ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra počítačů

Bakalářská práce

Promise – Project Management Information System

Pavel Krulec
Softwarové inženýrství a technologie

2017/2018
Vedoucí práce: Ing. Martin Tomášek

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Krulec** Jméno: **Pavel** Osobní číslo: **456920**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Systém pro správu a plánování projektů využívající servisně orientovanou architekturu.

Název bakalářské práce anglicky:

Information system for managing and project planing using service oriented architecture.

Pokyny pro vypracování:

Analyzujte, navrhněte a vytvořte systém [1] [2] pro správu projektů a organizací, v kterém lze zakládat více organizací, každá z organizací má oddělení v rámci kterého pracují lidé. Jednotlivá oddělení pracují na zakázkách, které mají svůj životní cyklus. V rámci aplikace lze plánovat práci do budoucna a manuálně rozvrhovat zdroje. Využijte jeden z moderních JavaScript frameworků a na BE dodržte princip Servisně orientovaných architektur k zabezpečení použijte protokol OAuth2. Aplikace musí mít vlastní datovou vrstvu. Cílová aplikace je nasazená a funkční v reálném prostředí. Ověřte možnosti generování automatizovaného uživatelského rozhraní v rámci vašich zvolených technologií [3] [4], popřípadě demonstrujte použití. Dále ověřte možnosti vytvoření mobilních verzí k této aplikaci. Aplikaci vhodně otestujte.

Seznam doporučené literatury:

1] ARLOW, J. ? NEUSTADT, I. UML 2 a unifikovaný proces vývoje aplikací: Objektově orientovaná analýza a návrh prakticky. Computer Press a.s., Brno, 2nd edition. ISBN 978-90-251-1503-9
[2] FOWLER, M. Patterns of Enterprise Application Architecture. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0321127420.
[3] M. Tomasek and T. Cerny. On web services ui in user interface generation in standalone applications. In Proceedings of the 2015 Conference on Research in Adaptive and Convergent Systems, RACS, pages 363{368, New York, NY, USA, 2015. ACM.
[4] M. Tomasek and T. Cerny. Automated User Interface Generation Involving Field Classification. In Software Networking Journal, pages 54{76, January 2017}

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Martin Tomášek, katedra počítačů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **24.11.2017** Termín odevzdání bakalářské práce: **25.05.2018**

Platnost zadání bakalářské práce: **30.09.2019**

Ing. Martin Tomášek podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

Poděkování / Prohlášení

Rád bych tímto poděkoval své rodině, která mě po celý život ve všech ohledech podporuje. Bez vás bych se nikdy nikterak daleko nedostal. Děkuji za to, že vás mám!

Poděkování patří mému vedoucímu, Ing. Martinu Tomáškovi. Děkuji za příležitost a za cenné zkušenosti, kterých se mi dostalo nejen v rámci této práce!

Nemalý dík si zaslouží i mí přátelé za to, že mě nenechali zapadnout do pracovních povinností.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne

.....

Abstrakt / Abstract

Tato práce se zabývá analýzou, návrhem, implementací a produkčním nasazením informačního systému na správu IT projektů se zaměřením na fázi podpory software.

V úvodní části popisuje současnou situaci a existující řešení. Dále se věnuje především návrhu řešení, které svým zaměřením bude oproti konvenčním nástrojům vhodněji a lépe podporovat procesy a potřeby týmů zabývajících se podporou software. Závěrečná část je věnována popisu konkrétní implementace a nasazení na produkční prostředí.

This project addresses analysis, design, implementation and production deployment of IT project management information system with a focus on the software support phase.

The opening part describes current situation and existing solutions. Next the main focus is placed on a design of a solution which will support processes and needs of software support teams better than conventional tools. The final part is dedicated to a description of actual implementation and deployment to a production environment.

Title translation: Promise – Project Management Information System

Obsah /

Tabulky / Obrázky

Kapitola 1
Úvod

Většina současných aplikací na správu projektů definuje jejich životní cyklus velmi podobně: Vznikne hlavní projekt, který se rozpadne na dílčí části. Tyto části se sestávají z úkolů, jejichž postupným plněním dochází k plnění dílčích částí a tím i k plnění samotného projektu. Práce může být plánována dopředu tím způsobem, že zaměstnancům jsou přiřazovány jednotlivé úkoly. Projekt je dokončen ve chvíli, kdy jsou dokončeny všechny jeho jednotlivé části.

Tento proces je typický pro týmy, které se primárně zabývají prvními částmi životního cyklu software, tj. jeho návrhem a vývojem. Není však příliš vhodný pro týmy, které mají na starosti tu nejdelší a nejnákladnější fázi [1] životního cyklu software – jeho podporu.

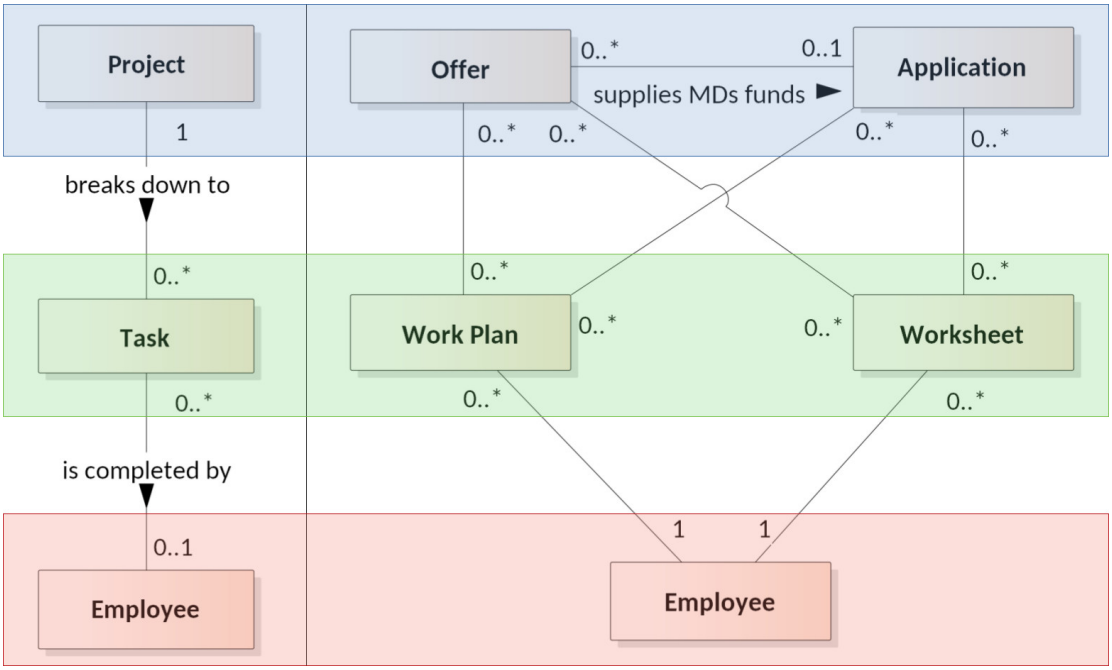
Tato práce se zabývá analýzou (kap. 2), návrhem (kap. 3), implementací (kap. 4, 5 a 6) a produkčním nasazením (kap. 7) informačního systému na správu IT projektů se zaměřením na fázi podpory software. V následujících kapitolách je popsán návrh a princip aplikace, která si klade za cíl stát se vhodným nástrojem pro správu projektů týmů zabývajících se podporou software.

Kapitola 2
Analýza

V analytické části jsou nejprve popsána a srovnána vybraná existující řešení. Následuje výčet funkčních a nefunkčních požadavků a dalších nároků na implementovaný systém.

2.1 Existující řešení

V oblasti IT existuje nespočet řešení, která se věnují správě projektů na vývoj software. Podporují různé metodiky (agilní vývoj, klasický vodopádový vývoj, ...); některé jsou zdarma, jiné placené a celkově lze říci, že nabídka možností je velmi široká. V jádru si jsou řešení podobná a liší se v přidaných hodnotách. Některé se zaměřují na malé týmy, jimž nabízejí bezplatné řešení, nízké nároky a střídmé uživatelské rozhraní, zatímco jiné obsahují nespočet funkcí, propojují další systémy a jsou zaměřené na náročné uživatele.



Obrázek 2.1. Paralela mezi jinými existujícími řešeními (vlevo) a Promise (vpravo).

Hlavním společným znakem všech řešení je fáze životního cyklu software, na níž se zaměřují – vývoj. Pracovníkům jsou zadávány konkrétní úkoly přispívající k dokončení projektu. Projekt má svůj rozpočet, který by neměl být překročen. Za tímto účelem dostupná řešení nabízí dashboard, na němž lze vývoj financí sledovat. Primární důraz je však kladen na zadávání a správu úkolů.

Jednotlivá řešení nabízí různé způsoby autentizace a autorizace uživatelů. Mezi nejvýznamnější se řadí OAuth2 a Security Assertion Markup Language (SAML). Zatímco první zmíněný slouží pouze k autorizaci uživatelů a nikoliv autentizaci, SAML zvládá obojí. SAML je velmi často použit na Single Sign-on (SSO) v rámci jedné společnosti.

OAuth2 je vhodnější na použití v rámci internetu nebo v ne-webových (např. mobilních) aplikacích [2].

Níže jsou stručné charakteristiky tří vybraných dostupných řešení.

Feature / Application	Jira	Aceproject	Lavagna
Project management	Yes	Yes	Yes
Task management	Yes	Yes	Yes
Work planning	Task assign. only	Task assign. only	Task assign. only
Work reports	Yes	Yes	No
Manager dashboard	Advanced	Advanced	Basic
Permissions	Roles + groups	Roles + groups	Roles
Login via 3rd party	SAML	Internal only	OAuth2
Source code	Proprietary	Proprietary	Open source
Mobile application	Yes	Yes	No
Non-paid version	7 days trial	Up to 3 users	Yes

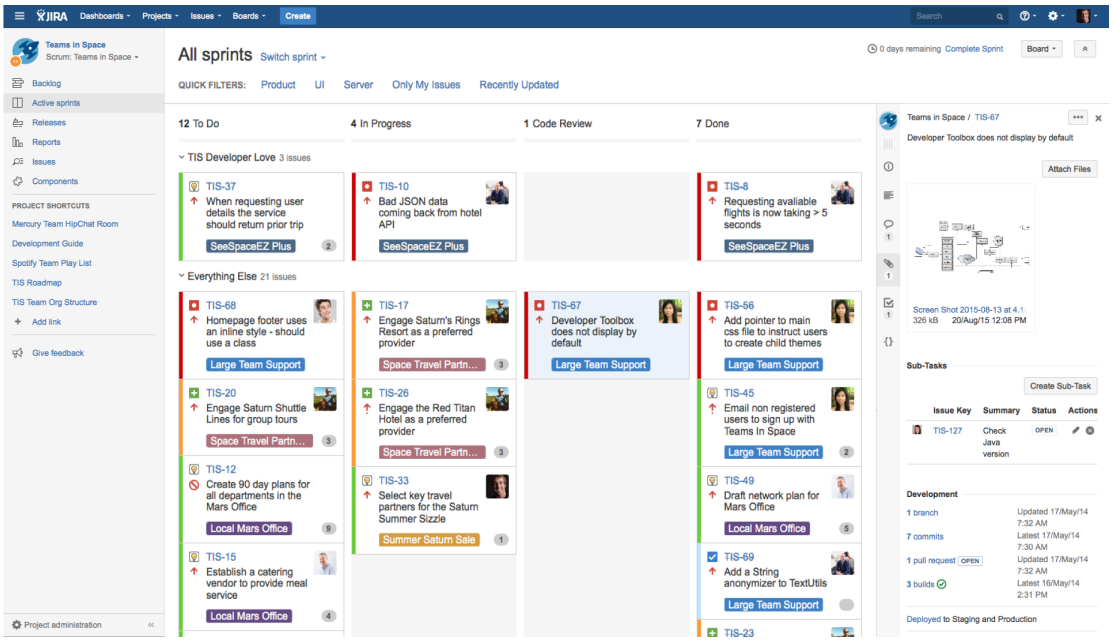
Tabulka 2.1. Porovnání existujících řešení

2.1.1 Jira

Jira [3] od společnosti Atlassian je jeden z nejznámější produktů v této oblasti. Zaměřuje se na týmy vyvíjející software. Velkou výhodou je integrace do dalších produktů této společnosti. Mínusem je absence bezplatné verze.

V současné době je po dobu testování zdarma dostupný modul, který přidává podporu dvoufázového ověření a uživatelům umožňuje autorizovat se pomocí SAML single sign-on. Po finálním uvolnění však bude tato komponenta zpoplatněna.

Plánování je podporováno formou vytváření tzv. *stories* a dílčích úkolů, které se následně přiřazují týmům a jednotlivcům. Úkoly mohou mít přiřazenou různou prioritu. Řešitel dané úlohy eviduje postup plnění a může vykazovat, kolik času řešením úkolu strávil.

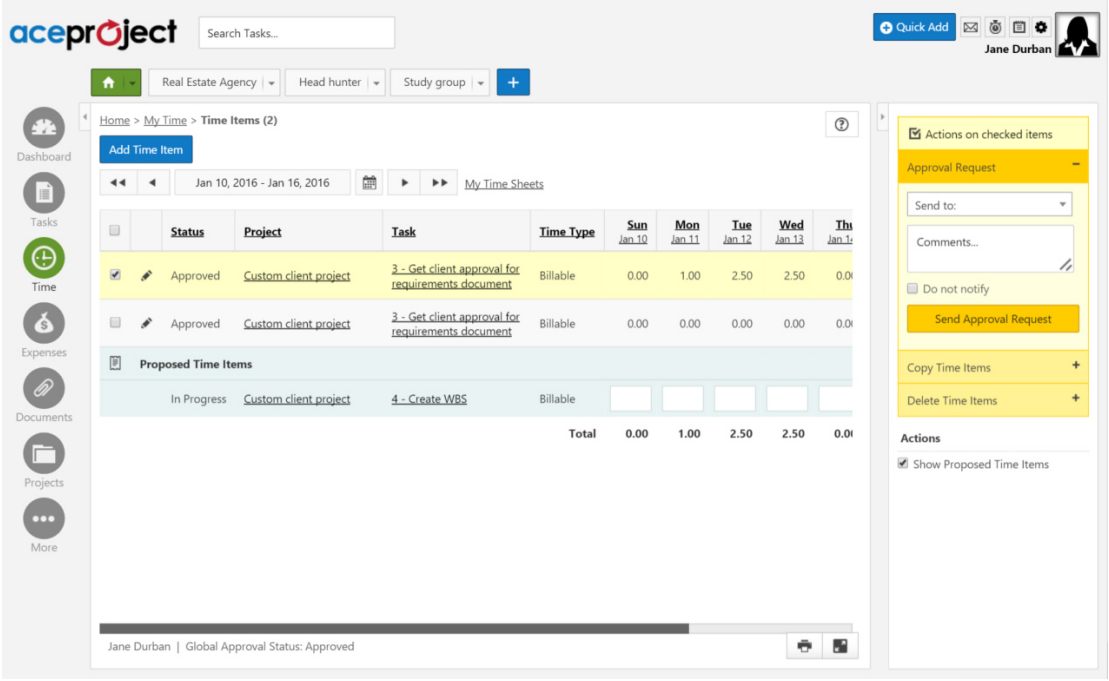


Obrázek 2.2. Ukázka prostředí aplikace Jira – seznam úkolů

2.1.2 Aceproject

Aceproject [4] je webová aplikace na správu úkolů, výkazů práce, dokumentů a nákladů. Nabízí i verzi zdarma, již však může využít nanejvýš tříčlenný tým. Na rozdíl od většiny ostatních produktů Aceproject neomezuje funkcionality u nižších (levnějších) balíčků. Připlácí se tak pouze za vyšší počet uživatelů, aktivních projektů a diskový prostor.

Nevýhodou je absence podúloh, tedy rozpadnutí větší úlohy na dílčí části. Přihlášení přes systém třetí strany (např. přes OAuth2) také není možné. Aceproject se nezaměřuje pouze na IT projekty, ale na libovolnou business oblast.



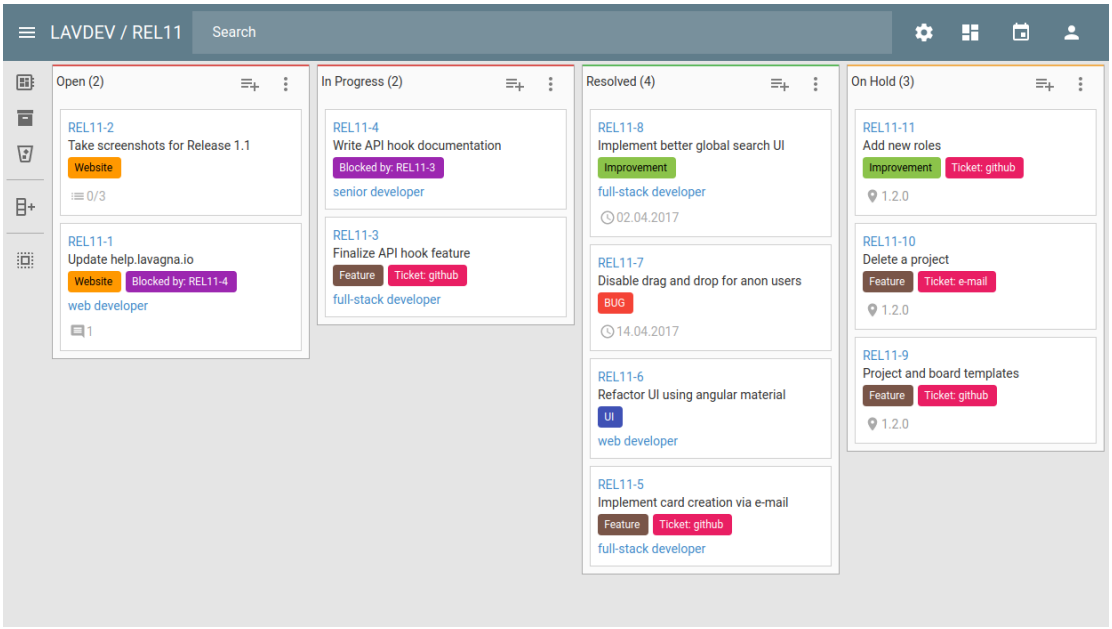
Obrázek 2.3. Ukázka prostředí aplikace Aceproject – výkaz práce

2.1.3 Lavagna

Lavagna [5] je zástupce opensource aplikací na správu projektů. Na rozdíl od předchozích řešení je tedy k dispozici její zdrojový kód a uživatelé si díky tomu mohou chybějící funkcionalitu doprogramovat. Vyznačuje se jednoduchostí a nízkými nároky, díky čemuž je předurčena malým týmům. Nabízí široké možnosti přihlašování uživatelů pomocí protokolu OAuth2.

Podobně jako ostatní řešení podporuje zobrazení úloh formou tzv. kanban board, tedy zjednodušeně řečeno sadou sloupců představujících fáze životního cyklu úkolů, po kterých se úkoly pohybují. Součástí je též obvyklý dashboard, který formou grafů prezentuje vybrané informace o projektech.

Na rozdíl od předchozích aplikací Lavagna nedisponuje nativní mobilní aplikací.



Obrázek 2.4. Ukázka prostředí aplikace Lavagna – seznam úkolů

2.2 Funkční požadavky

Funkční požadavky definují, co systém bude uživatelům umožňovat [6]. Pro účely následujícího výčtu se pojmem *oprávněný Zaměstnanec* rozumí takový Zaměstnanec, který má patřičnou Systémovou roli pro provedení akce v systému. Pojmem *spravovat* se rozumí vytváření, upravování a mazání záznamů, pokud takové nepozbývá smysl.

- **FP01** Systém bude umožňovat spravovat Nabídky.
  Nabídka bude mít dva názvy a popisy – jeden viditelný všem Zaměstnancům v Oddělení a druhý určený pouze manažerům. U Nabídky může být uvedena cena, která však bude viditelná taktéž pouze manažerům. Dále Nabídka bude mít rozpočet Man days (MDs), bude na ni možno alokovat jednotlivé Zaměstnance a bude k dispozici přehled stavu čerpání a alokace MDs. Na Nabídku bude možné vykazovat práci.
- **FP02** Systém bude umožňovat spravovat Objednávky, které budou potvrzovat Nabídky.
  Objednávka bude mít své jméno, cenu a rozpočet MDs stejně jako Nabídka. Objednávka může potvrzovat různé množství Nabídek, včetně těch, které už potvrzuje jiná Objednávka.
- **FP03** Systém bude definovat Systémové role Zaměstnanců, které je budou autorizovat k akcím v systému.
  Systémové role budou vhodným způsobem omezovat možné akce v systému a budou společné napříč všemi Odděleními. Z jednotlivých Systémových rolí budou sestaveny Business role.
- **FP04** Systém bude definovat Business role Zaměstnanců, které budou vyjadřovat jejich postavení v rámci Oddělení.
  Business role bude možné vytvářet v rámci Oddělení. Každé Business roli bude možné přiřadit vícero Systémových rolí, které pak budou Zaměstnance s přiřazenou Business rolí opravňovat k daným akcím.

- **FP05** Systém bude přiřazovat Systémové role Zaměstnancům na základě jejich Business role či přímého manuálního přiřazení.

  Každý Zaměstnanec může mít v rámci Oddělení vícero Business rolí; přiřazené Systémové role se pak budou sjednocovat.
- **FP06** Systém bude umožňovat oprávněným Zaměstnancům spravovat ostatní Zaměstnance a jejich Role.

  Bude k dispozici Systémová role, která bude opravňovat ke správě Business rolí.
- **FP07** Systém bude umožňovat spravovat Aplikace.

  Na Aplikace bude možné vykazovat a alokovat práci a bude k dispozici přehled množství alokovaného a vykázaného času na Aplikaci.
- **FP08** Systém bude umožňovat všem Zaměstnancům vykazovat svou práci formou Výkazu práce na Nabídkách přímo a nepřímo skrze Aplikace.

  Každý Zaměstnanec bude moci spravovat pouze své Výkazy práce.
- **FP09** Systém bude umožňovat oprávněným Zaměstnancům spravovat Výkaz práce ostatních Zaměstnanců.

  V systému bude k dispozici Systémová role, která bude opravňovat ke správě Výkazů práce i ostatních Zaměstnanců v daném Oddělení.
- **FP10** Systém bude umožňovat plánovat práci do budoucna, tj. alokovat ostatní Zaměstnance na určité Nabídky a Aplikace.
- **FP11** Systém bude umožňovat oprávněným Zaměstnancům sledovat celkový a zbývající rozpočet Man days (MDs) Nabídek a Objednávek.
- **FP12** Systém bude evidovat typ Úvazku Zaměstnanců.

  Systém bude zobrazovat vztah mezi možným a naplánovaným alokováním Zaměstnance. Například systém zobrazí informaci, že Zaměstnanec je zaměstnán na plný úvazek (40 hodin týdně) a aktuálně je alokován pouze na 20 hodin týdně.
- **FP13** Systém bude zobrazovat manažerský přehled (dashboard) pro plánování, výkazy a finance.
- **FP14** Systém bude zobrazovat ceny Nabídek a Objednávek pouze oprávněným Zaměstnancům.
- **FP15** Systém bude umožňovat evidovat ceny Nabídek a Objednávek v různých měnách, vždy však nanejvýš v jedné zároveň.

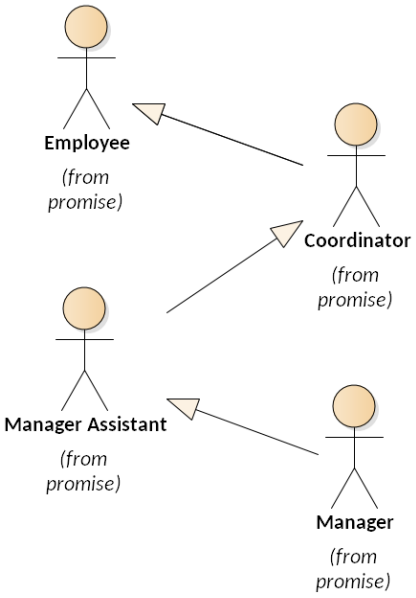
2.3 Nefunkční požadavky

Nefunkční požadavky definují, jaké bude mít systém vlastnosti.

- **NP01** Front-end (FE) aplikace bude vytvořen v JavaScriptovém frameworku.
- **NP02** Back-end (BE) aplikace bude dodržovat princip Servisně orientovaných architektur.
- **NP03** Grafické uživatelské rozhraní (GUI) bude automaticky generované na základě domain modelu.
- **NP04** Systém bude zabezpečen protokolem OAuth2.
- **NP05** Back-end a Security Token Service (STS) budou dvě oddělené komponenty.
- **NP06** Systém bude komunikovat přes šifrovaný protokol HTTPS.
- **NP07** Systém bude mít vlastní datovou vrstvu.
- **NP08** Aplikace, Nabídky, Objednávky, Přílohy, Výkazy práce, Úvazky, Pracovní plány a Business role budou dostupné pouze v Oddělení, ve kterém byly vytvořeny.
- **NP09** BE aplikace i STS budou bezestavové.

2.4 Aktéři a role

Systém bude definovat 4 aktéry: Zaměstnanec, Koordinátor, Asistent manažera a Manažer. Těmto aktérům zároveň přísluší stejnojmenné Business role.



Obrázek 2.5. Aktéři vystupující v systému.

2.4.1 Oprávnění k operacím s entitami

Každý aktér má různá práva provádět základní operace s entitami v systému. Operací může být buď čtení entity nebo její zápis (vytvoření, úprava případně odstranění). Tabulka 2.2 zachycuje, jaké operace mohou aktéři provádět s danými entitami. Operacemi se rozumí tzv. CRUD operace [7]

- C/c – CREATE,
- R/r – READ,
- U/u – UPDATE,
- D/d – DELETE

přičemž majuskula značí, že daná operace je povolena pro všechny instance dané entity, a minuskula značí, že daná operace je povolena pouze pro ty instance, které konkrétnímu aktérovi patří nebo jsou mu přiřazeny.

Pokud buňka v tabulce není vyplněna, pak je její hodnota zděděna od aktéra vlevo. Nevyplněná buňka u Zaměstnance znamená, že nemá k entitě žádná oprávnění.

Například Zaměstnanec může číst jen Nabídky, které jsou mu přiřazeny. Koordinátor může číst všechny Nabídky. Asistent manažera může Nabídky nejen číst, ale i je vytvářet, upravovat a mazat. Manažer má stejná oprávnění jako Asistent manažera.

Entity / Actor	Employee	Coordinator	Manager Assistant	Manager
Work report	crud	cRud		
Offer	-r--	-R--	CRUD	
Application	-r--	-R--		CRUD
Work plan		CRUD		
Work times		-R--	CRUD	
Department		-R--		CRUD
Organization			-R--	CRUD
Employee	cru-	-RU-		
Business role	-r--	-R--		CRUD
System role	-r--	-R--		

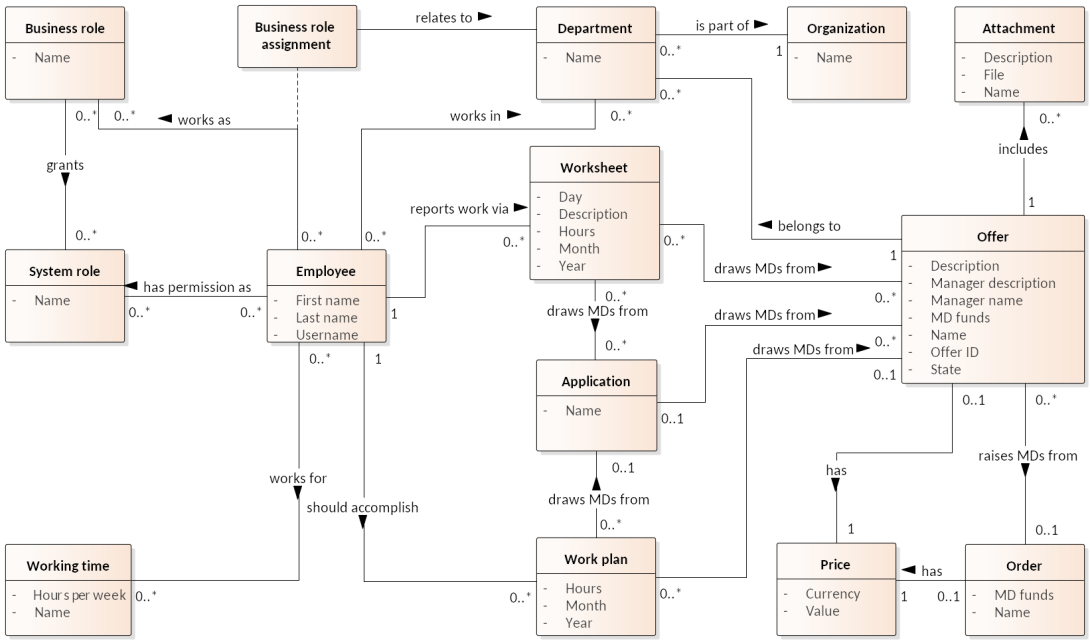
Tabulka 2.2. CRUD matice aktérů (Business rolí) a systémových entit.

Kapitola 3
Návrh řešení

Tato kapitola se zabývá technickým návrhem [8] a procesy aplikace [9]. Centrálním předmětem je Nabídka, jež dodavatel IT služeb vystaví zákazníkovi. Nabídka vyjadřuje záměr dodavatele poskytnout zákazníkovi konkrétní službu, například 100 MDs (Man Days) podpory aplikace Promise v lednu 2018. Na nabídky je možné plánovat práci zaměstnancům a ti na ni mohou vykazovat svou práci. Manažer v dashboardu vidí přehled rozdělení zaměstnanců na jednotlivé nabídky a množství zbývajících MDs, které lze ještě čerpat.

3.1 Doménový model tříd

Doménový model tříd na obrázku 3.1 identifikuje klíčové třídy systému.



Obrázek 3.1. Doménový model tříd

Základní třídou systému je Nabídka, *Offer*, kterou vydává naše strana straně partnerské. Nabídka má určitý rozpočet MDs, který je k dispozici pro čerpání. Nabídka by měla být potvrzena Objednávkou, *Order*, kterou nám zadává partnerská strana. Není to však vyžadováno a vykazovat práci lze i na nepotvrzenou Nabídku. Jedna Objednávka může potvrzovat více Nabídek, její rozpočet MDs je pak dán součtem rozpočtů MDs potvrzovaných Nabídek.

Další velmi podstatnou třídou v systému je Zaměstnanec, *Employee*. Zaměstnanci pracují v rámci Oddělení *Department*, kterým jsou přiřazovány Objednávky. Zaměstnanec může patřit do více Oddělení zároveň a v každém Oddělení mu mohou příslušet

různé Business role, *Business role*, které vyjadřují jeho postavení v rámci Organizace, *Organization*, resp. Oddělení. S Business rolemi se pojí Systémové role, *System role*. Systémové role autorizují Zaměstnance k provádění akcí v systému, například k přidávání nových Zaměstnanců je potřeba jistá Systémová role. Systémová role může být Zaměstnanci přiřazena přímo nebo skrze Business roli.

Zaměstnanci jsou v Organizaci zaměstnáni na Úvazek, *Working time*, jenž vyjadřuje, kolik hodin týdně by měli odpracovat. Tento údaj slouží především k plánování práce skrze Pracovní plán, *Working plan*. Zaměstnanec s určitou Systémovou rolí může alokovat Zaměstnance na práci na zvolené Aplikaci, *Application*, nebo přímo na Nabídce právě Pracovním plánem, přičemž by měl brát v potaz Úvazek. Díky tomu lze snadno plánovat práci do budoucna a efektivně využít dostupné zdroje.

Každý měsíc Zaměstnanci vyplní svůj Výkaz práce, *Work report*. V něm uvedou mimo jiné na jakých Nabídkách pracovali (ať už přímo nebo skrze Aplikace) a jak dlouho. Odpracovaný čas je následně strháván z rozpočtu uvedených Nabídek.

- **Application** (Aplikace) reprezentuje reálnou aplikaci, jejíž podporu dodavatel zajišťuje pro svého zákazníka. Zaměstnanci dodavatele vykazují práci, kterou na dané Aplikaci vykonali.

  Aplikace má tyto položky: *Name* (Název).
- **Attachment** (Příloha) je soubor, který lze evidovat u Nabídky. Může jím být například PDF soubor se smlouvou, zadávací dokumentace a podobně.

  Příloha má tyto položky: *Description* (Popis), *File* (Soubor), *Name* (Název).
- **Business role** (Business role) vyjadřuje pozici Zaměstnance v rámci organizační struktury společnosti. Zároveň ho opravňuje k různým akcím v systému skrze přidružené Systémové role.

  Business role má tyto položky: *Name* (Název).
- **Department** (Oddělení) ohraničuje viditelnost některých entit, které jsou v rámci Oddělení vytvořeny. Jsou jimi Aplikace, Nabídka a Objednávka. To například znamená, že Zaměstnanci z jednoho oddělení nevidí Aplikace, které jsou z jiného Oddělení.

  Oddělení má tyto položky: *Name* (Název).
- **Employee** (Zaměstnanec) primárně vykazuje svou činnost na Aplikacích a Nabídkách. Zaměstnanci s vyšším oprávněním mohou například plánovat ostatním Zaměstnancům práci do budoucna na základě jejich Úvazku.

  Zaměstnanec má tyto položky: *Username* (Uživatelské jméno), *First name* (Křestní jméno), *Last name* (Příjmení).

  Uživatelské jméno je unikátní, neměnné a shodné s uživatelských jménem v STS.
- **Offer** (Nabídka) je vystavena zákazníkovi buď na podporu některé z Aplikací nebo na vývoj. Zahrnuje určitý počet MDs, které budou na Nabídce odpracovány. Zaměstnanci následně na Nabídku vykazují svou práci. Zároveň jim může být práce na Nabídce naplánována předem. Může obsahovat Přílohy a cenu, která je však viditelná pouze Manažerovi.

  Nabídka má tyto položky: *Description* (Popis), *Manager description* (Manažerský popis), *Name* (Název), *Manager name* (Manažerský název), *MD funds* (Rozpočet MDs), *Offer ID* (ID Nabídky), *State* (Stav).

  Název a popis budou viditelné Zaměstnancům pro vykazování, zatímco Manažerský název a Manažerský popis jsou viditelné Business roli Koordinátor a vyšší a mají neveřejný charakter. Podobně i ID Nabídky, které složí jako libovolný alfanumerický business identifikátor Nabídky.
- **Order** (Objednávka) potvrzuje jednu či více Nabídek. Má též svůj rozpočet MDs, který nemusí být roven součtu rozpočtů MDs potvrzovaných Nabídek.

Objednávka má tyto položky: *Name* (Název), *MD funds* (Rozpočet MDs).

- **Organization** (Organizace) sdružuje Oddělení. Umožňuje využít 1 běžící instanci systému pro více reálných společností naráz.

  Organizace má tyto položky: *Name* (Název).
- **Price** (Cena) je vždy přidružena k Nabídce nebo Objednávce. Jedná se o dvojici Měna, v jaké je částka uvedena, a samotná Hodnota, kterou vyjadřuje.

  Cena má tyto položky: *Currency* (Měna), *Value* (Hodnota).
- **System role** (Systémová role) přímo opravňuje Zaměstnance k provádění akcí v systému. Existuje např. Systémová role umožňující vytváření a správu Nabídek či nahlížení do všech Výkazů práce.

  Systémová role má tyto položky: *Name* (Název).
- **Work plan** (Pracovní plán) umožňuje alokovat Zaměstnance na určitou Nabídku nebo Aplikaci. Díky tomu je možné zajistit, že práce bude včas a řádně dokončena, jelikož jí bude věnováno patřičné množství práce.

  Pracovní plán má tyto položky: *Month* (Měsíc), *Year* (Rok), *MD funds* (Rozpočet MDs).
- **Working time** (Úvazek) každému Zaměstnanci přiřazuje určitý počet hodin, které by měl za 1 týden odpracovat. To je podstatné vědět pro účely plánování práce.

  Úvazek má tyto položky: *Hours per week* (Hodin za týden), *Name* (Název)..
- **Work report** (Pracovní výkaz) je pravidelně vyplňován každým Zaměstnancem. Ten vždy uvede mj. i Nabídky nebo Aplikaci, na které pracoval, a jak dlouho. Díky tomu je možné sledovat, kolik MDs ještě v rozpočtu zbývá, zda bude nutné doobjednat další MDs a podobně.

  Pracovní výkaz má tyto položky: *Day* (Den), *Month* (Měsíc), *Year* (Rok), *Description* (Popis), *Hours* (Počet hodin).

3.2 Diagram nasazení

Digram nasazení B.1 zachycuje artefakty, které budou v rámci systému existovat, a fyzická zařízení, na kterých budou provozovány.

3.2.1 Artefakty

Výstupem této práce jsou 3 artefakty: Frontend, Backend a STS.

Frontend je zodpovědný za prezentaci dat uživateli, poskytuje tedy grafické uživatelské rozhraní. Kromě zobrazování informací umožňuje především manipulaci s daty, tj. vytvářet nová a upravovat a mazat stávající.

Frontend je spustitelný v libovolném prohlížeči podporujícím současné webové standardy. Nezbytnou podmínkou je podpora JavaScriptu; Cookies není vyžadováno. Frontend bude přizpůsoben jak desktopovým, tak i mobilním zařízením díky responzivnímu GUI.

Backend poskytuje služby frontendu. Skrze REST API [10] umožňuje perzistovat změny dat do databáze. Zároveň zajišťuje, že operace s daty budou povoleny pouze autorizovaným uživatelům.

Úkolem **STS** je jednak vydávat autorizační tokeny Frontendu na základě uživatelem zadaných přihlašovacích údajů a jednak validovat již vydané tokeny, aby Backend mohl autorizovat požadavky z Frontendu.

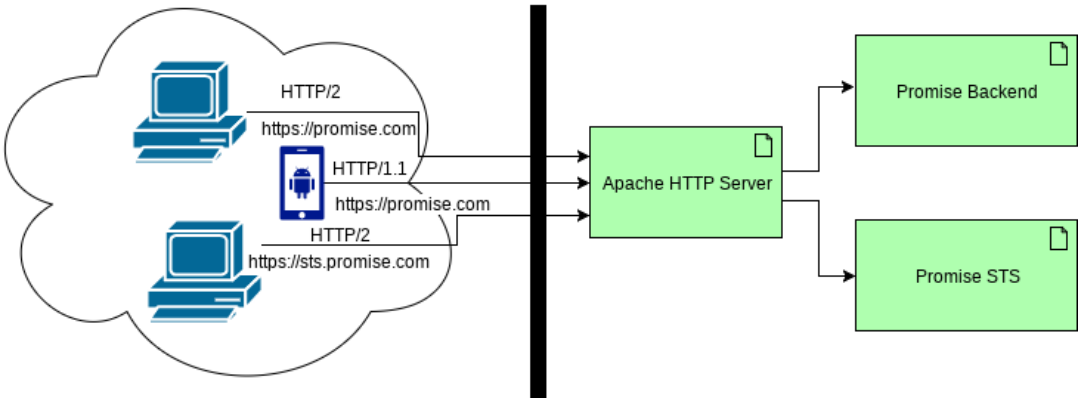
3.2.2 Infrastruktura serveru

Jako databázový server bude použit **PostgreSQL** 9.6 [11]. Jedná se o objektově relační databázový systém, který je více než 15 let vyvíjen a zveřejňován jako open source. Důrazně dodržuje SQL standardy a nabízí řadu rozšíření.

Aplikačním serverem bude **Apache Tomcat** 8 [12–13]. Ten je open source implementací řady Java technologií, včetně Java Servlet. Vyznačuje se jednoduchostí nastavení, nízkými nároky na hardwarové zdroje a vysokou stabilitou a bezpečností.

Na aplikačním serveru budou nasazené artefakty backendu a STS, ačkoliv technicky je možné oba provozovat i na zcela oddělených fyzických serverech.

Podstatnou roli bude mít i **Apache HTTP server** 2.4 [14]. Ten bude vstupní bránou mezi internetem a aplikačním serverem. Jeho hlavním úkolem bude zajistit šifrování komunikace do internetu (HTTPS), dále nabídnout použití nového HTTP/2. Také se bude starat o virtual hosting [15], tedy možnost provozovat vícero aplikací pod různými doménovými jmény na stoji s jednou IP adresou.



Obrázek 3.2. Ilustrace využití Apache HTTP serveru jako vstupní brány do systému.

3.3 Architektura backendu

Při návrhu architektury backendu byl dodržován vrstevnatý přístup. Systém se skládá celkem ze 4 vrstev:

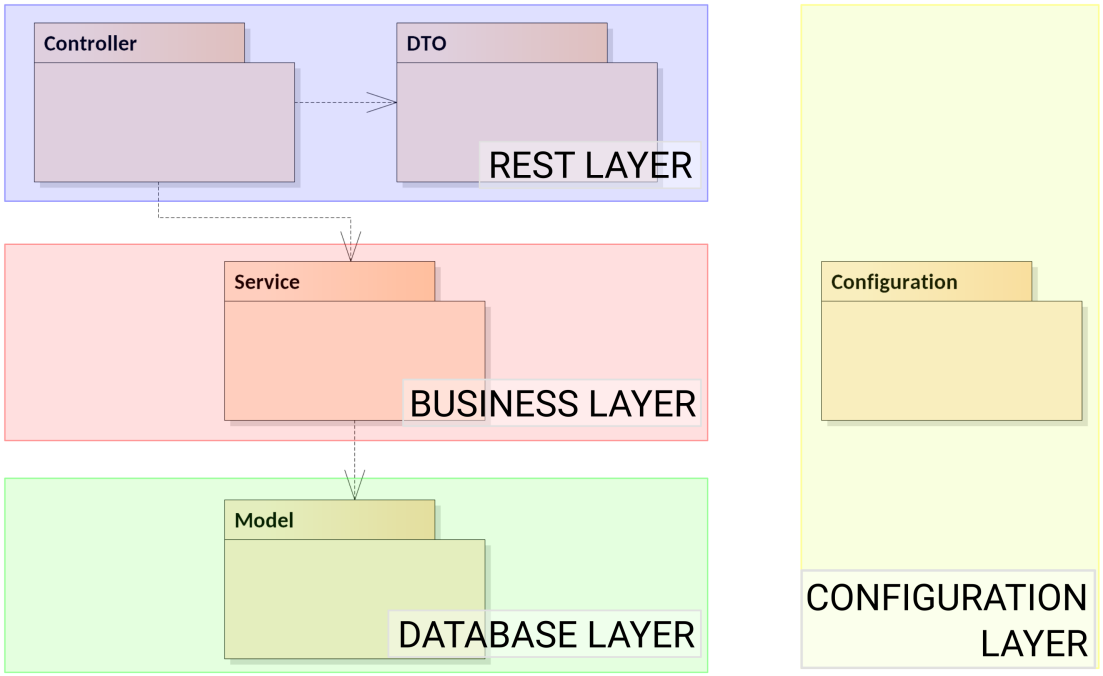
1. databázové,
2. businessové,
3. RESTové,
4. konfigurační.

Nejnižší vrstvou je databázová. Ta obsahuje především Plain Old Java Objects (POJO), tedy jednoduché datové třídy, které budou složit jako reprezentace databázových entit při objektově-relačním mapování.

Nad ní se nachází business vrstva, která poskytuje repozitáře pro práci s databázovými objekty a business logiku.

Nejvyšší vrstvou je RESTové rozhraní, jehož součástí jsou Data Transfer Objects (DTO) pro reprezentaci JavaScript Object Notation (JSON) formátu. Dále obsahuje např. logiku pro konvertování mezi DTO a POJO či zabezpečení.

Konfigurační vrstva se rozprostírá napříč všemi patry, jelikož ovlivňuje různé aspekty systému.



Obrázek 3.3. Diagram architektury backendu – vrstvy a balíčky

3.4 Případy užití

V systému byla identifikována řada případů užití pro každého aktéra, jejich detailní přehled lze nalézt v diagramech B.2 a B.3. Nejdůležitější z nich byly níže rozpracovány do detailních scénářů.

3.4.1 Scénář vytvoření záznamu ve Výkazu práce

Zadání: Vytvořte výkaz práce ze dne 7. 4. 2018 v oddělení *Pracanti*. Jako Aplikaci zvolte *Promise* a jako popis činnost uveďte *Zaškolení do aplikace*. Počet hodin nastavte na 4. Takto vyplněný Výkaz uložte.

Vstupní podmínky:

1. Uživatel je přihlášený do oddělení *Pracanti*.
2. Uživatel má přiřazenou roli pro vytváření Výkazů práce.
3. V oddělení *Pracanti* existuje aplikace *Promise*

Kroky scénáře:

1. Uživatel v menu aplikace zvolí možnost Pracovní výkazy.
2. Systém zobrazí uživatelovy záznamy z aktuální měsíce.
3. Uživatel zvolí možnost Vytvořit nový Výkaz práce.
4. Systém zobrazí formulář pro zadání informací o novém záznamu.
5. Uživatel zvolí datum 7. 4. 2018.
6. Uživatel vyplní popis činnosti *Zaškolení do aplikace*.
7. Uživatel klikne do políčka pro výběr Aplikace.
8. Systém zobrazí dostupné Aplikace.
9. Uživatel zvolí aplikaci *Promise*.
10. Uživatel nastaví počet hodin na 8.
11. Uživatel požádá systém o uložení záznamu.

12. Systém uloží nový záznam.
13. Systém zobrazí nově vytvořený záznam.

Výstupní podmínky:

1. Záznam Výkazu práce je vytvořený.
2. Záznam Výkazu práce je zobrazený.

3.4.2 Scénář alokace Zaměstnance na zvolenou Nabídku

Zadání: Alokujte zaměstnance *Jan Novák* na nabídku *Lukrativní nabídka* na 40 hodin za měsíc květen 2018 v rámci oddělení *Pracanti*.

Vstupní podmínky:

1. Uživatel je přihlášený do oddělení *Pracanti*.
2. Uživatel má přiřazenou roli pro vytváření Pracovních plánů.
3. Existuje zaměstnanec *Jan Novák* a je přiřazený na oddělení *Pracanti*.
4. V oddělení *Pracanti* existuje nabídka *Lukrativní nabídka*.

Kroky scénáře:

1. Uživatel v menu aplikace zvolí možnost Pracovní plány.
2. Systém zobrazí seznam Pracovních plánů na aktuální měsíc.
3. Uživatel zvolí možnost Vytvořit nový Pracovní plán.
4. Systém zobrazí formulář pro zadání informací o novém záznamu.
5. Uživatel zvolí měsíc květen 2018
6. Uživatel zvolí zaměstnance *Jan Novák*.
7. Uživatel nastaví počet hodin na 40.
8. Uživatel zvolí nabídku *Lukrativní nabídka*.
9. Uživatel požádá systém o uložení záznamu.
10. Systém uloží nový záznam.
11. Systém zobrazí nově vytvořený záznam.

Výstupní podmínky:

1. Záznam Pracovního plánu je vytvořený.
2. Záznam Pracovního plánu je zobrazený.

3.5 Proces přihlášení uživatele

Přihlašování probíhá přes protokol OAuth2 [16]. V systému definujeme 3 strany: *Autorizační server*, *Server se zdroji* a *Klient*.

Autorizační server spravuje přihlašovací údaje uživatelů, vydává tokeny Klientům a ověřuje platnost již vydaných tokenů. Roli Autorizačního serveru má komponenta STS (Security token service).

Server se zdroji poskytuje Klientům data. Klient, respektive uživatel používající daného klienta, se autorizuje pomocí tokenu, který mu vydal Autorizační server. Server se zdroji tento token u Autorizačního serveru při každém požadavku ověří, získá identitu Uživatele a pokud je vše v pořádku, odpoví Klientovi požadovanými daty. Roli Serveru se zdroji má Backend.

Klient prezentuje data uživateli a umožňuje mu s nimi provádět operace. K tomu využívá Serveru se zdroji, u nějž se autorizuje tokenem. Roli Klienta má Frontend.

V našem případě může přihlášení proběhnout dvěma způsoby:

1. Klient buď nemá uložený žádný token, nebo má uložený token již neplatný.
2. Klient má uložený validní token.

3.5.1 Přihlášení bez platného tokenu

V případě, že Klient buď nemá uložený žádný token, nebo token uložený má z předchozí relace, avšak ten již expiroval, je nutné požádat Autorizační server o token nový.

Tento případ nastane, pokud se Uživatel přihlašuje prvně nebo po delší době, kdy mu token již expiroval.

Proces přihlášení pak proběhne dle diagramu B.4.

Jakmile si Uživatel zapne Klientskou aplikaci, ta zkontroluje přítomnost a platnost tokenu. Zjistí, že je potřeba vyžádat si token nový. Přesměruje tedy Uživatele na STS, která si vyžádá Uživatelovu autentizaci – požádá ho o uživatelské jméno a heslo. Pokud jsou přihlašovací údaje správné, přesměruje Uživatele zpět na Klientskou aplikaci. Součástí přesměrované URL bude i nový token.

Klient extrahuje token z URL a uloží si ho pro případné použití v další relaci. Následně si Klient vyžádá od Serveru se zdroji informace o uživateli, kterému tento nový token patří.

Server se zdroji token ověří u Autorizačního serveru, který mu, pokud je platný, vrátí uživatelské jméno. Server se zdroji na základě poskytnutého uživatelského doplní další údaje o Uživateli, které si eviduje, a vše vrátí Klientovi.

V tuto chvíle je Klient správně inicializovaný a připravený obsloužit Uživatele.

3.5.2 Přihlášení s platným tokenem

V tomto případě Klient zná validní token. Tato situace nastane například tehdy, kdy uživatel obnovil webovou stránku v prohlížeči stisknutím klávesy F5.

Proces přihlášení pak proběhne dle diagramu B.5.

Jakmile si Uživatel zapne Klientskou aplikaci, ta zkontroluje přítomnost a platnost tokenu. Zjistí, že token má a není expirovaný.

Vyžádá se tedy od Serveru se zdroji informace o uživateli, kterému tento token patří.

Server se zdroji token ověří u Autorizačního serveru, který mu, pokud je platný, vrátí uživatelské jméno. Server se zdroji na základě poskytnutého uživatelského doplní další údaje o Uživateli, které si eviduje, a vše vrátí Klientovi.

Pokud token platný nebyl, Autorizační server odmítne poskytnout Serveru se zdroji informace o Uživateli a Server se zdroji vrátí Klientovi chybu. Klient pak postupuje dle předchozího scénáře, kdy nemá platný token.

V tuto chvíle je Klient správně inicializovaný a připravený obsloužit Uživatele.

Kapitola 4
Použité technologie

Při vývoji současných robustních aplikací vývojáři používají různé nástroje a řadu frameworků a knihoven. Ty přináší mnoho výhod, především zrychlení procesu vývoje, jelikož odpadá nutnost implementovat mnoho funkcionalit. Dalšími výhodami jsou například vyšší spolehlivost (dobrý framework bývá zpravidla kvalitně otestován) či bezpečnost (mnoho knihoven je open source, což umožňuje komunitě provádět kontrolu a opravy kódu).

Nejdůležitější z technologií, které byly použity při implementaci Promise, jsou popsány v této kapitole.

4.1 Backend

Hlavním programovacím jazykem pro vytvoření backendu byl zvolen Kotlin ve spolupráci s frameworkem Spring Boot.

4.1.1 Kotlin

Kotlin [17] je staticky typovaný programovací jazyk s otevřeným zdrojovým kódem, jenž cílí na JVM, Android, JavaScript a nativní platformy. První oficiální verze 1.0 byla uvolněna v únoru 2016, vývoj však začal již v roce 2010. Jazyk je zaštiťován a převážně vyvíjen společností JetBrains.

Mezi hlavní výhody Kotlinu patří především jeho stručnost. Odhady indukují zhruba 40% snížení počtu řádků kódu oproti Javě. Další velkou výhodou je vyšší typová bezpečnost v porovnání s Javou. Té je docíleno podporou tzv. non-null datových typů. To znamená, že datový typ implicitně nemůže nabývat hodnoty *null* a pokud tuto hodnotu chceme mít možnost využívat, musíme to explicitně vyjádřit. Díky tomu jsou aplikace méně náchylné na *Null pointer exception*. Další výhody zahrnují chytré přetypovávání (smart casting), funkce vyššího řádu (higher-order functions), rozšiřující funkce (extension functions) či lambda výrazy.

```
1  var user: User? = null // user is nullable
2  user = loadUser("Foo Bar")
3
4  // Print user's name or 'error' if user is null
5  print(user?.name ?: "error")
6
7  // Following condition is equivalent of the line above:
8  if(user != null) {
9      // user is smart-casted from User? (nullable) to User (non-nullable)
10     print(user.name)
11 } else {
12     print("error")
13 }
14
```

```
15  // A lambda function is passed as an argument for doWithSelf function.
16  // The lambda's only one argument is implicitly named 'it'.
17  user?.doWithSelf {
18      print(it.name)
19  }
20
21  ...
22
23  // In case we can't extend User class (e.g. it's final) we can take
24  // advantage of extension functions. This one takes a function
25  // as a parameter (therefore it's 'higher-order function').
26  // That parameter function takes a user instance as a parameter
27  // and returns Unit (Kotlin's equivalent of void).
28  fun User.doWithSelf(action: (User) -> Unit) {
29      action(this)
30  }
31
```

Obrázek 4.1. Ukázky možností jazyka Kotlin – kontrola na hodnotu null (null-safety), chytré přetypování (smart cast), lambda výrazy, rozšiřující funkce (extension functions) a funkce vyššího řádu (higher-order function).

Kotlin je plně interoperatibilní s Javou. To znamená, že kód psaný v Kotlinu může volat části kódu napsané v Javě a naopak, jelikož oba dva kódy budou zkompilované do *bytecode*. Díky této vlastnosti je možné v Kotlinu využívat veškeré knihovny psané v Javě, aniž by tím docházelo k jakýmkoliv nekompatibilitám.

4.1.2 Spring Boot

Spring Boot [18] je odlehčená a zjednodušená verze Spring Frameworku. V několika modulech poskytuje nástroje pro snadnou implementaci např. RESTového rozhraní, zabezpečení aplikace, podporu pro ORM (objektově-relační mapování) či dependency injection kontejner.

4.1.3 AFRest

AFRest je framework, který adaptivně generuje strukturu UI pomocí transformací nad strukturou datové vrstvy. Detailnějšímu popisu využití se věnuje kapitola 5.

4.2 Frontend

Frontendová část webové aplikace bude vytvořena standardními jazyky pro tvorbu moderních webových aplikací, tedy trojicí HTML5, CSS3 a TypeScript transpilovaným [19] do jazyka JavaScript. Aplikační logika bude využívat frameworku Angular 5. Design aplikace bude vytvořen s využitím CSS frameworku Semantic UI.

4.2.1 TypeScript

TypeScript [20] je nadstavba nad JavaScriptem s otevřeným zdrojovým kódem vytvořená a spravovaná společností Microsoft. Jejím hlavním přínosem je zavedení statických datových typů, což umožňuje odstranit značnou část potenciálních zdrojů chyb pramenících z dynamického typování, především volání neexistujících metod a přistupování k neexistujícím vlastnostem objektů. Většina dnešních IDE dokáže našeptávat vlastnosti a metody na základě datového typu proměnných, čímž značně urychluje vývoj aplikací.

TypeScript nabízí podporu nejnovějších JavaScriptových standardů jako např. asynchronní funkce, dekorátory a lambda výrazy. Výsledný kód je transpilován do JavaScriptu kompatibilního se staršími standardy, aby byla zajištěna podpora i starších zařízení.

4.2.2 Angular 5

Angular 5 [21] je front-endový framework pro vývoj moderních webových aplikací. Využívá deklarativní zápis HTML šablon, nabízí podporu dependency injection a řadu nástrojů pro obsluhu formulářových prvků.

4.2.3 Semantic UI

Nepsaným standardem na poli CSS frameworků se v poslední době stal Bootstrap. S jeho velkou popularitou se však pojí i obsáhlost, těžkopádnost a řada dalších neduhů, kvůli kterým bylo zvoleno vhodnější řešení [22].

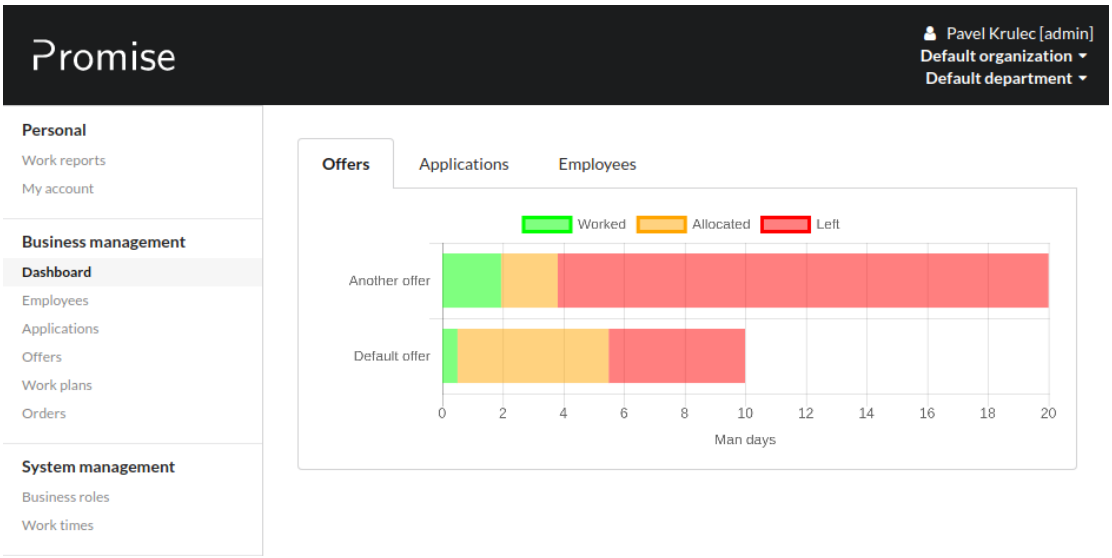
Semantic UI [23] je CSS3 framework, který používá syntax přirozeného jazyka k zápisu CSS tříd. Díky tomu je kód čitelnější a lépe se spravuje. Nabízí více než 50 modulů, z kterých je možné zvolit si jen ty potřebné a tím ušetřit množství přenesených dat, což je podstatné především pro mobilní zařízení. Originální verze vyžaduje k funkčnosti knihovnu jQuery, jejíž použití se neslučuje s filosofií frameworku Angular [24–25], proto vznikla odnož tohoto frameworku [26], která jQuery nevyžaduje.

4.3 Ukázky GUI

V této části jsou zachyceny vybrané části uživatelského rozhraní napříč systémem.

4.3.1 Dashboard

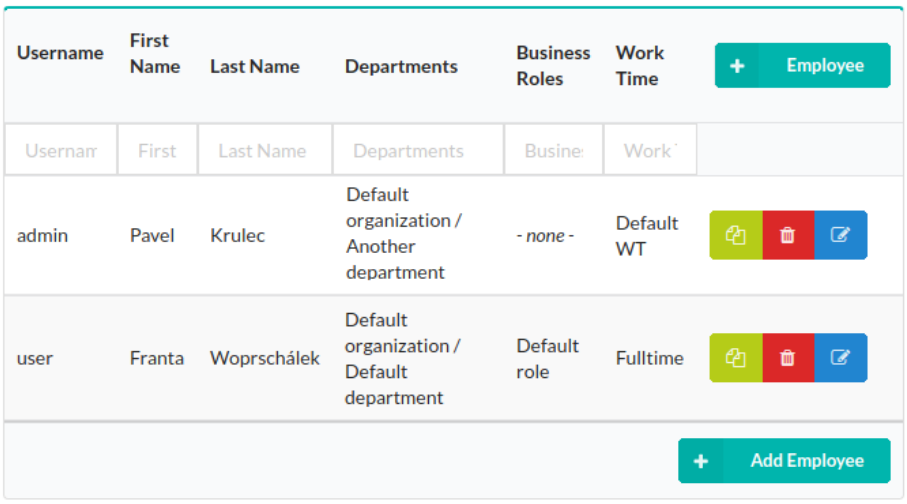
Obrázek 4.2 zachycuje manažerksý dashboard s přehledem Nabídek a Aplikací a jejich rozpočtů MDs celkem, vyčerpaných i alokovaných. Dashboard dále obsahuje přehled Zaměstnanců a jejich alokace a výkazy na jednotlivé Nabídky a Aplikace.



Obrázek 4.2. Screenshot dashboardu – přehled stavu Nabídek, Aplikací a Zaměstnanců.

4.3.2 Seznam zaměstnanců

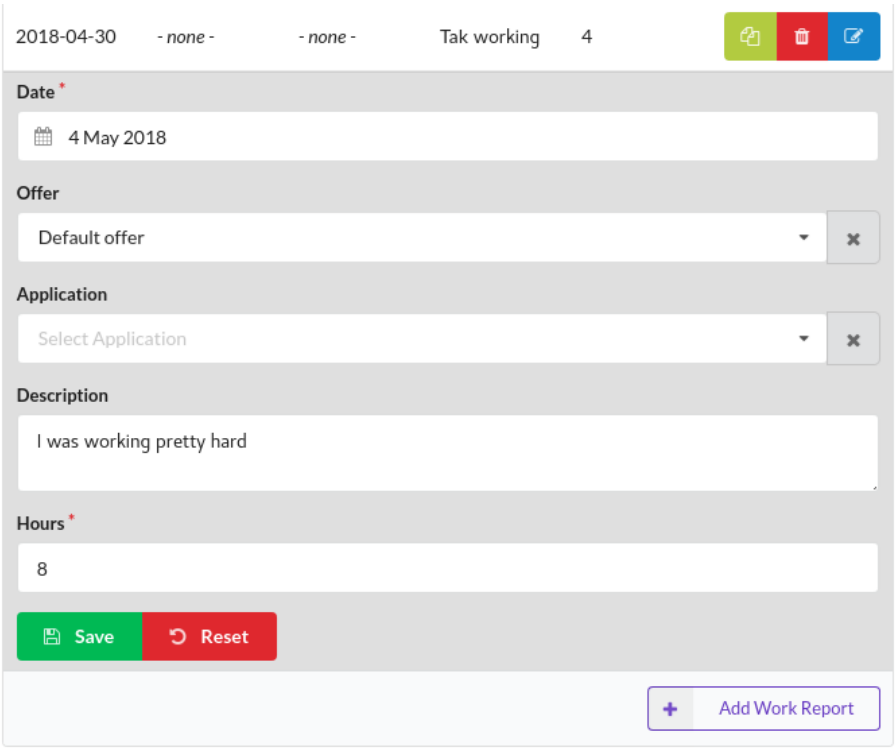
Obrázek 4.3 zachycuje seznam Zaměstnanců, jejich jmen, Oddělení, kde pracují a jaké Business role zastávají v aktuálním Oddělení a typ Úvazku, na který jsou v aktuálním Oddělení zaměstnaní.



Obrázek 4.3. Screenshot seznamu Zaměstnanců.

4.3.3 Výkazy práce

Obrázek 4.4 zachycuje formulář pro přidání nového Výkazu práce: datum, k jaké Nabídce či Aplikaci se vztahuje, popis činnosti a čas jí strávený. Uživatelé s patřičným oprávněním mohou zobrazovat a editovat Výkazy práce ostatních Zaměstnanců.



Obrázek 4.4. Screenshot seznamu Pracovních výkazů – přidání nového Výkazu.

Kapitola 5
AFRest

V rámci této práce byly mimo jiné zkoumány možnosti využití frameworku AFRest [27].

AFRest nadstavba nad frameworkem AspectFaces [28], který umožňuje generovat adaptivní UI na straně serveru. AspectFaces transformuje aktuální model na základě daného kontextu, díky čemuž je možné velmi snadno provádět změny UI v klientech, aniž by bylo třeba upravovat jejich kód [29–30].

AFRest umožňuje transformaci do platformově nezávislého formátu JSON [27], který je velmi dobře zpracovatelný ve frameworku Angular.

5.1 Integrace na backendu

AFRest bohužel není dostupný v žádném centrálním Maven repozitáři, proto je nezbytné si naklonovat Git repozitář AFSwinx [31], jehož součástí je AFRest, a z něj následně artefakt sestavit a nainstalovat do lokálního Maven repozitáře.

Pro potřeby využití v aplikaci Promise byly v AFRest provedeny drobné úpravy, například přidání nových podporovaných typů widgetů.

Následné využití v již existujícím projektu je poměrně přímočaré. Například pro vytvoření REST API na získání UI definice entity Organizace v podstatě stačí pouze vytvořit endpoint na získání definice.

Entita Organizace může vypadat například takto:

```
1  @Entity
2  data class Organization(
3          @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
4          val id: Long,
5
6          @get:UiRequired     // Mark this field as Required in UI
7          var name: String,
8
9          @OneToMany(fetch = FetchType.LAZY, mappedBy = "organization")
10         @get:UiIgnore       // Do not include this field in UI
11         var departments: MutableList<Department> = mutableListOf())
```

Entitě nastavujeme, že položka *name* má být na UI povinná a položka *departments* vůbec nemá být ve vygenerované struktuře UI obsažena. Důležité je, aby tyto anotace byly z pohledu JVM generované na getterch.

Endpoint na získání definice může být vytvořen například takto:

```
1  @GetMapping("/organization")
2  fun organization() = getDefinition(Organization::class.java)
3
4  fun <T> getDefinition(entityClass: Class<T>,
5                        mappingConfig: String = "html.config.xml")
6                                        : ResponseEntity<Any> {
```

```
7       val generator = AFRestGenerator(servletContext)
8       generator.setMapping(mappingConfig)
9
10      val data = generator.generateSkeleton(entityClass.canonicalName)
11      return ResponseEntity(data, HttpStatus.OK)
12  }
```

Zde je použita obecná transformační konfigurace *html.config.xml*, která transformuje datové typy Int a String na odpovídající widgety *Number input*, resp. *Text input*.

Pokud takto nastavené API provoláme, dostane se nám následujícího výstupu:

```
1   {
2       "classInfo":{
3           "name":"organization",
4           "fieldInfo":[
5               {
6                   "widgetType":"TEXTFIELD",
7                   "id":"name",
8                   "label":"Name",
9                   "readOnly":false,
10                  "visible":true,
11                  "rules":[
12                      {
13                          "validationType":"REQUIRED",
14                          "value":"true"
15                      }
16                  ],
17                  [...]
18              }
19          ],
20          [...]
21      }
22  }
```

5.2 Integrace na frontendu

Vygenerováním JSON výstupu práce AFRest končí a je na klientovi, jak získanou strukturu interpretuje.

V rámci provedené analýzy byla identifikována práce zabývající se integrací frameworku AspectFaces do Angular 2 [32]. Výsledky této práce není možné vzhledem k odlišnostem mezi AspectFaces a AFRest využít přímo, avšak lze na nich vhodně stavět při vytváření nových komponent, které budou kompatibilní s AFRest.

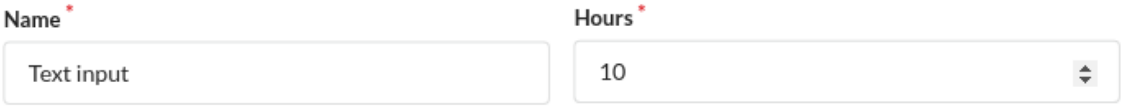
5.2.1 AFRest modul v Angular 5

Za účelem integrace AFRest do Angular byl vytvořen nový Angular modul, jenž je zamýšlen k obecnému použití – není tedy vázaný na aplikaci Promise, naopak, Promise využívá nový modul jako svou závislost.

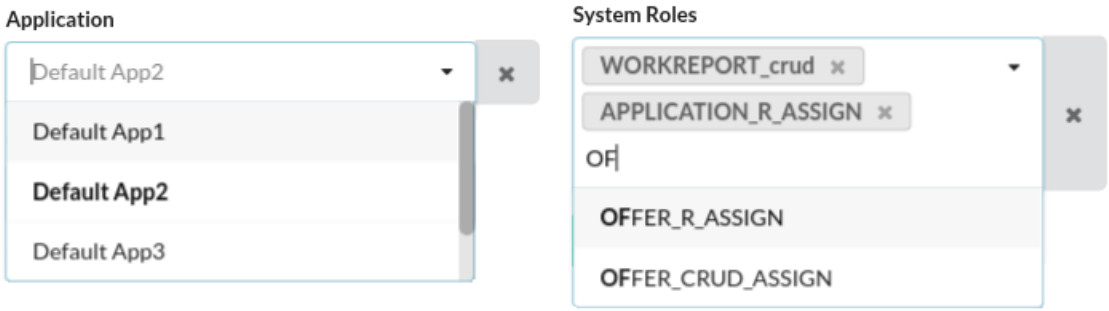
Tento modul se skládá ze šesti základních privátních komponent nedostupných mimo modul:

- AFInput,
- AFTextArea,

- AFDate,
- AFSelect,
- AFMultiSelect,
- AFFiles.



Obrázek 5.1. Vizuál AFInput komponenty v provedení pro zadání textu (vlevo) a zadání čísla (vpravo)



Obrázek 5.2. Vizuál komponent AFSelect (vlevo) a AFMultiSelect s aktivním filtrováním možností (vpravo)

Uvedené komponenty reprezentují základní formulářové prvky pro psaní textu, výběr data, výběr jedné nebo více možností z nabízených a nahrávání souborů. Na těchto komponentách staví další dvě komponenty, které jsou již veřejně dostupné i mimo modul ke koncovému využití:

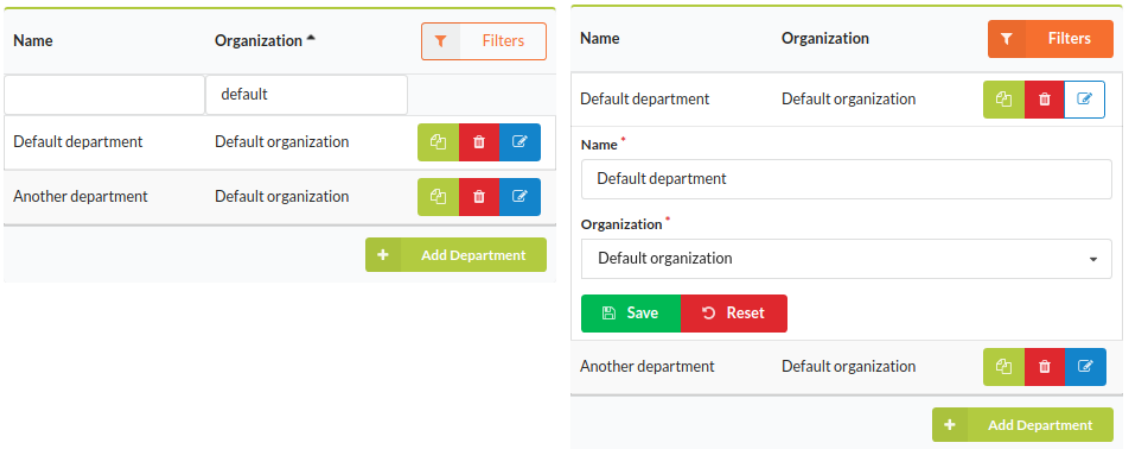
- AFForm,
- AFTable.

Komponenta AFForm reprezentuje jednu entitu a slouží především k editaci dané entity. Změny se před potvrzením provádí v pracovní kopii, díky čemuž AFForm umožňuje resetovat provedené změny bez uložení.

Na pomyslném nejvyšším hierarchickém stupni stojí komponenta AFTable, jejímž primárním účelem je prezentace seznamu entit. Zobrazené entity umožňuje dynamicky podle kterékoliv položky nejen řadit, nýbrž i filtrovat.

Dále umožňuje za využití ostatních komponent s jednotlivé entity editovat, duplikovat, odstraňovat a také vytvářet nové. To vše je plně automatizované a generované na základě dat získaných ze serveru.

Ačkoliv AFRest podporuje definici layoutů jednotlivých prvků, výše uvedené komponenty v současnosti tuto informaci ignorují používají předdefinovaný layout s využitím responzivity dosažené pomocí CSS.



Obrázek 5.3. Vizuál komponenty AFTable pro entitu Oddělení s aktivním filtrováním a řazením (vlevo) a s aktivní editací jedné ze dvou zobrazených entit (vpravo)

Kapitola 6
Testování

Každý software, od kterého je očekáván jistý stupeň spolehlivosti a funkčnosti, by měl být vhodně otestován. Testování je rozsáhlá disciplína, která pokrývá všechny fáze životního cyklu software.

Testy lze provádět na různých úrovních abstrakce [33]. Aplikace Promise je testována integračními, systémovými a uživatelskými testy; automatizovaně i manuálně.

6.1 Automatizované testování

Vzhledem ke komplexnosti a provázanosti testovaných komponent byl při vytváření testovacího prostředí důraz kladen na jednoduchost spouštění testů při zachování volnosti při jejich vytváření.

To mimo jiné v důsledku znamená, že end-to-end testy jsou spouštěny na serverové části a ne klientské, jak bývá zvykem. Díky tomu je možné, nikoliv však nezbytné, přistupovat přímo ke zdrojům (například databázi nebo RESTovým endpointům) a verifikovat na nižší úrovni.

6.1.1 Využité testovací nástroje

Hlavní podporu pro testování dodává samotný **Spring Boot**. Ten spouští embedded verzi Tomcat aplikačního serveru, na kterém běží testovaný systém, a H2 databázi v režimu *in-memory*. To znamená, že databáze není perzistentní a po ukončení testů je odstraněna. Zároveň pro každý start je databáze inicializována prázdná, bez nežádoucích dat, která by mohla testování ovlivnit.

Spring Boot dále poskytuje třídy na provolávání REST API, nastavování parametrů HTTP požadavků (především hlaviček) a ověřování odpovědí serveru.

AssertJ [34] je knihovna pro vytváření dobře čitelných *assertů* a uplatňuje se ve všech automatizovaných testech.

Komponenta STS je při testování nahrazena tzv. *mock serverem*. **WireMock** [35] je knihovna umožňující simulaci skutečného serveru, při které je možné ovlivňovat odpovědi na HTTP požadavky, tj. především rozhodovat o validitě tokenů a identitě příslušného uživatele, aniž by někde musela běžet skutečná reálná služba.

K E2E testům je použit nástroj **Selenium** [36]. Ten dokáže simulovat chování uživatele v prohlížeči a tím do vysoké míry efektivně nahradit manuální uživatelské testování. Obvykle se za tímto účelem používá nástroj Protractor, který má přímou podporu frameworku Angular a testování se spouští na frontendové části. Tím ovšem dochází ke ztrátě možnosti přímo přistupovat ke zdrojům na serveru, což se v případě použití Selenium na backendové části nestane. Nevýhodou použití Selenium je nutnost doimplementování synchronizace životního cyklu frameworku Angular, aby např. nedocházelo ke klikání na tlačítka, která ještě nebyla vykreslena.

6.2 Uživatelské testování

K uživatelskému testování bylo vybráno 6 uživatelů, kteří s aplikací Promise průběžně pracovali, zaznamenávali chyby a navrhovali změny. Všechny nalezené chyby byly opraveny a většině změnových požadavků bylo vyhověno.

Přehled 6.1 zachycuje nalezené chyby (bugs) a změnové požadavky (RFC) postupně tak, jak byly zaznamenávány.

1. **Bug:** "May days" místo "Man days" v Dashboardu
   Komentář: Správně má být "Man days"
   Vyřešeno: Ano
2. **Bug:** Nelze vytvářet Offers
   Komentář: Zastaralé databázové schéma na produkci
   Vyřešeno: Ano
3. **Bug:** Nelze vytvářet Orders
   Komentář: Zastaralé databázové schéma na produkci
   Vyřešeno: Ano
4. **RFC:** Aplikace dostupná na adrese promise.1pf.cz
   Komentář: Vytvořeny DNS záznamy a vystaven TLS certifikát
   Vyřešeno: Ano
5. **RFC:** Description u Offer má být textarea
   Komentář: -
   Vyřešeno: Ano
6. **Bug:** K Offers nelze přidávat Attachments
   Komentář: Chyběla komponenta pro správu příloh
   Vyřešeno: Ano
7. **Bug:** Vyplňování worksheetu – musím vybrat zaměstnance, abych mohl uložit nějaký výkaz – čekal bych alespoň předvyplněné moje jméno. Když to nechám prázdné, tak dostávám 401
   Komentář: Nemá být možné vytvářet výkaz někomu jinému. Má být předvyplněný a neměnný aktuální uživatel.
   Vyřešeno: Ano
8. **RFC:** Na seznamu worksheetu jsou tři ikony a není na nich popisek po najetí myší
   Komentář: Tlačítka pro editaci, duplikaci a odstranění záznamů obsahovala pouze piktogramy bez popisku
   Vyřešeno: Ano
9. **Bug:** Na seznamu worksheetu vidím, že můžu přepnout jednotlivé zaměstnance, ale když je změním, tak se nic nestane. Buď by mi je to nemělo nabízet, pokud na to nemám právo, nebo by to mělo napsat, že na to nemám právo a nebo nějaké tlačítko na potvrzení výběru.
   Komentář: Uživatel bez patřičné role nemá vidět ostatní uživatele v seznamu výkazů. Pokud roli má a přepne se na jiného uživatele, dojde k načtení výkazů automaticky.
   Vyřešeno: Ano
10. **Bug:** Když jsem na seznamu worksheetu, nejde se mi dostat zpět na Dashboard, abych viděl ty grafy
    Komentář: Uživatel bez patřičné role nemá mít k Dashboardu vůbec přístup
    Vyřešeno: Ano
11. **RFC:** Vyplňování worksheetu – když je hours povinné, tak by tam asi neměla být předvyplněná hodnota

Komentář: Byla předvyplněná hodnota 0
Vyřešeno: Ano

12. **RFC:** Vytvořit stránku na změnu STS hesla
Komentář: Změna hesla byla možná pouze manuálním HTTP requestem
Vyřešeno: Ano
13. **Bug:** Při zadávání do worksheetu reset úplně nefunguje, resetuje to, co by neměl
Komentář: Tlačítko Reset má navrátit formulář do původního stavu (včetně předvyplněných hodnot), nikoliv smazat všechny hodnoty
Vyřešeno: Ano
14. **RFC:** Lepší DatePicker – se současným se pracuje nepohodlně
Komentář: Pro výběr data/měsíce se nyní používá vhodnější komponenta namísto nativního HTML5 prvku
Vyřešeno: Ano
15. **RFC:** Worksheet přejmenovat na Work Report
Komentář: -
Vyřešeno: Ano
16. **Bug:** Pro reset MultiSelectu je třeba 2x kliknout
Komentář: -
Vyřešeno: Ano
17. **RFC:** Přidat tlačítko na odhlášení uživatele
Komentář: Zobrazí se při kliknutí na uživatelské jméno v pravém horním rohu
Vyřešeno: Ano
18. **RFC:** Přidat obrazovku s profilem uživatele
Komentář: -
Vyřešeno: Ano
19. **RFC:** Změna flow pro work report – nejprve vyplnit datum, pak nabídku, aplikaci, pak až popis a počet hodin
Komentář: -
Vyřešeno: Ano
20. **RFC:** Filter na offers
Komentář: Přidán filtr na všechny sloupečky v komponentně AFTable
Vyřešeno: Ano
21. **RFC:** Vizualizace: pokrytí offeru (plánované), vytížení lidí, skutečné a naplánované
Komentář: Patřičně upraven Dashboard
Vyřešeno: Ano
22. **RFC:** Dashboard (červená left, oranžová zůstává, zelená odpracováno)
Komentář: Barvy prohozeny (původně byly v opačném pořadí)
Vyřešeno: Ano
23. **RCF:** Dashboard – filter
Komentář: To be done
Vyřešeno: Ne
24. **RFC:** Tlačítko ADD nahoře i dole v tabulce – pro všechny tabulky
Komentář: -
Vyřešeno: Ano
25. **RFC:** Detaily k jednotlivým nabídkám – texty v tabulce jsou totiž moc velké....
Komentář: Za detail lze považovat např. rozkliknutí pro editaci nebo duplikaci
Vyřešeno: Ne
26. **RFC:** https://promise.1pf.cz/work-plans – a co když chci alokovat FTE? Nebylo by fajn mít pole buď pro hodiny nebo FTE a podle toho by se přepočetlo?

Komentář: Zadávání v jednotkách FTE ani následný přepočet FTE-MD v tuto chvíli nebudou implementovány vzhledem k automaticky generovanému UI
Vyřešeno: Ne

27. **RFC:** Viz předchozí, co když chci dlouhodobé plánování. Ať tam přibude od-do a podle toho se to bere. A od je kalendář a do také. Mít měsíc a rok zvlášť nedává vůbec smysl.
Komentář: Místo měsíce a roku zvlášť bude datePicker. OD a DO v tuto chvíli však implementováno nebude – dlouhodobé plánování lze řešit duplikací Work Planu a upravením měsíce. Tato úprava by vyžadovala zásadní změny, které v tuto chvíli nejsou z časových důvodů proveditelné
Vyřešeno: Ne
28. **RFC:** https://promise.1pf.cz/orders – náhled na offers, které jsou spojené s order (kromě jména i detaily)
Komentář: To be done – přidat tooltip s náhledem jako má např. wikipedia
Vyřešeno: Ne
29. **RFC:** Při volbě objednávky – nemůže pokrývat nabídku, která je již pokryta jinou objednávkou.
Komentář: -
Vyřešeno: Ano
30. **Bug:** Bylo by možná na úvodní stránce pro dashboard jako zaměstnance vidět pouze hodiny kam jsem vykazoval. Tzn. aby nebylo vidět, kolik je tam MD a pod.
Komentář: Uživatel bez patřičné role nemá mít přístup k manažerským informacím Nabídek a Objednávek
Vyřešeno: Ano
31. **Bug:** Zobrazit notifikaci, pokud se nepodaří načíst data GETem (při inicializaci komponent)
Komentář: Chybový response při inicializaci FE komponent způsobil nekonečný loading screen – má se zobrazit chybová hláška
Vyřešeno: Ano

Tabulka 6.1. Seznam chyb a změnových požadavků zaznamenaný během uživatelského testování.

6.3 Výkonnostní testování

Součástí testovací strategie bylo i sledování výkonu nástrojem Apache JMeter [37]. V rámci výkonnostního testování nás zajímá především průměrná doba odezvy a počet vyřízených požadavků za minutu.

Testování probíhalo na lokální vývojářském notebooku osazeném čtyřjádrovým procesorem Intel Core i7-3610QM @ 2.3 GHz s 16 GB RAM a na produkčním cloudovém prostředí se sdíleným procesorem Intel Xeon E5-2650L @ 1.70GHz a 1 GB RAM. Simulováno bylo vždy 5 současných uživatelů dotazujících se na své Výkazy práce v 300ms intervalech.

HTTP požadavek na lokálním vývojářském vývojovém prostředí vypadal takto:

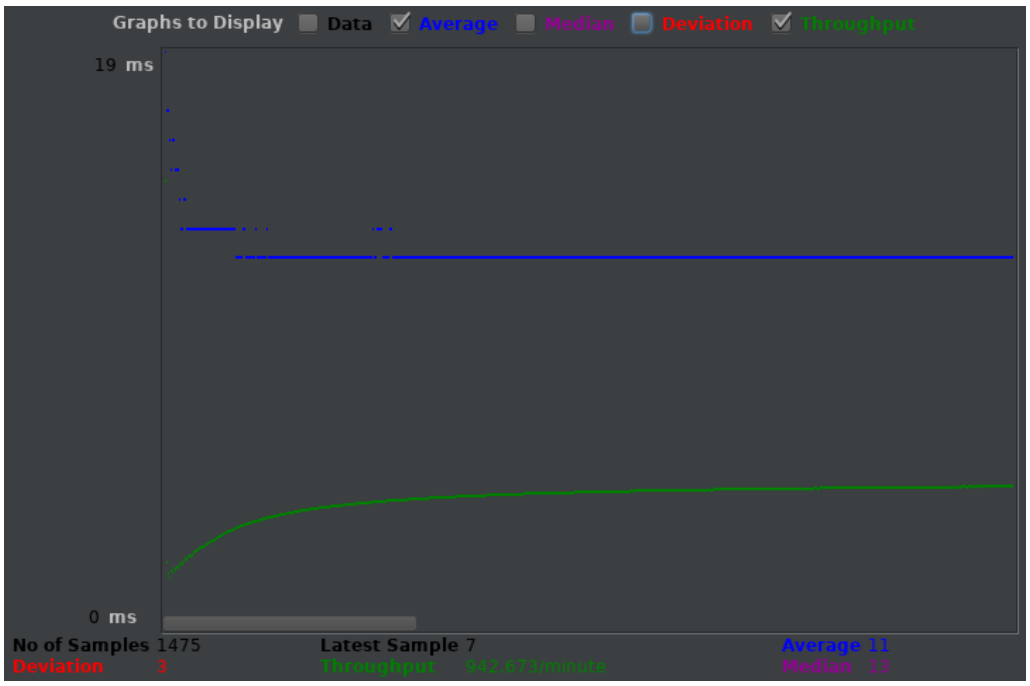
```
1 GET http://localhost:8080/promise/employees/1/work-reports
2 Authorization: Bearer ${access_token_local}
```

Pro testování na produkčním prostředí byl HTTP požadavek následující:

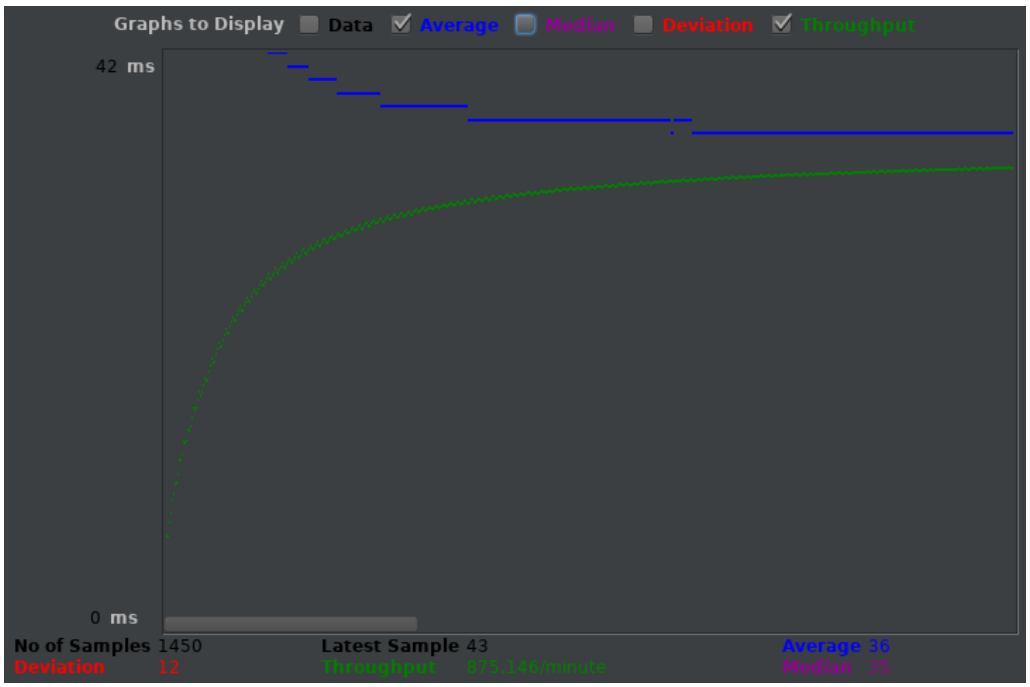
```
1 GET https://promise.1pf.cz/api/employees/1/work-reports
2 Authorization: Bearer ${access_token_prod}
```

Z grafů 6.1 a 6.2 je patrné, že zatímco na lokálním prostředí byla průměrná odezva po ustálení 11 ms, na produkci byla více než trojnásobná. Tak velký rozdíl je způsoben nutností přenášet data skrze síť internet v druhém testovaném případě. Odezva 36 ms v produkčním prostředí je velmi výborná.

Srovnatelná je i propustnost požadavků. Mírně nižší hodnoty na produkčním prostředí pravděpodobně souvisejí s poměrně nízkou velikostí RAM.



Obrázek 6.1. Výsledky výkonnostního testu na lokálním prostředí



Obrázek 6.2. Výsledky výkonnostního testu na produkčním prostředí

Kapitola 7
Nasazení aplikace

V souladu se zadáním byla aplikace již při vývoji nasazena a testována na produkčním prostředí. Prostředky (např. čistá instance Virtuálního privátního serveru, doménové jméno a pod.) byly dodány zadavatelem. Bylo však nezbytné provést veškeré potřebné konfigurace, instalace a nastavení.

7.1 Cloudové řešení

Produkční verze aplikace je nasazena na cloudovém VPS. Na dostupném operačním systému CentOS 7.3 běží aplikační server Tomcat 8, na němž je aplikace nasazena.

Databázi zajišťuje PostgreSQL 8.4, ke které se aplikace připojuje skrze aplikační server pomocí JNDI connection poolu.

Jelikož se aplikace skládá ze dvou částí, totiž samotné aplikace Promise a STS, byl pro každou z nich vytvořen virtuální host a zřízena doménová jména *promise.1pf.cz* a *sts.1pf.cz*.

Aby bylo vyhověno požadavkům na bezpečnost aplikace, bylo nezbytné nastavit komunikaci přes šifrovaný protokol HTTPS. Za tímto účelem byl zřízen bezplatný certifikát třídy 1 od autority `https://letsencrypt.org` pro již zmíněná doménová jména.

Vzhledem k v tuto chvíli již zajištěné komunikaci přes HTTPS bylo rozhodnuto, že ke komunikaci bude využita nová verze HTTP – HTTP/2. Všechny moderní internetové prohlížeče již HTTP/2 podporují [38], avšak vyžadují šifrování komunikace pomocí TLS [39]. V rámci navazování spojení přes TLS se obě strany dohodnou, zda HTTP/2 podporují a pokud ano, použijí jej.

TLS šifrování, HTTP/2 a virtuální hosty zajišťuje Apache HTTP Server.

7.1.1 HTTP/2

HTTP/2 [40] je nástupce HTTP/1.1. Na rozdíl od svého předchůdce nabízí řadu výhod. Mezi hlavní přednosti patří podpora multiplexních streamů skrze jedno navázané TCP spojení. Díky tomu může dojít ke znatelnému nárůstu rychlosti v přenosu většího počtu menších souborů oproti HTTP/1.1, kde je sice možné vyslat více požadavků skrze jedno otevřené TCP spojení, avšak pouze sériově. V případě zdržení jednoho požadavku tak ty následující musejí čekat. U staršího HTTP/1 je dokonce pro každý požadavek vytvořeno nové TCP spojení, což sice na jednu zamezovalo zásadnímu zdržení požadavků ve frontě při čekání na vyřízení předchozího požadavku, nicméně v případech, kdy si nevhodně naprogramovaná aplikace vyžádá více spojení, než je limit internetového prohlížeče (v řádu jednotek spojení na hostname [41]), může vést až k jeho pádu.

Další výhodou je komprimace HTTP hlaviček, což opět přispívá ke zvýšení rychlosti při vyřizování vícero požadavků. V neposlední řadě HTTP/2 zavádí binární kódování komunikace. Na úkor čitelnosti člověku tím opět získáme vyšší rychlost při vyřizování požadavků počítačem a zároveň nižší náchylnost k chybám.

Kapitola 8
Závěr

V rámci této práce byla vytvořena analýza aplikace Promise tak, aby odpovídala požadavkům a skutečným potřebám zadavatele. Na základě analýzy byla aplikace implementována, otestována a nasazena na produkční prostředí.

Implementace pokrývá vytvoření databázových schémat pro aplikaci Promise i STS, všech vrstev backendu aplikace Promise od databázové až po RESTové rozhraní umožňující základní CRUD operace s entitami, to vše včetně řady testů; a kompletní implementaci STS. Dále byl implementován frontend za využití frameworku AFRest k automatickému generování UI.

Aplikace byla v souladu s analýzou nasazena a zprovozněna na serveru dodaném zadavatelem, včetně veškerých potřebných konfigurací na serveru.

Vznikla i raná verze mobilní aplikace pro operační systém Android. Vzhledem k faktu, že webová aplikace byla tvořena s ohledem na použitelnost v mobilních zařízeních, doporučuji místo dalšího rozvoje mobilní aplikace soustředit se na další rozvoj v oblasti použitelnosti webové aplikace na mobilních zařízeních.

Následující vývoj by dále mohl probíhat směrem rozšiřování manažerského přehledu, refaktoringu vrstvy RESTového rozhraní či analýzou získaných dat.

Literatura

[1] Islam, Mohammad a Vinodani Katiyar. DEVELOPMENT OF A SOFTWARE MAINTENANCE COST ESTIMATION MODEL: 4 TH GL PERSPECTIVE. *International Journal of Technical Research and Applications*. 2014, č. 2, s. 65-68. ISSN 2320-8163.

[2] *Choosing an SSO Strategy: SAML vs OAuth2* [online]. [vid. 21. 4. 2018]. https://www.mutuallyhuman.com/blog/2013/05/09/choosing-an-sso-strategy-saml-vs-oauth2/.

[3] *Jira* [online]. [vid. 6. 1. 2018]. https://www.atlassian.com/software/jira.

[4] *Aceproject* [online]. [vid. 6. 1. 2018]. https://www.aceproject.com/.

[5] *Lavagna* [online]. [vid. 6. 1. 2018]. http://lavagna.io/.

[6] Komárek, Martin. Analýza a dokumentace požadavků. [online]. [vid. 6. 1. 2018]. https://moodle.fel.cvut.cz/pluginfile.php/42038/mod_resource/content/3/ANALYZA_A DOKUMENTACE POZADAVKU_3PREDNASKA.pdf.

[7] *Guideline: CRUD Matrix* [online]. [vid. 6. 1. 2018]. https://bit.ly/2r7xktk.

[8] Fowler, Martin. *Patterns of enterprise application architecture*. Boston: Addison-Wesley, 2003 [vid. 21. 4. 2018]. ISBN 03-211-2742-0.

[9] Arlow, Jim a Ila Neustadt. *UML 2 a unifikovaný proces vývoje aplikací*. 2., aktualiz. a dopl. vyd vyd. Brno: Computer Press, 2007 [vid. 21. 4. 2018]. ISBN 978-80-251-1503-9.

[10] *RESTful API Designing guidelines – The best practices* [online]. [vid. 6. 1. 2018]. https://bit.ly/2EKwqsp.

[11] *PostgreSQL* [online]. [vid. 6. 1. 2018]. https://www.postgresql.org/about/.

[12] *Apache Tomcat* [online]. [vid. 6. 1. 2018]. http://tomcat.apache.org/.

[13] *Five Reasons You Should Use Tomcat* [online]. [vid. 6. 1. 2018]. https://www.futurehosting.com/blog/five-reasons-you-should-use-tomcat/.

[14] *Apache HTTP Server project* [online]. [vid. 6. 1. 2018]. https://httpd.apache.org/.

[15] *Configuring Apache Virtual Hosts* [online]. [vid. 6. 1. 2018]. https://serversforhackers.com/c/configuring-apache-virtual-hosts.

[16] *An Introduction to OAuth 2* [online]. [vid. 6. 1. 2018]. https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2.

[17] *Kotlin* [online]. [vid. 6. 1. 2018].
https://kotlinlang.org/docs/reference/faq.html.

[18] *Spring Boot – Simplifying Spring for Everyone* [online]. [vid. 6. 1. 2018].
https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone/.

[19] KONDA, AJAY. *The difference between compiling and transpiling* [online]. [vid. 6. 1. 2018].
http://www.geekhours.com/2017/03/08/difference-compiling-transpiling/.

[20] *TypeScript* [online]. [vid. 6. 1. 2018].
https://www.typescriptlang.org/.

[21] *Angular 5* [online]. [vid. 6. 1. 2018].
https://angular.io/.

[22] *You Don't Need Bootstrap* [online]. [vid. 6. 1. 2018].
https://www.fourkitchens.com/blog/article/you-dont-need-bootstrap/.

[23] *Semantic UI* [online]. [vid. 6. 1. 2018].
https://semantic-ui.com/.

[24] *Using AngularJS? Stop using jQuery as a crutch* [online]. [vid. 6. 1. 2018].
http://www.joelhooks.com/blog/2013/07/27/using-angularjs-stop-using-jquery-as-a-crutch/.

[25] *Developing With AngularJS? Forget jQuery Exists.* [online]. [vid. 6. 1. 2018].
http://tech.zumba.com/2014/08/02/angularjs-forget-jquery/.

[26] *Ng2 semantic UI* [online]. [vid. 6. 1. 2018].
https://edcarroll.github.io/ng2-semantic-ui.

[27] TOMÁŠEK, Martin. *Aspektově orientovaný vývoj uživatelských rozhraní pro Java SE aplikace* [online]. [vid. 21. 4. 2018].
https://dspace.cvut.cz/bitstream/handle/10467/61230/F3-DP-2015-Tomasek-Martin-Tomasek-thesis-2014.pdf.

[28] *AspectFaces* [online]. [vid. 21. 4. 2018].
https://bitbucket.org/CodingCrayons/aspectfaces/wiki/.

[29] TOMASEK, Matrtin a Tomas CERNY. Automated User Interface Generation Involving Field Classification. *Software Networking*. 2017, ročník 2017, č. 1, s. 53-78. ISSN 2445-9739. Dostupné na DOI 10.13052/jsn2445-9739.2017.004.
http://www.riverpublishers.com/journal_read_html_article.php?j=JSN/2017/1/004.

[30] TOMASEK, Martin a Tomas CERNY. On web services UI in user interface generation in standalone applications. *Proceedings of the 2015 Conference on research in adaptive and convergent systems - RACS*. New York, New York, USA: ACM Press, 2015, s. 363-368 [vid. 21. 4. 2018]. Dostupné na DOI 10.1145/2811411.2811537.
http://dl.acm.org/citation.cfm?doid=2811411.2811537.

[31] *AFSwinx* [online]. [vid. 21. 4. 2018].
https://github.com/tomasma5/AFSwinx.

[32] RYŠAVÝ, Filip. *AspectFaces integration to Angular 2 framework*.
https://dspace.cvut.cz/bitstream/handle/10467/64705/F3-BP-2016-Rysavy-Filip-rysavfi1-thesis-2016.pdf.

[33] *Software Testing Levels* [online]. [vid. 21. 4. 2018].
http://softwaretestingfundamentals.com/software-testing-levels/.

[34] *AssertJ* [online]. [vid. 21. 4. 2018].
http://joel-costigliola.github.io/assertj/.

[35] *WireMock* [online]. [vid. 21. 4. 2018].
http://wiremock.org/.

[36] *Selenium* [online]. [vid. 21. 4. 2018].
https://www.seleniumhq.org/.

[37] *Apache JMeter* [online]. [vid. 21. 4. 2018].
https://jmeter.apache.org/.

[38] *HTTP/2 browser support* [online]. [vid. 21. 4. 2018].
https://caniuse.com/#feat=http2.

[39] *Can I use HTTP/2 protocol* [online]. [vid. 6. 1. 2018].
https://caniuse.com/#feat=http2.

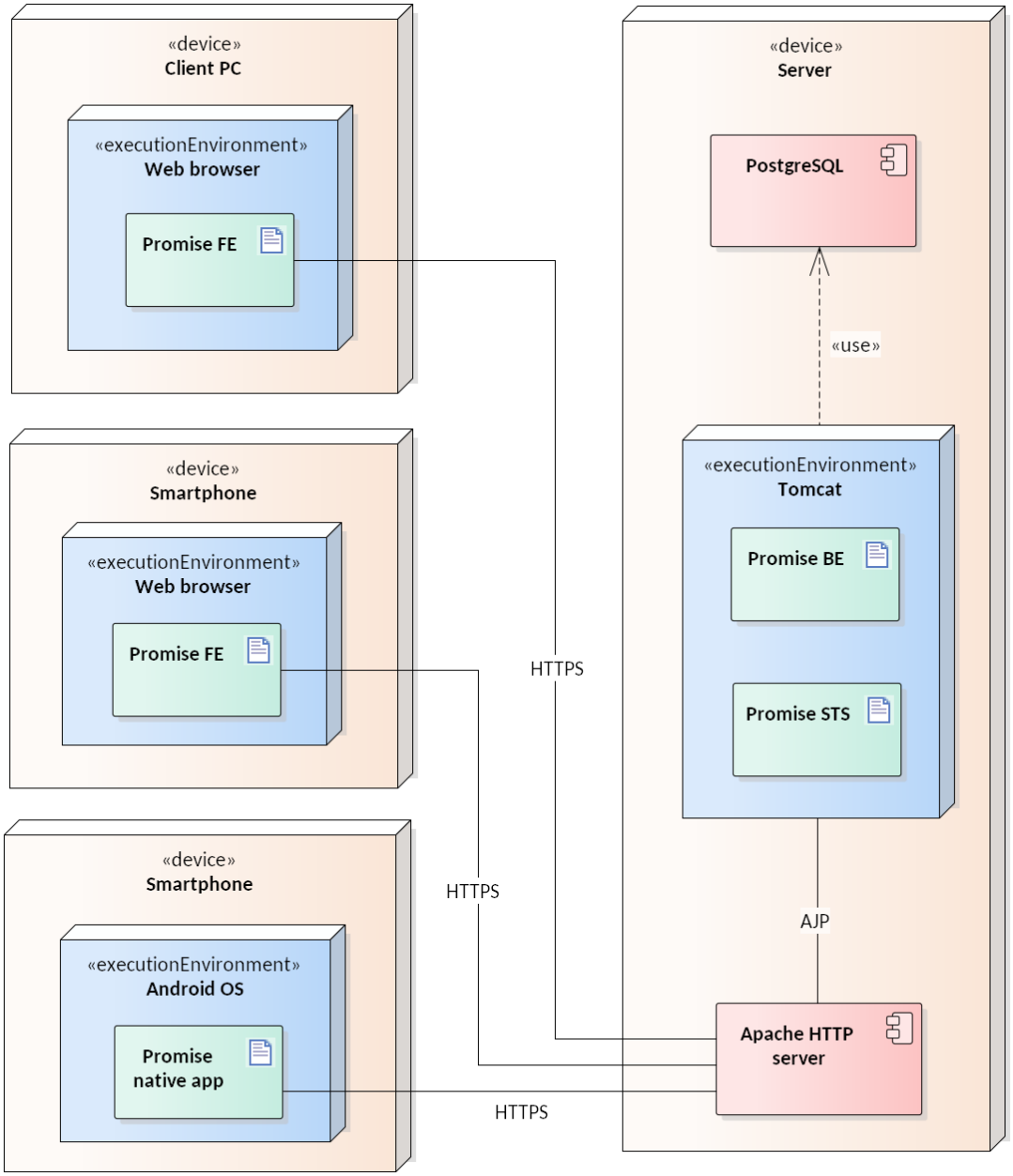
[40] *Jak funguje nový protokol HTTP/2* [online]. [vid. 6. 1. 2018].
https://www.root.cz/clanky/jak-funguje-novy-protokol-http-2/.

[41] *Browserscope* [online]. [vid. 6. 1. 2018].
http://www.browserscope.org/?category=network.

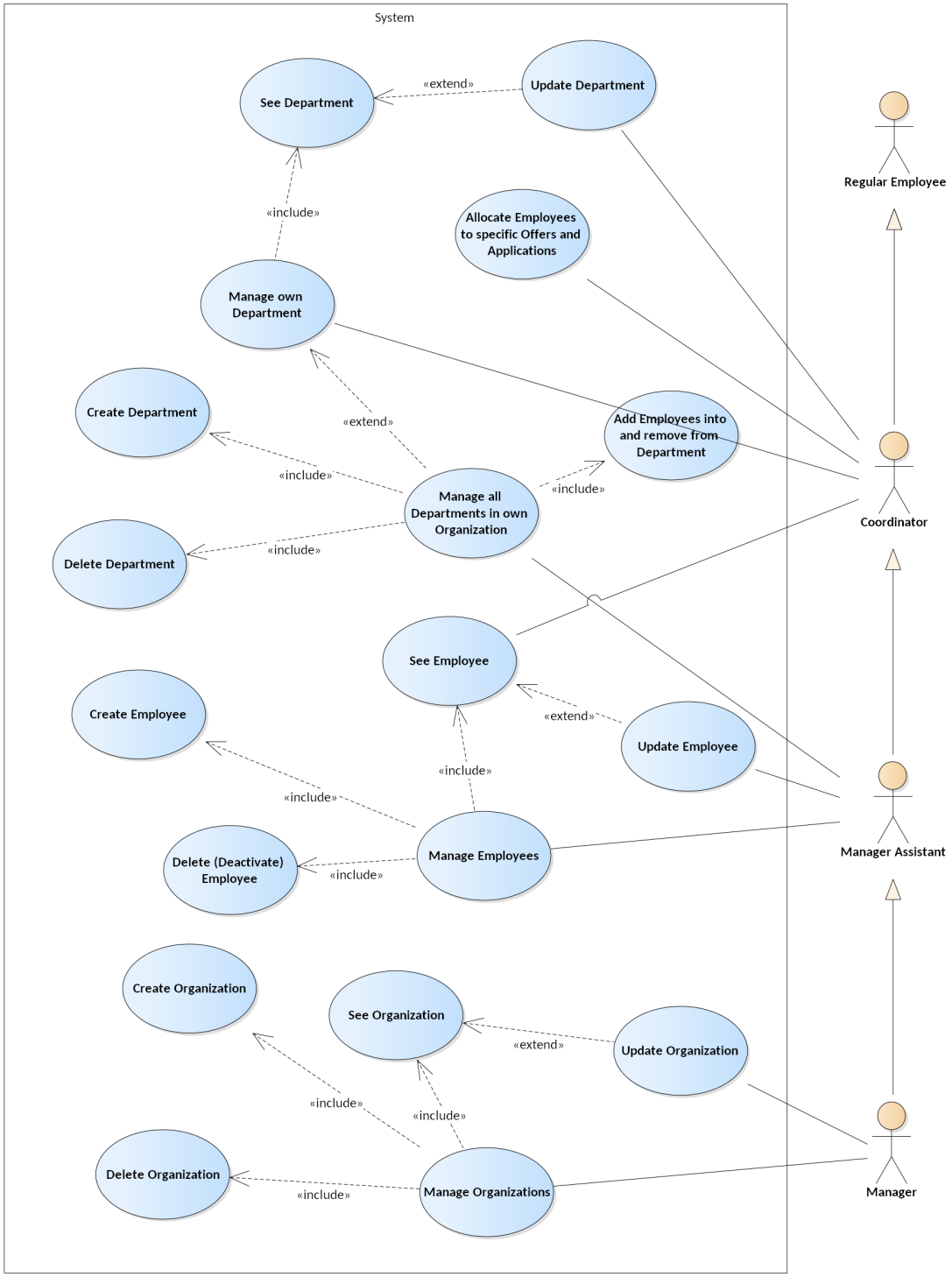
Příloha A
Seznam zkratek

- API — **Application programming interface**, popis rozhraní, které umožňuje komunikaci různých systémů mezi sebou.
- BE — **Back-end**, část webové aplikace, jež poskytuje své služby přes definované rozhraní.
- DTO — **Data Transfer Object**, datová struktura podobně jako POJO. Na rozdíl od něj je však navržena především pro přenos informací mezi různými systémy.
- E2E — **End-to-end** testy testují systém na nejvyšší úrovni abstrakce jako celek od začátku procesu až po jeho dokončení.
- FE — **Front-end**, část webové aplikace, která je viditelná uživateli.
- GUI — **Graphical user interface**, grafické uživatelské rozhraní.
- IDE — **Integrated development environment**, sada integrovaných nástrojů pro programátory. Typicky je jejich součástí textový editor se zvýrazněním syntaxe, kompilátor, debugger atd.
- JNDI — **Java Naming and Directory Interface**, API, které umožňuje přistupovat k datovým zdrojům pomocí názvu. Díky tomu odpadá nutnost konfigurace takového zdroje přímo v aplikaci.
- JSON — **JavaScript Object Notation**, platformově nezávislý formát textové reprezentace dat.
- JVM — **Java Virtual Machine**, interpret tzv. Bytecode, kódu, na který je přeložen např. Java nebo Kotlin.
- MDs — **Man day(s)**, jednotka práce. 1MD vyjadřuje množství práce, které je schopný 1 člověk vykonat za 1 pracovní den (8 hodin). Zde často použito ve spojení *rozpočet MDs* vyjadřující, kolik Man days lze z např. Nabídky čerpat.
- ORM — **Object-relational mapping**, technika umožňující mapovaní a reprezentace databázových tabulek a jednotlivých řádků na třídy, resp. instance v objektově orientovaných programovacích jazycích.
- POJO — **Plain Old Java Object**, datová struktura, která obsahuje pouze gettery a settery bez další logiky pro práci s daty.
- REST — **Representational state transfer**, architektura rozhraní využivající HTTP metod.
- RFC — **Request for change**, požadavek na změnu, např. ve funkcionalitě software.
- SSO — **Single Sign-on**, způsob autentizace, při kterém uživatel vyplní své přihlašovací údaje pouze do první navštívené aplikace a informace o přihlášení se dále přenáší do dalších uživatelem navštívených aplikací bez nutnosti se znovu přihlašovat.
- STS — **Security token service**, aplikace spravující uživatelské přihlašovací údaje. Umožňuje vydávat a ověřovat tokeny.

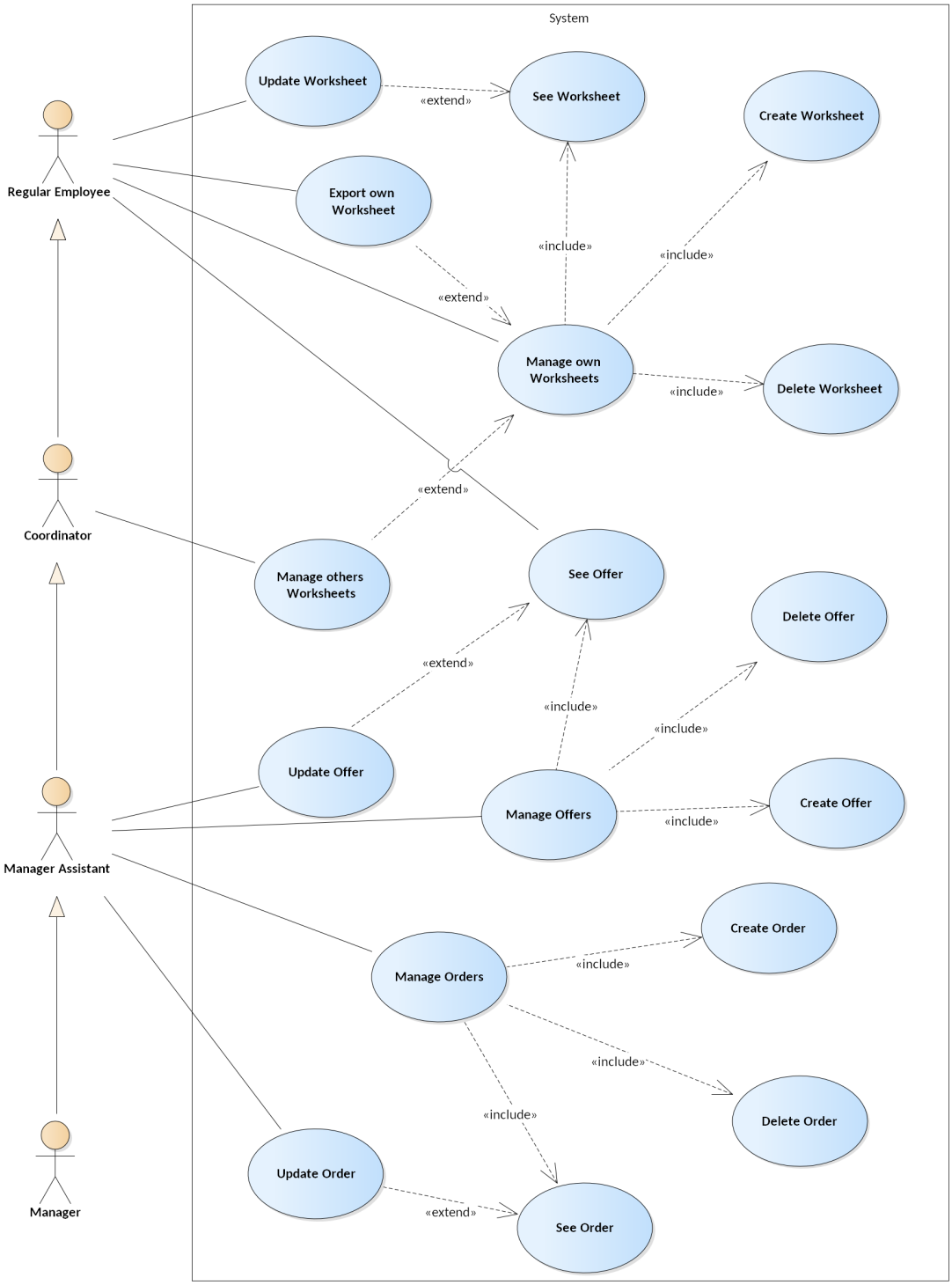
Příloha B
UML Diagramy



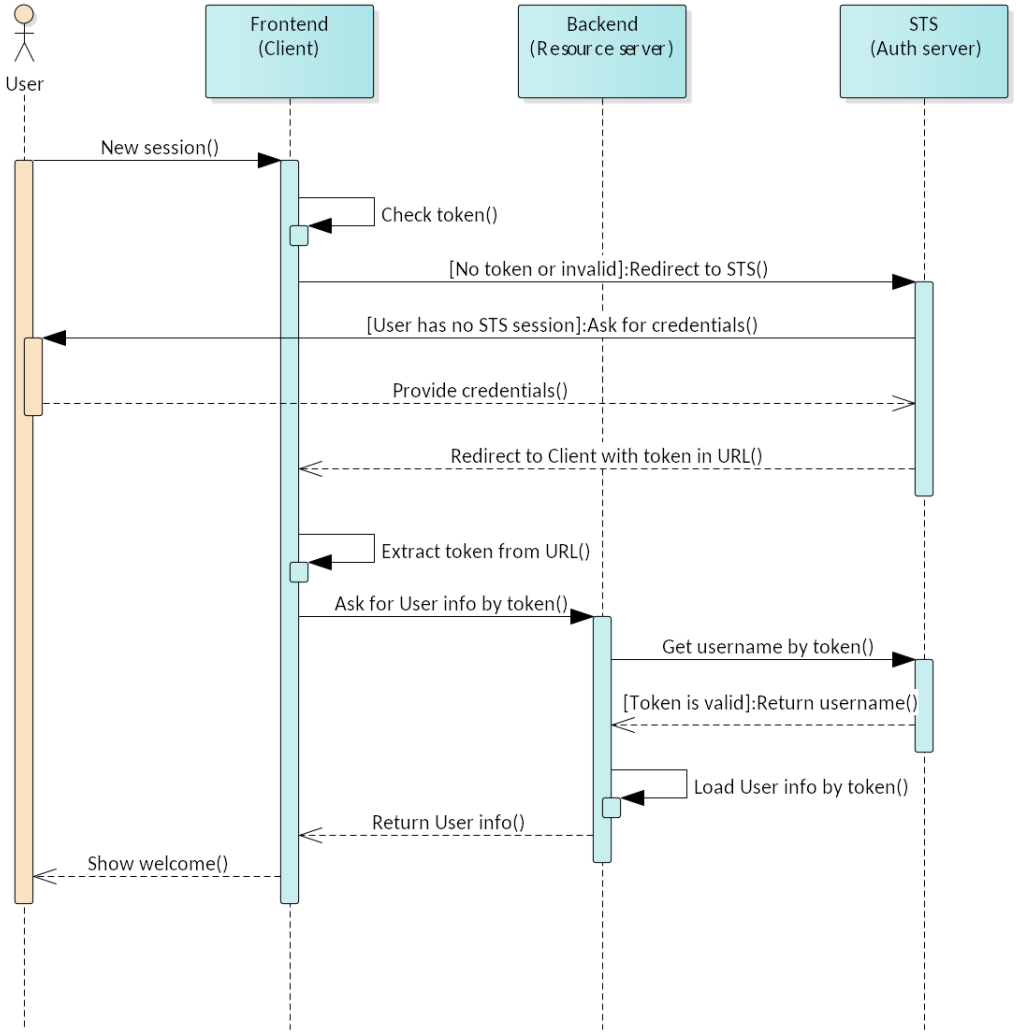
Obrázek B.1. Diagram nasazení



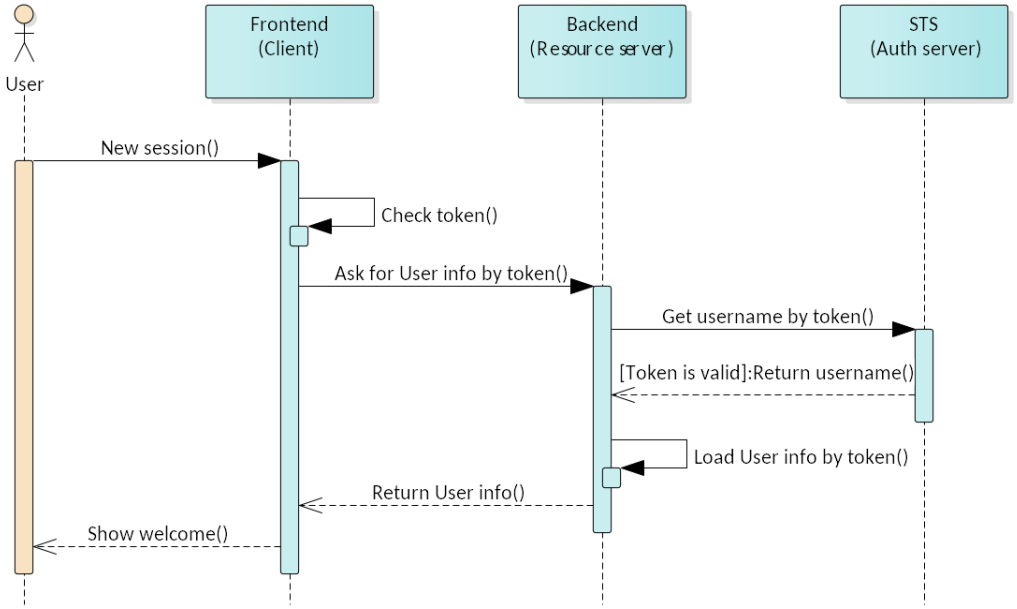
Obrázek B.2. Use case diagram – Případy užití 1/2



Obrázek B.3. Use case diagram – Případy užití 2/2



Obrázek B.4. Diagram přihlášení – Přihlášení Uživatele bez žádného tokenu nebo s tokenem expirovaným



Obrázek B.5. Diagram přihlášení – Přihlášení Uživatele s existujícím platným tokenem uloženým v Klientské aplikaci

Příloha C
Struktura přiloženého CD

```
/
|
+-+- Documents ............. # Final human readable documents
| |
| +--- promise.pdf ......... # PDF version of this thesis
| |
| +--- startup-guide.pdf ... # Prerequisites and setup instructions
| |
| +--- AFRest-demo.mp4 ..... # Showcase of automatic GUI generation
|
+-+- Sources ............... # Source codes
  |
  +-+- Web App ............. # Promise source codes
  | |
  | +--- promise ........... # Promise backend source code
  | |
  | +--- promise-fe ........ # Promise frontend source code
  | |
  | +--- sts ............... # STS source code
  |
  +--- Android app ......... # Android source code
  |
  +--- Thesis .............. # Thesis source code
  |
  +--- AFRest .............. # AFRest source code needed as dependency

```