou aplikaci, která implementuje REST API, nazýváme RESTful aplikací. [10, 11]

Veškerá data, která má webová aplikace komunikující pomocí REST API za úkol poskytovat, jsou v terminologii známá jako zdroj. K těmto zdrojům se lze jednoznačně odkazovat použitím Uniform Resource Identifiers (URI). V rozhraní aplikace jsou tak definována specifická URI, kterými je zajištěn přístup k určeným datům. REST API využívá ke své komunikaci protokol HTTP. Má tedy přístup k jeho metodám, které slouží k výběru akce na zvoleném zdroji. Mezi základní metody se řadí GET, POST, PUT a DELETE. [6, 7, 11]

- GET. Vyžádání dat ze zvoleného zdroje.
- POST. Odeslání dat vybranému zdroji.
- PUT. Aktualizace existujících dat na zdroji.
- DELETE. Smazání existujících dat identifikovaných pomocí URI zdroje.

Ve své disertační práci, ve které Fielding REST publikoval, nijak nespecifikuje jaký formát by měl být používán pro ukládání dat do těla HTTP. Vývojáři jej tak mohou implementovat pomocí již zmíněného XML. Nicméně častější volbou formátu se stal takzvaný JavaScript Object Notation (JSON). Jednou z mnoha výhod JSON je mimo jiné to, že je jednoduše čitelný člověkem, jak lze vidět na obrázku 1.2. Tyto dva formáty lze mezi sebou porovnávat v několika kategoriích a ohledech. Nicméně operace, kterou je třeba na formátovaných datech vykonávat prakticky pokaždé, když jej chceme strojově číst a zpracovávat, je parsování. Dle [9] je odhadováno, že JSON je možné parsovat až stokrát rychleji než XML. [6, 9, 10]

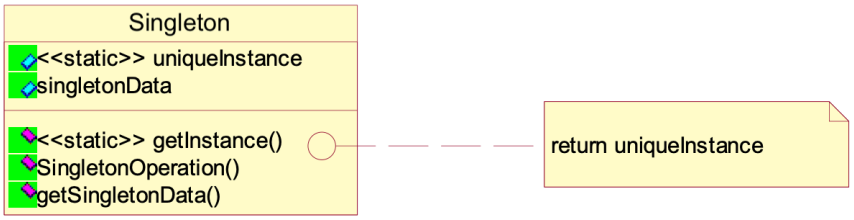
Výpis kódu 1.2 Ukázka formátu JSON. Získáno z [9]

```
{
    "firstname":"John",
    "lastname":"Smith"
}
```

1.3 Architektonické a návrhové vzory webových aplikací

Vývoj aplikací s sebou nese řadu problémů. Čím více se aplikace stává ambicióznější a postupem času roste, tím nevyhnutelně stoupá i míra její komplexnosti. S přibývajícím kódem roste nutnost důkladnějšího testování. Navazující vývoj se tak stává časově náročnější a ruku v ruce s tím i finančně nákladnější. Dalším kritickým neduhem složitých aplikací je doba trvání zasvěcení nových vývojářů, kteří se připojí k vývoji v průběhu jeho životního cyklu. Uvedení do architektury vznikající aplikace může trvat neúměrně dlouho. Z těchto důvodů začala vznikat řešení, která měla za cíl eliminovat tyto překážky. Taková řešení spadají do kategorie takzvaných návrhových a architektonických vzorů.

Účelem každého návrhového vzoru je poskytnout konkrétní šablonu pro řešení opakujících se problémů, podle které mohou vývojáři vytvářet části svých aplikací. Jedním z typických představitelů návrhových vzorů je singleton znázorněný na obrázku 1.3. Třída, která jej implementuje, má za úkol zajistit, aby vždy vznikla pouze jedna její instance. [12, 13]



Obrázek 1.3 Diagram návrhového vzoru singleton. Získáno z [13]

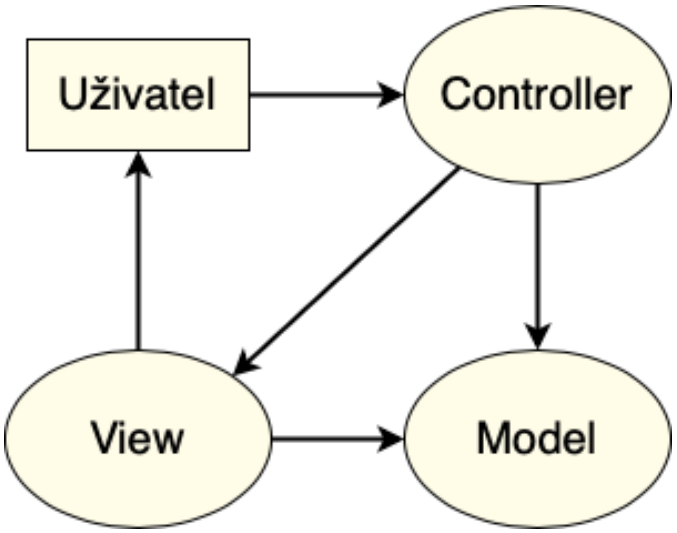
Architektonické vzory pak cílí na definování parametrů návrhu architektury aplikace. Mají za úkol poskytnou vývojářům postupy, podle kterých vytvářet udržitelné a srozumitelné návrhy. Vzory architektury rozdělují aplikaci na samostatné funkční celky a popisují, jakým způsobem by mezi sebou měly komunikovat. Díky tomu je dosaženo toho, že nově příchozí vývojáři mají o těchto architekturách povědomí a jejich zasvěcení do projektu je tak snazší a rychlejší. Mezi klasické vzory se řadí Model – View – Controller (MVC), Model – View – Presenter (MVP) a Model – View – ViewModel (MVVM). [14, 15]

1.3.1 Model – View – Controller

Počátky vzoru architektury MVC sahají až do 80. let minulého století a jedná se tak o jeden z prvních pokusů oddělit od sebe uživatelské rozhraní a zbytek aplikace. Tento vzor byl původně uveden jako implementace v programovacím jazyce Smalltalk. Následně se stal inspirací pro tvorbu architektonických vzorů MVP a MVVM. [14, 16]

Podstata MVC spočívá v rozdělení aplikace na 3 funkční celky, nebo také komponenty. Prvním z nich je Model. Ten je zodpovědný za zachování dat. Není nutné, aby tato data byla skladována v modelu samotném. Jednou z možných alternativ je opatřit data z externích zdrojů. Dalším prvkem MVC je View, jenž slouží jako uživatelské rozhraní zobrazující výstupy aplikace. Třetí a zároveň poslední součástí MVC je Controller. Jeho

rolí v systému je zajistit business logiku aplikace a reagovat na uživatelský vstup a v závislosti na něm interagovat s Modelem, ale také propagovat změny směrem do View. Na diagramu 1.4 lze vidět, že Controller tedy slouží jako prostředník v komunikaci Modelu a View. Současně mezi nimi existuje spojení takové, že View si může vyžádat informace o změnách přímo od Modelu. [14, 16]



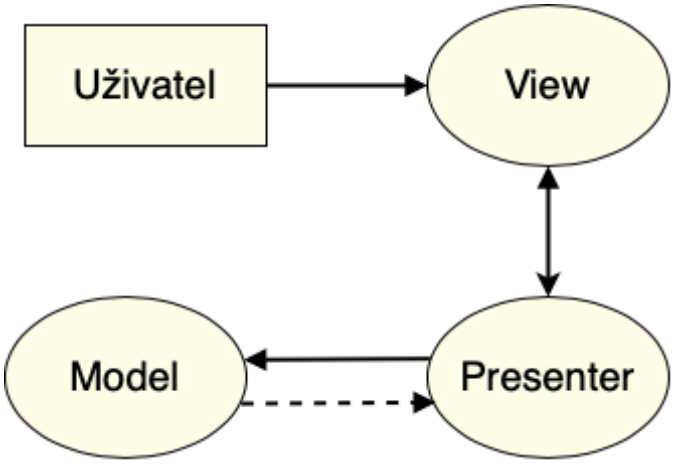
Obrázek 1.4 Diagram architektonického vzoru MVC. Získáno z [17]

1.3.2 Model – View – Presenter

Architektonický vzor MVP byl uveden v roce 1996. Jeho první implementace byla v rámci operačního systému od společnosti Taligent. Návrh vzoru byl silně ovlivněn a inspirován architekturou MVC. I v tomto případě má Model za úkol obstarávat data pro ostatní komponenty. Také View stále slouží k zobrazování výstupu aplikace uživateli přes uživatelské rozhraní. Tou zásadní změnou je nahrazení Controlleru za Presenter. Jak je znázorněno na obrázku 1.5, vzor MVP nedovoluje komunikaci mezi Modelem a View. Veškerou komunikaci tak musí zprostředkovat Presenter. Zde leží klíčový rozdíl oproti MVC, kde tyto dvě komponenty měly určený způsob spojení. Ve své podstatě tak Presenter pouze manipuluje s daty Modelu, interpretuje je a dále předává do View. [18, 19]

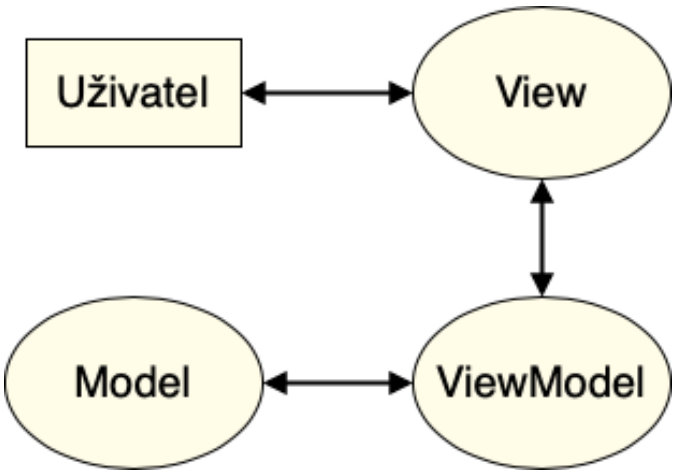
1.3.3 Model – View – ViewModel

Vzor architektury MVVM představil John Grossman ve svém blogu v roce 2005. Sám autor uvádí, že se jedná o variaci na MVC, která je upravena pro vývoj moderních uživatelských rozhraní. Jak lze vidět na obrázku 1.6, tak stejně jako v předchozích případech se MVVM skládá ze tří částí, kterými jsou Model, View a ViewModel. Model je zodpovědný za uchovávání a obstarávání dat. Oproti MVC, kde interakci s uživatelem zajišťuje Controller, je tato agenda povinností komponenty View, které ale nadále zůstává odpovědnost za zobrazování dat aplikace uživateli. Úkolem ViewModelu je udržovat aktuální



Obrázek 1.5 Diagram architektonického vzoru MVP. Získáno z [18]

stav aplikace a poskytovat spojení mezi View a Modelem. Obsahuje tedy logiku, jejíž pomocí transformuje data z Modelu podle požadavků View a obráceně. Tímto by se ale nijak nelišil oproti předchozím vzorům. Rozdíl MVVM tkví ve způsobu komunikace komponent View a ViewModel. Veškeré změny, které v dané komponentě nastanou, se ihned propagují do té druhé. Tento mechanismus se nazývá Data Binding. [14, 18, 20]



Obrázek 1.6 Diagram architektonického vzoru MVVM. Získáno z [21]

Kapitola 2

Technologie pro vývoj webových aplikací

Pro vývoj webových aplikací je potřeba ovládat a rozumět širokému spektru technologií. V minulé kapitole byly představeny ty klíčové z nich. Patřili mezi ně například protokol HTTP, na kterém stojí celý koncept WWW. Dále pak HTML, který je stavebním kamenem sdílení dokumentů mezi uživateli, a URL sloužící k odkazování na ně. Model architektury klient-server, který definuje vztahy mezi uživateli WWW. Návrhové a architektonické vzory, jenž mají za úkol zefektivnit a zlevnit vývoj aplikací s následnou snazší údržbou.

Zatímco technologie jako HTTP, HTML, URL, jsou již hotové řešení, které lze při vývoji přímo využít, tak například architektonické vzory MVC, MVVM, MVP jsou pouze šablonami uplatňovanými při designu a návrhu aplikace. K tomu, aby bylo možné tyto návrhy implementovat, je nutné zvolit vhodné technologie, které jsou pro tyto účely navrženy. Tato kapitola se věnuje analýze takových technologií nejdříve z pohledu vývoje backendu webové aplikace a poté jejího frontendu.

2.1 Backend

Celou webovou aplikaci lze rozdělit podle modelu klient-server do dvou částí na klienta a server. Serverová část webové aplikace, neboli backend, tvoří jádro logiky jejího fungování a definuje služby poskytované klientům. Stará se tedy nejen o zpracování dat, ale také o jejich korektní zachování. Zpravidla se tak backend samotný skládá z několika dalších komponent. Při vývoji webových aplikací se jedná konkrétně o server, aplikaci a databázi. První komponentou je webový server. Jeho úlohou je zajistit komunikaci s klienty prostřednictvím protokolu HTTP. Dále pak aplikace, která je spuštěna na serveru a zpracovává požadavky od klientů. Těm následně posílá patřičné odpovědi zpět. Poslední z nich je databáze sloužící k uchování dat aplikace. V posledních dvou komponentách tvořící backend se skrývá samotná podstata logiky webové aplikace. Komunikaci mezi databází a aplikací často vývojáři zprostředkovávají pomocí Objektově relačního mapování. To slouží pro vzájemné mapování objektů tvořených v rámci aplikace a databáze. [1, 22]

2.1.1 Logická vrstva

Při implementaci aplikace, která je provozována na straně serveru, je třeba provést zásadní rozhodnutí. To se týká výběru jazyka a s tím navazujících frameworků pro něj vyvinutých. Mezi typické jazyky, na které padá volba nejčastěji, se řadí Java, JavaScript, Python, nebo Go. [23]

2.1.1.1 Python

Python je programovací jazyk jehož tvůrce Guido van Rossum začal s jeho vývojem na konci 80. let minulého století. Jedná se o jazyk interpretovaný, objektově orientovaný a vysokoúrovňový s dynamickou sémantikou. Tyto jmenované vlastnosti definují způsob jeho fungování. Python má vestavěné vysokoúrovňové datové struktury. Jednou z dalších klíčových funkcí, které využívá, je dynamické typování. Z těchto důvodů je často velmi oblíbený jako skriptovací jazyk nebo jako nástroj pro spojování existujících aplikací v daném prostředí. Kód psaný pomocí Pythonu má tendenci být krátký a kompaktní. Jeho primární nevýhodou je jeho rychlost v porovnání oproti komplikovaným jazykům. Dle studie [24] zaměřené na porovnání rychlosti vykonání algoritmických instrukcí v různých programovacích jazycích vyplývá, že Python byl v průměru 8 až 29 krát pomalejší než kód zkompilovaný pomocí kompilátoru GCC v jazyce C++. V jednom z testů, který byl zaměřený na řazení, byl rozdíl mezi těmito dvěma jazyky největší a to takový, že Python byl 129 krát pomalejší. I přes své nevýhody se jedná o velmi populární jazyk a to primárně díky své jednoduché syntaxi. Python se také může chlubit velkou základnou komunity uživatelů, kteří vzájemně nejen sdílí své znalosti Pythonu, ale také se podílejí na jeho vývoji. [25, 26]

Pro podporu vývoje webových aplikací vznikla v Pythonu již řada frameworků. Jedním z nich je například Django. Jedná se o webový framework, který je bezplatný a open source. Vývojáři pracující s Djangem mají k dispozici předem připravená řešení již známých problémů, se kterými se lze potkat při vývoji webových aplikací. Poskytuje nástroje pro řešení bezpečnosti aplikace, dokáže zajistit komunikaci s celou řadou typů frontendu, implementuje vlastní objektově relační mapování (ORM) a cílí na udržení spravovatelnosti a snadnou škálovatelnost aplikací. Podporuje vývoj aplikací využívajících architekturu MVC. Mezi webové stránky, které přímo užívají framework Django, patří například Instagram, Pinterest, National Geographic a Mozilla. [25, 27, 28]

Dalším příkladem a zároveň největším konkurentem Djanga je Flask. Často bývá zařazován do kategorie mikroframeworků. Primárním důvodem k tomu je ten fakt, že Flask cílí na poskytování řešení pouze základních problémů vývoje webových aplikací. Jedním z nich je například směrování klientských požadavků. Flask tak nepřináší v mnoha případech vývojářům nadbytečně nástroje, které ani nevyužijí. Jeho velkou výhodou je jeho rozšiřitelnost. Jelikož Flask není tak robustní jako Django, je třeba mu funkcionalitu, kterou vývojáři potřebují, dodatečně doplnit. Jedním příkladem může být ORM. Oproti Djangu, které již vlastní implementaci poskytuje, si vývojáři musí Flask o ORM rozšířit. Mohou si tak ale vybrat takové řešení, které jim zrovna vyhovuje. [25, 29]

2.1.1.2 Java

Java je kompilovaný, vysokoúrovňový, objektově orientovaný a staticky typovaný programovací jazyk. Jeho počátky se datují do roku 1991. Tou dobou byl autor Javy James Gosling zaměstnancem společnosti Sun Microsystems. Oproti Pythonu je třeba Javou napsaný kód před jeho spuštěním zkompilovat. Samotný proces kompilace se skládá ze dvou základních kroků. Nejdříve je kód psaný v Javě kompilován do bytecode. Ten je následně podobně jako v případě Pythonu interpretován v prostředí zvaném Java Virtual Machine. Zároveň JVM v tomto procesu využívá Just In Time kompilátor, který převádí pouze potřebné části kódu. Z toho vyplývá, že Java je velmi dobře přenositelná mezi zařízeními. Jedinou podmínkou je, aby daná zařízení podporovala a implementovala JVM. Dalším rozdílem je statické typování oproti dynamickému, který využívá Python. Vývojář má tak povinnost deklarovat typy proměnných. Výsledkem těchto vlastností je konkurenceschopná výkonnost. Ze studie [24] vyplývá, že kód psaný v Javě a kompilovaný pomocí OpenJDK je v průměru pouze 1,3 až 1,43 krát pomalejší než kód psaný pomocí C++ a kompilátoru GCC. [29]

I v případě Javy již vznikla řada frameworků pro podporu vývoje webových aplikací. Jedním velmi populárním zástupcem je Spring, který je značně komplexní a co do funkcí bohatý framework. Mezi problémy, které řeší, se řadí například bezpečnost aplikací, podpora pro komunikaci s databázemi a implementace MVC architektury. Spring tak poskytuje mnoho řešení, ale jednou z jeho klíčových funkcí je vkládání závislostí. Jedná se o implementaci stejnojmenného návrhového vzoru. Jelikož je Spring komplexní a robustní nástroj, tak je jeho konfigurace složitá a náchylná ke vzniku chyb. Proto jeho tvůrci vyvinuli Spring Boot, který se zaměřuje právě na zjednodušení konfigurace Springu. Využívá k tomu mechanismus zvaný autokonfigurace. [30]

2.1.1.3 JavaScript

JavaScript je skriptovací, objektově orientovaný, dynamicky typovaný programovací jazyk. Jedná se o jazyk, který je provozován ve webových prohlížečích a využívá se tak primárně na straně klienta. Jeho úkolem je rozšířit původně statické stránky o dynamické a interaktivní prvky. Přestože JavaScript vznikl původně pro potřeby klientské stránky komunikace, vznikla iniciativa využít jeho schopnosti i na straně serveru. Ryan Dahl tak v roce 2009 vyvinul běhové prostředí pro podporu vývoje takových aplikací zvané Node.js, nebo také pouze Node. Jeho implementace stojí na Google Chrome V8 enginu. Nejen že Node podporuje běh JavaScriptu, ale také přidává další funkčnosti. [29, 31]

Pro potřeby vývoje webových aplikací existuje pro Node.js hned několik frameworků. Jedním z jejich zástupců je Express.js. Zaměřuje se na řešení problémů jako jsou směrování klientských požadavků na základě URL a HTTP metod, parsování cookies a HTTP požadavků, zároveň podporuje vývoj za použití architektury MVC. [29, 32]

2.1.1.4 Go

Programovací jazyk Go, nebo také Golang, byl vytvořen v roce 2007 a jeho autory jsou Robert Pike, Kenneth Lane Thompson a Robert Griesmer. Od roku 2009 je publikován a vyvíjen jako open source software. Na jeho vývoji se velkou mírou podílí společnost

Google. Go je staticky typovaný jazyk, který ale obsahuje některé prvky dynamicky typovaných jazyků. Zároveň není objektově orientovaný a neobsahuje třídy. I přes to podporuje objektově orientovaný styl vývoje. To umožňuje díky konstruktům zvaným rozhraní. Je silně inspirován jazykem C a cílí na systémové programování. Stejně jako Java využívá pro správu paměti garbage collector. Charakteristickým znakem jazyka Go je krátká doba kompilace. [29, 33]

Stejně jako u předchozích jazyků, tak i pro Go existuje řada webových frameworků. Tím nejvíce používaným z nich je Gin. Mezi problémy, které jako framework řeší, patří například validace JSON formátu v příchozích požadavcích nebo podpora tvorby middleware a dále také přináší aparát pro řešení systému chyb. Dalším zástupcem z řady webových frameworků Go je Fiber. Jeho základy jsou inspirovány webovým frameworkem Express.js pro JavaScript aplikace. Podporuje práci s komunikačním protokolem WebSocket. Cílí na co nejnižší paměťovou zátěž systému. A v neposlední řadě implementuje rate limiting, pro snížení zátěže aplikace klientskými požadavky. [34]

2.1.2 Persistence

Důležitým prvkem webových aplikací jsou data. Konkrétně je třeba zajistit jejich reprezentaci v aplikaci a zachování mimo ni v čase. Vývojáři mají řadu možností, jak tyto požadavky naplnit. Jednou z možností je ukládat data pouze v běžícím procesu. V tomto případě se ale nezachovají pokud by se aplikace ukončila. Další možností je ukládat data do souborů a definovat si vybraný formát pro manipulaci mezi aplikací a soubory. Pro tyto potřeby vznikly databáze, které nejsou nic jiného než řada souborů s danou strukturou. Pro jejich vzájemnou manipulaci vznikly takzvané databázové systémy. Tradičním typem databází, které v mnoha případech vývojáři využívají, se staly relační databáze. Typické systémy pro jejich správu jsou například MySQL, nebo PostgreSQL. Relační databáze s sebou nesou některé nevýhody. Se svým fixním schématem, je náročné takové databáze škálovat. Při narůstajícím objemu dat klesá výkonnost. Pro odstranění těchto nedostatků, vznikaly alternativní řešení k relačním databázím. [35]

Od roku 2009 mají vývojáři k dispozici staronový typ databází Not only SQL, více známe jako NoSQL. V roce 1998 jej představil Carlo Strozzi, ale až o 11 let později uvedl s větším ohlasem Eric Evans. Klíčovým rozdílem NoSQL databází oproti těm relačním je to, že nemusejí mít definované své pevné schéma. Způsobů implementace NoSQL databází je několik. Například využití grafových modelů, mapování klíč-hodnota, nebo pomocí dokumentů sdružujících se v kolekcích. Mezi nejběžněji používané NoSQL databáze patří MongoDB a Cassandra. I tento druh databází se neubránil nevýhodám svého přístupu. Oproti relačním databázovým systémům mají méně funkcí a nižší nároky na konzistenci dat. [35]

2.2 Frontend

Uživatelské rozhraní na straně klienta zprostředkovává komunikaci s webovou aplikací. Je tak důležité, aby byl tento zážitek plynulý a bezproblémový. Pro vývoj webových stránek se využívá HTML ve spojení s JavaScriptem. Stejně jako u backendu, tak při vývoji frontendu se vývojářům opakuje hrstka problémů, které musí vždy řešit. To přímo vede k nutnosti vzniku frameworků, které tento proces usnadňují. Třemi nejužívanějšími jsou

React.js, Vue.js a Angular. Dle [36] byl v květnu roku 2018 React tím nejčastěji používaným z nich, jak lze vidět v tabulce 2.1. React je framework vyvinutý společností Meta pro jejich vlastní potřeby. Od roku 2013 je ale dostupný jako open source software. Druhým nejpoužívanějším z tabulky 2.1 je Angular. Open source framework pochází z dílen společnosti Google a veřejnosti je k dispozici od roku 2010. Ve své nové verzi se Angular již nespoléhá na JavaScript, ale využívá pro své potřeby TypeScript. Ten na JavaScriptu sice staví, ale rozšiřuje jeho schopnosti. Mezi zásadní funkce, které TypeScript přidává, patří statické typování. Jelikož se jedná o nadstavbu nad JavaScriptem, je tak každý kód psaný v JavaScriptu také kódem kompatibilním s TypeScriptem. [36, 37]

Tabulka 2.1 Github statistiky použití frontend JavaScript frameworků. Získáno z [36]

	Angular	React	Vue
Počet stažení v milionech	4	9,2	1,5

Kapitola 3

Analýza stávajícího řešení

K procesu vytváření aplikace lze přistupovat mnoha způsoby. Nejvíce vágním je vytvoření seznamu požadavků na aplikaci v textové formě. Z těchto instrukcí je možné jen velmi obtížně vytvořit návrh kýžené aplikace. Proto je žádoucí tento text zpracovat do přesnější formy tak, aby jak strana zhotovitele tak strana zadavatele měly jasnou představu o tom, jaké požadavky z textu plynou. Jedním z mnoha způsobů, jak tento text upřesnit, je analýza funkčních a nefunkčních požadavků. Ty slouží k jasnému definování vlastností nové aplikace. Dalším možným krokem pro zpřesnění požadavků je uvedení případů užití. Jedná se o scénáře, které popisují způsob interakce uživatele a aplikace.

Při vytváření nové aplikace takzvaně na zelené louce, jsou hlavními zdroji informací zadavatelé. Ti ve společné komunikaci s analytiky zpracovatele vytváří právě takovou dokumentaci, jejíž cílem je přesně specifikovat dané požadavky. Na jejich základě pak může vzniknout aplikace s takovými parametry, na kterých se obě strany shodly.

Zásadní rozdíl nastává pokud cílem není vytvořit novou aplikaci, ale revitalizovat tu stávající. Nestačí tedy pouze požadavky z textu převést do nějaké jednoznačné formy. Prvním krokem musí být analýza již existujícího řešení. Je velmi důležité důkladně prozkoumat a respektovat parametry aktuální aplikace. Primárním zdrojem informací tedy není zadavatel.

V případě této práce se jedná o druhý ze zmíněných scénářů. Jelikož práce staví na již existujícím řešení, je potřeba nejdříve provést jeho analýzu. Z té vzejdou na světlo nejen parametry pro nové řešení, ale také neduhy toho stávajícího.

Aktuální řešení využívá několik základních technologií. Samotná webová služba se skládá z Linuxového serveru, Apache HTTP Serveru a aplikace zpracovávající klientské požadavky. První dvě zmíněné komponenty jsou již hotová řešení, která jsou součástí architektury. Neskrývá se v nich tedy logika aplikace a není třeba je hlouběji analyzovat.

Aplikace samotná funguje na tom principu, že klientům přistupujícím pomocí webových prohlížečů jsou na jejich požadavky vráceny již hotové HTML stránky s veškerými potřebnými daty a jedná se tedy o metodu zvanou server-side rendering. S veškerou výpočetní zátěží se musí vypořádat server a klient pouze přijímá požadované stránky s veškerým obsahem. To klade vyšší nároky na výkonnost serveru. Zatímco klient tak musí pouze zajistit dostatečně rychlé internetové spojení pro příjem. Během probíhající konference je třeba zajistit souběžný přístup vyšším desítkám klientů. [38]

Aktuální systém neimplementuje ve svém zdrojovém kódu žádný konvenční architek-

tonický návrh. Celá aplikace je tak úzce provázaná a to sebou nese řadu nevýhod. Tou klíčovou je obtížné testování. Je velmi náročné odzkoušet a ověřit funkčnost nově vytvořené komponenty díky velké závislosti na těch již existujících. Zároveň vývoj nových částí samotný je značně ztížen. Zakomponovat je do stávajícího systému je náročné. Nakonec každá změna ovlivňuje celou aplikaci. Můžou být způsobeny změny, které vývojář původně ani nezamýšlel a ve spojitosti s obtížným testováním je prakticky nemožné je objevit včas.

Kód aplikace je psaný pomocí jazyků PHP a JavaScriptu, jenž využívá knihovnu jQuery. PHP generuje elementy HTML, které skládá hierarchicky k sobě a vytváří tak výslednou stránku. Takto vysoká míra provázanosti stěžuje správu a potenciálně i budoucí vývoj.

Pro persistenci dat se v aktuální aplikaci nevyužívá žádná dostupná databáze. Veškerá data jsou uložena v běžných textových souborech s jejich vlastní strukturou. V nich jsou pak kódována do JSON formátu. Vývojáři tedy přichází o veškeré výhody související s operacemi s daty pomocí databází. Každou funkcionalitu je tak třeba zvlášť implementovat vlastními silami. U současného řešení existuje při manipulaci s daty několik zásadních problémů. Největším problémem aktuálního systému persistence je úprava souborů, které data skladují. Není vyřešeno, jak by se měl soubor zachovat v situaci, kdy k němu přistupuje synchronně 2 a více uživatelů a snaží se jej editovat. Může dojít k takovému nedefinovanému chování, že data v souboru nebudou konzistentní. Dalším neduhem persistence je podstata dat, které se v souborech ukládají. Ve většině případů obsahují pouze kýžené informace o dané entitě. Nicméně některá data jsou obalena do HTML elementů a je tak v databázi rovnou definováno, jak se má daný prvek zobrazit. Ukázka takových dat z aktuální aplikace je znázorněna ve výpisu kódu 3.1. Jedná se o jeden z příkladů úzké provázanosti aplikace.

Výpis kódu 3.1 Ukázka provázaných dat aktuálního řešení

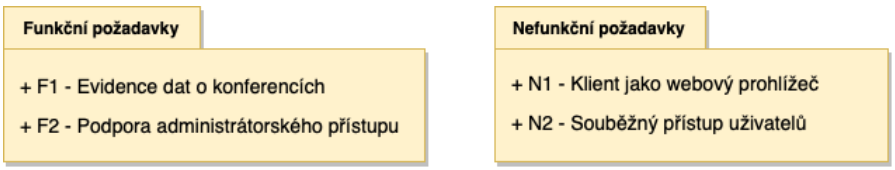
```
{
    "date": {
        "value": "<b style='color:#ffaa00;font-size:18px'>December 4,
        XXXX. Online only.<\/b>",
        "type": "text",
        "id": "date",
        "desc": "Conference date",
        "group": "main"
    },
}
```

Z analýzy zdrojových kódů webového portálu vyplývá model entit, které se v systému nacházejí. První z nich je samotná konference. V současném řešení vznikají HTML stránky pro každou novou konferenci znovu. Tento proces je automatizovaný pomocí nástroje make, který celý projekt sestaví. Na vybraných místech například změní rok konání podle přednastavených parametrů. O konferenci je potřeba evidovat mimo jiné datum začátku a konce konání, její název a popis v aplikaci zvaný charakteristika. Každá konference má vždy svůj program. Ten je vytvářen pořadateli. Program se skládá z přednášek a speciálních událostí, kterými jsou například přestávky, úvod a závěr konference. V současném systému má implicitně každá přednáška přidělených 20 minut trvání. O přednášce jsou ukládány informace jako je její název a abstrakt. Zároveň si ke každé přednášce

může její řečník vybrat, jestli chce přidat i poster, nebo o to nemá zájem. Pro přednášku je také důležitý řečník. O něm je v systému ukládáno jméno, emailová adresa, fakturační adresa a univerzitní příslušnost. Dalším důležitým atributem řečníka je jeho přítomnost. Řečník má možnost zúčastnit se svou prezentací fyzicky na místě konání, nebo online prostřednictvím internetového spojení. Poslední entitou vystupující v systému je účastník dané konference, který je pouze v roli návštěvníka a pasivního posluchače nebo diváka. Informace shromažďované o návštěvnících jsou naprosto identické jako o řečnících. Jediný rozdíl mezi nimi spočívá v tom, že návštěvníci nemají žádné povinnosti vůči programu v podobě prezentací.

V systému je také podporován administrátorský přístup. Uživatel s takovými právy má rozšířené možnosti oproti běžným uživatelům. Stejně jako oni může entity vytvářet a číst, ale jen administrátor může data editovat a odstraňovat.

Díky těmto jmenovaným vlastnostem je možné identifikovat a stanovit funkční a nefunkční požadavky aktuálního webového portálu. Ty je pak potřeba zachovat i při návrhu nového řešení. Jak lze vidět na obrázku 3.1 mezi funkční požadavky patří evidence dat o konferenci a administrátorský přístup. Evidence dat podporuje správu informací všech entit, které jsou určeny v této analýze. Administrátorský přístup povoluje danému uživateli operace a akce nad rámec běžného uživatele. Například zpřístupňuje možnost entity mazat a editovat. Nefunkčními požadavky jsou klientský přístup skrze webový prohlížeč a specifikace odhadu maximálního počtu souběžných přístupů.



Obrázek 3.1 Model funkčních a nefunkčních požadavků

Z analýzy vyplynulo, že aktuální aplikace je jen těžko rozšiřitelná a dále spravovatelná pro nově příchozí vývojáře, kteří by se chtěli na projektu podílet. Byly identifikovány entity, které je potřeba zachovat v novém návrhu včetně jejich parametrů a atributů. Zároveň byly také určeny jejich vzájemné vazby a návaznosti. Současně byly stanoveny funkční a nefunkční požadavky, které aktuální systém má a je nutné se jimi řídit při tvorbě návrhu nového řešení webového portálu.

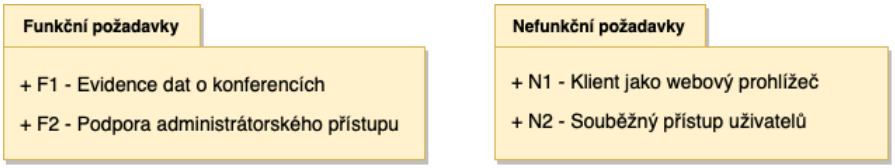
Kapitola 4

Návrh nového řešení

Na základě předchozí kapitoly, která se věnovala analýze stávajícího řešení, lze vytvořit návrh nové aplikace. Pro tu je klíčové, aby respektovala parametry a požadavky aktuálního systému vyplývající z analýzy.

4.1 Funkční a nefunkční požadavky

Jako první krok při vytváření návrhu nové aplikace je potřeba analyzovat funkční a nefunkční požadavky, které jsou na začátku jasně definované. Z analýzy současného systému byly takové základní požadavky určeny. Na obrázku 4.1 lze vidět, že pro dodržení funkčních požadavků je třeba zajistit možnost evidovat data o konferencích a všech údajích, které s tím souvisí. Dále pak podporovat rozdělení oprávnění uživatelů na běžné a administrátorské. Druhé zmíněné by měly umožňovat uživateli provádět operace nad rámec těch běžných, jako je například pouhé čtení dat. Pro naplnění nefunkčních požadavků je nutné navrhnout a implementovat klienta jako komponentu přistupující pomocí webového prohlížeče. Nakonec musí být zajištěno, aby systém podporoval desítky takových souběžných přístupů v daný moment.



Obrázek 4.1 Model funkčních a nefunkčních požadavků

Pro splnění funkčních požadavků je třeba navrhnout odpovídající řešení. Aby byl naplněn první funkční požadavek o evidenci dat, jsou ona data v nové aplikaci ukládána a spravována v příslušné databázi. Za pomocí vytvořené aplikace v roli prostředníka je s nimi možné provádět operace. Druhý funkční požadavek, který specifikuje administrátorský přístup, je dodržen zavedením autentizačního mechanismu. Ten zajišťuje, aby takovému uživateli, který se dokáže prokázat zvoleným způsobem, byly přístupné vybrané operace s daty. Splnění nefunkčních požadavků lze docílit v rámci implementace.

Například zajištění podpory synchronních přístupů lze pomocí vhodných technologií, které tuto funkčnost podporují.

4.2 Případy užití

Úkolem webového portálu pro pořádání vědeckých konferencí je implementovat procesy, které jsou definovány pořadateli. Z analýzy stávajícího řešení vyplývá, které entity v systému existují a jak spolu interagují. Těmi základními entitami jsou:

- Přednáška (Talk)
- Konference (Conference)
- Program (Programme)
- Člověk/Účastník (Person)

S těmito entitami je třeba v systému operovat. Tyto operace provádí klientský počítač, který lze identifikovat jako uživatele webového portálu. Na základě funkčních požadavků je nutné jej rozdělovat na dva typy. Prvním je běžný uživatel, který má k operacím omezený přístup. Druhým je administrátor, jehož pravomoci jsou rozšířené o operace, mezi které se řadí například editace entit.

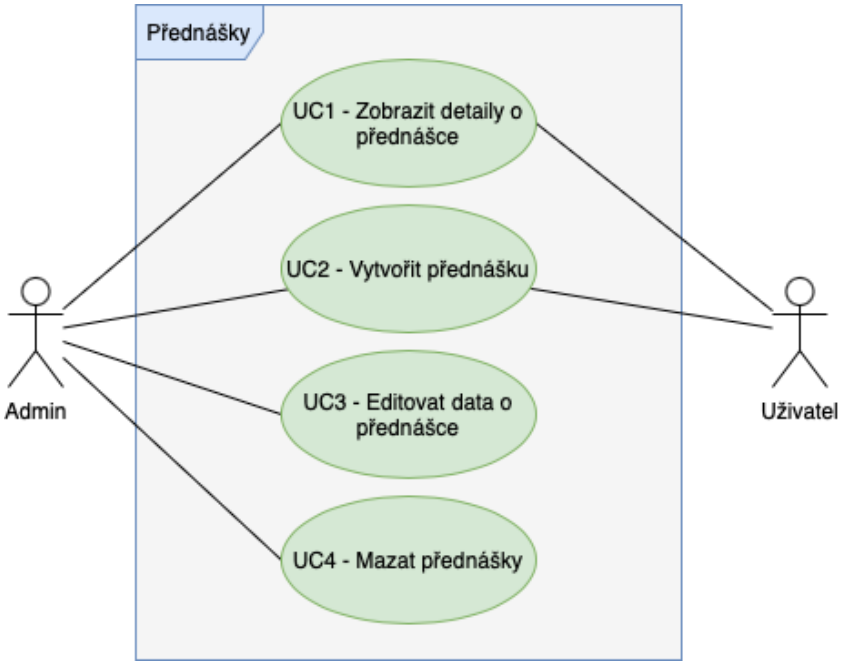
V následujících diagramech případů užití, jsou definovány vztahy mezi aktory a operacemi, které jsou jim přístupné. Prvním stanoveným aktorem je Admin. Jedná se o uživatele, který je v systému autentizovaný a vystupuje s administrátorským přístupem. Druhým je Uživatel. V tomto případě se jedná pouze o běžného uživatele, který má oproti Adminovi omezené funkce.

Na diagramu na obrázku 4.2 jsou definovány scénáře, které může provádět daný aktor. Zobrazovat a číst detaily přednášek můžou oba uživatelé vystupující v systému. Zároveň mají také oba pravomoci k vytváření přednášek. Zásadně se liší v tom, že pouze Admin může vytvořené přednášky editovat a odstraňovat.

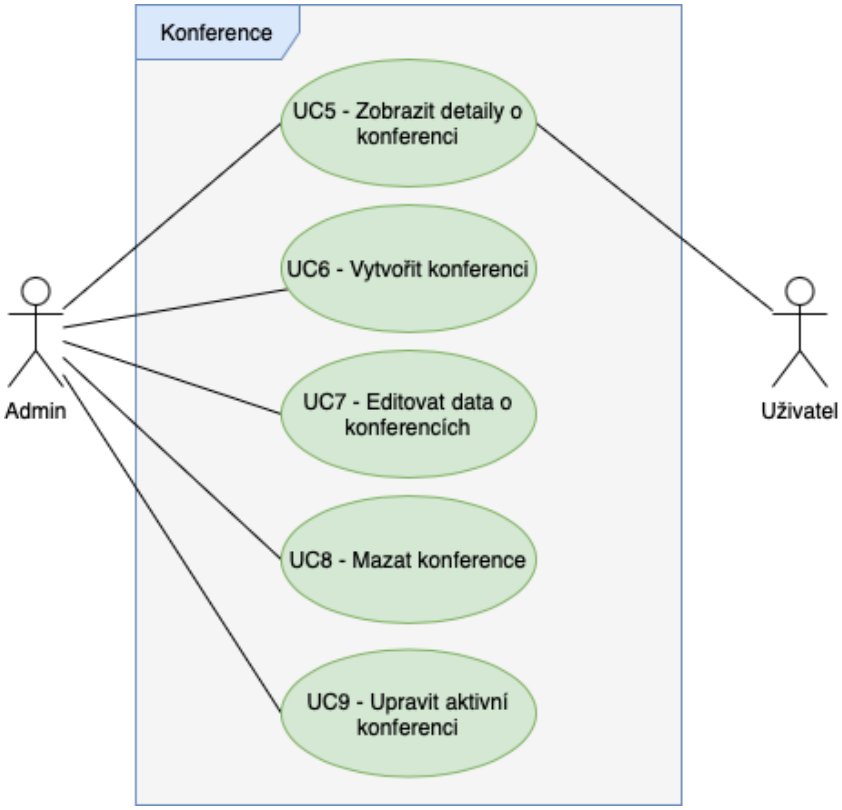
Z diagramu, který je znázorněn na obrázku 4.3, vyplývá, že Uživatel má ještě menší pravomoci ve vztahu k entitě konference než k entitě přednášky, a může taková data pouze číst. Oproti tomu Admin má práva nejen data o konferenci číst, ale také vytvářet, editovat a odstraňovat. Navíc k tomu, má možnost zvolit jednu z konferencí jako aktivní. To znamená jako tu, která v danou dobu probíhá a nebo je otevřená k přihlašování.

Na diagramu z obrázku 4.4 je znázorněno, že uživatel má opět pouze práva ke čtení dat o programech. Admin na druhou stranu může veškerá taková data nejen číst, ale také vytvářet, editovat a odstraňovat ze systému.

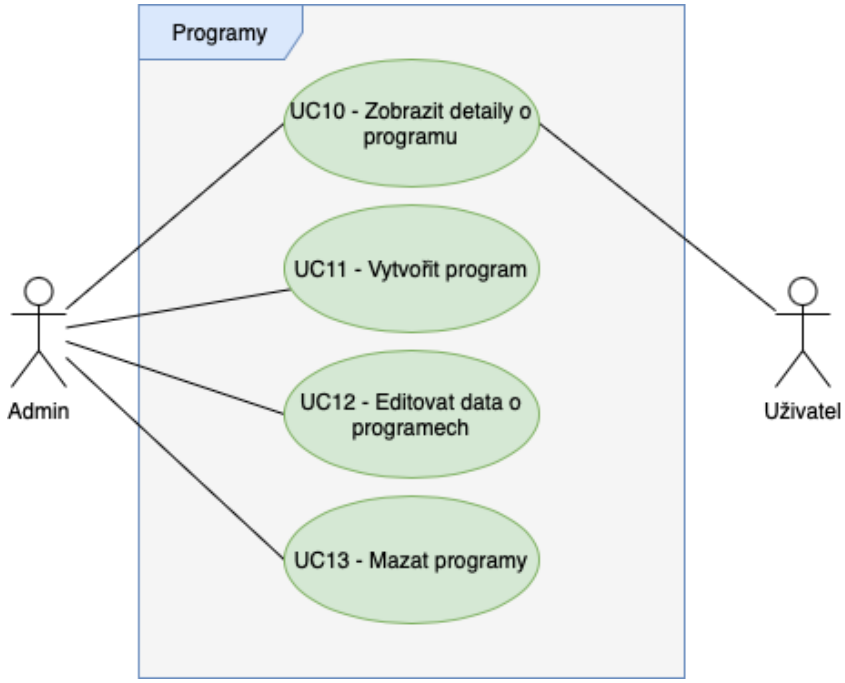
V posledním digramu na obrázku 4.5 je vytvořen návrh práv Uživatele a Admina ve vztahu k entitě lidí, nebo také účastníků konference. V tomto případě, stejně jako ve všech ostatních, má Uživatel práva data o lidech číst. Důležitou operací, kterou může Uživatel navíc vykonávat je registrace do systému, aby se mohl zúčastnit konference buďto v roli řečníka nebo návštěvníka. Admin má kromě toho možnost lidi v systému editovat a odstraňovat z něj.



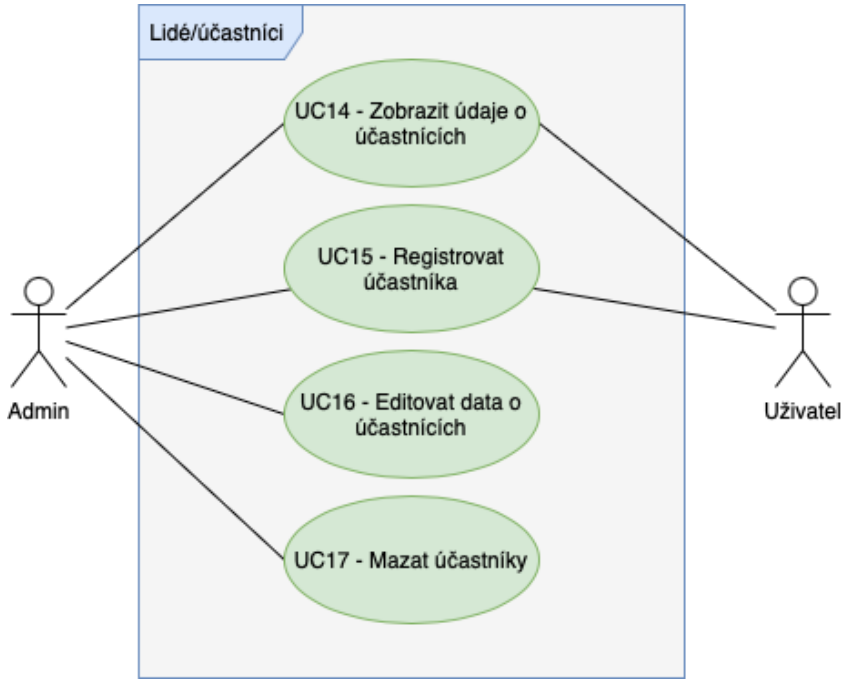
Obrázek 4.2 Diagram případů užití entity přednáška



Obrázek 4.3 Diagram případů užití entity konference



Obrázek 4.4 Diagram případů užití entity program

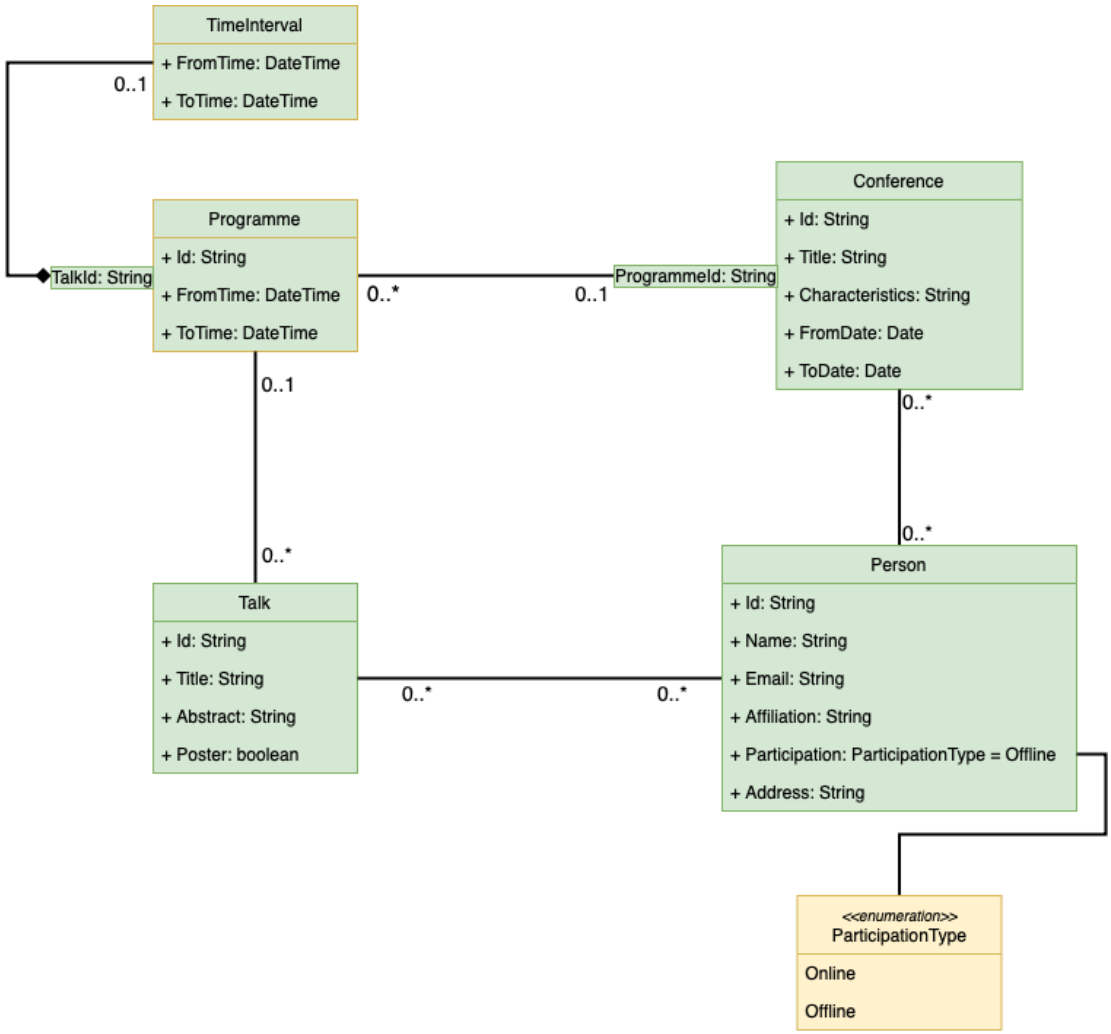


Obrázek 4.5 Diagram případů užití entity lidé

4.3 Doménový model webové aplikace

Na základě analýzy stávajícího řešení, je možné identifikovat 4 primární entity, které jsou nezbytné pro naplnění všech požadavků. Jak již bylo zmíněno jedná se o konferenci, program, přednášku a řečníka nebo návštěvníka. Poslední uvedená entita je ve své podstatě člověk, který vystupuje v systému ve dvou různých rolích. V obou případech je ale nutné shromažďovat identické informace jak o řečnících, tak o návštěvnících. Návrh musí tento fakt brát v potaz, aby nedocházelo ke zbytečné duplikaci dat.

Existuje několik způsobů, jak tyto požadavky a omezení zachytit ve vhodné a srozumitelné formě. Pro potřeby této analýzy jsou požadované koncepty, entity a jejich vzájemné vztahy vyobrazeny v doménovém modelu 4.6.



Obrázek 4.6 Doménový model webové aplikace

Na obrázku 4.6 doménového modelu jsou znázorněny vztahy nejen entit, jejichž nutnost zachovat je v novém návrhu vzešla z analýzy, ale také pomocných, které upřesňují některá data v modelu. Aby nedocházelo v systému k duplikaci dat řečníků a návštěvníku, existuje v doméně pouze entita Person. Její následná role se odvíjí ve vztahu

k ostatním entitám. Pokud existuje vazba Person na Conference, jedná se tak o pouhého návštěvníka. Na druhou stranu pokud existuje vazba Person a Talk, pak jde o řečníka, který na sebe navazuje nějakou přednášku. Díky tomuto návrhu nedochází k duplikaci dat, pokud je člověk zároveň řečníkem a návštěvníkem.

V modelu je také entita TimeInterval, která se nijak nevyskytovala v předchozí analýze. Jedná se pouze o entitu sdružující 2 časy tvořící časový interval. Programme pak mapuje každý Talk, se kterým je ve vzájemné vazbě, na patřičný časový interval. V původní aplikaci je vyhrazeno na každou přednášku 20 minut. Tímto způsobem lze docílit toho, aby bylo možné každé přednášce zvlášť určit její časové rozmezí, které vyžaduje.

4.4 Návrh API pro komunikaci

Pro komunikaci mezi klientem a serverem slouží aplikační rozhraní zvané API. Tento návrh má za cíl implementovat REST API namísto SOAP. Jedním z důvodů, proč je ke komunikaci zvoleno REST API, je menší míra jeho komplexnosti oproti SOAP. Zároveň práce s JSON působí mnohem lehčím a jednodušším dojmem pro nově příchozí vývojáře než zpracovávání XML.

V rámci implementace jsou využívány 4 základní operace. Těmi jsou vytváření entit, jejich čtení, upravování a mazání. Takovým operacím se přezdívá CRUD operace. Jedná se o zkratku anglických slov Create, Read, Update, Delete. V případě protoklu HTTP je možné přirovnat tyto operace jejich ekvivalentům v HTTP metodách.

- POST je ekvivalentem pro vytváření entit
- GET slouží pro čtení entit
- PUT je HTTP metodou pro editaci entit
- DELETE se využívá pro odstraňování entit

V následujících tabulkách jsou popsány veškeré způsoby, kterými uživatel může modifikovat entity vyobrazené v doménovém modelu. Nejdříve je uvedena metoda protokolu HTTP, kterou je třeba uvést v hlavičce požadavku. Následně URI, které jednoznačně definuje přístup k vybranému zdroji. Notace složených závorek, ve kterých je ve všech případech uvedeno id, značí, že se jedná o proměnný parametr, za který má uživatel v tomto případě dosadit právě id. Nakonec je uveden stručný popis akce, kterou zavolání zdroje vyvolá. Zároveň také uvádí, které informace je třeba zaslat v hlavičce nebo tělu požadavku.

V tabulce 4.1 lze pozorovat, jakým způsobem je navržena komunikace pomocí API s entitou Person neboli účastníkem nějaké konference. Návrh respektuje konvence klasického mapování CRUD operací na HTTP metody. Každé vytvoření a odeslání požadavku, na zde specifikovanou adresu, vyvolává akce na straně webové aplikace. Jelikož se jedná o návrh respektující diagramy užití, je třeba rozlišovat, které zdroje jsou přístupné daným uživatelům. Administrátor má možnost využívat všechny zdroje, zatímco běžný uživatel může využívat pouze zdroje pod metodami GET a POST.

Tabulka 4.2 popisuje návrh API komunikace pro práci s entitou Conference. V mnohém se neliší od návrhu API pro entitu Person, ale podporuje některé funkce navíc.

Tabulka 4.1 Návrh API pro správu účastníků

HTTP Metoda	URI	Popis
GET	/person	Vrací veškeré účastníky v systému
GET	/person/{id}	Vrací účastníka s daným id
POST	/person	Validuje, vytvoří a uloží účastníka zadaného v těle požadavku
PUT	/person/{id}	Validuje a aktualizuje účastníka s daným id podle dat z těla požadavku
DELETE	/person/{id}	Odstraní účastníka s daným id

Tabulka 4.2 Návrh API pro správu konferencí

HTTP Metoda	URI	Popis
GET	/conference	Vrací veškeré konference v systému
GET	/conference/{id}	Vrací konferenci s daným id
POST	/conference	Validuje, vytvoří a uloží konferenci zadanou v těle požadavku
PUT	/conference/{id}	Validuje a aktualizuje konferenci s daným id podle dat z těla požadavku
DELETE	/conference/{id}	Odstraní konferenci s daným id
GET	/conference/active	Vrací id aktuálně plánované konference
POST	/conference/activate/{id}	Validuje existenci konference s daným id a aktivuje ji
DELETE	/conference/deactivate	Odstraní id aktuálně aktivní konference

Jelikož oproti stávajícímu systému je možné ukládat data o více než jedné konferenci, je třeba mít aparát, kterým lze určit tu právě aktivní konferenci. K tomu slouží poslední tři zdroje z tabulky 4.2. Pomocí tohoto návrhu tak lze jednoduše přepínat aktuálně plánované nebo běžící konference. I v tomto případě má uživatel s administrátorským přístupem neomezené pravomoci na využívání všech navržených zdrojů. Oproti tomu běžnému uživateli jsou dostupné pouze první tři zdroje pro čtení dat a jejich vytváření.

Tabulky 4.3 a také 4.4 využívají u svých návrhů vesměs totožné struktury rozhraní. Cílem této unifikace je zjednodušení přístupu pro vývojáře. Ve všech čtyřech předchozích návrzích je určena sada pěti zdrojů, u kterých lze z podstaty očekávat obdobné chování při zaslání požadavku. Díky tomu lze s aplikací snadno a přehledně interagovat. I v návrhu těchto rozhraní má administrátor možnost využívat všechny uvedené zdroje. Běžný uživatel je omezen na všechny zdroje přístupné přes metody GET a POST.

Tabulka 4.3 Návrh API pro správu programů

HTTP Metoda	URI	Popis
GET	/programme	Vrací veškeré programy v systému
GET	/programme/{id}	Vrací program s daným id
POST	/programme	Validuje, vytvoří a uloží program zadaný v těle požadavku
PUT	/programme/{id}	Validuje a aktualizuje program s daným id podle dat z těla požadavku
DELETE	/programme/{id}	Odstraní program s daným id

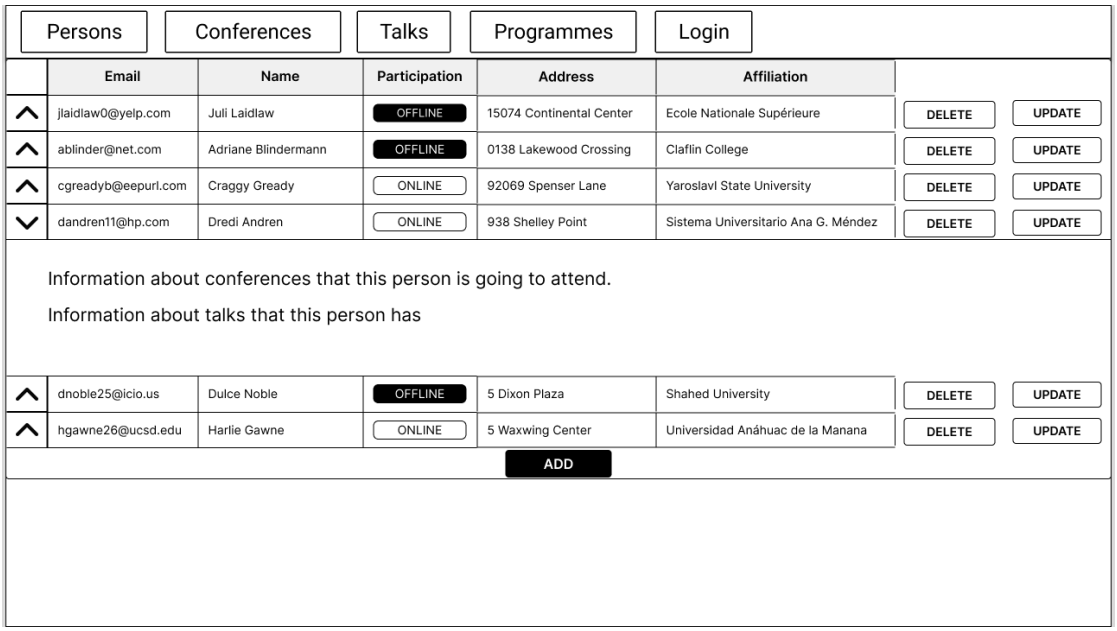
Tabulka 4.4 Návrh API pro správu přednášek

HTTP Metoda	URI	Popis
GET	/talk	Vrací veškeré přednášky v systému
GET	/talk/{id}	Vrací přednášku s daným id
POST	/talk	Validuje, vytvoří a uloží přednášku zadanou v těle požadavku
PUT	/talk/{id}	Validuje a aktualizuje přednášku s daným id podle dat z těla požadavku
DELETE	/talk/{id}	Odstraní přednášku s daným id

4.5 Frontend wireframe

Pro potřeby demonstrace funkčnosti nově navrženého a implementovaného backendu, je žádoucí vytvořit uživatelské rozhraní, které uživateli zpřístupní základní nástroje a bude tak možné prokázat splnění stanovených cílů. K zachycení právě takového návrhu se využívá wireframe.

Na obrázku 4.7 je zachycen wireframe popisující nový návrh uživatelského rozhraní. Konkrétně se jedná o zobrazení dat souvisejících s entitou Person. V tomto návrhu jsou zobrazena základní data společně s akčními tlačítky. Ta následně modifikují vybranou entitu. Při stisknutí tlačítka DELETE dojde k odstranění daného záznamu z tabulky a z databáze. Pokud uživatel stiskne sousedící tlačítko UPDATE, dostane možnost upravit data vybraného řádku. Zadaná data se následně zpracují a zašlou na server. Ten ověří zda jsou data z požadavku validní a vrátí patřičnou odpověď. Celý proces je téměř identický pro akci spouštěnou tlačítkem ADD. Jediný rozdíl spočívá v tom, že uživatel needituje existující data, ale musí zadat zcela nová. Tlačítko Login poskytuje možnost přihlášení se a získání přístupu k administrátorským operacím. Jedné co musí uživatel znát je heslo.

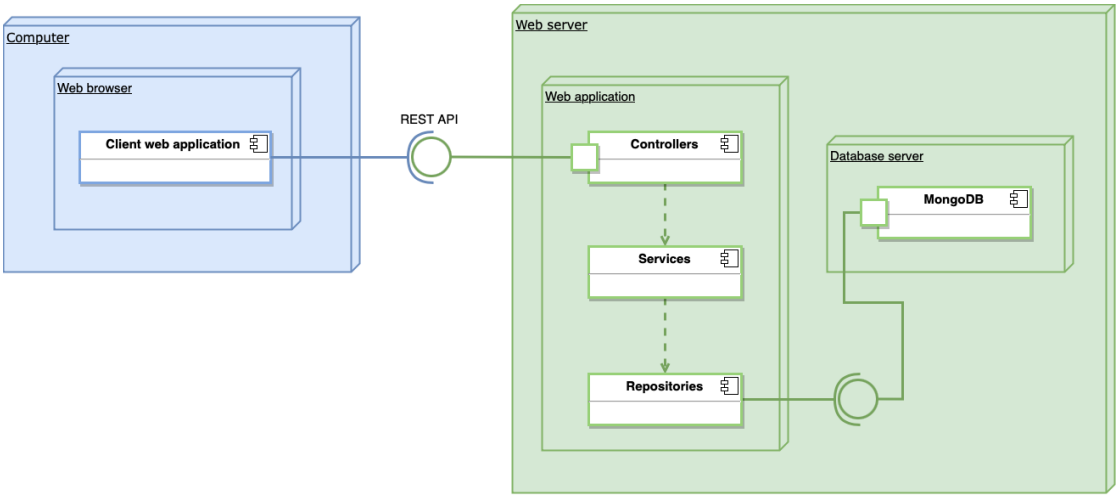


Obrázek 4.7 Wireframe uživatelského rozhraní pro entitu Person

Kapitola 5
Implementace

Z kapitoly věnované analýze vyplývá, jaké požadavky je třeba respektovat při návrhu nového řešení realizace webového portálu pro organizaci vědeckých konferencí. Z kapitoly věnující se takovému návrhu vzešlo konceptuální schéma a model udávající technologicky nezávislé koncepty, na kterých má výsledná implementace stavět. V rámci implementace je nutné zvolit vhodné technologie, které budou naplňovat požadavky dané návrhem.

Vhodným nástrojem pro zachycení plánů popisujících konkrétní komponenty navrhovaného systému je například diagram komponent.



Obrázek 5.1 Diagram komponent webové aplikace

Na obrázku 5.1 lze vidět znázorněný diagram komponent sestavený na základě předchozích kapitol. Tento návrh komponent respektuje veškeré koncepty stanovené a určené v návrhu nového systému. Implementuje REST API pro komunikaci klienta a serveru. Data aplikace se ukládají v dokumentově orientované NoSQL databázi MongoDB. Dále je využíván architektonický vzor MVC. Controller tak přijímá požadavky od klienta a hierarchicky je zpracovává a posílá dalším komponentám. Service v sobě skrývá business logiku aplikace. Komunikuje dále s komponentou Repository. Ta slouží pro zprostředkování operací mezi webovou aplikací a databází.

5.1 MongoDB

V rámci volby vhodné technologie pro implementaci databázové komponenty systému, padla volba na řešení MongoDB. Jedná se o populární technologii mimo konvenční SQL databáze. Primárním faktorem, který ovlivnil tuto volbu, je jeho jednoduchost pro prototypování. Díky jeho vlastnostem nevyžaduje žádné pevně dané schéma a dokumenty je tak snadné vnořovat jeden do druhého. Místo entit, které by musely mít záznam a mezi sebou vzájemné vazby v relační databázi, je v MongoDB možné vložit objekty do sebe přímo jako atributy. Toho je využito pro entitu uvedenou v doménovém modelu na obrázku 4.6 TimeInterval. Jelikož je vždy entita TimeInterval vázána na entitu Programme, která má vazbu alespoň na jednu entitu Talk, a nikdy jindy existovat nemůže, je možné ji přímo vložit do Programme bez nutnosti zavádět její vlastní kolekci. Při vkládání objektů do dokumentů a kolekcí, není ve výchozím nastavení žádné omezení, které by podmiňovalo zápis do databáze. Takové vlastnosti dělají z MongoDB velmi flexibilní řešení. Někdy je ale žádoucí, aby kolekce měly stanovená pravidla, podle kterých se operace nad nimi řídí. K tomuto restriktivnímu přístupu lze využít kolekcím nastavitelné schéma. To zavádí povinnosti všem objektům dané kolekce dodržet definovanou strukturu. V nové implementaci tato schémata nejsou využitá. Jedná se o variantu, která přináší budoucím vývojářům větší míru svobody pro úpravu celé domény, bez nutnosti měnit kód na mnoha místech. Možnou nevýhodou MongoDB oproti klasickým relačním databázím je ten fakt, že podporuje méně funkcí. Nemá například vestavěné nástroje pro spojování dokumentů. Zatímco relační databáze tyto operace podporují.

Nové řešení využívá pro každou základní entitu (Conference, Programme, Talk, Person) svou vlastní kolekci. V budoucnu je tak možné přistoupit k více restriktivnímu omezení a nastavit kolekcím jejich schéma pro dokumenty.

5.2 Go

Při výběru jazyka pro implementaci webové aplikace, která zajišťuje komunikaci nejen s databází, ale zároveň také obstarává příchozí klientské požadavky, padla volba na programovací jazyk Go. Ten splňuje základní požadavky na jazyk, který je k tomuto projektu vhodný. Mezi takové patří například to, že pro daný jazyk musí existovat kvalitní webový framework, který podporuje tvorbu takové aplikace. Dalším kritériem jazyka je jeho snadné pochopení a naučení se. Go se velmi silně inspiruje syntaxí jazyka C. Ten se v rámci studia na Fakultě informačních technologie ČVUT vyučuje povinně. Pro všechny absolventy těchto kurzů je tak porozumění Go přímočaré. Oproti jiným programovacím jazykům Go neobsahuje mnoho abstrakcí datových struktur. Například neimplementuje kolekci množiny. Náročnost naučení se takového jazyka je nízká, ale nese s sebou právě nevýhodu v úbytku funkčnosti. Jedním z dalším kritérií pro výběr jazyka byla jeho bezpečnost. Respektive vybrat takový jazyk, který minimalizuje náchylnost vývojářů k tomu napsat chybový kód. K tomu přispívá statické typování, jenž Go vyžaduje. Zároveň využívá automatickou správu paměti pomocí garbage collectoru.

Vývojáři, kteří zvolí pro svou práci programovací jazyk Go, mají od výchozí instalace k dispozici stejnojmenný nástroj pro příkazovou řádku. Ten zpřístupňuje vývojářům mnoho funkcionalit bez nutnosti instalovat software třetích stran. Součástí nástroje je například manažer pro správu závislostí projektu a stahování externích souborů, formá-

tovací šablona a podpora pro testování zdrojového kódu. V rámci otestovaného kódu je možné vygenerovat HTML soubor s popisem udávajícím, která část zdrojového kódu byla a nebyla otestována společně s procentuálním zastoupením vůči danému souboru.

V implementaci nového backendu webového portálu pro pořádání vědeckých konferencím je využit webový framework Gin určený pro programovací jazyk Go. Důvodem pro jeho využití je snazší práce s odpověďmi na klientské požadavky. V případě, že se vývojář rozhodne využít jen a pouze prostředky ze standardní knihovny, čeká jej řada překážek při práci s nativními konstrukty jazyka Go. Gin má již vyřešené techniky, jak odbavovat více klientských požadavků najednou. V pozadí využívá gorutiny, takzvaná lehká vlákna. Díky tomu tak odpadá potřeba řešit tento technologický problém místo soustředění se na vytváření backendu samotného.

Způsob, jakým je v aplikaci řešena autentizace, je pomocí JSON Web Tokenu (JWT). Jedná se o cestu, jak ověřit, že daný klientský požadavek přichází od ověřeného uživatele. V rámci zachování bezpečnosti má JWT Token stanovenou dobu platnosti. Po té je zneplatněn a není možné se jím dále prokazovat. Uživatel si tak musí vyžádat nový tím, že se znovu přihlásí. Dobu platnosti lze nastavovat na libovolně dlouhé, nebo krátké časové období.

5.2.1 Testování

Důležitým aspektem každé aplikace je její testovatelnost. V nové implementaci je na tuto oblast brán zřetel a to díky využití architektonického vzoru MVC. Použitím jasně definovaných zodpovědností každé části, je možné tyto části nezávisle na sobě testovat. Je nutné mít možnost každou vrstvu nahradit takzvaným mockem. Jedná se o imitaci původní vrstvy, jenž má dopředu jasně definované chování. Tímto způsobem lze docílit otestování vybraných komponent.

Nástrojem, který značně zpřehledňuje napsané testy, je testify. Jedná se o balíček přinášející metody pro psaní čitelnějších a jasnějších testů v jazyce Go. Testify využívá při psaní testů jednoduché příkazy assert pro kontrolu odpovídajícího obsahu. Dalším nástrojem, který je v implementaci využíván přímo ve spojení s testify, je balíček mockery. Tento nástroj slouží ke generování imitací sloužících k nahrazení a testování původních vrstev. Spolupráce mockery a testify spočívá v možnosti definovat na imitovaných objektech generovaných pomocí mockery, jakým způsobem se mají chovat při vybrané akci. Díky tomu lze psát efektivně a spolehlivě veškeré testy.

Korektnost nové aplikace je ověřována jednotkovými testy. Ty mají za úkol ověřit funkčnost vždy vybrané sekce zdrojového kódu. Cílem je otestovat nejen pozitivní scénáře, kdy vše skončí bez chyby, ale také cesty, které by měly končit chybou. Zároveň je využit nástroj Go. Ten pod podmínkou úspěšného absolvovaní všech testů, vygeneruje HTML soubor, kde je možné detailněji projít, že byly řádně otestovány veškeré klíčové sekce zdrojového kódu.

5.3 Klient

Úkolem implementace jednoduchého uživatelského rozhraní je zprostředkovat způsob, jakým demonstraci funkce nově vytvořeného backendu. Jelikož se má jednat pouze o podpůrnou funkci, která nemá sloužit k budoucímu vývoji, není potřeba využívat přehnaně

komplexní technologie a postupy. K tomu účelu je pro tvorbu takového rozhraní využit webový framework React.js. Hlavním důvodem, proč volba nepadla na Angluar, nebo Vue je ten fakt, že React má podle statistik z tabulky 2.1 mnohem více uživatelů než ostatní dva zmínění. To znamená, že existuje velká spousta dostupných online materiálů, na které se dá obrátit pro rady v případě nouze. Je k dispozici také mnoho fór, kde si uživatelé Reactu vyměňují zkušenosti nabrané během vývoje.

Místo Reactu nativního JavaScriptu je použit TypeScript. To hlavně pro jeho statické typování. Díky tomu lze stejně jako u Go snížit šanci na potenciální chyby ve zdrojovém kódu.

Ke komunikaci s webovou aplikací pomocí REST API, je využíván na frontendu nástroj axios. Ten slouží pro práci s požadavky klienta. Poskytuje například automatickou serializaci a deserializaci dat v tělech požadavků a odpovědí. Dalším příkladem činnosti podporované nástrojem axios je specifikace metody použité pro pro komunikaci pomocí použité funkce. Axios ve výsledku zjednodušuje a zpřehledňuje zdrojový kód, který na straně klienta slouží ke komunikaci se serverem. Vývojář tedy nemusí pracovat s konstrukty jazyka JavaScript, to zprostředkovává axios.

Posledním rozšířením pro přehledný a čitelný kód na frontendu je Material UI. Jedná se o knihovnu sdružující již navržené vizuální komponenty. Celá tato knihovna je psána přímo pro webový framework React.js. Dává do rukou vývojářům sadu komponent, které přináší do implementace základní funkčnosti. Místo toho, aby vývojáři pokaždé implementovali to samé znovu a znovu, mohou využít prvky z Material UI. To značně přispívá k přehlednějšímu zdrojovému kódu. Použití této knihovny zároveň také usnadňuje a zrychluje vývoj.

5.4 Pomocné technologie

Pro potřeby softwarového vývoje existuje řada pomocných technologií, které pomáhají řešit často se opakující problémy. V tomto projektu bylo použito hned několik takových pomocných nástrojů. Prvním z nich je Git. Jedná se o systém pro správu verzí. Jednoduše lze Git popsat jako nástroj, díky kterému lze různé verze projektu nejen ukládat, ale také se k nim vracet, upravovat je a slučovat. K zachování a sdílení verzování je použit fakultní GitLab.

Dalším nástrojem využitým při vývoji je balíček godotenv. Jeho primární funkcí je načíst do aplikace proměnné nakonfigurované v prostředí, ve kterém je kód umístěn. Tímto způsobem se dají na jednom místě definovat konstanty jako jsou číslo portu, adresa databáze nebo tajná hesla, která by neměla být pevně zadrátovaná ve zdrojovém kódu.

Dokumentace tvoří klíčovou část každého projektu. Speciálně v tomto případě, kdy cílem budoucích projektů bude spojit toto řešení s paralelně vznikajícím frontendem pro webový portál v rámci jiné bakalářské práce. Je tak důležité vytvořit dokumentaci pro realizované rozhraní vzájemné komunikace klienta a server REST API. K tomu je využit framework Swagger. Ten vývojářům slouží k tvorbě dokumentace specificky pro REST API. Pro jeho napojení na projekt psaném v jazyce Go a využívající webový framework Gin existuje nástroj ve formě balíčku gin-swagger. Ten umožňuje vývojářům podle dané specifikace anotovat napsaný zdrojový kód. Na základě této anotace jsou pak vygenerovány soubory, které jsou použity k vytvoření dokumentace. Z těchto souborů

následně vznikne HTML stránka, kde se nachází veškerá vzniklá dokumentace. Tento proces lze tak jednoduše automatizovat při dodržení anotací nově napsaného kódu.

Kapitola 6

Ekonomický přínos

Tato kapitola je věnována zhodnocení ekonomického přínosu nově vytvořeného prototypu, který je výsledkem této práce. Klíčovým faktorem, který ovlivňuje celou podstatu tohoto zhodnocení, je vlastní cíl práce a webového portálu jako takového. Jedná se o nekomerční projekt pro interní potřeby v rámci FJFI (Fakulta jaderná a fyzikálně inženýrská) ČVUT v Praze. Z toho vyplývá, že aplikace není nástrojem pro tvorbu příjmů. Jejím účelem je spíše optimalizace a aktualizace původní aplikace pro nadcházející roky. Z těchto důvodu se jen velmi těžko dá vyjadřovat . Proto se toto zhodnocení věnuje přínosům v podobě například časových úspor, které lze vynaložit na další vývoj a správu nového systému. Pomocí těchto údajů je možné v jakýkoliv okamžik provést přepočet na nějakou úsporu nominální hodnoty. Výhoda toho přístupu tkví v tom, že nevytváří falešnou představu o nic nevypovídajících a imaginárních cenách dosazením arbitrárních čísel.

Prvním přínosem nového prototypu jsou zvolené technologie pro jeho vývoj. Jak již bylo zmíněno, programovací jazyk Go zakládá svou syntaxi na starším jazyce C. V rámci studia na Fakultě informačních technologií musí každý student absolvovat povinné kurzy, které k výuce používají programovací jazyk C. A společně s tím, že Go implementuje pouze omezený počet abstraktních struktur a obsahuje pouhých 25 klíčových slov (pro porovnání Java jich má 50), je pro studenty naší fakulty relativně jednoduché se jej naučit a pracovat s ním. Díky tomu mohou na tento projekt navazovat a rozšiřovat ho v rámci svých bakalářských prací. Není tak potřeba shánět externí vývojáře se zkušeností s jQuery nebo PHP.

Nová aplikace je také snadno rozšiřitelná a modifikovatelná. Díky tomu, že implementuje konvenční architektonické vzory a rozdělují tak aplikaci na úzce provázané vrstvy, je možné takové vrstvy jednoduše nahrazovat. Například pokud by k budoucím účelům organizátorům z různých důvodů nevyhovovala aktuálně zvolená databáze MongoDB, stačí pouze vyměnit vrstvu zajišťující komunikaci s databází a ostatní mohou zůstat beze změny. Z toho plyne další časová úspora nového řešení. Možným rozšířením, které by organizátorům mohlo přinést určitý benefit, je schopnost aplikace napojit různé moduly. Takovým může být analytický modul, díky kterému mohou vyhodnocovat data z proběhlých konferencí.

V neposlední řade je tento prototyp poměrně univerzální. I přesto, že je vytvořen přímo na míru potřebám organizátorů vědeckých konferencí na FJFI ČVUT, nemusí

se pomocí něj pořádat identické akce. Případným vývojářům může v mnoha případech stačit pouze přidat nebo upravit některé atributy daných entit a následně využít systém pro svou potřebu. Zde implementace těží z využití databázového řešení MongoDB a jeho flexibility. Nově vytvořená aplikace se tak dá využít znovu v podobných scénářích s minimálními změnami. Současně je možné na nové aplikaci stavět a rozšiřovat ji o další entity a to díky její univerzálnosti.

Jediným reálným nákladem spojeným s využíváním nové aplikace je způsob hostingu a provozování serveru. V současnosti je možné využívat řady služeb od externích dodavatelů a provozovat aplikaci v cloudu. Důležité je, že tato implementaci od počátku vznikala na základě již existujícího portálu. Ten má veškerý provoz zajištěný lokální na univerzitě. S tímto způsobem nasazení se počítá i pro řešení nové. Tím jediným nákladem je tak spotřeba a cena elektřiny pro běh serveru.

Závěr

Cílem práce bylo revitalizovat backend aktuálního webového portálu pro pořádání vědeckých konferencí. Zároveň k tomu navrhnout a implementovat jednoduché uživatelské rozhraní pro prezentaci funkčnosti nového řešení. Během toho bylo klíčové také respektovat již existující systém a stavět na něm.

Výsledkem práce je backend webové aplikace psaný v programovacím jazyce Go. Aplikace pro persistenci dat využívá NoSQL databázi MongoDB. Veškeré zdroje poskytuje klientům pomocí REST API. Dané API je dokumentováno automaticky prostřednictvím komentářů v rámci zdrojového kódu a knihovny Swagger, která dokumentaci generuje jako HTML stránku pro snadné a přehledné prohlížení. Oproti původní aplikaci, nově vyvinuté řešení využívá prvky, kterými usnadňuje nejen svůj budoucí vývoj, ale také aktuální údržbu. Takovým prvkem je například implementace architektury MVC. Její využití zjednodušuje proces testování, pro které je v práci využita knihovna Mockery a vestavěné nástroje Go. Díky nim lze generovat HTML dokument popisující pokrytí napsanými testy. Dalším výstupem je jednoduchý frontend psaný pomocí knihovny React.js a programovacím jazykem TypeScript. Pro samotné sestavování HTML dokumentů využívá knihovnu komponent Material UI určenou pro React.

V budoucnu by bylo možné navázat na výslednou aplikaci vzniklou v rámci této práce. Jak bylo zmíněno v úvodu, tato práce je jedna ze dvou soustředících se na revitalizaci existujícího webového portálu. V rámci druhé práce vznikla podoba jeho frontendu. Navazující práce by tak mohla mít za cíl sjednotit tyto dva projekty do jedné celistvé aplikace.

Bibliografie

1. JAZAYERI, Mehdi. Some trends in web application development. In: *Future of Software Engineering (FOSE'07)* [online]. IEEE, 2007, s. 199–213 [cit. 2023-04-11]. Dostupné z: https://www.researchgate.net/profile/Mehdi-Jazayeri-2/publication/4250861_Some_Trends_in_Web_Application_Development/links/57d02f5b08ae0c0081dea3bd/Some-Trends-in-Web-Application-Development.pdf.
2. POWELL, John; CLARKE, Aileen. The WWW of the World Wide Web: who, what, and why? *Journal of Medical Internet Research* [online]. 2002, roč. 4, č. 1, e4 [cit. 2023-04-11]. Dostupné z: https://www.jmir.org/2002/1/e4/.
3. OLUWATOSIN, Haroon Shakirat. Client-server model. *IOSR Journal of Computer Engineering* [online]. 2014, roč. 16, č. 1, s. 67–71 [cit. 2023-04-11]. Dostupné z: https://www.researchgate.net/profile/Shakirat-Sulyman/publication/271295146_Client-Server_Model/links/5864e11308ae8fce490c1b01/Client-Server-Model.pdf.
4. DOROFEEV, Dmitriy; SHESTAKOV, Sergey. 2-tier vs. 3-tier Architectures for Data Processing Software [online]. 2018, s. 63–68 [cit. 2023-04-11]. Dostupné z: https://dl.acm.org/doi/pdf/10.1145/3274856.3274869.
5. SABRINA, Ni Kadek Dwi; PRAMANA, Dian; KUSUMA, Tubagus Mahendra. Implementation of Golang and ReactJS in the COVID-19 Vaccination Reservation System. *ADI Journal on Recent Innovation* [online]. 2023, roč. 5, č. 1, s. 1–12 [cit. 2023-04-11]. Dostupné z: https://adi-journal.org/index.php/ajri/article/view/877/608.
6. SONI, Anshu; RANGA, Virender. API features individualizing of web services: REST and SOAP. *International Journal of Innovative Technology and Exploring Engineering* [online]. 2019, roč. 8, č. 9, s. 664–671 [cit. 2023-04-11]. Dostupné z: https://www.researchgate.net/profile/Virender-Ranga/publication/335419384_API_Features_Individualizing_of_Web_Services_REST_and_SOAP/links/5d64960ea6fdccc32cd31171/API-Features-Individualizing-of-Web-Services-REST-and-SOAP.pdf.

7. MUMBAIKAR, Snehal; PADIYA, Puja et al. Web services based on soap and rest principles [online]. 2013, roč. 3, č. 5, s. 1–4 [cit. 2023-04-11]. Dostupné z: https://www.ijsrp.org/research-paper-0513/ijsrp-p17115.pdf.
8. KOSEK, Jiří. Využití webových služeb a protokolu SOAP při komunikaci [online]. [B.r.] [cit. 2023-04-11]. Dostupné z: https://www.kosek.cz/diplomka/html/websluzby.html.
9. NURSEITOV, Nurzhan; PAULSON, Michael; REYNOLDS, Randall; IZURIETA, Clemente. Comparison of JSON and XML data interchange formats: a case study. *Caine* [online]. 2009, roč. 9, s. 157–162 [cit. 2023-04-11]. Dostupné z: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=84321e662b24363e032d680901627aa1bfd6088f.
10. FIELDING, Roy Thomas. Architectural styles and the design of network-based software architectures [online]. 2000 [cit. 2023-04-11]. Dostupné z: https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf.
11. MASSE, Mark. *REST API design rulebook: designing consistent RESTful web service interfaces*. "O'Reilly Media, Inc.", 2011.
12. THUNG, Phek Lan; NG, Chu Jian; THUNG, Swee Jing; SULAIMAN, Shahida. Improving a web application using design patterns: A case study. In: *2010 International Symposium on Information Technology* [online]. IEEE, 2010, sv. 1, s. 1–6 [cit. 2023-04-11]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5561301.
13. VOLF, Jiří. Využití návrhovVých vzorů přt tvorbě softwaru [online]. 2008 [cit. 2023-04-11]. Dostupné z: https://dspace.vutbr.cz/bitstream/handle/11012/52671/final-thesis.pdf?sequence=-1.
14. LOU, Tian et al. A comparison of Android native app architecture MVC, MVP and MVVM. *Eindhoven University of Technology* [online]. 2016 [cit. 2023-04-11]. Dostupné z: https://pure.tue.nl/ws/portalfiles/portal/48628529/Lou_2016.pdf.
15. AVGERIOU, Paris; ZDUN, Uwe. Architectural patterns revisited-a pattern language [online]. 2005, s. 1–3 [cit. 2023-04-11]. Dostupné z: http://eprints.cs.univie.ac.at/2698/1/ArchPatterns.pdf.
16. LEFF, Avraham; RAYFIELD, James T. Web-application development using the model/view/controller design pattern. In: *Proceedings fifth ieee international enterprise distributed object computing conference* [online]. IEEE, 2001, s. 118–127 [cit. 2023-04-11]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=950428.
17. SELFA, Diana M; CARRILLO, Maya; BOONE, M Del Rocio. A database and web application based on MVC architecture. In: *16th International Conference on Electronics, Communications and Computers (CONIELECOMP'06)* [online]. IEEE, 2006, s. 48–48 [cit. 2023-04-11]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1604744.

18. FÉSÜS, Miklós István. Using Silverlight and XAML in MVC, MVP, MVVM patterns [online]. 2009 [cit. 2023-04-11]. Dostupné z: https://dea.lib.unideb.hu/server/api/core/bitstreams/4a7de64d-0323-42de-b7f1-7b33143d2f3b/content.

19. MUZIKÁŘ, Václav. Návrh a implementace „lehkého“ CMS [online]. [B.r.], s. 11–12 [cit. 2023-04-11]. Dostupné z: https://is.muni.cz/th/hb85d/BP_Archive.pdf.

20. GROSSMAN, John. *Introduction to Model/View/ViewModel pattern for building WPF apps* [online]. 2005. [cit. 2023-04-11]. Dostupné z: https://learn.microsoft.com/cs-cz/archive/blogs/johngossman/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps.

21. LI, XiaoLong; CHANG, DaLiang; PEN, Hui; ZHANG, XiaoYu; LIU, YuanXin; YAO, YaXian. Application of MVVM design pattern in MES. In: *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)* [online]. IEEE, 2015, s. 1374–1378 [cit. 2023-04-11]. Dostupné z: https://ieeexplore.ieee.org/abstract/document/7288144.

22. ABDULLAH, Hanin M; ZEKI, Ahmed M. Frontend and backend web technologies in social networking sites: Facebook as an example. In: *2014 3rd international conference on advanced computer science applications and technologies* [online]. IEEE, 2014, s. 85–89 [cit. 2023-04-11]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7076874.

23. PASTORINO, Marcelo. *Front end vs. back end: What's the difference?* [online]. 2022. [cit. 2023-04-11]. Dostupné z: https://www.pluralsight.com/blog/software-development/front-end-vs-back-end.

24. LION, David; CHIU, Adrian; STUMM, Michael; YUAN, Ding. Investigating Managed Language Runtime Performance: Why JavaScript and Python are 8x and 29x slower than C++, yet Java and Go can be Faster? In: *2022 USENIX Annual Technical Conference (USENIX ATC 22)* [online]. 2022, s. 835–852 [cit. 2023-04-11]. Dostupné z: https://www.usenix.org/system/files/atc22-lion.pdf.

25. ROUSE, Margaret. *Python* [online]. 2022. [cit. 2023-04-11]. Dostupné z: https://www.techopedia.com/definition/3533/python.

26. *What is python? executive summary* [online]. [B.r.]. [cit. 2023-04-11]. Dostupné z: https://www.python.org/doc/essays/blurb/.

27. *Django Introduction - Learn Web Development: MDN* [online]. 2023. [cit. 2023-04-11]. Dostupné z: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction.

28. *Django overview* [online]. [B.r.]. [cit. 2023-04-11]. Dostupné z: https://www.djangoproject.com/start/overview/.

29. GHIMIRE, Devndra. Comparative study on Python web frameworks: Flask and Django [online]. 2020 [cit. 2023-04-11]. Dostupné z: https://www.theseus.fi/bitstream/handle/10024/339796/Ghimire_Devndra.pdf?sequence=2.

30. SIVA, Prasad Reddy K. *Beginning Spring Boot 2: Applications and microservices with the Spring Framework*. Introduction. Apress, 2018.

31. CHHETRI, Nimesh. A comparative analysis of node. js (server-side javascript) [online]. 2016, s. 1–18 [cit. 2023-04-11]. Dostupné z: https://repository.stcloudstate.edu/cgi/viewcontent.cgi?article=1004&context=csit_etds.

32. MARDANOV, Azat. What is Express.js? In: *Express.js Guide: The comprehensive book on express.js*. Lean Publishing, 2013, s. 2.

33. TODOROVA, Magdalina; NISHEVA-PAVLOVA, Maria; PENCHEV, Georgi; TRIFONOV, Trifon; ARMYANOV, Petar; SEMERDZHIEV, Atanas. The Go Programming Language: Characteristics and Capabilities. *Annual of "Informatics" Section Union of Scientists in Bulgaria* [online]. 2013, roč. 6, s. 76–85 [cit. 2023-04-11]. Dostupné z: http://old.usb-bg.org/Bg/Annual_Informatics/2013/SUB-Informatics-2013-6-076-085.pdf.

34. JONAH, Victor. *5 top go web frameworks* [online]. 2022. [cit. 2023-04-11]. Dostupné z: https://blog.logrocket.com/5-top-go-web-frameworks/.

35. ONGO, Gregorius; KUSUMA, Gede Putra. Hybrid database system of MySQL and MongoDB in web application development. In: *2018 International Conference on Information Management and Technology (ICIMTech)* [online]. IEEE, 2018, s. 256–260 [cit. 2023-04-11]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8528120.

36. XING, YongKang; HUANG, JiaPeng; LAI, YongYao. Research and analysis of the front-end frameworks and libraries in e-business development. In: *Proceedings of the 2019 11th International Conference on Computer and Automation Engineering* [online]. 2019, s. 68–72 [cit. 2023-04-11]. Dostupné z: https://dl.acm.org/doi/pdf/10.1145/3313991.3314021.

37. BIERMAN, Gavin; ABADI, Martín; TORGERSEN, Mads. Understanding typescript [online]. 2014, s. 257–281 [cit. 2023-04-11]. Dostupné z: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1469b0cbb109c2a788a346dd0480070de8334dea.

38. ISKANDAR, Taufan Fadhilah; LUBIS, Muharman; KUSUMASARI, Tien Fabrianti; LUBIS, Arif Ridho. Comparison between client-side and server-side rendering in the web development [online]. 2020, s. 1–5 [cit. 2023-04-11]. Dostupné z: https://iopscience.iop.org/article/10.1088/1757-899X/801/1/012136/pdf.

Obsah přiloženého archivu

```
README.md ..... stručný popis obsahu přiloženého archivu
src
    backend ..... zdrojové kódy implementace backendu
    frontend ..... zdrojové kódy implementace frontendu
    thesis ..... zdrojová forma práce ve formátu LaTeX
text
    pecenja2-thesis.pdf ..... text bakalářské práce ve formátu PDF
```