



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**

FACULTY OF INFORMATION TECHNOLOGY

**ÚSTAV INFORMAČNÍCH SYSTÉMŮ**

DEPARTMENT OF INFORMATION SYSTEMS

**INFORMAČNÍ SYSTÉM VÝZKUMNÉ ORGANIZACE**

RESEARCH ORGANIZATION INFORMATION SYSTEM

**BAKALÁŘSKÁ PRÁCE**

BACHELOR'S THESIS

**AUTOR PRÁCE**

AUTHOR

**RADEK VEVERKA**

**VEDOUcí PRÁCE**

SUPERVISOR

**doc. Ing. RADEK BURGET, Ph.D.**

BRNO 2022

## Zadání bakalářské práce



Student: **Veverka Radek**  
Program: Informační technologie  
Název: **Informační systém výzkumné organizace**  
**Research Organization Information System**  
Kategorie: Informační systémy

### Zadání:

1. Analyzujte požadavky zadavatele na informační systém pro řízení výzkumných projektů v rámci organizace CEITEC.
2. Prostudujte současné technologie pro tvorbu webových informačních systémů.
3. Navrhněte architekturu informačního systému na základě provedené analýzy. Uvažujte zejména evidenci projektů, podporu workflow projektů, evidenci žádostí a další.
4. Po dohodě s vedoucím implementujte navržený systém pomocí vhodných technologií.
5. Proveďte testování vytvořeného systému na různých skupinách uživatelů.
6. Zhodnoťte dosažené výsledky.

### Literatura:

- Swicegood, T.: Programming Node.js, O'Reilly, 2012
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015

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

Vedoucí práce: **Burget Radek, doc. Ing., Ph.D.**

Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.

Datum zadání: 1. listopadu 2021

Datum odevzdání: 11. května 2022

Datum schválení: 11. října 2021

## Abstrakt

Pracovní témata práce je vývoj webového informačního systému k administrativnímu řízení výzkumných projektů v rámci infrastruktury CIISB. Mezi nejdůležitější funkce systému patří evidence žádostí o projekty, jejich ověřování, plná podpora emailové komunikace, řízení stavů projektů a správa dokumentů. Maximální automatizace administrativních procesů a neomezená rozšiřitelnost jsou stěžejními vlastnostmi systému. V implementaci převažuje programovací jazyk C# a související technologie s otevřeným kódem od společnosti Microsoft. Text práce provází čtenář ve všech fázích základního vývojového cyklu softwaru vyjma nasazení a údržby.

## Abstract

This work aims to develop a web information system capable of handling administrative processes within research projects of the CIISB infrastructure. Base functionality of the system consists of project registration, reviews, full support of the communication via email, controlling projects' statuses and managing documents. Maximal automation of administrative processes and unlimited extensibility are key aspects of the system. The implementation is lead by C# programming language and related open source technologies from Microsoft. The text of the thesis accompanies the reader through all phases of the basic software development cycle except deployment and maintenance.

## Klíčová slova

informační systém, webová aplikace, výzkumná organizace, administrativa, projekt, email, dokument, evidence, Blazor, SPA, CMS, MariaDB, C#, .NET 6, ASP.NET Core, CEITEC, CIISB

## Keywords

information system, web application, research institution, administration, project, email, document, register, Blazor, SPA, CMS, MariaDB, C#, .NET 6, ASP.NET Core, CEITEC, CIISB

## Citace

VEVERKA, Radek. *Informační systém výzkumné organizace*. Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Radek Burget, Ph.D.

# Informační systém výzkumné organizace

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana docenta Radka Burgeta. Další informace mi poskytli zaměstnanci instituce CEITEC Jiří Novák a Kateřina Hošková. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....  
Radek Veverka

8. května 2022

## Poděkování

Tímto děkuji Jiřímu Novákovi, Kateřině Hoškové a Michalu Babiakovi, kteří mně umožnili dále rozvíjet a v praxi realizovat vývojářské dovednosti v oblasti webových informačních systémů. Mé díky samozřejmě patří i docentu Burgetovi za formální vedení práce, ochotu probírat dotazy a poskytnutí zpětné vazby.

# Obsah

<b>1 Úvod</b>	<b>3</b>
<b>2 Analýza a specifikace požadavků</b>	<b>4</b>
2.1 Prototypování . . . . .	4
2.2 Funkční požadavky . . . . .	4
2.2.1 CIISB . . . . .	4
2.2.2 Diagram p ípad užití . . . . .	6
2.2.3 Rozbor p ípad užití z pohledu aktér . . . . .	6
2.3 Nefunkční požadavky . . . . .	8
2.4 P íhlašování a správa identit . . . . .	9
<b>3 Technologie pro tvorbu informačních systému</b>	<b>11</b>
3.1 Sou asné architektury informa ních systém . . . . .	11
3.1.1 Monolitická t ívrstvá architektura . . . . .	12
3.1.2 Mikroslužby . . . . .	12
3.2 Typy moderních web . . . . .	12
3.2.1 Tradi ní web (tenký klient) . . . . .	13
3.2.2 SPA (tlustý klient) . . . . .	13
3.3 Komunika ní protokoly . . . . .	13
3.4 Serializa ní formáty . . . . .	13
3.5 Aplika ní rozhraní . . . . .	14
3.6 Autentiza ní mechanismy . . . . .	15
3.6.1 Problematika uložení identity . . . . .	15
3.7 Databáze . . . . .	16
3.8 Programovací jazyky a aplika ní rámce . . . . .	16
3.8.1 PHP . . . . .	17
3.8.2 JavaScript a Node.js . . . . .	17
3.8.3 C# a .NET . . . . .	18
3.8.4 Princip techniky DI (dependency injection) . . . . .	19
3.9 Výb r jazyka, frameworku a dodate ných technologií . . . . .	20
<b>4 Architektonický návrh systému</b>	<b>23</b>
4.1 Identifikace koncept ů a entit systému . . . . .	23
4.1.1 Organizace . . . . .	23
4.1.2 Dokumenty a jejich evidence . . . . .	23
4.1.3 Projekt . . . . .	27
4.1.4 Workflow . . . . .	27
4.1.5 Emaily . . . . .	28

4.1.6	Uživatelé, role a oprávnění . . . . .	28
4.2	Uživatelské rozhraní (frontend) . . . . .	29
4.2.1	Pohled pro zaměstnance laboratoří . . . . .	29
4.2.2	Pohled pro administrátory projektů . . . . .	29
4.2.3	Pohled emailového klienta . . . . .	30
4.2.4	Pohledy pro ostatní uživatele . . . . .	30
4.2.5	Pohled pro obecnou správu obsahu . . . . .	30
4.2.6	Pohled pro správu uživatelů . . . . .	30
<b>5</b>	<b>Implementace</b>	<b>31</b>
5.1	Zdrojový kód, projekty, řešení a závislosti . . . . .	31
5.2	Konfigurace systému . . . . .	32
5.3	Unikátní identifikátory . . . . .	33
5.4	Automatizace formulářů . . . . .	33
5.4.1	Reflexe . . . . .	33
5.4.2	Atributy . . . . .	34
5.5	Autentizace . . . . .	35
5.5.1	Vlastní SAML 2 handler . . . . .	36
5.6	Autorizace . . . . .	36
5.7	Dokumenty a správa obsahu . . . . .	37
5.7.1	Dokumenty HTML jako šablony . . . . .	37
5.7.2	Dokumenty DOCX jako šablony . . . . .	37
5.7.3	Generování dokumentů PDF . . . . .	38
5.7.4	Odkazy v dokumentech . . . . .	38
5.8	Řazení akcí . . . . .	39
<b>6</b>	<b>Testování</b>	<b>40</b>
6.1	Průběžné testování . . . . .	40
6.2	Poznámky různých skupin uživatelů . . . . .	41
6.3	Migrace systémových dat . . . . .	41
<b>7</b>	<b>Závěr</b>	<b>43</b>
	<b>Literatura</b>	<b>44</b>
<b>A</b>	<b>Obsah příloženého paměťového média</b>	<b>46</b>

# Kapitola 1

## Úvod

Tento text pojednává o analýze, návrhu, implementaci a testování webového informačního systému pro správu výzkumných projektů institucí CEITEC a BIOCEV. Výzkumné instituce zahrnují několik laboratoří, zvaných *core facility*, které se v nich využívají v různých oborech biologických věd. Každá laboratoř má své zázemí, jak provádět měření, a své přístroje. Laboratoře zprostředkovávají měření, výzkum a odbornou konzultaci k projektům zákazníků. Cílem této práce není vyvinout informační systém pro evidenci a zpracování dat experimentů v jednotlivých laboratořích, nýbrž systém, jenž bude operovat spíše na administrativní úrovni. To znamená, že primárním úkolem systému bude sběr žádostí o projekty, monitorování stavu projektů, evidence souvisejících dokumentů a žádostí a komunikace mezi zákazníky a zaměstnanci.

Tato práce bude implementovat systém konkrétně pro projekty spadající pod infrastrukturu *CIISB* (Czech Infrastructure for Integrative Structural Biology), která zajišťuje kooperaci jednotlivých laboratoří. Sdílení projektů mezi laboratořemi je velmi důležité, nebo jejich technologie se vzájemně doplňují a pro dosažení dobrých výsledků je třeba jejich kombinace. Systém bude navržen tak, aby jej bylo možné snadno rozšiřovat o libovolnou logiku a implementovat i jiné infrastruktury než CIISB.

Následující kapitola se bude věnovat analýze a specifikaci požadavků na implementaci systému pro infrastrukturu CIISB. Čtenář bude dále seznámen se současnými (2022) technologiemi pro vývoj webových informačních systémů. Na základě analýzy požadavků proběhne architektonický návrh systému s důrazem na jeho rozšiřitelnost a flexibilitu obecných prvků informačního systému tohoto typu. Se zvolenou technologií bude systém následně implementován dle návrhu a testován.

## Kapitola 2

# Analýza a specifikace požadavků

Analýza a specifikace požadavků je první etapou vývojového cyklu softwaru. Jedním z kroků této etapy je identifikace *potřeb* užítelů. Požadavky se dělí na *funkční* a *nefunkční*, oba druhy budou zanalyzovány v následujících sekcích. Dobrou specifikaci požadavků lze získat technikou *prototypování*.

### 2.1 Prototypování

Prototypování je technika získávání požadavků, při které je vyvíjen systém (prototyp) a testován zákazníkem. Prototyp je zpravidla implementován rychle, má za cíl demonstrovat potenciální chování systému a při jeho tvorbě se tolik nehledí na kvalitu návrhu a kódu [11]. Zákazník je při používání prototypu schopen lépe specifikovat požadavky. Prototyp také může pomoci odhalit úskalí architektonického návrhu nebo použitých technologií.

Funkční a dokonce do produkce nasazený prototyp pro tuto práci byl implementován v programovacím jazyce PHP bez využití aplikačního rámce. Ačkoliv v jednodušší podobě využíval architektonický vzor MVC, změny se a přibývajících požadavků negativně zasáhly do architektury systému. Prototyp zadal být poměrně nepřijemný na údržbu a rozšiřování, a proto tato práce pokračuje kompletní reimplementací projektu s využitím vhodných moderních technologií a zpracováním jeho návrhu. Většina požadavků rozebraných v dalších sekcích byla odhalena a specifikována právě díky prototypu.

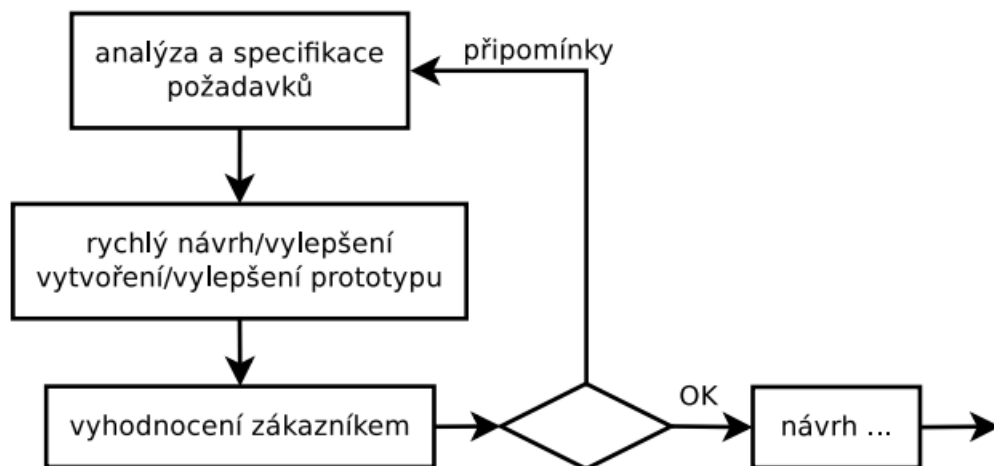
### 2.2 Funkční požadavky

Funkční požadavky formulují, co by měl systém dělat [5]. Podstatou informačního systému je vymodelování nějaké reality. Poznání této reality je pochopitelně nutné k identifikaci požadavků na systém, který realitu modeluje. Proto následující podsekcce seznámí čtenáře blíže s infrastrukturou CIISB.

#### 2.2.1 CIISB

Cílem organizace CIISB je zprostředkovat výzkumné technologie pro tuzemské i zahraniční uživatele z akademického i průmyslového odvětví [15]. CIISB zahrnuje dvě výzkumná centra pro poskytování služeb, a to CEITEC v Brně a BIOCEV v Praze. Obě centra se dále dělí na další organizační jednotky - již zmíněné laboratorie. Každá laboratorie se zaměřuje svými technologiemi na jiné odvětví výzkumu. CIISB tedy poskytuje služby tohoto aparátu





Obrázek 2.1: Vývojový cyklus softwaru s využitím prototypování. Pevzato z [11]

zákazník m, k tomu navíc definuje procesy a formální náležitosti, které musí být dodrženy p i pln ní zakázek. Ú el systému je práv maximální automatizace t chto proces .

*Projekt* je vedle organizace základní logickou jednotkou celého systému, vymezuje n jaký celistvý výzkumný proces. V CIISB o projekty žádají zákazníci. Žádost zákazníka o projekt zahrnuje laborato e a jejich služby, které je v rámci projektu pot eba využít. Žádost m že podat úpln kdokoliv, proto prochází schvalovacím procesem. K žádosti se vyjád í všichni vedoucí laborato í, se kterými projekt souvisí, a bu to ji schválí, nebo zamítnou. Provád jí tím vyhodnocení *technické zp sobilosti projektu* (technical feasibility). Je-li projekt schválen všemi laborato emi a žadatel je externí osoba, tedy využívající služby za jinou afilaci než CEITEC, postupuje do dalšího kola schvalování, kde žádost reviduje práv jeden externí odborník, zvaný také *peer reviewer*. Každá laborato má seznam odborník , se kterými spolupracuje a m že kohokoliv z nich požádat o revizi žádosti. Jakmile je žádost schválena ve všech krocích, p istoupí se k provád ní výzkumu. Uživatel m jsou nabídnuty ceníky služeb a externí žadatelé navíc musí potvrdit dokument s podmínkami užívání služeb. Procesy výzkumu se liší v závislosti na laborato i a nejsou p edm tem tohoto informa ního systému. Jakmile je výzkum dokon en, zam stnec laborato e projekt ukon í. Po ukon ení projektu se o ekává od zákazníka publikace, v níž prezentuje dosažené výsledky. Zam stnanec m že rovn ž úpln uzav ít projekt bez výsledk .

Je t eba vzít v potaz, že projekt m že být dlouhodobá záležitost. Práce na projektu m že trvat dny, týdny, m síce i roky. Je proto t eba ve všech fázích projektu pravideln vhodným zp sobem upozor ovat všechny zainteresované osoby, aby se na n který projekt nezapomn lo nebo p íliš dlouho nedlel v jedné fázi. Z d vodu ú tování za služby (není sou ástí systému) a faktu, že se zákazník m že p estat chtít projektu dále v novat, je nutné nastavit projekt m ur itou expira ní lh tu, po které budou automaticky ukon eny. Tato doba je zhruba rok a p l. Pokud je projekt dlouhodobý, zákazník bude p ed expirací upozorn n a bude moci zažádat o jeho prodloužení. Po schválení žádosti bude automaticky vytvo en nový projekt navazující na ten p edchozí.

## 2.2.2 Diagram případů užití

Případ užití je nějaká činnost, kterou může se systémem provádět *aktér*. Aktér je tedy osoba či neživá entita (například osoba), která iniciuje jednotlivé případy užití. Dobře identifikované případy užití by neměly zachycovat pouze strohé akce, které je možné v systému provádět, ale především jejich účel. Například místo případu užití „odeslat email“ dává v téže smysl formulace „upozornit uživatele na zmenu“, nebo cílem opravdu není odeslání emailu, ale upozornění uživatele, což se dá implementovat různými způsoby. Stejně tak „pohlásit uživatele“ není případ užití. Z vhodně stanovených případů užití vyplyne nejen co všechno systém musí umět, ale také co umět nemusí a bylo by zbytečné implementovat. Model případů užití a jejich aktérů se znázorňuje *diagramem případů užití*.

Z výše uvedeného popisu CIISB už je možné sestavit diagram případů užití pro systém. Diagram je znázorněn na obrázku 4.3, jednotlivé případy užití jsou podrobněji analyzovány dále v textu. Diagram a další specifikace vycházejí kromě popisu výše také z *prototypu* systému a detailních požadavků zadavatele.

## 2.2.3 Rozbor případů užití z pohledu aktérů

Cílem této podsekcce je dále zanalyzovat model případů užití, a stanovit tak podrobnější požadavky na systém.

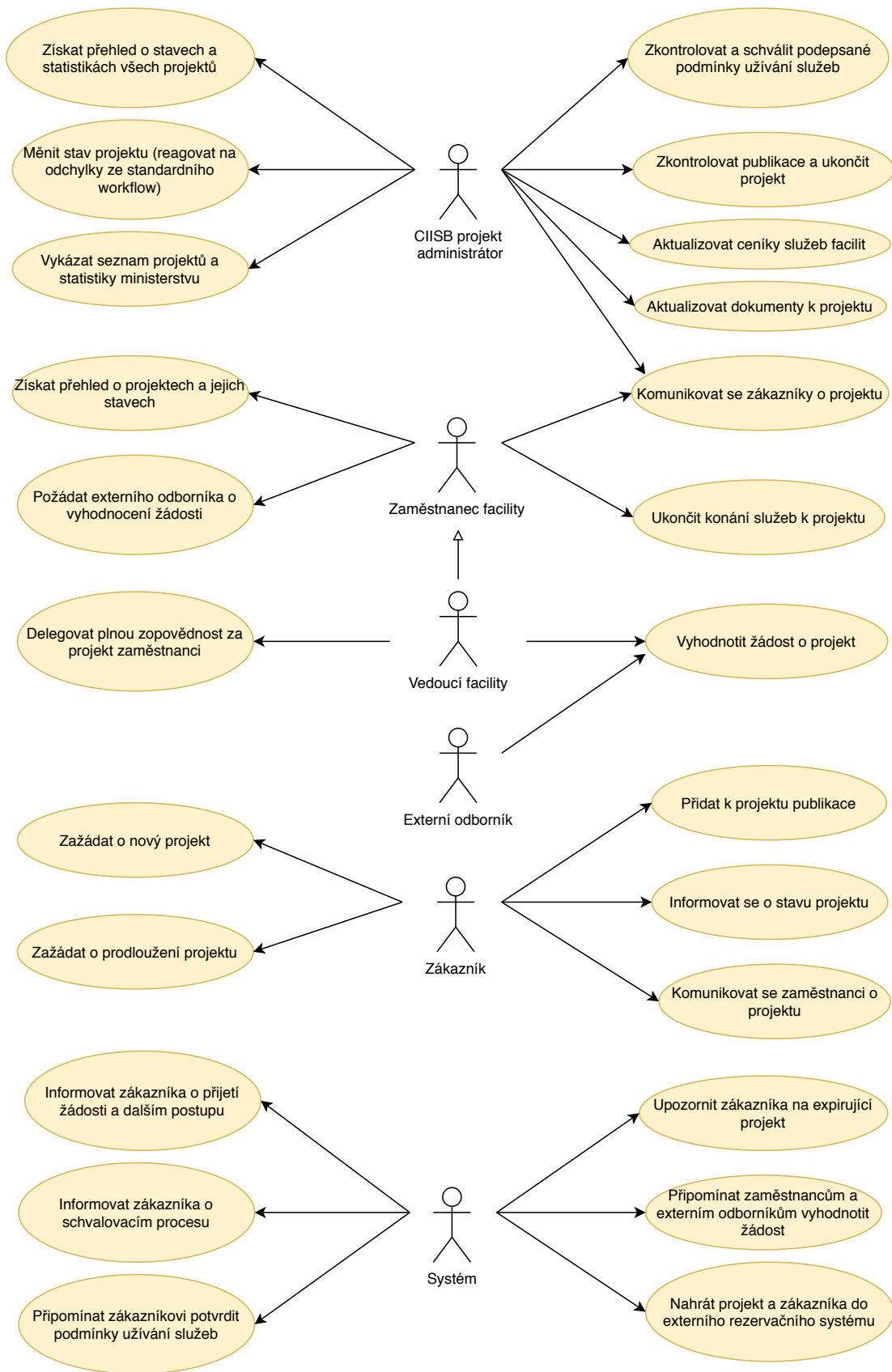
### Zákazník

Zákazníkem jsou všechny osoby na straně žadatele projektu. Obecně se v projektu může angažovat více typů osob:

- **Žadatel (applicant)** je osoba provádějící výzkum. Může být v roli studenta nebo samostatného výzkumníka.
- **Vedoucí výzkumu (principal investigator)** je osoba zodpovědná za výzkum, ať už například podepisování podmínek, platí za služby. Typicky, je-li žadatelem student, má nějakého vedoucího práce. Žadatel a vedoucí výzkumu však může být jedna a tatáž osoba.
- **Další zainteresované osoby.** Bude jim chodit korespondence ohledně vývoje projektu.

Zákazník podává žádost v podobě vyplněného formuláře. Bez systému je toto řešeno dokumentem .docx, který žadatel vyplňuje a následně odesílá administrátorovi CIISB, jenž žádost dále předeposílá souvisejícím laboratorům. Tento dokument je nepřehledný a praxe ukázala, že je špatně vyplněvaný a často v něm chybí podstatné informace. Dokument je rozsáhlý a každá laborator poskytuje svůj podformulář. Formulář tedy bude třeba převést do webové podoby s výsvětlivkami a validacemi. Na tento formulář bude odkázáno z webu CIISB. Před vstupem k formuláři a založením žádosti proběhne ověření uživatelské identity nebo emailu.

Zákazníci se nemusí přihlašovat do systému – museli by se registrovat, pamatovat si přihlašovací údaje a seznamovat se se systémem, což je bezpochyby na obtíž. Případy užití zákazníka lze realizovat pomocí emailu, který všichni znají, mají, umí používat a hlavně pravidelně kontrolují.



Obrázek 2.2: Diagram p ípad užití pro infrastrukturu CIISB

Důležitým požadavkem je tedy podpora emailových služeb v plném rozsahu. Systém musí umět odesílat, přijímat i posílat emaily (i s přílohami), tedy dle potřeb jednotlivým projektům, ale také zpracovávat, uchovávat a reagovat na jejich obsah. Všechny pokyny budou zákazníkovi automaticky posílány emailem. Všechny jeho povinnosti v rámci CIISB mu budou emailem pravidelně připomínány. Zprávy také mohou obsahovat unikátní URL pro konání jednotlivých akcí v systému zákazníkem, například vyplnění formuláře žádosti o prodloužení projektu.

### **Externí odborník**

Podobně jako zákazník, ani externí odborník nemá účet v systému, emailem jsou mu zasílány odkazy na formuláře k vyhodnocení projektu. Obsah žádosti je přiložen.

### **Vedoucí a zaměstnanec laboratoře**

Tyto osoby mají účet v systému, nebo sledují projekty své laboratoře a mají jejich stav. Zaměstnanec může pouze sledovat projekty, nicméně může být vedoucím pracoviště povinen zodpovědností nad konkrétními projekty – v takovém případě získává všechna práva vedoucího v rámci tohoto projektu.

Zaměstnanci a zákazníci spolu o projektu komunikují prostřednictvím emailu. Systém poskytne pro tuto komunikaci speciální emailovou adresu. Identifikace projektu a organizace se bude vyskytovat v potřebě každé zprávy, aby mohl systém email správně zařadit a zpracovat.

### **Administrátor CIISB**

Tento uživatel musí být schopen držet kontrolu nad všemi projekty systému, důležitou funkcí je možnost filtrovat projekty dle různých kritérií (datum, stav, laboratoř, výsledek). Občas se stane, že se projekt vychýlí ze standardního workflow, například uživatel dodá podepsané podmínky osobně a nepošle jejich sken emailem. V takovém případě by měl administrátor možnost zasáhnout – aktualizovat dokumenty nebo posunout stav projektu.

### **Systém**

Tento aktér má na starosti generování spousty pravidelných a informačních emailů. Denně se bude automaticky spouštět rutina, která bude kontrolovat časová razítka u projektů a generovat pravidelně. Musí se také umět připojit k systému *Microsoft CRM* a vložit tam nový projekt i zákazníka, aby je personál nemusel ze žádosti popisovat ručně. Zaměstnanci a zákazníci laboratoří pak používají CRM systém pro rezervaci služeb a zařazení pro práci na projektech.

Důležitou funkcionalitou systému je také schopnost z dokumentů v systému (žádost, vyhodnocení) generovat dokumenty PDF, tyto dokumenty umožnit stahovat a přidávat jako přílohy automatickým emailům.

## **2.3 Nefunkční požadavky**

Tyto požadavky definují vlastnosti systému, nikoliv jeho konkrétní chování [5].

- **Bezpečnost**  
Bude použit protokol HTTPS, certifikát dodá Masarykova univerzita společně s organizací CESNET. Postará se o základní bezpečnostní zásady (server-side validace, ochrana před SQL injekcí, neukládání hesel v otevřené podobě ...), protože systém nepracuje s financemi a jeho obsah ve finále kontroluje kompetentní osoba. Hlavním úskalím je emailová komunikace, která obecně není zabezpečena.
- **Provoz systému**  
Systém bude běžet na jednom virtuálním linuxovém serveru, poskytnutým IT oddělením Masarykovy univerzity.
- **SEO - optimalizace pro vyhledávače**  
Není nutné, aby nějaká část webu byla vůbec byla zaindexována ve vyhledávačích. Není cílem, aby o systému vědělo co nejvíce lidí na internetu. Vstupem uživatele do systému je v podstatě formulář žádosti, na který je odkázáno z webu CIISB.
- **Výkon**  
Systém bude používat celkem pár desítek uživatelů. To zvládne bez potíží jeden server bez větších optimalizací aplikace i bez vyvažování zátěže (load balancingu).
- **Interoperabilita**  
Nejde o izolovanou aplikaci, je nutná komunikace s dalšími systémy – Microsoft CRM nebo IMAP server.
- **Rozšiřitelnost a znovupoužitelnost**  
Systém musí být po všech stránkách rozšiřitelný, aby bylo možné reagovat na případné změny požadavků. Musí být možné přidávat nové projekty, formuláře, žádosti, organizace, emailové šablony a obecné dokumenty a všem těmto prvkům implementovat libovolnou logiku. Právě tento požadavek je velmi důležitý v potaz s návrhem architektury systému.
- **Podpora prohlížečů**  
Dostupná je podpora aktuálních verzí nejpopulárnějších prohlížečů s výjimkou MS Internet Explorer.
- **Optimalizace pro mobilních zařízení**  
Není nutné, u zaměstnanců se předpokládá použití počítače na pracovišti. Stejně tak se nepředpokládá, že by zákazníci vyplňovali rozsáhlý formulář na telefonu.
- **Globalizace a lokalizace**  
v oblasti vývoje a výkumu je nezvyklé, že by nějaký uživatel neovládal angličtinu, navíc zde figuruje spousta zahraničních uživatelů. Angličtina tedy bude primárním a jediným jazykem systému. Datum a čas však budou zobrazovány dle českých zvyklostí, ve 24-hodinovém formátu. Například celým systémem bude výhradně použito kódování textu UTF-8.

## 2.4 Přihlašování a správa identit

Z analýzy vychází, že není nutné autentizovat zákazníky, jejich interakce se systémem bude ešena výhradně emailem s unikátními odkazy. Přihlašovat je třeba pouze zaměstnance,

a protože všichni zaměstnanci mají identitu u své instituce, je vhodné namísto implementace vlastní registrace a přihlašování využít již existujících služeb. Česká akademická federace identit eduID.cz [8], provozována sdružením CESNET, poskytuje jednotnou bránu k identitám, které spravují různé instituce v České republice. Služby této federace budou využity k autentizaci zaměstnanců, díky čemuž nebude v systému nutné uchovávat jejich přihlašovací údaje. Údaje zaměstnanců související se systémem, jako třeba jejich role, však budou v systému uchovány. Z nabízených institucí budou využity:

- **Masarykova univerzita** pro zaměstnance laboratoří CEITECu v Brně.
- **Biotechnologický ústav AV ČR, v.v.i.** pro zaměstnance laboratoří BIOCEVu v Praze.

Podrobnější rozbor fungování federace eduID.cz je k dispozici v sekci [5.5](#).

## Kapitola 3

# Technologie pro tvorbu informačních systémů

Tato kapitola seznamuje čtenáře s moderními (2022) technologiemi pro tvorbu webových informačních systémů a srovnává je. Na základě této a předchozí kapitoly budou také vybrány vhodné technologie, na jejichž bázi bude dále navržena architektura systému a provedena implementace.

Technologie není jednoduché zvolit, protože s popularitou webových aplikací roste pochopitelně počet nástrojů k jejich tvorbě. Technologie nutné pro implementaci a používání plnohodnotného webového informačního systému zahrnují:

- Webový server.
- Programovací jazyk na serveru.
- Aplikační rámec na serveru.
- Databázi.
- Webového klienta (prohlížeč).
- Programovací jazyk na klientovi.
- Aplikační rámec na klientovi.
- Komunikační protokol mezi klientem a serverem.
- Serializační formáty.

Toto je pouze hrubé rozdělení, jednotlivé technologie se mohou vzájemně prolínat.

### 3.1 Současné architektury informačních systémů

Architektura systému popisuje systém na nejvyšší úrovni – jaké jsou jeho prvky a jak jsou vzájemně propojeny. V souvislosti má smysl uvažovat dvě architektury:

### 3.1.1 Monolitická třívrstvá architektura

Je nejrozšířenější standardní přístup k tvorbě systému. Hlavní myšlenkou je rozdělení systému do tří vrstev:

- *Prezentační vrstva* zobrazuje výstupy uživateli a zpracovává vstupy uživatele. Komunikuje pouze s aplikační vrstvou.
- *Aplikační vrstva* je jádrem systému, obsahuje business logiku. Zpracovává vstupy z prezentační vrstvy a vrací jí výstupy.
- *Datová vrstva* definuje podobu dat systému a poskytuje persistentní úložiště aplikační vrstvy.

Architektura je monolitická, protože vrstvy jsou zpočátku úzce propojené a systém je vyvíjen a nasazován jako celek. Výhodou je právě jednodušší nasazení a testování systému ve srovnání s *mikroslužbami*. Problémem je pomalejší vývoj nových verzí, protože jedna změna požadavku často implikuje nutnost zasáhnout do všech tří vrstev [7].

### 3.1.2 Mikroslužby

Mikroslužby jsou jednotky, které v systému fungují samostatně a jsou zodpovědné za jednu úlohu. Každá mikroslužba se dá představit jako malý podsystém založený na třívrstvé architektuře, prezentační vrstva zde vystupuje ve funkci aplikačního rozhraní pro ostatní mikroslužby. Výhodou je, že na každé mikroslužbě může pracovat tým vývojářů odděleně a třeba i s použitím jiných technologií, za předpokladu že bude dodrženo stanovené aplikační rozhraní. Systém se rychleji vyvíjí, lépe škáluje, ale hůře testuje a nasazuje jako celek [7]. Pro tuto práci nejsou mikroslužby vhodné, protože je systém vyvíjen jednou osobou, nasazován na jeden stroj a používán pár desítkami uživatelů. Nevýhody režie a složitosti spojené s architekturou mikroslužeb převyšují výhody flexibility a škálovatelnosti.

## 3.2 Typy moderních webů

Před tvorbou webového informačního systému je třeba stanovit jeho rozdělení na klientskou a serverovou část. Toto rozdělení určuje *tloušť ku klientu*, která říká, do jaké míry je aplikační vrstva systému v kompetenci klientského kódu.

Rozdělení zodpovědnosti však není přímočaré a jednoduché. Je třeba myslet na skutečnost, že uživatel má přístup ke zdrojovému kódu buď přímo v prohlížeči, nebo může modifikovat webovou stránku a posílat nevhodná data. Systém musí zajistit, aby tímto způsobem nemohla být narušena business logika a konzistence systému. Proto také nelze přesunout kompletně celou aplikační vrstvu na stranu klienta.

**HTML, CSS a JavaScript** je trojice jazyků sloužících k prezentaci dat uživateli ve webovém prohlížeči. HTML je značkový jazyk, který popisuje strukturu zobrazovaných dokumentů pomocí *stromu uzlů*, známému jako DOM (Document Object Model). Jazyk CSS dodává vzhled jednotlivým uzlům ve stromu. Jazyk JavaScript modifikuje strukturu DOM a doplňuje web o interaktivní prvky. Bez využití JavaScriptu je interakce webové stránky s uživatelem a serverem omezená na *formuláře*. Právě podílem kódu v JavaScriptu je dána *tloušť ku klientu*, tlustý klient totiž obsahuje hodně logiky, jež je třeba implementovat v plnohodnotném programovacím jazyce, kterým HTML a CSS nejsou.



### 3.2.1 Tradiční web (tenký klient)

Tyto weby fungují tak, že klientská část (prohlížeč) pouze vykresluje obsah, který dostane od serveru. Kliknutím na odkaz prohlížeč provede dotaz HTTP GET na nějakou URL, smaže celý vykreslený obsah, načte obsah HTTP odpovědi a dle ní vykreslí kompletnou novou stránku. Přestože je to starý způsob, stále se využívá v systémech prezentujících jednoduchý neinteraktivní obsah, například ve zpravodajských a blogových portálech.

### 3.2.2 SPA (tlustý klient)

Weby SPA (Single page application) neobsahují pouze jednu stránku, jak napovídá název, ale načítají obsah odlišným způsobem. Při přechodu na web je načtena pouze základní kostra HTML dokumentu a zdrojové kódy klientské části (obvykle JavaScript), žádný další pohyb uživatele po webu nezpůsobí kompletní smazání a znovunačtení obsahu stránky. Další komunikace se serverem je zajištěna *asynchronní* technologií AJAX, která umožňuje kódu v JavaScriptu komunikovat se vzdálenými servery.

SPA snižuje množství přenesených dat při delším pobytu na webu, velká část logiky webu je načtena hned na začátku, později už klient stahuje pouze čísta data. Prostedí také působí plynuleji, nepřebíhá a může s pomocí animací ukazovat uživateli, že se část stránky aktualizuje. Na druhou stranu, první přístup na web nebo uživatelský příkaz k novému načtení celého webu mohou způsobit až nějakou inovou odezvu, při které se společně se základní HTML kóstrou musí načíst zdrojový kód pro obsluhu klientské části aplikace s *aplikací rámecem*. Tyto kódy následně musí prohlížeč *interpretovat*. Pro eliminaci tohoto problému se využívá technika *cacheování*, kdy prohlížeč místo opakovaného stahování totožných souborů využívá lokální úložiště, a *lenivého načítání (lazy loading)*, při kterém se načítají pouze ty moduly, které budou až v momentě, kdy uživatel přistoupí ke službě, jež tyto moduly využívá.

## 3.3 Komunikační protokoly

Pokud je o *webovém* informačním systému, předpokládá se užití aplikačního protokolu **HTTP** (Hypertext transfer protocol), protože tento protokol používají webové prohlížeče a pro komunikaci se serverem. Protokol je to bezstavový – HTTP požadavky jsou mezi sebou nezávislé. Komunikaci zahajuje vždy pouze klient (webový prohlížeč) a dostává odpověď na požadavek. Server však může také vystupovat v roli klienta a zahájit komunikaci s aplikačním rozhraním jiného HTTP serveru. Pokud klientská aplikace potřebuje reagovat na změny stavu serveru, je nutné využít techniku *opakovaného dotazování (polling)*, kdy se klient v pravidelných časových intervalech dotazuje serveru na aktuální stav.

**WebSocket** je protokol umožňující otevřít oboustranné spojení mezi prohlížečem a serverem, je alternativou k technice *polling*. Je možné jej využít k transportu libovolných dat oběma směry a odpadá režie komunikace v podobě HTTP hlaviček, proto je také využíván *real-time* webovými aplikacemi, jako jsou například hry s podporou více hráčů.

## 3.4 Serializační formáty

Serializační formáty umožňují reprezentovat strukturovaná data (kolekce a struktury) v textovém, alternativně binárním formátu, přenositelným po síti, například v těle požadavku či odpovědi protokolu HTTP. Tyto formáty jsou nezávislé na programovacím jazyce, ze kterého

strukturovaná data pocházejí. Nejrozšířenější formáty jsou JSON a XML. Za zmínku stojí také uživatelsky přívětivý YAML. Součástí každého programovacího jazyka vhodného pro web jsou nástroje k transformaci dat mezi těmito formáty a interní reprezentací dat jazyka.

**XML** (eXtensible Markup Language) je starší než JSON, ale stále hojně používán. K jazyku XML existuje rodina standardizovaných podpůrných nástrojů pro jeho vizualizaci a transformaci, zvaná XSL (eXtensible Stylesheet Language) [17]. Nechybí ani možnost definovat metadata, která popisují schéma XML dokumentu, a využít je k automatické validaci dokumentu. K definici schématu slouží jazyk XSD [16].

**JSON** (JavaScript Object Notation) je podstatně jednodušší formát než XML. Vychází z jazyka JavaScript, kde je velmi jednoduše použitelný, protože formát odpovídá přesně zápisu objektu v jazyku. K JSONu však neexistují dodatečně standardizované nástroje a formáty jako k jazyku XML.

### 3.5 Aplikační rozhraní

Pomocí aplikačního rozhraní (API) spolu komunikují různé aplikace, čímž se liší od rozhraní uživatelského, přes které vzájemně komunikuje uživatel s aplikací. U webových aplikací je nejpodstatnější komunikace *klient-server*. Aplikační rozhraní obvykle operují nad *komunikačním protokolem*.

**REST (Representational State Transfer) API** [18] je úzce spjata s protokolem HTTP a využívá některé jeho vlastosti. Server nabízí několik koncových bodů (*endpoint*), na které může klient posílat HTTP dotazy různých typů. Typy dotazů obvykle odpovídají akci, kterou má server provést. Mezi typické akce prováděné na entitách v systému patří především CRUD - *Create, Remove, Update a Delete*. Tabulka 3.1 znázorňuje typické použití REST API k manipulaci s konkrétní entitou.

Typ požadavku	Koncový bod	Akce
GET	/api/users	Získat informace o všech uživateli.
GET	/api/users/{id}	Získat informace o uživateli s identifikátorem {id}.
POST	/api/users	Vytvořit nového uživatele.
PATCH/PUT	/api/users/{id}	Upravit údaje o uživateli s identifikátorem {id}.
DELETE	/api/users/{id}	Smazat uživatele s identifikátorem {id}.

Tabulka 3.1: Typický návrh REST API pro operace CRUD nad konkrétní entitou.

Struktura REST API není striktně specifikována a záleží na konkrétní aplikaci a jejím doménovém modelu. Pro popis struktury a schopností rozhraní pro uživatele i další software lze využít specifikaci OpenAPI [3].

**GraphQL** je dotazovací jazyk pro aplikační rozhraní. Problémem REST API je, že jsou klientovi často vrácena nepotřebná data, což znamená zbytečný provoz navíc. Pomocí jazyka GraphQL si klient vybere, která data konkrétně od serveru potřebuje. Server implementující GraphQL také poskytuje metadata o všech datech, která systém skrze GraphQL nabízí. Klient tak má informaci, co v odpovědi očekávat. Server typicky vystaví pouze jeden koncový bod, na který se klient dotazuje. Veškeré další detaily dotazu jsou kódovány do těla požadavku a server je zpracovává jednotným způsobem.

## 3.6 Autentizační mechanismy

*Autentizace uživatele* znamená z pohledu systému získání a ověření uživatelské identity. Znalost identity je poté potřebná k provedení *autorizace*, tedy rozhodnutí o tom, zdali má daný uživatel přístup k vybranému zdroji.

Při **autentizaci jménem a heslem** se uživatel identifikuje jedním nechráněným údajem (email, uživatelské jméno) a jedním chráněným údajem (heslo). Úskalím tohoto typu autentizace je bezpečnost a uchování tajného údaje. Bezpečnost přenosu je dnes téměř výhradně zajištěna protokolem HTTPS, kdy je heslo šifrováno ještě před opuštěním počítače. Způsob uchování hesel je zcela na aplikaci, do které se uživatel přihlašuje. Zpravidla se používají *hashovací funkce* a *kryptografická síla*. Při přihlašovací údaje putují od prohlížeče buď jen jednou, kdy probíhá hned výměna údajů za unikátní *identifikátor sezení* (*session id*), nebo s každým požadavkem (HTTP Basic Auth).

**Autentizace pomocí třetí strany** eliminuje nutnost uchování přihlašovacích údajů v aplikaci a uživatel se nemusí do systému registrovat, vlastní-li již účet na jiném serveru. Obě strany však musí tyto služby podporovat.

**Dvoufázové ověřování** vyžaduje při autentizaci od uživatele dodatečné potvrzení jeho identity. Typicky se využívá mobilní telefon uživatele, buď SMS s kódem, nebo autentizací aplikace. Dvoufázové ověřování je využíváno zejména systémy, které pracují s financemi, což jsou třeba banky, spojitelný nebo burzy. Dvoufázové ověřování lze také aktivovat u některých velkých korporací jako Google nebo Apple, které provozují spoustu služeb pod jedním uživatelským účtem.

### 3.6.1 Problematika uložení identity

*Sezení* (*session*) je časový interval, během kterého je uživatel autentizován a používá aplikaci. Protokol HTTP je však bezstavový a správu sezení neumožňuje, to je třeba vyřešit v aplikaci.

#### Session cookie

Tradičním způsobem zachování sezení je využití *cookies*, které jsou součástí HTTP protokolu. Po přihlášení je uživateli na straně serveru přidělen unikátní náhodný identifikátor. V odpovědi je zaslán prohlížeči pomocí hlavičky Set-Cookie a následně na straně klienta uložen. Tento identifikátor je prohlížečem automaticky připojen do všech následujících HTTP požadavků v dané doméně, což umožňuje požadavek na straně serveru namapovat na konkrétního uživatele.

#### JSON Web token (JWT)

Další variantou správy sezení je sezení na serveru vůbec neukládat a ponechat tuto zodpovědnost na straně klienta. Prohlížeči po autentizaci uživatele uloží kompletní informace, které o uživateli získá, typicky unikátní identifikátor uživatele, jméno, příjmení a role v systému. Tato informace se říká *tvrzení* (*claims*). *Tvrzení* se následně odesílají serveru zakódované do hlavičky každého HTTP požadavku Authentication. Samotná *tvrzení* však nestačí, protože server nemusí vědět, v jakém jsou formátu, a navíc jim nemá důvod věřit. Autentizační služba vydá prohlížeči kromě samotných *tvrzení* také *metadata* a vše podepíše svým soukromým klíčem. JWT je formát, do kterého se zakódují *metadata*, *tvrzení* i *podpis*. Server při přijímání JWT v hlavičce požadavku má pak možnost pomoci ve věcnosti klíče posky-

tovatele ověřit, zda je JWT originál. Pokud je ověření úspěšné a server dále uje poskytovateli *tvrzení*, může uživatele bezpečně autorizovat a pustit ke zdroji.

Do JWT typicky poskytovatel uloží také datum a čas expirace. Po expiraci nelze JWT považovat za platný.

Hlavní výhodou JWT i jiných tokenů je jejich bezstavovost, proto je vhodné je používat také při vzájemné komunikaci mikroslužeb. Zdrojový subjekt, který autentizací token generuje, se nazývá *issuer*. Cílový subjekt, tedy ten, co token konzumuje a validuje, se označuje jako *audience*.

### 3.7 Databáze

Systém řízení báze dat (SDB) je software sloužící k operaci nad daty. Se SDB uživatel komunikuje pomocí specializovaného jazyka. V případě *relačních databází* jde nejčastěji o jazyk SQL v různých dialektech. Pro relační databázi se data modelují v podobě tabulek a jejich propojení. Vzniká *schéma databáze*. Schéma databáze jsou *metadata* popisující datový model - tabulky, jejich sloupce, datové typy sloupců a integritní omezení. Mezi nejpopulárnější databáze v oblasti webu patří MySQL a z ní vycházející MariaDB s otevřeným kódem. Komerčními relačními databázemi jsou MS-SQL od Microsoftu a Oracle. Zajímavá je databáze SQLite, která je odlehlejší verzí ostatních relačních databázových systémů. Nepodporuje přihlašování uživatelů, nekomunikuje přes síť a celý obsah ukládá do jednoho souboru. SQLite je díky své jednoduchosti oblíbená ve vestavných zařízeních nebo jako testovací databáze informačního systému.

NoSQL databáze jsou novější než databáze relační a nevyužívají tabulky. Data ukládají pomocí dvojic klíč-hodnota. Narozdíl od tabulek, jejichž sloupce mohou nabývat pouze primitivních hodnot, NoSQL databáze umožní ukládat komplexní a zanořené záznamy. Populárním zástupcem v oblasti webu je databáze MongoDB. MongoDB reprezentuje data pomocí *kolekcí* a *dokumentů*, nevyužívá databázové schéma.

Struktura dat v NoSQL databázích mnohem více odpovídá datovým strukturám v programovacích jazycích. Teba kolekce musí být v relační databázi vždy modelována další tabulkou, zatímco v NoSQL ji lze uložit přímo. Pro omezení tohoto nedostatku se u relačních databází využívá nástroj ORM (objektově relační mapování), které umožní modelovat relační data pomocí nativních struktur a kolekcí programovacího jazyka. O tvorbu SQL dotazů a transformaci dat se automaticky stará ORM. Mezi nástroje ORM patří TypeORM pro jazyk TypeScript nebo Entity framework core pro jazyk C#.

### 3.8 Programovací jazyky a aplikační rámce

Tato sekce se zaměřuje na konkrétní programovací jazyky pro tvorbu webu, jejich aplikační rámce (frameworky) a další vlastnosti ekosystému okolo těchto jazyků.

Pro každý populární programovací jazyk existuje celá řada aplikačních rámců (frameworků) pro různé účely. Webové frameworky poskytují kostru pro základní architekturu aplikace (typicky MVC) a usnadní řešení častých problémů při implementaci a návrhu informačního systému. Mezi tyto problémy patří autentizace a autorizace, správa uživatelů a rolí, komunikace s databází, tvorba aplikačního rozhraní, validace vstupů uživatele, šablony, správa závislostí, serializace a deserializace, provoz webového serveru a další. Ase různé aplikační rámce liší množstvím funkcionality, kterou nabízejí, principiálně fungují velmi podobně a výběr často závisí na osobní preferenci vývojáře.

### 3.8.1 PHP

PHP [10] je populární interpretovaný jazyk pro všeobecné užití, vhodný zejména pro vývoj dynamického webu. Získal si špatnou pověst především kvůli jeho nedůmyslnému návrhu, špatné modularitě a absenci OOP, bezpečnosti a nekonzistenci vestavěných funkcí. Postupem času se však vyvinul v plně objektově orientovaný jazyk.

PHP vyniká v několika vlastnostech, především v rychlosti vývoje menších aplikací, díky dynamickému typování a asociativním polím, a v levném provozu. Proces vývoje je pohodlný, není nutné aplikaci překládat ani restartovat webový server, všechny změny v PHP skriptech se okamžitě projeví.

Pro provoz PHP ve webové aplikaci je však vyžadován externí webový server, který se stará o komunikaci protokolem HTTP s klienty. Mezi nejznámější patří servery Apache nebo nginx. Tyto servery obsahují moduly pro spouštění a interpretaci PHP skriptů.

Níže je seznam některých klíčových vlastností jazyka PHP:

- Je to interpretovaný, dynamicky typovaný, objektově orientovaný jazyk.
- Inicivnost je iniciována HTTP požadavkem na webový server, který je konfigurován separátně.
- Ve skriptech PHP nelze spouštět a plánovat tak vykonávání kódu v budoucnu. To musí být řízeno z operačního systému, například unixovou utilitou cron. V rámci zpracování jednoho požadavku v PHP nelze využívat *paralelního* ani *asynchronního* programování. Požadavek běží na jednom vlákně, které jakákoliv I/O operace blokuje.
- PHP skript vyřizující HTTP požadavek běží v jednom vlákně, veškerý kód je tedy synchronní. To může být problém při provádění souběžných operací, jejichž výsledek není pro klienta relevantní. Klient dostane HTTP odpověď až po skončení celé operace.
- Standardní výstup PHP skriptu je směřován webovým serverem do HTTP odpovědi.
- Lze použít pouze na straně serveru, prohlížeče nemají s PHP nic společného.

Nejpopulárnější aplikační rámce pro PHP jsou v roce 2021 Laravel, Symfony a Lumen [6]. Za zmínku stojí také český framework Nette, populární v České republice.

### 3.8.2 JavaScript a Node.js

JavaScript, původně vznikl pro podporu dynamických prvků webu na straně prohlížeče, je dnes jeden z nejrozšířenějších programovacích jazyků, kterým lze programovat *frontend* i *backend*. Běhové prostředí Node.js, využívající engine V8, lze nainstalovat na běžné operační systémy a provozovat jej jako webový server. Níže je seznam některých klíčových vlastností JavaScriptu a Node.js:

- Je to jazyk interpretovaný, dynamicky typovaný, prototypově objektově orientovaný.
- Při provozu Node.js aplikace není třeba externí HTTP server jako u PHP - v Node.js lze k provozu serveru využít patřičné moduly přímo v procesu aplikace.

- Je událostmi řízený, umožňuje plánovat asynchronní akce. Podporuje *asynchronní programování*, výsledek asynchronní operace je zpracován pomocí *zpětného volání (callback)*. *Zpětné volání* probíhá hned v moment, kdy se vyprázdní *zásobník volání (callstack)* a *smýčka událostí (eventloop)* na něj přejde. Přijde toto zpětné volání z *fronty zpětných volání (callback queue)*. Programování v JavaScriptu tedy není *paralelní*, protože kód není interpretován souběžně, ale sériově. Verze standardu ECMAScript 2015 uvádí *promisy (promises)*, které nahrazují zpětná volání, protože promisy více zapadají do objektově orientovaného paradigmatu. Dvojice klíčových slov *async/await* syntakticky vylepšuje práci s promisy tak, že programátor může psát klasický synchronní kód, jak je zvyklý, a vkládat do něj asynchronní volání, na jejichž výsledek může pomocí klíčového slova *await* čekat.
- Základním balíkovacím systémem pro Node.js je npm. Podobně jako v systému unix, npm obsahuje spousty primitivních balíčků, kde každý má jednu funkci, kterou dělá požadovanou věc. Z těchto balíčků jsou potom skládány větší knihovny a frameworky.
- Veškeré závislosti jsou uloženy a spravovány ve složce `node_modules`.

Zajímavý je jazyk TypeScript, který zavádí statické typování a transpiluje se před spuštěním do JavaScriptu. Toto má však svá úskalí, protože některé typové informace, jako třeba rozhraní, jsou během zahazeny a nelze tedy s typy začít pracovat. Toto omezuje například techniku *Dependency Injection*, která je více popsána v kapitole o architektonickém návrhu.

Aplikací rámeček Express.js je minimalistický nástroj pro zpracování HTTP požadavků pomocí *middleware pipeline*, což je řetězec funkcí definujících životní cyklus požadavku, od směřování dle URL, autentizace a autorizace, po generování a zaslání HTTP odpovědi. Na Express.js zpravidla staví pokročilejší aplikací rámečky, například Loopback.io nebo nest.js.

Mezi aplikací rámečky pro tvorbu klientské části patří Angular, React a Vue. Zajímavá je kombinace Angularu na klientovi a nest.js na serveru, protože tyto rámečky jsou psané nativně v TypeScriptu a založené na podobné architektuře.

### 3.8.3 C# a .NET

Jazyk C# syntakticky vychází z jazyka C. Byl vytvořen Microsoftem jako velká konkurence staršího jazyka Java. Platforma .NET Framework spolu s webovým aplikací rámečkem ASP.NET se staly robustním řešením pro tvorbu desktopových a webových aplikací pro operační systém Windows. Postupně však byla vyvinuta nová platforma .NET Core<sup>1</sup>, což je software s otevřeným zdrojovým kódem (opensource), je spustitelný i na operačním systému Linux a obsahuje CLI (Command line interface) pro kompletní automatizaci pomocí terminálu. Tímto krokem Microsoft eliminoval největší dosavadní nedostatek své platformy a jazyk C# se tak stal plně konkurenceschopný Java. Mezi klíčovými vlastnostmi jazyka C# platformy .NET patří následující:

- C# je objektově orientovaný jazyk, silně staticky typovaný.
- Kódy jazyka C# a dalších jazyků využívaných platformou .NET (F#, VB) jsou překládány do společného mezikódu, který je před spuštěním interpretován *virtuálním strojem*.

<sup>1</sup>Současná verze (jaro 2022) platformy .NET Core je .NET 6

- C# podporuje asynchronní i paralelní programování. Asynchronní operace jsou reprezentovány pomocí *úkol (tasks)*, což jsou objekty datového typu Task. Úkoly v C# jsou konceptem podobné příslibům v JavaScriptu. C# také obsahuje klíčová slova *async/await* pro vylepšení čitelnosti asynchronního kódu. Úkoly jsou spouštěny na různých *vláknech*, jsou tedy na rozdíl od příslibů paralelní. Pro správu a recyklaci vláken pro úkoly se využívá „bazén vláken“ (*thread pool*).
- Modularita je zajištěna pomocí *jmenných prostor (namespaces)* a *sestavení (assemblies)*. Pro správu balíků se používá nástroj NuGet. Balíky jsou stahovány jako hotová *sestavení* v podobě binárních souborů DLL (dynamic linking libraries), uživatelé tedy nejsou přímo k dispozici zdrojové kódy stažených knihoven, pouze jejich rozhraní.

**Blazor** je aplikační rámec C# sloužící k programování SPA webového klienta, podobně jako Angular nebo React. Blazor může být provozován buď na serveru, nebo na klientovi. V případě provozu na serveru probíhá spojení s prohlížečem pomocí technologie SignalR, operující nad protokolem WebSocket. Veškerá aktivita uživatele na webové stránce (například kliknutí na tlačítko nebo odeslání formuláře) je vysílána v reálném čase na server, kde je zpracována. Server obstarává logiku a vysílá klientovi pouze to, co má zobrazovat. Nevýhodou tohoto přístupu je vyšší odezva uživatelského rozhraní.

Pro provoz Blazoru na klientovi se využívá moderní W3C standard WebAssembly (WASM). WASM je nízkourovňový programovací jazyk podobný Assembleru, s kompaktním binárním formátem, spustitelný ve všech moderních prohlížečích [4]. Díky existenci tohoto standardu a jeho podpory v prohlížečích je možné v klientské webové aplikaci využít jiný programovací jazyk než JavaScript.

Blazor WebAssembly zahrnuje kompletní běhové prostředí .NET ve WebAssembly, které je staženo spolu s klientskou aplikací [19]. Toto běhové prostředí je poté spuštěno přímo prohlížečem a umožňuje běh kódu pro platformu .NET. Staženy jsou také všechny knihovny DLL, na kterých aplikační kód závisí, například System.Linq pro pokročilou práci s kolekcemi. Nevýhodou je pomalejší načítání po příchodu na web, které může trvat několik sekund, nicméně běhové prostředí a potřebné knihovny DLL jsou uloženy prohlížečem v paměti *cache*, takže následně načtení je podstatně rychlejší. Výhodami jsou rychlá odezva uživatelského rozhraní a oddělenost klientské aplikace od webového serveru. V Blazoru se používá služba HttpClient pro komunikaci s aplikačním rozhraním, která interně volá fetch API prohlížeče. Blazor také poskytuje potřebné služby pro kompletní interoperabilitu kódu C# a JavaScript.

**ASP.NET Core** je oficiální webový aplikační rámec pro platformu .NET Core sloužící k tvorbě webových serverů. Pro generování webových stránek používá šablonovací engine Razor.

**Entity Framework Core** [2] je nástroj pro objektově-relační mapování. Funguje na základě *databázového kontextu*, což je objekt obsahující kolekce všech databázových entit. Po vytvoření kontextu lze pracovat s obsahem databáze transparentně - kontext sleduje změny v nativních C# objektech a kolekcích, po skonění práce s kontextem může uživatel vyvolat transakci a všechny změny uložit do databáze.

### 3.8.4 Princip techniky DI (dependency injection)

Informační systém je zpravidla rozdělen na několik menších částí, které se zaměřují na konkrétní problémy, a z nich následně sestaven. V objektově orientovaných jazycích jsou tyto jednotky reprezentovány třídami. Říká se jim také *služby (services)*. Služby často ke

své funkci potřebují jiné služby. *Závislost (dependency)* je označení pro službu, kterou daná služba potřebuje ke své činnosti. Problém předávání závislostí mezi službami řeší technika zvaná *Dependency injection*. Následující část seznámí čtenáře blíže s *Dependency injection*, protože jde o stěžejní část architektury aplikace.

Hlavní myšlenka *DI* říká, že služby se vůbec nestarají o to, *jak* a *kde* si obstarají závislosti. Jinak řečeno, služby svoje závislosti *nikdy* nevytvářejí, pouze je používají. Seznam všech závislostí je držen v datové struktuře zvané *DI kontejner*, který je inicializovaný při startu aplikace a následně automaticky vytváří instance služeb a vkládá jim potřebné závislosti. *DI* spadá pod sadu návrhových vzorů *IoC* (Inversion of Control) [13].

*DI* v podstatě zavádí do zdrojového kódu prvky *deklarativního programování* - tedy pouze *deklarují* svoje závislosti, ale už neobsahují kód k jejich získání.

Výhodné je, pokud aplikace používá *DI* a její služby navíc dodržují princip *Dependency inversion principle* z rodiny principů správné výstavby softwaru *SOLID*. Tento princip říká, že služby by měly záviset pouze na *rozhraní*, nikoliv na konkrétních implementacích. Rozhraním je pak možno předávat různé implementace při inicializaci *DI kontejneru* bez nutnosti zásahu do zdrojových kódů služeb.

Další výhodou použití *DI* je snazší *jednotkové testování* služeb - závislosti služeb je možné simulovat pomocí „falešných“ implementací (*Mock*) a soustředí se tak na funkčnost konkrétní služby. Bez *DI* by se muselo zasáhnout do kódu služeb a testování by bylo tak či tak nemožné.

Implementací **DI kontejneru** existuje pro každý programovací jazyk spousta, a kolik nabízejí různá rozhraní a různou funkcionalitu, všechny pracují na stejném základu. Nejdůležitější jsou metody *register(provider)* a *resolve(token)*

**Metoda register(provider)** slouží uživateli k vložení služby do kontejneru. Podstatné je, že parametrem není služba, ale takzvaný *provider*, což je obecná dvojice (*token, value*). *Token* je identifikátor služby a *value* je reprezentace služby. Hodnoty *token* a *value* mohou být různého typu v závislosti na programovacím jazyce a kontejneru. *Token* zpravidla bývá *rozhraní (interface)* nebo *řetězec (string)*. Pro *value* je typická *třída (class)* nebo *funkce* vracějící službu. Aplikací rámcem *ASP.NET Core* pohlíží na tuto dvojici jako na (*rozhraní, implementace*).

Po zavolání *register(provider)* je předaný *provider* pouze uložen do vnitřní datové struktury kontejneru.

**Metoda resolve(token)** vrátí objekt pro daný *token* se všemi jeho závislostmi a závislostmi závislostí (rekurzivně). Je vytvořená *strom závislostí*. Výsledná služba je připravena k použití, protože všechny její závislosti jsou automaticky vytvořeny a vloženy (injektovány) kontejnerem.

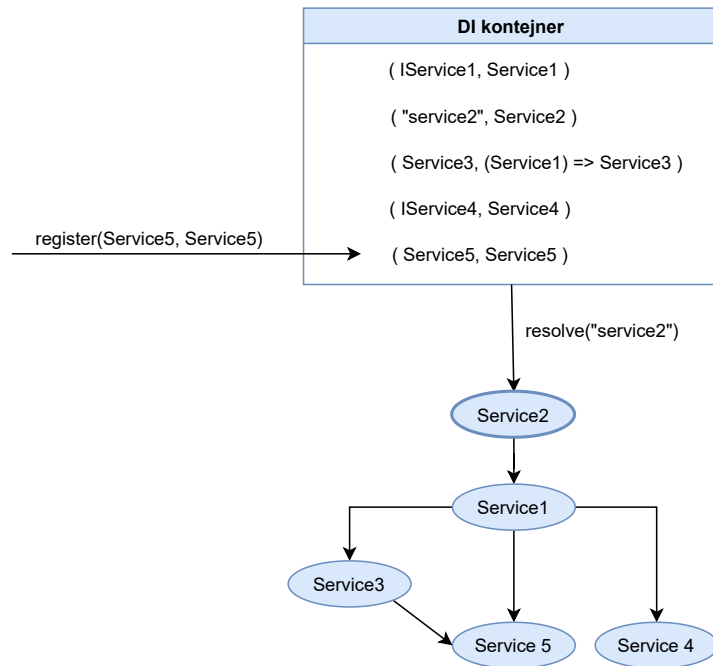
Kontejner v *ASP.NET Core* využívá k uchování a tvorbě závislostí programovací techniku *reflexe*, která je blíže popsána v sekci 5.4.1.

### 3.9 Výběr jazyka, frameworku a dodatečných technologií

Při výběru konkrétních technologií pro tuto práci jsem vycházel z požadavků na systém, z předchozích osobních zkušeností s některými technologiemi a s osobních preferencí. Následuje seznam technologií, které jsem zvolil pro různé oblasti systému, a dále, pro které jsem tak učinil.

- **Backend:** C# ASP.NET Core  
Dobrá typová kontrola a robustní opensourcová platforma je pro mě základním kom-





Obrázek 3.1: Ukázka innosti DI kontejneru - registrace služby Servi ce5 a vrácení služby Servi ce2 se závislostmi.

fortem p i programování rozsáhlejšího systému. Z toho d vodu zavrhuji PHP i JavaScri pt. V úvahu p ipadal i TypeScri pt s Node. js, nicmén schopnost *reflexe* je v tomto ja-zyku experimentální a neúplná, což se podepisuje i na nedokonalém fungování DI. Výb r mezi Javou a C# už je hlavn otázka osobní preference.

- **Frontend:** C# Blazor Server, SPA

Programování v ístém JavaScri ptu považuji za nep íjemnou zkušenost. Nabízí se robustní framework Angul ar, který je pln v TypeScri ptu. Nabízí se však také mož- nost programovat *frontend* i *backend* ve stejném jazyku - C#. To vidím jako vel- kou výhodu, protože je možné sdílet ásti kódu mezi frontendem a backendem, p i modelu Webassembl y typicky jen t ídy popisující DTO (data transfer objects), avšak využitím modelu Server lze snadno využít veškerý kód, protože a koliv jsou fronten- dová i backendová ást logicky odd leny, jsou zkombinovány v jednom projektu, což umož ũje provolávat aplika ní vrstvu p ímo z presenta ní vrstvy bez nutnosti exist- ence dalšího aplika ního rozhraní. Blazor je také skv lá p íležitost, jak se vyhnout JavaScri ptu a dát šanci rychle se vyvíjející nové technologii. Autor elektronické knihy o ASP. NET Core [21] je toho názoru, že pro jednoduchá uživatelská rozhraní není po- užití SPA vhodné kv ũli své komplexnosti. A koliv uživatelské rozhraní tohoto systému je spíše jednodušší co se tý e interaktivity s uživatelem, p esto volím SPA. D íve i pozd ji je žádoucí zavád t do UI dynamické prvky na klientské stran a dle mého názoru je vhodné na to p ipravit aplikaci od úplného základu. Použití aplika ního rámce SPA pro UI výrazn usnad ũje rozši ování uživatelského rozhraní a p idávání komponent. Využitím varianty Blazor Server dojde k eliminaci obtíží zp sobých kom- pletním odd lením kódu klientské a serverové ásti - nebude nutné implementovat dodate né aplika ní rozhraní pro komunikaci t chto ástí.

- **Databáze: MariaDB**  
Open source, dostupný ovladač pro Entity framework. Je sice výrazně složitější k instalaci a nasazení než SQLite, který nevyžaduje instalaci a konfiguraci databázového serveru, nicméně je výhodnější z hlediska výkonu a správy. Protože SQLite nebíží jako server, nelze tuto databázi jednoduše spravovat vzdáleně. NoSQL databáze se nehodí z důvodu špatné podpory v Entity frameworku.
- **Aplikační rozhraní a serializační formát:** Není třeba  
Díky vzájemné komunikaci front-endu a back-endu přes websocket, zajištěné frameworkem, není potřeba využívat žádné další aplikační rozhraní. Pokud bude v budoucnu potřeba kvůli interoperabilitě implementovat API, využije se REST. GraphQL sice může ušetřit množství dat přenesených po síti, nicméně vzhledem k potu požadavků v systému by to bylo irelevantní. GraphQL navíc nemá nativní podporu v ASP.NET Core. Serializace dat se vyskytne pouze v oblastech konfigurace a databáze. Konfigurační soubory budou ve formátu YAML, který je čitelnější než JSON. V případě serializace komplexnějších dat do jednoho databázového sloupce bude použit JSON.

## Kapitola 4

# Architektonický návrh systému

Správný návrh architektury je předpokladem pro budoucí rozšiřitelnost a udržitelnost systému. U mikroslužeb je základní architektura zajištěna už faktem, že jde o mikroslužby - systém je složen z mnoha vzájemně komunikujících částí, kde jsou vystaveny rozhraní a implementace zapouzdřeny. U monolitického systému, jako je tento, je nutné společně s rozdělením na vrstvy také rozdělit funkcionalitu do samostatných jednotek, které budou vzájemně komunikovat. V prezentované vrstvě jsou tyto jednotkami *komponenty* Blazoru. V aplikační vrstvě se jedná o již zmíněné služby. Datová vrstva se pak skládá ze tříd, které modelují schéma tabulek relační databáze. Vrstvy jsou propojeny tak, že komponenty uživatelského rozhraní přímo využívají služeb aplikační vrstvy, které dále využívají databázového kontextu a datového modelu. Obrázek 4.1 znázorňuje podrobněji obecný návrh architektury tohoto systému rozděleného do tří vrstev.

### 4.1 Identifikace konceptů a entit systému

Aby bylo možné vytvořit rozšiřitelný systém, je nutné nalézt podstatné entity a komponenty systému a stanovit míru jejich obecnosti. Jinak je třeba stanovit, co je obecné a co konkrétní. Obecné (abstraktní) prvky musí být dobře znovupoužitelné, zatímco konkrétní snadno vyměnitelné a nahraditelné. Na obrázku 4.2 je k vidění datový model systému ve formě ER diagramu.

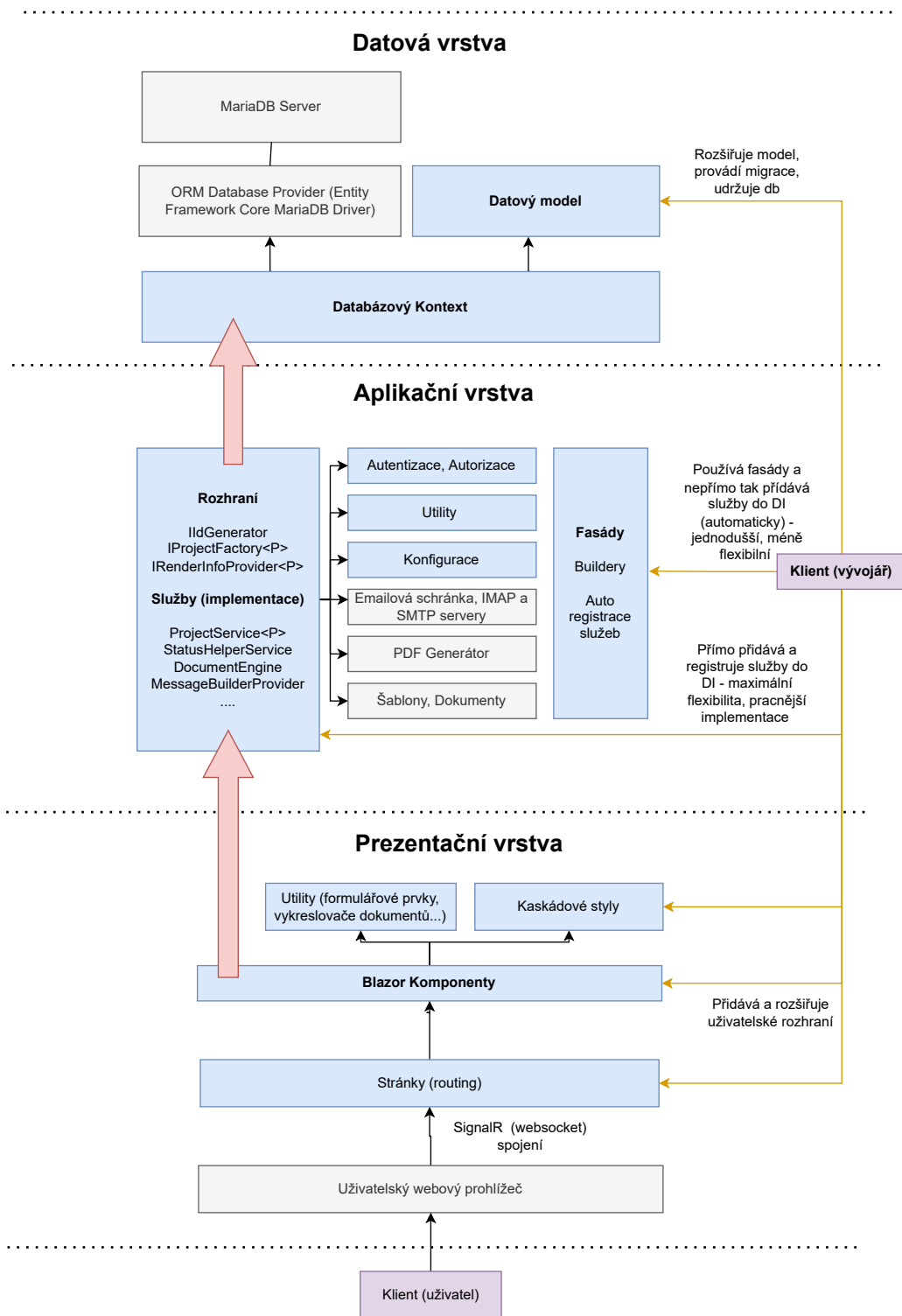
#### 4.1.1 Organizace

Organizace je libovolným členem administrativního aparátu. Může být různého typu, například výzkumná instituce nebo její laboratoř. Podstatné je, že tyto organizace mají společný základ a je možné je propojovat vztahy tak, aby vznikl strom organizací, kde nižší uzly jsou podřízené těm vyšším. Organizace jsou samostatné jednotky a nejsou závislé na dalších entitách systému.

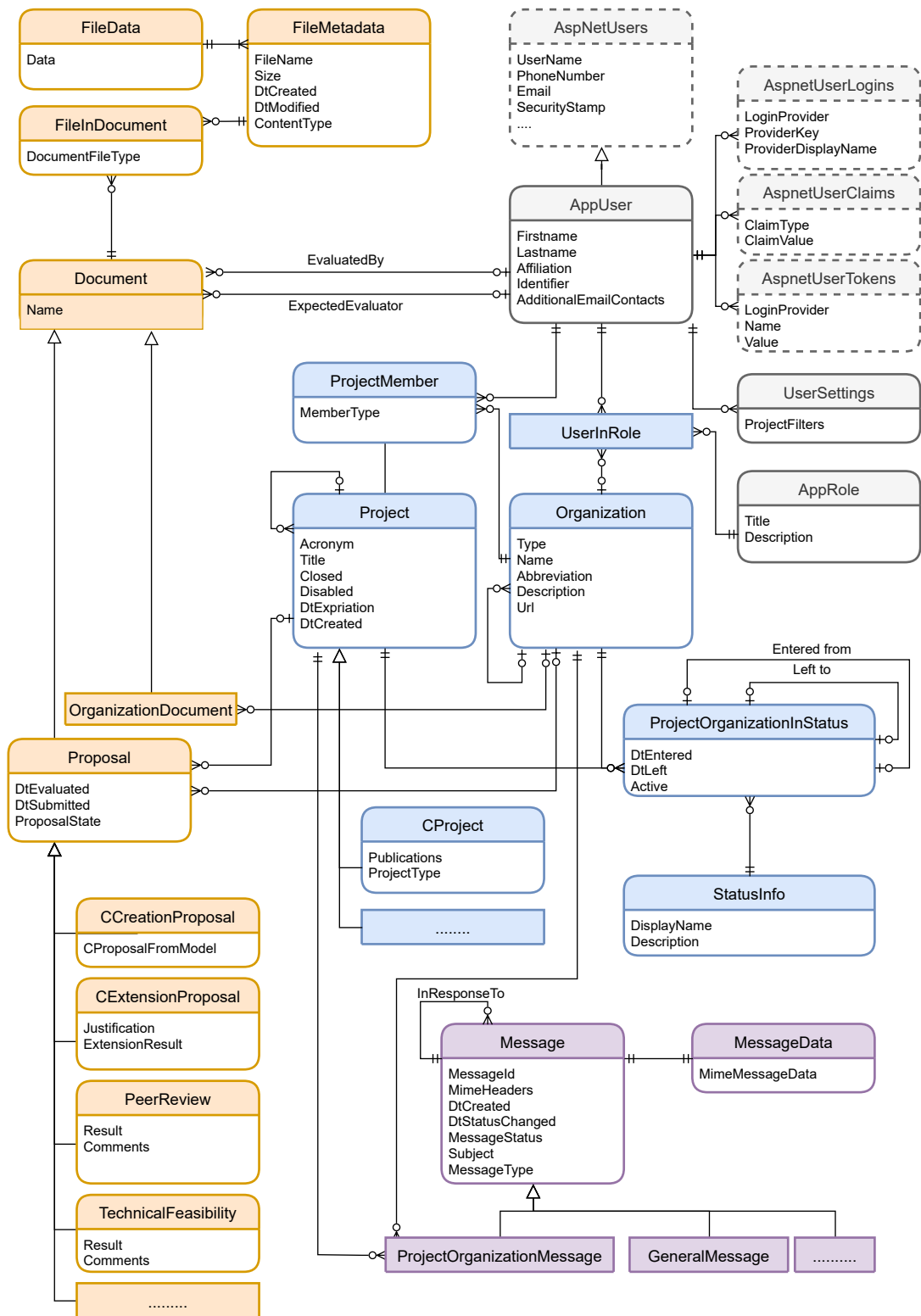
#### 4.1.2 Dokumenty a jejich evidence

Každý systém týkající se administrativy musí umět pracovat s dokumenty. Důležité je, aby bylo možné snadno vytvořit nový typ dokumentu, který bude následně možné využít ve workflow projektu. Z ostatních částí systému by mělo být možné s dokumenty pracovat transparentně. Musí tedy vzniknout modul, který bude umět:

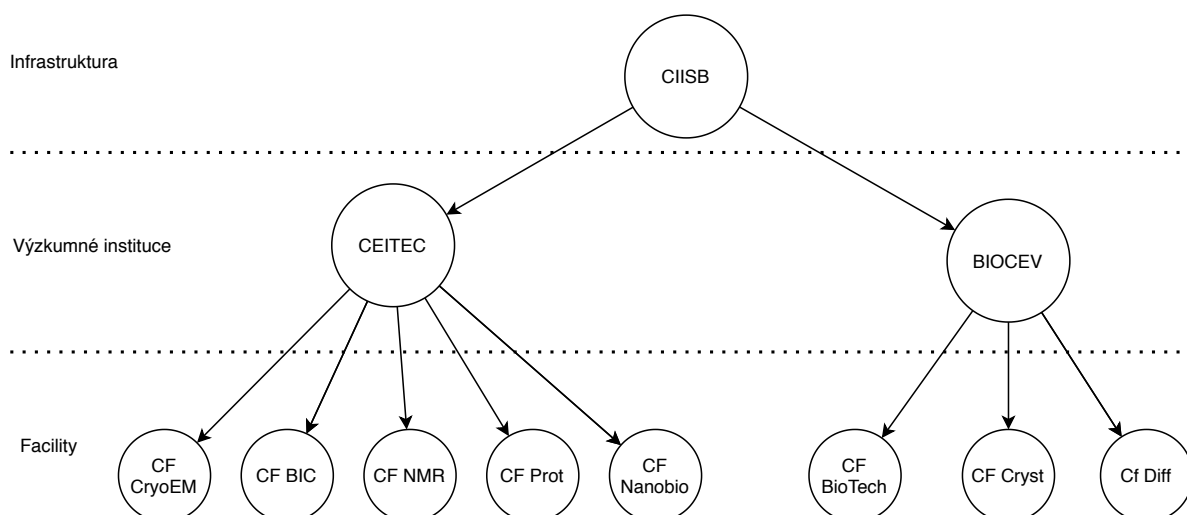
- Vytvářet, uchovávat a archivovat dokumenty různých typů.



Obrázek 4.1: T-ívrstvá architektura systému. Mod e jsou prvky implementované na platformě .NET. Červené šipky znázorňují propojení mezi vrstvami.



Obrázek 4.2: ER diagram. Znáor uje podrobn entity a jejich vztahy v datové vrstvě systému. Oranžov jsou entity související s žádostmi a dokumenty, šed entity týkající se uživatele a rolí, mod e entity reprezentující projekty, organizace a stavy a fialov entity související s emailovým modulem systému. Prázdné entity s te kami znáor ují místa, kde je vhodné datový model rozší rovat d d ním.



Obrázek 4.3: Znárodn ní stromové struktury organizací figurujících v systému CIISB

- P i azovat dokumenty k projekt m a organizacím.
- Generovat dokumenty z šablon. Generovat PDF.
- Zobrazovat dokumenty v dostupných formátech.
- Odesílat dokumenty v podob p íloh nebo odkaz .

Koncept dokument b ude rozd len na dokumenty a soubory. Jeden dokument m že být reprezentován více soubory, což zajistí pot ebnou flexibilitu. Jeden dokument tak bude moci být reprezentován ve více formátech, bude možné držet historii zm n. N které typy dokument , konkrétn p íloha k uživatelské žádosti, p ímo vyžadují podporu více soubor - a koliv osoba p í odeslání formulá e vybere více soubor , stále se sémanticky jedná o jeden a tentýž dokument.

O sémantice dokument a jejich soubor tedy tento modul nerozhoduje. Modul pouze vystaví API pro realizaci bod výše. Toto API bude využitelné ze zbytku systému.

D ležitou funkcionalitou systému je tvorba dokument pomocí šablon. Šablony jsou dokumenty, které obsahují zna ky k nahrazení konkrétními hodnotami p í generování nového dokumentu, a budou v systému reprezentovány obdobn jako ostatní dokumenty, nebudou však vázány na konkrétní projekt. Typicky to budou soubory ve formátu HTML, ale například smluvní podmínky užití laborato e zákazníkem jsou ve formátu .docx. Proto se p edpokládá existence r zných služeb pro generování dokument z šablon r zných formát . Aby bylo možné pracovat s dokumenty transparentn , výb r služby pro generování dokumentu provádí jedna obecn jší služba. Uživatel (programátor) této služby pouze vybere zdrojový dokument (šablonu), cílový dokument a kontext, což je objekt, jehož hodnoty budou do šablony vloženy. Stejn jako obsah bude generován i název souboru, který rovn ž podporuje šablonové zna ky a m že využít hodnot v kontextu. V tšina šablon bude využívat stejné prom nné, což jsou typicky informace o projektu, organizaci a url adresy (pro odkazy v emailch - emaily budou také tvo eny pomocí HTML šablon). Z toho d vodu bude možné zaregistrovat výchozí továrnu (factory) kontextu, která bude automaticky používána enginem dokument , pokud vývojá nep edá parametrem jinou. To usnadní práci s generátorem dokument , protože v tšinou nebude pot eba ru n vytvá et a p edávat kontext pro

šablonu. Výchozí továrna by tedy měla ideálně generovat kontext, který pokryje potřeby všech šablon. K tomu potřebuje pouze identifikátor projektu a organizace, které lze získat z cílového dokumentu.

### 4.1.3 Projekt

Entita projekt je jádrem systému. Každý projekt spadá pod určitý *typ projektu*. Infrastruktura CI/CD obsahuje pouze jeden typ projektu, obecně však může být implementováno více typů. Jednotlivé typy projektů jsou na sobě nezávislé, každý má svůj datový model. Datový model typu projektu může obsahovat libovolné množství dalších entit, potřebných k realizaci *workflow* projektu. Všechny projekty odcházejí od základového typu projektu, jenž je napojen na společnou část datového modelu - uživatele, stavy, role, emaily a dokumenty. Tento návrh zajišťuje neomezenou možnost přidávat a rozšiřovat typy projektů s libovolnou aplikací a datovou logikou.

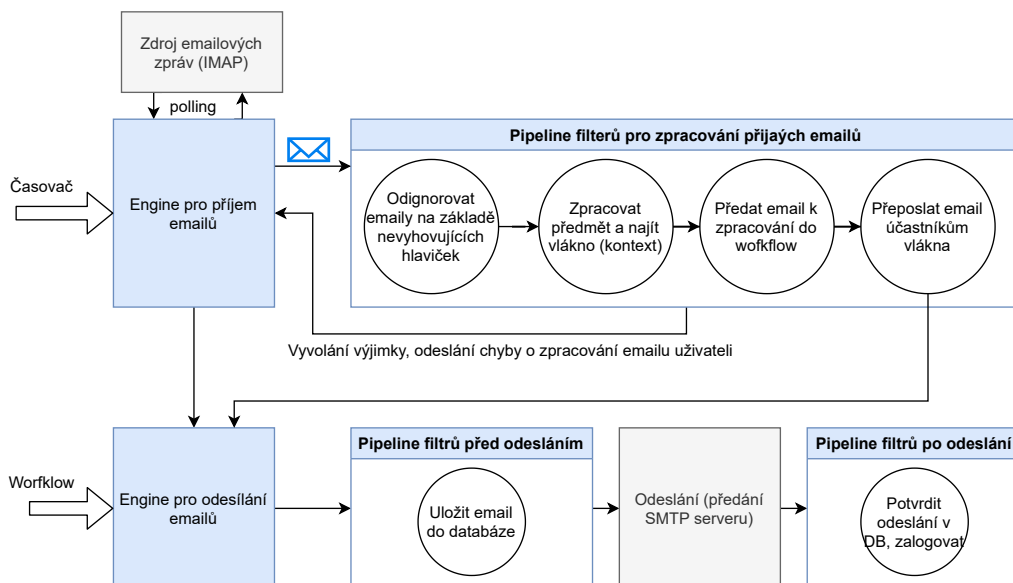
### 4.1.4 Workflow

Workflow projektu zahrnuje všechny operace nad projektem, a už vyvolané uživatelem nebo systémem. Workflow projektu může být natolik různorodý pro odlišné typy projektů, že v podstatě nelze implementovat jeden obecný workflow a ten pak využívat. Workflow by bylo velice komplikované implementovat deklarativně. Kód obsluhující workflow musí mít přístup ke všem službám systému přes DI, aby bylo možné naimplementovat jakoukoliv funkcionalitu. Jediným společným konceptem pro programování workflow různých typů projektů jsou *stavy*. Pomocí stavů může kód workflow rozhodovat o svém dalším průběhu. Důležité je, aby stav byl vázán kromě projektu také na organizaci, v rámci jednotlivých organizací tedy může být projekt v různých stavech. Žádoucí je také uchovávat historii všech stavů, vstoupení do stavů a jejich opuštění. *Akce* je jakákoliv činnost související s workflow projektu, může vykonávat neomezené operace nad projektem a jeho stavem. Příkladem akce je potvrzení technické způsobilosti projektu. Akce mohou být spouštěny těmito způsoby:

- **Přímým voláním akce.** Všechny služby a komponenty uživatelského rozhraní součástí DI, díky čemuž mohou získat libovolnou službu a zavolat libovolnou metodu (akci).
- **Příchodem emailu do systému.** Služba fungující jako *pozorovaný (observable)* bude předávat emaily *pozorovateli (observer)*, kteří budou registrováni. Pozorovatelé obdrží kontext emailu, který zahrnuje také související projekt a organizaci. Případné výjimky musí být zachyceny a zalogovány.
- **Časovačem.** Funguje obdobně jako příchod emailu. Spouští se však pravidelně jednou denně. Pozorovatelé obdrží pouze projekt a organizaci.

Pozorovatelé mohou být zaregistrováni při inicializaci systému pro jednotlivé stavy. Díky tomu obslužná rutina nemusí obsahovat kód pro kontrolu stavů a může tak být soustředěna na samotné provádění akce. Navíc lze pro více stavů využít stejnou rutinu, pokud je to žádoucí.

Služby s akcemi by měly být odděleny od jejich spouštění, aby nedocházelo k duplicitě kódu. Akcí, buď spouštěnou časovačem nebo příchozím emailem, může být někdy užitečné zavolat také z uživatelského rozhraní, nejen pro testovací účely.



Obrázek 4.4: Zet zené zpracování p íchozích i odchozích email

#### 4.1.5 Emaily

Modul pro práci s emaily musí um t p íjímat i odesílat emaily a za azovat je do kontextu *vláken (thread)* pro jejich další zpracování v systému. Musí být také schopen v rámci vlákna p eposílat emaily ve stejné podob ostatním účastník m, což umožní zú astn ým diskutovat. Žádanou funkcí je též poskytnutí email modulu workflow, protože workflow pot ebuje na n která vlákna reagovat na základ obsah email í p íloh.

Pro zpracování email se nabízí použít *zet zené zpracování (pipeline)*. Služba pro p íjem emailu bude periodicky spoušt na na pozadí asova em a pokud zjistí, že zdroj obsahuje nový email, na te jej a p edá pipeline ke zpracování. V této pipeline budou v daném po adí zaregistrovány zpracovávací uzly, které dostanou ístý email a kontext emailu, pokud existuje. Uzly mohou provád t libovolný kód a upravovat kontext. Po skon ení je ízení p edáno dalšímu uzlu v po adí. Každý uzel m že vyvolat výjimku, zastavit tak zpracování a odeslat uživateli email s notifikací o chyb , nap íklad že projekt pro identifikátor v p edm tu nebyl nalezen. Kontextu emailu odpovídá *vlákno* a jednozna n ur uje, kterému projektu a organizaci email pat í.

U odchozích email to bude fungovat stejn , jen budou mít jinou pipeline. Tento princip znázor uje obrázek 4.4. Do pipeline je možné v budoucnu registrovat další uzly a rozší ovat tak funkcionalitu celého modulu email .

#### 4.1.6 Uživatelé, role a oprávnění

Systém bude jednotn uchovávat informace o všech osobách, a už jde o zam stance institucí nebo žadatele o projekty. Uživatel m budou p í azovány role, které definují nejen oprávn ní osob, ale také jejich význam. Aby bylo možné pokrýt všechny možné vztahy uživatel v í systému, je nutné role modelovat jako komplexní entity. Budou existovat dva typy rolí:

- *Projektové role* slouží k vytvo ení vztahu mezi uživatelem, organizací, projektem nebo uživatelem a projektem. Mezi projektové role pat í žadatel projektu, vedoucí projektu, zam stanec zodpov dný za provedení projektu v rámci organizace, externí hodnotitel



zp sobilosti projektu. . . D ležit é je, že tyto role jsou vytvá eny vždy pro konkrétní projekt. Role jsou ásto p idávány i m n ny za b hu systému dle *workflow* projekt .

- *Zam stnanecké role* jsou obecn ější než role projektové. Vytvá í vztah mezi organizací a uživatelem, nevážou se na konkrétní projekt. Spadají sem vedoucí laborato í, zam stanci laborato í, administráto i projekt , externí odborníci laborato í. . . Zam stnanci a jejich role budou vytvo eny staticky p i nasazení systému, m nit se budou výjime n a bude k tomu t eba zásah správce systému, potažmo databáze.

Jeden uživatel m že vystupovat ve více rolích, nehled ěna jejich typ. Nap íklad zam stnanec laborato e nemá nic spole ného s projektem, dokud od vedoucího laborato e neobdrží projektovou roli pro zodov dnost za vy ízení projektu. Stejn ě tak funguje externí odborník pro vydání posudku zp sobilostí. Dle zam stnaneckých rolí jsou tedy uživatel m v pr b hu projektu ásto p i azovány role projektové, které jsou konkrétn ější.

Jednotlivé role budou identifikovány jednozna ným et zcem, každou roli je však ješt nutné opat it názvem a popisem, aby bylo možné informace o roli zobrazit uživateli v UI.

Oprávn ění p ímo nebudou systémem uchovávána. Autoriza ní logika bude implementována v systému a bude vycházet z aktuálních rolí uživatel . Zp sob implementace ízení p ístupu ke zdroj m více rozebírá sekce 5.5.

## 4.2 Uživatelské rozhraní (frontend)

Uživatelské rozhraní je spíše minoritní ástí systému, což je dob e, protože cílem systému je maximální automatizace. Uživatel m proto bude poskytnuto minimalistické rozhraní, kde každému uživateli bude zobrazeno pouze to, co je pro n j relevantní dle jeho role.

### 4.2.1 Pohled pro zaměstnance laboratoří

T mto uživatel m se zobrazí ve sloupci seznam projekt , každý p jde rozkliknout. Po rozkliknutí se zam stnanec dostane na detaily projektu. Zde uvidí základní stavové informace o projektu a kompletní dokument žádosti o projekt. Bude moci zobrazovat a stahovat další relevantní dokumenty. D ležit é je, že pokud zrovna bude uživatel mít v í projektu n - jakou povinnost, nap íklad vyhodnotit technickou zp sobilost žádosti pro jeho laborato , bude taková akce vždy dostate n zvyrazn a. Zvýrazn ny budou i položky v postranním seznamu projekt , aby uživatel hned v d l, kam kliknout pro spln ění povinností. Projekty, které nevyžadují akutní pozornost, budou vyzna eny neutráln ě. Úpln ě dokon ené projekty budou vyzna eny slab ě.

### 4.2.2 Pohled pro administrátory projektů

Pro administrátory je d ležit é držet p ehled o všech projektech najednou. Budou tedy mít k dipozici velkou tabulku všech projekt s n kterými jejich detaily. V této tabulce by m li mít umožn ěno filtrovat dle vhodných kritérií, nap íklad podle roku, kdy byl projekt evidován nebo podle stavu, ve kterém se nachází. S ohledem na filtry by m lý být po ítány a zobrazovány souhrnné statistiky, nap íklad kolik žadatel ě je zahrani ních nebo interních. I administráto i však mají n jaké úkoly, t eba potvrdit podepsané smluvní podmínky uživatele, proto pot ebné projekty budou rovn ě zvyrazn ny. V pohledu na konkrétní projekt jim bude umožn ěno s projektem manipulovat – nap íklad posunout nebo vrátit stav nebo znovu odeslat automatický systémový email.

### 4.2.3 Pohled emailového klienta

Jednoduchý emailový klient bude dostupný jak pro administrátory, tak pro zaměstnance laboratoří. Zde budou k dispozici k prohlédnutí všechna vlákna k projektu. Půjde odtud také odeslat email do vlákna nebo založit vlákno nové namísto použití jiného emailového klienta.

### 4.2.4 Pohledy pro ostatní uživatele

Na tyto pohledy bude odkazováno z emailu. Externí odborník obdrží odkaz na jednu stránku, kde vyplní posudek o způsobilosti projektu. Zákazník může být odkázán na stránku, kde vyplní žádost o prodloužení projektu. Pohledy vyžadující identitu uživatele musí být chráněny, budou proto vždy za odkazem s unikátním identifikátorem, který bude znám jen příjemci emailu. Důležitým pohledem je také formulář žádosti, který je jediný úplně veřejný.

### 4.2.5 Pohled pro obecnou správu obsahu

Každý dokument v systému bude možné editovat. Sekce 4.1.2 rozebírá návrh modulu dokumentů. V souladu s tímto návrhem bude existovat pohled, který umožní spravovat dokumenty na nejnižší úrovni, tedy přidávat soubory různých formátů, odebírat soubory, měnit jejich názvy, nahrávat binární soubory a editovat textové soubory. Dokument je jednoznačně dán typem, organizací a projektem, proto se předpokládá, že před editací uživatel tyto parametry zvolí. Po změně parametrů budou na něj všechny soubory přidružené k dokumentu a bude možné s nimi manipulovat. Tento pohled slouží zejména tomu, kdo má na starost vývoj a údržbu systému, nicméně bude přístupný i administrátorům projektu, kteří tak s trochou opatrnosti mohou lehce upravovat šablony nebo spravovat dokumentaci ke konkrétním projektům.

### 4.2.6 Pohled pro správu uživatelů

Úkolem tohoto pohledu bude umožnit procházet, přidávat a upravovat uživatele systému. Administrátor by měl být schopen rovněž smazat uživatele ze systému a spojit dohromady uživatele duplicitní. Měla by existovat možnost vyhledávat uživatele dle jména i emailu, upravovat role uživatelů v jednotlivých organizacích a posílat uživatelům pozvánky na přihlášení do systému s využitím externího poskytovatele identit.

# Kapitola 5

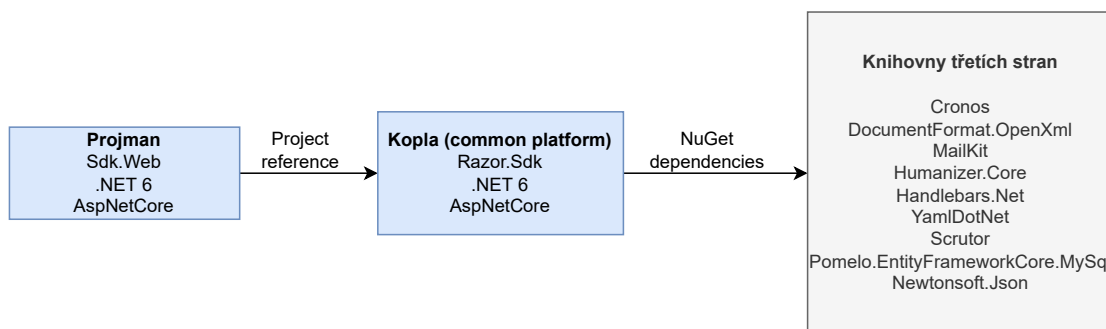
## Implementace

Tato kapitola se zaměřuje na vybrané konkrétní implementační detaily a popíše na jakém principu fungují a které nástroje a knihovny využívají.

### 5.1 Zdrojový kód, projekty, řešení a závislosti

Zdrojové kódy v jazyce C# se dělají do *projekt*. Projekty se následně sdružují do *řešení* (*solution*). Systém je rozdělen na dva projekty. Projekt zvaný Kopl a (common platform) obsahuje veškeré prvky, které mohou být využitelné a rozšiřitelné v dalších systémech. Většina rozhraní, služeb, komponent, utilit a data model se nachází právě v projektu Kopl a. Kopl a se tedy skládá z prvků ze všech tří vrstev a má roli společné knihovny. Druhý projekt, zvaný Projman (project management), je hlavním projektem, který se překládá do spustitelné aplikace. Importuje knihovnu Kopl a a obsahuje více konkrétní a obtížnější znovu-použitelný kód, který obecnou knihovnu nastavuje a využívá. Je zde implementována například logika CIISB, konkrétní stavy pro workflow CIISB, specifické role, emailové šablony a organizace... Obecná knihovna se stará například o autentizaci, správu dokumentů, správu projektů a správu uživatelů a nabízí spoustu rozhraní, která mohou nebo musí být hlavním projektem implementována a zaregistrována do DI.

Jelikož množství rozhraní k implementaci není zatím tak malé a předpokládá se, že bude stále růst, dává smysl vytvořit nad celou množinou rozhraní, služeb a registrací *fasádu*. Fasáda je návrhový vzor, jehož úkolem je poskytnout jednodušší rozhraní pro použití komplexnějšího systému služeb. Fasáda je v systému implementována dvěma způsoby, které se navzájem doplňují:



Obrázek 5.1: Projekty a závislosti systému

- Pomocí *builder*. Tento způsob je velmi používaný napříč celým rámcem ASP.NET Core. Provoláním různých metod se na pozadí registrují potřebné služby do kontejneru a automaticky se konfigurují na základě parametrů metod.
- Pomocí knihovny *Scrutor*, která slouží ke skenování projektu, vyhledání všech potřebných tříd a rozhraní a jejich automatické registraci do kontejneru dle specifikovaných parametrů. Tato varianta je automaticky rychlejší, ale méně flexibilní než buildery.

## 5.2 Konfigurace systému

Hodnoty konfiguračních proměnných, jako jsou třeba adresa SMTP serveru nebo přihlašovací údaje k databázi, je třeba uchovat odděleně od zdrojového kódu aplikace, aby je bylo možné snadno upravovat při změně prostředí. ASP.NET Core využívá k načítání těchto hodnot *options pattern*, který umožňuje načítat konfigurace různých zdrojů přímo do silně typovaných tříd v C#. Díky tomuto vzoru mohou služby záviset pouze na konfiguracích, které se jich týkají (*Interface segregation Principle*), a zároveň platí, že konfigurace pro různé části aplikace jsou na sobě nezávislé (*Separation of Concerns*) [12].

Veškerá konfigurace systému je uložena v souborech `appsettings.json` pro produkční prostředí a `appsettings.Development.json` pro vývojové prostředí. Mezi těmito prostředími se mění *proměnnou prostředí* `ASPNETCORE_ENVIRONMENT`, kterou lze nastavit při spuštění procesu na operačním systému. ASP.NET Core se postará o výběr konfiguračního souboru dle aktuálního prostředí, deserializaci dat a namapování různých sekcí konfigurace na konkrétní objekty v aplikaci. Pomocí techniky DI, popsané v sekci 3.8.4, lze tyto objekty vkládat do služeb, které je potřebují. Programátor pouze specifikuje, které sekce namapovat na které třídy, o zbytek se postará aplikační rámec.

```
// ----- EmailConnectionOptions.cs -----
// Definice konfiguračních parametrů
// pro připojení k IMAP.
public abstract class EmailConnectionOptions
{
    [Required] public string Host { get; set; }
    = null!;
    [Required] public int Port { get; set; }
    public string? Login { get; set; }
    public string? Password { get; set; }
    public SecureSocketOptions SecureSocket {
        get; set; }
}

public class IMAPOptions :
    EmailConnectionOptions { }
```

Výpis 5.1: Třída v C# popisující nastavení připojení k IMAP serveru

```
// -- appsettings.yml --
// Konfigurační YAML.
Messaging:
  Imap:
    Host: imap.gmail.com,
    Port: 993,
    Login: "***@gmail.com",
    Password: "*****",
    SecureSocket: None
```

Výpis 5.2: Část konfigurace v YAML

```
// Registrace služby do DI a-mapování konfigurace.
services.AddSingleton<IMailSource, IIMAPMailSource>();
services.Configure<IMAPOptions>(Configuration.GetSection("Messaging: Imap"));
```

```
// Konstruktor služby pro připojení k-IMAP a-injektování nastavení pomocí DI.
```

```

public IMapMailSource(IOptions<IMAPOptions> options, ILogger<IMapMailSource>
    logger)
{
    this.options = options;
    this.logger = logger;
}

```

Výpis 5.3: Kód registrace konfigurace a její využití ve službě .

K dispozici jsou 3 generická rozhraní pro konzumaci konfiguračních objektů službami: `IOptions<TOptions>`, `IOptionsSnapshot<TOptions>`, `IOptionsMonitor<TOptions>`, kde `TOptions` je třeba reprezentující konkrétní část konfigurace.

U `IOptionsSnapshot` a `IOptionsMonitor` je zajímavé, že podporují automatickou aktualizaci voleb po změně zdroje bez nutnosti restartu celé aplikace.

### 5.3 Unikátní identifikátory

Entity v databázi mají zpravidla nějaký primární klíč, dle kterého se jednoznačně identifikují. Typické je použití celočíselné hodnoty, která automaticky roste s přibývajícím počtem záznamů (*autoincrement*). Tento systém však místo celočíselných klíčů využívá identifikátory UUID (Universal unique identifier), známé také jako GUID (Globally unique identifier) [14]. Výhodou tohoto přístupu je, že identifikátor je v podstatě náhodný a není snadno predikovatelný, což jej umožňuje využívat třeba v adresách URL k odkázání na nějaký zdroj, třeba náhled nebo stažení dokumentu. Díky UUID není nutné provádět žádnou další autentizaci nebo autorizaci, protože uživatel se nedostane k UUID, která s ním nesouvisí. Tento způsob zabezpečení je sice jednoduchý, ale slabý, a proto ne vždy vhodný. UUID neslouží ke kryptografii, protože není zcela náhodný, do vygenerované hodnoty se promítá například systémový čas a MAC adresa [14].

Entity reprezentující projekty jsou výjimkou – UUID jako identifikátor nepoužívají. Identifikátory projektů musí být srozumitelné pro uživatele, krátké (kvůli použití v předmetu emailu) a poskytující alespoň nějakou informaci. Například projekt 210035C je 35. projekt typu CILSB v roce 2021.

UUID je číslo o velikosti 16 bajtů, v textové podobě se reprezentuje hexadecimálními skupinami oddělenými pomlčkami. Jazyk C# poskytuje nativní datový typ `Guid` pro generování, zpracování a uchování tohoto identifikátoru.

```

Guid uuid = Guid.NewGuid();
uuid.ToString(); // -> "122e4577-e39b-12d3-a452-426615874000"

```

Výpis 5.4: Ukázka vygenerování UUID v jazyce C#

### 5.4 Automatizace formulářů

Před popisem fungování formulářů v systému je třeba seznámit čtenáře s pokročilejšími technikami v programování – reflexí a atributy.

#### 5.4.1 Reflexe

Reflexe je v programování technika, která umožňuje programátorovi pracovat s interní reprezentací datových struktur jazyka za běhu aplikace (runtime). Staticky typované jazyky

jako je C# s sebou bez reflexe nesou jistá omezení ve flexibilitě - nelze třeba jednoduše vytvářet instance třídy, pokud je požadovaná třída dána jejím názvem v datovém typu string. V C# existuje datový typ Type, jehož instancí lze získat z libovolné instance pomocí metody GetType() nebo z libovolné třídy i struktury pomocí operátoru typeof(Třída). Reflexe nabízí několik možností využití:

- Získání názvu reflektovaného datového typu.
- Dynamická tvorba instance reflektovaného datového typu.
- Dynamické zavolání metody i přizpůsobení hodnoty do vlastnosti reflektovaného datového typu.
- Rekursivní zanoření do komplexního datového typu, a tím způsobem získání metadat o celé hierarchické struktuře.
- Práce s metadaty, které jsou přidány k třídám, metodám a parametrům prostřednictvím atributů.

## 5.4.2 Atributy

Atributy, známé též jako *dekorátory* i *anotace*, jsou metadata, která lze ve zdrojovém kódu přizpůsobit jednotlivým jazykovým konstrukcím (definice třídy, vlastnosti třídy, metody, prvky výčtového typu...). Podstatné je, že pomocí těchto metadat a reflexe je možné doplnovat automatickou funkcionalitu, aniž by bylo nutné tomu přizpůsobovat jednotlivé konstrukce. Hojně toho využívají aplikační rámce, například ASP.NET Core poskytuje třídě atribut [ApiController], který specifikuje, že obyčejná C# třída je nyní MVC kontrolér a framework automaticky zaregistruje endpointy. To vše bez nutnosti třídu odněkud dít nebo konfigurovat na jiném místě.

Toho také využívá modul automatických formulářů v systému. Pro každý formulář stačí vytvořit pouze jeho model, který se skládá z hierarchie objektů. Vlastnosti třídy reprezentují jednotlivé formulářové prvky. Pomocí atributů se doplňují validační požadavky (například zda je pole povinné) a vykreslovací požadavky (titulek pole nebo nápis nad polem). Díky použití klientského aplikačního rámce C# Balzor Webassembly lze kód modelu sdílet mezi klientem a serverem a zautomatizovat server-side i klient-side validace, vykreslování formulářů na klientovi i vykreslení dokumentu z formulářových dat na straně serveru.

```
public class PersonDetails
{
    [Required]
    public string FirstName { get; set; }
    [Required]
    public string Surname { get; set; }
    [EmailAddress]
    [Required]
    [Render(Tip = "If you actively use multiple email addresses, please provide
        all of them (comma separated), so we can identify your future email
        messages. Emails with unknown sender address will be rejected and not
        processed.")]
    public string Email { get; set; }
    [Phone]
    public string PhoneNumber { get; set; }
    [Selection("Researcher", "Ph.D. Student", "MSc student")]
```

```
} public string Position { get; set; }
```

Výpis 5.5: Ukázka formulářového modelu pro osobu.

## 5.5 Autentizace

Převodním plánem bylo implementovat autentizaci podobným způsobem jako v prototypu. To by znamenalo využít software Shibboleth, který implementuje protokol SAML 2. Toto však předpokládá použití webového serveru Apache (Shibboleth je jeho modulem) jako *reverzního proxy serveru* mezi klientem a serverem Kestrel, který je součástí .NET Core. Tento přístup by jednak zkomplikoval prostředí, ve kterém systém běží, druhak by omezil flexibilitu následné autorizace – Apache by blokovalo přístup neautentizovaným uživatelům na všechny URL s nakonfigurovaným prefixem, mohlo by ale být žádoucí připustit uživatele na dané URL, když je autentizován jiným způsobem, třeba unikátním tokenem nebo jiným autentizačním mechanismem než SAML 2. Proto je lepší nechat autentizaci a autorizaci plně v kompetenci .NET Core systému. Problém je, že protokol SAML 2 je mnohem starší než současné alternativní protokoly (OpenID, OAuth) a nový .NET Core neposkytuje žádnou implementaci protokolu SAML. A koliv s pomocí knihovny téže strany lze relativně jednoduše dosáhnout potřebné funkcionality, většina knihoven je placená a jinak nevhodně licencovaná. Součástí nového systému je tedy i velmi odlehčené vlastní řešení autentizace pomocí SAML 2.

Důležitým rozhodnutím je, zdali využít JWT nebo Cookies [3.6.1](#). A koliv JWT je považováno za vhodnější řešení v případě SPA aplikací a API, bylo zvoleno řešení pomocí Cookies. Důvodem je, že tento způsob je přirozenější a snažší na implementaci v případě, že klientem je webový prohlížeč. Cookies jsou součástí protokolu HTTP a prohlížečem implementovány, zatímco u JWT je vyžadována další režie na straně klienta.

Příkladem, kdy má být JWT problematické, je zobrazování dokumentu pomocí elementu `<i>frame</i>`. V případě cookies stačí předat elementu cílovou URL adresu, prohlížeč připojí cookie automaticky. Není ale možné elementu nastavit, aby odeslal JWT v hlavičce požadavku. Právě element `<i>frame</i>` se hodí například pro zobrazování HTML a PDF dokumentů jako náhled přímo ve webové stránce.

V .NET Core je navíc jednodušší implementovat cookies než JWT. Zajímavé je, že ve výchozím nastavení není obsahem cookies identifikátor sezení, ale všechny informace o autentizovaném uživateli v podobě *tvrzení (claims)*. Cookies je také kryptograficky podepsáno a chráněno proti podvržení. V tomto případě se cookies od JWT liší v podstatě jen tím, že využívá mechanismu HTTP cookies namísto HTTP hlaviček, server tedy neukládá sezení a je zachována bezstavovost.

Přestože hlavní klientská autentizace je implementována na bázi cookies, systém umožňuje také generovat a validovat autentizační tokeny (JWT), což je využito například při generování odkazu na chráněný zdroj a jeho odeslání emailem. V případě nutnosti v budoucnu založit API kvůli komunikaci s jinou aplikací je již nyní existencí tokenů také výhodou.

K implementaci JWT je využito oficiální knihovny, která poskytuje takzvaný *autentizační handler*. Integrace tohoto handleru je jen otázkou konfigurace. Handler však neslouží k vytváření JWT tokenů – tato funkcionality je řešena vlastní dodatečnou službou, která využívá kryptografických knihoven platformy .NET.

### 5.5.1 Vlastní SAML 2 handler

Systém obsahuje vlastní autentizační modul pro SAML 2, který je navržen a proveden tak, aby byl v souladu s ostatními autentizačními službami pro .NET Core a integroval se do .NET Core podobným způsobem. V procesu autentizace figurují 3 strany: uživatel (webový prohlížeč), server požadující autentizaci, zvaný Service provider (SP) a server provádějící autentizaci, zvaný Identity provider (IDP). Proces autentizace vypadá následovně:

1. Uživatel zvolí přihlášení pomocí federace identit.
2. Systém přesměruje uživatele na stránku, kde si uživatel zvolí ze seznamu konkrétní instituci (IDP). Tato stránka je poskytnuta federací. Součástí URL na tuto stránku je parametr specifikující URL, na kterou se má prohlížeč po výběru instituce vrátit. Stránka s výběrem tam uživatele přesměruje a přidá URL parametr s konkrétním identifikátorem IDP, kterého uživatel zvolil.
3. Systém zjistí identifikátor IDP a vyhledá si jeho metadata. Pokud nejsou metadata v cache, stáhnout se ze serveru federace identit. Metadata jsou ve formátu XML soubor a probíhá jejich zpracování, podstatné informace jsou uloženy do standardních C objektů. Metadata jsou uložena do cache, kde vydrží až do restartu systému.
4. Pokud jsou metadata IDP k dispozici, uživatel je přesměrován na server IDP, kde autentizační služby této strany (SP) probíhají. Adresa je získána z metadata IDP. K URL je připojena dvojice parametrů. Parametr SAMLRequest je systémem generovaná XML žádost o přihlášení zakódována pomocí base64 kódování. Parametr RelayState obsahuje libovolnou hodnotu, která má být během komunikace udržena a navrácena systému společně s výsledkem autentizace.
5. Uživatel, nyní na webu IDP, provádí autentizaci jménem a heslem.
6. Web IDP přesměruje uživatele zpět na web SP. Cílová adresa SP se nachází v metadatach SP, která IDP získává ze serveru federace. Součástí URL je opět dvojice parametrů. SAMLResponse obsahuje XML odpověď generovanou IDP. Tato odpověď obsahuje mimo jiné údaje o přihlášeném uživateli a digitální podpis, který musí SP ověřit. RelayState obsahuje totožnou hodnotu, která byla přítomna v požadavku o přihlášení.
7. SP extrahuje informace z XML odpovědi, ověří datum expirace a podpis odpovědi. Veřejný klíč ověření podpisu získá z metadata IDP.
8. Pokud je odpověď validní, vyhledá SP ve své databázi uživatele dle dostupného identifikátoru v odpovědi. Pokud neexistuje, založí jej.
9. Všechny dodatečné lokální informace o uživateli jsou na něj (například role uživatele) a uloženy v objektu ClaimsPrincipal, ze kterého je vytvořeno zabezpečené cookie. O tvorbu cookie se stará jiný autentizační handler, který je součástí frameworku. Tento handler také cookie validuje a autentizuje uživatele v následujících požadavcích.

## 5.6 Autorizace

Autorizace je v .NET Core oddělena od autentizace, propojujícím prvkem je již zmíněný objekt ClaimsPrincipal, který je vytvářen autentizací a využíván autorizací. ClaimsPrincipal



obsahuje informace o uživateli v podobě tvrzení, která jsou reprezentována jako jednoduché dvojice typ-hodnota, kde typ i hodnota jsou obyčejné et zce. N které typy tvrzení (například jméno, email, telefonní číslo, role) jsou standardizovány.

Autorizovat požadavky je možné bu to deklarativně nebo imperativně. Deklarativní autorizace spoívá v dekoraci jednotky (Blazor stránky, metody v kontroleru...) atributem, který obsahuje informaci, která uživatelská role musí být přítomna nebo který autorizační proces (*policy*) musí být úspěšný. Autorizace probíhá automaticky. Imperativní autorizace volá přímo autorizační služby frameworku v momentě potřeby. Oba způsoby systém využívá.

## 5.7 Dokumenty a správa obsahu

V sekci 4.1.2 bylo rozebráno, jakým způsobem má být obsah logicky členěn, pro implementaci je nutné ještě stanovit, jak budou data soubor uložena. Možným řešením je využití souborového systému a tedy práci se soubory přímo na úrovni operačního systému. Dalším řešením je uložit soubory přímo v databázi. Každý přístup má svoje výhody a nevýhody.

Souborový systém operačního systému udržuje kromě dat i metadata souboru, tedy jeho název, datum vytvoření a datum změny. Databáze MariaDB nabízí pro uložení obsah souborů různé varianty datového typu BLOB (Binary Large Object), které se liší maximální velikostí uložených dat, konkrétně TINYBLOB (do 255B), BLOB (do 65 535B), MEDIUMBLOB (do cca 16MB) a LARGEBLOB (do cca 4GB). Tento datový typ se vyznačuje tím, že data jsou uložena v binární podobě stejně tak, jak byla předána databázovému enginu [1]. Nicméně neukládá automaticky metadata souboru jako operační systém, je proto nutné toto implementovat na úrovni aplikace. Dalším faktorem ke zhodnocení je rychlost, otázka srovnání rychlosti je však komplikovaná a záleží na spoustě faktorech. Dle publikace [9] je práce se soubory menšími než 256KB efektivnější, pokud je řešena databází, naopak pro soubory větší než 1MB je výhodnější souborový systém. Tato studie se však týká databáze MS-SQL a souborového systému NTFS. Většina souborů v systému bude do 100 Kb.

Pro systém byl zvolen přístup uložení všech dat aplikace v databázi. Data a metadata je vhodné uložit zvlášť, tedy do samostatných tabulek. Nejčastěji se pracuje pouze s metadaty souboru a data se tak nahrají jen tehdy, až je to nutné.

### 5.7.1 Dokumenty HTML jako šablony

Pro generování dokumentů z HTML šablon je využita knihovna Handlebars.NET. Pro optimalizaci knihovna nejprve „zkompiluje“ šablonu a tím ji upraví. Následně lze předávat zkompilevané šablony datové kontexty a generovat výstupní dokumenty. Při inicializaci knihovny lze registrovat *pomocníky (helpers)*, což jsou metody, které lze podle identifikátoru provolávat přímo z šablony. Pomocníci rozšiřují celý šablonovací systém o další funkcionality, například generování adresy URL.

### 5.7.2 Dokumenty DOCX jako šablony

Soubor .docx je ve skutečnosti archiv .zip. V obsahu archivu lze nalézt dokument ve standardizovaném formátu Office Open XML. Šablonovací služba pro .docx tedy nejprve získá tento dokument a následně předání již existující služba pro formát HTML. Po nahrazení šablonovacích znaků se dokument XML uloží zpět do archivu .docx. Problematika nahrazování v těchto dokumentech je však rozsáhlejší, protože textový obsah dokumentu může být a zpravidla bývá rozdíl mezi více XML tagy, což znemožňuje provést vyhledání

a nahrazení. Proto je třeba XML nejdříve upravit tak, aby se všechny značky pro nahrazení nacházely v jednom XML elementu. Toho lze docílit spojením dotyčných elementů v první element, kde značka začíná.

### 5.7.3 Generování dokumentů PDF

Dokumenty PDF lze vytvářet manuálně nebo převodem z formátu HTML. Manuální převod je příliš pracný, proto je pro systém zvolen druhý způsob. Existuje množství knihoven realizujících převod dokumentů HTML do PDF. Ty, co jsou zdarma, však zpravidla nepodporují CSS. Alternativou ke knihovně je také využití hotového webového vykreslovacího engine, který umí tisknout do PDF.

Opensource projekt `wkhtmltopdf` je konzolová aplikace k převodu webových stránek do PDF. Používá framework Qt a renderovací engine WebKit. Funguje zároveň jako HTTP klient - na vstupu přijímá URL adresu webové stránky k převodu. Načítá všechny obrázky, CSS styly a umí spouštět i JavaScript. Pomocí parametrů lze nastavit okraje stránky, formát, záhlaví a zápatí. Pro snazší integraci tohoto nástroje do C# aplikace existuje několik komunitních *adaptérů*, které se starají o spouštění procesu a předání vstupů a výstupů. Tento nástroj byl používán prototypem, nicméně měl dvě úskalí - nemožnost vložení záhlaví a zápatí přímo pomocí HTML a CSS (nutné použít argument procesu) a chybějící podpora pro stylování rozložení pomocí `float` boxu.

V nové implementaci systému je použit nástroj `weasyprint`, vytvořený v jazyce Python. `Weasyprint` lze po instalaci využít jako modul pro aplikaci v Pythonu nebo jako samostatnou konzolovou aplikaci s CLI (Command line interface). Tento nástroj není postaven nad kompletním vykreslovacím enginem jako jsou WebKit nebo Gecko. Využívá více menších knihoven a pro rozložení pomocí CSS používá vlastní implementaci [20].

Při použití takového je nutné dát pozor na dva problémy:

- **Ošetření vstupů.** Je naprosto nepřijatelné, aby se jakýkoliv vstup od uživatele mohl dostat do parametru prováděného příkazu. Proto je při použití aplikace `weasyprint` využíván standardní vstup. Jediným argumentem je výstupní formát, který je zadán staticky.
- **Rychlost.** Generování je prováděno na pozadí a neblokuje výsledek HTTP požadavku.

### 5.7.4 Odkazy v dokumentech

Některé dokumenty, zejména emaily, obsahují odkazy na části webu. Bylo by nerozumné psát odkazy ve finální podobě přímo do šablon, protože v emailech a souborech PDF musí být odkazy vždy absolutní, tedy obsahovat i cílovou doménu. Při změně domény, která je závislá na prostředí, ve kterém je systém provozován, by bylo nutné změnit také všechny šablony. Pro šablonovací engine `Handlebars` je proto registrován pomocník, který konvertuje relativní odkazy na absolutní a vypisuje je jako HTML kód.

Část obsahu webu musí být možné zpřístupnit uživateli, kteří se v systému neautentizují. Příkladem je formulář posudku externího odborníka. Externí odborník obdrží email s unikátním odkazem. Systém umožní řešit tento problém dvěma způsoby:

1. Využít faktu, že většina entit v systému používá UUID jako svůj identifikátor. Stránka týkající se konkrétní entity nemusí být chráněna autorizací, protože UUID obsažené v URL je náhodné a není snadno odhadnutelné. Toto je řešení vhodné pro méně citlivé zdroje.

2. Vygenerovat JWT token autentizující uživatele a připojit jej k URL jako parametr s názvem token. Systém token v URL detekuje a provede autentizaci.

## 5.8 Časování akcí

Pravidelné zpracování přijatých emailů a denní kontrola všech projektů jsou prováděny na pozadí, což je možné zajistit díky tomu, že C# podporuje vícevláknové programování pomocí *úloh (tasks)*. Není tak nutné zavádět systém cron, jako by tomu bylo v případě PHP. Do DI kontejneru jsou zaregistrovány speciální služby, které slouží k spouštění a provádění akcí na pozadí. Systém obsahuje dvě časované akce - denní, která se stará o spouštění registrovaných událostí pro všechny projekty v systému, a pětiminutovou, která stahuje a zpracovává příchozí emaily z IMAP serveru. V případě potřeby nové časované akce je možné ji vytvořit odděleně od společné rodičovské třídy `SchedulerService` a naimplementovat metodu `ExecuteRoundAsync()` a společně s konfigurací zaregistrovat do DI kontejneru.

## Kapitola 6

# Testování

Testování softwaru lze rozdělit na manuální a automatické. Manuální testování provádí osoba, zatímco automatické program. Automatické testy se dále dělí na typy dle vrstev, které testují. Mezi typy testů patří například testy *jednotkové*, *integrační* a *akceptační*. Automatizované testování je časově náročnější disciplína, mnohdy pracnější než tvorba samotného programu, který je testován. Automatizované testování není součástí této práce.

### 6.1 Průběžné testování

Největší část testování probíhala jak při vývoji a nasazení prototypu, tak při vývoji nového systému. Během reimplementace systému byl prototyp dlouhodobě vystaven plnému provozu a nasazen na produkčním serveru. Během provozu byla odhalena spousta chyb a problémů, které pak mohly být při implementaci a nasazení nového systému podchyceny. Systém byl navržen s ohledem na možnost budoucích změn, tedy testovacím a produkčním. Všechny parametry, jež se mohou s prostředím měnit, jsou plně konfigurovatelné prostřednictvím jednoho souboru. Níže je seznam úskalí testování spojená nejen se změnami prostředí, která bylo nutné nějakým způsobem vyřešit.

- **Odesílání a příjem emailů:** Při testování bylo využito SMTP a IMAP serveru společnosti Google a byl vytvořen nový účet pro testovací účely. Na produkčním prostředí jsou využívány servery Masarykovy univerzity, které nebylo však možné v lokálním prostředí pohodlně použít, protože jsou limitované firewallem na specifický rozsah adres.
- **Adresáři emailů:** Testování je nejlepší provádět na datech odpovídajících datům reálným. Je však nepřípustné při testování posílat emaily na reálné adresy i neexistující adresy. Pro tyto účely byl připraven emailový modul tak, aby bylo možné v konfiguraci nastavit finálního adresáta. SMTP služba před samotným odesláním emailu zahodí z hlaviček všechny příjemce, vepíše je do těla emailu a nastaví pouze testovacího příjemce.
- **Impersonace:** Při testování je dobré vidět, jak rozhraní systému vypadá pro reálné uživatele po jejich přihlášení. Proto byla implementována funkcionálita, která umožňuje virtuální přihlášení. Oprávněný uživatel si může vybrat uživatele, na kterého se přihlásí a následně vidí rozhraní přesně tak, jako ho vidí daný uživatel autentizovaný doopravdy.

- **Metadata federace:** Autentizaci pomocí SAML2 nebylo možné simulovat ve vývojovém prostředí, protože produkční metadata služby ve federaci jsou vázána na konkrétní URL, které je ve vývojovém prostředí jiné. Nemožnost autentizovat přes SAML2 byl jeden z důvodů, proč byla do systému přidána autentizace přes Google.

V době, kdy ještě systém nebyl nasazen, byla velká část společné platformy otestována používáním jiného projektu, který tuto platformu využívá.

## 6.2 Poznatky různých skupin uživatelů

V průběhu vývoje a především už po nasazení *prototypu* měli koncoví uživatelé možnost systém vyzkoušet, nkteří se podíleli o své poznatky a návrhy na zlepšení.

- **Žadatelům o projekty** se nelíbil fakt, že vstupní formulář je příliš rozsáhlý a jeho opakované vyplnění v případě zamítnutí projektu kvůli několika nesrovnalostem v žádosti je časově náročné. Řešením je zahrnout speciální odkaz do emailu, který potvrzuje přijetí žádosti. Tento odkaz navede uživatele na vstupní formulář, který ale bude už vyplněn daty z předchozího projektu. Další žádostí bylo vylepšení formuláře tak, aby některá pole nabízela komplexnější formátování textu, tedy například bylo možné psát vzorce s horními a dolními indexy. Tento návrh je stále na zvážení, nebo jde o poměrně složitou problematiku i z hlediska bezpečnosti, protože by to znamenalo zobrazovat uživateli obsah HTML, který je list vstupem jiného uživatele. To vyžaduje dobré zabezpečení proti útokům XSS.
- **Administrátoři projektů** by potřebovali komplexnější funkcionalitu k přehledové tabulce s projekty. Kromě běžných filtrů podle roku, státní příslušnosti žadatele nebo stavu projektu by bylo vhodné umožnit filtrovat pomocí textu. Bez této funkcionality musí administrátoři v lepším případě používat vyhledávání v prohlížeči pomocí CTRL+F, v horším případě tabulku kopírovat do excelu a filtrovat tam, což je zbytečně kontraproduktivní.
- **Externí odborníci** neměli žádný problém. Jejich celá interakce se systémem spočívala ve vyplnění jednoduchého potvrzovacího formuláře.
- **Zaměstnanci laboratoří**, konkrétně vedoucí laboratoří, by ocenili možnost zmítní externího odborníka pro posudek projektové žádosti, pokud vybraný externí odborník delší dobu nereaguje. Žádoucí byla také možnost projekt dočasně odmítnout a nechat uživatele, aby do žádosti doplnil informace. Tato funkcionalita je na zvážení, protože výrazně komplikuje workflow projektu. V současné verzi je po zamítnutí projektu nutné vytvořit projekt nový.

## 6.3 Migrace systémových dat

Testování nového systému bylo vhodné provádět na skutečných datech vzniklých během více než ročního provozu prototypu. Jelikož datové modely systému jsou navzájem naprosto nekompatibilní, migrace starých dat byla poměrně problematická. Proto musela pro tuto migraci dat vzniknout speciální služba, která načítá data ze staré databáze, zpracovává je, změní jejich strukturu a následně ukládá data do databáze nové. Kromě načítání dat pomocí SQL dotazů musí služba načítat i soubory přidružené k žádostem a projektům, protože

prototyp narozdíl od nového systému neukládal dokumenty do databáze, ale na disk. Musely být také vygenerovány PDF dokumenty všech CIISB žádostí o projekt a natvrdo vloženy do nového systému bez zdrojových dat, protože datový model těchto žádostí je tak rozsáhlý, že implementovat jeho migraci by se nevyplatilo.

## Kapitola 7

### Závěr

V rámci práce byl vyvinut webový informační systém postavený na technologiích platformy .NET 6. Před samotnou implementací byly pomocí prototypování získány požadavky na systém a následně analyzovány. Text práce se dále v noval technologiím v rozsáhle oblasti webového vývoje. Na základě požadavků, průzkumu technologií a zkušeností vývojářů byly vybrány technologie pro implementaci. Data z prototypu byla přemigrována do nového systému a prototyp byl na produkčním serveru nahrazen novým systémem.

Na které části výsledné implementace však nejsou úplně v souladu s tím, jak byly původně navrženy. Ukázalo se, že některé návrhy byly zbytečně složité a pracně realizovatelné, zejména v oblastech emailů a dokumentů. Systém je proto implementuje trochu jednodušším a přímějším způsobem. Ve výsledném systému také chybí pohled pro obecnou správu dokumentů a emailový klient. Tyto pohledy však nejsou tolik zásadní a je možné je dodat až později, přestože už je systém v provozu. Implementace těchto pohledů tedy pravděpodobně bude předmětem budoucího vývoje.

Hlavním cílem práce bylo vytvořit systém, který by byl lépe rozšiřitelný a udržovatelný než jeho prototyp. Zavedením techniky DI byly tyto předpoklady dobře podchyceny. Přidání fasád pak usnadnilo a zautomatizovalo samotnou inicializaci DI kontejneru.

Systém je nasazen v produkci a je funkční. V budoucnu bude vhodné ještě provést v rámci refaktORIZACE kódu, doplnit komentáře a sepsat technickou dokumentaci, která vysvětlí detailně architekturu jednotlivých modulů a poskytne návod, jak na systému dále stavět a rozšiřovat jej.

# Literatura

- [1] *BLOB and TEXT Data Types* [online]. MariaDB [cit. 2022-05-03]. Dostupné z: <https://mariadb.com/kb/en/blob-and-text-data-types/>.
- [2] *Entity Framework Core* [online]. Microsoft, 20. září 2020 [cit. 2021-04-08]. Dostupné z: <https://docs.microsoft.com/en-us/ef/core/>.
- [3] *OpenAPI Specification* [online]. Swagger, 20. února 2020 [cit. 2021-04-08]. Dostupné z: <https://swagger.io/specification/>.
- [4] *WebAssembly / MDN*. Mozilla, 27. března 2021 [cit. 2021-04-06]. Dostupné z: <https://developer.mozilla.org/en-US/docs/WebAssembly>.
- [5] Arlow, J. a Neustadt, I. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky*. 2. vyd. Computer Press, 2007. ISBN 978-80-251-1503-9.
- [6] Brotherton, C. *The Most Popular PHP Frameworks to Use in 2021* [online], 20. března 2021 [cit. 2021-03-27]. Dostupné z: <https://kinsta.com/blog/php-frameworks/>.
- [7] Burget, R. *Informační systémy: Architektury informačních systému* [online], 26. září 2019 [cit. 2021-04-07]. Dostupné z: [https://wis.fyt.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FIS-IT%2Flectures%2Fp02\\_Architektury.pdf&cid=13985](https://wis.fyt.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FIS-IT%2Flectures%2Fp02_Architektury.pdf&cid=13985).
- [8] CESNET. *eská akademická federace identit eduID.cz* [online]. [cit. 27.3.2021]. Dostupné z: <https://eduid.cz>.
- [9] Gray, J. *To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem*. MSR-TR-2006-45. April 2006. 10 s. Dostupné z: <https://www.microsoft.com/en-us/research/publication/to-blob-or-not-to-blob-large-object-storage-in-a-database-or-a-filesystem/>.
- [10] Group, T. P. *PHP: Hypertext Preprocessor* [online]. [cit. 2021-01-17]. Dostupné z: <https://php.net>.
- [11] Koří, R. a Křena, B. *Úvod do softwarového inženýrství, 2. přednáška* [online]. 2020 [cit. 2021-04-04]. Dostupné z: <https://wis.fyt.vutbr.cz/FIT/st/cfs.php.cs?file=%2Fcourse%2FIS-IT%2Flectures%2FIS2.pdf&cid=12807>.
- [12] Larkin, K. a Anderson, R. *Options pattern in ASP.NET Core* [online], 20. května 2020 [cit. 2021-04-05]. Dostupné z: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/options?view=aspnetcore-5.0>.



- [13] Larkin, K., Smith, S., Addie, S. a Dahler, B. *Dependency injection in ASP.NET Core* [online], 21. srpence 2020 [cit. 2021-04-07]. Dostupné z: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection?view=aspnetcore-5.0>.
- [14] Leach, P., Mealling, M. a Salz, R. *A Universally Unique Identifier (UUID) URN Namespace* [online]. srpenec 2005 [cit. 2021-04-05]. Dostupné z: <https://tools.ietf.org/html/rfc4122>.
- [15] Masaryk University. *Czech Infrastructure for Integrative Structural Biology* [online]. [cit. 2020-01-16]. Dostupné z: <https://www.cisb.org/about-cisb/about-cisb>.
- [16] Microsoft. *What is XML Schema (XSD)?* [online], 27. října 2016 [cit. 2021-01-17]. Dostupné z: [https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms765537\(v=vs.85\)](https://docs.microsoft.com/en-us/previous-versions/windows/desktop/ms765537(v=vs.85)).
- [17] Quin, L. *The Extensible Stylesheet Language Family (XSL)* [online]. 2017 [cit. 2021-01-17]. Dostupné z: <https://www.w3.org/Style/XSL/>.
- [18] Redhat. *What is a REST API?* [online]. [cit. 2021-03-27]. Dostupné z: <https://www.redhat.com/en/topics/api/what-is-a-rest-api>.
- [19] Roth, D. *Blazor WebAssembly 3.2.0 now available* [online]. Microsoft, 19. května 2020 [cit. 2021-04-06]. Dostupné z: <https://devblogs.microsoft.com/aspnet/blazor-webassembly-3-2-0-now-available/>.
- [20] Sapin, S. a contributors. *WeasyPrint - WeasyPrint 51 documentation* [online]. FIT VUT v Brně [cit. 2021-04-03]. Dostupné z: <https://weasyprint.readthedocs.io/en/v51/#:~:text=From%20a%20technical%20point%20of%20view,available%20under%20a%20BSD%20license>.
- [21] Smith, S. *Architecting Modern Web Applications with ASP.NET Core and Microsoft Azure*. 5. vyd. Microsoft Developer Division, .NET, and Visual Studio product teams, prosinec 2020 [cit. 2021-03-20]. Dostupné z: <https://dotnet.microsoft.com/download/e-book/aspnet/pdf>.

## Příloha A

# Obsah přiloženého paměťového média

V následující stromové struktuře je vidět hierarchické uspořádání podstatných složek a souborů v odevzdané práci. Pro přehlednost je vynechána drtivá většina zdrojových souborů.

