FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Webová aplikace pro správu robotů - backend
Student:	Dan Zatloukal
Vedoucí:	Ing. Miroslav Skrbek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Navrhněte a realizujte část webové aplikace pro správu robotů, která je určena pro laboratoř inteligentních vestavných systémů. Zaměřte se zejména na backend, který bude tvořen databází robotů, sadou knihoven v PHP pro zobrazení a editaci dat ve webových stránkách a napojením na výkonné funkce správy robotů jako je např. jejich konfigurace, přenos souborů, aplikací a zobrazování stavu. Na základě rešerše navrhněte vhodné technologie s ohledem na nutnost autorizovaného přístupu ke zdrojům, autentizaci uživatelů a snadné napojení na frontend webové aplikace. Rozsah práce upřesněte po dohodě s vedoucím práce.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 21. listopadu 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Webová aplikace pro správu robotů - backend

Dan Zatloukal

Katedra softwarového inženýrství
Vedoucí práce: Ing. Miroslav Skrbek, Ph.D.

14. května 2018

Poděkování

Prvně bych chtěl poděkovat svému vedoucímu Ing. Miroslavu Skrbkovi, Ph.D. za odborné vedení, veškeré rady a hlavně za jeho čas, který mi během tvorby bakalářské práce věnoval. Dále také děkuji své rodině za podporu během celého studia. V neposlední řadě bych také chtěl poděkovat svému bratrovi Eriku Zatloukalovi za veškeré rady a konzultace.

Bakalářská práce byla spolufinancována Evropskou unií z Operačního programu Výzkum, vývoj a vzdělávání.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla. Souhlasím s tím, aby Dílo bylo veřejně šířeno pod některou z licencí Creative Commons 4.0 ve variantě BY a BY-SA nebo GPLv3.

V Praze dne 14. května 2018 ……

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Zatloukal, Dan. *Webová aplikace pro správu robotů - backend.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato bakalářská práce se zabývá tvorbou jádra webového informačního systému pro správu robotů využívaných v laboratořích inteligentních vestavných systémů. Jádro systému bude tvořeno databází zařízení, sadou knihoven v PHP pro zobrazení a editaci dat ve webových stránkách a rozhraním pro napojení na výkonné funkce správy zařízení jako je např. jejich konfigurace, přenos souborů, aplikací a zobrazování stavu. Práce se nejprve věnuje analýze dostupných technologií a uživatelských požadavků. Poté, na základě této analýzy, se dále práce zaměřuje na výběr technologií a implementaci systému samotného. Zhotovený informační systém usnadní výuku spojenou s roboty a vestavnými systémy využívanými ve výše zmíněných laboratořích. V příloze práce jsou k dispozici zdrojové kódy, vygenerovaná kódová dokumentace, návod pro integraci a také galerie obrázků zhotoveného systému.

Klíčová slova vývoj webové aplikace, vývoj backendu, informační systém, správa robotů, správa vestavných systémů, LIVS FIT ČVUT, usnadnění výuky, aplikace, PHP

Abstract

This bachelor thesis covers the analysis and implementation of a robot management system backend which will be used in laboratories of intelligent embedded systems. Mentioned backend will consist of device database, set of PHP libraries for data display and manipulation in web pages and a interface for executive device functions like remote file upload, remote configuration and status reporting. The thesis firstly deals with the analysis of available technologies and user requirements. Then, based on the previous analysis, the thesis focuses on the choices of technologies used in the project and the actual implementation of the system itself. The finished information system will help with the teaching of subjects which make use of the devices in the mentioned laboratories. You can find the source code, generated code documentation, integration instructions and picture gallery of the finished system in the attachments.

Keywords web application development, backend development, information system, robot management, embedded system management, LIVS FIT ČVUT, facilitate learning, application, PHP

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Informační systémy jsou v dnešní době považovány za nedílnou součást výuky. Sjednocují potřebné informace, a tím ulehčují práci žákům i učitelům. Tyto systémy zvyšují efektivitu práce jak na hodinách, tak při domácí přípravě a jsou v dnešní době již standardem.

Systém pro správu robotů, kterým se zabývá tento projekt, usnadní práci nejen učitelům, ale i studentům využívajícím laboratoře inteligentních vestavných systémů Fakulty informačních technologií ČVUT v Praze. Studentům poskytne přístup k důležitým materiálům týkajících se výuky spojené se zařízeními, která jsou v laboratořích využívána. Učitelům systém poskytne možnost evidence laboratorních zařízení včetně archivace materiálů spojených s jejich výukou. V neposlední řade také systém bude podporovat výkonné funkce správy zařízení, jako je vzdálené nahrávání souborů, či vzdálená konfigurace.

Předměty, které využívají tyto laboratoře nyní postrádají jakýkoliv informační systém, který by práci s výše zmíněnými roboty usnadnil. Tvorbu tohoto systému jsem si vybral právě proto, že jeho absence je v dnešní době poměrně velkým nedostatkem.

Tato práce se přesněji zabývá tvorbou jádra celého systému. Tvorba jádra primárně obsahuje modelování úložných struktur a aplikační databáze, návrh tříd pro práci s těmito úložnými strukturami a zabezpečení uložených dat (autorizace, autentizace). Dále také do tvorby jádra spadá poskytnutí veškerých potřebných funkcí a tříd pro uživatelské rozhraní aplikace.

První část této práce se věnuje analýze problematiky spojené s návrhem a integrací projektu. Toto zahrnuje analýzu uživatelských požadavků, průzkum dostupných technologií a vhodných implementačních metodik. Další kapitola je již věnována návrhu databázových struktur a volbě technologií využitých k realizaci projektu. V poslední řadě se práce zabývá samotnou implementací, což mimo jiné také zahrnuje výběr vhodných knihoven a nástrojů.

Tato práce je úzce spojena s tématem bakalářské práce studenta (FIT ČVUT v Praze) Erika Zatloukala, který tvoří uživatelské rozhraní pro tento

informační systém. Dále je pak volně navázána na téma bakalářské práce studenta (FIT ČVUT v Praze) Petra Koláře, který vyvíjí skripty pro napojení výkoných funkcí robotů.

Vývoj projektu touto bakalářskou prací nekončí a je počítáno s dlouhodobější podporou vytvořeného systému. Součástí práce není tvorba písemné dokumentace, která bude vypracována v dalších fázích vývoje. Také do této práce nespadá analýza externí funkcionality napojené na výsledný informační systém.

Kapitola 1

Cíl práce

Cílem této bakalářské práce je na základě rešerše navrhnout a implementovat jádro informačního systému pro laboratoř inteligentních vestavných systémů.

Cílem rešeršní části práce je analyzovat uživatelské požadavky na informační systém a jeho integraci. Na základě těchto požadavků provést průzkum a porovnání dostupných technologií jako jsou implementační jazyky, vývojové frameworky a případné knihovny. Dílčím cílem je analyzovat vhodné databázové struktury v ohledu na zvolené databázové a implementační technologie. V neposlední řade také v závislosti na uživatelských požadavcích provést analýzu vhodných integračních a implementačních postupů.

Cílem praktické části práce je návrh a realizace jádra informačního systému. Jádro systému bude tvořeno databází zařízení, sadou knihoven v PHP pro zobrazení a editaci dat ve webových stránkách a rozhraním pro napojení na výkonné funkce správy zařízení jako je např. jejich konfigurace, přenos souborů, aplikací a zobrazování stavu. Dílčím cílem je také zajistit bezproblémové napojení jádra aplikace na uživatelské webové rozhraní.

Kapitola 2

Analýza

2.1 Aktuální stav a možná řešení

První část této sekce se zabývá analýzou procesů, které jsou aktuálně zavedeny v předmětech jako je BI-ZIVS nebo MI-IVS. Protože ještě není zhotoven systém, který by řešil problematiku této práce, věnuje se tato sekce primárně analýze existujících řešení, která by se dala případně použít pro základ aplikace. V první části sekce je také uvedeno i pár obdobných systémů jejichž analýza by mohla být přínosná pro tento projekt. Cílem této kapitoly není porovnání systémů a zvolených řešení, jde pouze o informační analýzu potřebnou k následné volbě technologií, kterou se zabývá sekce 3.2.

Analýza současného stavu je důležitou součástí při tvorbě informačních systému. Pomáhá pochopit problematiku zadavatele a tím zvyšuje kvalitu specifikace projektu (co má systém dělat a jakým způsobem to bude dělat). Průzkum již existujících řešení může ušetřit čas vývojářům a peněžní prostředky zadavateli.

2.1.1 Aktuální procesy a stav

Ve výuce předmětů, které využívají laboratoří vestavných systémů je aktuálně přítěží časté vyhledávání a předáváním informaci ohledně jednotlivých zařízení. Současně také učitelé nemají možnost tato zařízení evidovat a kontrolovat jejich stav.

Nahrávání dat do zařízení je prováděno „ručně“. Učitel nebo student je nyní nucen si dohledat adresu zařízení, zapamatovat, nebo si dohledat jaké příkazy má použít a pomocí příkazové řádky danou sekvenci příkazů spustit. Pokud je potřeba nahrát stejný spustitelný soubor do více zařízení, je nutno ho nahrát u každého zařízení zvlášť (přípojit se k zařízení, přihlásit se, přesunout spustitelný soubor a ukončit relaci).

Tyto problémy pomáhá řešit systém pro správu robotů, který je navržen právě pro zjednodušení práce s výše zmiňovanými zařízeními. Jednou z hlav-

ních funkcionalit systému je také evidence a správa školních projektů, které jsou se zařízeními spojené.

Aktuálně je systém primárně vytvářen pro předměty BI-ZIVS (předmět bakalářského programu – základy inteligentních vestavných systémů) a MI-IVS (předmět magisterského programu – inteligentní vestavné systémy). Jedním z využívaných typů zařízení v těchto předmětech je například humanoidní robot SoftBank NAO, jehož operační systém je vlastní linuxová distribuce. Předměty zatím nedisponují velkým počtem zařízení na kterých by se daly nově navržené postupy v praxi otestovat. To se ale má během léta roku 2018 změnit a měl by se inventář laboratoří zvětšit.

V následujících sekcích se práce věnuje analýze již existujících systémů, které řeší obdobné problémy, nebo by se daly využít jako případný základ aplikace.

2.1.2 Systém EDUX a DokuWiki

Jedním z největších informačních systémů, které jsou aktuálně v provozu na Fakultě informačních technologií (FIT) je webový systém EDUX, který je dostupný na adrese https://edux.fit.cvut.cz/. Tento systém je dle [1] napsaný v programovacím jazyce PHP: Hypertext Preprocessor (PHP) a je založený na softwaru DokuWiki. Slouží pro podporu výuky na Fakultě informačních technologií České vysoké učení technické (ČVUT) v Praze. Nabízí sdílenou správu převážně textového obsahu a jeho prezentaci studentům a vyučujícím.

Tím, že je systém postaven na DokuWiki podporuje velké množství možností formátování obsahu stránek. Také spousta základní funkcionality je již implementována přímo v systému DokuWiki a tedy není potřeba ji znovu vytvářet. Následující seznam je založený na [2], [3] a [4] a je neúplným výčtem funkcionalit, které systém EDUX podporuje:

- Tvorba stránek s textovým obsahem a souborovými přílohami.
- Formátování obsahu, včetně pokročilejších funkcí jako je tvorba tabulek či dokonce agregace externích datových kanálů.
- Konverze textového obsahu do prezentace.
- Seskupování stránek do jmenných prostorů.
- Automatické verzování upravovaného obsahu.
- Autorizovaný přístup skrze Access Control List (ACL).
- Autentizace skrze externí služby včetně Shibboleth Single Sign-On (SSO).

2.1.2.1 Úskalí řešení

DokuWiki v základu nabízí velké množství funkcionalit, jednoduchou rozšiřitelnost a má poměrně důkladnou dokumentaci [5]. Nebylo by však moudré volit tento systém pro základ aplikace, aniž bychom se zamysleli nad možnými úskalími.

Jedním z největších problémů s aplikacemi založenými na již existujících řešeních je, že prostudování dokumentace, která je mnohdy poměrně komplexní, může zabrat hodně času. Je tedy nutno myslet na to, že systém, který má podporovat pokročilejší funkcionalitu úzce svázanou s využívanými daty, by nemusel naplno využít možností, které tato řešení nabízí a v závislosti na zvýšené složitosti implementace by ve výsledku pouze vývojáře zpomalil. Popřípadě je potřeba dávat pozor na rozsah potřebných rozšíření, která se mohou stát implementačně náročnějšími, než aplikace tvořená na míru.

DokuWiki je jedním z jednodušších systému, na kterém by tyto problémy neměly být až tak výrazné. Je však také dobré zvážit, zda se vyplatí na takovém systému aplikaci postavit v závislosti na primárním využití. Často je potřeba ve webových systémech řešit hodně doménově specifických problémů. Prezentace textového obsahu tedy nemusí být prioritou a je možné, že se více vyplatí využití obecnějších přístupů.

2.1.3 Systém pro předmět BI-DBS

Dalším, velmi aktuálním výukovým systémem na Fakultě informačních technologií je systém pro předmět BI-DBS (databázové systémy). Tento systém je také realizován v podobě webové aplikace (dostupný z https://dbs.fit.cvut.cz/) a jeho zaměření je v některých směrech podobné jako zaměření systému, kterému se věnuje tato práce. Tím je myšlena hlavně evidence studentských projektů, která je podporována v obou zmiňovaných případech.

Systém pro předmět BI-DBS usnadňuje žákům tvorbu a odevzdání semestrálních prací, které jsou povinnou součástí předmětu. Dále se v systému píší i některé testy, které jsou pak z části vyhodnocovány automaticky.

Podle [6, s. 17-18] je systém napsaný, stejně jako EDUX, v programovacím jazyce PHP. Liší se však poměrně výrazně, a to primárně tím, že není postavený na wiki softwaru, ale na frameworku Nette.

Systém sám je složen z několika modulů, kterým byly věnovány samostatné bakalářské práce. Některé z těchto modulů jsou v podobě čistě front-end komponent, které se starají například o vykreslování grafů. Jiné jsou integrovány přímo do systému. Informace ohledně struktury systému byly převzaty z [6, s. 17-22], [7, s. 22-25] a také z [8, s. 19-26].

2.1.3.1 Úskalí řešení

Systém postavený na aplikačním frameworku má větší volnost ohledně implementačních detailů. Frameworky jsou méně restriktivní než kdyby byl systém

postaven na již existujícím řešení (jako je například WordPress nebo DokuWiki) a rozšířen o dodatečnou funkcionalitu. Vývoj aplikací, které nestaví na komplexnějším softwaru také často bývá v počátečních fázích svižnější. To je hlavně díky tomu, že pro implementaci jednodušších částí není potřeba nastudovat tolik dokumentace. Většina středně velkých až větších systémů tvořených na míru zákazníkům je postavena na nějakém aplikačním frameworku.

Stejně jako u ostatních řešení se ani tento přístup neobejde bez problémů. Těmto problémům se však dá lehce předejít. Jednou z věcí, které je potřeba věnovat pozornost je právě vhodný výběr aplikačního frameworku. Je dobré se vyhýbat již starším a nepoužívaným technologiím. Je také nutné brát v potaz velikost projektu. Pro větší projekty, které mají být dlouhodobě podporovány, je vhodné volit spíše ustálený a stabilní framework, nežli úplnou novinku na trhu (nehledě na její popularitu).

Dalším faktorem je zkušenost vývojového týmu. Pokud je tvořen velký systém založený na aplikačním frameworku nezkušeným týmem, je častým problémem nekvalitní jádro celého systému. Toto dále znemožňuje udržovatelnost celého systému.

2.1.4 Systém Moodle

Dalším systémem, který je v této sekci analyzován, je výukový systém Moodle. Moodle je „*neplacený systém pro správu výuky, který dovoluje učitelům vytvářet soukromé stránky s dynamickými kurzy rozšiřujícími výuku*“ [9] (překlad dle autora). Projekt je open source a distribuovaný pod licencí GNU (verze 3). Moodle je snadno rozšiřitelný a dle [9] a [10] v základu mimo jiné podporuje:

- Velké množství způsobů pro ověření totožnosti jako je například integrace se Single Sign-On systémy (včetně Shibboleth), Lightweight Directory Access Protocol (LDAP) servery nebo externími databázemi.
- Podpora rolí a autorizovaného přístupu.
- Kolaborativní úprava textového obsahu.
- Tvorbu akademických kalendářů.
- Notifikace ohledně úkolů, příspěvků na fóru nebo soukromých zpráv.
- Úpravu vzhledu aplikace.
- Úpravu uživatelských rolí a navazující omezení.
- Podpora připojitelných částí (rozšíření aplikace).

2.1.5 Úskalí řešení

Moodle nabízí spoustu užitečných funkcí a je dostatečně modulární na to, aby se dal rozšířit i o složitější funkcionalitu. Je však, stejně jako bylo zmíněno u systému DokuWiki, nutno dávat pozor na rozsah úprav, které by bylo potřeba provést. V případě potřeby vytvoření velkého množství zásuvných modulů je možné, že by původní výhody tvorby aplikace založené na systému Moodle byly zastíněny zvýšenou obtížností implementace. Také se musí brát v úvahu, že nastudování dokumentace takovýchto systémů může být časově náročné nebo v případě časové tísně může vést k vážné degradaci kvality kódu. V neposlední řadě je také potřeba dávat pozor na copyleft licenci, kterou systém využívá.

2.1.6 WordPress

WordPress je dle [11] volně distribuovaný Content Management System (CMS) pod licencí GNU (verze 2), který uživateli nabízí jednoduchou tvorbu vlastních webových stránek. Tento systém je využívaný pro osobní blogy, firemní stránky či dokonce ve výjimečných případech pro složitější systémy jako jsou internetové obchody.

Systém je modulární a díky tomu, že je jeho jádro minimalisticky pojato, je většina složitější funkcionality dodávána v podobě zásuvných modulů [12]. Skrze tyto moduly lze základní funkcionalitu rozšířit i o velkou škálu autentizačních možností včetně přihlášení skrze Shibboleth SSO.

Veškerá rozšíření v podobě zásuvných modulů jsou také psána v programovacím jazyce PHP a svojí funkcionalitu poskytují aplikaci skrze tzv. „hooky“ (neboli háky). Uživatelské rozhraní je realizováno za pomoci interního systému šablon. Tyto šablony využívají výše zmíněných „hooků“ pro přístup k aplikační funkcionalitě.

Systém nabízí vývojářům komplexní Application Programming Interface (API) ke většině funkcionality obsažené v jádře. Vývoj komplexnějších zásuvných modulů by tedy neměl být problém.

2.1.6.1 Úskalí řešení

Pokud by bylo třeba na systémů zásuvných modulů postavit větší aplikaci, mohlo by to omezit čitelnost a znovupoužitelnost kódu. Vyskytují se tu problémy, které najdeme i u ostatních komplexnějších řešení jako je Moodle (popsané výše). Také je i u tohoto systému potřeba dávat pozor na copyleft licenci.

2.1.7 Drupal

Drupal je „*volně distribuovaný framework pro správu obsahu a lze ho využít k tvorbě široké škály webových aplikací, od jednoduchých webových stránek až po propracované monolity poháněné skrze Application Programming Interface*

(API)“ [13] (překlad dle autora). Software je distribuovaný pod licencí GNU. Drupal je dle [13] napsaný v programovacím jazyce PHP a je velmi odlehčeným systémem, který se zaměřuje hlavně na modularitu pro zjednodušení tvorby aplikací na něm založených.

Dokumentace systému je nejobsáhlejší z výše zmiňovaných řešení. Je již nejspíš jasné, že pro rozšíření základní funkcionality využívá zásuvných modulů. Avšak od běžných CMS systémů se liší v tom, jak zásuvné moduly využívá. Samotný základní systém je sám složen z modulů. Tyto moduly využívají vysokou úroveň abstrakce a tím se stává systém velmi rozšiřitelným [14]. Takovýto typ modulárního přístupu zvyšuje komplexitu celého systému. Dle [15, s. 12] je také Drupal výrazně komplexnější jak výše zmíněný WordPress. Pro úpravu vzhledu je využito šablon psaných v interním formátu [16].

2.1.7.1 Úskalí řešení

Drupal je velice komplexní systém a pro tým programátorů, který s ním nemá moc zkušeností, by mohlo být časově náročné se s ním seznámit. Je také potřeba předem zkontrolovat, zda jsou již hotové moduly pro potřebnou funkcionalitu. (Implementace některých složitějších modulů by mohla trvat déle.) Podle [15, s. 6] je totiž dostupnost modulů u Drupalu výrazně nižší než u systému WordPress. I zde je potřeba brát v úvahu copyleft licenci pro případ, že by výsledná aplikace postavená na tomto systému vyžadovala speciální licenční opatření.

2.2 Analýza uživatelských požadavků

Tato sekce se věnuje analýze uživatelských požadavků kladených na informační systém, kterým se tato práce zabývá. „*Porozumění uživatelským požadavkům je nedílnou součástí návrhu informačních systémů a je kritické pro úspěch interaktivních systémů.*“ [17, s. 1] (překlad dle autora) Sběr těchto požadavků je při vývoji interaktivního systému kritický proto, že pomáhá stanovit jakým způsobem a k čemu bude výsledný systém využíván. Tyto požadavky také vymezují jakou funkcionalitu bude systém podporovat. Požadavky na systém jsou primárně získávány od zákazníka, zadavatele práce nebo z předem vypracovaných zadávacích specifikací.

Sekce se ještě před vymezením uživatelských požadavků věnuje jejich kategorizaci. Sekce tedy nejprve vyjasní jakým způsobem je na extrakci požadavků pohlíženo a teprve poté přejde k vymezení požadavků samotných.

2.2.1 Kategorizace analýzy požadavků

Při analýze uživatelských požadavků je vhodné si požadavky rozdělit do různých kategorií. Rozdělení do kategorií pomáhá vývojářům při implementaci

jednotlivých funkcionalit a usnadňuje extrakci požadavků od zadavatele. Existuje množství již ověřených kategorizací, podle kterých je vhodné se řídit. V tomto projektu je využito modelu FURPS+. Alternativou může být například standard ISO/IEC 9126. Na konci této sekce je pro zajímavost také krátce popsáno jedno z alternativních rozdělení dle [18, s. 35-36].

2.2.1.1 Kategorizace FURPS

Model Functionality, Usability, Reliability, Performance and Supportability (FURPS) byl představen společností Hewlett-Packard a je široce používán pro přehlednou kategorizaci uživatelských požadavků. Model rozděluje požadavky na dvě hlavní kategorie:

- Funkční požadavky: Specifikují akce, které musí systém být schopen provést. Neberou však v potaz fyzická omezení. [19, s. 6]
- Nefunkční (neboli obecné) požadavky: „*Popisují pouze atributy systému nebo systémového prostředí.*“ [19, s. 12] (překlad dle autora)

Nefunkční požadavky dále dělí na:

- Použitelnost (Usability): Udává požadavky například na uživatelské rozhraní nebo dokumentaci.
- Spolehlivost (Reliability): Určuje například jak často bude systém dostupný nebo jak dlouhé a jak časté mohou být systémové výpadky.
- Výkon (Performance): Specifikuje například jak výkoný či efektivní má systém být.
- Podporovatelnost (Supportability): Definuje požadavky například na rozšiřitelnost a udržitelnost aplikačního kódu nebo délku zákaznické podpory produktu.

2.2.1.2 Kategorizace FURPS+

Model FURPS téměř úplně postrádá kategorie pro omezení systémového prostředí. Proto je FURPS často rozšiřován na model FURPS+, který doplňuje následující kategorie nefunkčních požadavků:

- Návrhová omezení (Design constraints): Omezení, která ovlivňují celkový návrh systému. Například nutnost využití relační databáze výrazně omezuje možnosti návrhu.
- Implementační omezení (Implementation constraints): Taková omezení, která specifikují standardy, platformu nebo přímo implementační jazyk.

- Omezení rozhraní (Interface constraints): Omezení, která jsou kladena na interakci a navázání externích systému.
- Fyzická omezení (Physical constraints): Tato omezení se vztahují na hardware, který systém či aplikace využívá

2.2.1.3 Ukázka alternativní kategorizace

Dle [18, s. 35] je možné informace získané od zákazníka dělit do dvou hlavních kategorií. Těmito kategoriemi jsou „omezení“ a „požadavky“ a jsou popsány v následujících podsekcích. Ve výsledném systému je potřeba splnit jak požadavky, tak omezení.

Omezení určují limitující faktory, jako jsou technologická omezení (většinou v závislosti na prostředí ve kterém je aplikace provozována), omezení funkcionality (například restriktivním rozhraním jiné napojené aplikace) nebo také mohou specifikovat vyžádané vývojové postupy. Omezení tedy zužují možnosti vývojového týmu. Tato sekce byla parafrázována z [18, s. 35].

„*Požadavky jsou primárním zaměřením vývojového procesu, neboť hlavní účel tohoto procesu je převést tyto požadavky na systémové návrhy.*“ [18, s. 35] (překlad dle autora) To znamená, že určují co bude systém podporovat a jakým způsobem bude daná funkcionalita využívána. Požadavky jsou dále rozděleny na uživatelské, operační, funkční, výkonnostní, návrhové, odvozené a alokační [18, s. 35-36]. Toto rozdělení je poměrně komplexní a nejspíš by bylo vhodnější pro projekty většího rozsahu.

2.2.2 Uživatelské požadavky

Požadavky na systém byly získávány přímo od zadavatele projektu. V době kdy se na systému začalo pracovat, měl zadavatel základní vizi ohledně toho, co by systém měl podporovat. Hlavní požadavky na systém byly zadány již z počátku vývoje a byly postupně adjustovány a rozšiřovány. Dosavadní požadavky jsou k nalezení níže a zachycují funkcionalitu i omezení systému. **Poznámka**: Níže specifikovaná funkcionalita nemusí být všechna předmětem této práce. Jde o kompletní přehled dosavadních požadavků na systém, které budou doplňovány i po zakončení této práce.

2.2.2.1 Implementační omezení

Jak již bylo zmíněno výše je v pro analýzu uživatelských požadavků využito rozdělení FURPS+. V závislosti na tomto rozdělení se tato sekce věnuje požadavkům spadajícím do kategorie implementačních omezení.

N1 – Nasazení na zadavatelem zvolený server: Systém musí být provozuschopný na serveru s linuxovou distribucí FreeBSD. Je třeba počítat s tím, že zvolený server nemusí podporovat nejnovější verze zvolených technologií.

2.2.3 Návrhová omezení

N2 – Databáze nevázaná na implementaci: Databáze musí být navržena s ohledem na to, že může být využita i jiným systémem, který nemusí být psaný v jazyce PHP. Je tedy nutné návrh databáze vytvořit implementačně nezávislý.

2.2.3.1 Požadavky na podporovatelnost

N3 – Systém obdrží dlouhodobější podporu: Systém bude vývojářským týmem podporován delší dobu a bude dodáván ve funkčních celcích. Doba podpory systému bude úměrná tomu, kolik funkcionality je potřeba dodat a otestovat vzhledem k základním funkčním požadavkům na systém. Požadavek na dodání funkčních celků znamená, že za produkční verzi nebude považován systém, kde jsou uživatelsky přístupné nefunkční části systému.
Vysvětlení: Vývoj systému nekončí touto prací a bude dále rozvíjen a dokumentován aby se případně dal rozšířit o další funkcionalitu až ho původní vývojářský tým opustí.

N4 – Kód bude vhodně dokumentován: Systém bude obsahovat kódovou dokumentaci na místech, kde je jí nutně třeba. Pokud systém využije složitější funkcionality, bude zvlášť zdokumentována strukturovaným textem. Tato dokumentace usnadní budoucí rozšiřitelnost systému. Dodaný systém bude obsahovat návod na zprovoznění.
Poznámka: Externí textová dokumentace bude dodána až po otestování stabilní verze v praxi a není součástí této práce.

2.2.3.2 Požadavky na použitelnost

N5 – Systém bude dostupný skrze webového rozhraní: Webové rozhraní bude vhodně navrženo s ohledem na cílovou skupinu a využití systému. Rozhraní bude dávat přednost zobrazení dat před rozptylujícími prvky.
Poznámka: Vhodný vzhled je pro rozhraní webové aplikace velice důležitý, neboť pomáhá uživateli s orientací v systému a tím šetři jeho čas.

N6 – Responzivní webové rozhraní: Webové rozhraní se musí přizpůsobit velikosti obrazovky a rozlišení. Rozhraní by mělo být stále použitelné i při využití zařízení s menším rozlišením.
Poznámka: Tento požadavek nebyl explicitně zadán zadavatelem, ale je odvozen z požadavku na vhodné řešení webového rozhraní, neboť velké množství studentů často využívá ke své práci jako pomůcku mobilní telefon.

2.2.4 funkční požadavky

Tato část obsahuje nejdůležitější funkční požadavky kladené na informační systém, kterým se tato práce zabývá.

F1 – Evidence laboratorních zařízení: Systém bude ukládat informace ohledně zařízení využívaných v laboratořích inteligentních vestavných systémů. Dále budou tyto informace zobrazitelné skrze webové rozhraní. Minimálně bude systém obsahovat informace potřebné k napojení externí funkcionality dodávané skrze bakalářský projekt studenta (Fakulta informačních technologií (FIT) ČVUT v Praze) Petra Koláře. Kromě zařízení samotných bude také systém evidovat modely a typy zařízení, které jsou v laboratořích využívány.

Priorita Vysoká

Složitost Nízká

Poznámka Jaké informace je potřeba uložit, bylo primárně konzultováno s autorem externí funkcionality Petrem Kolářem. Informace uložené v databázi, kromě toho že jsou potřebné pro napojení externí funkcionality, pomohou uživatelům systému například se získáním adres potřebných pro nahrání spustitelných souborů do vybraného zařízení.

F2 – Evidence uživatelských projektů: Systém bude obsahovat evidenci uživatelských projektů. Projekty se budou dělit na veřejné a privátní. Veřejné projekty budou dostupné i pro nepřihlášeného uživatele. Projekty budou verzovány a budou obsahovat vícejazyčné hlavičky. Každý projekt bude obsahovat konfiguraci, na kterou budou napojeny softwarové komponenty, které jsou v projektu využívány. Projekty však nebudou podporovat odevzdávání řešení či jiných příloh, které nejsou spojeny se samotným zadáním projektu.

Priorita Střední

Složitost Střední

F3 – Autorizace a autentizace: Systém bude podporovat přihlášení skrze službu Shibboleth SSO a bude mapovat získaná metadata na interní uživatelské záznamy v databázi. Po systému není vyžadováno napojení na server, který eviduje uživatelské účty pro systém Shibboleth. Informační systém bude podporovat autorizaci přístupu k zabezpečeným zdrojům. Zabezpečenými zdroji budou projekty, webové stránky, pravidla přístupu samotná a spustitelné funkce na zařízeních. Každý zabezpečený zdroj bude mít základní pravidla read, create, edit, delete a k tomu ještě další specifická zabezpečení. Specifická zabezpečení pro projekty jsou read-header a clone. Jediný zabezpečený zdroj, který nebude obsahovat základní práva, ale pouze právo *can execute*,

je spustitelná funkce na zařízení. Administrátor nebo pověřená osoba musí být schopná přidělovat či odebírat práva. Autorizační systém musí být snadno upravitelný pro případ, že bude nutné přidat další zabezpečené zdroje.

Priorita Vysoká

Složitost Vysoká

Poznámka Zabezpečení dat a autorizovaný přístup je jedna ze složitějších částí celé aplikace a tato práce se jí poměrně důkladně zabývá v kapitole 4.

F4 – Napojení na externí funkcionalitu realizující výkonné funkce zařízení: Systém bude podporovat napojení na externí funkcionalitu realizující výkonné funkce zařízení jako je jejich konfigurace, přenos souborů, aplikací a zobrazování stavu. Je potřeba evidovat veškeré potřebné informace pro spuštění těchto výkonných funkcí. Potřebná funkcionalita bude dodávána skrze shell skripty, které jsou součástí projektu studenta (FIT ČVUT v Praze) Petra Koláře. Účelem je zprostředkovat rozhraní pro tyto výkonné funkce.

Priorita Střední

Složitost Nízká–střední

F5 – Tvora webových stránek: Systém bude podporovat tvorbu jednoduchých webových stránek s případným rozšířením o podstránky. Tyto stránky musejí být dostupné skrze aplikaci a měly by podporovat editaci za pomoci Hypertext Markup Language (HTML) značek s případným rozšířením o komplexnější editor.

Priorita Nízká

Složitost Nízká

2.3 Analýza případů užití

Tato sekce se věnuje analýze případů užití. Případy užití popisují procesy, které jsou za pomoci systému prováděny. Pomáhají vývojovému týmu pochopit jakým způsobem se systém bude využívat. Na konci této sekce je k nalezení tabulka zobrazující realizace funkčních požadavků jednotlivými případy užití.

2.3.1 Případy užití

U každého případu užití je uveden minimálně název, popis a hlavní aktér. Hlavním aktérem je osoba, role či skupina uživatelů, která bude daný případ užití primárně provádět. U případů, u kterých není z popisu úplně jasné jakým způsobem budou přesně prováděny, jsou také uvedeny detailnější scénáře. Díky

velikosti vývojového týmu není potřeba důkladně analyzovat všechny případy užití a jsou zde tedy uvedeny pouze ty nejdůležitější.

Často se v textových popisech analyzovaných případů vyskytuje „aktér“ zvaný „pověřený uživatel“ či „autorizovaný uživatel“. Toto je právě proto, že v tomto systému může každý uživatel získat práva na provedení určité akce. Tento efekt je potřeba při modelování případů užití minimalizovat, neboť obecný uživatel by byl aktérem ve všech případech užití (takové případy užití by nebyly velkým přínosem).

Řešení V textové reprezentaci jsou případy uvedeny v obecném znění (aktér je většinou „autorizovaný uživatel“). Tímto je zdůrazněno, že případy nemají pevně definované typy uživatelů, které je budou využívat. Pro přehlednost byl do textové podoby přidán také hlavní aktér, který bude daný případ využívat nejčastěji. Dále na konci sekce je k nalezení UML diagram, kde je využito hlavních aktérů pro grafické vyobrazení toho, kdo bude nejčastěji daný případ využívat. Toto je jediný způsob, kterým se dá rozumně využít UML notace a přehledně vyobrazit kdo bude dané operace v systému provádět nejčastěji. Případy užití s jediným aktérem by nepomohly ujasnit jak se systém bude využívat.

2.3.1.1 Seznam účastníků

Pro lepší orientaci je v této sekci také uveden seznam nejčastějších účastníků, kteří budou systém využívat. Někteří z níže uvedených účastníků/aktérů se v následujících případech užití nevyskytují. Je tomu tak protože seznam případů užití není vyčerpávající.
Následuje seznam účastníků včetně očekávaných rolí, které budou v systému zastávat (role nejsou pevně dané):

- Administrátor: spravuje práva pro přístup k zabezpečeným zdrojům a také má neomezený přístup k veškerým informacím, které systém eviduje.
- Zaměstnanec: eviduje zařízení a připravuje materiály.
- Učitel: eviduje zařízení, materiály a projekty.
- Student: využívá informací evidovaných u zařízení a projektů.
- Nepřihlášený uživatel: má přístup k veřejným projektům v systému.

2.3.1.2 UC1 – Evidence nového zařízení

Popis: Umožňuje učiteli či jinému pověřenému uživateli vložit nově obdržené zařízení do systému. Tato akce se dá provést z webového rozhraní buď přímo

dohledáním a zadáním všech potřebných údajů pro uložení zařízení nebo spuštěním externí funkcionality, která připojené zařízení automaticky zaregistruje.
Hlavní scénář: Dohledání informací a zadání potřebných údajů do systému.

1. Uživatel obdrží nové zařízení, nastaví a ručně dohledá veškeré potřebné informace ke vložení daného modelu.
2. Systém zobrazí uživateli podporované typy zařízení a u každého nabídne možnost vytvořit nový záznam.
3. Uživatel zvolí příslušný typ zařízení a požádá systém o jeho vytvoření.
4. Pokud uživatel nemá oprávnění k vytvoření nového záznamu:
   a) Systém zobrazí chybovou hlášku a případ končí.
5. Jinak:
   a) Systém nabídne formulář pro vytvoření nového zařízení vybraného typu, který obsahuje název zařízení, model a další specifická data pro daný typ jako je například adresa IP.
   b) Uživatel zadá název a model zařízení a případně vyplní další typově specifické povinné údaje a potvrdí uložení.
   c) Pokud nebyly některé z údajů zadány, zobrazí systém chybové hlášení. Jinak systém zpracuje vložená data a uloží je do databáze.

Alternativní scénář: Dohledání adresy zařízení a využití inicializační funkcionality.

1. Uživatel obdrží nové zařízení, připojí ho k síti, která je dostupná z informačního systému a dohledá si adresu přístroje.
2. Systém uživateli zobrazí dostupné výkonné funkce zařízení, které lze spustit.
3. Uživatel zvolí funkci „inicializace nového přístroje“.
4. Pokud uživatel nemá oprávnění ke spuštění této funkce:
   a) Systém zobrazí chybovou hlášku a případ končí.
5. Jinak:
   a) Systém zobrazí formulář pro inicializaci nového přístroje, který obsahuje adresu zařízení a nové jméno, pod kterým bude zařízení uloženo.

b) Systém spustí vybranou funkci a zpracuje výstup. Pokud vše proběhlo v pořádku, vráti uživateli hlášku ohledně provedených akcí. Pokud nastala chyba při spouštění funkce, vráti systém chybovou hlášku.

Hlavní aktér: Učitel

2.3.1.3 UC2 – Získání informací ohledně vybraného zařízení

Předpoklady: Uživatel vlastní příslušná práva pro zobrazení informací o daném zařízení a zná alespoň jeden z jeho identifikátorů.
Popis: Umožňuje uživateli snadno dohledat informace o vybraném zařízení za využití evidenčního systému. Uživatel může v systému skrze evidenční seznam dohledat informace potřebné pro připojení k jím vybranému zařízení. Dále uživatel může v závislosti na typu a modelu zařízení dohledat informace jako je využívaný mikroprocesor, programátor, identifikátor výrobku, IP adresa nebo fyzická MAC adresa, které potřebuje k práci s daným zařízením.
Hlavní aktér: Student

2.3.1.4 UC3 – Úprava existujícího zařízení

Předpoklady: V systému je evidováno zařízení, uživatel vlastní příslušná práva a chce upravit evidovaná data.
Popis: Umožňuje uživateli upravovat evidované údaje o zařízení včetně typově specifických informací jako je IP adresa nebo číslo výrobku.
Hlavní aktér: Učitel

2.3.1.5 UC4 – Evidence nového projektu

Předpoklady: Uživatel vlastní základní práva pro vytvoření projektu.
Popis: Umožňuje uživateli vytvořit nový projekt. Při tvorbě projektu si uživatel může zvolit zda má být projekt veřejný, či ne. Pokud uživatel vlastní administrátorská práva, může vytvořit projekt i pro jiné uživatele než je on sám.
Hlavní aktér: Učitel

2.3.1.6 UC5 – Úprava existujícího projektu

Popis: Umožňuje autorovi projektu či pověřenému uživateli upravit údaje existujícího projektu. Pří úpravě může uživatel připojit k projektu využitá zařízení, programové komponenty, programové obrazy a samostatné souborové přílohy. Také umožňuje autorizovanému uživateli vytvořit novou verzi projektu (stará verze se automaticky archivuje, pokud není explicitně smazána).
Hlavní aktér: Učitel

2.3.1.7 UC6 – Duplikace projektu

Předpoklady: V systému existuje projekt, který je potřeba duplikovat. Učitel či pověřená osoba vlastní potřebná práva na duplikaci.
Popis: Umožňuje učiteli nebo jinému pověřenému uživateli duplikovat existující projekt. Při duplikaci projektu může učitel nebo pověřená osoba zvolit nového autora projektu a tím efektivně přiřadit již existující projekt novému uživateli.
Hlavní aktér: Učitel

2.3.1.8 UC7 – Povolení přístupu na zabezpečený zdroj

Popis: Umožní administrátorovi nebo pověřenému uživateli přidělovat či odebírat práva na zabezpečené zdroje jako jsou projekty, webové stránky, přístupová pravidla samotná, výkonné funkce zařízení a uživatele. Administrátor může povolit, či odebrat přístup přímo určitému uživateli, skupině uživatelů nebo roli. Práva také mohou být udělena přímo na určitý zabezpečený zdroj nebo na všechny zabezpečené zdroje daného typu. Administrátor nebo pověřený uživatel má možnost odebírat již vytvořená práva
Hlavní aktér: Administrátor

2.3.1.9 UC8 – Zamezení přístupu do aplikace

Popis: Umožňuje administrátorovi zamezit přihlášení do aplikace skrze úpravu uživatelských údajů a změnu aktivního stavu uživatele na neaktivní.
Hlavní aktér: Administrátor

2.3.1.10 UC9 – Spuštění nebo přesunutí souboru na zařízení

Předpoklady: Zařízení, na kterém bude akce prováděna, je evidováno v systému a je připojeno k síti. Spustitelný soubor je nahraný v systému jako programová komponenta, a uživatel má potřebná práva pro provedení akce.
Popis: Umožňuje uživateli přesunout či spustit soubor na vybraném zařízení. Uživateli stačí si v systému vybrat soubor, který chce přesunout nebo spustit, zvolit cílový přístroj a spustit korespondující akci.
Hlavní aktér: Uživatel

2.3.1.11 UC10 - Vytvoření nové webové stránky

Popis: Umožní učiteli nebo jinému pověřenému uživateli vytvořit novou webovou stránku. V této stránce může při tvorbě vyplnit titulek a obsah stránky, který může vytvořit za využití HTML značek.
Hlavní aktér: Učitel

2.3.2 Realizace funkčních požadavků

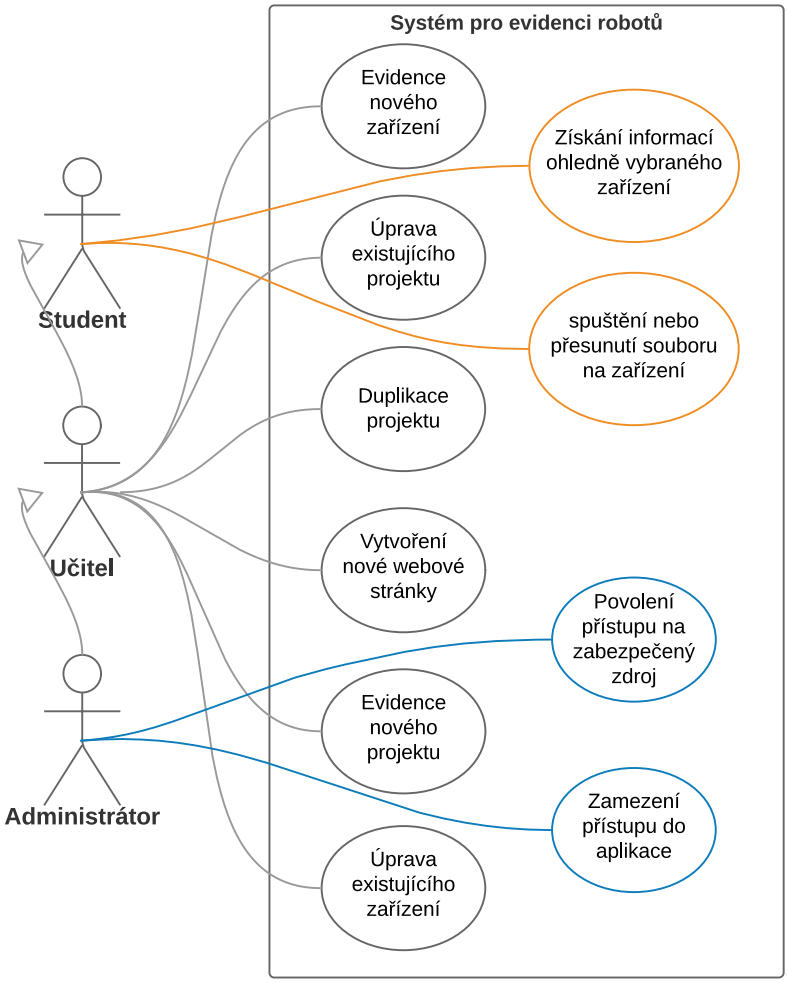
V tabulce 2.1 je zobrazen vztah realizace uživatelských požadavků jednotlivými případy užití.

	Případy užití									
požadavky	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10
F1	✓	✓	✓							
F2				✓	✓	✓				
F3							✓	✓		
F4	✓								✓	
F5										✓

Tabulka 2.1: Realizace uživatelských požadavků

2.3.3 Diagram případů užití

Pro přehlednost jsou na diagramu 2.1 zobrazeny výše popsané případy užití společně s hlavními aktéry. Diagram není kvůli požadavku na dynamická přístupová práva modelován podle běžných standardů, je tedy doporučeno přečíst si paragraf „řešení“ v sekci 2.3 na straně 15, který vysvětluje jakým způsobem jsou případy užití modelovány. Záměrně je vynechán aktér „zaměstnanec“ neboť tato role by se měla v systému objevovat poměrně zřídka a učitel je v roli nadřazené, která bude dané akce provádět mnohem častěji.



Obrázek 2.1: Diagram případů užití

2.4 Doménový model

Tato sekce se zabývá tvorbou doménového modelu pro celý systém evidence robotů. „*Tvorba doménového modelu je způsob, jakým lze popsat entity reálného světa a relace mezi nimi, což kolektivně popisuje problémový doménový prostor.*“ [20] (překlad dle autora) Doménový model tedy popisuje entity, vztahy mezi entitami a data, která tyto entity obsahují. Pomáhá pochopit jak jejich význam, tak kontext, ve kterém jsou data využívána. Tento model je také základem pro tvorbu specifičtějšího databázového modelu.

Doménový model je rozdělen do několika logických celků, které jsou vzájemně provázány. Tyto logické celky seskupují související data z každé části systému. V tomto modelu nejsou uváděna implementačně specifická data (ani data, která lze odvodit z jiných dat) nebo entity. Finální model má pomoci pochopit s jakými daty zákazník pracuje nebo bude pracovat při provozu systému.

Také je dobré všimnout si, že v doménovém modelu úplně chybí entity obsahující data ohledně řízení přístupu k zabezpečeným zdrojům. Tyto datové struktury jsou záměrně vynechány. Hlavním důvodem je, že jsou to implementační záležitosti a aktuálně s nimi zákazník nepracuje. Těmto strukturám se důkladněji věnuje sekce 3.4 a 4.2.

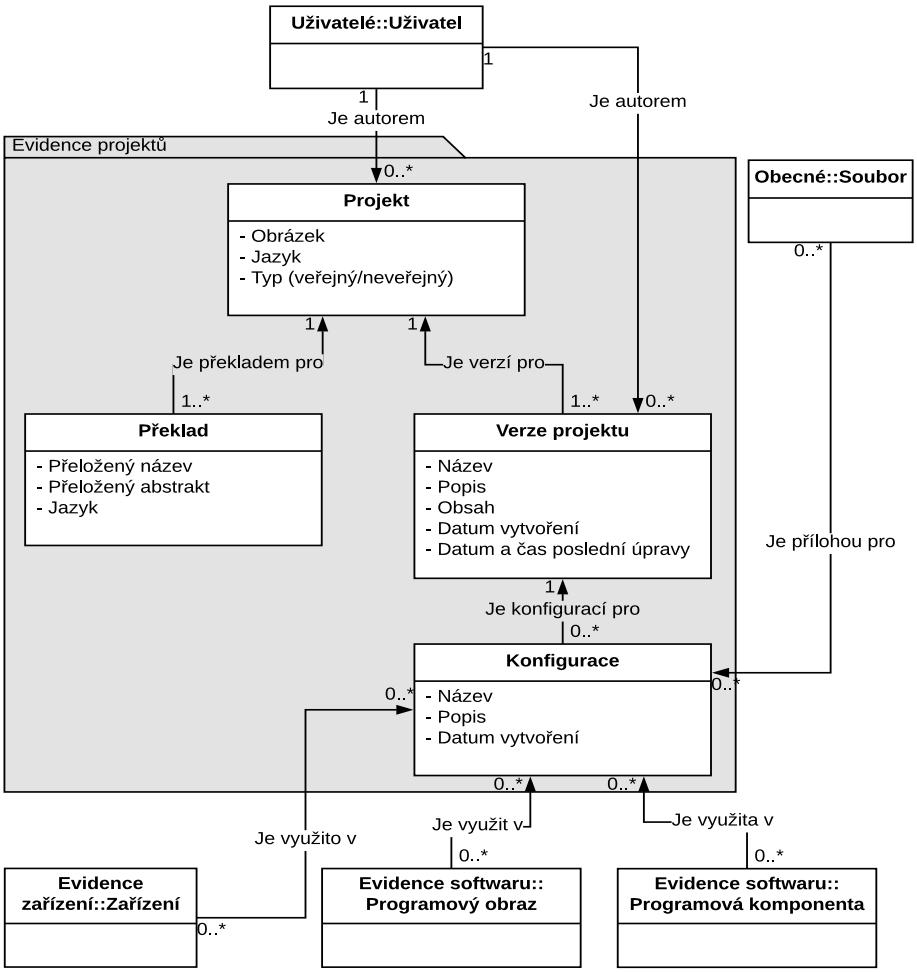
2.4.1 Kompletní doménový model

Doménový model pro celý systém je poměrně rozsáhlý, ale obsahuje velké množství jednoduchých entit a relací, které není potřeba dopodrobna probírat. Proto se následující část zaměřuje na analýzu pouze zajímavějších částí problémové domény. Kompletní doménový model pro systém evidence robotů je k nalezení na přiloženém CD ve složce *visual_attachments/diagrams*.

2.4.2 Evidence projektů

Na diagramu 2.2 je zobrazena část doménového modelu, která popisuje entity a relace úzce spojené s evidencí projektů. Jak je vidět na diagramu, projekty mají vícejazyčné hlavičky a mohou být verzovány. Překlad hlavičky je určený primárně pro nepřihlášené uživatele, kteří se chtějí dozvědět jaké projekty jsou v předmětech (které systém využívají) vytvářeny. Je tedy potřeba evidovat zda-li je projekt veřejný či ne, neboť veřejné projekty jsou dostupné právě i pro nepřihlášeného uživatele. Kromě autora není u verzí ani u projektu evidován uživatel, který provedl poslední změny.

Autor Každý projekt je svázán pouze s jedním autorem, který však může zastávat také roli vlastníka projektu (nemusí být nutně autorem obsahu). Například pokud na projektu pracuje více uživatelů, a tito uživatelé vlastní práva na jeho úpravu, je na autora pohlíženo jako na vlastníka projektu.



Obrázek 2.2: Doménový model evidence projektů

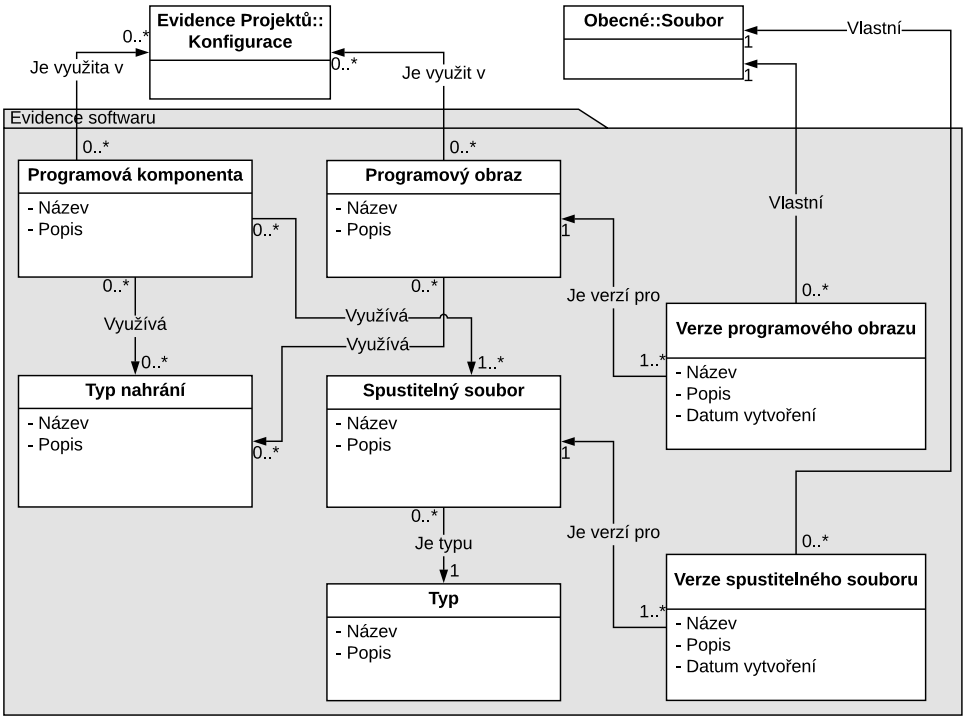
Verze Obsah projektu je uložen ve verzích. Projekt může vlastnit více verzí a tím umožní uživateli archivovat jeho obsah. Tyto verze musejí být datovány aby šlo rozlišit nejnovější verzi. Datování je také vhodné pro orientaci v archívu starších verzí. Každá verze má také svého autora (podobné jako u projektu samotného).

Překlad Každý projekt musí obsahovat překlad hlavičky a krátkého popisu, který je v modelu označován jako „abstrakt“. Projekt musí vlastnit alespoň jeden překlad ve svém jazyce (na tento překlad je v této práci odkazováno jako na „primární překlad“). Záměrně jsou vynechány veškeré generované záznamy (spíše implementační nebo návrhová záležitost).

Konfigurace Konfigurace projektu představuje určité nastavení pro připojená zařízení. V konfiguracích je evidováno datum vytvoření pro dohledání nejnovější konfigurace. Dále konfigurace evidují využívaná zařízení, programové obrazy a programové komponenty. Tyto záznamy určují jaké komponenty a obrazy jsou zařízeními v projektu využívány. Konfigurace může využívat mnoho různých typů zařízení, což uživateli nabízí velkou flexibilitu při tvorbě projektů. V poslední řadě je možné ke konfiguraci připojit také souborové přílohy (je vhodné například pro pdf dokumentaci nebo návody).

2.4.3 Evidence softwaru

Část doménového modelu popisující evidenci softwaru je k nalezení na diagramu 2.3. V systému jsou evidovány dvě hlavní kategorie softwarových celků. První kategorií jsou programové komponenty a druhou jsou programové obrazy. Je možné si všimnout, že většina entit obsahuje pouze název a popis. Toto je efekt, který vzniká při dekompozici datových struktur na menší části. Název většinou hraje roli identifikátoru pro uživatele systému. Popis může být jakýkoliv text vztažený k dané entitě (například návod k použití).



Obrázek 2.3: Doménový model evidence softwaru

Programová komponenta Reprezentuje aplikace, či programové celky. Programovým celkem je myšlen například soubor Shell skriptů, který nabízí určitou funkcionalitu. Každá programová komponenta může obsahovat více spustitelných souborů (některé nástroje či programy mohou obsahovat více spustitelných souborů).

Spustitelné soubory Samotný spustitelný soubor je uložen ve verzích. Je tedy možné evidovat starší verze spustitelného souboru. Toto se může hodit například při nekompatibilitě projektu s novější verzí. Každá verze eviduje také datum vytvoření, což umožní dohledání nejnovější verzi pro každý spustitelný soubor.

Programový obraz Programový obraz je podobný programové komponentě, ale reprezentuje spíše nedělitelný celek, jako je například záloha systému zařízení. Obrazy jsou také verzovány, ale nemohou mít více než jeden soubor.

2.4.4 Evidence zařízení

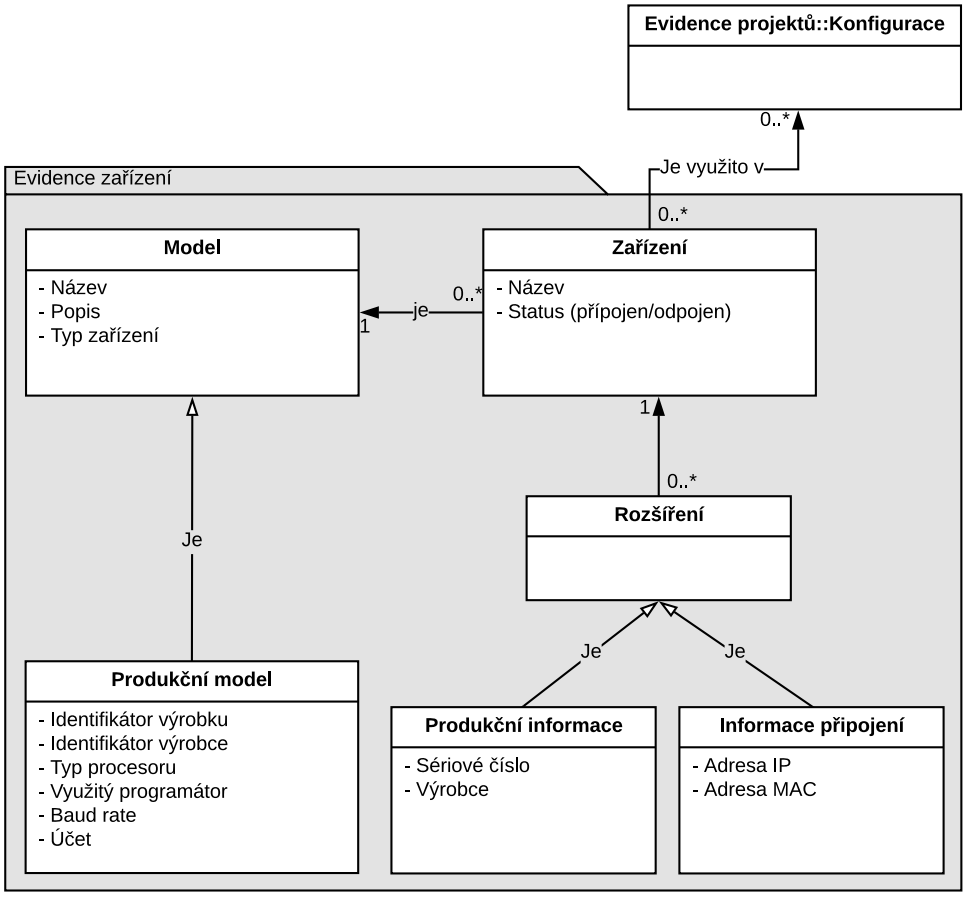
Samotná zařízení jsou poměrně přímočará. Čeho je dobré si všimnout, je dědičnost u vlastností zařízení a modelu zařízení. Část doménového modelu pro evidenci zařízení je zobrazena na diagramu 2.4.

Rozšíření Zařízení může být rozšířeno o dodatečná data jako je sériové číslo nebo adresa IP. Tato rozšíření poskytují flexibilitu ohledně toho, co bude u zařízení uchováváno za data. V tomto stavu je možné evidovat data, která jsou nyní uložena v rozšířeních přímo v zařízení. Avšak s přibývající funkcionalitou, či připojením dalších modulů by se počet evidovaných dat mohl zvyšovat. Proto je raději zvolen komplexnější, ale flexibilnější přístup.

Model Zařízení jsou určitého modelu (model může být například Arduino UNO). Každý model je pouze jednoho typu a tento typ nelze rozšiřovat. Model také eviduje typ zařízení, který se od typu modelu výrazně liší. Typ modelu určuje jaké informace jsou v modelu evidovány, zatímco typ zařízení určuje zda-li se jedná o zařízení Arduino, Robot nebo jiné. Typ modelu byl přidán pro případ, že by bylo potřeba evidovat jiná zařízení než „produkční“ (například počítačové servery).

Produkční model je základním typem modelu a předpokládá se, že převážná většina modelů bude právě tohoto typu. Obsahuje informace, které jsou potřeba pro práci se zařízeními využívanými v laboratořích inteligentních vestavných systémů.

Nejasnými položkami by mohly být bad rate a účet. Baud rate je využíván při vzdáleném připojení k zařízení a určuje rychlost spojení. Účet je evidován také kvůli vzdálenému připojení na zařízení, neboť udává jméno účtu, ke kterému je potřeba se přihlásit na daném zařízení.



Obrázek 2.4: Doménový model evidence zařízení

2.4.5 Uživatelé

V poslední řade je také dobré zmínit uživatele. Tato sekce není moc rozvedena, neboť neobsahuje žádné složitější nebo nestandardní struktury. Zprvu je dobré zmínit, že nejsou evidovány samostatné uživatelské účty a každý uživatel může tedy vlastnit pouze jeden účet. Dle uživatelských požadavků je evidován pouze status, jméno, příjmení a email. Status určuje zda-li je uživatel aktivní nebo ne. Tento status je primárně určen pro administrativní účely jako je zamezení přístupu do aplikace. Záměrně je vynechán záznam *valid to*, který je, jak je popsáno v sekci 3.4, využit pro obnovu starších externě mapovaných účtů (datově nevýrazný, avšak implementačně důležitý).

Role Každý uživatel může mít více rolí. Role mohou mít další podřazené role a to umožní sestavit hierarchickou strukturu. Pokud uživatel vlastní roli

nadřazenou, obdrží minimálně stejná oprávnění jako veškeré role podřazené.

Skupina Uživatele lze řadit do skupin. Tyto skupiny reprezentují primárně vyučované předměty, které mají systém využívat. Doménový model by se obešel i bez uživatelských skupin, je však vhodnější role a skupiny oddělit už jen z orientačního hlediska.

2.5 Technologie

Volba správných technologií je jedním z klíčů k úspěchu jakéhokoliv softwarového projektu. Proto je potřeba důkladně analyzovat dostupné technologie aby byly zvoleny ty, které nejlépe odpovídají povaze projektu. Tato sekce se zabývá právě touto problematikou. V první části jsou krátce analyzovány technologie využívané pro tvorbu uživatelského rozhraní (nebo také „client side“, či front-end technologie) a teprve poté sekce pokračuje na analýzu technologií pro realizaci serverové části aplikace (tzv. back-end technologie). Ke každé analyzované technologii jsou vybrány dva až tři nejpoužívanější frameworky, které jsou také krátce popsány.

2.5.1 Technologie pro tvorbu uživatelského rozhraní

Tato sekce se věnuje technologiím využívaným pro tvorbu front-end části aplikací. Základem prezentace obsahu na webu je jazyk popisující jakým způsobem se mají data zobrazovat. Dvěma takovými nejznámějšími jazyky jsou HTML a Extensible Hypertext Markup Language (XHTML), které jsou popsány níže.

Tyto jazyky ale nenabízí velké množství způsobů jak stylizovat prezentaci popisovaných dat. Proto se pro stylizaci používají externí Kaskádové styly (CSS), které umožňují přesněji popsat způsob prezentace dat. Zároveň také vznikly tzv. preprocessory, které přidávají funkcionalitu a vylepšují syntaxi těchto Kaskádových stylů. Za využití těchto technologií byly pro rychlou a jednoduchou stylizaci uživatelského rozhraní vytvořeny designové frameworky jako je Twitter Bootstrap.

Nedílnou součástí moderních webových stránek je dynamické uživatelské rozhraní, které reaguje na uživatelský vstup okamžitě. Takováto dynamická uživatelská rozhraní jsou realizována za pomoci programovacích jazyků určených pro běh přímo v prohlížeči. V dnešní době je daleko nejvyužívanější technologií pro tvorbu dynamických uživatelských rozhraní javascript, nebo případně jazyky, které lze na javascript překompilovat [21]. Existuje také velké množství front-end frameworků pro tvorbu takovýchto rozhraní.

2.5.1.1 HTML nebo XHTML

XHTML je téměř identické s HTML4, ale využívá eXtensible Markup Language (XML) elementů, které rozšiřují možnosti původního HTML. Toto roz-

šíření také řešilo mnoho problémů s tehdejší kompatibilitou mezi prohlížeči. Mimo jiné má XHTML také striktnější syntaxi (kvůli základu na XML). S příchodem HTML5 byla však většina těchto problémů, které XHTML adresovalo, vyřešena a tím se XHTML stalo téměř zastaralým. [22]

2.5.2 Technologie pro serverovou část

Tato sekce se věnuje nejrelevantnějším a nejpopulárnějším back-end technologiím pro tvorbu webových aplikací. Šesti nejpopulárnějšími programovacími jazyky v tomto odvětví jsou (neseřazené):

- Java,
- JavaScript,
- Python,
- C#,
- C++,
- PHP

Informace o popularitě a relevantnosti byli získány z [23], [24] a také z [25].

2.5.2.1 Java

Java dává velký důraz na Object-oriented programming (OOP) a OOP principy. Je perfektní alternativou k jazyku C++, který je mnohem náročnější na naučení. Velkou výhodou je, že je tento jazyk platformě nezávislý, neboť aplikace běží ve virtualizovaném prostředí. [23]

Dle [23] 90 % společností, které se umístily v žebříčku Fortune 500 (500 nejvýdělečnějších firem, k nalezení na: http://fortune.com/fortune500/) využívají programovací jazyk Java pro své back-end systémy a aplikace. Každá verze tohoto jazyka je také zpětně kompatibilní, což je výhodné při budoucím rozšiřováním aplikace. Výhodou je také velká komunita, která zaručuje nemalé množství materiálů ze kterých lze čerpat zkušenosti. [25]

Enterprise edice programovacího jazyka Java nabízí velkou škálu technologií za pomoci kterých lze vytvářet rozsáhlé strukturované webové aplikace. Vzniklo však nemalé množství frameworků, které se snaží usnadnit vývoj webových aplikací. Následující seznam obsahuje několik vybraných, dle [26] nejpopulárnějších, webových frameworků:

- Spring
- Google Web Toolkit (GWT)
- Vaadin

- Play!

Velkou nevýhodou aplikací postavených na programovacím jazyce Java je složitost jejich implementace. Java nabízí obrovské množství funkcionality a i komunitou tvořené frameworky jsou velice rozsáhlé. Nastudování dokumentace tedy zabere velké množství času, což nemusí být optimální pro menší projekty nebo projekty, které musí být rychle zhotoveny.

2.5.2.2 JavaScript

„*JavaScript je multiplatformní, objektově orientovaný skriptovací jazyk využíván k tvorbě interaktivních webových stránek.*“ [27] (překlad dle autora). V posledních letech se JavaScript prosadil, díky technologii Node.js, i na straně serverových technologií. JavaScript se, v sekci nejpopulárnějších technologií průzkumu [24] z roku 2017, umístil na prvním místě s 62.5 %. Je však potřeba brát v úvahu, že tento programovací jazyk je v dnešní době téměř nezbytnou součástí uživatelských rozhraní a je tedy využíván dvěma různými způsoby (serverová část a uživatelské rozhraní).

Node.js Node.js je „*prostředí poháněné asynchronními událostmi umožňující běh aplikací psaných v jazyce JavaScript, Node.js je navržený pro tvorbu škálovatelných síťových aplikací.*“ [28] (překlad dle autora). Rozšiřuje původně pouze „client side“ jazyk o knihovny a funkce, které dodávají potřebnou funkcionalitu pro tvorbu plnohodnotných webových aplikací. Tato technologie se velmi rychle stala oblíbenou a stále byla i v roce 2017 jednou z nejoblíbenějších technologií [24].

Výhodou aplikací postavených na Node.js je jejich jednoduchost a tedy i rychlé nasazení. Při využití frameworku jako je Express (více informací na: https://expressjs.com/) je tvorba dynamické webové aplikace otázkou několika málo hodin. Další velkou výhodou je popularita, která zaručuje velké množství udržovaných knihoven a také studijních materiálů. V neposlední řadě je výhodou také rychlost výsledné aplikace oproti „standardním“ skriptovacím jazykům.

Jednou z nevýhod může být to, že JavaScript nepodporuje striktní typování proměnných. Velkou nevýhodou může být rychle se měnící prostředí, protože tato technologie je velmi populární a poměrně nová, často se stává, že každý měsíc vyjde nová verze nějaké knihovny (pozor na využívání netestovaných knihoven). Toto může být problém u větších projektů, které v budoucnu počítají s rozšiřováním funkcionalit a narazí na nekompatibilitu novějších verzí s verzí, na které byla aplikace původně postavena. Dále také nevýhodou, při tvorbě větších projektů, může být volnost, která je programátorům dána (pokud je vývojářský tým nezkušený).

2.5.2.3 Python

Python se v žebříčku nejpopulárnějších programovacích jazyků průzkumu [24] umístil na pátém místě s 32 % a dle [25] je čtvrtým nežádanějším programovacím jazykem. Python je „*univerzální programovací jazyk, využívaný pro vývoj webových aplikací a jako podpůrný jazyk pro vývojáře software*“ [29] (překlad dle autora). Podpůrným jazykem může být proto, že je jazykem interpretovaným a dovoluje psát jednoduché skripty, které mohou využívat složitých knihoven. Tímto umožňuje jednoduše, bez kompilace, spustit složitější funkcionalitu, která se dá často shrnout do několika málo souborů. Tento jazyk se tedy zaměřuje na jednoduchost a čitelnost kódu.

Python nabízí frameworky pro tvorbu webových aplikací. Nejznámějšími takovými frameworky jsou například:

- Django
- TurboGears
- Web2py

Hlavní výhodou Pythonu je: objektově orientovaný, přenositelný, přehledný kód. Nevýhodou je nutnost extenzivního testování veškeré funkcionality, neboť python je, jak již bylo zmíněno, interpretovaný jazyk a chyby jsou odhaleny teprve při běhu aplikace.

Tedy Python je spíše vhodnější na menší webové aplikace. Při tvorbě aplikací většího rozsahu je potřeba důkladně kód testovat, což může zabrat hodně času.

2.5.2.4 C#

C# je programovací jazyk navržený firmou Microsoft. Je objektově orientovaným, univerzálním programovacím jazykem. Používán je primárně pro tvorbu aplikací na .NET frameworku, ale je také doporučeným programovacím jazykem pro herní engine Unity. Jazyk je možné použít téměř na cokoliv od tvorby enterprise webových aplikací až po tvorbu aplikací mobilních. Stejně jako Java je velice vhodný na komplexní a rozsáhlé aplikace. [23]

Výhodou je, že již v základu platforma .NET obsahuje framework ASP.NET, který umožňuje tvorbu rozsáhlých webových aplikací. Jazyk C# je striktně typovaným a kompilovaným jazykem, což může být také velkou výhodou, neboť velké množství chyb je odchyceno ještě před samotným během aplikace. Dále je výhodou také perfektní vývojové prostředí VisualStudio.

Největší nevýhodou je, že pokud je potřeba provozovat webový server na operačním systému, který nepodporuje správnou verzi .NET frameworku nemusí to být možné. Existují způsoby jak provozovat C# mimo .NET framework, ale je potřeba si důkladně zkontrolovat, zda je možné je využít na cílovém serveru.

2.5.2.5 C++

C++ je rozšířením jazyka C a je tedy striktně typovaným, kompilovaným jazykem. I pro jazyk C++ existují webové frameworky jako je například Wt (více info na: https://www.webtoolkit.eu/wt/doc/tutorial/wt.html). C++ je velmi univerzálním jazykem a programátora prakticky vůbec neomezuje v realizaci jeho vize, což je jeho vůbec největší výhodou. Tvorba aplikací v C++ je však velmi časově náročná a hlavní výhody tohoto jazyka se projeví teprve pokud bude aplikace většího rozsahu. To dělá C++ nevhodnou volbou pro menší webové aplikace nebo aplikace, které musejí být rychle zhotoveny.

2.5.2.6 PHP

PHP je skriptovací jazyk využívaný primárně pro serverové části webových aplikací. „*80 % z top 10 miliónů webových stránek nějakým způsobem využívají PHP, včetně portálů Facebook a Wikipedia.*“ [23] (překlad dle autora) PHP je také využíváno množstvím známých CMS systémů jako je WordPress či Drupal. Jazyk je velice oblíbený a jednoduchý na naučení. Je vhodný pro menší až středně velké aplikace, avšak pokud je dobře využit, lze v něm napsat i aplikace většího rozsahu. S kvalitou a udržitelností aplikačního kódu se snaží také částečně pomoci aplikační frameworky.

Následuje dle [30] seznam nejpoužívanějších PHP frameworků:

1. Laravel (https://laravel.com/)
2. CodeIgniter (https://codeigniter.com/)
3. Symfony (https://symfony.com//)
4. Zend (https://framework.zend.com/)
5. Yii 2 (https://www.yiiframework.com/)

Laravel Laravel využívá plně možností, které nabízí programovací jazyk PHP. Díky tomu je psaní aplikací v tomto frameworku velice rychlé a kód je pro programátora znalého PHP dobře čitelný. Velkým plusem je podrobná a přehledně psaná dokumentace. Laravel je postavený za využití Symfony Components. Dále také obsahuje velké množství přidaných knihoven jako je například šablonovací systém Blade [30]. Pro připojení k databázi využívá ORM Eloquent. (oficiální stránky: https://laravel.com/docs/master/eloquent)

Symfony Symfony je komplexní, výkonný a vysoce modulární framework. Je podporován francouzskou vývojářskou společností SensioLabs, kterou je také společně s komunitou vyvíjen [30]. Framework samotný je sestaven z modulů (základem jsou také Symfony Components) a tím pádem lze některé

z modulů, v případě nutnosti, jednoduše nahradit a od základů změnit chování celého systému. Veškeré komponenty jsou velice dobře zdokumentovány, což je také velkou předností tohoto frameworku. Symfony, na rozdíl od Laravelu, v základu využívá pro připojení k databázi Doctrine ORM (oficiální stránky: https://www.doctrine-project.org/).

CodeIgniter CodeIgniter je minimalistický framework, který dává vývojáři velkou volnost při implementaci aplikace. Tento framework je také „*ideální pro rapidní vývoj aplikací*“ [30] (překlad dle autora). Díky svému minimalistickému pojetí je také velmi rychlý. Dle [30] se dokonce umístil na 4 místě nejrychlejších PHP frameworků. Také je velice jednoduchý na naučení a obsahuje množství užitečných knihoven [30].

Nette I přesto, že se tento framework v žebříčku, který je k nalezení výše neumístil, bylo by minimálně neslušné ho vynechat. Nette je framework českého původu, je však uznávaný i mimo Českou republiku. Využívá vzoru Model View Presenter (MVP), na místo běžněji využívaného Model View Controller (MVC). Díky této volbě se v některých částech výrazně liší od ostatních frameworků. Nejvíce je tato změna znát při tvorbě uživatelského rozhraní a komponent s ním spojených (formuláře, ajax, …). Pro tvorbu uživatelského rozhraní využívá vlastního šablonovacího systému zvaného Latte (více info na: https://latte.nette.org/en/). Slabou stránkou tohoto frameworku je nepříliš detailní dokumentace.

2.6 Metodiky

Metodiky jsou sady postupů, které mají za účel unifikovat procesy při realizaci nejrůznějších projektů. Unifikací a standardizací pomáhají zvýšit kvalitu finálního produktu.

Tato sekce se zabývá analýzou vývojových, dokumentačních a implementačních metodik. Zmíněná analýza slouží jako podklad pro volbu správných metodik, které budou využity v tomto projektu. Nejprve se sekce věnuje metodikám vývoje. Dále jsou krátce popsány možné dokumentační metodiky a v poslední řadě jsou analyzovány také implementační postupy.

2.6.1 Metodiky vývoje

„*Metodika vývoje je způsob jakým lze řídit a spravovat vývoj softwarových projektů.*“ [31] (překlad dle autora) Těchto metodik je velké množství, neboť každý projekt se více či méně liší a neexistuje způsob, kterým by se daly veškeré postupy pro všechny typy projektů sjednotit. I přes to existují velmi obsáhlé metodiky, které se zaměřují na velkou škálu nejrůznějších problémů. Tyto obsáhlé metodiky jsou často velice komplexní a mohou být nevhodné pro

menší projekty. Dle [31, s. 1-2] je následující seznam výčtem metodik, které byly v roce 2012 nejdiskutovanější:

- Vodopád (Waterfall)
- Agilní metodiky (Agile family)
- Metodiky Unified Process (Unified Process family)
- PRINCE2 (PRojects IN Controlled Environments 2)
- Project Management Body of Knowledge (PMBOK)
- Capability Maturity Model Integration (CMMI)
- Microsoft Solutions Framework (MSF)
- Crystal Methods Methodology
- Joint Application Design (JAD)
- Lean Development (LD)
- Spiral Model
- Systems Development Life Cycle (SDLC)
- Apple design process

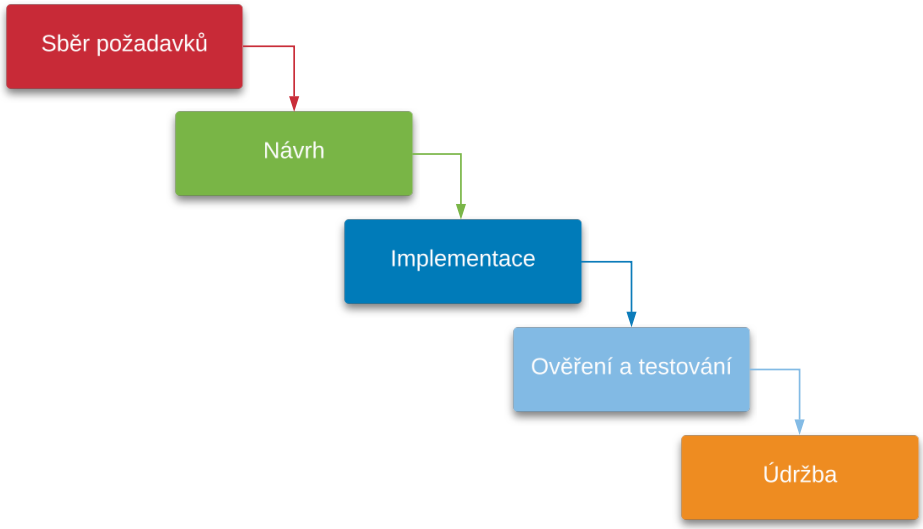
Tato sekce se však nebude zabývat analýzou všech výše zmíněných metodik, neboť to by dalece přesahovalo rozsah této práce. V sekci je nejprve k nalezení krátký popis metodiky zvané „Vodopád“. Dále pak jsou krátce popsány metodiky z rodiny Unified Process a poslední jsou analyzovány metodiky agilní.

2.6.1.1 Vodopád

Metoda vodopádu „*je původní, tradiční metoda vývoje softwaru*“ [31] (překlad dle autora), která přistupuje k vývojovému procesu jako k vodopádu. Začíná sběrem uživatelských požadavků, pokračuje na návrh, dále na implementaci, ověření a v poslední řadě údržbu. Metoda se nikdy nevrací k předchozímu kroku (proto název vodopád). Na obrázku 2.5 je pro lepší představu graficky vyobrazen „vodopádový“ průběh metody.

Tato metoda je „*lineární metoda, která dává velký důraz na sběr požadavků a návrh softwarové architektury ještě před samotným vývojem a testováním*“ [31, s. 1] (překlad dle autora).

Výhody Hlavní výhodou je propracovaný plán celého projektu. Tato metodika také navádí k tvorbě podrobné dokumentace projektu. Díky své linearitě dovoluje jasně určit stav vývoje projektu. [31, s. 1]



Obrázek 2.5: Grafické zobrazení metodiky Vodopád

Obrázek je inspirován článkem [32].

Nevýhody Je velmi obtížné přidávat či upravovat funkcionalitu během vývoje, což je často potřeba, neboť většinou jsou nedostatky odhaleny právě během vývoje. Proto organizace často ve smlouvě uvádějí tzv. „feature freeze“ (neboli zamrznutí funkcionality), čímž odmítají změnu či přidávání další funkcionality do aktuálně dodávané verze softwaru po započetí implementační části vývoje. A tyto změny ve funkcionalitě jsou poté odsunuty do dalších verzí softwaru, což u větších projektů znamená i roky čekání na dodání určité funkcionality. [31, s. 1]

2.6.1.2 Agilní metodiky

Agilní metodiky dávají důraz na flexibilitu a komunikaci se zákazníkem. „*Minimalizují výdaje na vývoj a přesto dodávají kvalitní software.*“ [31, s. 2] (překlad dle autora) Software je dodávaný v kratších časových úsecích než u klasických metodik jako je například metoda vodopádu. Tím pádem se tyto metodiky zaměřují na vývoj založený primárně na zpětné vazbě od zákazníka. „*Typicky jsou všichni členové týmu zapojeni do všech aspektů plánování, implementace a testování.*“ [31, s. 2] (překlad dle autora) Zákazník, či jeho reprezentace může být také přímo součástí vývojového týmu. Agilní metodiky často dávají důraz na průběžné testování softwaru. Tyto informace ohledně agilních metodik byly převzaty z [31, s. 1-3].

Nejzajímavějšími agilními metodikami jsou SCRUM a Extrémní Progra-

mování (XP), které jsou popsány níže. Dalšími agilními metodikami jsou například: Test Driven Development (TDD) nebo Rapid Application Development (RAD).

SCRUM SCRUM byl dle [31, s. 2] v roce 2012 nejpopulárnější agilní metodikou. Hlavním rysem této metodiky jsou tzv. „sprinty“ a každodenní SCRUM. Sprint je jedna iterace, při které je dodávána určitá funkcionalita systému. Tato iterace standardně trvá přibližně měsíc, je však možné zvolit různé délky iterací podle potřeb projektu. Vývojový tým má také každodenní schůzky zvané SCRUM, které mají za účel udržet tým soustředěný. Tým vede „scrum master“, který má za účel dohlédnout na dodržení pravidel a také na to, aby tým nebyl „rozptylován“. [31, s. 2].

Extrémní programování Velmi zajímavou metodikou je extrémní programování, které dává důraz na spolupráci a neustálou kontrolu kódu. Vývojáři pracují ve dvojicích a tím se zvyšuje kvalita kódu výsledného softwaru. Nevýhodou však je vysoká cena, neboť je potřeba platit za stejný čas dvojnásobek. [31, s. 2].

2.6.1.3 Unified Proces

Metodiky z rodiny Unified Proces jsou vhodné spíše pro větší projekty s hierarchickou strukturou vývojových týmů. Tyto metodiky bývají často velmi obsáhlé a komplexní. Zaměřují se, podobně jako metoda vodopádu, na důkladnou analytickou část vývoje. V této analytické části se primárně zaměřují na uživatelské požadavky a případy užití. Velice často probíhá analýza, návrh, implementace a testování v těsné návaznosti nebo paralelně.

Metodikami z této rodiny jsou například: Rational Unified Process (RUP), Enterprise Unified Process nebo Open Unified Process.

2.6.2 Dokumentační metodiky

Dokumentační metodiky bývají součástí vývojových metodik. Základní dokumentací k projektu je jeho návrhová a analytická část. V této sekci je pouze shrnuto pár informací ohledně kódové dokumentace.

Dle [33, s. 74-93] by se programátor měl vyhýbat tvorbě komentáři zaplněného kódu. Programátor je v knize nabádán, aby se raději vyjadřoval přímo kódem a jen v nejnutnějších situacích využil komentářových bloků přímo v kódu.

A tedy pokud systém obsahuje složitější funkcionalitu a očekává její rozšiřitelnost, je lepší vytvořit externí dokumentaci v podobě strukturovaného textu. Avšak toto neplatí pro veřejná rozhraní, či abstraktní třídy, které by měly obsahovat dobrou kódovou dokumentaci, aby programátoři, kteří těchto tříd chtějí využít věděli co přesně mají implementovat.

V případě, že je potřeba tvořit dokumentaci přímo v kódu je dle [33, s. 83-84] nutné vyvarovat se špatným typům komentářů. Za špatné jsou považovány takové komentáře, které jsou příliš upovídané, pouze přeříkávají informace (které jsou již obsaženy v kódu) nebo komentáře, které lze nahradit názvem funkcí, parametru nebo třídy.

2.6.3 Implementační metodiky

Existuje velké množství aspektů implementace projektu, které je vhodné analyzovat, avšak tato sekce se zaměřuje pouze na způsob propojení kódové části s datovým úložištěm. Přesněji se zabývá způsoby napojení Object Relational Mapping (ORM) na aplikační databázi. Existují tři hlavní způsoby jak lze k této problematice přistupovat:

- Database first (Nejdříve databáze)
- Model first (Nejdříve model)
- Code first (Nejdříve kód)

Tyto způsoby se liší v pořadí v jakém jsou jednotlivé části vyvíjeny. Následující informace jsou založeny na [34] a [35].

Database first „Database first“ znamená, že aplikační kód pro mapování objektů je tvořený ve chvíli, kdy už je k dispozici funkční databáze. Tedy kódové mapování je generováno na základě již existující databáze. Tento způsob je vhodný, pokud je potřeba jasně oddělit datové struktury od kódové implementace. Také je velmi užitečný, pokud je počítáno s tím, že na databázi bude připojeno více různých aplikací (každá aplikace může být napsaná v jiném jazyce).

Nevýhodou je časová náročnost, neboť je potřeba definovat schéma databáze a zároveň definovat i veškeré kódové mapování. Další nevýhodou je, že ORM nástroje často neumějí namapovat některá platformě specifická metadata. Dále mapování vygenerované podle existující databáze také nemusí odchytit všechna omezení.

Model first Tento přístup je velmi podobný přístupu „database first“. Liší se v tom, že kód pro mapování není generovaný z existující databáze, ale z modelu popisujícího datové struktury a vztahy v databázi. Z tohoto modelu jsou následně také generovány Data Definition Language (DDL) příkazy pro vytvoření databáze samotné. Model může popisovat i celý systém (existují nástroje, které z tohoto modelu umějí vygenerovat části aplikačního kódu), tím se ale tato sekce nezabývá. Takový přístup má stejné nevýhody jako přístup „database first“. Výhodou je, že model se mnohem snadněji upravuje než ručně psané DDL příkazy. Model je také jednodušší převést na jiný databázový stroj pokud je to třeba.

Code first „Code first“ je postup, kdy je databáze vygenerována na základě mapovacích metadat (aplikačního kódu). Tento přístup je časově nejúspornější, protože je celá databázová struktura definována na jednom místě (nedochází k duplikaci informací v metadatech a databázovém modelu). Nevýhodou tohoto přístupu je, že schéma databáze je definováno mapovacími metadaty a pokud chceme databázi upravit, musíme to provést jedním ze tří způsobů: Úpravou těchto metadat, ručně pomocí DDL příkazů, nebo extrahovat model z existující databáze a ten poté upravit (tato extrakce nemusí být kompletní a tím pádem může být časově náročné tento extrahovaný model opravit). Při využití tohoto postupu je také těžší objevit případné chyby v databázovém modelu.

Kapitola 3

Návrh

3.1 Přehled systému

Ještě před tím, než bude práce pokračovat samotným návrhem systému, je vhodné uvést, jaké části systému jsou předmětem této práce. Informace ohledně jednotlivých částí systému jsou průběžně uváděny i dále v práci, je však vhodné věnovat jednu sekci jejich rychlým shrnutím. V této sekci je také uvedeno několik pojmů, které jsou využívány v následujících sekcích a kapitolách. V práci je občas zaměněn pojem „webová aplikace“ za „systém“ ve smyslu, že není potřeba explicitně definovat zda se jedná o webovou aplikaci, či celý systém (většinou v případech, kdy je z kontextu jasné o jaký z těchto pojmů se jedná).

3.1.0.1 Front-end a Back-end systému

Tato práce se věnuje analýze, návrhu a implementaci jádra systému. Jádro systému (neboli také back-end systému) je tvořeno databází zařízení, back-end částí webové aplikace (pro zobrazení a editaci dat) a rozhraním pro napojení na výkonné funkce správy zařízení. Front-end část systému se v tomto projektu skládá z veškerých softwarových komponent, úzce spojených s uživatelským rozhraním webové aplikace. Na obrázku 3.1 jsou minimalisticky zobrazeny jednotlivé části systému. Modře vyznačená část s nadpisem „Jádro systému“ na obrázku je částí, kterou se zabývá tato práce. Tento obrázek je velice zjednodušený, neboť architektuře aplikace a celého systému se věnuje sekce 3.5 na straně 53 a také sekce 4.1 na straně 59.

3.1.0.2 Back-end webové aplikace

Do back-end části webové aplikace patří komponenty poskytující rozhraní pro práci s daty uloženými v databázi. Dále do této části patří autorizační a autentizační komponenty, které využívají datových struktur uložených v databázi

pro realizaci autorizovaného přístupu ke zdrojům. Do back-end části také patří veškerá business logika včetně rozhraní poskytující tuto logiku front-end části aplikace. Back-end je nasazen na aplikační server společně s front-end částí aplikace. Hrubá struktura aplikace a jednotlivých částí systému je také k nalezení na obrázku 3.1.

3.1.0.3 Front-end webové aplikace

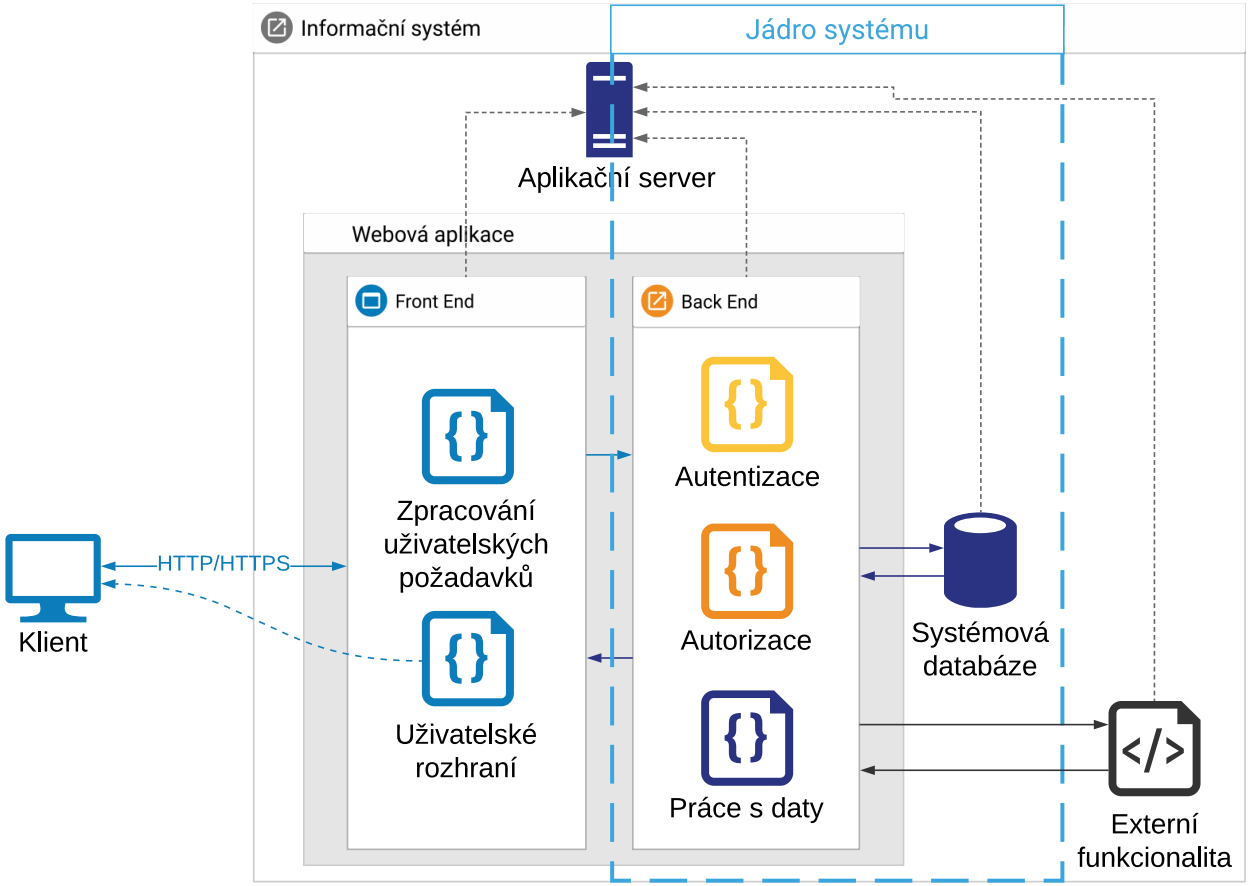
Front-end webové aplikace není předmětem této práce, je však vhodné uvést, že se skládá ze softwarových komponent, které mají za účel prezentovat data uživateli a také interpretují uživatelské příkazy. Tato část využívá back-endu aplikace pro práci s daty a také pro spuštění business logiky.

3.1.0.4 Databáze

Systémová databáze obsahuje veškerá data evidovaná systémem a je oddělena od webové aplikace, neboť databázové schéma musí, dle uživatelských požadavků, být implementačně nezávislé. Pokud by toto omezení nebylo zadáno, byla by databáze na obrázku 3.1 uvedena jako součást webové aplikace. Databáze, kromě doménově specifických dat, obsahuje také veškeré struktury potřebné pro realizaci autorizačního a autentizačního systému, které lze využít v jakémkoliv jiném projektu. Detailním návrhem databázového schématu se dále zabývá sekce 3.4 na straně 46.

3.1.0.5 externí funkcionalita

Externí funkcionalita je také oddělena od webové aplikace a je dodávána v podobě spustitelných souborů. Jak je vidět na obrázku 3.1, externí funkcionalita je součástí systému, avšak není součástí webové aplikace. Je to proto, že aplikace může fungovat i bez této funkcionality a externí funkcionalita může fungovat i sama o sobě. Tvorba externí funkcionality také není předmětem této práce.



Obrázek 3.1: Rozdělení systému

3.2 Volba technologií

Na základě uživatelských požadavků, kterým se tato práce zabývá v sekci 2.2 na straně 10, je potřeba zvolit technologie, které lze bez větších problémů provozovat na klientově serveru s linuxovou distribucí FreeBSD. Nejprve je potřeba se rozhodnout, zda při tvorbě projektu bude využito již existujícího řešení, které by stačilo pouze rozšířit. To by mohlo výrazně zúžit výběr technologií pouze na ty, které jsou řešením podporovány.

Dále je potřeba v závislosti na zvoleném řešení vybrat vhodné datové úložiště a implementační technologie. Do těchto implementačních technologií patří volba programovacího jazyku a případných frameworků. Projekt je již v pokročilé fázi vývoje a proto jsou všechny technologie již pevně zvoleny.

3.2.1 Shrnutí potřeb

Nejprve je na místě si krátce připomenout hlavní požadavky na systém. Kompletní přehled požadované funkcionality je k nalezení v sekci 2.2 na straně 10.

Systém je primárně určen pro evidenci dat s pevnou strukturou, jako jsou zařízení či projekty. Musí podporovat autorizovaný přístup k zabezpečeným zdrojům a editaci těchto práv skrze uživatelské rozhraní. Je také potřeba do systému integrovat autentizační systém Shibboleth SSO. Systém musí počítat s dlouhodobější podporou a měl by tedy počítat s tím, že bude dále rozšiřován i dalšími vývojovými týmy.

3.2.2 Zvolené řešení

Na základě analýzy provedené v sekci 2.1 na straně 5, bylo zvoleno řešení za využití aplikačního frameworku. Tento projekt tedy nevyužívá žádného existujícího řešení jako je Drupal či WordPress. Takový přístup byl zvolen proto, že existující řešení se z velké části zaměřují na tvorbu textového obsahu, což není primární účel tohoto systému. Systém také musí být snadno rozšiřitelný a proto využití dobře zdokumentovaného frameworku může v tomto ohledu výrazně pomoci (pokud jsou dodrženy standardní přístupy).

Nejprve se tedy sekce zaměří na volbu datového úložiště a poté naváže volbou programovacího jazyka a příslušného aplikačního frameworku.

3.2.3 Datové úložiště

V dnešní době je k dispozici velká škála datových úložišť, od běžných relačních databází až po cloudové NoSQL databáze. Cloudová řešení nejsou v tomto případě vhodná, neboť většina z nich je placená a jejich výhody by v tomto projektu nebyly využity. Je však na místě řešit zda zvolit SQL či NoSQL databázi.

SQL vs NoSQL Je potřeba si ujasnit, že ani jeden z těchto typů databází není univerzálně lepší. NoSQL na rozdíl od SQL nemá pevně dané schéma a umožňuje dynamicky přidávat a odebírat ukládaná data (schéma lze měnit během běhu databáze). Každý přístup má své výhody a nevýhody. NoSQL databáze jsou z pravidla mnohem lépe škálovatelné nežli databáze relační, ale kvůli dynamickému schématu jsou omezeny v operacích, které mohou nad daty provádět. Výhodou pevně daného schématu je, že databázový stroj může s daty důkladněji operovat (řazení podle více sloupečků, složité agregace, propojování dat, …). Pro NoSQL databáze existují externí nástroje, které tuto funkcionalitu doplňují, ale cenou je poměrně složitá konfigurace. [36]

Volba databáze a databázového stroje Je potřeba se zamyslet zda-li by systém výhody NoSQL databáze využil a jestli není vhodnější tradiční relační databáze. Protože ukládaná data jsou, až na pár výjimek, pevně strukturována, je relační databáze ideální volbou pro tento projekt.

Existuje velké množství dostupných relačních databázových strojů. Volba databázového stroje je také důležitou součástí výběru úložiště. Avšak rozbor veškerých možností a jejich výhod by si zasloužil vlastní práci.

Nejpopulárnějšími databázovými stroji jsou dle [37] (seřazeny dle popularity):

- Oracle Database (https://www.oracle.com/database/index.html)
- MySQL (https://www.mysql.com/)
- Microsoft SQL Server (https://www.microsoft.com/en-us/sql-server/)
- PostgreSQL (https://www.postgresql.org/)

Pro tuto práci byl zvolen databázový stroj MySQL, který je volně distribuovaný a open source. Tento databázový stroj byl zvolen proto, neboť podporuje veškerou potřebnou funkcionalitu a také protože je již instalován na serveru, na kterém systém bude provozován.

3.2.4 Programovací jazyk

Pro projekt byl na základě analýzy v sekci 2.5 na straně 27 zvolen programovací jazyk PHP. Tento jazyk byl zvolený proto, že je velice oblíbeným (není problém sehnat programátora, který by se o systém dále staral). Dále také existují knihovny a frameworky, které podporují veškerou potřebnou funkcionalitu. V neposlední řadě je zvolen také proto, že aktuální vývojový tým s ním již má zkušenosti z jiných projektů, což výrazně urychlí vývoj celého systému.

3.2.5 Framework

Jako stavební kámen celého systému byl zvolen framework Symfony, který je také analyzován ve výše zmíněné sekci 2.5. Zvolen byl primárně pro jeho perfektní dokumentaci a velmi dobrou rozšiřitelnost. Dále také byla Symfony zvolena proto, že existuje velké množství komunitou vytvořených knihoven a rozšíření, které urychlí integraci standardních funkcionalit, jako je nahrávání souborů nebo překlad uživatelského rozhraní. Jedním z důvodů byla také poměrně jednoduchá integrace autentizačního systému Shibboleth skrze tzv. „GuardAuthenticator“ (ohledně této problematiky více v kapitole 4). V poslední řadě také využívá (v základu) ORM Doctrine, které plně podporuje mapování dat pro vybraný databázový stroj MySQL.

Uživatelské rozhraní Front-end aplikace je standardně tvořený za pomoci HTML5 (HTML5 bylo zvoleno na základě analýzy v sekci 2.5.1.1), Kaskádových stylů (CSS) a technologie JavaScript (využito bude verze ES6). V závislosti na využití frameworku Symfony je pro tvorbu uživatelské rozhraní zvolen šablonovací systém Twig, který je tímto frameworkem doporučovaný. Tvorba uživatelského rozhraní však není předmětem této práce a proto se touto problematikou nebude dále zabývat.

3.3 Vybrané metodiky

Aby vývoj projektu postupoval bez větších problému je potřeba zvolit správné vývojové metodiky. Díky tomu, že se vývojový tým skládá pouze ze dvou lidí (tří lidí, pokud je počítána externí funkcionalita), není potřeba se zabývat strukturováním týmů. Je však nutné vyřešit jakým způsobem bude projekt spojen s externí funkcionalitou. Dále je také potřeba vyřešit propojení back-end a front-end části systému/aplikace.

3.3.1 Přehled vývoje

Celý projekt je rozdělen na tři části, kde za každou část je odpovědný jiný člen týmu. Uživatelské rozhraní a celý front-end systému vyvíjí Erik Zatloukal (student FIT ČVUT v Praze). Externí funkcionalitu pro realizaci výkonných funkcí robotů vyvíjí Petr Kolář (student FIT ČVUT v Praze). Za návrh a implementaci jádra systému je zodpovědný autor této práce (Dan Zatloukal, student FIT ČVUT v Praze).

3.3.2 Postup vývoje a integrace částí

Protože vývoj front-end a back-end části systému je prováděn členy týmu, kteří jsou v neustálém kontaktu, není potřeba se ohledně integrace těchto dvou částí pravidelně scházet. Pro napojení externí funkcionality na systém byly

pořádány týdenní schůzky, kde bylo napojení konzultováno a také testováno. Těchto schůzek se také zúčastnil zadavatel projektu, se kterým byl konzultován stav a vývoj systému.

Tento postup vývoje se dá přirovnat k některým agilním metodikám analyzovaným v sekci 2.6. I přes to, že žádná metodika není při vývoji aplikována, pomohla výše zmíněná analýza k získání přehledu o tom, jak lze nebo jak by se mělo k vývoji takovýchto systému přistupovat. Samozřejmě díky velikosti týmu není potřeba vést detailní záznamy o prováděných změnách, implementovaná funkcionalita je každý týden konzultována přímo se zadavatelem.

Celý projekt je, včetně databázového modelu, verzován skrze systém Git. Externí funkcionalita dodávaná studentem Petrem Kolářem je také verzována skrze systém Git a je přidána jako submodul do repositáře systému.

3.3.3 Napojení uživatelského rozhraní

Při integraci větších systémů je běžným postupem jasně oddělit back-end od front-end části pomocí aplikačního rozhraní. V tomto projektu jsou tyto dvě části odděleny pomocí několika aplikačních tříd. Tyto třídy jsou součástí jádra systému a poskytují rozhraní pro práci s databází a také ke všem business operacím (více v sekci 3.5).

Díky velikosti vývojového týmu a neustálého kontaktu není potřeba vytvářet složité specifikace rozhraní, které by bylo nutno dodržovat. Obě části systému jsou vyvíjeny ve stejném prostředí (stejný projekt a stejný repositář), vzájemný zásah je však minimalizován právě za pomoci výše zmíněných aplikačních tříd. Striktní oddělení zmiňovaných částí je porušeno pouze pokud je potřeba připravit kusy generického kódu (abstraktní třídy či rozhraní), určujícího jakým způsobem by měly některé front-end části systému být realizovány. Tvorba těchto generických částí kódu spadá do návrhu a implementace jádra systému.

3.3.4 Metoda implementace

Dále je potřeba zvolit správné postupy při samotné realizaci webové aplikace a databázového schématu. Součástí realizace aplikace je také dokumentace kódu. Tato sekce se tedy krátce věnuje také volbě dokumentačních metodik vhodných pro tento projekt.

Postup vývoje Na základě uživatelských požadavků na nezávislost datového úložiště a analýzy v sekci 2.6 na straně 32 byl zvolen přístup „database first“. Databázové schéma je tedy navrženo ještě před implementací systému samotného. Tento přístup přináší jednoduchou upravitelnost a přenositelnost databázového schématu. Dále také pomůže s brzkým odhalením nedostatků v tomto schématu. Schéma je však navrhováno s ohledem na „best practices“ projektu Doctrine.

Dokumentační metodiky Správná dokumentace systému je klíčem k jeho udržitelnosti. S ohledem na časové možnosti a na základě analýzy v sekci 2.6 je systém explicitně dokumentován pouze na místech, kde je to nezbytně nutné. Tato místa jsou převážně abstraktní třídy, rozhraní nebo třídy, které nevyužívají standardních přístupu frameworku Symfony. V ostatních částech je kód systému inspirován knihou [33], a tedy využívá popisné názvy tříd, funkcí, metod a proměnných. Kód také bere v úvahu „best practices“ zvoleného frameworku, které však občas kolidují s výše zmíněnou knihou. V případě kolize je dávána přednost „best practices“ zvoleného frameworku. Je také plánována kompletní externí dokumentace systému v podobě strukturovaného textu. Tato dokumentace však již není předmětem této práce.

3.4 Databáze

Databázový model je velice podobný modelu doménovému, je však specifičtější. Od doménového modelu se liší primárně tím, že u jednotlivých položek jsou specifikovány datové typy. Dále také, na rozdíl od doménového modelu, obsahuje veškeré (i implementačně specifické) entity. Vztahy jsou, na rozdíl od modelu doménového, již realizovány pomocí cizích klíčů.

Tento model tedy detailně popisuje způsob uložení dat v databázi. Databázový model však může popisovat omezení, která nelze odchytit přímo v databázovém stroji, jako jsou například výlučné vztahy. Uvedený model, jak již bylo zmíněno v sekci 3.4, byl vypracován ještě před implementací systému samotného. Během vývoje však bylo potřeba původní model mírně upravit, neboť v něm byly odhaleny nedostatky (většinou pouze chybějící položky), které nebyly při analýze a návrhu odhaleny.

3.4.1 Normalizace

Při tvorbě databázového modelu bylo dbáno na dodržení „normálních forem“. Normální formy udávají omezení pro data uložená v tabulkách a také pro vztahy mezi jednotlivými tabulkami. Tabulka je v normální formě, pokud splňuje všechna omezení definovaná danou formou/pravidlem. Proces transformace modelu na model vyhovující těmto pravidlům se nazývá „normalizace“. „*Normalizace je systematický přístup rozkladu tabulek za účelem eliminace redundantních dat a nežádoucích charakteristik, jako jsou anomálie při vkládání, upravování nebo mazání.*“ [38] (překlad dle Autora)

Normalizační formy se dle [38] dělí do:

- 1NF – První Normální Forma (First Normal Form)
- 2NF – Druhá Normální Forma (Second Normal Form)
- 3NF – Třetí Normální Forma (Third Normal Form)

- BCNF – Boyce and Codd Normal Form
- 4NF – Čtvrtá Normální Forma (Fourth Normal Form)

3.4.1.1 1NF – První Normální Forma

Tabulka je v První Normální Formě pokud splňuje:

- Sloupečky/atributy obsahují pouze atomické hodnoty.
- Žádné dva sloupečky/atributy v tabulce nemají stejné jméno.
- Každá skupina souvisejících dat je extrahována do vlastní tabulky a je identifikována primárním klíčem.
- Nezáleží na pořadí v jakém jsou data uložena.

Omezení jsou převzata z [38] a také z [39]. Tyto vlastnosti jsou celkem přímočaré, přesto pro upřesnění následuje krátký popis.

První omezení První omezení říká, že sloupeček musí obsahovat pouze jednu hodnotu daného typu. Například tabulka „uživatel“ ve sloupečku „skupina“, který eviduje název skupiny do které uživatel patří, nesmí mít více než jeden název skupiny (například oddělený čárkou „skupina1, skupina2“).

Druhé a třetí omezení Druhé omezení není třeba rozebírat a třetí omezení vyžaduje aby každá tabulka držela data pouze z jedné domény. Tedy například aby tabulka *předměty* nedržela informace ohledně učitelů (příjmení učitele, email, …).

Čtvrté omezení Poslední omezení vyžaduje, aby uložená data nebyla závislá na pořadí, v kterém jsou uložena. Tedy pokud prohodíme pořadí sloupečků, význam dat se nemění.

3.4.1.2 2NF – Druhá Normální Forma

Tabulka je ve Druhé Normální Formě pokud:

- Tabulka je v První Normální Formě.
- Tabulka neobsahuje částečné závislosti.

Předešlá omezení jsou převzata z [38]. Pro pochopení těchto omezení je potřeba vysvětlit, co znamená „částečná závislost“.

Závislost Každý řádek v tabulce je jasně identifikován nějakým kandidátním klíčem (viz. slovník: Kandidátní Klíč). Pokud známe nějaký kandidátní klíč, můžeme získat data uložená v příslušném řádku/záznamu. Tím pádem jsou tato data závislá na tomto kandidátním klíči. Tomuto vztahu se říká „závislost“. [40]

Částečná závislost Tento typ závislosti může vzniknout, pokud má tabulka primární či kandidátní klíč složený z více sloupečků (kompozitní klíč). Pokud mají nějaké atributy/sloupečky závislost pouze ná části tohoto klíče, vzniká částečná závislost.

Například realizujeme many-to-many vztah mezi uživatelem a rolí pomocí tabulky *user_role*. Pokud tato tabulka obsahuje, kromě primárního klíče uživatele a role (které tvoří kompozitní primární klíč této tabulky), například křestní jméno uživatele, vzniká částečná závislost křestního jména na primárním klíči uživatele, což je pouze část primárního klíče této tabulky. [40]

3.4.1.3 3NF – Třetí Normální Forma

Tabulka je ve Třetí Normální Formě pokud splňuje všechny následující omezení:

- Tabulka je ve Druhé Normální Formě.
- Tabulka neobsahuje tranzitivní závislosti.

Tato omezení jsou převzata z [38]. Nyní je potřeba alespoň krátce nastínit co znamená „tranzitivní závislost“.

Tranzitivní závislost Tento typ závislosti vzniká pokud sloupeček/atribut (může být i více atributů), který není součástí žádného kandidátního klíče, je závislý na sloupečku/atributu, který také není součástí žádného kandidátního klíče (avšak je závislý na nějakém kandidátním klíči). Pokud tabulka obsahuje tranzitivní závislost, je téměř jisté, že bude obsahovat také duplicitní data. [38]

3.4.1.4 BCNF – Boyce and Codd Normal Form

Tato forma je rozšířením Třetí Normální Formy. Aby tabulka byla v BCNF musí splňovat následující:

- Pro každou závislost X na Y, X musí být Super Klíčem.

Vysvětlení Pokud se vyskytne závislost X na Y, je potřeba, aby X bylo částí nějakého kandidátního klíče.

3.4.2 Modelová omezení

Přestože byl databázový model vytvořen ještě před realizací systému samotného, bylo nutné myslet na případná omezení, která by mohla výrazně zlepšit výkon systému po integraci databáze skrze ORM Doctrine. Proto byly při návrhu databázového modelu dodržovány „best practices“ projektu Doctrine. Toto mimo jiné dle [41] zahrnuje:

- Omezení relací mezi jednotlivými tabulkami na minimum.
- Eliminaci kompozitních klíčů v místech, kde jich není nutně třeba (přidávají zátěž na systém).
- Omezení automatického kaskádování persistencí či mazání přímo na úrovni ORM (Z toho plyne nutnost tyto problémy řešit na straně databázového stroje).

3.4.3 Nástroje

V závislosti na zvoleném databázovém stroji byl model vytvořen za využití nástroje MySQL Workbench (https://www.mysql.com/products/workbench/). Tento nástroj umožňuje vygenerovat DDL příkazy pro vytvoření schématu a také nabízí velmi jednoduchou správu celé databáze.

3.4.4 Kompletní databázový model

Kompletní databázový model je k nalezení na přiloženém CD ve složce *visual_attachments/diagrams*. Diagramy zobrazující databázový model nebo jeho části jsou realizovány skrze Entity Relationship Diagram (ERD) za využití vazební notace „crow's foot“. V těchto diagramech jsou již veškeré názvy v anglickém jazyce, neboť se jedná téměř o implementační záležitost a tento projekt pro implementaci využívá výhradně anglického jazyka. Dále se tato sekce věnuje primárně tabulkám/entitám, které byly vynechány při tvorbě doménového modelu.

3.4.4.1 Zabezpečené zdroje a autorizační struktury

Největším rozdílem databázového modelu od doménového je, že databázový model již obsahuje implementačně specifické struktury. V databázovém modelu přibyly, mimo jiné, čtyři velice důležité tabulky:

- Zabezpečený zdroj (v modelu pod názvem *secured_resource*)
- Resource Accessor – Objekt, kterému lze udělit práva ke zdroji (v modelu pod názvem *resource_accessor*)
- Záznam řízení přístupu (v modelu pod názvem *access_control_entry*)
- Typ zabezpečeného zdroje (v modelu *secured_resource_type*)

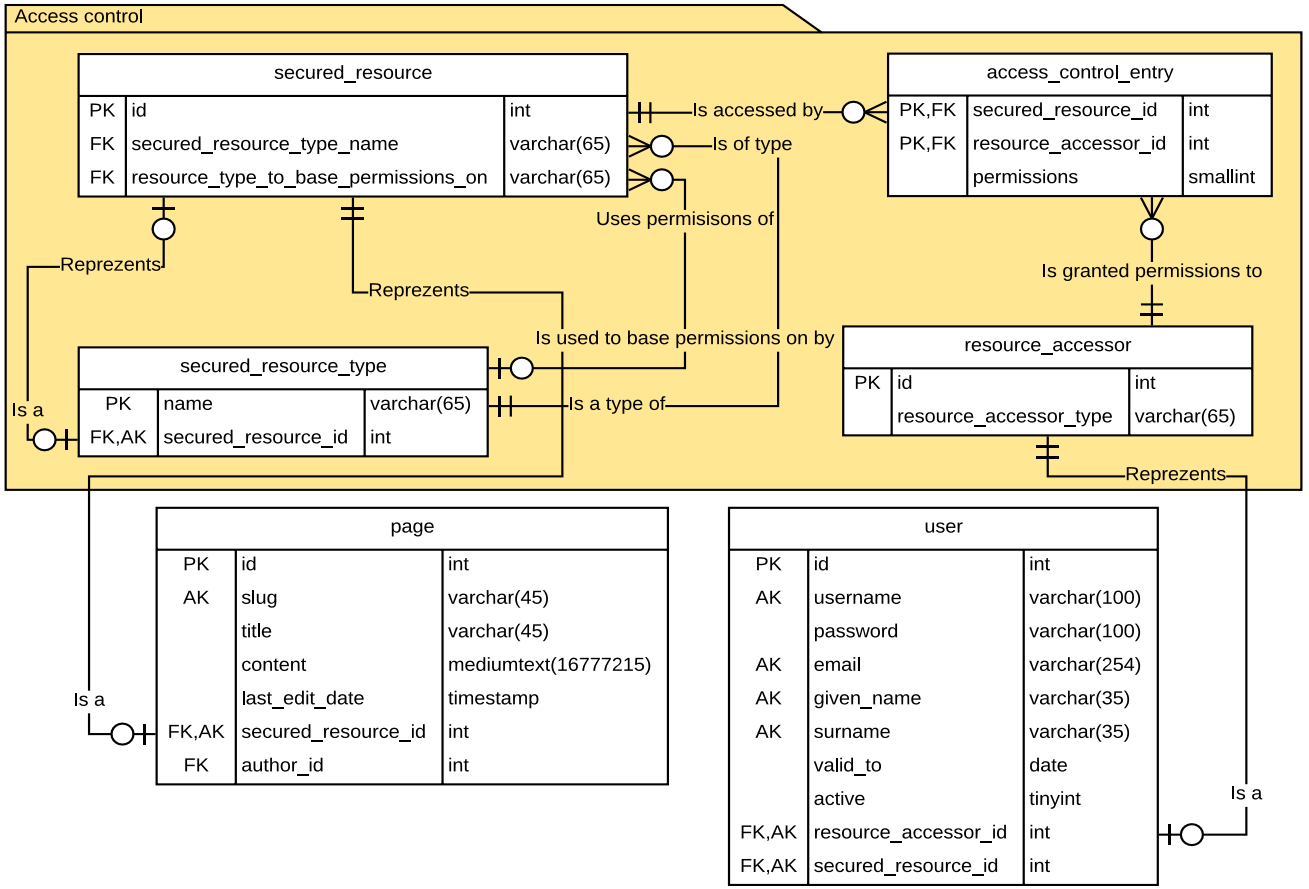
Termíny při popisování diagramu V následující sekci jsou používány některé nestandardní výrazy a je tedy vhodné je zde vysvětlit.
Následuje seznam využívaných termínů:

Řízení přístupu — Tento výraz reprezentuje autorizační systém odpovídající uživatelským požadavkům na autorizaci, kterým se práce věnuje v sekci 2.2 na straně 10. Takový autorizační systém dovoluje dynamicky udělovat či odebírat práva na zabezpečené zdroje.

Objekt — V této sekci je „objektem“ nazýván jeden záznam v tabulce. Tento výraz je využíván proto, že zpřehledňuje části, kde je nutno vysvětlit funkční propojení tabulek mezi sebou (jaký význam mají tabulky a jejich záznamy v systému). Pokud by nebyly popisy generické, nebylo by tohoto výrazu třeba.

Řešení autorizace Většina řešení jako je Symfony Security ACL (dostupný na: https://github.com/symfony/security-acl) využívají pro identifikaci objektů některé implementačně specifické atributy (jako je například název tříd). Protože v uživatelských požadavcích je požadována nezávislost databáze na aplikaci, bylo pro realizaci dynamických přístupových práv využito vlastní řešení. Toto řešení bylo inspirováno také článkem [42]. Důraz byl dán na jednoduchost a na udržení čistoty tabulek (do každé tabulky přidat co nejméně dat vztažených k autorizačnímu systému).

Na diagramu 3.2 jsou zobrazeny tabulky, které umožňují realizaci řízení přístupu k zabezpečeným zdrojům. K modelu byly přidány také dvě ukázkové tabulky *page* a *user*, které reprezentují zabezpečený zdroj a objekt, kterému lze udělit práva. Kromě těchto dvou, byly veškeré další tabulky s vazbou na řízení přístupu, vynechány (pro přehlednost). Dále budou jednotlivé části výše zmíněného diagramu krátce popsány za účelem vysvětlení jejich významu v systému.



Obrázek 3.2: ERD – Řízení přístupu

Zabezpečený zdroj Pro realizaci dynamické autorizace dat je potřeba jednotně evidovat objekty, na které lze udělit práva. Právě toto realizuje tabulka *secured_resource* (zabezpečený zdroj).

Jakékoliv tabulce, která chce využít řízení přístupu, stačí vytvořit vazbu na zabezpečený zdroj. Při vkládání dat do této tabulky stačí vytvořit korespondující záznam v zabezpečených zdrojích. Od chvíle, kdy je nově vložený objekt svázán se záznamem v zabezpečených zdrojích, je možné na něj udělit přístup.

Typ zabezpečeného zdroje Je velice přínosné evidovat typ zabezpečeného zdroje, neboť poté je možné získat objekt, který reprezentuje. Tabulka *secured_resource_type* eviduje tyto typy.

Je vhodné upozornit na to, že tato tabulka má také vazbu zpět na zabezpečený zdroj. Toto není chyba ani náhoda, neboť typ zabezpečeného zdroje je také zabezpečeným zdrojem. Tímto je umožněno přidávat práva na všechny zdroje daného typu.

Resource Accessor Protože je možné udělovat práva pouze vybraným typům objektů (například uživatelům), je nutno evidovat minimálně tento typ. V tomto případě však byl zvolen univerzálnější přístup, kdy jsou evidovány veškeré objekty, kterým lze práva udělit. Tyto objekty eviduje právě tabulka *resource_accessor*.

Každé tabulce, jejíž objekty chtějí využít autorizovaného přístupu k zabezpečeným zdrojům, stačí vytvořit vazbu na Resource Accessor. Při vkládání nového objektu je poté možné vytvořit záznam ve výše zmíněné tabulce. Ve chvíli, kdy existuje vazba mezi nově vloženým objektem a záznamem v tabulce *resource_accessor*, je možné tomuto objektu udělit práva na kterýkoliv evidovaný zdroj.

Záznam řízení přístupu Záznam řízení přístupu eviduje kdo má práva na jaký zabezpečený zdroj. Práva jsou ukládána jako unsigned small integer (tedy číslo v rozsahu 0–65535) a jsou využívána stejně jako práva linuxová (každý bit reprezentuje jednu nastavitelnou hodnotu). Je již na aplikaci, aby těmto bitům přiřadila význam (tento přístup bude v budoucnu změněn a význam bude pevně definovaný). Každý záznam v této tabulce eviduje:

- Komu jsou práva udělena (v tabulce položka *resource_accessor_id*)
- Jaká práva jsou udělena (v tabulce položka *permissions*)
- Na jaký zdroj jsou práva udělena (v tabulce položka *secured_resource_id*)

3.4.4.2 Obnovení hesla

Poslední tabulka, které se tato sekce věnuje je *password_reset_token*. Protože systém podporuje autentizaci bez využití nástrojů třetích stran, je potřeba evidovat také uživatelská hesla. Problémem může být ztráta hesla a proto tato tabulka nabízí možnost evidence dočasných „žetonů“ (viz. slovník: token), které mohou být využity v aplikaci pro ověření identity uživatele, aniž by zadával přihlašovací údaje.

3.5 Architektura systému

Pro snížení provázanosti je vhodné rozdělit systém do několika vrstev. Tento projekt využívá rozdělení inspirované třívrstvou architekturou. Pro lepší rozdělení prezentační vrstvy také využívá návrhového vzoru MVC. Toto pomůže zvýšit udržovatelnost systému a také jasně oddělit front-end část od back-end části systému.

3.5.1 Hlavní součásti systému

Pro osvěžení paměti bude v této sekci nejdříve krátce uvedeno několik základních pojmů ze sekce 3.1. Uvedené pojmy jsou potřeba pro pochopení k jakým částem systému se následující sekce vztahují.

Celý systém se skládá ze tří částí. Zmiňovanými částmi jsou: webová aplikace, datové úložiště (nebo také databáze) a externí funkcionalita (realizující výkonné funkce robotů).

Datové úložiště pouze uchovává data, která využívají aplikace na toto úložiště napojené. Tedy nestará se o žádnou doménově specifickou logiku.

Webová aplikace je napojena na úložiště. Interpretuje uložená data a nabízí uživateli rozhraní pro jejich manipulaci. Tato aplikace je středem celého systému.

Externí funkcionalita je dodávána v podobě spustitelných souborů a není pevně svázána s tvorbou aplikace. Aplikace využívá funkcionalit, které tyto soubory poskytují pro realizaci výkonných funkcí zařízení. Zmíněná funkcionalita je také přímo napojena na datové úložiště.

3.5.2 Třívrstvá architektura

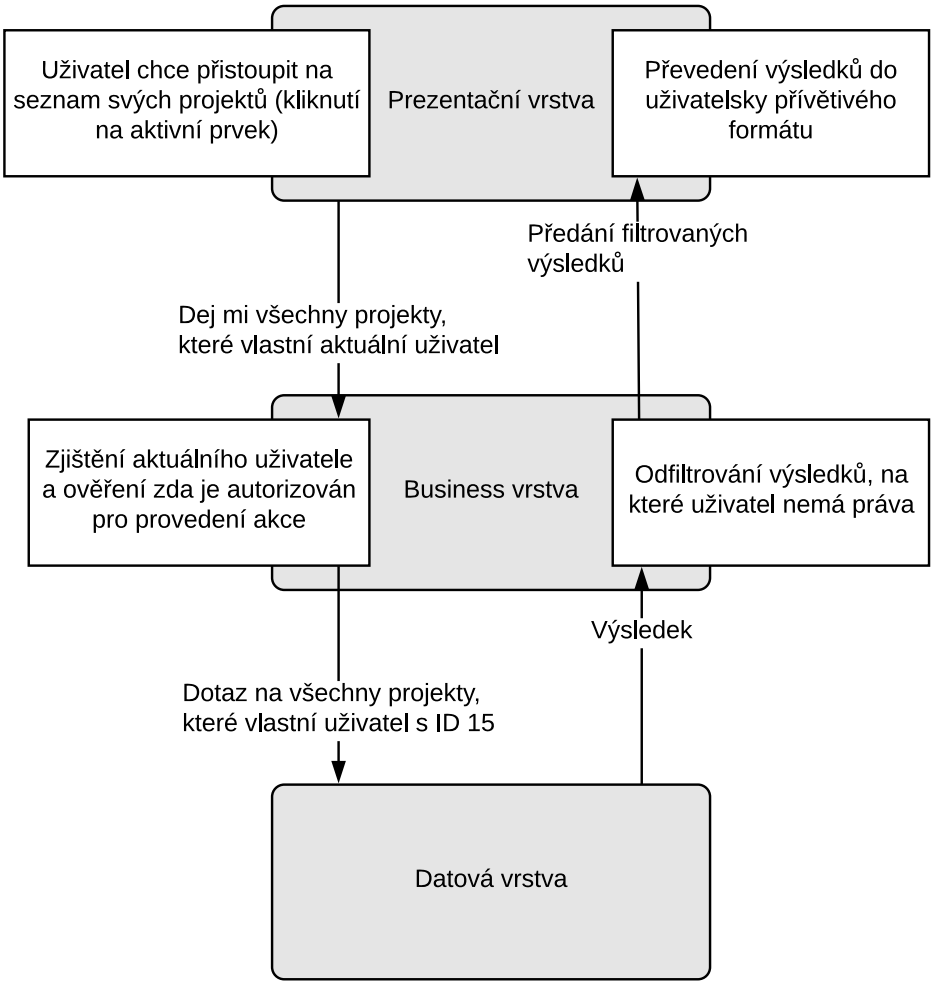
„*Model třívrstvé architektury, který je […], segmentuje aplikační komponenty do tří úrovní služeb. Tyto úrovně nemusí nutně korespondovat s fyzickými lokacemi počítačů na síti, ale spíše s logickými vrstvami aplikace. Jak jsou části aplikace distribuovány ve fyzické topologii lze měnit v závislosti na systémových požadavcích.*“ [43] (překlad dle autora) Výše zmíněné úrovně služeb jsou:

- Prezentační vrstva – Stará se o prezentaci dat uživateli a delegování uživatelských požadavků na business vrstvu.

- Business vrstva – Tato vrstva se stará o veškerou doménově specifickou logiku aplikace.
- Datová vrstva – Skládá se z komponent, které poskytují přístup k datům a z datových úložišť samotných.

3.5.2.1 Komunikace vrstev

V třívrstvé architektuře mohou jednotlivé úrovně komunikovat pouze s úrovní o jednu nižší. Tedy Prezentační vrstva komunikuje s business vrstvou a ta komunikuje s vrstvou datovou. [44] Na obrázku 3.3 je pro lepší představu zobrazena možná interakce jednotlivých vrstev.



Obrázek 3.3: Třívrstvá architektura

3.5.2.2 Vztah k využitému rozdělení

Tento projekt, jak již bylo zmíněno, využívá rozdělení inspirované právě třívrstvou architekturou. Systém není takového rozsahu, aby bylo třeba využít přímo třívrstvé architektury, která se zaměřuje hlavně na možnost oddělení jednotlivých vrstev a jejich distribuci na síti.

Rozdělení je inspirováno využitím tří stejných vrstev (prezentační, business a datové). Na rozdíl od běžné třívrstvé architektury se však nezaměřuje na striktní oddělení těchto vrstev (například pomocí API), ale pouze na zavedení logické struktury mezi aplikační komponenty. Vrstvy jsou od sebe odděleny pouze aplikačním kódem realizujícím rozhraní mezi těmito vrstvami. Není tedy možné jednotlivé vrstvy provozovat odděleně.

Komunikace vrstev Komunikace mezi vrstvami probíhá, až na jeden rozdíl, stejně jako ve třívrstvé architektuře. Tento rozdíl je v tom, že vrstva prezentační může také přímo komunikovat s vrstvou datovou.

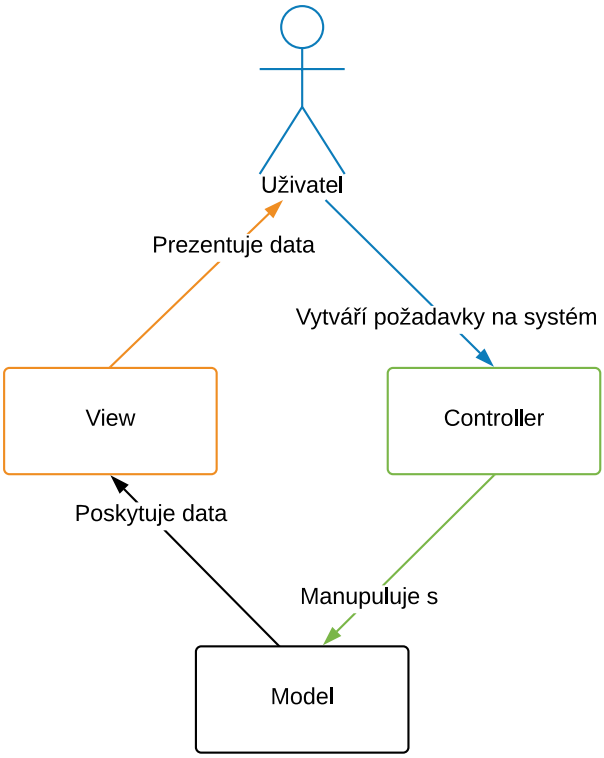
Prezentační vrstva Prezentační vrstva se v tomto systému skládá z uživatelského rozhraní a tříd, které reagují na uživatelský vstup. Uživatelské rozhraní je složeno z HTML stránek, Kaskádových stylů (CSS) a skriptů, běžících přímo v klientských prohlížečích. Toto obstarávají View objekty (viz. popis MVC v podsekci 3.5.3). Do této vrstvy spadají také Controllery, neboť reagují na uživatelský vstup a mohou tedy být považovány za součást uživatelského rozhraní. Controllery však neobsahují žádnou aplikační logiku, pouze interpretují požadavky zasílané na systém skrze uživatelské rozhraní. Tato vrstva komunikuje jak s business vrstvou, tak s vrstvou datovou.

Business vrstva Business vrstva se stará o veškerou doménovou logiku aplikace. Do této vrstvy spadají třídy, které se starají o autentizaci, autorizaci a doménově specifické operace jako je spuštění výkonných funkcí robotů. V této vrstvě je často využito návrhového vzoru Fasáda. Každý Model reprezentující doménovou entitu by měl mít přiřazenu jednu Fasádu, která zapouzdří business operace spojené s tímto Modelem. Tato vrstva využívá vrstvy datové pro získání potřebných informací a dat z datového úložiště.

Datová vrstva Datová vrstva se stará o přístup k datovému úložišti (databáze). Nabízí rozhraní pro dotazy na databázi a také se stará o ukládání, úpravu a mazání záznamů. Tato vrstva je v tomto systému realizována za využití ORM Doctrine. Pro získávání dat z úložiště jsou primárně využívány entitní repositáře, které využívají návrhového vzoru Repository.

3.5.3 Model View Controller

Návrhový vzor Model View Controller je využíván pro oddělení dat od způsobu jakým jsou prezentována. Na diagramu 3.4 je zobrazen vztah jednotlivých typů objektů. Dle [45] MVC rozděluje objekty do tří různých typů: Model, View a Controller.



Obrázek 3.4: Model View Controller

Obrázek je inspirován článkem [45].

Model „*Modely drží data a definují logické operace pro manipulaci s těmito daty.*“ [45] (překlad dle autora) Model je možné si představit jako kontejner pro určité informace. Tento kontejner může také specifikovat jak lze s těmito informacemi manipulovat. Modelem může být například uživatel, který obsahuje jméno a příjmení a definuje přístupové (getName) a modifikační (setName) metody pro manipulaci s těmito daty.

View „*View objekty reprezentují viditelné části uživatelské rozhraní, například panel či tlačítko.*“ [45] (překlad dle autora) View objektů je využito k prezentaci dat uživateli. Mohou například definovat jaké atributy Modelu budou

zobrazeny a jaké ne. Dále popisuje jakým způsobem budou data zobrazena (umístění na obrazovce, velikost písma, barva, …).

Controller Nejdůležitější částí tohoto vzoru je Controller. Controller obsahuje aplikační logiku a reaguje na uživatelský vstup. Slouží jako „*mediátor mezi Modely a View objekty*“ [45] (překlad dle autora). Tedy interpretuje uživatelský vstup a na jeho základě předává data mezi Modely a View objekty.

3.5.3.1 Vztah k navrhovanému systému

Aplikace, jak je popsáno výše, je rozdělena do tří vrstev. View a Controller spadají do vrstvy prezentační. Není však úplně jasné, kde se v těchto vrstvách nachází Model.

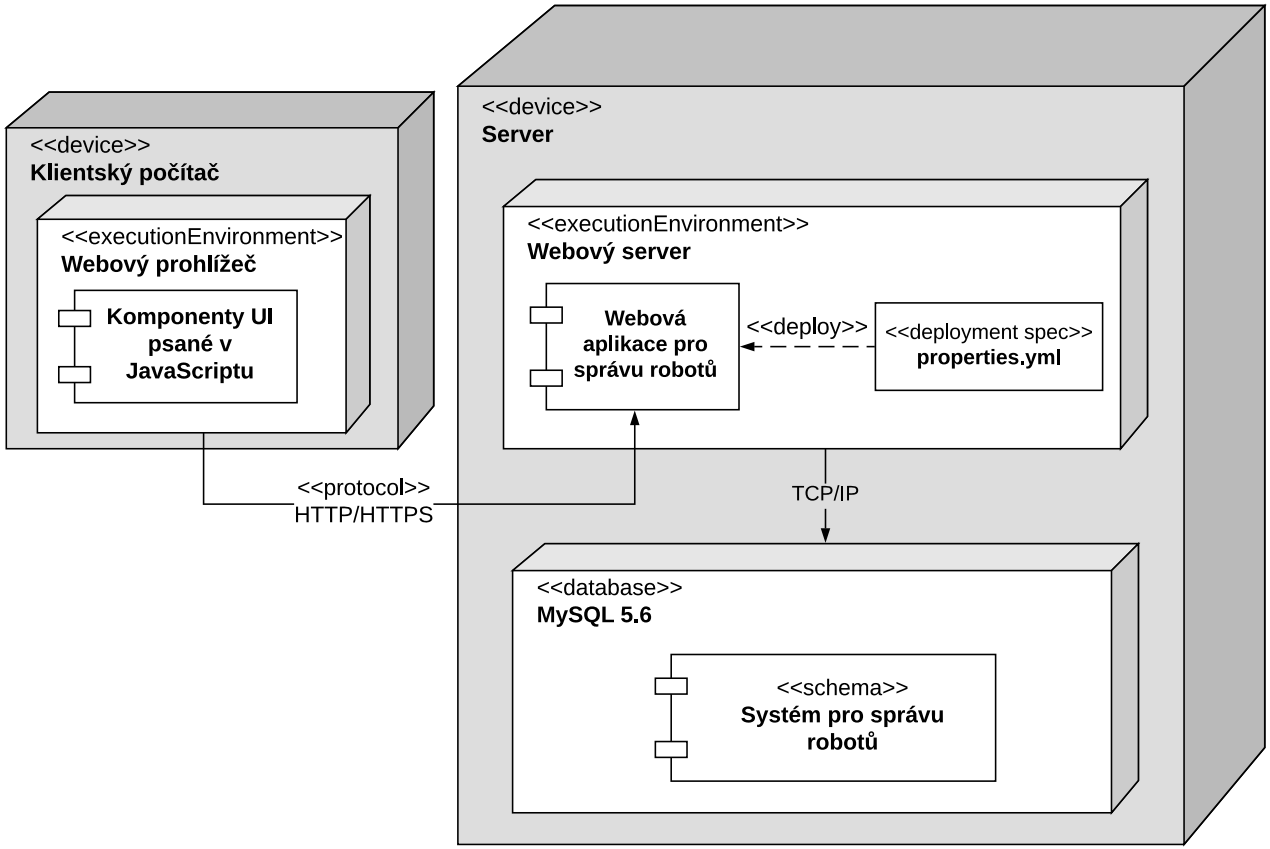
View a Controller – Prezentační vrstva View vrstva je v systému realizována za využití šablonovacího systému Twig. V Symfony Controllery rozhodují o tom, jaké View bude využito na základě uživatelského požadavku. Dále také využívají business a datové vrstvy (viz. podsekce 3.5.2) pro získání potřebných dat nebo pro spuštění vyžádané funkcionality. Controllery tedy v tomto systému neobsahují žádnou business logiku, o tuto logiku se stará nižší vrstva.

Model Odpovídajícími objekty v Symfony jsou Entity, které navíc také definují metadata pro mapování na datové úložiště. Model je v systému využíván napříč všemi vrstvami, což porušuje striktní oddělení jednotlivých vrstev systému. Prezentační vrstva využívá Model pro prezentaci dat (operuje přímo nad Modelem). Business vrstva provádí nad Modelem operace jako je tvorba nového Modelu či úprava již existujícího. A vrstva datová Model využívá pro mapování informací na databázové tabulky (při získávání dat z databázového stroje mapuje získaná data na Model a obráceně).

Model je v tomto systému zařazen do vrstvy datové a je povolena komunikace mezi vrstvou prezentační a datovou. Business vrstva může Model využít bez problému, neboť je vrstvou o jedna vyšší od datové.

3.5.4 Nasazení

Pro lepší orientaci ohledně rozložení systému je také v této sekci uveden diagram nasazení. Tento diagram může také sloužit jako grafická pomůcka při integraci systému, neboť popisuje fyzické rozmístění systémových komponent. Diagram je k nalezení na obrázku 3.5.



Obrázek 3.5: Diagram nasazení

Kapitola 4

Implementace

4.1 Struktura aplikace a implementační detaily

Tato sekce se zabývá strukturou celé webové aplikace a detaily implementace. Nejprve se věnuje detailněji zvoleným technologiím a poté popisuje strukturu webové aplikace. Informace v této sekci lze také využít jako velice hrubou dokumentaci aplikace.

Velmi důležitou podsekcí této sekce je 4.1.4.2, která popisuje jaké části aplikace byly tvořeny autorem této práce a jaké naopak nebyly.

4.1.1 Jazyk PHP

Celá webová aplikace, včetně části front-endu je psána v jazyce PHP 5.6. Verze 5.6 byla zvolena, protože server, na kterém má být systém nasazen, již tuto verzi PHP obsahuje a je napojena na webový server Apache. Původně měla být webová aplikace napsána ve verzi 7.1, ale z důvodu lepší kompatibility bylo využito právě této starší verze. Při implementaci, jak je již zmíněno v sekci 2.6.1, bylo využito popisných pojmenování a kódová dokumentace se vyskytuje na místech, kde je to nezbytně nutné. Externí dokumentace aplikace a celého systému bude dodána zvlášť.

4.1.2 Nástroje

Pro správu knihoven byl využit nástroj „composer“ (http://getcomposer.org/), který je již při použití programovacího jazyka PHP standardem. Jak již bylo zmíněno v sekci 3.3, pro verzování projektu bylo využito nástroje Git. Nově je také využíváno webového rozhraní systému GitLab pro sledování chyb a problémů. Tento systém dovoluje týmům vytvářet záznamy o problémech a tyto záznamy poté diskutovat a řešit. Také podporuje odkázání se na problém skrze id ve zprávě k zápisu (commit) [46].

4.1.3 Symfony

Framework Symfony je složený z několika komponent (viz. slovník: Symfony Components). Tyto komponenty lze využít jako samostatné knihovny i v projektech, které nevyužívají kompletního frameworku. Symfony lze samozřejmě také dále libovolně rozšiřovat o další vybrané knihovny. Tyto knihovny mohou například řešit překlad uživatelského rozhraní nebo zpracování nahraných souborů.

Využitá verze frameworku je 3.4, neboť Symfony 4.0 vyžaduje PHP 7.1.3 a vyšší [47]. Dále je tato verze také zvolena proto, že je již dlouho podporována a verze 4.0 byla ještě poměrně novou v době, kdy započaly práce na projektu.

4.1.3.1 Best practices

Po aplikacích, které využívají frameworku Symfony, není vyžadována žádná pevná struktura, kterou je nutné dodržovat (většinu komponent lze nakonfigurovat dle potřeby). Je však dobré držet se určitých standardů, neboť tyto standardy pomohou s udržovatelností systému v případě, kdy na projektu nebude pracovat původní vývojářský tým. Při tvorbě aplikace, kterou se zabývá tato práce, byl kladen důraz na dodržení „best practices“ popisovaných ve článku [48].

4.1.3.2 Konfigurace

Komponenty a knihovny, které aplikace postavená na Symfony využívá, lze nakonfigurovat skrze konfigurační soubory. Pro tvorbu těchto souborů je doporučeno využít jazyka YAML Ain't Markup Language (YAML) [48]. Dále lze také využít parametrů pro konfiguraci aplikace samotné. Platformě specifické parametry, či parametry, které je potřeba měnit při nasazení aplikace, jsou standardně umístěny v souboru *parameters.yml.dist*. V sekci 4.1.4 je popsána lokace aplikačních souborů (včetně souborů konfiguračních).

Soubor s příponou *.dist* slouží pouze jako definice parametrů, které je možné nastavit. Z bezpečnostních důvodů tento soubor obsahuje pouze ukázkové nastavení, které je nutno při instalaci aplikace změnit. Pokud je pro instalaci aplikace využito nástroje Composer, je automaticky vygenerován soubor *parameters.yml*, který je načten aplikací. V případě, že není využito nástroje Composer (nedoporučováno!), je potřeba tento soubor vytvořit ručně.

4.1.3.3 Služby

Službou je v Symfony označována jakákoliv třída, která poskytuje určitou funkcionalitu a lze ji využívat napříč celou aplikací. Tato služba může například poskytovat funkce pro čtení/zápis dat ze/do souboru.

Služby mohou být nakonfigurovány skrze konfigurační soubory aplikace. Velkou výhodou konfigurovaných služeb je, že lze podle potřeb vyměnit jejich

implementaci v celé aplikaci. Tedy pokud je využito služby, která načítá data ze souborů a je potřeba tyto data nově ukládat v databázi, stačí pouze vyměnit implementaci této služby a vše bude fungovat jak má. Předpokladem je abstrakce příslušných aplikačních tříd.

Využití služeb v tomto projektu Služby jsou prakticky základním kamenem této aplikace. Většina služeb však není explicitně konfigurována a využívá automatického načítání. Dále je také využito „značkování“ za účelem získání veškerých implementací určitého rozhraní či abstraktní třídy.

Značkování „*Značky nijak neovlivňují funkcionalitu vašich služeb. Ale pokud budete chtít, můžete požádat tvůrce kontejneru o list všech služeb, které byly označeny specifickou značkou.*“ [49] Tohoto mechanizmu je využito pro usnadnění rozšiřitelnosti aplikace. V určitých případech stačí pro rozšíření funkcionality implementovat rozhraní, které je označkované a nová třída je automaticky registrována a přiřazena příslušné službě.

4.1.4 Realizace aplikace

Nejdříve je důležité získat přehled ohledně celkového rozložení aplikačních souborů. Logická struktura aplikace je poté shodná s umístěním zdrojových souborů. Dále je velice důležité specifikovat jaké části aplikace jsou realizovány autorem této práce a které ne. Tímto se také zabývá tato sekce.

4.1.4.1 Souborová struktura aplikace

Souborová struktura reprezentuje lokaci jednotlivých částí aplikace na souborovém systému. Při tvorbě aplikace, jak již bylo zmíněno, byl kladen důraz na dodržování standardů frameworku Symfony. Toto platí i při volbě rozložení aplikace. Doporučená struktura dle [50] je zobrazena na obrázku 4.1.

```
/
├── app ..... aplikační konfigurace, šablony a překlady
├── bin ..... spustitelné soubory
├── src ..... PHP kód projektu
├── tests ..... automatické testy (například Unit testy)
├── var ..... vygenerované soubory (cache, logy, …)
├── vendor ..... knihovny a komponenty třetích stran
└── web ..... kořenová složka webové aplikace
```

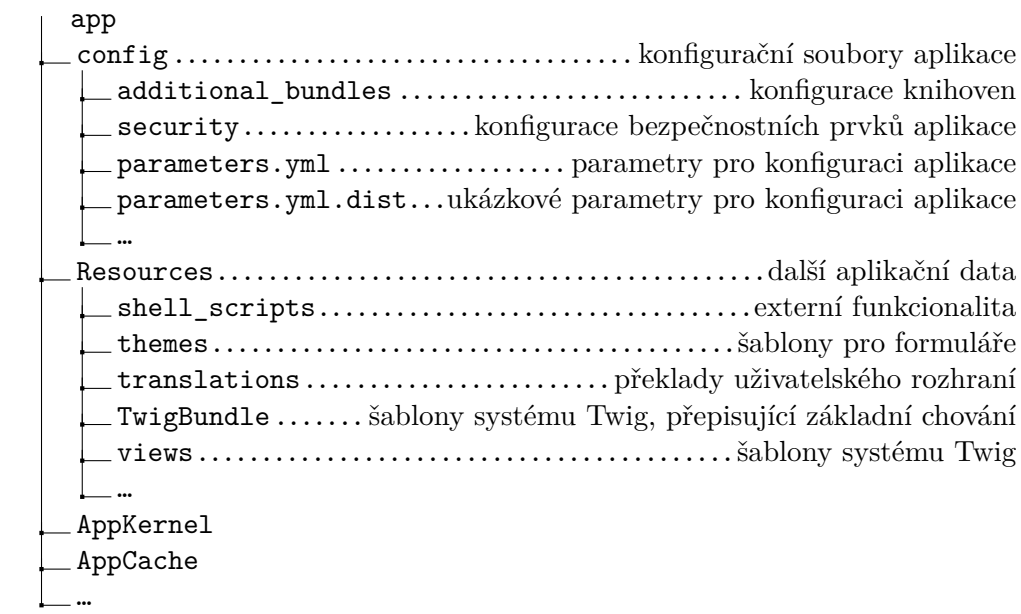
Obrázek 4.1: Doporučená souborová struktura pro aplikace využívající framework Symfony

Tato doporučená struktura byla dodržena a je tedy shodná se strukturou aplikace, které se věnuje tato práce. Dále také přibyla složka *web-dev*, která ob-

sahuje zdrojové kódy pro kompilaci Kaskádových stylů a zdrojových souborů psaných v jazyce JavaScript.

Dále jsou detailně popsány dvě nejdůležitější složky v této struktuře. Jsou však vynechány složky či soubory, které není nutné vyobrazovat v závislosti na využití těchto vizualizací. Tři tečky tedy znamenají blíže neurčený počet souborů nebo složek.

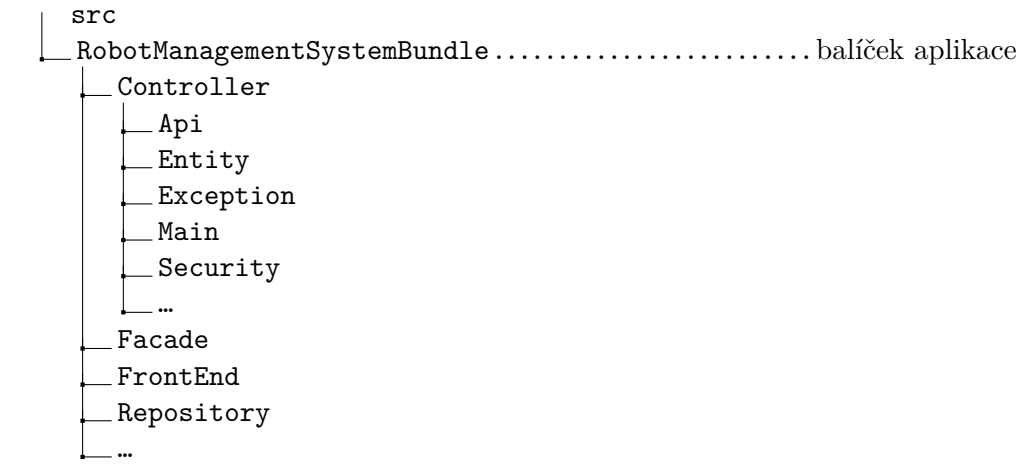
Složka *app* a konfigurace aplikace První z těchto složek je složka *app*. Detail této složky je důležitý, neboť obsahuje konfiguraci celé aplikace. Struktura složky je k nalezení na obrázku 4.2.



Obrázek 4.2: Struktura složky *app*

Veškerá konfigurace kromě parametrů v souboru *parameters.yml* je považována za interní a tedy není třeba ji měnit. Některé části interní konfigurace budou během dalšího vývoje zpřístupněny přes administraci aplikace. Do té doby by neměly být upravovány bez předešlé konzultace s vývojovým týmem.

Zdrojové kódy aplikace Druhou velice důležitou složkou je složka *src*, která obsahuje veškeré zdrojové kódy psané v jazyce PHP. Struktura této složky je uvedena z důvodu, že pomáhá vizualizovat oddělení jednotlivých částí aplikace. Na tuto vizualizaci se dále odkazuje sekce 4.1.4.2. Struktura této složky je zobrazena na obrázku 4.3.



Obrázek 4.3: Struktura složky *src*

Vyznačeny jsou jen složky potřebné k jasnému rozdělení aplikace mezi front-end a back-end.

4.1.4.2 Spolupráce při implementaci

Na realizaci webové aplikace, jak již bylo zmíněno v sekci 3.3 na straně 44, se aktuálně podílí dva vývojáři. Každý z vývojářů je zodpovědný za část webové aplikace. Za front-end je zodpovědný Erik Zatloukal (student ČVUT FIT v Praze) a za back-end je zodpovědný autor této práce (Dan Zatloukal). Je tedy nutné jasně definovat, které zdrojové soubory aplikace spadají do jaké části.

Front-end Tato část nezasahuje do business logiky aplikace, pouze se stará o prezentaci dat a interpretaci uživatelských požadavků. Následující seznam definuje, které zdrojové soubory spadají do front-end části aplikace:

- Neabstraktní třídy ve složce:
  src/RobotManagementSystemBundle/Controller/Entity.
- Veškeré zdrojové soubory ve složce:
  src/RobotManagementSystemBundle/FrontEnd.
- Veškeré soubory ve složce: *src/web-dev*.
- Šablony ve složkách:
  - *app/Resources/themes*,
  - *app/Resources/TwigBundle*,
  - *app/Resources/views*.
- Veškeré překlady uživatelského rozhraní: *app/Resources/translations*.

- Soubory ve složkách:
  - *web/images*,
  - *web/dist*.

Back-end Ostatní zdrojové soubory, včetně konfigurace a dodatečných dat jsou součástí back-endu aplikace. Back-end je od front-end části oddělen Fasádami a repositáři (viz. slovník: repositáře). Součástí back-endu bylo také připravit abstraktní Controllery, které přidávají standardní funkcionalitu a definují jak mají určité metody vypadat. V neposlední řadě byla také součástí back-endu příprava aplikace pro implementaci API.

4.1.5 Externí funkcionalita

Aplikace využívá externí funkcionality v podobě spustitelných souborů, přesněji skriptů psaných v jazyce Shell, pro realizaci výkonných funkcí zařízení. Tyto výkonné funkce obsahují například:

- Přesun souborů na zařízení
- Spuštění souboru na zařízení
- Inicializace zařízení (vložení do databáze)
- Záloha zařízení

4.1.5.1 Napojení na aplikaci

Skripty realizující výkonné funkce zařízení jsou v základu umístěny ve složce: *app/Resources/shell_scripts/robot_management_system* (lze nakonfigurovat skrze parameter *app.scripts.location* v souboru *parameters.yml*). V aplikaci je zhotoven prototyp API, které umožňuje spustit jednotlivé akce.

Třídy (Controllery) realizující toto API jsou k nalezení ve složce: *src/RobotManagementSystemBundle/Controller/Api/Internal* a pouze určují které business operace se mají spustit. O operace samotné se starají Fasády. V aplikaci je každý skript reprezentován objektem, kterému lze nastavit parametry a tento objekt se stará o spuštění skriptu, který reprezentuje. Tento objekt také kontroluje, zda je skript spustitelný, zda obsahuje správná metadata jako je mime type a zda je to běžný soubor.

Pro spuštění skriptu je využito standardních systémových operací. Veškerý uživatelský vstup je opatřen příslušnými sekvencemi a není tedy možné, aby na serveru, kde jsou skripty uloženy, škodil. Pro ukázku, jakým způsobem jsou tyto skripty využívány ve Fasádách, je přiložena část kódu, která realizuje inicializaci robota, viz. listing 1.

```php
/**
* @param string $robotConnectionInfo
* @param string $newDeviceName
* @return \RobotManagementSystemBundle\ScriptProvider\ExecutionResult
*/
public function initializeRobotViaScript($robotConnectionInfo, $newDeviceName)
{
    $connection = $this->entityManager->getConnection();
    $host = $connection->getHost();
    $user = $connection->getUsername();
    $password = $connection->getPassword();

    $script = $this->deviceShellScriptProvider->getScript(
        DeviceShellScriptProvider::INITIALIZE_ARDUINO_DEVICE
    );

    DeviceShellScriptProvider::setMysqlLoginFlags(
        $script,
        $host,
        $user,
        $password
    );
    $script->setFlagValue('i', $robotConnectionInfo);
    $script->setFlagValue('n', $newDeviceName);

    return $script->executeWithSafeOutput([$password]);
}
```

Listing 1: Ukázka kódu: inicializace robota skrze shell skript

4.2 Autentizace a autorizace

Jedním z požadavků na systém byla možnost přihlášení skrze Shibboleth SSO. Napojení na tento autentizační systém bylo díky zvoleným technologiím poměrně jednoduché. Složitější však je realizace autorizačního systému, který je také jedním z požadavků na systém. Autorizační systém, jak již bylo zmíněno v sekci 2.2, musí podporovat jednotné přidělování a odebírání práv na zabezpečené zdroje. Na tento autorizační systém se také tato sekce odkazuje pod názvem „dynamická autorizace“. Tato sekce se nejprve zabývá implementací autentizačních metod a poté se věnuje implementaci autorizačního systému.

4.2.1 Autentizace

Framework Symfony v základu podporuje pouze jednoduché autentizační způsoby jako je Hyper Text Transfer Protocol (HTTP) autentizace či interní login formulář. Pro tento systém bylo jako primární autentizační způsob zvoleno přihlášení skrze Shibboleth SSO. Dále však byla přidána také možnost přihlášení přes interní login formulář pro případ, kdy je potřeba vytvořit lokální účet, například pro administrátora. V poslední řadě byla do systému integro-

vána také autentizace API skrze Json Web Tokens (JWT). Pro login a získání příslušného „žetonu“ (viz. slovník: token) je v aplikaci připraven login formulář, který přihlásí pouze uživatele, kteří mají práva pro přístup na API (nyní musejí vlastnit příslušnou roli).

4.2.1.1 Shibboleth a Single Sing-On

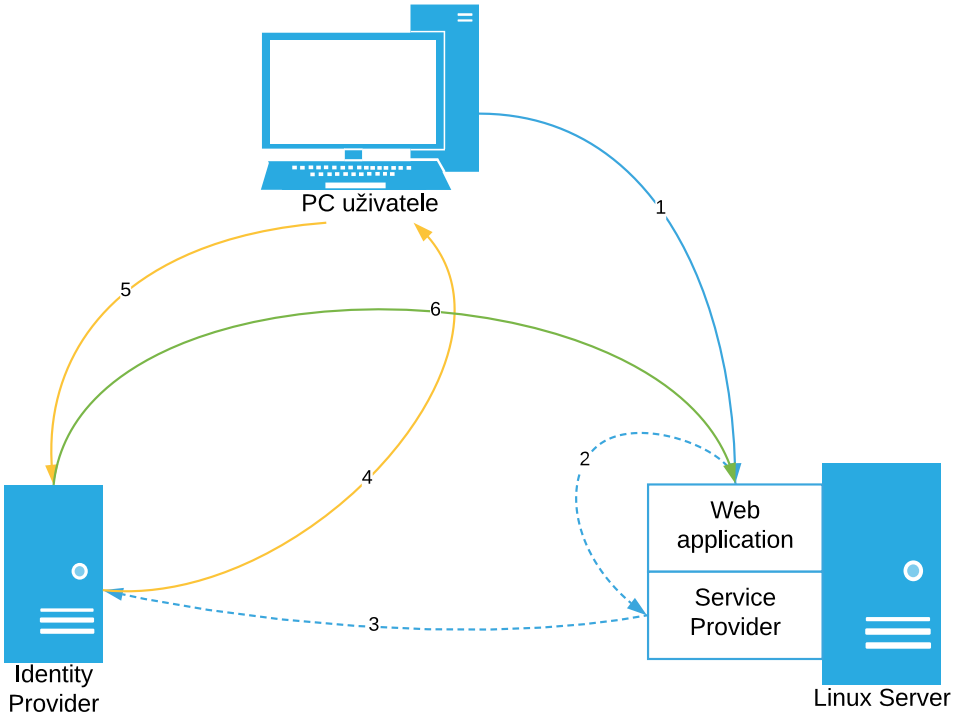
„Single Sign-On (SSO) je autentizační služba, která dovoluje uživateli využít jedny přihlašovací údaje k přístupu do více aplikací.“ [51] (překlad dle autora) Shibboleth je open source projekt, který poskytuje SSO funkcionalitu [52]. Tento projekt je velice rozsáhlý a proto jsou zde uvedeny pouze nejdůležitější informace, které je nutné znát pro pochopení toho, jak Shibboleth funguje. Shibboleth se skládá ze dvou hlavních částí: Identity Provider (IdP) a Service Provider (SP). Na obrázku 4.4 je znázorněna komunikace uživatelského prohlížeče s SP a IdP (vztažená ke způsobu jakým je Shibboleth využit v tomto systému).

Zprovoznění systému Shibboleth není součástí této práce. Příslušní SP je již nainstalován a nakonfigurován na serveru, na kterém systém bude provozován.

Identity Provider *„IdP autentizuje uživatele a poté přeposílá autentizační autentizační odpověď a uživatele zpět na SP.“* [53] (překlad dle autora) Tato část se tedy stará o samotné přihlášení uživatele. Proto, aby mohl IdP uživatele přihlásit, musí využít nějakého autentizačního mechanismu. Tímto mechanismem může být například autentizace skrze LDAP nebo dokonce i autentizace za využití IP adres [54]. Na jeden IdP může být napojeno více SP a tím je realizováno jednotné přihlášení. IdP je běžná Java web aplikace založená na Servlet 2.4 (záleží na verzi Shibboleth, toto platí pro verzi Shibboleth 2) a může tedy běžet na jakémkoliv kompatibilním kontejneru, jako je například Apache Tomcat [53].

Service Provider SP je část systému Shibboleth, která umožňuje napojení určité aplikace na IdP. Tuto část lze instalovat přímo do webového serveru (např. Apache) pomocí rozšíření (pro Apache *mod_shib*). SP se stará o komunikaci s IdP. Pokud se chce uživatel přihlásit za využití Shibboleth SSO, požádá SP o přihlášení, ten uživatele přesměruje na příslušný IdP, uživatel zadá potřebné údaje (pokud je to třeba) a IdP přesměruje uživatele zpět na zvolenou adresu. [53]

Ověření autentizovaného uživatele Pro ověření, zda je uživatel autentizovaný, je možno využít více způsobů. Shibboleth SP může data o přihlášeném uživateli poskytnout například skrze proměnné prostředí nebo dokonce za využití hlaviček požadavku (může být nebezpečné).



Obrázek 4.4: Přihlášení pomocí Shibboleth SSO ve webové aplikaci

Obrázek je inspirován článkem [53].

1. Uživatel přistoupí k zabezpečenému zdroji.
2. Aplikace zkontroluje, zda je uživatel přihlášený. Pokud není, přesměruje ho na SP.
3. SP přesměruje uživatele na příslušný IdP.
4. IdP si vyžádá přihlašovací údaje uživatele.
5. Uživatel zadá údaje.
6. IdP informuje SP o úspěšné autentizaci a uživatel je přesměrován zpět na webovou aplikaci.

V tomto projektu je využito proměnné *REMOTE_USER*. Tato proměnná je nastavena přímo webovým serverem a aplikaci je předána skrze Common Gateway Interface (CGI), proto se lze na hodnotu této proměnné spolehnout a metoda je tedy bezpečná. Na straně SP je potřeba nakonfigurovat, která hodnota bude do této proměnné vložena, neboť SP má po přihlášení uživatele

k dispozici více metadat. V tomto projektu je v této proměnné předáváno uživatelské jméno (např. „zatlodan“).

4.2.1.2 Autentizace v Symfony

Než se sekce bude věnovat integraci jednotlivých typů přihlášení, je dobré zmínit jakým způsobem Symfony přistupuje k této problematice. Symfony využívá tzv. „firewall“, které zabezpečují aplikaci. Každá firewall zabezpečuje určitou část aplikace. Jakou část aplikace firewall chrání, lze specifikovat pomocí regulárních výrazů (specifikují jaké adresy jsou chráněny). [55]

Je také potřeba specifikovat, jakým způsobem se bude uživatel autentizovat pod určitou firewall. Každá firewall může obsahovat více způsobů autentizace (např. skrze HTTP nebo interní login formulář). Symfony také dovoluje implementovat vlastní způsoby autentizace. [55]

4.2.1.3 Integrace autentizace skrze Shibboleth

Jak již bylo zmíněno v sekci 4.2.1.1, pro ověření zda je uživatel přihlášený je využíváno proměnné *REMOTE_USER*. Tato proměnná obsahuje uživatelské jméno přihlášeného uživatele. Aplikace tedy ví, že pokud je tato proměnná nastavena, je uživatel přihlášen. Stačí tedy implementovat třídu, která určí zda je uživatel přihlášen či ne (v podstatě).

Aplikační logika, která kontroluje, zda je uživatel autentizován skrze Shibboleth a stará se i o jeho odhlášení, je realizována za využití *AuthenticatorInterface*. Třídy, které implementují rozhraní *AuthenticatorInterface*, mohou být využity jako jedna z metod autentizace pro zvolený firewall. [56]

Třída implementující tuto logiku je *ShibbolethAuthenticator* a je k nalezení ve složce *Security/GuardAuthenticator* zdrojových kódů. Tato třída také při požadavku na odhlášení přesměruje uživatele na SP, který provede odhlášení uživatele.

Uživatelské účty Problém je v tom, že po přihlášení sice systém zná veškerá potřebná data pro identifikaci uživatele, ale nemá způsob, jakým by se na něj interně odkazoval (uživatel není nikde evidován). Proto při prvním přihlášení skrze systém Shibboleth je uživateli vytvořen účet v databázi. Protože po přihlášení nemá aplikace k dispozici uživatelské heslo, je jeho heslo vynulováno.

Takto vytvořený účet má nastavenou dobu platnosti na jeden měsíc (lze nakonfigurovat skrze parameter *account_expiration_time_from_last_refresh*). Pokud se uživatel přihlásí do neplatného účtu, je při přihlášení tento účet obnoven. Obnovení účtu znamená, že jsou znovu načtena a uložena veškerá metadata předaná systémem Shibboleth (tedy je to tedy aktualizace účtu). Po obnovení je prodloužena doba platnosti účtu o další měsíc.

Pokud je uživatelský účet odstraněn u IdP, webová aplikace se o tom nedozví a proto je vhodné účty, které jsou nevalidní déle jak půl roku, mazat.

Toto aktuálně není v aplikaci řešeno, je však možné promazávání účtů implementovat v budoucnu (již bylo probíráno se zadavatelem).

Mapování atributů Veškeré informace ohledně mapovaných atributů byly získávány z portálu wiki.cvut.cz a také z portálu rozvoj.fit.cvut.cz. Systém mapuje uživatelské skupiny a role. Konfigurace mapování je k nalezení v souboru *security/shibboleth.yml* ve složce a aplikační konfigurací (viz. sekce 4.1.4 na straně 61). Mapování atributů bude později přesunuto do databáze a bude obsahovat uživatelské rozhraní pro případné úpravy.

4.2.1.4 Autentizace skrze interní formulář

Aplikace podporuje přihlášení skrze standardní přihlašovací formulář. Tento způsob autentizace byl přidán pro případ, kdy je potřeba přihlásit uživatele, který nemá založený účet na ČVUT FIT. V aplikaci je možné se přihlásit buď přes tento interní formulář anebo skrze systém Shibboleth.

Účty vytvořené skrze Shibboleth autentizaci Problémem samozřejmě jsou účty, které byly vytvořeny skrze Shibboleth SSO. Aplikace nemá přístup k heslům od těchto účtů a proto jim, jak bylo zmíněno výše, při vytvoření heslo vynuluje. Toto by však zapříčinilo, že kdokoliv, kdo by znal uživatelské jméno, by mohl využít interního formuláře pro přihlášení k tomuto účtu.

Problém přihlášení pouze na základě uživatelského jména (bez nutnosti zadání hesla) je řešený rozšířením základní autentizační metody pro interní formuláře. Rozšíření zakazuje přihlášení přes interní formulář účtům, které nemají nastavené heslo. Pokud si však uživatel v systému nastaví heslo, může pro přihlášení využít i interního formuláře. Implementace výše zmíněného rozšíření je k nalezení ve třídě *ExtendedFormLoginAuthenticator*, která se nachází ve složce se zdrojovými kódy a přesněji v podsložce *Security/GuardAuthenticator*.

4.2.1.5 API autentizace

Protože systém je připravený pro integraci API, byla do systému přidána možnost autentizace skrze JWT. Tato autentizace je využita pro zabezpečení veřejného API aplikace. Implementace API, ani jeho příprava nebyly uživatelskými požadavky na systém, avšak je dobré mít aplikaci připravenu pro případné rozšíření. Toto API může být například využito client side skripty jako je JavaScript, který bude jistě využit při dalším vývoji.

4.2.2 Autorizace

Tato sekce se zabývá nejrozsáhlejší částí systému, kterou je autorizace. Protože aplikační třídy, které zajišťují autorizaci jsou poměrně složité, bude se tato

sekce věnovat z velké části logické struktuře nežli implementačním detailům. V důležitých částech však tyto detaily nebudou vynechány.

Jak již bylo zmíněno v sekci 3.4.4.1 na straně 49, dynamický autorizační systém se skládá z následujících částí:

- Zabezpečený zdroj
- Resource Accessor (objekt, který přistupuje ke zdroji)
- Typ zabezpečeného zdroje
- Záznam pro řízení přístupu, neboli také Access Control Entry (ACE)

Těmito datovými strukturami se již zabývala výše zmíněná sekce a není tedy třeba je znovu popisovat. Tato sekce se věnuje jejich využitím v aplikaci.

4.2.2.1 Aplikační nezávislost autorizace

Výše zmíněné struktury se nacházejí v databázi systému a jsou tedy využitelné i dalšími napojenými aplikacemi. Veškerá pravidla, která Votery využívají pro rozhodnutí zda má uživatel přístup či ne, jsou definovány daty uloženými v databázi. Toto znamená, že aplikace nepřidává žádnou „hard coded“ logiku do autorizačních pravidel.

Jedinou výjimkou jsou samozřejmě objekty, které nevyužívají dynamické autorizace (nevyužívají databázových struktur pro zabezpečení dat). Tyto objekty mají pevně daný přístup určený rolí v systému, toto bude při vývoji dále upraveno podle potřeb zadavatele.

4.2.2.2 Symfony a autorizace

Symfony využívá pro základní autorizaci uživatelů interní role. Lze tedy zamezit přístup na určité stránky na základě rolí, které uživatel vlastní.

Pro zamezení přístupu na určité objekty lze využít tzv. „Voter“. Tento koncept je velice jednoduchý, ale také univerzální. Pokud je potřeba zamezit přístup na určitý objekt, aplikace se zeptá Voterů, jestli má aktuální uživatel zvolená práva. Každý Voter má možnost definovat, zda může hlasovat na daném typu objektu s danými právy. Pokud Voter může hlasovat na typu objektu a podporuje vyžádaná práva, provede logiku potřebnou pro zjištění, zda má uživatel přístup a vrátí *true*, pokud je uživatel autorizován.

Je více způsobů jak rozhodnout, zda je uživatel autorizován. První způsob je jednotné hlasování, tedy Votery se musí shodnout všechny, že uživatel je autorizován. Další je většinové hlasování, tedy větší polovina Voterů musí hlasovat kladně. Posledním způsobem je povolení na základě alespoň jednoho kladného hlasu. V konfiguraci lze zvolit jaký způsob autorizace je využit.

4.2.2.3 Zabezpečené zdroje

Základem autorizace v této aplikaci jsou právě Votery. V aplikaci je implementován jeden Voter, který umí hlasovat na všech zabezpečených zdrojích. Nedoporučuje se tento Voter rozšiřovat (mohlo by docházet k duplicitním dotazům na databázi). Pro zjištění zda je uživatel autorizován je využito databázového dotazu. Tento Voter navíc, kromě tohoto dotazu také využívá přepisujících pravidel, například pro administrátora systému.

Jakákoliv Entita může být v systému zabezpečeným zdrojem. V aplikaci to znamená pouze implementovat několik poměrně jednoduchých rozhraní nebo vytvořit specializaci abstraktních tříd. Na úrovni databáze je potřeba přidat entitě (tabulce) vazbu na zabezpečený zdroj.

Poté, co jsou implementována všechna potřebná rozhraní či specializace abstraktních tříd, není nutná žádná další konfigurace. Entita je automaticky aplikací považována za zabezpečený zdroj a je možné využít uživatelského rozhraní pro přidání práv. Toto není úplně pravdivé tvrzení. Aby vše fungovalo tak, jak má, je nejdříve ještě potřeba přidat nový typ zdroje do databáze. Následující seznam popisuje co je třeba implementovat za účelem převedení běžné Entity na zabezpečený zdroj:

1. Entita musí implementovat *SecuredResourceEntityInterface* nebo rozšířit *AbstractSecuredResourceEntity*.

2. Je nutno implementovat korespondující „Transformer“.

3. V poslední řadě je také potřeba implementovat rozhraní pro „Permission Provider“.

Zabezpečené Entity První bod je poměrně přímočarý. Z důvodu, aby aplikace mohla jednotně pracovat se zabezpečenými zdroji, je nutné implementovat příslušné rozhraní. Pro pohodlí programátora je již většina potřebných metod implementována ve třídě *AbstractSecuredResourceEntity*.

Transformer Druhý bod už úplně jasný není. Transformer, neboli měnič, je třída, která ze záznamu v zabezpečených zdrojích zpětně dohledává objekt, ke kterému zabezpečený zdroj patří. Toto se hodí hlavně při výpisu zdrojů v uživatelském rozhraní. Pro úspěšnou implementaci Transformeru je potřeba dodat specializaci třídy *AbstractBasicSecuredResourceTransformer*. Ve chvíli dodání implementace není potřeba nic konfigurovat, tento transformer je automaticky registrován v příslušných službách. **Poznámka**: Resource Accessor také musí implementovat vlastní Transformer.

Permission Provider Permission Provider definuje jaká práva lze na zabezpečený zdroj aplikovat. Tento Provider také přiřazuje jednotlivým číselným

hodnotám, uloženým v záznamu pro řízení přístupu, význam. Každý zabezpečený zdroj musí dodat implementaci příslušného rozhraní. I zde není potřeba další konfigurace. Od chvíle, kdy je dodána implementace, je vše funkční.

4.2.2.4 Resource Accessor

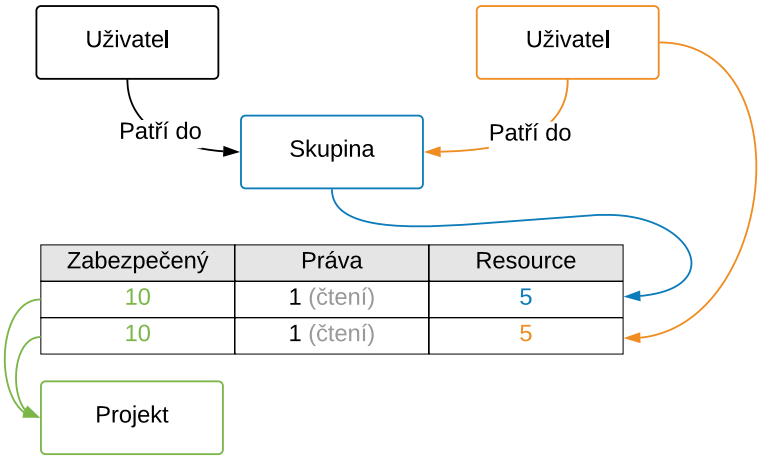
Objekty, které přistupují k zabezpečeným zdrojům, jsou v aplikaci, a celkově v systému, zdrojům velmi podobné. Každá Entita, která chce využít dynamické autorizace a stát se Resource Accessorem, musí rozšířit příslušné rozhraní. Dále musí implementovat Transformer a na rozdíl od zabezpečeného zdroje (který dodává Permission Provider) také může dodat tzv. „Access Control Entry Provider“ (ACEP). ACEP je následně využit univerzálním Voterem pro sestavení databázového dotazu pro objekty, na které má aktuální uživatel přístup.

Access Control Entry Provider ACEP dodává potřebná omezení, která jsou přidána do výše zmíněného databázového dotazu. Přesněji do tohoto dotazu přidávají omezení („where“ klauzule), která specifikují dodatečné záznamy pro řízení přístupu, které mají být přidány do výsledku. Toto je jednou z nejdůležitějších částí systému, neboť umožňuje ověřit autorizaci čistě skrze jeden databázový dotaz. ACEP také umožňuje libovolně rozšiřovat autorizační systém.

Příklad využití ACEP Vezmeme v potaz případ kdy chceme povolit uživateli přístup ke zdroji, protože patří do určité skupiny uživatelů, která na zdroj přístup má. Je potřeba hledat záznamy pro řízení přístupu, které se vztahují jak na uživatele, tak na skupinu uživatelů. Jak skupina, tak uživatel jsou objekty, které mohou přistupovat k zabezpečenému zdroji. Pokud je vytvořen záznam pro řízení přístupu na jeden z těchto objektů, je v tomto záznamu určeno který z objektů má přístup a k jakému zdroji. Pro vizuální podporu je toto vyobrazeno na obrázku 4.5.

Kdyby byly kontrolovány pouze záznamy vztahující se na uživatele, nezjistilo by se, jestli náhodou nemá přístup tranzitivně skrze jiný objekt. Pokud je potřeba tato tranzitivní práva zjistit, je nutné projít všechny uživatelovi skupiny a kontrolovat také jejich záznamy. Přesně toto ACEP řeší, ale velice elegantně. Místo toho, aby se dotazoval na záznamy od každé skupiny zvlášť, tak jen definuje, jaké mají být přidány do výsledku hlavního dotazu.

Implementace Celý dotaz je konstruován za využití Doctrine Query Builder, díky tomu je přidání alternativních omezení velice jednoduché. Kód v listing 2 ukazuje právě konstrukci omezení pro uživatelské skupiny z příkladu, který byl prezentován výše. Tato omezení jsou poté přidána do hlavního dotazu.



Obrázek 4.5: Vizualizace záznamů pro řízení přístupu

```php
class UserGroupAccessControlEntryProvider extends AbstractBaseAccessControlEntryProvider
{

    /**
    * @param User $user The user that is checked for the a.c entries
    * @param AccessControlEntryQueryBuilderAttributes $queryBuilderAttributes The[...]
    * @return mixed The "where" expression to specify the corresponding resource [...]
    */
    protected function getConditionalExpressionForRelevantResourceAccessors(
        User $user,
        AccessControlEntryQueryBuilderAttributes $queryBuilderAttributes
    )
    {
        /** @var UserToUserGroupAssignment[] $userToUserGroupAssignments */
        $userToUserGroupAssignments = $user->getUserGroupAssignments();
        $resourceAccessorRecords = [];
        foreach ($userToUserGroupAssignments as $userGroupAffiliation) {
            $group = $userGroupAffiliation->getUserGroup();
            $resourceAccessorRecords[] = $group->getResourceAccessorRecord()->getId();
        }
        $qb = $queryBuilderAttributes->queryBuilder;
        if (count($resourceAccessorRecords) === 0) return $qb->expr()->eq(0, 1);
        return $qb->expr()->in(
            $queryBuilderAttributes->accessControlEntryAlias.'.resourceAccessorRecord',
            $resourceAccessorRecords
        );
    }
}
```

Listing 2: Ukázka implementace Access Control Entry Provider (ACEP).

4.2.2.5 Výhody zvoleného řešení

Jednou z největších výhod zvoleného přístupu je, že lze odfiltrovat veškeré objekty, na které uživatel nemá přístup již v databázovém stroji. Je dokonce možné požádat databázi o počet všech objektů, na které má uživatel či jaký-

koliv jiný Resource Accessor přístup, což dovoluje využití paginace přímo na úrovni databázových dotazů. Není tedy třeba filtrovat záznamy lokálně a tím odpadá většina problémů spojených s ukládáním velkého množství záznamů v databázi.

Toto řešení není implementačně závislé a je možné na databázi napojit další aplikace, které tohoto autorizačního systému mohou plně využít. Tabulky/entity také nejsou zanášeny velkým množstvím doménově nespecifických atributů. Nakonec je tento systém také velice flexibilní a dobře rozšiřitelný.

4.3 Bezpečnost a testování

Společně s autorizací a autentizací byl při vývoji také kladen důraz na bezpečnost celého systému. Vzhledem k tomu, že webová aplikace obsahuje velké množství formulářů, kterými jsou předávány informace od uživatele k aplikaci, je potřeba věnovat pozornost ošetření vstupu z těchto formulářů. Dále také aplikace podporuje přihlášení skrze interní login formulář, který vyžaduje přeposlání citlivých údajů po síti.

Důležitou součástí tvorby větších systémů je testování funkcionality. Automatické testování je v dnešní době poměrně rozšířené. Při tvorbě webové aplikace však testování funkcionalit nebylo úplně jednoduché.

Tato sekce se nejprve zabývá obecnou bezpečností systému a poté krátce popíše jakým způsobem byl systém během vývoje testován.

4.3.1 Bezpečnost

„*S tím, jak se software stává více a více komplexním a propojeným, se složitost dosažení bezpečnosti zvedá exponenciálně. Rychlé odhalení a vyřešení nejčastějších rizik je při rapidním tempu moderního vývoje softwaru nezbytné.*“ [57, s. 2] (překlad dle autora) Je tedy nutno se bezpečností systému zabývat již během vývoje. S řešením bezpečnostních problémů také pomáhají využité knihovny a komponenty frameworku.

OWASP top 10 „*Open Web Application Security Project (OWASP) je otevřená komunita, jejímž cílem je umožnit organizacím vyvíjet, pořizovat a udržovat aplikace a API, které jsou důvěryhodné.*“ [57, s. 1] (překlad dle autora) Tato komunita již publikovala několik vydání deseti největších bezpečnostních rizik při vývoji webových aplikací. Existuje obrovské množství potenciálních rizik a není reálné veškerá rizika rozebrat v této sekci. Proto se sekce zaměří primárně na vybraná rizika z nejnovějšího vydání „OWASP Top 10 - 2017“. Následující seznam popisuje deset nejčastějších bezpečnostních rizik při vývoji webových aplikací dle [57]:

1. Injekce

2. Neošetřená autentizace (Broken Authentication)
3. Zobrazení citlivých dat (Sensitive Data Exposure)
4. XML External Entity (XXE)
5. Chyby v zamezení přístupu (Broken Access Control)
6. Špatná konfigurace zabezpečení (server, knihovny, framework, …)
7. Cross-Site Scripting (XSS)
8. Neošetřená deserializace
9. Komponenty se známými zranitelnostmi
10. Nedostatečné logování a monitorování

Dále se tato sekce zabývá každou z těchto hrozeb a jejím řešením v této aplikaci.

4.3.1.1 1 – Injekce

Ošetření injekce škodlivých dat do databázových dotazů z velké části řeší využité ORM Doctrine. Doctrine ošetřuje veškeré hodnoty, které jsou vkládány a aktualizovány skrze *persist* operace vykonané aplikační třídou *EntityManger* [58]. To znamená, že veškeré hodnoty, které jsou ukládány přímo skrze Entity, jsou ošetřeny.

Je však potřeba explicitně ošetřit vstup při ruční tvorbě databázových dotazů, například pomocí *QueryBuilder*. Při implementaci bylo na ošetření vstupu v databázových dotazech myšleno a veškerý vstup, který není přímo dodán aplikaci, je ošetřen.

Další místo, kde by mohlo dojít k případnému vložení škodlivých dat, je spouštění externí funkcionality za využití systémových příkazů. Externí funkcionalita je spuštěna přes Shell a uživatelský vstup je předáván skrze parametry příslušnému spustitelnému souboru. I zde bylo myšleno na případnou injekci škodlivých dat a vstup je tedy řádně ošetřen za využití *escapeshellcmd* a obalení vstupu příslušnými znaky.

4.3.1.2 2 – Neošetřená autentizace

Pro autentizaci skrze interní formulář a JWT, jsou využity knihovny třetích stran, které jsou bezpečné. Autentizace přes Shibboleth je také v základu integrována do bezpečnostní komponenty, kterou Symfony využívá. Tato komponenta však neověřuje, zda je daný typ autorizace správně implementován (logicky nemůže). Při implementaci autorizace přes Shibboleth byl dán důraz na využití pouze nezfalšovatelných dat, která jsou předána aplikaci přímo webovým serverem a ne klientem.

Autorizace aktuálně nepodporuje explicitní ochranu proti útokům „hrubou silou“ (brutal force), do kterých také spadá „credential stuffing“. Předpokládá se, že většina uživatelů se bude přihlašovat přes systém Shibboleth a proto tato ochrana bude implementována až během dalšího vývoje. V aplikaci je zatím pouze vyžadována minimální síla hesla (pokud si uživatel chce nastavit heslo), což výrazně ztíží tento typ útoků. Je také využito černé listiny nejpoužívanějších hesel. Pro databázi nejpoužívanějších hesel je využit projekt SecLists, který je k nalezení na adrese: https://github.com/danielmiessler/SecLists.

4.3.1.3 3 – Citlivé údaje

Základem bezpečnosti je uchování citlivých údajů v nečitelné formě. Veškeré citlivé udaje (zatím pouze uživatelská hesla) jsou před uložením do databáze „hešována“ (hashed). Při ukládání těchto údajů jsou dodrženy „best practices“ frameworku Symfony. Je tedy využito doporučovaného *bcrypt* enkodéru, který navíc, kromě jiného, také k „heši“ (hash) přidá ještě „sůl“ pro ochranu před „rainbow table“ útoky a útoky založených na hrubé síle [59].

Dále jsou na serveru, na kterém bude systém nasazen, instalovány certifikáty pro šifrování datového přenosu za využití protokolu HTTP Secure (HTTPS). Toto je velice důležité pro zabezpečení přenosu citlivých dat po síti.

4.3.1.4 4 – XML External Entity

„*Tento útok nastává ve chvíli, když vstup z XML obsahující reference na externí entity, je zpracován špatně konfigurovaným XML parserem.*“ [60] (překlad dle autora) Protože aktuálně aplikace nevyužívá XML parserů ke zpracování dat dodávaných uživateli či externími službami, není třeba se tímto problémem zabývat.

4.3.1.5 5 – Chyby v zamezení přístupu

Omezení přístupu je integrováno skrze bezpečnostní komponentu frameworku Symfony a díky tomu například není možné falšovat identitu či roli uživatele v systému. Je však potřeba se zaměřit na řádnou konfiguraci a validnost business logiky, která autorizaci v systému realizuje.

Při úpravě dat jsou formuláře editovány a validovány na základě uživatelských práv. Proto není možné poslat dodatečná data ve formulářích, neboť data by neprošla validací formuláře.

Velká část problémů s povolením přístupu na zdroj, je díky dynamické autorizaci převedena na administrátora systému a případně další subjekty, které mohou práva udělovat. Systém však musí dávat pozor při tranzitivním zpřístupňování zdrojů, které nevyužívají dynamické autorizace. Například zařízení nejsou jednotlivě zabezpečena, neboť nevyužívají dynamické autorizace (dle uživatelských požadavků). Je však potřeba uživateli, který má přístup na

projekt, zpřístupnit také zařízení, která projekt využívá. Zatím jsou tranzitivně přenesena práva z projektu na zařízení, tedy uživatel může se zařízením manipulovat ve stejném rozsahu jako může manipulovat s projektem, který ho využívá.

4.3.1.6 6 – Špatná konfigurace zabezpečení

Za konfiguraci zabezpečení na straně serveru není zodpovědný vývojový tým tohoto projektu, proto je možné brát v potaz pouze konfiguraci webové aplikace.

Je důležité, aby byla správně nakonfigurována konkrétnost chybových hlášek, například, aby běžný uživatel nedostal důvěrné informace o systému (např. heslo do databáze). Aplikace většinu chybových hlášek před běžným uživatelem skrývá. Je však dovoleno vytvořit chybové hlášky, které jsou uživateli vypsány. Texty těchto chybových hlášek musejí být řádně ošetřeny programátorem. Aktuálně jsou výpisy ošetřeny ve všech případech.

Dále je potřeba zkontrolovat nastavení příslušných knihoven třetích stran, jako je autentizace skrze JWT. Příslušná nastavení byla během integrace těchto knihoven důkladně kontrolována.

Při nasazení aplikace je také nutné nastavit parameter *secret* v konfiguraci. Tento parametr přidává entropii do operací spojených se zabezpečením. Symfony hodnotu zmíněného parametru využívá například pro šifrování cookies.

4.3.1.7 7 – Cross-Site Scripting

„*Cross-Site Scripting (XSS) útoky jsou typem injekce, ve které je škodlivý skript vložen do jinak neškodné a důvěryhodné stránky. XSS útok vzniká ve chvíli, kdy útočník využije webové aplikace k odeslání škodlivého kódu, obecně ve formě browser side skriptu, jinému uživateli.*“ [61] (překlad dle autora) Například může být do databáze, pod určitým textovým atributem, uložen škodlivý skript, který je poté zobrazen při výpisu dat zpět do aplikace. Takový skript po vypsání do webové stránky může škodit uživateli, který dané stránce věří. [61]

Tento typ útoku lze ošetřit buď již při validaci dat získaných od uživatele nebo až při samotném výpisu dat. S řešením tohoto problému také pomáhá šablonovací systém. V aplikaci, které se věnuje tato práce, je tento problém částečně řešen již při validaci dat od uživatele a částečně šablonovacím systémem Twig. Některé textové záznamy však povolují formátování za využití HTML značek. V těchto případech je využito odfiltrování veškerých případně škodlivých elementů, ale základní neškodné elementy jsou ponechány.

4.3.1.8 8, 9, 10 – Shrnutí

Osmý bod je aktuálně pro aplikaci nerelevantní, neboť nejsou deserializována žádná data, která nebyla přímo dodána aplikací samotnou. Také je pro serializaci a deserializaci využito externích knihoven, což převádí pozornost k bodu devátému. Veškeré knihovny, které aplikace využívá, byly voleny s vědomím, že mohou potenciálně ohrozit bezpečnost aplikace. Z tohoto důvodu byly voleny pouze dlouhodobě podporované a široce využívané knihovny. Logování a monitorování obstarává samotný framework (za využití knihovny Monolog), který veškeré chyby a varování detailně loguje do souborů. Tyto soubory jsou k nalezení ve složce *var/logs*. Rozšířené možnosti logování budou přidány po další konzultaci se zadavatelem.

4.3.1.9 Cross-Site Request Forgery

„*Cross-Site Request Forgery (CSRF) je útok, který nutí koncového uživatele ke spuštění nechtěných akcí na webové aplikaci, ve které je aktuálně autentizován. CSRF útoky jsou cíleny na požadavky měnící stav, ne na krádež dat, neboť útočník nemá způsob, jakým by se dozvěděl odpověď na zmanipulovaný požadavek.*“ [62] (překlad dle autora)

Tento typ útoku je téměř shodný s útokem XSS. V obou případech jde o vložení škodlivých dat do obsahu, kterému uživatel věří. Rozdíl je v tom, že CSRF využívá autentizovaného uživatele pro změnu určitého stavu webové aplikace (webová aplikace autentizovanému uživateli věří). Zatímco útok XSS se snaží od uživatele získat data či ho poškodit jiným způsobem.

V aplikaci se v aktuálním stavu nachází malé množství koncových bodů, které mohou být na tento typ útoku náchylné. Tyto koncové body se nacházejí ve vnitřním API, kde je spouštěna externí funkcionalita za využití GET požadavků. Náchylná místa jsou postupně z aplikace odstraňována a v další verzi aplikace (zatím neprodukční verze) je již tento problém odstraněn.

Formuláře, včetně interního login formuláře jsou chráněny CSRF tokeny, o které se stará samotný framework Symfony.

4.3.2 Testování

Testování se tato kapitola věnuje pouze velice krátce, neboť systém je stále ve vývoji a nemá být v nejbližší době nasazen do produkčního prostředí.

Velice vhodným způsobem testování aplikační funkcionality jsou testy jednotkové. Tyto testy se zaměřují na otestování jednotlivých částí systému v izolovaném prostředí. Pro usnadnění tohoto testování existují nástroje jako je PHPUnit (https://phpunit.de/). Problémem v tomto projektu však je, že v aplikaci se nachází velice málo business logiky, kterou by šlo izolovaně testovat.

4.3.2.1 Autorizační systém

Nejsložitější částí aplikace je autorizační systém, který pro ověření autorizace využívá dotazů na aplikační databázi. Pro otestování této funkcionality je tedy potřeba kontrolovat výsledky sql dotazů, což již není považováno za jednotkový test. Dále je také potřeba vytvořit před otestováním autorizační funkcionality testovací schéma a naplnit ho daty. Z důvodů složitější integrace těchto testů a častého refaktorování celého systému byla jejich tvorba z časových důvodů vynechána.

Testy autorizačního systému mohou být na požádání vytvořeny v další implementační iteraci. Pokud bude potřeba systém autorizace dále rozšiřovat, budou tyto testy zcela jistě dodány.

4.3.2.2 Funkcionalita systému

Jak již bylo řečeno, systém neobsahuje téměř žádné složitější business operace a tím pádem není k dispozici funkcionalita, kterou by šlo izolovaně testovat. Testování externí funkcionality realizující výkoné funkce robotů není předmětem tohoto projektu. Jediným případem je hluboká kopie Projekt, pro kterou byl vytvořen vlastní jednotkový test za využití nástroje PHPUnit. Většinu ostatní funkcionality je potřeba testovat skrze funkční testy. To znamená za využití HTTP požadavků, neboť z velké části jde o správné zpracování uživatelských dat.

4.3.2.3 Provedené testování

Systém je během vývoje pravidelně uživatelsky testován (interně). Jádro systému je testováno skrze front-end aplikace, neboť je již napojeno na uživatelské rozhraní. Pro účely kompletního otestování systém také nabízí generické koncové body, které zpřístupňují veškerou funkcionalitu jádra. Dále bylo využito externích nástrojů pro tvorbu HTTP požadavků, kterými bylo otestováno připravované API a jeho zabezpečení. V neposlední řadě bylo také otestováno napojení externí funkcionality na jádro systému. Více ohledně testování uživatelského rozhraní je k nalezení v bakalářské práci, studenta FIT ČVUT v Praze, Erika zatloukala.

Systém je rozsáhlý a je teprve v počátečních fázích testování, zatím byly odhaleny běžné nedostatky většinou menšího rozsahu. Před přechodem systému do produkčního prostředí bude potřeba provést extenzivnější uživatelské testování (veřejné).

4.4 Technický přehled a budoucnost projektu

Tato sekce se zabývá technickými detaily implementace a budoucností projektu. Dále tato sekce také shrnuje implementační část celého systému. Pro-

jekt je stále v aktivním vývoji a je tedy dobré separovat do vlastní podsekce části, které jsou předmětem budoucího vývoje.

4.4.1 Technický přehled implementace

Tato sekce shrnuje již zmíněné informace o implementační části projektu a doplňuje některé technické detaily. V této sekci se bude opakovat velké množství informací, které jsou k nalezení v předchozích kapitolách, neboť jde z velké části o rekapitulaci. Účelem této sekce je také poskytnout kvantifikovatelná data ohledně implementační části projektu.

4.4.1.1 Aktuální stav

Projekt je nyní v první alfa verzi. Tato verze obsahuje, zatím ne zcela kompletní, uživatelské rozhraní a základní funkcionalitu systému včetně první verze databázového schématu. Verzování systému je logicky rozděleno do „alfa“, „beta“ a „production“ verzí. Alfa verze jsou testovány interně a po dostatečném testování mohou přejít do beta verzí, které jsou již testovány veřejně (studenty a učiteli, kteří budou systém využívat). Za produkční verzi je považována verze systému, která neobsahuje žádné nedostatky a lze ji plně využívat při výuce.

4.4.1.2 Databázové schéma

Jak již bylo zmíněno dříve, databázové schéma je optimalizováno pro databázový stroj MySQL a ORM Doctrine. V aktuální verzi obsahuje 36 tabulek, více než 45 asociací a přes 140 sloupečků. Tento model je primárně modelován v nástroji MySQL workbench, ale pro nejnovější verzi byl také vytvořen obecnější ERD diagram, který je k nalezení v přílohách této práce (ve složce *visual_attachments/diagrams*). Databázový model bude při dalším vývoji upravován dle potřeb aplikace a zadavatele.

4.4.1.3 Webová aplikace

Webová aplikace je založena na frameworku Symfony a je rozdělena na dvě hlavní části. Těmito částmi jsou front-end a back-end. Back-end je předmětem této práce a realizuje veškerou business logiku aplikace a napojení na databázové úložiště. Je složen z více jak 230 aplikačních tříd a obsahuje přes 26 000 řádků kódu v jazyce PHP (přes 34 000 včetně front-end části). Přesněji je to cirka 10 000 fyzických řádků (bez komentářů a prázdných řádků) pro back-end část a circa 17 500 řádků celkem (pouze PHP soubory). Je nutno brát v úvahu, že velká část kódu ve frameworku Symfony se nachází právě v komentářových blocích (v podobě anotací), které jsou v druhém součtu vynechány. Tento počet byl získán za využití nástroje SLOCCount (k nalezení

na: https://www.dwheeler.com/sloccount/). Dále aplikace také obsahuje 730 řádků konfiguračního kódu, avšak více jak polovina je standardní konfigurace.

Čistě generovaného kódu je v aplikaci velmi malé množství, neboť i generovaný kód je potřeba správně anotovat a upravit dle potřeb. Většinou jsou generovány deklarace proměnných (včetně ORM mapování) nebo jednoduché asociace v entitních třídách.

Následuje krátké shrnutí složitosti implementace a největších částí aplikace. Tato shrnutí přesněji specifikují množství práce, která byla do implementace projektu vložena.

Rozšiřitelnost a abstrakce Protože systém má být rozšiřitelný, byla většina komponent webové aplikace abstrahována. Abstrahovány jsou Fasády, Entity, Controllery a veškeré komponenty využívané autorizačním a autentizačním systémem. Díky této abstrakci je systém poněkud komplexnější, ale zato je duplicita kódu omezena na minimum. Abstrakce jednotlivých částí systému zabrala poměrně hodně času, ale pro systém tohoto rozsahu je nezbytná.

Mapování entit Velká část úsilí byla také vynaložena do mapovaní a validace jednotlivých entit. Protože uživatelské požadavky na systém a externí funkcionalitu se během vývoje měnily, bylo potřeba databázové schéma několikrát upravit. Tyto úpravy byly většinou pouze menšího rozsahu, ale bylo potřeba provést i několik větších úprav. Realizace těchto větších úprav znamenala výrazně změnit mapování databázových objektů a přepracovat třídy na ně napojené, což společně s drobnými úpravami také zabralo poměrně hodně času. Tyto úpravy jsou však nezbytné pro zaručení nejvyšší možné kvality výsledného systému.

Napojení externí funkcionality Protože webová aplikace a externí funkcionalita v podobě spustitelných souborů, která realizuje výkonné funkce zařízení, byly vyvíjeny paralelně, je poměrně obtížné obě části udržet synchronizovány. Od doby, kdy byla externí funkcionalita do systému implementována poprvé, se rozhraní změnilo, a proto bylo obtížné veškerou funkcionalitu udržet řádně propojenu. Webová aplikace, jako řešení těchto problémů, nabízí velmi jednoduchý způsob konfigurace propojení obou funkcionalit (za pomoci abstrakce spustitelných souborů) a v případě úpravy rozhraní se jedná pouze o přidání nových parametrů. I přesto je lepší počkat na stabilní rozhraní externí funkcionality a teprve poté synchronizovat propojení pro danou verzi rozhraní.

Autentizace Do aplikace bylo implementováno několik způsobů autentizace i přes to, že v požadavcích na systém byla specifikována pouze jediná metoda. Hlavní metodou autentizace je, dle uživatelských požadavků, autentizace skrze systém Shibboleth. Dále aplikace podporuje autentizaci za využití interního

formuláře. V poslední řadě také podporuje autentizaci skrze JWT pro veřejné API. První autentizační metoda nevyužívá knihoven a je implementována autorem za využití *AuthenticationInterface*. Autentizace skrze JWT a interní formulář využívají knihoven či základní funkcionality frameworku Symfony, ale jsou upraveny pro potřeby aplikace. V aplikaci je také implementováno mapování metadat ze systému Shibboleth na interní autentizační identity (aplikační třídy). Toto mapování lze upravit skrze konfigurační soubory aplikace.

Testování autentizační metody za využití systému Shibboleth není úplně jednoduché. Toto je z důvodu, že není možnost autentizační metodu plně testovat lokálně při vývoji, neboť je potřeba nastavit připojení k IdP (což je složité a vyžaduje spolupráci provozovatele IdP). Proto testování této funkcionality zabralo více času. Bylo nutné kopírovat proměnné prostředí z produkčního serveru a za jejich pomocí testovat systém lokálně.

Autorizace Autorizační systém aplikace je implementován bez využití knihoven třetích stran a je plně rozšiřitelný. Aplikační logika realizující autorizační systém je složena z více jak 40 tříd/rozhraní a je také nejsložitější částí celé aplikace. Jednou z největších výhod tohoto autorizačního systému, je možnost filtrování veškerých databázových dotazů přímo na straně databázového stroje (nemusí nejdříve data načítat do paměti). Nejvíce úsilí zabrala právě realizace autorizačního systému za využití pouze databázových dotazů (tedy implementační nezávislost autorizačního systému). Webová aplikace tedy nemusí poskytovat autorizační API a aplikace, které chtějí autorizačního systému využít, se mohou připojit přímo na databázi.

Skrytá funkcionalita Protože front-end a back-end webové aplikace je vyvíjen nezávisle, není veškerá funkcionalita aplikace nabízena skrze uživatelské rozhraní. Z funkcionality, kterou back-end aplikace nabízí je v aktuálním stavu využito přibližně 40 %. V rámci tvorby back-end části aplikace bylo také zpřístupněno několik koncových bodů, bez integrace uživatelského rozhraní. Seznam veškerých koncových bodů aktuální verze aplikace je k nalezení na adrese */cs/routes* (adresa v aplikaci), nebo je také k nalezení na přiloženém CD ve složce *attachments/misc*. Z tohoto seznamu jsou vynechány multijazyčné koncové body (uvedeny jsou pouze v jednom jazyce). Poměrně velkou část práce tedy nelze v aktuálním stavu otestovat přes uživatelské rozhraní, avšak v budoucnu bude zcela jistě využita.

Využité knihovny Aplikace využívá knihoven třetích stran, které realizují některé standardní funkcionality systému (jako je například překlad UI). Následující seznam popisuje knihovny využité při implementaci aplikace, avšak neobsahuje knihovny, které jsou obsaženy v základním balíčku Symfony:

- FakerBundle – Generování testovacích dat.
  GitHub: https://github.com/willdurand/BazingaFakerBundle

- BreadcrumbsBundle – Realizace drobečkové navigace.
  GitHub: https://github.com/whiteoctober/BreadcrumbsBundle
- VichUploaderBundle – Zpracování nahraných souborů.
  GitHub: https://github.com/dustin10/VichUploaderBundle
- DoctrineExtensionsBundle – Využito rozšíření Timestampable a Sluggable. Timestampable dovoluje aplikaci jednoduše upravovat časové údaje evidované v Entitách na základě událostí (jako je uložení Entity). Sluggable je využito pro transformaci nadpisů stránek a projektů do textového identifikátoru (tento identifikátor se nazývá „slug“). Rozšíření Uploadable není využito, neboť ve verzi knihovny pro PHP 5.6 byla v době vývoje chyba.
  GitHub: https://github.com/Atlantic18/DoctrineExtensions
- PasswordStrengthBundle – Ověření síly hesla.
  GitHub: https://github.com/rollerworks/PasswordStrengthBundle
- LexikJWTAuthenticationBundle – Autentizace API.
  GitHub: https://github.com/lexik/LexikJWTAuthenticationBundle
- JMSTranslationBundle – Překlad uživatelského rozhraní.
  GitHub: https://github.com/schmittjoh/JMSTranslationBundle
- JMSi18nRoutingBundle – Vícejazyčné cesty (webové).
  GitHub: https://github.com/schmittjoh/JMSI18nRoutingBundle
- FOSRestBundle – Příprava pro REST API.
  GitHub: https://github.com/FriendsOfSymfony/FOSRestBundle

4.4.2 Budoucnost projektu

Vývoj systému touto prací nekončí. Je počítáno s delší podporou, přibližně do konce léta 2018 (dle požadavků zadavatele). Systém byl vyvíjen s myšlenkou na další rozvoj, a proto je většina aplikačních komponent připravena k rozšíření. Tato sekce se zabývá rekapitulací všeho, co je plánováno pro budoucí rozvoj systému.

4.4.2.1 Externí funkcionalita

V době vývoje systému byla paralelně vyvíjena externí funkcionalita v podobě spustitelných souborů. Tato funkcionalita byla integrována do systému téměř na konci celého vývoje, neboť bylo potřeba stabilního rozhraní. Otestování napojení proběhlo úspěšně a je tedy ověřeno, že systém může této funkcionality plně využít. Tato externí funkcionalita bude zcela jistě dále rozšiřována a s tím bude potřeba rozšířit také příslušné rozhraní realizující napojení v systému.

4.4.2.2 Dokumentace

Tvorba dokumentace je pro systém tohoto rozsahu nezbytná. Během dosavadního vývoje byl systém dokumentován přímo v kódu. Tato kódová dokumentace se primárně nachází u komponent, které je potřeba rozšiřovat či u komponent, které jsou často využívány. Kód je také psaný poměrně expresivním způsobem, avšak nelze předpokládat, že každý kdo k systému přijde, bude vědět jak ho rozšířit. Proto bude součástí dalšího vývoje tvorba externí dokumentace v podobě strukturovaného textu.

4.4.2.3 Rozšíření uživatelského rozhraní

V aktuálním stavu webové rozhraní nevyužívá žádných „client side“ skriptů. To se má s budoucím vývojem systému změnit a uživatelské rozhraní bude vylepšeno o interaktivní prvky. Aby vše fungovalo, bude potřeba v jádru implementovat API, které dovolí těmto skriptům jednoduše komunikovat s aplikací.

4.4.2.4 Testování a úprava datových struktur

Dále je také počítáno s extenzivnějším testováním systému. Na základě tohoto testování bude jistě potřeba přidat či odebrat některé atributy v systému. To však díky zvoleným technologiím není problém.

Závěr

Cílem této práce bylo na základě rešerše navrhnout a implementovat jádro webového informačního systému pro správu robotů. První částí návrhu a implementace jádra systému je tvorba a nasazení systémové databáze. Druhou částí je návrh a implementace back-end části webové aplikace pro správu robotů včetně autentizace a autorizace. Po jádru je také vyžadováno napojení na externí funkcionalitu dodávanou studentem (FIT ČVUT v Praze) Petrem Kolářem, čímž jsou realizovány výkonné funkce zařízení. Dále je jádro zodpovědné za snadné napojení back-end části na front-end webové aplikace, který je předmětem bakalářské práce studenta (FIT ČVUT v Praze) Erika Zatloukala.

V rámci této práce jsem provedl analýzu uživatelských požadavků, aktuálního stavu, existujících řešení, dostupných technologií a vhodných vývojových metodik. Na základě této analýzy jsem vybral vhodné technologie a postupy pro zhotovení jádra. Všechny cíle práce byly splněny. Systém tedy podporuje veškerou funkcionalitu, která byla požadována. Některé části však ještě nejsou plně otestovány.

Nejprve jsem navrhl databázové schéma, na které jsem poté napojil back-end webové aplikace. Databázové schéma je navrženo tak, aby umožnilo realizaci veškerých uživatelských požadavků.

Ve webové aplikaci jsem využil vlastní implementace autentizační metody, za využití systému Shibboleth. Autentizační systém je díky zvoleným technologiím snadno rozšiřitelný a v aktuálním stavu podporuje tři různé typy autentizačních metod, včetně Shibboleth Single Sign-On a Json Web Tokens (pro API). Dále jsem implementoval rozšiřitelný autorizační systém, využívající přístupových listů dle uživatelských požadavků.

Do webové aplikace jsem přidal příslušné aplikační třídy a metody, které realizují napojení externí funkcionality. Úspěšně jsme, společně s Petrem Kolářem (autorem externí funkcionality), otestovali napojení spustitelných souborů na webovou aplikaci.

Na základě analýzy jsem navrhl propojení back-end a front-end části systému. Propojení bylo realizováno za využití fasádních tříd a repozitářů, které

oddělují business logiku od uživatelského rozhraní.

Vývoj systému touto prací nekončí a projekt bude vývojovým týmem dále podporován dle požadavků zadavatele. Další vývoj se bude z velké části soustředit na uživatelské testování a úpravy systému na něm založené. Dále také bude tvořena externí dokumentace v podobě strukturovaného textu, neboť systém je poměrně komplikovaný. Projekt je možné rozvíjet dle potřeb, neboť při tvorbě bylo myšleno na jeho rozšiřitelnost.

Bibliografie

1. JONÁT, Libor. *EDUX API a Klasifikace*. Praha, 2012. Diplomová práce. České vysoké učení technické.
2. KADLETO2. *Návody: Syntaxe wiki* [online]. 2009. 14. 10. 2009 [cit. 2018-04-18]. Dostupné z: `https://edux.fit.cvut.cz/howto/syntax`.
3. KADLETO2; BARINKL. *Návody: Tvorba stránek* [online]. 2009. 14. 10. 2009 [cit. 2018-04-18]. Dostupné z: `https://edux.fit.cvut.cz/howto/stranky`.
4. ANDI et al. *DokuWiki Development* [online]. 2018. 5. 4. 2018 [cit. 2018-04-22]. Dostupné z: `https://www.dokuwiki.org/development`.
5. ALEKSANDR et al. *DokuWiki Manual* [online]. 2018. 21. 3. 2018 [cit. 2018-04-18]. Dostupné z: `https://www.dokuwiki.org/manual`.
6. GLAZAR, Filip. *Systém pro podporu BI-DBS - semestrální práce*. Praha, 2016. Bakalářská práce. České vysoké učení technické. Dostupné také z: `https://dspace.cvut.cz/handle/10467/65184`.
7. SÝKORA, Jan. *Optimalizace hodnocení a automatické kontroly semestrální práce v předmětu Databázové systémy*. Praha, 2015. Bakalářská práce. České vysoké učení technické. Dostupné také z: `https://dspace.cvut.cz/bitstream/handle/10467/63060/F8-BP-2015-Sykora-Jan-thesis.pdf`.
8. SLAVOTÍNEK, Jiří. *Webová komponenta na kreslení ER diagramů*. Praha, 2016. Bakalářská práce. České vysoké učení technické.
9. COOCH, Mary et al. *About Moodle: Features* [online]. 2017. 28. 3. 2017 [cit. 2018-04-21]. Dostupné z: `https://docs.moodle.org/34/en/Features`.
10. FOSTER, Helen et al. *Managing a Moodle site: Authentication* [online]. 2017. 13. 11. 2017 [cit. 2018-04-21]. Dostupné z: `https://docs.moodle.org/34/en/Authentication`.

11. WORDPRESS. *WordPress Features* [online]. 2018 [cit. 2018-04-22]. Dostupné z: https://codex.wordpress.org/WordPress_Features.

12. WORDPRESS. *Developer Documentation* [online]. 2018 [cit. 2018-04-22]. Dostupné z: https://codex.wordpress.org/Developer_Documentation.

13. BNJMNM et al. *Understanding Drupal: Overview* [online]. 2016. 3. 4. 2018 [cit. 2018-04-22]. Dostupné z: https://www.drupal.org/docs/8/understanding-drupal-8/overview.

14. ZORYA et al. *Community Documentation: Module developer's guide* [online]. 2004. 26. 1. 2018 [cit. 2018-04-22]. Dostupné z: https://www.drupal.org/developing/modules.

15. MARTINEZ-CARO, Jose-Manuel et al. *A Comparative Study of Web Content Management Systems*. Spain, 2018. Review. Department of Information Technologies and Communications, Universidad Politécnica de Cartagena (UPCT), Department of Information a Communications Engineering, Universidad de Murcia (UM). Dostupné také z: www.mdpi.com/2078-2489/9/2/27/pdf.

16. DISHA.ADDWEB et al. *Theming Drupal 7: Overview of theme files* [online]. 2007. 23. 12. 2016 [cit. 2018-04-22]. Dostupné z: https://www.drupal.org/docs/7/theming/overview-of-theme-files.

17. MAGUIRE, Martin; BEVAN, Nigel. User Requirements Analysis: A Review of Supporting Methods. In: *Proceedings of the IFIP 17th World Computer Congress - TC13 Stream on Usability: Gaining a Competitive Edge* [online]. United Kingdom, 2002, s. 133–148 [cit. 2018-04-23]. ISBN 1-4020-7187-6. Dostupné z: https://link.springer.com/content/pdf/10.1007/978-0-387-35610-5_9.pdf.

18. ARMY, United States Government US. *User Requirements Analysis* [online]. Virginia, 2001 [cit. 2018-04-24]. ISBN 978-1484120835. Dostupné z: https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-885j-aircraft-systems-engineering-fall-2005/readings/sefguide_01_01.pdf.

19. SILVESTRE, F. *Software Requirements Specification* [online]. Brussel, 2003. Verze 1 [cit. 2018-04-24]. DG TREN a TACHOnet. Dostupné z: https://ec.europa.eu/transport/sites/transport/files/modes/road/social_provisions/doc/tcn_srs_01-00.pdf.

20. SCALED AGILE. Domain Modeling [online]. 2017 [cit. 2018-04-28]. Dostupné z: https://www.scaledagileframework.com/domain-modeling/.

21. KHARCHENKO, Natalia. 8 JavaScript Alternatives for Web Developers to Consider. *codeburst.io* [online]. 2017 [cit. 2018-04-30]. Dostupné z: https://codeburst.io/8-javascript-alternatives-for-web-developers-to-consider-22f8d38bdfa9.

22. WODEHOUSE, Carey. HTML vs. XHTML vs. HTML5: Understanding the Difference Between These Commonly Confused Terms. *UpWork.com* [online]. 2018 [cit. 2018-04-30]. Dostupné z: https://www.upwork.com/hiring/development/html-vs-xhtml-vs-html5/.

23. BROVKIN, Dmytro. Best 10 Programming Languages to learn in 2018. *medium.com* [online]. 2017 [cit. 2018-04-30]. Dostupné z: https://medium.com/swlh/best-10-programming-languages-to-learn-in-2018-2d6cbc5ffc2a.

24. STACKOVERFLOW. *Developer Survey Results 2017* [online]. 2017 [cit. 2018-04-30]. Technická zpráva. stackoverflow. Dostupné z: https://insights.stackoverflow.com/survey/2017#technology.

25. PUTANO, BEN. Most Popular and Influential Programming Languages of 2018. *Stackify.com* [online]. 2017 [cit. 2018-04-30]. Dostupné z: https://stackify.com/popular-programming-languages-2018/.

26. POWER, Vince. Most Popular Java Web Frameworks. *Rollbar.com* [online]. 2018 [cit. 2018-04-30]. Dostupné z: https://dzone.com/articles/most-popular-java-web-frameworks.

27. BASE et al. Introduction. *developer.mozilla.org* [online]. 2018 [cit. 2018-04-30]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction.

28. NODE.JS FOUNDATION. About Node.js®. *nodejs.org* [online]. 2018 [cit. 2018-04-30]. Dostupné z: https://nodejs.org/en/about/.

29. MISIRLAKIS, Speros. The 7 Most In-Demand Programming Languages of 2018. *codingdojo.com* [online]. 2017 [cit. 2018-04-30]. Dostupné z: https://www.codingdojo.com/blog/7-most-in-demand-programming-languages-of-2018/.

30. CODERSEYE. 11 Best PHP Frameworks for Modern Web Developers in 2018. *coderseye.com* [online]. 2018 [cit. 2018-05-01]. Dostupné z: https://coderseye.com/best-php-frameworks-for-web-developers/.

31. YOUNG, David. Software Development Methodologies. 2013. Dostupné také z: https://www.researchgate.net/profile/David_Young30/publication/255710396_Software_Development_Methodologies/links/0c960520515475fc8e000000/Software-Development-Methodologies.pdf?origin=publication_detail.

32. SUBBAIAH, Vinod. Why Agile Is The Best Alternate Methodology To Waterfall? *AsahiTechnologies.com* [online]. 2016 [cit. 2018-04-29]. Dostupné z: https://www.asahitechnologies.com/blog/why-agile-is-the-best-alternate-methodology-to-waterfall/.

33. MARTIN, Robert C. *Čistý kód: Návrhové vzory, refaktorování, testování a další techniky agilního programování*. COMPUTER PRESS, 2009. ISBN 978-80-251-2285-3.

34. DEVART. Database-First Approach. *Devart.com* [online]. 2018 [cit. 2018-05-05]. Dostupné z: https://www.devart.com/entitydeveloper/database-first.html.

35. DEVART. Model-First Approach. *Devart.com* [online]. 2018 [cit. 2018-05-05]. Dostupné z: https://www.devart.com/entitydeveloper/model-first.html.

36. SINGH, Durgesh Kumar; KUMAR, Harish. SQL vs NoSQL: The Holy War. *International Journal of Computer Engineering and Applications*. 2016, roč. X, Special Issue. Dostupné také z: http://www.ijcea.com/wp-content/uploads/2016/09/160314097.pdf. ISSN: 2321-3469.

37. IT GMBH, solid. *DB-Engines Ranking* [online]. 2018 [cit. 2018-05-01]. Technická zpráva. solid IT gmbh. Dostupné z: https://db-engines.com/en/ranking.

38. STUDYTONIGHT. Normalization of Database. *studytonight.com* [online]. 2018 [cit. 2018-02-05]. Dostupné z: https://www.studytonight.com/dbms/database-normalization.php.

39. CHAPPLE, Mike. Database Normalization Basics. *lifewire.com* [online]. 2018 [cit. 2018-02-05]. Dostupné z: https://www.lifewire.com/database-normalization-basics-1019735.

40. STUDYTONIGHT. What is Second Normal Form? *studytonight.com* [online]. 2018 [cit. 2018-02-05]. Dostupné z: https://www.studytonight.com/dbms/second-normal-form.php.

41. PIVETTA, Marco; EBERLEI, Benjamin. *Documentation: Best Practices* [online]. 2015 [cit. 2018-04-18]. Dostupné z: https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/reference/best-practices.html.

42. SCHWARTZ, Baron. How to build role-based access control in SQL. *www.xaprb.com* [online]. 2006 [cit. 2018-03-05]. Dostupné z: https://www.xaprb.com/blog/2006/08/16/how-to-build-role-based-access-control-in-sql/.

43. MICROSOFT. *Using a Three-Tier Architecture Model* [online]. 2018 [cit. 2018-04-05]. Dostupné z: https://msdn.microsoft.com/en-us/library/windows/desktop/ms685068(v=vs.85).aspx.

44. TARHINI, Ali. Concepts of Three-Tier Architecture. *alitarhini.wordpress.com* [online]. 2011 [cit. 2018-04-05]. Dostupné z: https://alitarhini.wordpress.com/2011/01/22/concepts-of-three-tier-architecture/.

45. SCHATTEN, Alexander et al. Best Practices: Model View Controller Pattern. *best-practice-software-engineering.ifs.tuwien.ac.at* [online]. 2013 [cit. 2018-06-05]. Dostupné z: http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns/mvc.html.

46. RAMOS, Marcia et al. Issues. *GitLab Documentation* [online]. 2018 [cit. 2018-04-05]. Dostupné z: https://docs.gitlab.com/ee/user/project/issues/.

47. MIJAILOVIC, Marko et al. Requirements for Running Symfony. *Symfony.com* [online]. 2018 [cit. 2018-05-05]. Dostupné z: https://symfony.com/doc/current/reference/requirements.html.

48. BERNARD, Maximilien et al. The Best Practices Book. *Symfony.com* [online]. 2017 [cit. 2018-05-05]. Dostupné z: https://symfony.com/pdf/Symfony_best_practices_3.4.pdf.

49. EGUILUZ, Javier et al. How to Work with Service Tags. *Symfony.com* [online]. 2017 [cit. 2018-05-05]. Dostupné z: https://symfony.com/doc/3.4/service_container/tags.html.

50. EGUILUZ, Javier et al. Quick Tour: The Architecture. *Symofny.com* [online]. 2018 [cit. 2018-05-05]. Dostupné z: https://github.com/symfony/symfony-docs/blob/master/quick_tour/the_architecture.rst.

51. ROUSE, Margaret. single sing-on (SSO). *searchsecurity.techtarget.com* [online]. 2016 [cit. 2018-06-05]. Dostupné z: https://searchsecurity.techtarget.com/definition/single-sign-on.

52. SHIBBOLETH CONSORTIUM. What's Shibboleth? *shibboleth.net* [online]. 2017 [cit. 2018-06-05]. Dostupné z: https://www.shibboleth.net/index/.

53. CANTOR, Scott et al. Shibboleth Concepts. *wiki.shibboleth.net* [online]. 2017 [cit. 2018-06-05]. Dostupné z: https://wiki.shibboleth.net/confluence/display/CONCEPT.

54. YOUNG, Ian et al. IdPConfiguration: IdPUserAuthn. *wiki.shibboleth.net* [online]. 2012 [cit. 2018-06-05]. Dostupné z: https://wiki.shibboleth.net/confluence/display/SHIB2/IdPUserAuthn.

55. EGUILUZ, Javier et al. Security: The Firewall and Authorization. *Symfony.com* [online]. 2018 [cit. 2018-06-05]. Dostupné z: https://github.com/symfony/symfony-docs/blob/master/components/security/firewall.rst.

56. EGUILUZ, Javier. How to Create a custom Authentication Provider. *Symfony.com* [online]. 2018 [cit. 2018-06-05]. Dostupné z: https://symfony.com/doc/3.4/security/guard_authentication.html.

57. OWASP. *OWASP Top 10 - 2017: The Ten Most Critical Web Application Security Risks* [online]. 2017 [cit. 2018-07-05]. Technická zpráva. The OWASP Foundation. Dostupné z: https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf.

58. CARUSO, Gabriel et al. *Security* [online]. 2017. 31. 12. 2017 [cit. 2018-07-05]. Dostupné z: `https://www.doctrine-project.org/projects/doctrine-orm/en/latest/reference/security.html`.

59. EGUILUZ, Javier et al. Best Practices: Security. *Symfony.com* [online]. 2018 [cit. 2018-07-05]. Dostupné z: `https://symfony.com/doc/3.4/best_practices/security.html`.

60. OSWP. Vulnerability: XML External Entity (XXE) Processing. *owasp.org* [online]. 2017 [cit. 2018-07-05]. Dostupné z: `https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing`.

61. OWASP. Attack: Cross-site Scripting (XSS). *owasp.org* [online]. 2018 [cit. 2018-07-05]. Dostupné z: `https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)`.

62. OWASP. Attack: Cross-Site Request Forgery (CSRF). *owasp.org* [online]. 2018 [cit. 2018-07-05]. Dostupné z: `https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)`.

63. SOURCEMAKING. Design Patterns: Facade Pattern. In: *SourceMaking.com* [online]. 2018 [cit. 2018-04-21]. Dostupné z: `https://sourcemaking.com/design_patterns/facade`.

64. SENSIOLABS. Symfony at a Glance?: Symfony in 5 minutes. In: *Symfony.com* [online]. 2016 [cit. 2018-04-19]. Dostupné z: `https://symfony.com/at-a-glance`.

65. COMPUTERHOPE. Definitions: Framework. In: *ComputerHope.com* [online]. 2017 [cit. 2018-04-18]. Dostupné z: `https://www.computerhope.com/jargon/f/framework.htm`.

66. MICROSOFT. *The Repository Pattern* [online]. 2018 [cit. 2018-04-18]. Dostupné z: `https://msdn.microsoft.com/en-us/library/ff649690.aspx`.

Dodatek A

Slovník

back-end Představuje vrstvu aplikace, která provádí veškerou business logiku a stará se o vše co není přímo spojeno s prezentací obsahu.

baud rate Udává rychlost přenosu dat v komunikačním kanálu.

best practices Osvědčené postupy.

cloud Jako „cloud“ je označováno velké množství propojených počítačových serverů, které nabízejí určité služby (většinou úložné prostory, či výpočetní výkon).

copyleft V kontextu licencí tento výraz znamená, že je vyžadována určitá forma zachování licence v dílech odvozených z původního.

Fasády Aplikační třídy, využívající návrhového vzoru Facade, které obalují složitější funkcionalitu a poskytují k ní jednoduché rozhraní [63].

framework Sada knihoven a nástrojů, které vývojářům poskytují standardizovaný postup vývoje a pomáhají jim soustředit se na doménově specifické záležitosti místo znovu objevování již existujících řešení známých podproblémů. Definice převzata z [64] a také z [65].

front-end Představuje vrstvu aplikace, která má za účel prezentovat obsah uživateli. Tato vrstva se nestará o žádnou logiku, která není úzce spojena s prezentací obsahu.

Kandidátní Klíč V relačních databázích se takto značí klíč obsahující minimální počet sloupečku, který může jednoznačně identifikovat každý záznam v tabulce.

open source Ve spojení se softwarem znamená, že kód je chráněn licencí, která uživateli dovoluje kód číst, měnit a případně dále distribuovat za jakýmkoliv účelem.

preprocessor Program, který transformuje vstup a jehož výstup je dále zpracován jinými programy.

programová komponenta Jakýkoliv spustitelný soubor, včetně instalačních balíčků.

programový obraz Reprezentuje větší objemy dat, které byly vytvořeny například při běhu nějakého programu. Může například reprezentovat zálohu systému.

repositáře Aplikační třídy, využívající návrhového vzoru repository, které se navenek chovají jako kolekce datově mapovaných objektů a primárně podporují operace pro vyhledávání těchto objektů [66].

shell skript Spustitelné soubory psané v jazyce Shell.

Symfony Components Soubor samostatně použitelných knihoven určených primárně pro tvorbu webových aplikací.

token Nejčastěji je token, ve spojení se softwarem, objekt (například řetězec nebo číslo), který může být využit jako průkaz totožnosti.

wiki software Software, který umožňuje uživateli vytvářet a měnit obsah webových stránek, většinou zaměřený na práci s textem.

Dodatek B

Seznam použitých zkratek

ACE Access Control Entry.

ACEP Access Control Entry Provider.

ACL Access Control List.

API Application Programming Interface.

CGI Common Gateway Interface.

CMS Content Management System.

CSRF Cross-Site Request Forgery.

CSS Cascading Style Sheets.

DDL Data Definition Language.

ERD Entity Relationship Diagram.

FIT Fakulta informačních technologií.

FURPS Functionality, Usability, Reliability, Performance and Supportability.

HTML Hypertext Markup Language.

HTTP Hyper Text Transfer Protocol.

HTTPS HTTP Secure.

IdP Identity Provider.

JWT Json Web Tokens.

LDAP Lightweight Directory Access Protocol.

MVC Model View Controller.

MVP Model View Presenter.

OOP Object-oriented programming.

ORM Object Relational Mapping.

OWASP Open Web Application Security Project.

PHP PHP: Hypertext Preprocessor.

SP Service Provider.

SSO Single Sign-On.

XHTML Extensible Hypertext Markup Language.

XML eXtensible Markup Language.

XSS Cross-Site Scripting.

XXE XML External Entity.

YAML YAML Ain't Markup Language.

ČVUT České vysoké učení technické.

Dodatek C

Návod k instalaci

C.1 Docker a vývojové prostředí

Aplikace podporuje zprovoznění vývojového prostředí systému za pomoci nástroje Docker Compose. Toto prostředí je vhodné pro lokální testování aplikace. V kořenovém adresáři aplikace (pro přiložený disk *src/app* a pro Git *Sources/robot-management-system/*) je k nalezení adresář *laradock-extended-rms*. Stačí se přesunout do výše zmíněného adresáře a spustit příkaz *docker-compose up -d mysql apache2 workspace php-fpm*. Poté dohledat adresu příslušných kontejnerů (jména začínají na *laradockextendedrms*), nahrát databázi dle návodu níže a aplikace by měla být dostupná na adrese *apache2* kontejneru.

C.2 Instalace na serveru

C.2.1 Příprava aplikace

Nejprve je nutné do aplikace stáhnout veškeré potřebné knihovny. O to se stará nástroj Composer. Následující kroky popisují jak získat zdrojové kódy aplikace a potřebné knihovny.

C.2.1.1 Využití Git

1. Spustit příkaz *git clone git@gitlab.fit.cvut.cz:zatlodan/robot-management-system.git* (naklonování git repositáře, pokud ještě není stažený).

2. Přejít do kořenové složky Git repositáře.

3. Spustit příkaz *git pull*.

4. Přejít do adresáře *Sources/robot-management-system* (tento adresář je nazýván „kořenový adresář aplikace“).

5. Spustit příkaz *php „cesta k souboru composer.phar“ install* (nebo *composer install*).

    - Tento krok stáhne všechny nejnovější knihovny. Při prvním spuštění také rovnou vytvoří soubor *parameters.yml*.
    - Tento krok může občas vyvolat konflikty ve verzích knihoven. Pokud se tak stane, je potřeba smazat složku *„kořenový adresář aplikace“/vendor* a krok opakovat.

C.2.1.2 Přiložené CD

1. Přejít do *„kořenová složka CD“/app* (tento adresář je nazýván „kořenový adresář aplikace“).

2. Pokračovat od kroku 5 sekce C.2.1.1 (Využití Git).

C.2.2 Konfigurace aplikace

Proto, aby aplikace byla nakonfigurována pro cílové prostředí, je potřeba správně nastavit parametry v souboru *parameters.yml*, který se nachází ve složce *„kořenový adresář aplikace“/app/config*. Tento soubor je vytvořen nástrojem Composer při prvním spuštění příkazu *install* viz. sekce C.2.1. Pokud soubor neexistuje, je možné zkopírovat soubor *parameters.yml.dist* ve stejné složce a kopii přejmenovat na *parameters.yml*.

V tomto konfiguračním souboru je potřeba vyplnit adresu databázového serveru, přihlašovací jméno do databáze a heslo. V základu je vypnuta externí funkcionalita a pro její úspěšné zprovoznění je potřeba se obrátit na autora projektu (Petr Kolář). Systém také podporuje odesílání emailu. Je tedy vhodné nakonfigurovat také emailový server. Dále je potřeba vypnout veškeré komponenty, které nejsou potřeba, nastavením příslušného parametru na hodnotu *false* (například Shibboleth autentizaci). V souboru *parameters.yml* jsou pouze často se měnící parametry a pro kompletní možnou konfiguraci je dobré si projít složku *config* a vyhledat všechny dostupné parametry (vždy pod slovíčkem *parameters:*).

C.2.3 Příprava databáze

Pro úspěšné zprovoznění aplikace je potřeba jí napojit na systémovou databázi. Skript pro vytvoření databázového schématu s názvem *create_script.sql* je umístěn na jednom z následujících míst:

- V případe přiloženého CD se nachází ve složce *src/database/SQL*.
- V případě git repositáře se nachází ve složce *Database/SQL*.

Dále je také potřeba vložit počáteční data. Skript pro vložení dat se nachází ve stejné složce jako skript pro vytvoření databáze a jmenuje se *insert_prod.sql* (insert_dev.sql v případě testovacích dat).

C.2.3.1 Přihlašovací údaje

V aplikaci je v základu vytvořený „super admin“ účet, do kterého se lze přihlásit skrze interní login formulář. Přihlašovací údaje pro tento účet jsou:

- Jméno: admin
- Heslo: 123321

C.2.4 Konfigurace webového serveru

Veškeré cesty odkazované v této části předpokládají, že se uživatel nachází v kořenovém adresáři aplikace. Tato konfigurace je platformě specifická, proto nelze vše popisovat detailně.

- Index aplikace se nachází ve složce *./web/app.php* (*app_dev.php* v případě vývojového prostředí. Toto prostředí je však potřeba dále nakonfigurovat, v základu povolí přístup pouze z lokálního počítače.)
- Webový server stačí navést do složky ./web a poté povolit přepsání pravidel pomocí *.htaccess* souborů.
- Aplikace potřebuje přístup i ke složkám, které se nacházení o jednu úroveň výš (*../var ../app* atd.). Tyto složky tedy musejí být aplikaci dostupné přes standardní příkazy (cd *../var*).
- Aplikace potřebuje vlastnit práva na zápis do adresářů *../var*, *../web* a do jakýchkoliv dalších adresářů, které byly nastaveny v konfiguraci, například jako cílová složka pro nahrávání souborů.

C.2.5 Řešení problému

Pokud nastane jakýkoliv problém, je dobré vyzkoušet několik věcí:

- Přejít do kořenového adresáře aplikace.
- Spustit příkaz *php bin/console cache:clear –env=dev* (vyčistí cache vývojového prostředí).
- Spustit příkaz *php bin/console cache:clear –env=prod* (vyčistí cache produkčního prostředí).
- Případně smazat složku *vendor* a znovu provést příkaz *composer install*.

Pokud jsou stále problémy, je potřeba zkontrolovat log soubory. Aplikační log soubory se nacházejí ve složce *„kořenový adresář aplikace“/var/log*. Pokud v těchto záznamech nejsou nalezeny žádné chyby, je také potřeba zkontrolovat logy od webového serveru.

Dodatek D

Obsah přiloženého CD

- `src`
  - `app` ..... zdrojové kódy webové aplikace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
    - `images` ..... obrázky a diagramy využité v písemné práci
  - `database` ..... databázový model a SQL skripty
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
- `attachments` ..... veškeré přílohy
  - `diagrams` ..... diagramy nevyužité v písemné práci
  - `showcase_images` ..... ukázky designu webové aplikace
  - `misc` ..... další nezařazené přílohy
- `docs` ..... systémová dokumentace
  - `setup.pdf` ..... návod k instalaci
  - `code_docs` ..... vygenerovaná kódová dokumentace