ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Reklamační portál
Student:	Bc. Lukáš Kořán
Vedoucí:	Ing. Jaroslav Kuchař, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2017/18

Pokyny pro vypracování

Cílem práce je zjednodušení a konsolidace procesů spojených s reklamacemi produktů a služeb v různých odvětvích pomocí jednotného modulárního webového portálu. Analyzujte současnou situaci, navrhněte a implementujte vhodné řešení pro hlášení, sledování stavu a řešení reklamací, případně dalších funkcí na základě zjištěných požadavků.
1. Seznamte se s požadavky na webový portál definovanými zadavatelem.
2. Analyzujte stávající procesy v oblasti reklamací a systémy sloužící k obdobnému užití.
3. Na základě analýzy navrhněte vlastní řešení – vyberte technologie, navrhněte uživatelské rozhraní, komunikační rozhraní a datový model.
4. Implementujte a řádně zdokumentujte prototyp systému.
5. Vhodně otestujte vytvořený prototyp.
6. Zhodnoťte přínos a použitelnost vlastního řešení.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 4. prosince 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Reklamační portál

Bc. Lukáš Kořán

Vedoucí práce: Ing. Jaroslav Kuchař, Ph.D.

3. května 2017

Poděkování

Tímto bych chtěl poděkovat vedoucímu práce Ing. Jaroslavu Kuchařovi, Ph.D., za poskytnuté náměty a cenné rady v průběhu tvorby práce. Zároveň bych chtěl poděkovat své rodině a přátelům za podporu během studia a vytváření diplomové práce.

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, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 3. května 2017 .....

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Kořán, Lukáš. *Reklamační portál*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Tato diplomová práce pokrývá analýzu, návrh a implementaci systému pro správu reklamací. První část práce analyzuje existující řešení, procesy probíhající při vyřizování reklamací a požadavky na nový systém. Následující část se zabývá návrhem systému – datovým modelem, architekturou systému a návrhem komunikačního a uživatelského rozhraní. Také se zabývá výběrem vhodných technologií. Poslední část shrnuje samotný vývoj, testování, zhodnocení výsledků a možný budoucí vývoj. Výsledný systém sestává z RESTful webové služby a webové aplikace. API je založeno na platformě Spring, webový klient na frameworku Angular. Webová služba je vystavěna na principu cloudové služby s podporou řízení přístupu uživatelů dle jejich oprávnění. Klientská aplikace určená ke správě zákaznických reklamací nabízí uživatelům rozhraní založené na doporučeních Material Design.

Klíčová slova web, REST, správa reklamací, podpora zákazníků, Java, Spring, TypeScript, Angular

Abstract

This diploma thesis deals with analysis, design and implementation of an issue tracking system. First part of the thesis describes analysis of existing solutions, ongoing processes while handling customer issues and requirements for a new system. The following part describes system design and the selection of suitable technology. System design consists of data modelling, system architecture and web application programming and user interface design. The last part summarizes development, testing, evaluates results of the development and possible further improvements. The result of system development consists of RESTful web service and web application. Web API is created using Spring platform, web client is developed with Angular framework. Web service is built on cloud service principles with user right management support. The issue management client application provides users with a user interface based on Material Design guidelines.

Keywords web, REST, issue management, customer support, Java, Spring, TypeScript, Angular

Obsah

Úvod **1**

1 Analýza **3**
1.1 Analýza existujících řešení 3
1.2 Analýza současných procesů 11
1.3 Analýza požadavků 12

2 Návrh **25**
2.1 Doménový model 25
2.2 Výběr technologií 30
2.3 Datový model 35
2.4 Komunikační rozhraní 37
2.5 Architektura systému 43
2.6 Uživatelské rozhraní 44

3 Realizace **49**
3.1 Vývojové prostředí 49
3.2 Specifika implementace a použité knihovny 50
3.3 Nasazení 65
3.4 Současný stav a budoucí vývoj 66

4 Testování **69**
4.1 Automatizované testování 69
4.2 Heuristické vyhodnocení 71
4.3 Testování programátorem 74

Závěr **77**

Použité zdroje **79**

A Seznam použitých zkratek **85**

B Diagramy aktivit **87**

C Databázový model **91**

D Návrh uživatelského rozhraní **97**

E Ukázka prototypu aplikace **101**

F Obsah přiloženého CD **107**

Seznam obrázků

1.1 Ukázka prostředí YouTrack 4
1.2 Ukázka prostředí Zendesk 6
1.3 Ukázka prostředí Tustena CRM 7
1.4 Ukázka prostředí SugarCRM 8
1.5 Ukázka mobilní verze SugarCRM 9
1.6 Případy užití – administrace 18
1.7 Případy užití – práce s tickety 21
1.8 Případy užití – uživatelská sekce 24

2.1 Doménový model – tickety 28
2.2 Doménový model – systém 31
2.3 Využití Java webových frameworků 32
2.4 Responzivní vzor Column Drop 45
2.5 Návrh editace reklamace 46

4.1 Ukázka webového rozhraní Gitlab CI 72

B.1 Aktivity integrace se službami třetích stran – bez portálu 87
B.2 Aktivity správy problému – bez portálu 88
B.3 Aktivity správy problému – s portálem 89

C.1 Datový model – kontakty 92
C.2 Datový model – systém 93
C.3 Datový model – uživatelé 94
C.4 Datový model – tickety, 1. část 95
C.5 Datový model – tickety, 2. část 96

D.1 Návrh seznamu problémů 97
D.2 Návrh editace problému 98
D.3 Návrh editace kontaktu 98
D.4 Návrh vytvoření lokace 99

D.5 Návrh editace uživatele 99
D.6 Návrh editace klienta 100
D.7 Návrh nastavení klienta 100

E.1 Prototyp – editace problému 102
E.2 Prototyp – editace reklamace 103
E.3 Prototyp – seznam reklamací 104
E.4 Prototyp – editace osoby 104
E.5 Prototyp – editace firmy 105
E.6 Prototyp – seznam uživatelů 105
E.7 Prototyp – editace uživatele 106

Seznam tabulek

1.1 Souhrn vlastností jednotlivých systémů 10
1.2 Pokrytí funkčních požadavků – administrace 19
1.3 Pokrytí funkčních požadavků – práce s tickety 22
1.4 Pokrytí funkčních požadavků – uživatelská sekce 23

2.1 Angular framework – podpora prohlížečů 35

4.1 Srovnání velikosti odpovědí serveru 75
4.2 Srovnání velikosti webové aplikace 75

Seznam ukázek kódu

2.1 Reprezentace zdroje 41
3.1 Dotazy nad repozitářem Spring Data JPA 51
3.2 Použití odkazů směrovaných přes router 54
3.3 Validace číselného pole 56
3.4 Použití knihovny Http 57
3.5 Použití knihovny Flex Layout 59
3.6 Použití media queries 60
3.7 Použití knihovny Ngx–datatable 62
3.8 Hlavička Link v HTTP 63
3.9 Hlavička Link jako objekt v jazyce JavaScript 63
3.10 Použití komponenty TinyMCE 65
4.1 Příklad integračního testu 70

Úvod

Reklamace jsou nechtěnou, ale běžnou součástí našich životů. Tato práce si klade za úkol zpříjemnit a zlepšit práci s reklamacemi z pohledu zaměstnance, který pracuje v procesu zpracování reklamací. Ať už se jedná o pracovníka zákaznické podpory, či technika pracujícího na opravě, měl by mu výsledek této práce přinést užitek. Zjednodušit, sjednotit a zpřehlednit práci na reklamacích a v důsledku šetřit časové i finanční prostředky.

Z předběžného průzkumu vyplynulo, že vyhovujících aplikací zabývajících se touto problematikou není mnoho, použitelných v českém prostředí ještě méně. V důsledku vznikl návrh vypracovat nový systém v rámci diplomové práce se snahou eliminovat problémy existujících systémů. Cílem této diplomové práce je navrhnout a implementovat prototyp systému k vyřizování reklamací ve webovém prostředí s důrazem na další rozšiřitelnost. Portál bude sloužit zejména ke správě reklamací, základní správě zákazníků a správě systémových uživatelů. Systém by měl být složen z webového serveru pracujícího s daty a webového klienta. Systém by měl fungovat jako cloudová služba, která od sebe bude oddělovat klientské firmy a v rámci nich budou pracovat jednotliví uživatelé. S pomocí portálu by bylo následně možné snižovat administrativní zátěž při práci s reklamacemi a zajišťovat jejich lepší organizaci.

Práce je složena z několika částí. Z analytické, popisující analýzu existujících řešení a procesů, na jejichž základě jsou definovány požadavky na vytvářený systém. V následující části je možné se dočíst o návrhu systému, od doménového modelu mapujícího danou doménu, přes výběr vhodných technologií, až po návrh komunikačního a uživatelského rozhraní. Poslední část se zabývá samotnou realizací – specifiky implementace, použitými knihovnami, zhodnocením výsledku a záměry pro budoucí vývoj, také shrnuje testování systému. Na konci této písemné zprávy jsou přiloženy přílohy – diagramy aktivit, databázový model, návrhy uživatelského rozhraní a ukázky prototypu aplikace. Zdrojové kódy jsou dostupné na přiloženém médiu.

Kapitola 1

Analýza

1.1 Analýza existujících řešení

Před tvorbou vlastního systému je vhodné zanalyzovat již existující řešení. Zjistit jejich možnosti, funkce, dobré i špatné vlastnosti. Tato sekce se zabývá jednotlivými systémy a na závěr shrnuje zjištěné poznatky.

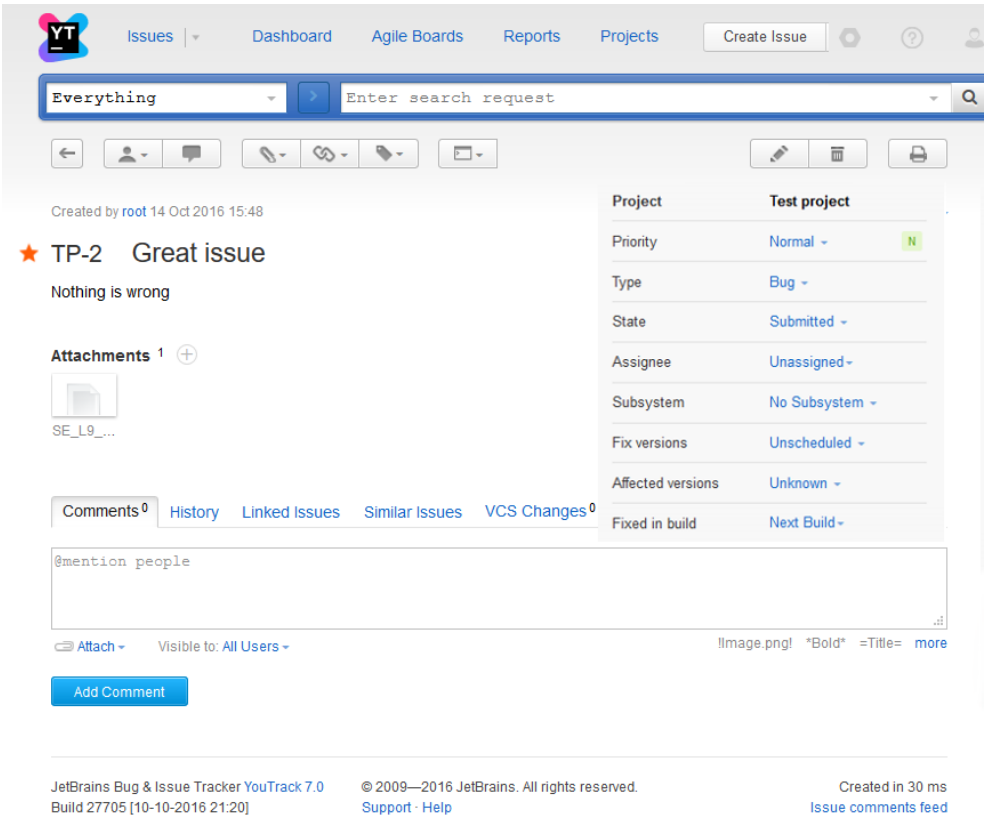
1.1.1 YouTrack

Jedná se o systém zaměřený na softwarové projekty. To znamená, že podporuje pouze podmnožinu vyžadovaných funkcí, navíc je určený pro jinou cílovou skupinu. Konkrétní cílovou skupinou jsou zejména vývojáři, testeři, manažeři a designéři. I přesto je vhodné tento nástroj zanalyzovat pro zjištění stavebních prvků systému a uživatelského rozhraní, které budou pravděpodobně obdobné.

Hlavním stavebním kamenem je hlášení problémů v aplikacích. Problémy lze vkládat k jednotlivým softwarovým projektům. V rámci problému lze specifikovat detailní popis, přikládat přílohy, prioritu, typ, případně komu je problém přiřazen. K jednotlivým problémům lze následně přidávat komentáře, štítky, související problémy, případně prohlížet historii problému a související zdrojový kód.

Centrem pro uživatele je tzv. Dashboard, který uživatel vidí po přihlášení se do systému. Zde je rychlý přehled informací, například problémy řešené přihlášeným uživatelem, vlastní poznámky, případně různé grafy. Možné je i plánování práce do etap, sprintů, organizování práce dle agilních metodik, tyto funkce jsou ale specifické vzhledem k softwarovým projektům.

Obsažena je i integrace s e-mailovou schránkou, je možné periodicky stahovat zprávy ze serveru a následně z nich automaticky vytvářet tickety, případně je předtím filtrovat podle daných kritérií. Vzhledem k cílové skupině systému ovšem není možné na e-maily přímo z rozhraní odpovídat. Ticketem bude v této práci souhrnně nazýván jakýkoliv typ zákaznického problému v systému.



Obrázek 1.1: Řešení problému v YouTrack [1]

Systém je možné používat jak ve variantě hostované v cloudu, tak ve variantě samostatného hostování. Na vlastním serveru i v cloudu je možné pro jednotky uživatelů provozovat systém zdarma, pro více uživatelů je systém zpoplatněn, ceny jsou nižší při provozu na vlastním serveru. Vzhledem k existenci verze zdarma byl vybrán k analýze tento systém, který díky bezplatné verzi může vyhovovat i velmi malým firmám. Velmi podobné alternativy, jako například JIRA, Redmine, Mantis či Trac, nebudou podrobeny podrobnější analýze. [1]

1.1.2 Zendesk

Zendesk [2] je platforma pro správu vztahu se zákazníky. Umožňuje propojení s e-maily, sociálními sítěmi, chatem a dokonce i s telefonními hovory. Poskytuje jak webové rozhraní, tak mobilní aplikaci pro komunikaci se zákazníky a správu ticketů.

Spojení s e-maily je možné jak pomocí Zendeskem vytvořených e-mailových účtů na subdoméně uživatele, tak skrze již existující e-mailové schránky. Z pří-

chozích e-mailů se následně automaticky vytvoří ticket a přímo z aplikace je možné na e-mail odpovědět, případně připsat interní poznámky. Také je možné operovat s ticketem, přiřadit mu typ, prioritu, štítky nebo přiřadit řešitele problému.

Reporting v systému obsahuje statistiky ticketů podle jejich stavu, zobrazitelné v absolutních číslech či grafech. Podporovány jsou i funkce pro samostatnost zákazníků, například sekce často kladených dotazů, aplikační rozhraní pro přístup z mobilních aplikací a webové widgety sloužící k rychlému prohledávání nápovědy, či k chatu se zákaznickou podporou. Pro zákazníky je možné i zpřístupnit chat spolupracující s ticketovacím systémem.

Systém má částečnou podporu češtiny. Koncoví zákazníci by měli mít možnost používat českou lokalizaci, jsou poskytovány šablony v češtině, případně je lze částečně upravit, nicméně uživatelé portálu si musí vybrat jeden z několika podporovaných jazyků, čeština mezi nimi není.

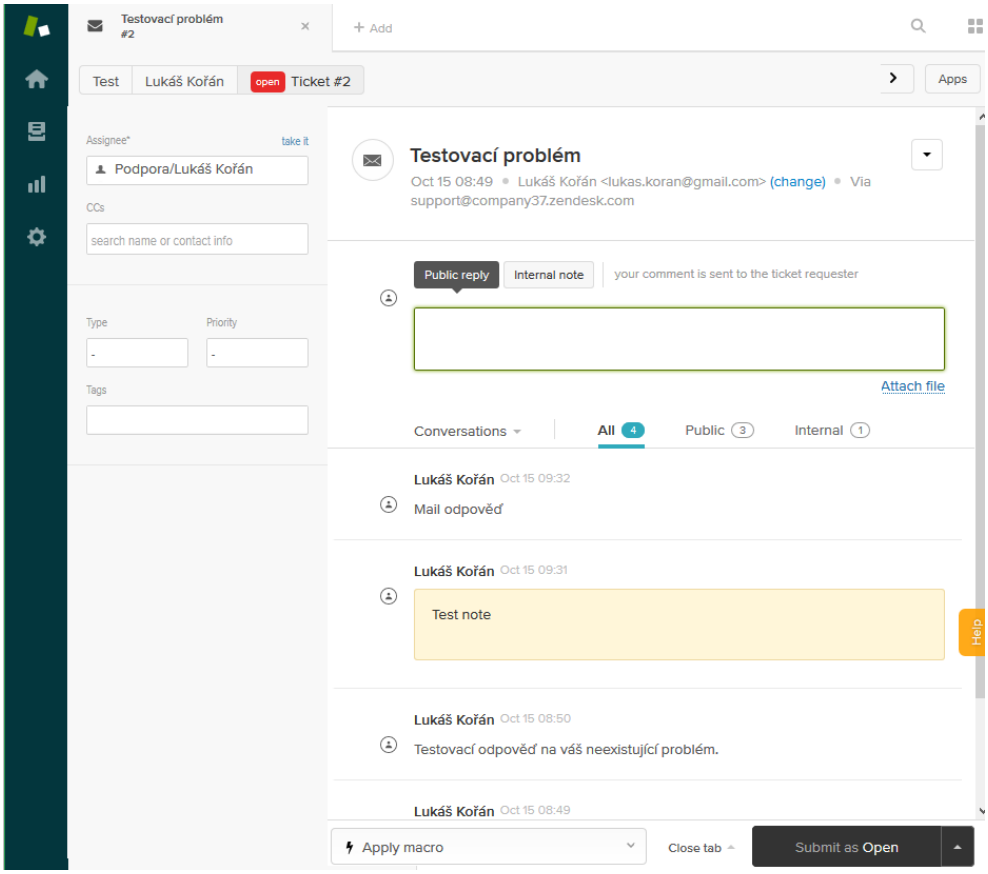
K dispozici je i nepřeberné množství doplňků, které integrují Zendesk s dalšími službami. Z těch, které se mohou hodit i uživateli našeho kýženého systému, bych jmenoval možnost vytváření ticketů z recenzí na obchodě Google Play, vytváření ticketů z e-mailů v programu Outlook ze sady Office 365. Dalším doplňkem je Mailchimp určený pro vytváření e-mailových kampaní. Pro technicky orientované uživatele je nabízen doplněk JIRA, který Zendesk propojí se stejnojmennou službou sloužící vývojářským týmům. Jeho prostřednictvím je možné zadat do systému chyby v softwaru. Některé doplňky však do jisté míry duplikují funkce samotného Zendesku. Například oficiální doplněk k aplikaci pro správu úkolů MeisterTask, případně neoficiální integrace obdobného systému Asana. V rámci další analýzy bude vyhodnoceno, zda, případně jakým způsobem je vhodné rozlišovat tickety a úkoly.

Systém nabízí bezplatné vyzkoušení na 30 dní, následně je zpoplatněn za cenu od 5 $ za uživatele, při zájmu o více jazyků, což je v ČR předpokládané, se cena pohybuje od 49 $.

1.1.3 Tustena CRM

Další systém pro CRM (Customer Relationship Management) je zajímavý tím, že poskytuje i komunitní verzi zdarma [3]. Obsahuje obdobné funkce jako jiné systémy, tedy podporu zákazníků s pomocí ticketů pro jednotlivé problémy, obligátní reporty pro analýzu dat v systému a podobně. Navíc obsahuje webový e-mail, případně posílání SMS zákazníkům. Systém je bohužel velmi starý a na to doplácí zejména uživatelské rozhraní. I verze hostovaná v cloudu má znatelně zastaralé rozhraní, u verze zdarma, která nebyla aktualizována od roku 2009, je rozhraní ještě horší. Ukázka rozhraní je k nahlédnutí na obrázku 1.3.

Nabízelo by se vystavět vlastní systém na již hotovém zdrojovém kódu tohoto CRM [4], nicméně systém je natolik komplexní a zároveň poněkud zastaralý, že by přizpůsobení pro vlastní použití pravděpodobně nebylo časově



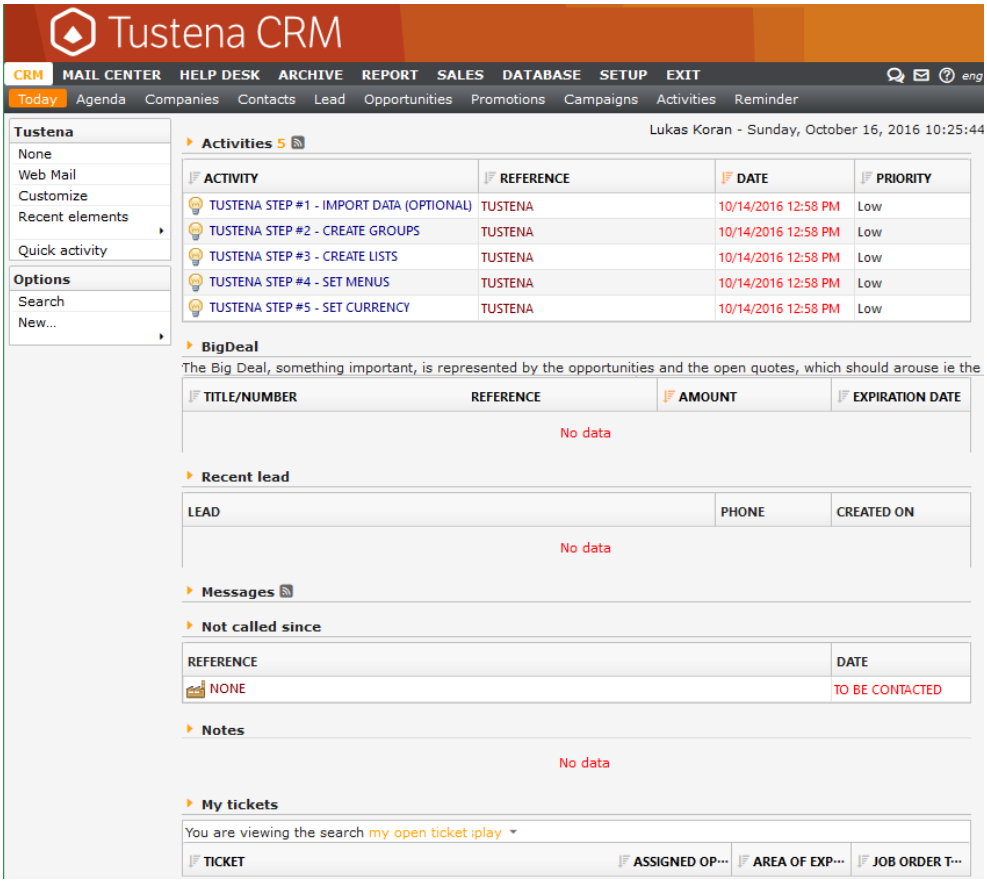
Obrázek 1.2: Řešení problému v Zendesk [2]

výhodné. Dalším problémem je vazba na technologie Microsoftu, zejména kvůli placenému OS Windows Server a SQL Server, použití tohoto řešení by v důsledku přinášelo finanční zátěž.

1.1.4 SugarCRM

SugarCRM [5] je jedním z nejkomplexnějších systémů pro řízení vztahu se zákazníky. Jedná se opět o celou platformu obsahující jak webovou aplikaci, tak i mobilní aplikace. Webová aplikace je navíc dostupná ve verzi pro stolní počítač i pro mobilní zařízení, podpora platforem je tedy velmi široká. Analýza je založena na bezplatně poskytované časově omezené verzi naplněné zkušebními daty.

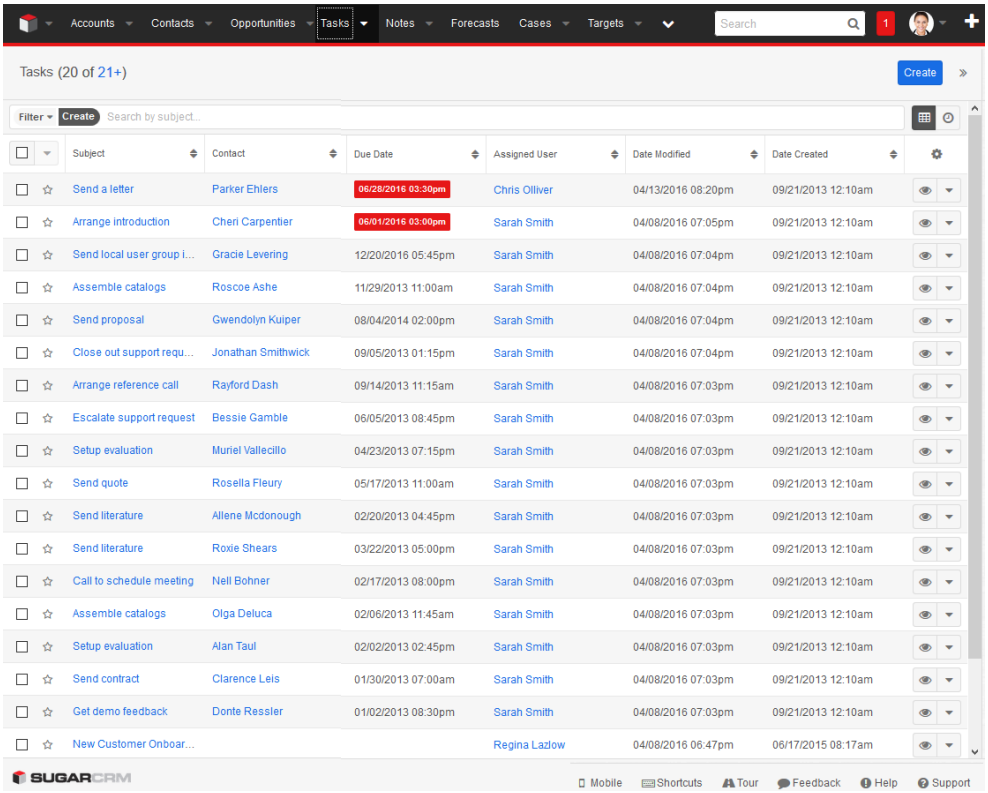
Jako jedno z mála obdobných řešení podporuje i mobilní zobrazení webové aplikace. Toto zobrazení je velmi vhodně přizpůsobeno mobilním displejům. Aplikace je díky úspornější práci s detaily i o něco svižnější, to je při práci na mobilním zařízení jistě vítané. Mobilní verze funguje na principu sezna-



Obrázek 1.3: Prostředí Tustena CRM [3]

m/detail, z položky v menu se uživatel dostane do seznamu prvků, ať už se jedná o seznam zaměstnanců, zákazníků či úkolů. Ze seznamu je dále možné se dostat do detailu prvku. Z detailu je dále možné přejít na editaci. Obdobným způsobem je strukturována i verze pro stolní počítač, detail je zde striktně oddělen od editačního módu, později bude analyzováno, zda a proč je vhodnější editace přímo v detailu, či separátní mód editace.

Webová aplikace je složena ze vzájemně více či méně propojených modulů. Pro příklad uvažujme jako uživatele aplikace větší firmu. Každý zaměstnanec může mít přístup k jiným modulům, přístupy nastaví místní administrátor. Moduly se následně zobrazují v horní liště webové aplikace, zde vzniká problém při využívání mnoha modulů. Typicky u administrátorského účtu je zobrazeno více odkazů na modul, než se vejde na obrazovku, zbylé jsou dostupné z rozevíracího menu. Nicméně uživatel pracující se systémem denně si jistě dokáže zvyknout a nezabere mu orientace v menu mnoho času. S množstvím modulů ovšem vzniká nejednoznačnost funkcionalit systému. Například uživatel může mít problém rozlišit, kam zařadit problém, zda do případů, úkolů

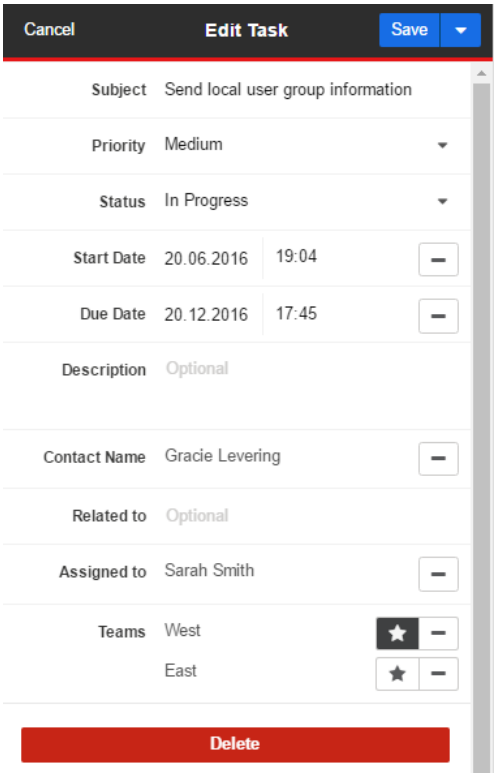


Obrázek 1.4: Prostředí SugarCRM [5]

či bugů. Dostupná je široká škála nastavení přístupových práv každému uživateli, jaké moduly či jejich části mají být přístupné.

Systém je nesmírně komplexní, nabízí editory jednotlivých modulů, formulářů, ale i jejich vytváření, spolu s vytvářením dodatečného datového modelu. Editovat je možné jak webové formuláře, tak jejich mobilní verze. To vše je dostupné přímo v administračním rozhraní. Problémem je, že ovládnutí editoru zabere běžnému uživateli mnoho hodin, pravděpodobně je třeba i přečíst dokumentaci, případně podstoupit školení. Spolu s editací je umožněna i lokalizace každého popisku formuláře, systém ve výchozím stavu podporuje i češtinu.

K dispozici je komerční verze dostupná za poplatek od 40 $ za měsíc za uživatele aplikace. Nasazení je možné jak na vlastní server, tak využití v cloudu. Existuje i verze komunitní, se kterou je možný provoz na vlastním PHP a databázovém serveru. Ta má bohužel dva problémy, prvním je ukončená podpora komunitní verze roku 2014 [6]. Druhým problémem pro vlastní nadstavbu je licence AGPL v3 [7], která nutí zveřejňovat změněné zdrojové kódy. Pro případ pozdějšího komerčního nasazení výsledku této práce je tato licence nepřijatelná.



Obrázek 1.5: Editace v mobilní verzi SugarCRM [5]

1.1.5 Agile CRM

CRM s velmi moderním uživatelským rozhraním zaměřené na tři hlavní oblasti – prodej, marketing a uživatelskou podporu. Rozhraní je striktně rozděleno na tři výše zmíněné oblasti, nicméně některé kategorie funkcí jsou sdíleny napříč těmito moduly. Takovou sdílenou kategorií jsou například kontakty.

Oblast prodeje se jeví velmi stroze, poskytuje zaznamenávání schůzek s pomocí kalendáře a úkolů. Hlavní prodejní složka sestává z kategorie obchody, které jsou ovšem velmi povrchně zpracované, bez nějakého propojení na účetní či skladový systém je tato funkce v podstatě nepoužitelná.

Oblast marketingu je na druhou stranu zpracována kvalitně, obsahuje hodnocení průchodu webových stránek klienta zákazníkem, napojení na sociální sítě, vedení reklamních kampaní, newsletterů a tak dále.

Oblast podpory je standardní, jako v ostatních analyzovaných řešení. To znamená tickety na základě e-mailů, skupiny ticketů, historie aktivit a veřejná znalostní databáze.

Nabízeno je částečné přizpůsobení systému uživatelem. Například je možné přidávat ke kontaktům či firmám vlastní pole různých datových typů, přidávat kategorie úkolů či upravovat šablony e-mailů. Hlavním kamenem úrazu je

Tabulka 1.1: Souhrn vlastností jednotlivých systémů. Legenda: CUST: Zákazník; PRG: Programátor; MNG: Manažer; SUP: Zákaznická podpora; EMP: Zaměstnanec; DP: Dodatečná pole; P: Programovatelné; NAPP: Neoficiální mobilní aplikace; OAPP: Oficiální mobilní aplikace; MW: Mobilní web; RW: Responzivní web

Funkce \ Systém	YouTrack	ZenDesk	Tustena CRM	SugarCRM	Agile CRM
Podpora češtiny	✗	CUST	✗	✓	CUST
Cílová skupina	PRG, MNG	SUP, CUST	EMP	EMP	SUP, EMP, CUST
Backend	Java	Ruby	C#	PHP	?
Úpravy	DP	DP	P	Starší P, aktuální DP	DP
Interakce zákazníka	✗	✓	✗	✗	✓
Single page	✗	✓	✗	✓	✓
Mobilní přístup	OAPP	OAPP	OAPP italsky	MW	RW, OAPP
API	REST	REST	✗	REST	REST

ovšem jazyková podpora, čeština opět není zastoupena, pouze prostřednictvím přizpůsobení šablon lze alespoň zákazníkům navenek prezentovat lokalizované zprávy.

Toto zhodnocení je sestaveno na základě verze poskytované zdarma až deseti uživatelům. [8]

1.1.6 Shrnutí

Z analýzy vyplývá, že každé řešení má své klady a zápory, funkce užitečné i chybějící. V tabulce 1.1 jsou shrnuty důležité parametry, které mají existující platformy. Největším problémem většiny existujících systémů je chybějící nebo částečná čeština. To brání rozšíření v českém prostředí, pro které je kýžená platforma vyvíjena. Hledaná je tedy pouze platforma s dostupnou češtinou či programovatelný systém, kam by bylo možné češtinu přidat. Cílovou skupinou hledaného řešení je co nejširší skupina uživatelů. S cílovou skupinou také souvisí možnost interakce koncového zákazníka se systémem, tzn. měl by nabízet i funkce pro nepřihlášeného uživatele. Také by bylo vhodné, aby systém fun-

goval v režimu takzvané single page, který se uživatelům jeví jako plynulejší. Existují případy, kdy se uživatel bude chtít do systému připojit i z mobilního zařízení, na něm by ho neměl čekat nepříjemný zážitek v podobě neoptimalizované verze právě pro tato zařízení. Pro rozšíření systému například pomocí doplňků z kancelářských aplikací, či interních systémů společnosti, je důležité i programové rozhraní (API), které umožňuje do systému zasahovat. Ideálním systémem pro absolutní kontrolu nad požadavky uživatelů, by měl být programovatelný, nebo alespoň velmi konfigurovatelný.

Z tabulky 1.1 vyplývá, že jediným analyzovaným portálem plně v češtině je SugarCRM, který je ovšem určen pouze pro vnitřní užití firmy, a proto nepodporuje snadnou interakci s neregistrovaným zákazníkem. Bylo by možné funkcionalitu doprogramovat, nicméně systém je velmi komplexní a taková, či další kýžené úpravy, by zabraly mnoho času na orientaci v daném systému a doimplementování. Navíc již open source verze systému není podporována. Dalším problémem by mohl být i programovací jazyk PHP, který je dynamicky typovaný a nepoužívá JIT kompilaci, obě vlastnosti jazyka mohou způsobovat pomalejší běh systému. Z těchto důvodů je vhodné vytvořit zcela nový systém.

1.2 Analýza současných procesů

Diskusí se zadavatelem a analýzou procesů vykonávaných pracovníky byly zmapovány současné procesy při zpracování reklamací. Diagramy aktivit jsou umístěny v příloze B. Popis v této sekci slouží k hlubšímu pochopení situace a záměrů při tvorbě systému.

1.2.1 Objednání svozu reklamací

Diagram aktivit probíhajících při objednání svozu reklamací bez podpůrného systému je k nahlédnutí na diagramu B.1. K plnému porozumění procesů je třeba doplňkový slovní komentář. Sepsání zprávy zahrnuje vždy velmi obdobné úkony, kde velká část z nich využívá totožnou šablonu. Takovou šablonu je vhodné podpořit systémem, který vede zadavatele požadavku, tento přístup má několik výhod. Standardizovaný vstup, který poslouží k rychlejšímu zadání požadavku a návodnému postupu pro nové zákazníky, ale i standardizovaný výstup, který znatelně urychlí zpracování požadavku na straně vykonavatele požadavku. Zaměstnanec v tuto chvíli nemusí pročítat celou zprávu, která zejména při velkém množství zákazníků mění strukturu, ale v rychlosti nahlédne na data již standardní struktury. Do procesů mohou být zapojeny i další integrace systémů, které mají zásadní problém se zpracováním nestrukturovaného textu, jakým e-mail bezpochyby je. Založení zakázky přepravy s využitím informačního systému může využít například přímé napojení jak na interní systém zaměstnance, tak na vzdálené rozhraní přepravce. Přepravce může pro potvrzení svozu zboží využít portál nebo se data mohou opět stáhnout pomocí integrace služeb. Bez použití systému je po obdržení potvrzení

o svozu zboží nutné ručně psát potvrzující e-mail o detailech uskutečněného svozu a zákazník ho musí po obdržení přečíst.

Navíc ke všem výše jmenovaným nedostatkům řešení bez podpůrného portálu existuje problém s historií provedených akcí. I pokud klienti a servery jak zákazníka, tak zaměstnance, umožňují vlákna e-mailů, je prohledávání a zpracování historických dat velmi nepřehledné a nepohodlné, opět z důvodu nestandardizovaného formátu dat, ale i z důvodu množství e-mailů, které se vůbec netýkají stejného požadavku.

1.2.2 Založení reklamace

Aktivity související se založením a správou reklamace jsou naznačené na diagramu B.2. Problémy jsou velmi obdobné jako v případě *Objednání svozu reklamací*, opět se potýkáme se špatně strukturovanými a spravovatelnými e-maily. Navíc při procesu reklamace často vyřizuje reklamaci více osob, což znesnadňuje orientaci. Ve chvíli prvního přeposílání již původní adresát o aktuálním stavu reklamace ztrácí přehled, maximálně je následně adresován v kopii e-mailu, což stále není ideálním řešením. Řešení skrze e-maily je problematické i z hlediska dovolených, případně nemocí. Ve chvíli, kdy přijde žádost, či upřesnění od zákazníka a zaměstnanec není delší dobu dostupný, zhoršuje se zákaznický komfort a celkový dojem o společnosti. Tyto problémy jsou řešitelné informačním systém. Místo přeposílání e-mailu je možné spravovat tzv. ticket, přiřadit ho jinému zaměstnanci, a přesto ho nadále bez problémů sledovat. Zároveň je možné v rámci skupin společnosti zavést větší kontrolu nad aktuálně zpracovávanými problémy. V rámci rychlého přehledu může být buď skupina či nadřízený, informován o neřešených problémech. To řeší problém při dovolené nebo nemoci zaměstnance. Při dalším řešení servisem, objednání svozu reklamace, je opět vše mnohem snazší s využitím portálu, mohou být využity systémové integrace objednávající svoz či automaticky informující servis. V případě malých firem může servisní oddělení používat tentýž portál a přehled o aktuálním stavu reklamace je ideální. Kýžený stav při použití portálu je k dispozici na diagramu B.3.

1.3 Analýza požadavků

V této sekci naleznete stanovené požadavky na vytvářený systém.

1.3.1 Funkční požadavky

Administrace

- **Podpora uživatelských účtů** Uživatelé budou využívat vlastní přihlašovací údaje, administrátoři budou mít možnost přidat, editovat a zablokovat uživatele.

- **Uživatelské role** Uživatelům budou administrátorem přidělovány role spolu s oprávněními.
- **Podpora více klientů** Jednotlivým klientům bude možné vytvořit podstránku, do které se budou uživatelé příslušící danému klientovi přihlašovat. Klientem je typicky firma využívající systém. Administrátor klienta bude spravovat nastavení a uživatele v rámci klienta.
- **Skupiny uživatelů** Uživatele bude možné přiřadit do skupin, které mohou reprezentovat vnitřní uspořádání struktury klienta. Tyto skupiny mohou být užitečné například pro přiřazení ticketu na oddělení, případně pro supervizi problémů zpracovávaných daným oddělením. Supervize může posloužit i v rámci skupiny, jiný uživatel může převzít řešení ticketu například v době dovolené či nemoci jiného uživatele.
- **Řízení přístupu k modulům** Jednotlivé funkční moduly budou moci být administrátorem povoleny pro klienta či uživatele.
- **Zabezpečený přístup** Přístup do systému bude zabezpečen přihlašovacími údaji.
- **Anonymní přístup** Koncový zákazník bude mít přístup k určitým částem některých modulů bez přihlášení. Například k vytvoření ticketu s problémem.

Uživatelská sekce

- **Správa ticketů** Uživatelé budou moci vytvářet a zpracovávat tickety s dotazy či problémy.
- **Zpracování ticketů** V rámci ticketu bude možné měnit prioritu, typ problému, současný stav ticketu, evidovat interní poznámky a reagovat na tvůrce či zpracovatele ticketu. Obsaženo bude datum vytvoření a datum předpokládaného řešení. Nahrát také bude možné související dokumenty. Vzájemně půjdou problémy propojit a označit jako sledované.
- **Řešitelé ticketů** K ticketům bude možné přidělit řešitele problému.
- **Skupiny ticketů** Tickety bude možné zařadit do skupin, ty dle typu modulu mohou zastupovat například projekt či program, kterého se ticket týká.
- **Historie interakce s tickety** Části ticketu, pro které je klíčové, aby zachovávaly historii, budou mít zobrazitelnou historii. Jedná se například o historii stavů ticketu a změny řešitelů.

- **Záznam stráveného času** U jednotlivých ticketů bude možné zaznamenat čas strávený řešením. Ten může být užitečný například při následné fakturaci a podobně.

- **Dashboard** Základní obrazovka obsahující souhrny pro rychlou orientaci v aktuální situaci. V rámci dashboardu bude možné zobrazit komponenty, které si uživatel zvolí. Mezi základní komponenty patří například přehled aktuálně nevyřešených problémů, uživateli přiřazené tickety, uživatelem sledované tickety, problémy řešené vybranou skupinou uživatelů, nové, nepřiřazené tickety.

- **Kontakty** Zákazníky, kteří do systému zasahují, ať už zvenčí, například pomocí e-mailu, či skrze anonymní přístup, bude možné uložit do systému. K nim bude možné doplnit i doplňkové informace typu adresa, zaměstnavatel a podobně. Zároveň bude možné skrze kontakt dohledat jeho interakce se systémem, například historii ticketů, ve kterých figuroval.

1.3.2 Nefunkční požadavky

- **Modularita systému** Systém bude snadno rozšiřitelný po pozdější přidání modulů s novou funkcionalitou. Ta může rozšířit použitelnost systému pro další skupinu potenciálních uživatelů.

- **Uživatelské rozhraní** Rozhraní bude konzistentní a navržené s ohledem na modulární strukturu systému.

- **Podpora zařízení** Webová aplikace musí být plně funkční na desktopových a mobilních prohlížečích s podílem na trhu vyšším pěti procent na platformách s neukončenou podporou.

- **Podpora více jazyků** Webová aplikace bude připravena pro překlad do dalších jazykových mutací.

- **Zpracování e-mailů** Systém umožní modulům periodicky zpracovávat příchozí e-maily (například k vytváření ticketů) a pro snadné reakce či upozornění také odesílat e-maily dle nastavené e-mailové konfigurace.

1.3.3 Konkrétní moduly

Reklamace a dotazy zákazníků

Problémy a dotazy zákazníků budou přijímány skrze pravidelný příjem e-mailů, případně zadány ručně uživateli. Z nich bude vytvořen ticket, se kterým bude možné dále nakládat. Bude možné přiřadit ho uživateli k řešení a provést další standardní operace s ticketem.

Svoz reklamací

Zákazník na portálu zadá požadavek na svoz a vloží přílohu s dokumenty potřebnými pro svoz zakázky. Následně zaměstnanec mimo systém založí zakázku přepravy a dokumenty zašle na depo, které ho bude realizovat. Po svozu zboží na sklad klienta o tom zaměstnanec informuje na portálu. Klient následně potvrdí příjem skrze uzavření ticketu na portálu. Tento modul může být do budoucna vylepšen o integrace s API přepravce, integrací s interním systémem pro zadání svozu a podobně.

1.3.4 Případy užití

Role uživatelů

- Klient – Firma využívající portál.
- Uživatel – Zaměstnanec klienta, vyřizuje požadavky zákazníků.
- Administrátor systému – Administrátor spravující klienty a uživatele v systému.
- Administrátor klienta – Administrátor spravující nastavení a uživatele daného klienta.
- Zákazník – Zákazník klienta, který kupuje/využívá jeho produkty a má k nim dotaz/problém. Může a nemusí využívat systém.

Scénáře případů užití

Tato sekce popisuje scénáře případů užití vyplývající z funkčních požadavků.

- **Přihlášení** Umožňuje uživatelům a administrátorům přihlásit se do systému.
  1. Uživatel přijde na webový portál
  2. Pokud nejsou zapamatovány uživatelské údaje, zobrazí se přihlašovací obrazovka
  3. Uživatel vyplní své přihlašovací údaje
  4. Portál zobrazí uživatelův dashboard

Následující scénáře předpokládají přihlášeného uživatele.

Administrace [1]

- **Založení uživatele** Přidává uživatele do systému.
  1. Administrátor vybere přihlašovací jméno, heslo uživatele a klienta, pod kterého spadá, případně další volitelné údaje (skupinu uživatelů, povolené moduly, doplňková oprávnění).
  2. Systém zkontroluje zadané údaje
  3. Administrátor uloží korektně zadané údaje
- **Procházení uživatelů** Zobrazuje uživatele v systému
  1. Administrátor vybere sekci Uživatelé
  2. Systém zobrazí uživatele v systému
- **Editace uživatele**
  1. Administrátor provede případ užití *Procházení uživatelů*
  2. Administrátor vybere možnost Editace uživatele
  3. Administrátor upraví údaje uživatele
  4. Systém ověří zadané údaje
  5. Administrátor uloží korektně zadané údaje
- **Přidání do skupiny**
  1. Administrátor provede případ užití *Editace uživatele*
  2. Administrátor přidá uživatele do klientské skupiny
- **Přidělení role**
  1. Administrátor provede případ užití *Editace uživatele*
  2. Administrátor přidělí uživateli roli udávající jeho práva
- **Povolení/zákaz modulu uživateli**
  1. Administrátor provede případ užití *Editace uživatele*
  2. Administrátor provede zvolí moduly, které má mít daný uživatel povolené
- **Blokace uživatele** Uživatel se nebude moci přihlásit do systému, například při ukončení pracovního poměru
  1. Administrátor provede případ užití *Editace uživatele*

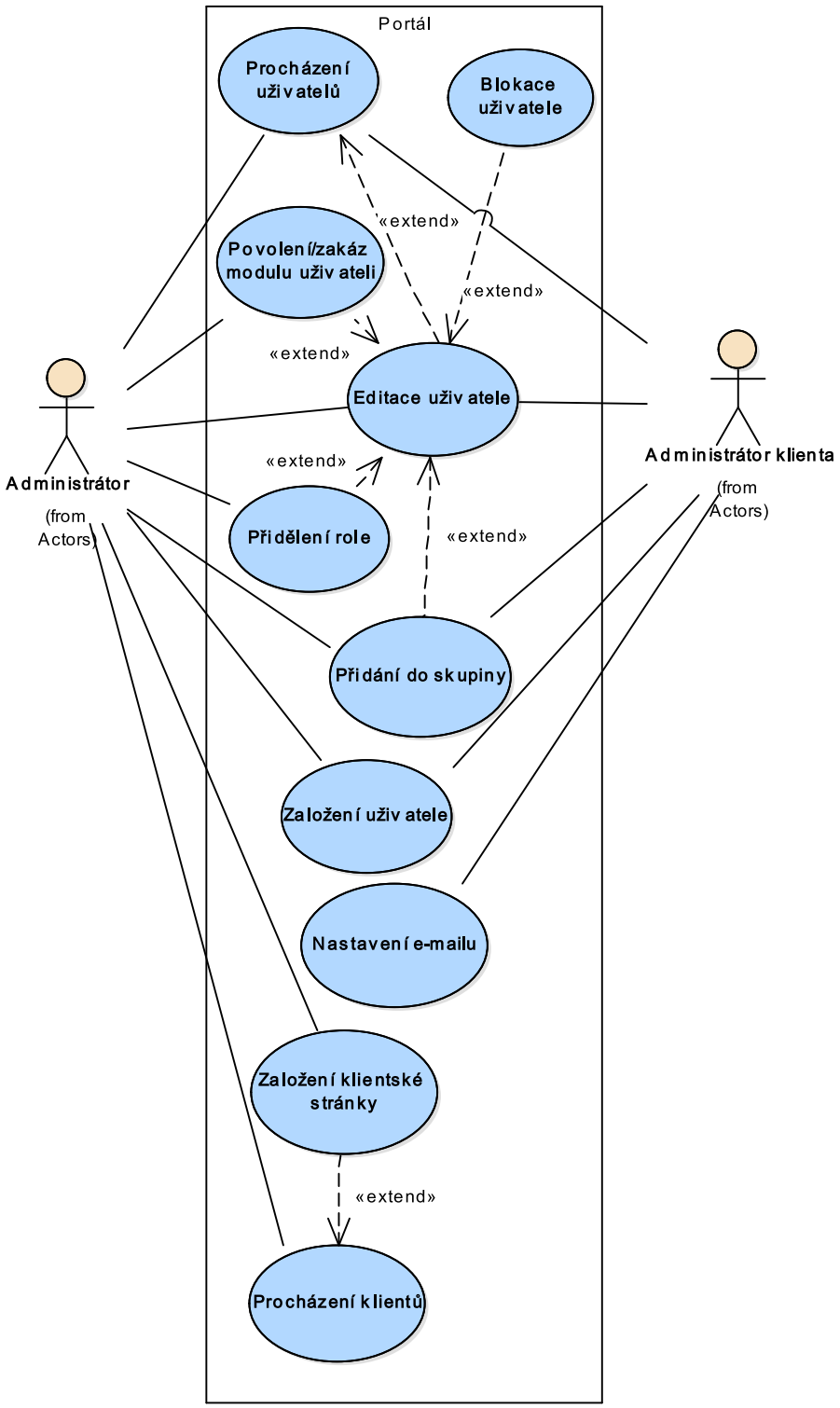
[1] Tato sekce předpokládá uživatele oprávněného k dané operaci, administrátora či administrátora klienta, viz diagram.

2. Administrátor vybere možnost blokace uživatele a potvrdí blokaci

- **Procházení klientů**
  1. Administrátor vybere sekci Klienti
  2. Systém zobrazí klienty v systému
- **Založení klientské stránky**
  1. Administrátor provede případ užití *Procházení klientů*
  2. Administrátor vybere možnost Založit nového klienta
  3. Administrátor zadá potřebné údaje
  4. Systém ověří korektnost vložených údajů
  5. Administrátor uloží korektně zadané údaje
- **Nastavení e-mailu**
  1. Administáor klienta přejde do sekce Nastavení klienta
  2. Administrátor klienta zvolí možnost Nastavit e-mail
  3. Administrátor klienta vybere typ e-mailu (odchozí/příchozí)
  4. Administrátor klienta vyplní údaje o připojení
  5. Administrátor klienta uloží zadané údaje

Tickety

- **Správa ticketu**
  1. Uživatel změní kýžené údaje
  2. Po provedení úprav uživatel uloží zadané údaje
- **Vytvoření ticketu**
  1. Uživatel vybere možnost vytvořit nový ticket
  2. Uživatel provede případ užití *Správa ticketu*
- **Přiřazení do skupiny**
  1. Uživatel provede případ užití *Správa ticketu*
  2. Uživatel vybere skupinu, do které ticket patří
- **Sledování ticketu**
  1. Uživatel provede případ užití *Správa ticketu*
  2. Uživatel vybere možnost sledovat ticket



Obrázek 1.6: Případy užití – administrace

Tabulka 1.2: Pokrytí funkčních požadavků – administrace

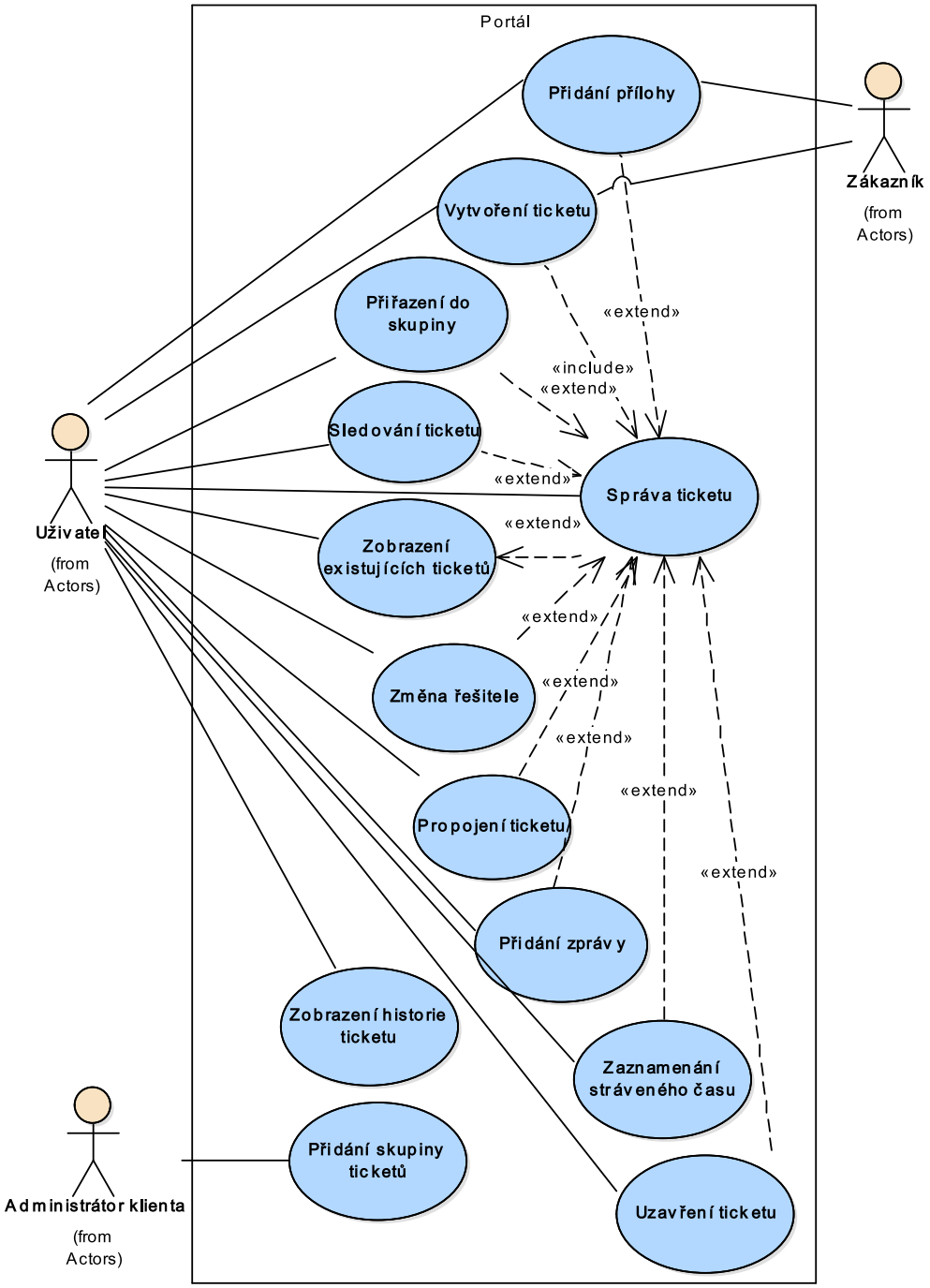
Případy užití \ Funkční požadavky	Podpora uživatelských účtů	Uživatelské role	Podpora více klientů	Skupiny uživatelů	Řízení přístupu k modulům	Zabezpečený přístup
Založení uživatele	↑					
Blokace uživatele	↑					
Procházení uživatelů	↑					
Editace uživatele	↑	↑				
Přidělení role	↑	↑				
Přihlášení	↑					↑
Procházení klientů			↑			
Založení klientské stránky			↑			
Nastavení e-mailu			↑			
Přidání do skupiny				↑		
Povolení/zákaz modulu uživateli					↑	

- **Změna řešitele**
  1. Uživatel provede případ užití *Správa ticketu*
  2. Uživatel vybere ze seznamu uživatelů daného klienta uživatele, kterému bude ticket přiřazen
- **Propojení ticketu**
  1. Uživatel provede případ užití *Správa ticketu*
  2. Uživatel vyhledá v seznamu existujících ticketů ten, který s aktuálním souvisí
- **Přidání zprávy**
  1. Uživatel provede případ užití *Správa ticketu*
  2. Uživatel vybere druh zprávy, veřejnou, či interní
  3. Uživatel zadá text zprávy
  4. Uživatel uloží zadanou zprávu
  5. Pokud byla vybrána veřejná zpráva a ticket byl zadán zákazníkem, systém mu ji na vyplněný e-mail odešle

- **Zaznamenání stráveného času**
  1. Uživatel provede případ užití *Správa ticketu*
  2. Uživatel zaznamená strávený čas při řešení ticketu
- **Přidání přílohy**
  1. Uživatel provede případ užití *Správa ticketu*
  2. Uživatel vybere soubor, který chce přiložit k ticketu
  3. Uživatel vybere možnost nahrát soubor
- **Uzavření ticketu**
  1. Uživatel provede případ užití *Správa ticketu*
  2. Uživatel vybere možnost Uzavřít ticket
  3. Uživatel zvolí důvod uzavření ticketu (např. chyba se neprojevila, případ vyřešen atd.)
  4. Uživatel potvrdí uzavření ticketu
- **Zobrazení existujících ticketů**
  1. Uživatel vybere možnost Zobrazit tickety
  2. Systém zobrazí existující tickety
  3. V případě zájmu uživatel vybere možnost filtrování ticketů
  4. Systém zobrazí tickety odpovídající daným filtrům
- **Zobrazení historie ticketu**
  1. Uživatel vybere možnost Zobrazit historii ticketu
  2. Systém zobrazí provedené aktivity s ticketem
- **Přidání skupiny ticketů**
  1. Administrátor klienta přejde do sekce Nastavení klienta
  2. Administrátor klienta vybere možnost Přidat skupinu ticketů
  3. Administrátor klienta vyplní název, případně popis skupiny ticketů
  4. Administrátor klienta uloží skupinu ticketů

Doplňkové případy užití

- **Zobrazení dashboardu**



Obrázek 1.7: Případy užití – práce s tickety

Tabulka 1.3: Pokrytí funkčních požadavků – práce s tickety

Případy užití \ Funkční požadavky	Správa ticketů	Zpracování ticketů	Řešitelé ticketů	Skupiny ticketů	Historie interakce s tickety	Záznam stráveného času
Zobrazení existujících ticketů	↑					
Vytvoření ticketu	↑					
Správa ticketu	↑	↑				
Sledování ticketu		↑				
Propojení ticketu		↑				
Přidání zprávy		↑				
Přidání přílohy		↑				
Uzavření ticketu		↑				
Změna řešitele			↑			
Přidání skupiny ticketů				↑		
Přiřazení do skupiny				↑		
Zobrazení historie ticketu					↑	
Zaznamenání stráveného času						↑

1. Po přihlášení systém uživateli zobrazí dashboard, který dle zvoleného modulu zobrazí například přiřazené tickety, sledované tickety, či počet nevyřešených ticketů

- **Zobrazení kontaktů** [2]

  1. Uživatel vybere možnost Kontakty
  2. Systém zobrazí kontakty daného klienta
  3. V případě zájmu uživatel vybere filtrovací kritéria
  4. Systém zobrazí kontakty splňující daná kritéria

- **Přidání kontaktu**

  1. Uživatel provede případ užití *Zobrazení kontaktů*
  2. Uživatel vybere možnost Přidat nový kontakt

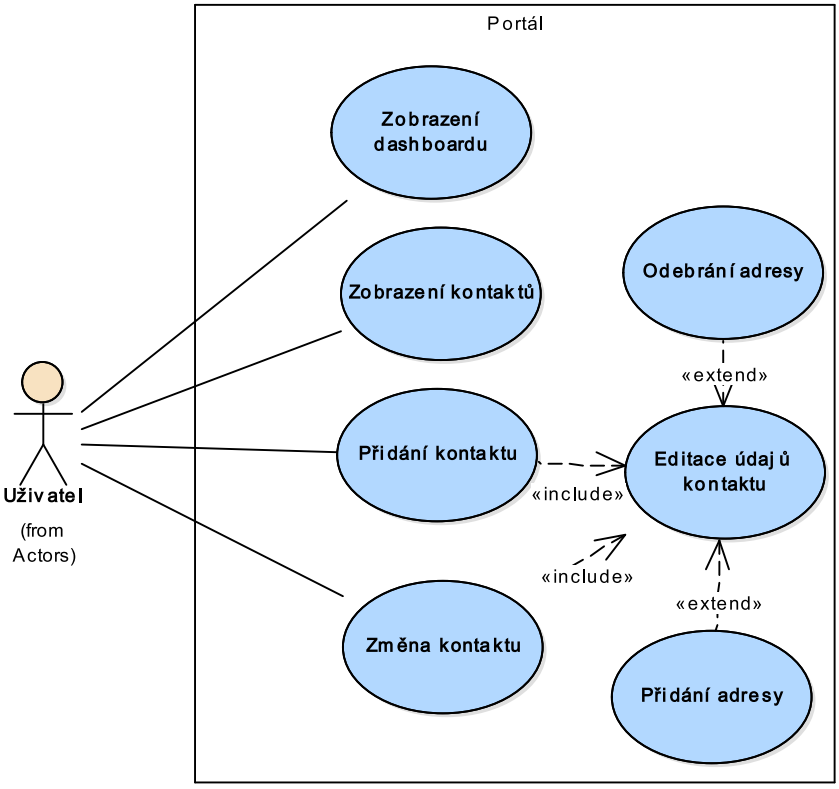
[2] Nejsou rozlišovány uživatelské a firemní kontakty, které si jsou z pohledu případů užití velmi podobné.

Tabulka 1.4: Pokrytí funkčních požadavků – uživatelská sekce

Případy užití \ Funkční požadavky	Dashboard	Kontakty
Zobrazení dashboardu	↑	
Zobrazení kontaktů		↑
Přidání kontaktu		↑
Změna kontaktu		↑
Editace údajů kontaktu		↑
Přidání adresy		↑
Odebrání adresy		↑

3. Je proveden případ užití *Editace údajů o kontaktu*

- **Změna kontaktu**
  1. Uživatel provede případ užití *Zobrazení kontaktů*
  2. Uživatel vybere existující kontakt
  3. Je proveden případ užití *Editace údajů o kontaktu*
- **Editace údajů kontaktu**
  1. Uživatel vyplní údaje o kontaktu
  2. Uživatel vybere uložení
  3. Systém ověří vložené údaje
  4. Systém uloží daný kontakt
- **Přidání adresy**
  1. Uživatel provede případ užití *Editace údajů o kontaktu*
  2. Uživatel zvolí možnost přidání adresy
  3. Uživatel zadá potřebné údaje k vytvoření adresy
- **Odebrání adresy**
  1. Uživatel provede případ užití *Editace údajů o kontaktu*
  2. Uživatel zvolí možnost odebrání adresy
  3. Uživatel potvrdí záměr smazat adresu



Obrázek 1.8: Případy užití – uživatelská sekce

Kapitola 2

Návrh

2.1 Doménový model

Po zmapování klíčových entit a vztahů mezi nimi byl vytvořen doménový model, zobrazen je na diagramech 2.1 a 2.2.

2.1.1 Tickety

- **IssueTicket** Základní ticket, kolem kterého se slučují aktivity spojené s tickety. Je navázán na partnerskou společnost, u které došlo k problému, na adresu, kde k danému problému došlo, kontaktní osobu, která vytvořila daný ticket, klienta systému, pod kterého spadá, modul aplikace, ve kterém byl vytvořen, tickety, se kterými je spojený a na skupinu ticketů, do které spadá.
  - **name** název ticketu
  - **deliveryNoteNr** číslo dodacího listu pro spárování ticketu s transakcí v externím systému
  - **clientId** id pro spárování problému s klientským systémem
  - **updatedTime** poslední změna ticketu
- **UserTicketActivity** Základ pro uživatelskou aktivitu týkající se konkrétního ticketu, zachovává historii akcí. Konkrétní aktivity, které zachovávají historii, dědí od tohoto objektu.
  - **executedTime** čas vytvoření aktivity
- **CommentActivity** Veřejný (zobrazí/odešle se uživateli) či soukromý komentář k ticketu
  - **content** obsah komentáře

  - **isInternal** zda má být komentář označen jako veřejný nebo soukromý

- **TimeSpentActivity** Záznam stráveného času při zpracovávání ticketu.
  - **timeSpent** strávený čas
- **AttachmentActivity** Připojení přílohy k ticketu.
- **StateActivity** Změna stavu ticketu.
- **PriorityActivity** Změna priority řešení ticketu.
- **DueDateActivity** Změna předpokládaného času vyřešení ticketu.
  - **dueDate** předpokládaný čas řešení
- **TicketTypeActivity** Změna typu ticketu.
- **AssignmentActivity** Změna řešitele ticketu.
- **FollowTicketActivity** Označení ticketu pro jeho sledování.
- **TicketGroup** Představuje klientem definovanou skupinu ticketů.
  - **name** název skupiny
  - **description** popis skupiny
- **TicketType** Typ ticketu, například úkol, problém či spam.
  - **codename** kódové označení typu ticketu
- **TicketState** Stav ticketu, typickými stavy jsou například otevřený, řešený, uzavřený. Spadá do modulu aplikace.
  - **codename** kódové označení stavu ticketu
- **Priority** Priorita řešení ticketu.
  - **codename** kódové označení priority
- **PersistedFile** Soubor uložený na disku, typicky příloha.
  - **filePath** cesta k souboru
  - **creationTime** čas vytvoření souboru
- **Product** Produkt, kterého se problém týká.
  - **name** název produktu
  - **description** popis produktu

  - **clientId** id produktu v klientském systému

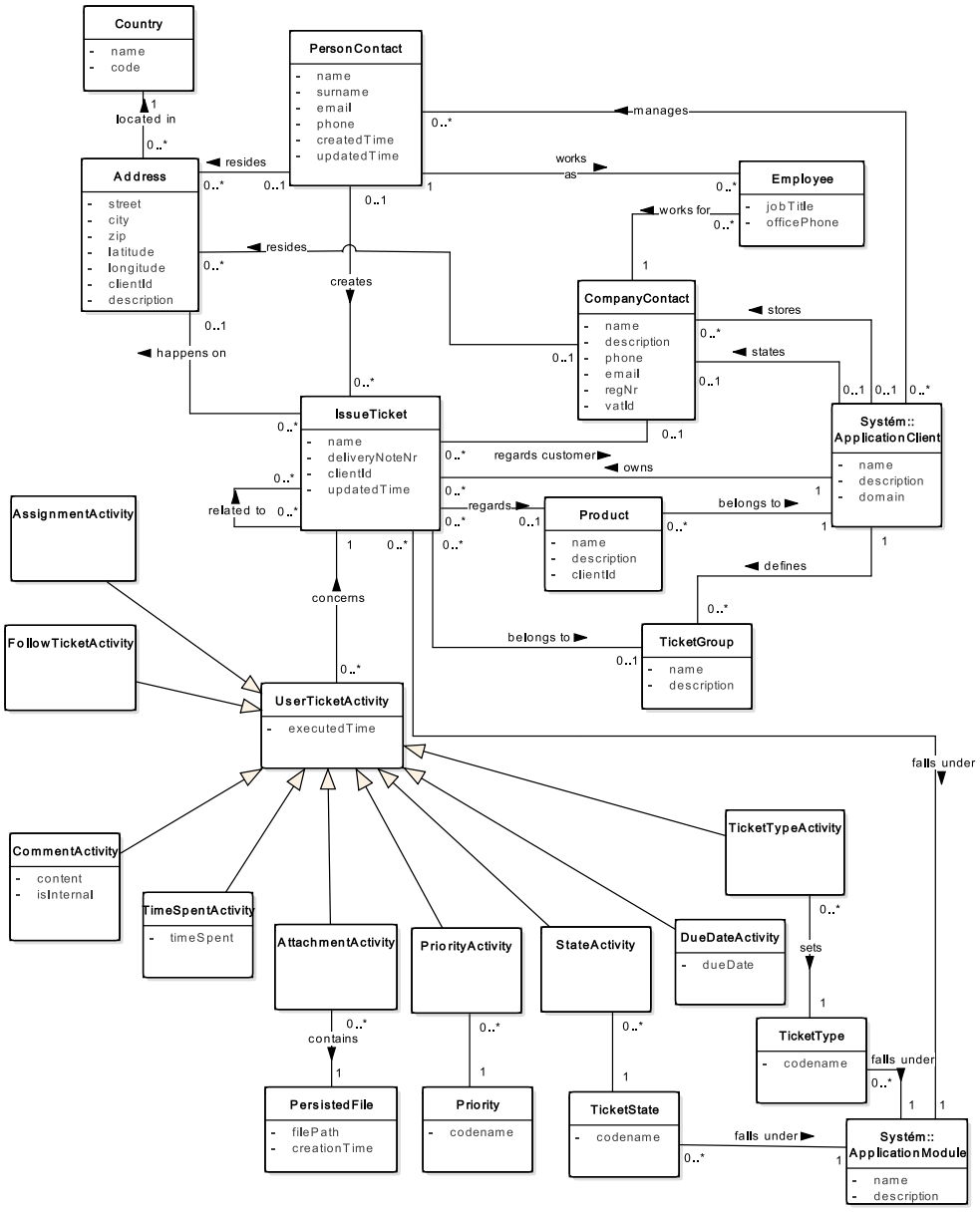
- **PersonContact** Podrobnosti o uživateli.
  - **name** jméno
  - **surname** příjmení
  - **email** e-mailová adresa
  - **phone** telefonní číslo
  - **createdTime** čas vytvoření
  - **updateTime** čas poslední změny

- **Employee** Informace o zaměstnanci, vázáno na kontaktní osobu a firmu, ve které pracuje.
  - **jobTitle** pracovní pozice
  - **officePhone** telefon do kanceláře

- **CompanyContact** Informace o firmě. Je vázán na zaměstnance, kteří ve firmě pracují, adresy, na kterých se firma nachází, klientskou firmu, která je danou firmou a klientskou firmu, která si zaznamenala kontaktní údaje této firmy.
  - **name** název firmy
  - **description** popis firmy
  - **phone** telefonní kontakt
  - **email** e-mailový kontakt
  - **regNr** IČO
  - **vatId** DIČ

- **Address** Adresa, může reprezentovat adresu firmy či konkrétní osoby.
  - **street** ulice a číslo domu
  - **city** město
  - **zip** poštovní směrovací číslo
  - **latitude** zeměpisná šířka
  - **longitude** zeměpisná délka
  - **clientId** klientské id adresy, například id skladu
  - **description** popis adresy

- **Country** Stát.
  - **name** název státu
  - **code** dvoupísmenný kód státu ve formátu ISO 3166-1 alpha-2



Obrázek 2.1: Doménový model – tickety

2.1.2 Systém

- **UserCredentials** Uživatelský účet se základními údaji.
  - **username** uživatelské jméno
  - **password** hešované heslo
  - **createdTime** čas vytvoření uživatele
  - **updatedTime** čas poslední změny uživatele
  - **enabled** povolení užívání účtu
- **UserRole** Role uživatele, například administrátor, běžný uživatel atp. Vytváří hierarchii rolí pomocí vazby na rodičovskou roli.
  - **name** název role
  - **description** popis role
- **Locale** Jazyk.
  - **name** celý název jazyka
  - **code** kód jazyka ve formátu ISO 639-1
- **EmailTemplate** Šablona pro vytvoření e-mailu. Je vázána na modul aplikace, ve kterém se má použít.
  - **name** název šablony
  - **content** obsah šablony
- **ApplicationClient** Klient aplikace, jedná se typicky o organizaci/-firmu, která systém využívá.
  - **name** jméno klienta
  - **description** popis klientské entity
  - **domain** doména, kterou může používat pro přístup k webové aplikaci
- **UserGroup** Skupina uživatelů, usnadňuje rozčlenění uživatelů v rámci organizace, také slouží k případnému převzetí kontroly nad ticketem v rámci skupiny při nečinnosti uživatele.
  - **name** název skupiny
  - **description** popis skupiny
- **ApplicationModule** Modul aplikace, například reklamace zákazníků, svozy reklamací či hlášení problémů s IT systémy.
  - **name** název modulu

  - **description** popis modulu

- **ModuleSetting** Upřesňující nastavení modulu uživatelem.
  - **codename** kódové jméno nastavení
  - **value** hodnota nastavení

- **UserSetting** Uživatelská nastavení.
  - **codename** kódové jméno nastavení
  - **value** hodnota nastavení

2.2 Výběr technologií

V této sekci jsou shrnuty vybrané technologie mající vliv na návrh systému.

2.2.1 Server

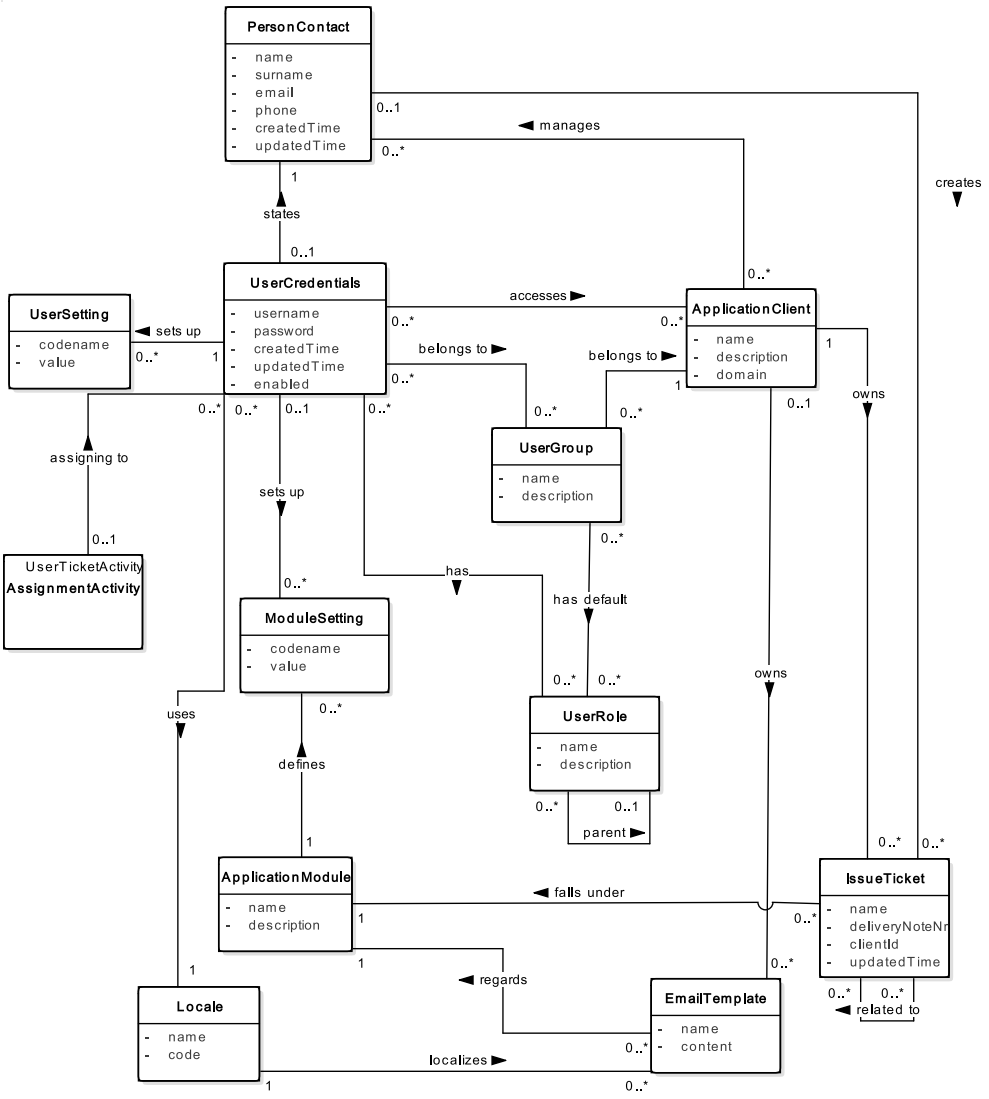
REST API

Záměrem je vytvořit aplikaci stylem single-page application (SPA), tzn. s klientskou aplikací, která bude mít předem načtenou strukturu webu, nebude rušit přechody ze stránky na stránku. Tím zvýší uživatelský komfort a v rámci prohlížeče bude simulovat chování samostatné aplikace. S tímto záměrem přichází nutnost přenosu uživatelem vyžádaných dat a samotnou aplikací. Standardně užívanou praxí je mezi databázový server a klientskou aplikaci vložit aplikační server, který zpracovává uživatelské požadavky a na jejich základě vrací data. Tento aplikační server bude sloužit jako webová služba, která bude požadavky zpracovávat. Tato služba se nazývá API.

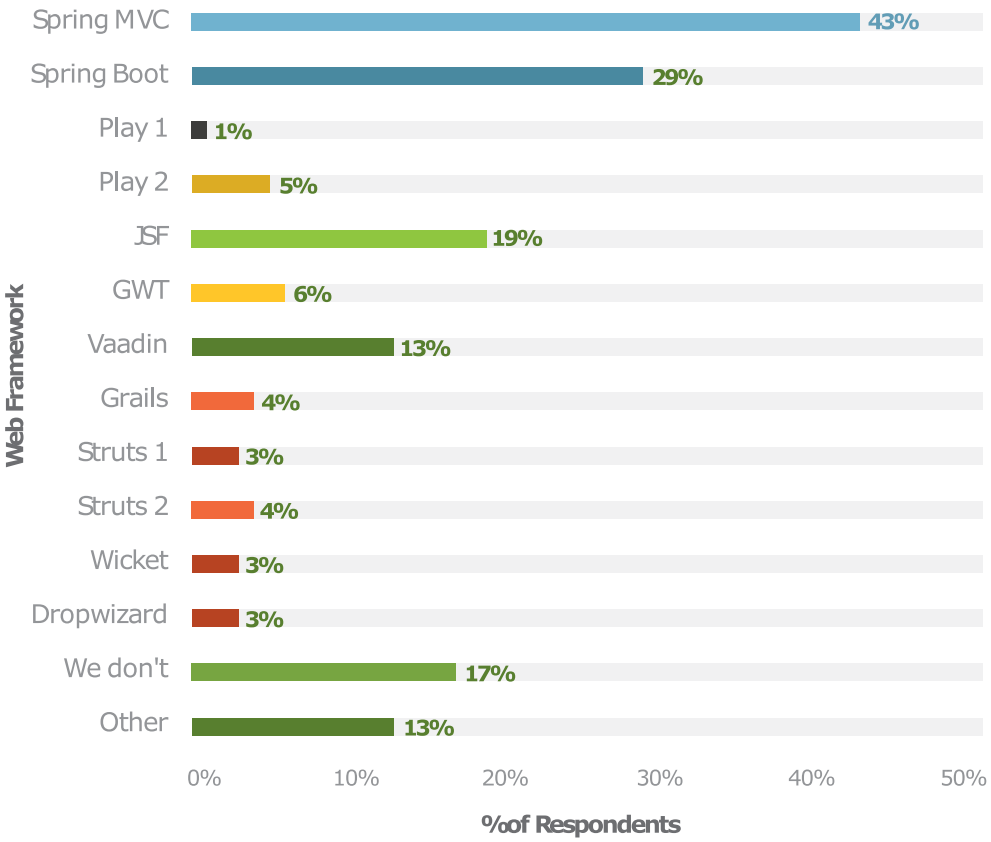
Z hlediska typu rozhraní je možné zvážit nejčastěji používané REST (Representational State Transfer) a SOAP (Simple Object Access Protocol). Lze nalézt mnoho zdrojů, ze kterých vyplývá, že REST rozhraní je pro tento typ aplikace vhodnější. Například dle [9] jsou hlavním důvodem pro zvolení SOAP různá rozšíření, ta však souvisí s bezpečností, atomicitou transakcí a podobně, pro náš systém nejsou rozhodující. Všechny důležité vlastnosti totiž obsahuje i méně náročné (jak na systémové prostředky serveru i klienta, tak na celkovou architekturu systému) RESTful rozhraní. Toto rozhraní poskytuje větší volnost při výběru reprezentace dat i přenosového protokolu. Pro účely tohoto systému bude využito rozhraní REST přes HTTP.

JSON reprezentace

Rozhraní typu REST není vázané na konkrétní formát reprezentace dat. Nejpoužívanějšími formáty jsou JSON a XML. JSON formát je pro využití v pro-



Obrázek 2.2: Doménový model – systém



Obrázek 2.3: Využití Java webových frameworků [12]

jektu tohoto typu výhodnější, jelikož je obvykle datově úspornější, má jednodušší strukturu, z toho důvodu je snáze (rychleji) parsovatelný. JSON nahlíží na data z pohledu kolekce párů jmen a hodnot a seřazených seznamů. V případě vybraných technologií se na data pohlíží jako na objekty, jejich atributy a pole objektů/hodnot. [10]

Spring Framework

Aplikační server bude vytvořen s pomocí Spring Frameworku, respektive pomocí podpůrných knihoven z platformy Spring IO [11]. Důvody pro výběr této technologie jsou současné produkční prostředí, již spolupracující s daným frameworkem, osobní zkušenost, ale zejména vyspělost a celistvost celé platformy Spring. Navíc má dle průzkumu RebelLabs Spring Framework největší zastoupení na trhu Java frameworků [12], náhled na grafu 2.3. Nejpoužívanějším jazykem pro implementaci aplikací s daným frameworkem je programovací jazyk Java, proto bude použit při vývoji aplikačního serveru.

Spring Boot [13] je zastřešujícím prvkem usnadňujícím nasazení a nasta-

vení celé aplikace. Umožněn je samostatný provoz aplikace pomocí vestavěného webového/servlet serveru Tomcat, Jetty či Undertow, což je užitečné zejména při vývoji či testování. Také nabízí automatickou konfiguraci obvykle používaných funkcí, případně snadné nastavení těchto funkcí skrze konfigurační soubor. Pro konfigurace je možné využít formát souboru properties či YAML. Zároveň je ovšem možné stále využívat standardní konfiguraci pomocí XML souborů či Java notací. Webovým serverem byl zvolen Apache Tomcat [14], jak z důvodu cílového produkčního prostředí, tak z důvodu popularity. Dle dat z W3Techs [15] je v současné době nejpoužívanějším webovým/servlet serverem na trhu.

Gradle

Gradle je nástrojem pro kompilaci softwaru, automatizaci úkolů s ní spojených a řízení závislostí [16]. V projektu bude použit zejména k řízení závislostí, tedy ke stažení knihoven. Nejpoužívanějšími nástroji k těmto účelům jsou právě Gradle a Maven (opět dle [12]). Maven je technologií starší, více používanou, ovšem dle [17] s méně funkcemi a horšími vlastnostmi. Navíc Gradle pro soubory s nastavením využívá celé skripty založené na programovacím jazyku Groovy, Maven používá konfiguraci v XML. Repozitáře knihoven jsou shodné, použitím Gradle neztrácíme knihovny stažitelné přes Maven.

Hibernate ORM

Přístup k databázi bude realizován skrze objektově relační mapování (ORM). Umožňuje přístup k datům skrze objekty, které jsou hlavním stavebním prvkem vybraného jazyka Java. Podporovány jsou i další vlastnosti objektově orientovaných jazyků typu dědění či využití kolekcí objektů. ORM nás také dokáže do jisté míry odstínit od konkrétního databázového stroje, v případě potřeby je tedy možné vyměnit databázový stroj, který aplikace využívá. V Javě EE (Enterprise Edition) je tento přístup standardizován pod názvem Java Persistence API (JPA). V rámci platformy Spring je navíc možné využít projektu Spring Data JPA [18], který nad rozhraním JPA umožňuje využít QueryDSL, snadno auditovat změny v tabulce, stránkovat, validovat dotazy atd. Z dostupných ORM podporujících JPA standard byla vybrána knihovna Hibernate [19].

OAuth 2

OAuth 2 je standardizovaným protokolem k autorizaci požadavků [20]. Zároveň nabízí uvolnění struktury oproti původnímu protokolu OAuth 1. Umožňuje snadnější splnění kryptografických požadavků skrze využití protokolu HTTPS, je lépe vystavěno pro současný druh webových služeb a zařízení, které je využívají, navíc se lépe škáluje. V tomto systému bude využit primárně typ povolení (grant type) heslem, jelikož je klientská aplikace vytvořena přímo vlastníkem

webové služby. Po ověření uživatelského jména a hesla klient obdrží kód, který bude doprovázet další požadavky na server, ten se nazývá bearer token [21]. Ke snadnější implementaci serveru bude využita knihovna Spring Security OAuth [22].

HTTPS

Komunikace mezi klientskou aplikací a aplikačním serverem bude v produkčním prostředí probíhat přes protokol HTTPS. Poskytuje dodatečné zabezpečení nad protokolem HTTP, který bude použit v testovacím prostředí. V dnešní době se typicky jedná o protokol HTTP zabezpečený přes TLS. Poskytuje ochranu před komunikací s nesprávným serverem, ale i ochranu dat na cestě k serveru. Ochrana dat v průběhu přenosu slouží k ochraně proti odposlouchávání citlivých údajů a hesel při cestě na serveru, ale zároveň proti útoku „Man-in-the-Middle“, který dokáže pozměnit požadavky na server či jeho odpovědi. [23]

PostgreSQL

V rámci požadavků na systém je požadován databázový server dostupný zdarma. Na základě tohoto parametru se nabízí výběr z nejpoužívanějších databázových strojů MySQL, MariaDB a PostgreSQL. Z pohledu produkčního prostředí je preferovanou volbou PostgreSQL. Proti preferované volbě nebyly nalezeny zásadní protiargumenty, jedná se o hojně využívaný, výkonný databázový systém v aktivním vývoji s více než dvacetiletou historií. Při vývoji systému bude využit databázový systém PostgreSQL [24].

2.2.2 Klient

Angular

Vzhledem k záměru vytvořit SPA je třeba zvolit technologii, která je k jejich vytváření určena. Používat k chování SPA pouze JavaScript v rámci běžných stránek renderovaných na serveru je náročně udržitelné a snadno vede k nepřehlednému kódu. Existují serverové technologie, které podobné chování dokáží zajistit, jmenovitě například Vaadin Framework. Tyto nástroje ovšem na druhou stranu připravují o JavaScriptové knihovny dostupné na frontendu, respektive nedovolují je snadno použít, případně vyžadují nemalé množství programového kódu k jejich zprovoznění. Z toho důvodu je výběr omezen na technologie fungující přímo na frontendu.

S ohledem na vlastní znalosti a popularitu byly zvažovány frameworky React [25], AngularJS [26] a Angular [27]. AngularJS je ve vývoji mnoho let, má stále rozsáhlou komunitu, nicméně je navržen odlišně než novější framework Angular, s důrazem na spojení HTML a JavaScriptu. Frameworky React a Angular jsou modernější a modulárnější. Budoucnost frameworku

AngularJS je nejistá, sice je stále vyvíjen, ale jedná se spíše o udržování, nikoliv o vývoj dalších funkcí. Kompletní přepis frameworku z AngularJS na Angular je varovným signálem, že s ním samotní tvůrci už nebyli spokojeni. Pro zvýšení výkonu a optimalizaci pro vyhledávače nabízí Angular i React renderování na serveru. Angular je z vybíraných frameworků nejnovější, dají se tedy čekat problémy s nedostatkem materiálů, menší komunitou, případně chybami ve frameworku. Původním názvem byl Angular 2, postupně přešel do jména Angular. Vývojáři ze společnosti Google tím chtějí naznačit dlouhou budoucnost s bezproblémovými přechody na další verze (na rozdíl od přechodu z AngularJS na Angular). Hlavními rozdíly jsou doporučené programovací/skriptovací jazyky. V rámci Angularu jazyk TypeScript [28] ke zdrojovému kódu a HTML s doplňkovými tagy k zápisu šablon. V Reactu JSX, vlastní způsob zápisu JavaScriptu, zejména kvůli usnadnění tvorby šablon. Framework Angular je přímo naprogramován v jazyce TypeScript. Ten je typovanou nadmnožinou jazyka JavaScript. K běhu v prohlížeči je třeba TypeScript kompilovat. To je výhodou i nevýhodou. Nevýhodou je samozřejmě složitější nastavení, výhodou je možnost použít i nové funkce dosud neimplementované ve všech prohlížečích. Navíc se v současné době, právě kvůli možnosti použít novou funkcionalitu, běžně kompiluje i přímo JavaScript. JSX se kompiluje také. Výhodou typovaného jazyka je lepší nápověda v rámci vývojových prostředí a také snazší refaktorování kódu. Angular lze použít nejen s jazykem TypeScript, ale také s jazyky JavaScript a Dart. JavaScript má ovšem výše zmíněné nevýhody. Dart sdílí výhody s TypeScriptem, ovšem naráží na nekompatibilitu s knihovnami v jazyce JavaScript. Konkrétně je třeba použít, vygenerovat či naprogramovat prostředníka se statickým typováním. Podpora prohlížečů, naznačená v tabulce 2.1, je pro zaměření portálu dostatečná. Angular je frameworkem obsahujícím více propojených knihoven, které tvoří balík doporučení vývoje aplikace. Framework React nabízí větší možnost volby v dalších knihovnách. Zejména z důvodu preference jazyka TypeScript a většího množství doporučení přímo v rámci frameworku byl k implementaci webového klienta zvolen framework Angular.

Tabulka 2.1: Angular framework – podpora prohlížečů [29]

Chrome	**Firefox**	**Edge**	**IE**	**Safari**	**iOS**	**Android**	**WP IE**
Aktuální	Aktuální	13+	9+	7+	7+	4.1+	11

2.3 Datový model

Datový model vychází z navrženého doménového modelu. V této sekci jsou shrnuty rozdíly a jejich odůvodnění.

K převodu dědičnosti do datového modelu existuje několik způsobů. Vzhledem k práci nad knihovnou Hibernate jsou dle [30] doporučené následující

způsoby reprezentace dědičnosti.

1. Tabulka na konkrétní třídu s implicitním polymorfismem – Tabulky zvlášť pro každou třídu, separátní id pro každou z nich. Z pohledu SQL jsou tabulky naprosto odděleny. Problém s polymorfním dotazem, je třeba provést více dotazů, pro každou tabulku zvlášť. Společné prvky v abstraktní třídě anotované `@MappedSuperclass`, podtřídy od ní dědí.

2. Tabulka na konkrétní třídu se sjednocením – Podobné předchozímu případu. Rozdíl je v mapování, kde je možné do abstraktní třídy přidat i vlastní id. Abstraktní třídě doplníme anotaci `@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)`. Tento typ řeší polymorfní dotazy pomocí užití spojení (klíčové slovo `UNION` v SQL). V rámci databáze se však stále jedná o zcela oddělené tabulky.

3. Tabulka na hierarchii – Z celé hierarchie tříd vytvoříme jedinou tabulku. Konkrétní typ je identifikován pomocí pomocného sloupce s typem, obvykle nazývaným diskriminátor. V případě již existujících schémat s nemožností změny je také možné využít dotaz pro určení typu sloupce. Výhodou této strategie je vysoký výkon a jednoduchost. V rámci Java kódu je stále zachována přehlednost pomocí separátních tříd pro každý podtyp. Nevýhodou schématu je ztráta datové integrity a nesplnění třetí normální formy v důsledku denormalizace. Nemůžeme použít například `NOT NULL` a další omezení týkající se podtypů.

4. Tabulka na podtřídu – Rodičovská tabulka obsahuje společné atributy a společné id. Toto id používají podtřídy, v tomto případě samostatné tabulky, jako primární a zároveň cizí klíč. V tomto způsobu je dědičnost v rámci databázového stroje reprezentována nejčistším způsobem. V Java kódu dostáváme v podstatě totožné třídy jako v předchozích bodech. Dotazy jsou však méně výkonné, spojení je vyžadováno pro všechny čtení celých podtříd. Pokud je třeba přečíst více typů najednou, musí být spojeno mnoho tabulek.

Dědičnost aktivit uživatelů (`UserTicketActivity` v doménovém modelu) je v databázovém modelu reprezentována způsobem tabulka na hierarchii. Tato strategie je využita z důvodu malého počtu, navíc vzájemně se vylučujících, atributů doménových objektů dědících z `UserTicketActivity`. Výsledná tabulka `t_user_ticket_activity` tudíž nebude zásadně větší než při jejím rozdělení na několik tabulek. Navíc použití jedné tabulky zbavuje nutnosti častého a navíc výpočetně náročného spojování mnoha tabulek. V samotné implementaci API bude skrze ORM tato tabulka namapována na několik objektů a přehlednost řešení pomocí dědičnosti zůstane zachována. K odlišení typu aktivity je využito jednoznakové označení `activity_type`.

Primární klíče jsou implementovány pomocí datového typu `serial`, který umožňuje automatickou inkrementaci číselného pole. Vztahy mezi entitami

jsou reprezentovány spojením přes primární klíče tabulek a zároveň omezeny cizími klíči pro zajištění konzistence databáze. Výjimkami jsou tabulky `t_locale` a `t_country`, které používají jako primární klíč kód jazyka, respektive kód země. Využíván je standard ISO 639-1 pro kód jazyka a ISO 3166-1 alpha-2 pro kód země. Tyto standardy definují dvoupísmenné kódy, díky své unikátnosti a malé velikosti jsou tedy ideálními kandidáty pro primární klíč. Navíc je vhodné je použít i pro cizí klíč, jelikož uživatelům jsou tyto kódy typicky známé a není třeba používat spojení pro zobrazení celého názvu.

Výsledný diagram je k náhlednutí na diagramu C.

2.4 Komunikační rozhraní

2.4.1 Zdroje

Tato část obsahuje návrh zdrojů ve formě tabulek obsahujících sloupce URI (Uniform Resource Identifier – identifikátory zdrojů [31]), HTTP metodu a popis zdroje.

Tickety

URI následujících zdrojů začínají řetězcem `/clients/{id klienta}/modules/{id modulu}`.

`/tickets`	**GET**	Získání ticketů daného klienta v daném modulu
`/tickets`	**POST**	Vytvoření nového ticketu

URI následujících zdrojů začínají řetězcem `/clients/{id klienta}/modules/{id modulu}/tickets`.

`/{id}`	**GET**	Získání ticketu dle id
`/{id}`	**PUT**	Aktualizace ticketu s daným id
`/{id}/comments`	**POST**	Přidání komentáře k danému ticketu
`/{id}/attachments`	**POST**	Přidání přílohy k danému ticketu
`/{id}/attachments/{id}`	**GET**	Stažení dané přílohy daného ticketu
`/{id}/time`	**POST**	Přidání stráveného času k danému ticketu
`/{id}/activities`	**GET**	Získání aktivit u daného ticketu
`/{id}/activities`	**POST**	Přidání aktivity k danému ticketu
`/{id}/follow`	**GET**	Zjištění, zda je ticket sledován
`/{id}/follow`	**PUT**	Sledování ticketu

`/{id}/follow`	**DELETE**	Ukončení sledování ticketu
`/types`	**GET**	Získání typů ticketů v daném modulu
`/states`	**GET**	Získání stavů ticketů v daném modulu
`/priorities`	**GET**	Získání priorit ticketů v daném modulu
`/groups`	**GET**	Získání skupin dle klienta
`/groups`	**POST**	Vytvoření skupiny
`/groups/{id}`	**GET**	Získání skupiny dle id
`/groups/{id}`	**PUT**	Editace skupiny s daným id

URI následujících zdrojů začínají řetězcem `/clients/{id klienta}`.

`/products`	**GET**	Získání klientských produktů
`/products`	**POST**	Vytvoření nového produktu
`/products/{id}`	**GET**	Získání produktu dle id
`/products/{id}`	**PUT**	Editace produktu s daným id
`/products/{id}`	**DELETE**	Označení produktu s daným identifikátorem za smazaný

Kontakty

URI následujících zdrojů začínají řetězcem `/clients/{id klienta}`.

`/people`	**GET**	Získání kontaktů osob
`/people`	**POST**	Vytvoření nové osoby
`/people/{id}`	**GET**	Získání osoby dle id
`/people/{id}`	**PUT**	Editace osoby s daným id
`/people/{id}`	**DELETE**	Označení osoby s daným id za smazanou

`/companies`	**GET**	Získání firemních kontaktů
`/companies`	**POST**	Vytvoření nové firmy
`/companies/{id}`	**GET**	Získání firmy dle id
`/companies/{id}`	**PUT**	Editace firmy s daným id
`/companies/{id}`	**DELETE**	Označení firmy s daným id za smazanou

`/locations`	**GET**	Získání lokací
`/locations`	**POST**	Vytvoření nové lokace
`/locations/{id}`	**GET**	Získání lokace dle id
`/locations/{id}`	**PUT**	Editace lokace s daným id

`/locations/countries`	**GET**	Získání dostupných zemí a jejich kódů

Uživatelé

URI následujících zdrojů existují v uvedené formě pro správce celého systému a ve variantě začínající řetězcem `/clients/{id klienta}`, pro účely správy místním administrátorem.

`/users`	**GET**	Získání uživatelů
`/users`	**POST**	Vytvoření nové uživatele
`/users/{id}`	**GET**	Získání uživatele dle id
`/users/{id}`	**PUT**	Editace uživatele s daným id
`/users/{id}`	**DELETE**	Blokace uživatele s daným id
`/users/{id}/enable`	**PUT**	Odblokování uživatele s daným id

Další uživatelské zdroje:

`/oauth/token`	**POST**	Přihlášení uživatele k rozhraní, vyžádání tokenu
`/oauth/token`	**POST**	Obnovení tokenu
`/user`	**GET**	Získání informací o přihlášeném uživateli
`/user/roles`	**GET**	Získání informací o všech dosažitelných rolích přihlášeného uživatele
`/user/{id}/roles/master`	**GET**	Zjištění, zda je uživatel administrátorem celého systému
`/user/{id}/roles/master`	**PUT**	Učinění uživatele administrátorem systému
`/user/{id}/roles/master`	**DELETE**	Odebrání práv pro administraci celého systému uživatelem

Systém

`/clients`	**GET**	Získání klientů
`/clients`	**POST**	Vytvoření nové klienta
`/clients/{id}`	**GET**	Získání klienta dle id
`/clients/{id}`	**PUT**	Editace klienta s daným id
`/modules`	**GET**	Získání modulů v systému

URI následujících zdrojů začínají řetězcem `/clients/{id}`.

`/templates`	**GET**	Získání šablon komentářů dle identifikátoru klienta
`/templates`	**POST**	Vytvoření šablony komentáře
`/templates/{id}`	**GET**	Získání šablony komentáře dle id
`/templates/{id}`	**PUT**	Změna šablony komentáře
`/templates/{id}`	**DELETE**	Odstranění šablony komentáře
`/modules`	**GET**	Získání povolených modulů klienta
`/modules/all`	**GET**	Získání všech existujících modulů klienta
`/modules/user/{id}`	**GET**	Získání povolených modulů uživatele
`/modules/{id}/settings`	**GET**	Získání klientských nastavení modulu
`/modules/{id}/settings`	**PUT**	Zapsání klientských nastavení modulu

URI následujících zdrojů začínají řetězcem `/clients/{id klienta}`/users.

`/groups`	**GET**	Získání skupin dle klienta
`/groups`	**POST**	Vytvoření skupiny
`/groups/{id}`	**GET**	Získání skupiny dle id
`/groups/{id}`	**PUT**	Editace skupiny s daným id

2.4.2 Reprezentace zdrojů

Reprezentace zdrojů je analogická pro všechny zdroje v sekci, proto je naznačena pouze pro zdroj companies, konkrétně 1 firmu z kolekce. Reprezentace nepoužívá tzv. obálku, místo toho využívá query parametry a HTTP hlavičky se stejnou výpovědní hodnotou (více v následujících sekcích).

Pole	**Datový typ**	**Popis**
id	Number	Id firmy
createdTime	String	Čas ve formátu ISO 8601 [32]
updatedTime	String	Čas ve formátu ISO 8601
name	String	Název firmy
description	String	Popis firmy
phone	String	Telefonní kontakt firmy
email	String	E-mailový kontakt firmy
vatId	String	DIČ
registrationNumber	String	IČO
residences	Pole objektů	Objekty typu Address

```
{
  "id": 2,
  "updatedTime": "2017-04-17T11:58:26Z",
  "createdTime": "2017-04-03T10:58:12Z",
  "name": "Company",
  "description": null,
  "phone": "+420 123456789",
  "email": "a@a.com",
  "vatId": null,
  "registrationNumber": null,
  "residences": [
    {
      "id": 2,
      "updatedTime": "2017-04-17T11:58:26Z",
      "createdTime": null,
      "street": "Street",
      "city": "City",
      "zip": "122",
      "latitude": null,
      "longitude": null,
      "clientId": null,
      "description": "Residence",
      "country": "CZ"
    }
  ]
}
```


Ukázka 2.1: Reprezentace zdroje

2.4.3 Query parametry

Následující zdroje v této sekci podporují následující query parametry.

Parametr	**Popis**
query	text k nalezení ve jméně/názvu
page	číslo stránky
size	počet položek stránky
sort	pole, dle kterého řadit položky

Zdroje podporující query parametry s případnými doplňkovými parametry:

Zdroj	**Parametr**	**Popis**
`/tickets`		
	state	id stavu ticketu (nový, řešený, ...)

	type	id typu ticketu (problém, dotaz, doprava, spam, ...)
	priority	id priority (urgentní, běžná, ...)
	group	id skupiny ticketů
	assignee	id řešitele ticketu
	followed	zda získat pouze sledované tickety
`/people`		
	company_query	text k nalezení ve jméně společnosti, ve které jsou hledaní lidé zaměstnaní
	company	id společnosti, ve které lidé pracují
`/users`		
	client	id klienta, ke kterému hledaní uživatelé přísluší
	group	id skupiny klienta, ve kterých jsou daní uživatelé
`/companies`		
`/locations`		
`/clients`		
`/products`		

Zdroje nepodporující vyhledávání a stránkování:

Zdroj	**Parametr**	**Popis**
`/{id klienta}/modules/{id modulu}/settings`		
	codename	kódové jméno nastavení
	value	hodnota nastavení
`/clients/{id klienta}/locations/{id}`		
	force	zda má být adresa zcela smazána

2.4.4 HTTP hlavičky

Následující tabulka obsahuje klíčové HTTP hlavičky podporované v komunikaci s API.

Hlavička	**Metoda**	**Použití**
Location	POST/PUT	obsahuje URI nového/změněného zdroje
Link	GET	obsahuje navigaci na předchozí/další/-poslední stránku při stránkování
X-Total-Count	GET	celkový počet výsledků při stránkování
Content-Type	*	popisuje typ dat, typicky application/json

Content-Encoding	*	popisuje kódování dat
Date	*	čas odeslání zprávy
Authorization	*	autorizace, typicky pro žádost a přenos bearer tokenu

Stavové kódy

API vrací následující HTTP stavové kódy, založené na HTTP/1.1 specifikaci [33].

- **200 OK** – Úspěšné splnění požadavku. V doprovodném těle je obsažen požadovaný obsah.
- **201 Created** – Úspěšné vytvoření nového zdroje. Typicky zároveň obsahuje pole Location s URI nového zdroje.
- **204 No Content** – Server splnil požadavek. Odpověď nemá tělo. Využito zejména jako odpověď `DELETE` metody.
- **400 Bad Request** – Chybný požadavek. Obvykle chyba deserializace JSON těla, požadavek neodpovídá kontraktu dané metody.
- **401 Unauthorized** – Uživatel není autorizován. Zpravidla je odpovědí na požadavek s chybným/žádným bearer tokenem.
- **403 Forbidden** – Uživatel nemá přístup k dané operaci.
- **404 Not Found** – Zdroj s daným URI nebyl nalezen.
- **405 Method Not Allowed** – Metoda není podporována daným zdrojem.
- **500 Internal Server Error** – Neočekávaná chyba serveru. Tato chyba by měla být nahlášena.

2.5 Architektura systému

Systém bude složen ze tří hlavních částí. První částí je databázový server, který uchovává data, se kterými celý systém pracuje. Druhou částí je aplikační server. S databázovým strojem komunikuje skrze rozhraní JDBC (Java Database Connectivity). Poskytuje RESTful rozhraní, které je dostupné k připojení klientů. Zpracovává data od klientů, data směřující ke klientům, obsahuje aplikační logiku, ověřuje uživatele, jejich oprávnění a tak dále. Třetí částí je webový klient. V tomto případě aplikace založená na frameworku Angular. Ta komunikuje s aplikačním serverem skrze protokol HTTP, respektive HTTPS v případě produkčního prostředí.

2.6 Uživatelské rozhraní

Pro možnost diskuse nad modelem rozhraní s možností snadných a rychlých úprav je vytvářen tzv. wireframe (v překladu „drátěný model“), koncept uživatelského rozhraní. Koncept je návrhem struktury aplikace, dostupných ovládacích prvků a podobně. K vytvoření tohoto modelu byl zvolen nástroj Balsamiq Mockups [34]. Oproti modelu přímo v HTML a CSS nabízí rychlé přetahování prvků na konkrétní pozici v modelu (princip drag and drop).

Koncept rozhraní je vytvářen v souladu s principy Material Designu, souboru pravidel založených na aplikaci teorie návrhu rozhraní a osvědčené praxe. Obsahuje pravidla/doporučení pro pohyb prvků rozhraní, barvy, typografii, rozmístění prvků, vzhled konkrétních prvků, ... Definuje zejména chování mobilních aplikací, ale doporučení jsou popsána i při zásadních rozdílech chování u webových aplikací.

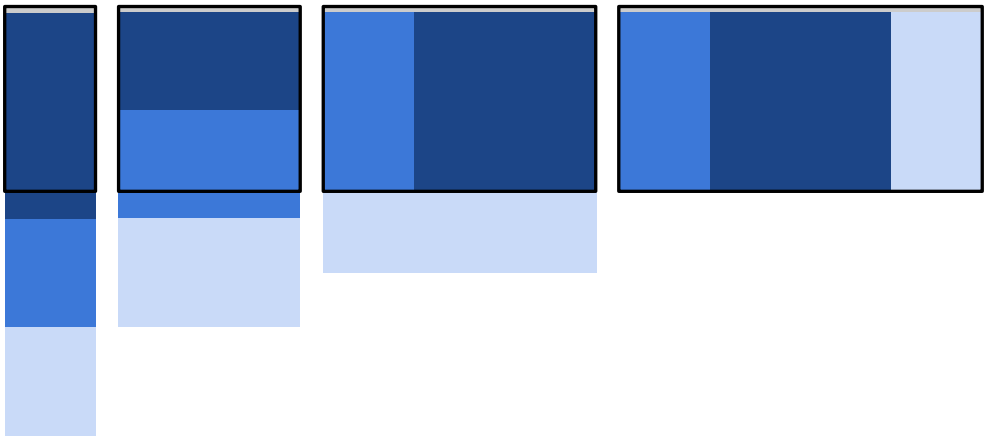
Rozhraní v rámci celé aplikace obklopuje totožná struktura. Horní lišta obsahující aktuálně vybranou sekci, aktuální výběr klientské firmy a aktuálně přihlášeného uživatele. Aktuální výběr klientské firmy je zároveň menu pro výběr jiné firmy. Menu aktuálně přihlášeného uživatele obsahuje možnost přechodu do nastavení či odhlášení. Levá lišta obsahuje navigaci skrz sekce aplikace. Detail vybraných položek navíc v horní liště zobrazuje tlačítko zpět. To způsobuje zrušení provedených změn. Tlačítko má za úkol eliminovat separátní editační mód, který zdržuje nuceným přepnutím módu před provedením změn.

Wireframy jsou navrženy pro primární zobrazení, tedy pro webový prohlížeč na počítači či velkém tabletu. Responzivní verze bude viditelná na mobilních zařízeních s menším displejem. Založena bude na designovém vzoru Column Drop [35]. Tento vzor je založen na umisťování sloupců pod sebe se zmenšující se šířkou displeje / okna prohlížeče, zásadní změnou web prochází až při přechodu do mobilního zobrazení. Princip tohoto vzoru je zobrazen na obrázku 2.4. Navigace sekcemi bude v mobilním zobrazení otevřena dodatečným tlačítkem v horní navigační liště, po jehož stisku se zobrazí menu překrývající současný obsah. Návrhy rozhraní je možné prohlédnout v příloze D.

2.6.1 Uživatelská sekce

Seznam problémů

Seznam problémů sdružuje obecné problémy zákazníků. Zobrazeny jsou v tabulce, která umožňuje řazení dle dat v jednotlivých sloupcích. Umožněno je vyhledávání dle názvu problému. Problémy je možné třídit dle vybraných kritérií, skupin problémů či řešitele.



Obrázek 2.4: Responzivní vzor Column Drop [35]

Řešení problému

Řešení problému je nejzásadnější obrazovkou v systému. Středobodem této obrazovky je možnost přidání komentáře s volbou viditelnosti komentáře. K problému je také možné přiřadit přílohu. Komentář je možné založit na předem připravené šabloně. Další sekcí je přehled aktivit, který obsahuje aktivity vykonané na aktuálním ticketu, seřazené od nejnovější události.

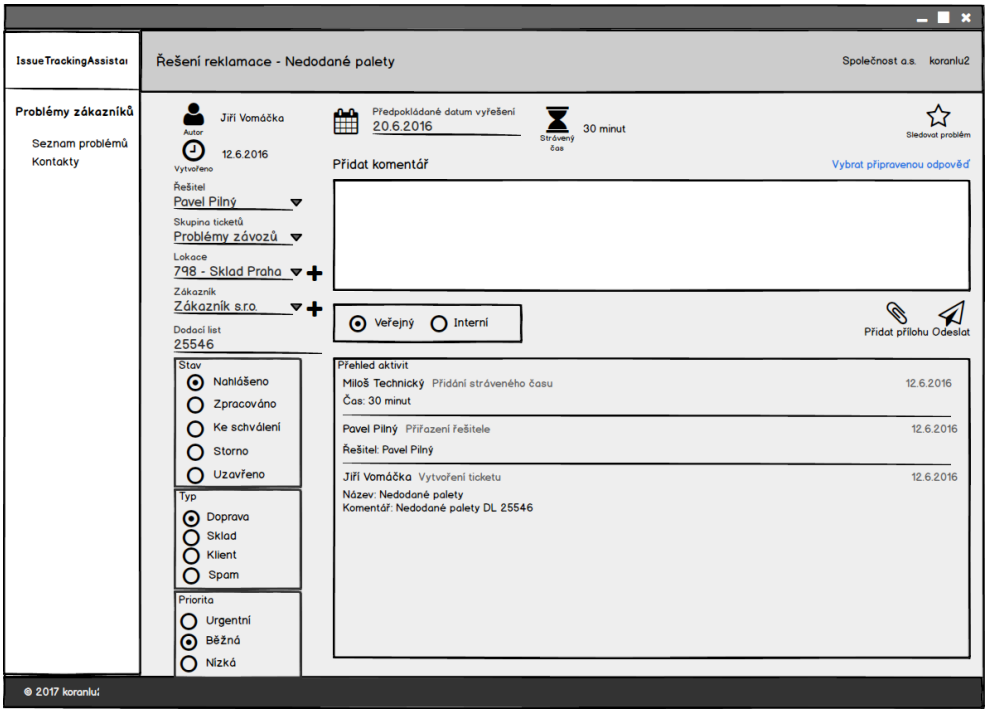
V levé liště se nachází výběr stavu, typu a priority ticketu. Dále je možné vybrat skupinu ticketů a řešitele. Také je možné vyplnit předpokládané datum vyřešení problému, přidat strávený čas či přidat ticket mezi sledované. Také jsou zobrazeny statické informace – autor problému a čas vytvoření problému.

Seznam reklamací

Seznam reklamací je založen na stejném základu jako seznam problémů, rozdílem je míra detailů jednotlivých reklamací a odlišné stavy, skupiny a typy reklamací.

Řešení reklamace

Řešení reklamací opět kopírují strukturu řešení problému se změnami ve stavech, typech a skupinách problémů. Navíc je pro tento typ problémů klíčové znát lokaci a zákaznickou firmu, které se reklamace týká, proto je možné i tyto hodnoty vyplnit. Lokace a zákazníky lze hledat v podobě automaticky dokončovaného výběrového pole. V případě, že lokace či zákazník neexistuje, je možné ho vytvořit.



Obrázek 2.5: Návrh editace reklamace

Vytvoření lokace

Ke snadnému vytvoření neexistující lokace slouží dialog vytvoření lokace. Požaduje zásadní informace pro vytvoření lokace. Po úspěšném uložení dojde k automatickému výběru lokace v rámci právě otevřené reklamace.

Seznam kontaktů

Seznam kontaktů slouží k přehledu o kontaktních osobách, zákaznících společnosti. V tabulce jsou zobrazeny základní údaje s možností filtrování.

Seznam společností zachovává shodnou strukturu.

Editace kontaktu

Kontaktní osobě lze měnit základní údaje o osobě, adresy (ať už bydliště či zaměstnání), s možností uchovávat zaměstnání. To může být užitečné například při nereagování dané osoby, dle jeho zaměstnání je možné dohledat další osoby.

2.6.2 Administrace

Administrace je složena ze dvou částí, administrace v rámci klientské firmy a administrace celého systému. Struktura je do jisté míry obdobná, seznam uživatelů, stejně jako jejich editace, se vyskytuje v obou částech. Rozdílem jsou požadovaná oprávnění a míra detailů.

Seznam uživatelů

Seznam uživatelů umožňuje zobrazení informací o existujících uživatelích. Administrátorům celého systému je umožněno zobrazit i přiřazené klienty.

Editace uživatele

V rámci editace uživatele je třeba vybrat uživatelské jméno a heslo. Editace se opět lehce odlišuje dle oprávnění administrátora. Pokud se jedná o klientského administrátora, může uživateli připojit taková práva, jaká jsou povolena pro celého klienta. Administrátor celého systému navíc může uživateli povolovat i další klienty a oprávnění v jejich režii. Také je možné zvolit skupiny uživatelů. Dále je možné uchovat údaje o osobě a zaměstnání.

Seznam klientů

Seznam klientských firem nabízí náhled na klientské firmy v rámci systému.

Editace klienta

V rámci vytvoření či změny klientských firem je možné povolit či zakázat jim dostupné oprávnění či celé moduly.

Nastavení klienta

Nastavení klienta seskupuje nastavení, které nastavují administrátoři klienta. Je možné vytvářet skupiny problémů v rámci jednotlivých modulů systému. Do těchto skupin je následně možné zařazovat problémy. Také je zde možné vytvářet šablony odpovědí.

Přihlášení uživatele

Přihlášení po uživateli vyžaduje uživatelské jméno a heslo, v případě chyby je mu sděleno, že vyplněné údaje nejsou správné. Není možné upozornit, zda se jedná o nesprávné jméno či heslo, toto by systém vystavovalo ohrožení. Ve chvíli, kdy by byla známá existence daného uživatelského jména, by mohl být použit slovníkový či jiný útok k prolomení uživatelského hesla.

Kapitola 3

Realizace

3.1 Vývojové prostředí

Ke snazšímu vývoji aplikací je využito integrované vývojové prostředí (Integrated Development Environment – IDE) IntelliJ IDEA. To usnadňuje vývoj jak aplikačního serveru, tak webové aplikace. Obsahuje doplňky, ať už přímo od výrobce či od komunity, doplňky pro integraci námi využívaných technologií. Konkrétně je využita integrace s frameworkem Spring, která usnadňuje navigaci mezi částmi aplikace. Integrace s databázovou vrstvou umožňuje přehlednou navigaci mezi mapovanými vztahy a přímé zobrazení tabulek a jejich vlastností. Pro vývoj webové aplikace nabízí podporu jazyka TypeScript, šablon frameworku Angular, podporu jazyka CSS i SASS. V šablonách je navíc nabízena služba Angular, která kontroluje chyby přímo při vývoji. K práci s frameworkem Angular Material nabízí doplněk potřebné rychlé návrhy kódu v šablonách. K dodržování konvencí kódu webového klienta nabádá nástroj TSLint, konfigurovaný skrze soubor *tslint.json*. Další nastavení pro styl kódu se nachází v souboru *.editorconfig*.

Ke správě závislostí se pro server používá systém Gradle (více popsáno v sekci 2.2.1). Ten je také integrován v rámci IDE, je možné spouštět separátně úkoly, například pro vytvoření jar archivu, dokumentace a dalších bez příkazové řádky. K rychlejšímu zavádění změn do běžícího serveru jsou použity Spring Boot Developer Tools [36]. Ve chvíli, kdy běží vývojářský server a jsou zkompilovány změněné zdrojové soubory, automaticky restartuje běžící server. To oproti manuálnímu restartování zrychlí načtení zhruba o polovinu.

Pro vývoj webové aplikace je integrován systém npm, manažer balíčků pro projekty využívající JavaScript [37]. IDE opět dovolí spouštět skripty bez použití příkazové řádky. Také je dostupná integrace s Angular CLI [38]. Slouží k usnadnění vytváření částí aplikace, umožňuje generovat šablony služeb, komponent, modulů, ... CLI kompiluje aplikace a umožňuje provozovat vývojářský server. Ten byl v průběhu vývoje používán, jelikož umožňuje kompilaci při změně zdrojových souborů a tím urychluje vývoj. CLI také slouží k importo-

vání modulů (stažených pomocí npm) skrze integrovaný webpack balíčkovací systém [39].

K uchovávání zdrojových kódů, respektive verzí a změn mezi nimi, byl využit verzovací systém Git [40]. Jako úložiště dat posloužil portál GitLab, který nabízí neomezený počet repozitářů zdarma [41].

3.2 Specifika implementace a použité knihovny

Aplikační server i webový klient využívají vzoru vkládání závislostí (Dependency Injection – DI) ke správě závislostí v rámci aplikace. Tento vzor, také známý pod názvem Inversion of Control (IoC), je ověřený a frameworky doporučený k tomuto účelu. Ve frameworku Spring je tento princip implementován pomocí IoC kontejneru [42]. Reprezentován je pomocí aplikačního kontextu (třídou implementující rozhraní ApplicationContext). Kontext je následně použit ke konfiguraci, vkládání závislostí konstruktorem a nespočtu dalších funkcí. V rámci frameworku Angular je DI využita zejména k injektování závislostí na službách pomocí konstruktoru [43]. Injektované služby jsou jak ve frameworku Spring, tak Angular, obvykle singletony, jejich použití je tedy nejen praktičtější (podporují lepší testovatelnost a udržovatelnost), ale i efektivnější než konstruování nových objektů přímo.

3.2.1 Server

Práce s databází *Spring Data JPA*

Knihovna Spring Data JPA [18] umožňuje vytvářet datové repozitáře, kterými lze snadno definovat dotazy nad databázovým úložištěm. Konkrétně skrze rozhraní JPA [44]. Repozitáře jsou definovány jako interface s anotací `@Repository`. Ten následně může rozšiřovat repozitáře definované knihovnou. Hlavním užitým existujícím repozitářem je `PagingAndSortingRepository`, který sestavuje metodu hledání dle objektu implementujícího `Pageable` interface. V něm lze definovat velikost stránky k navrácení, číslo stránky, kterou chceme vrátit a další údaje. Navíc obsahuje i pole typu `Sort`, rozhraní, které definuje dle kterých vlastností objektů budou výsledky řazeny. Tento repozitář dále rozšiřuje `CrudRepository`, který obsahuje základní metody typu uložení, nalezení dle id, nalezení všech, smazání a podobně. Tyto základní metody samozřejmě nejsou dostatečné pro složitější případy, z toho důvodu je možné definovat vlastní dotazy. K definici dotazů je možné použít názvy metod, Java Persistence Query Language (JPQL) v rámci anotace `@Query` a dotazy pomocí QueryDSL rozšířením repozitáře `QueryDslPredicateExecutor` rozhraním. Výhodou tohoto přístupu oproti dotazům přímo nad Hibernate Query Language (HQL) je větší univerzálnost. Program následně není závislý přímo na Hibernate, ale na knihovně, která implementuje JPA rozhraní.

```
//dotaz pomocí názvu metody
Page<ApplicationClient> findAllByNameContainingIgnoreCase
        (Pageable pageable, String query);

//dotaz pomocí JPQL
@Query("select case when count(e) > 0 " +
        "then true else false end " +
        "from IssueTicket e join e.followers f " +
        "where e.id = ?1 and f.id = ?#{ principal?.id }")
boolean isUserFollowing(Integer ticketId);

//dotaz pomocí QueryDSL
QIssueTicket ticket = QIssueTicket.issueTicket;
BooleanExpression predicate =
  ticket.applicationClient.id.eq(appClientId)
        .and(applicationModule.id.eq(moduleId));
Page<IssueTicket> result =
  ticketRepository.findAll(predicate, pageable);
```

Ukázka 3.1: Dotazy nad repozitářem Spring Data JPA

Uživatelská oprávnění *Spring Security*

K podpoře uživatelských oprávnění v rámci aplikačního serveru jsou využity komponenty frameworku Spring. API využívá protokolu OAuth 2 k autorizaci požadavků. Při žádosti o bearer token klient zasílá serveru svoji identifikaci (client id) a uživatelské údaje uživatele. Tento přístup používá typ ověření (grant type) heslem. Na serveru následně implementace rozhraní `UserDetailsService` ověří daného uživatele, v případě, že jsou údaje korektní, je klientovi vrácen bearer token. Uživatel může mít roli administrátora celého systému, případně role klientské, buď v rámci klienta, nebo v rámci klienta a modulu.

Role jsou vytvořeny na základě oprávnění, jelikož samotné uživatelské role nenabízejí dostatečné rozlišení oprávnění. Uloženy jsou v databázi a jsou vytvářeny pro každého klienta v následujícím formátu: `ROLE_{id klienta}_{název oprávnění}`, případně `ROLE_{id klienta}_{id modulu}_{název oprávnění}`. Tento formát zachovává základní podporu Spring rolí ve formátu `ROLE_{název role}`, jednodušší dotaz nad databází nad jedním sloupcem a pro snazší práci s rolemi u klienta. Oprávnění jsou následně ověřována v kritických částech serveru, obvykle ve službách, občas v controlleru pro větší granularitu oprávnění. Pro snadnou kontrolu oprávnění byl rozšířen `SecurityExpressionRoot` (spravující systémovou bezpečnost) o vlastní metody zabezpečení. Tento způsob zabezpečení se nazývá řízení přístupu na základě výrazů

(Expression-Based Control). Jazyk k jejich vyjádření se nazývá Spring Expression Language (SpEL). K ochraně přístupu jsou následně použity anotace `@PreAuthorize()`, které ověří výraz před vstupem do metody a v případě chybějícího oprávnění server vrátí chybu typu neoprávněný přístup. Ukázkou využitého výrazu je například `@PreAuthorize(hasClientPermission(-#clientId, 'PEOPLE_READ'))`. Tento přístup je navíc dále rozšiřitelný o další výrazy, ochranu konkrétních objektů pomocí metod `hasPermission()` a dalších. Přístup byl vyhodnocen jako dostačující, přehlednější, praktičtější a méně náročný než implementace celé vlastního Access control listu (ACL, v překladu seznamu pro řízení přístupu) dle doporučení pro framework Spring. Role navíc podporují hierarchii. Ta je vytvořena také z databáze, pomocí Spring komponenty, která při svém vytvoření role projde a sestaví z nich řetězec hierarchie rolí. Ten umí zpracovat implementace integrovaná ve frameworku. Vytvořením nového klienta se ovšem mění i hierarchie. K tomu je použita anotace `@RefreshScope`, s jejíž pomocí je po vytvoření nového klienta komponenta znovu sestavena, čímž se hierarchie obnoví. Komponenta je implementací Spring `RoleHierarchy`, ale slouží jako fasáda pro skutečnou implementaci, která může být dynamicky vyměňována. Při ověření rolí je následně povolen přístup i uživateli s nadřazenou rolí, tedy například uživatel s oprávněním zápisu kontaktů je smí i číst. Administrátor celého systému je nadřazen všem dostupným rolím.

Zpracování e-mailů *Spring Integration, JavaMail*

S pomocí knihoven Spring Integration Mail [45] a Javax Mail [46] je na aplikačním serveru implementována funkcionalita zpracovávání e-mailů. Při spuštění aplikace jsou načtena nastavení jednotlivých klientů v systému, následně je pro každý z nich spuštěno načítání e-mailů. Pro každou e-mailovou schránku, ze které mají být přijímány e-maily, je vytvořena instance třídy `ImapIdle-ChannelAdapter`. Z toho důvodu je pro přijímání e-mailů v rámci prototypu systému podporován pouze protokol IMAP a server s podporou funkce idle. Ke správě běhu těchto adaptérů je použit `TaskScheduler`, který se stará o vhodné využití vláken. Tento způsob je šetrný k systémovým prostředkům i k e-mailovému serveru. V případě, že administrátor změní nastavení e-mailu za běhu, je původní přijímací adaptér zastaven a spustí se nový, s již změněnými údaji.

Pro zpracování přijatých e-mailů je využit `DirectChannel`, kanál, který po přijetí zprávy provede definovanou akci. Zprávy následně zpracuje konkrétní implementace rozhraní `MessageHandler`. Problémem této třídy je její vytváření za běhu dle konfigurace jednotlivých klientů systému. Z toho důvodu je obtížné v ní použít vkládání závislostí, ale také udržet v ní databázovou transakci standardně udržovanou pomocí anotace `@Transactional`. Řešením tohoto problému je využití přímo instance JPA `EntityManager` bez dalších abstrakcí, například v podobě repozitáře. Transakce je pak vytvořena právě

na této instanci, na které probíhají i dotazy na databázi a příkazy k uložení dat. V rámci zpracování příchozí zprávy je následně zjištěno, zda jde o odpověď na e-mail, tedy zda se má zprávy přiřadit k již existujícímu ticketu. V případě, že jde o novou zprávu, je vytvořen nový ticket. Dále se zjistí, zda již v systému existuje daná kontaktní osoba. Pokud ano, jejím autorem se stane totožná osoba a tím získáváme možnost snadného nalezení dalších zpráv od totožného autora. Když e-mail obsahuje přílohy, jsou ticketu také přiřazeny.

K odesílání e-mailů skrze SMTP server je opět použita knihovna Javax Mail. Nejprve je zjištěno, zda již existuje vlákno e-mailů, pokud ano, je z databáze načtena minulá zpráva v serializované formě. Původní zprávu je možné knihovně předložit s žádostí o odpověď, knihovna následně provede potřebná nastavení pro správné vytvoření vlákna e-mailů. Pokud ještě nebyl přijat žádný e-mail, je vytvořena nová zpráva dle zadané e-mailové adresy.

3.2.2 Klient

Framework pro webovou aplikaci *Angular Core*

Knihovna Angular Core je základním stavebním prvkem pro vývoj aplikací ve frameworku Angular. Tato část slouží jako nastínění architektury aplikací využívajících framework Angular. Slouží jak ke snadnějšímu pochopení zdrojového kódu webové aplikace, tak k lepšímu porozumění následujících částí textu. Zdrojem pro tuto část je dokumentace frameworku, zejména [47].

- **Modul** Moduly slouží k organizaci aplikace a jejímu rozšíření pomocí knihoven. Hlavním vstupním bodem aplikace je kořenový modul. Další moduly slouží k oddělení funkcí do samostatných celků. Takovým modulem může být celá knihovna určená k jedné funkci, nebo modul uvnitř aplikace, například administrace.

- **Komponenta** Komponenty slouží k práci s vrstvou uživatelského rozhraní. Je velmi podobná controlleru v návrhovém vzoru MVC (Model-View-Controller).

- **Šablona** Při srovnání s MVC se jedná o vrstvu view. Jde o definici vzhledu pro komponentu. Na rozdíl od ostatních komponent se zapisují prostřednictvím jazyka HTML s malými úpravami. Základními doplňkovými elementy se zápisem uvedeným v závorce jsou:

  - **Interpolace** (`{{...}}`) Vyhodnocuje výrazy a následně je reprezentuje jako řetězec. Podporuje operátory roura (|, v originále pipe) a operátor bezpečné navigace (?.). Roury lze použít například k formátování data, měny apod.
  - **Binding vlastností** (`[vlastnost]`) Slouží k nastavení vlastností typu zdroj obrázku, parametr podřazené komponenty, ...

  - **Binding událostí** (`(událost)`) Specifikuje chování při dané události. Například funkce po kliknutí, události v child komponentě, ...
  - **Obousměrný binding** (`[(ngModel)]="vlastnost"`) Vazba mezi vstupním polem šablony a proměnnou komponenty.
  - **Direktivy** Existují atributové a strukturální direktivy. Atributové se zapisují jako HTML atribut, případně v hranatých závorkách, pokud je třeba jim předat výraz. Využity jsou například k dynamickému přiřazení HTML třídy. Strukturální se zapisují pomocí hvězdičky, například *ngIf. Mohou měnit strukturu HTML DOM (Document Object Model). NgIf je schopné skrytí elementu z HTML DOMu, ngFor zobrazuje elementy z kolekce atp. Je možné implementovat vlastní direktivy obou typů.

- **Metadata** K označení tříd pro rozpoznání frameworkem Angular se používají metadata. V jazyce TypeScript se značí pomocí dekorátoru u dané třídy. Dekorátor přijímá konfigurační objekt, který slouží k nastavení dané třídy. Jsou jí přiřazeny například selektory pro využití v HTML, adresa šablony a podobně. Příkladem zápisu pro komponentu je `@Component({...})`.

- **Služba** Slouží k udržení přehlednosti a jednoduchosti komponent. Jsou hlavním místem pro aplikační logiku. Typicky jsou využívány ke komunikaci s API.

Navigace v aplikaci *Angular Router*

Jelikož se pohybujeme v prostředí SPA, nejsou k navigaci mezi stránkami využity standardní HTML `<a href="">` elementy, které vyvolávají přechod na novou stránku a v našem případě nové načtení celé aplikace. Pro přechody mezi stránkami ve frameworku Angular slouží knihovna Router [48]. Pro každý modul aplikace jsou cesty definovány zvlášť pomocí pole cest s jejich definovanými vlastnostmi. Definované cesty mohou mít i podřazené cesty, kterým jsou rodičem. Užívaným názvem jsou child routes. Tyto cesty sdílejí začátek URL se svým rodičem, zároveň mohou sloužit pro automatické rozpoznání místa v hierarchii.

```
<a routerLink="/administration"
   routerLinkActive="active">Administration</a>
```

Ukázka 3.2: Použití odkazů směrovaných přes router

Pro lepší představu viz ukázku 3.2. Parametrem direktivy `routerLink` je cesta definovaná v rámci modulu. Direktivou `routerLinkActive` žádáme o přidání HTML třídy active v případě, že je cesta aktivována, případně odebrání ve chvíli, kdy aktivována není. Pokud má navíc cesta `/administration`

definované child routes, případně jiná cesta začíná také řetězcem `/administration`, je třída active přiřazena také. To je užitečné při tvorbě navigace v rámci aplikace a splnění bodu „Visibility of system status“ Nielsenovy heuristiky (více v sekci 4.2). Spolu s definicí cest je možné definovat tzv. guards.

- `CanActivate` guard – Zabraňuje neoprávněnému přístupu na stránku. Ve webové aplikaci je využit k zabránění přístupu nepřihlášeným uživatelům a uživatelům bez potřebného oprávnění/role.
- `CanActivateChild` guard – Zabraňuje v přístupu na child routes v rámci modulu.
- `CanDeactivate` guard – Zabraňuje odchodu z aktuální stránky. Typicky se využívá k dotazu na zrušení změn v případě odchodu ze stránky bez explicitního uložení.
- `CanLoad` guard – Zabraňuje načtení modulů, ke kterým uživatel nemá přístup. Spolu s lazy načítáním a přednačítáním jsou vhodnými nástroji k optimalizaci výkonu aplikace. V rámci prototypu nejsou využity.
- `Resolve` – Slouží k přednačtení dat před přechodem na stránku. Odpovědnost za načtení dat, případně neexistenci načítaných dat, je tudíž zanechána na routeru.

Formuláře a validace vstupu *Angular Forms*

Webová aplikace vyžaduje využití formulářů pro editaci dat. Framework Angular opět nabízí potřebné nástroje ke snadné realizaci formulářů v prostředí SPA. K dispozici jsou dva přístupy tvorby a zpracování formulářů. Prvním přístupem jsou formuláře řízené šablonou, druhým reaktivní formuláře.

Template–driven forms [49], neboli formuláře řízené šablonou stránky, nabízejí obousměrný data binding (v překladu „navázání dat“). To v praxi znamená navázání prvku v šabloně (například HTML `<input>`) na proměnnou v komponentě (v případě zmíněného `<input>` například vazba na proměnnou typu string či number). Tento přístup nabízí běžnou HTML syntaxi pro formulář, kde jsou navíc vstupním polím přiřazeny atributy `[(ngModel)]`, které definují, na které proměnné se pole váže. Validaci je možno určovat skrze direktivy v šablonách, ať už standardními HTML5 (například `required`), tak vlastními direktivami. Polím jsou navíc dle jejich stavu knihovnou přiřazeny následující vlastnosti, respektive HTML třídy. Na základě těchto vlastností je možné uživateli zobrazit chyby validace.

Stav	Pokud pravda	Pokud nepravda
Pole navštíveno	ng-touched	ng-untouched
Hodnota pole změněna	ng-dirty	ng-pristine
Pole je validní	ng-valid	ng-invalid

```
export function validateMoreThanZero(): ValidatorFn {
  return (c: FormControl) =>
    c.value > 0 ?
      null : {validateMoreThanZero: {valid: false}};
}

private createForm(): FormGroup {
  return formBuilder.group({
    elapsedTime: [time, [Validators.required,
      validateMoreThanZero]]
  });
}
```

Ukázka 3.3: Validace číselného pole

Reactive forms [50], neboli reaktivní formuláře, mají jiný přístup ke zpracování formulářů. Jejich zobrazení je opět definováno v HTML šablonách. Chování je ale, na rozdíl od template–driven formulářů, synchronní. Jejich obsah je možné změnit pomocí zdrojového kódu, tato změna je ihned reprezentována v zobrazení. Jejich synchronní povaha umožňuje i jednodušší testovatelnost. Není nabízen obousměrný data binding, vždy pracujeme přímo s objektem daného formuláře. Z toho důvodu je tato metoda náročnější na code behind (zdrojový kód komponent). Tímto oddělujeme datový model (typicky přejatý ze serveru) a model formulářový (tzn. to, co je zobrazeno). Šablonu se zdrojovým kódem provazujeme direktivou `[formGroup]="nazevPromenne"` na HTML elementu `<form>` a `formControlName="nazevPrvku"` na jednotlivých vstupních polích. Formuláře se vytváří tzv. form builderem. Objekt formuláře je strukturován následujícím způsobem. FormGroup je kořenovým elementem formuláře. Může obsahovat formControl – vstupní pole či formArray – pole dalších formGroups. Jednotlivé formControl vystavují Observable, na kterém je možné poslouchat změny ve vstupním poli, případně v celém formuláři. Navíc nabízí snadné programové rozhraní pro definici formulářů a jejich validaci. Díky programovému přístupu k tvorbě formulářových objektů je možné validaci přidávat i podmíněně, například při prvním vytvoření může být využita jiná validace, než při editaci. Další vítanou vlastností je snadné rozdělení podformulářů do komponent. Ve webové aplikaci je tato funkcionalita využita například při editaci adres. Editace adresy je totiž využívána ve více formulářích, vlastní komponenta obsahující její editaci tedy slouží ke znovupoužitelnosti. Navíc i při rozdělení do podkomponent stále stačí držet referenci na kořen formuláře, tedy hlavní formGroup. Té je umožněn přístup i do podformulářů, tudíž je možné vyhodnotit celý formulář kompletně z rodičovské komponenty. Po vyhodnocení formuláře je nutné opět namapovat formulářový objekt na datový objekt určený k přenesení na API.

V rámci aplikace jsou použity oba způsoby definice formulářů. Typicky jsou u jednodušších formulářů využívány formuláře řízené šablonou a u složitějších reaktivní formuláře, které umožňují snazší rozdělení do podkomponent.

Komunikace s API *Angular Http*

Angular framework poskytuje vlastní knihovnu, respektive zejména fasádu, pro práci s HTTP požadavky. Odstiňuje od přímého použití XMLHttpRequest (XHR) a JSONP programového rozhraní. Knihovna tedy slouží k požadavkům na webové API a zpracování odpovědí. Požadavky na server jsou obvykle umístěny ve vhodné službě, kde je definováno volání, zpracování případných chyb a mapování na konkrétní návratový objekt. Tímto způsobem podporujeme oddělení zodpovědností (návrhový vzor Separation of Concerns) jednotlivých částí aplikace. Do komponent se tudíž vrací konkrétní instance třídy a nikoliv generická odpověď (respektive instance objektu Response). Knihovna používá objekty typu Observable z důvodu ideálních vlastností vzhledem k povaze požadavků na server. Více o Observable objektech, respektive celé knihovně RxJS v sekci 3.2.2. Zásadním důsledkem použití Observable a zavedené praxe je, že metody služby vrací pouze Observable, tedy definici volání. To v praxi znamená, že zavolání metody na službě nevyvolá požadavek na server, ale pouze vrátí Observable. Ukázku použití viz na ukázce 3.4. Reálného volání serveru docílíme například metodou `subscribe`.

```
// kód v komponentě
this.service.get(id).subscribe(res => this.single = res);

// kód ve službě
get(id: number) {
  return this.http.get(`${endpoint}/people/${id}`);
}
```

Ukázka 3.4: Použití knihovny Http

Pro jednodušší škálování aplikace a nezávislost aplikačních serverů je aplikačním serverem podporován mechanismus CORS (Cross-origin Resource Sharing) [51]. Tento mechanismus je přímo vyžadován v určitých situacích klientskou knihovnou Http, respektive její součástí XMLHttpRequest. Situací, ve které je vyžadována podpora CORS, je požadavek na zdroj z jiné domény či portu. V tuto chvíli server a klient využívá jiný aplikační server, každý na jiném portu, proto je CORS podpora vyžadována. Pro určité HTTP metody, HTTP hlavičky a typ obsahu je vyžadován navíc takzvaný „preflight požadavek“. V našem případě je vyžadován téměř vždy, z toho důvodu, že je využit typ obsahu (hlavička `Content-Type`) `application/json`, který preflight požadavek vyžaduje. Preflight požadavek využívá HTTP metody OPTIONS,

kterou vyšle požadavek s hlavičkami `Access-Control-Request-Method`, specifikující zamýšlenou HTTP metodu pro reálný požadavek, `Origin` pro specifikaci původce požadavku a `Access-Control-Allow-Headers` pro specifikaci hlaviček k odeslání. Server následně vrací odpověď s následujícími hlavičkami:

1. `Access-Control-Allow-Origin` pro specifikaci povolených původců požadavků na server.
2. `Access-Control-Allow-Methods` specifikující povolené HTTP metody.
3. `Access-Control-Allow-Headers` s povolenými HTTP hlavičkami (bez tzv. bezpečných hlaviček).
4. `Access-Control-Max-Age` určující počet vteřin platnosti údajů v těchto hlavičkách. Prohlížeč obvykle hlavičky uloží do mezipaměti po stanovenou dobu.

V případě shody požadované specifikace v preflight požadavku a odpovědi serveru, je následně odeslán reálný požadavek na server.

Vzhled aplikace *Angular Material 2*

Angular Material 2 [52] je nástupcem knihovny Angular Material, která byla populární ke stylování aplikací založených na původním frameworku AngularJS. Snaží se minimalizovat nutné zásahy do CSS souborů a poskytnout jednoduchý nástroj k vytváření aplikací se vzhledem vyhovujícím doporučením Material. K použití jednotlivých prvků knihovny jsou obvykle využity direktivy či celé Angular komponenty.

V knihovně jsou obsaženy například vstupní pole, select boxy, seznamy, tlačítka, ikony, dialogy či menu. Nevýhodou knihovny oproti jiným je její nízká vyspělost. Ve vývoji je teprve od začátku roku 2016. Jiné knihovny, například Bootstrap, Materialize, jsou ve vývoji podstatně déle a existuje k nim více dokumentace a větší komunita uživatelů. Na druhou stranu se však jedná o CSS knihovny. Jejich využití znamená více práce s CSS, používání specifických HTML tříd a také nutnost využití knihovny jQuery. Ta sice je vyspělá, ale zároveň příliš objemná v situaci, kdy by byla využita pouze CSS frameworkem. Nevýhodou je také jazyk zdrojového kódu JavaScript a nikoliv přímo TypeScript. Výše zmíněné výtky se týkají frameworku Materialize. Pro Bootstrap existuje obalující knihovna, která nabízí Angular direktivy, takže je její užití obdobné jako u Material 2. Nicméně náš návrh rozhraní počítá s využitím vzhledu Material, ve kterém jsou některé prvky a doporučení lépe zpracované, navíc se jedná o vzhled živější a pro uživatele pravděpodobně líbivější.

Výhodou knihovny Angular Material 2 je přehlednost zdrojového kódu. Celý projekt je vystavěn přímo pro framework Angular a je také napsán v jazyce TypeScript. Cílem knihovny je vysoká kvalita nabízených komponent, přehledné programové rozhraní a kvalitní dokumentace. Každý prvek rozhraní

je podroben důkladnému návrhu, diskusi, implementaci a testování. Záštitu nad projektem má navíc Google, který vyvíjí i framework Angular. Vzhledem k velké popularitě předchozí knihovny Angular Material se navíc dá očekávat popularita i tohoto nástupce.

Některé zásadní prvky či funkce rozhraní bohužel chybí. V případě chybějících náročných prvků byly zvoleny alternativní knihovny, o kterých se lze dočíst dále. V případě chybějících drobnějších funkcí byla chybějící funkcionalita řešena košatějším zdrojovým kódem, než by bylo třeba například u původní knihovny Material. Osobně však volby knihovny nelituji. Jsem přesvědčený, že knihovna bude populární, dokumentace a funkce budou rozšířeny a že dočasná řešení v rámci prototypu aplikace půjde nahradit novými prvky ve chvíli, kdy budou dokončeny.

Animace *Angular Animations*

Animace jsou v moderních aplikacích důležitým prvkem pro zaujetí uživatele, ale i pro dokreslení významu či hierarchie prvků rozhraní. Animace jsou použity zejména při přechodu mezi obrazovkami typu seznam a detail. [53]

Responzivní chování *Angular Flex Layout*

Knihovna Flex Layout [54] slouží ke zjednodušení práce s responzivním designem a pozicí prvků na stránce. Odstiňuje od využití CSS souborů a tzv. media queries v nich. Media queries umožňují v CSS souborech definovat sekce, které se týkají pouze určitého typu zařízení, formátu a podobně. Typicky se využívá pro různá rozlišení displejů či pro styly ve verzi k tisku [55].

Díky knihovně je možné přímo v rámci HTML šablon na elementy aplikovat různé direktivy. Hlavními jsou `fxLayout` pro řazení prvků do sloupce či řádku, `fxFlex` pro nastavení velikosti prvků (například vzhledem k sobě) a `ngClass`, podmíněná HTML třída na základě podmínky. Pro práci s responzivním vzhledem jsou k dispozici předdefinované aliasy media queries. V rámci knihovny jsou definovány na základě šířky okna v pixelech. Direktivy tyto aliasy podporují a v důsledku je tímto způsobem možné použít například následující zdrojový kód.

```
<div fxLayout="column" fxLayout.gt-sm="row" fxLayoutGap="1em"
        class="flex-container">
        <div>...</div>
        <div>...</div>
</div>
```

Ukázka 3.5: Použití knihovny Flex Layout

Tento kód způsobí zobrazení vnořených prvků ve sloupci ve výchozím případě, v řádku pro šířku okna větší než malou (`gt-sm` znamená „greater than

small“). Direktiva `fxLayoutGap` navíc mezi prvky vytvoří mezeru pro přehlednější zobrazení. Konfigurace samotného chování je tedy pouze na prvním řádku. Ekvivalentní konfigurace pomocí CSS je zobrazena na ukázce 3.6. S pomocí SASS je možné kód zredukovat pomocí mix-inu, který odstíní od rozdílů mezi jednotlivými prohlížeči. Flex Layout tuto práci odvede sám. Navíc se i při použití SASS funkcionalit knihovna jeví jako pohodlnější a přehlednější.

```
.flex-container {
  flex-direction: column;
  -ms-flex-direction: column;
  -webkit-flex-direction: column;
}

@media screen and (min-width: 960px) {
  .flex-container {
    flex-direction: row;
    -ms-flex-direction: row;
    -webkit-flex-direction: row;
  }
}
```

Ukázka 3.6: Použití media queries

Podpora více jazyků *Ngx–translate*

Jedním z požadavků na webovou aplikaci je podpora více jazyků. Tento požadavek má mnoho aplikací, proto jistě existuje knihovna určená k tomuto účelu. Přestože je k dispozici oficiální doporučení k internacionalizaci a lokalizaci aplikace na oficiálním webu projektu Angular [56], bylo zvoleno alternativní řešení.

Překlad pomocí oficiálního nástroje probíhá označením textových zpráv k překladu, následnou extrakcí označených zpráv, samotným překladem v souboru typu XML a finální kompilací, která zamění zprávy k překladu za reálné překlady. Výhodou nástroje je možnost označit překládaný text kontextem překladu, který překladateli dovolí lépe pochopit, jakým způsobem se text používá. Vhodným doplňkem je také pluralizace, odlišné překlady dle číslovek či slov vyjadřujících množství. Podporována je i možnost volby překladu dle vlastnosti objektu, použitelná například u pohlaví osob. Díky náhradě při kompilaci je překlad šetrný na systémové prostředky klienta. Překladové soubory jsou uloženy v XML souborech, konkrétně ve formátu XLIFF, který se snaží o standardizaci překladových souborů. Předpokládá se překládání celých vět v původním jazyce (typicky anglických). Nevýhodou této metody je závislost na typu kompilace (AOT či JIT, více v sekci 3.3) a vytváření separátní apli-

kace pro různé jazykové verze. Navíc je nutné generovat identifikační řetězec pro překládané části, v případě změny překládaného řetězce se id mění a způsobuje chybu kompilace pro všechny jazykové verze. Také není možné (bez obcházení principu metody) využít řetězce v rámci zdrojového kódu, pouze v HTML šablonách.

K překladu Angular aplikací je také možné využít knihovnu Ngx–translate [57]. Vyznačuje se zejména větší volností práce s překlady. Je možné využít různých formátů uložení dat, ať JSON, .po soubory, či vlastní formát po doprogramování potřebných rozšíření. Také je možné využít různé metody načtení dat, například HTTP ve webové aplikaci, či načtení ze souborového systému při vykreslování na serveru nebo v lokální aplikaci. Větší svoboda volby je i v možnostech použití překladů. V rámci souborů se zdrojovým kódem lze překlad použít pomocí injektované služby. V šablonách je možné použít jak roury, tak direktivy, také je možné vkládat celé HTML elementy. Knihovna také podporuje změnu jazyka či překladu za běhu aplikace. Aplikaci tedy není třeba znovu kompilovat pro přidání jazyka či přeloženého slova. Na rozdíl od oficiálních překladů se používají kódová slova, případně s parametry, pokud by výsledný řetězec obsahoval nepřehlednou interpolaci řetězců. Nevýhodou je vyšší výpočetní náročnost, překlady jsou vždy klientem dohledávány. Na druhou stranu nastavení umožňuje stažení překladového souboru pouze při spuštění aplikace, následné vyhledávání je prováděno v operační paměti. Hledání v mapě řetězců pro současný hardware nepředstavuje žádný problém.

Z výše uvedených důvodů byla zvolena knihovna Ngx–translate, která nabízí větší svobodu při jejím užívání a více vyhovuje vyvíjené aplikaci.

Zobrazení tabulek *Ngx–datatable*

K přehlednému zobrazení dat jsou vhodné tabulky. Projekt Angular Material 3.2.2 má implementaci tabulek v plánu, ovšem v době tvorby práce ještě nebyly dostupné. Vybrána byla knihovna Ngx–datatable [58], která je dlouhodobě podporována a nabízí velké množství funkcí. Důležitými podporovanými funkcemi jsou stránkování a řazení skrz volání API. Vyhovuje také nabízeným stylem tabulek v designu Material. Použita je napříč celou aplikací při zobrazování seznamů položek. Ukázka 3.7 zobrazuje zjednodušený výňatek z aplikace, pro úsporu prostoru zobrazuje pouze jednu šablonu sloupce.

Výběr data *Ic Datepicker*

Ic Datepicker slouží k výběru data graficky, z kalendáře [59]. Důvodů pro použití této knihovny je několik. HTML5 `<input>` element definuje typ vstupu `date`, tedy datum. Ten většina prohlížečů rozpozná a korektně zobrazí klikací kalendář pro výběr data. V některých prohlížečích, například Firefoxu, tato funkce implementována není. Tito uživatelé by v důsledku byli odkázáni na zadávání data textem, což by vyžadovalo další validaci, navíc závislou na jazy-

```
<ngx-datatable
  [rows]="users" [loadingIndicator]="loading"
  (page)="onPage($event)" (sort)="onSort($event)">

  <!-- šablony sloupců -->
  <ngx-datatable-column [name]="'UPDATED_TIME' | translate"
    prop="updatedTime">
    <ng-template let-value="value" ngx-datatable-cell-template>
      {{value | date: 'short'}}
    </ng-template>
  </ngx-datatable-column>
</ngx-datatable>
```

Ukázka 3.7: Použití knihovny Ngx–datatable

kovém prostředí. Také je to samozřejmě méně pohodlné. Druhým problémem s kalendářem přímo v prohlížeči je nekonzistentní vzhled se zbytkem systému, který je řízen designem Material. Je tedy třeba vybrat jiné řešení.

Psát celou komponentu kalendáře by bylo relativně náročné, proto je vhodné použít již existující knihovnu. Ideální by bylo využít přímo část knihovny Angular Material, bohužel v době implementace této práce ještě nebyla komponenta Datepicker dostupná, její vydání je plánované až na další měsíce. Byla tedy vybrána knihovna Ic Datepicker, která respektuje Material Design a zároveň nabízí přehledné programové rozhraní. Výhodou je i kód napsaný přímo v jazyce TypeScript. S největší pravděpodobností se však i přes kvality této knihovny jedná o dočasnou variantu. Knihovny tohoto typu se často stanou nepodporovanými ve chvíli, kdy je k dispozici oficiální řešení, kterým je, respektive bude, komponenta Datepicker v rámci Angular Material. Ve webové aplikace je výběr data použit k nastavení předpokládaného data vyřešení problému.

Zpracování stránkování *Parse–link–header*

Knihovna parse–link–header slouží, jak je dle názvu patrné, k parsování HTTP hlavičky Link [60]. Z formátu definovaného navrženým standardem RFC 5988 [61], který sice vyhovuje standardu HTTP, ale méně programovacím jazykům, převádí hlavičku do objektu jazyka JavaScript. Změnu formátu lze vidět na ukázce 3.8 a 3.9.

Hlavním důvodem k použití Link hlaviček je možnost informovat klienta o počtu objektů a stránek vyhovujících volání API a také slouží ke snadnější navigaci klientů nevyužívajících grafické rozhraní. Využití Link hlaviček také podporuje princip HATEOAS (Hypermedia as the engine of application state) – odkaz je unikátním identifikátorem zdroje, přechod na zdroj skrze odkaz je

přechodem mezi stavy, podporuje bezstavovost rozhraní. Tento přístup má několik výhod proti využití Atom Links či obdobných formátů založených na odpovědi v „obálce“. Celé API nemusí zabalovat odpovědi do obálek, to přináší menší velikost a vyšší přehlednost odpovědí. Navíc je možné zjistit tuto hlavičku pouze pomocí HTTP metody HEAD. [62]

```
<http://127.0.0.1:8082/users?page=1&size=20>; rel="next",
<http://127.0.0.1:8082/users?page=2&size=20>; rel="last"
```

Ukázka 3.8: Hlavička Link v HTTP

```
{
    next: {
        page: "1",
        size: "20",
        rel: "next",
        url: "http://127.0.0.1:8082/users?page=1&size=20"
    },
    last: {
        page: "2",
        size: "20",
        rel: "last",
        url: "http://127.0.0.1:8082/users?page=2&size=20"
    }
}
```

Ukázka 3.9: Hlavička Link jako objekt v jazyce JavaScript

Zpracování asynchronních volání *RxJS*

Ve webové aplikaci se setkáváme s nutností použít asynchronní volání, typicky pro zpracování požadavků na aplikační server. Z toho důvodu je třeba zajistit způsob pro snadné zajištění jejich zpracování. [63]

1. **Callbacky** Argumentem volání funkce požadavku na server je funkce, která má být zavolána po odpovědi serveru. Funkce akceptující funkci v argumentu se nazývají funkce vyššího řádu. Tento způsob má nevýhodu v nepřehlednosti kódu, snadno způsobí tzv. špagety kód, navíc nelze použít `return` a `throw` v původním kontextu. Na druhou stranu je snadné je používat, mají plnou podporu prohlížečů a nejsou výpočetně náročné.

2. **Async modul** Poskytuje užitečné funkce pro zpracování callbacků. Zvyšuje čitelnost, ale stále zachovává nevýhody callbacků.

3. **Promise** Zpracování pomocí Promise je oproti callbackům novější, specifikace jsou pouze několik let staré. Umožňují řetězení funkcí pro zpracování odpovědi pomocí `then` a zpracování chyb pomocí `catch`. Přístup není podporován všemi prohlížeči. Je třeba použít polyfill, který zajistí podporu pro prohlížeče, které rozhraní neimplementují.

4. **Generátory** Tento způsob nepřináší v naší situaci zásadní změnu oproti Promise přístupu, nebude dále rozváděn.

5. **Await/async** Umožňuje využití klíčového slova `await` pro vyčkání na odpověď. Je nejmenším odklonem od běžného programování se synchronním kódem. Jedná se ovšem o velmi novou funkcionalitu v rámci jazyka, není podporována v rámci prohlížečů, kód musí být překompilován.

6. **RxJS** Knihovna RxJS [64] prosazuje funkcionální přístup k práci s událostmi. Nabízí mnoho funkcí, od pouhého čekání na odpověď, po transformaci kolekcí, opakování požadavků a tak dále. Výhodou je implementace kompletně v jazyce TypeScript. Její hlavní výhodou je však integrace přímo v rámci frameworku Angular. Přímo s ní pracují knihovny Http a Forms. Lze ji použít k načtení dat z API, ale i k prodlevám požadavků při vyhledávání uživatelem k odlehčení požadavků na server, reakcím rozhraní na změny ve formuláři, ... Nevýhodou knihovny je její systémová náročnost, nicméně tím, že framework rozhraní přímo využívá, jejím použitím situaci nijak zásadně nezhoršíme. Z těchto důvodů byla vybrána tato knihovna pro zpracování asynchronních volání. Ukázka použití na ukázce 3.4.

Editace formátovaného textu *TinyMCE*

Pro psaní plně formátovaných e-mailů, které jsou vhodné při komunikaci se zákazníky, nenabízí standardní HTML textové pole dostatečné možnosti. Formátovaného textu lze pohodlně docílit použitím jedné ze tří nejpopulárnějších možností jeho tvorby.

První možností je editor typu WYSIWYG („What You See Is What You Get“, v překladu „Co vidíš, to dostaneš“). Ten nabízí formátování známé z běžných kancelářských aplikací typu MS Word, dodatečné prvky volitelné tlačítky, obvykle i alespoň elementární kontrolu pravopisu. Pokročilejším uživatelům je umožněn náhled HTML zdrojového kódu.

Druhou možností je využití značkovacího jazyka typu Markdown či Wiki markup. Neklade takový důraz na skripty na pozadí textového pole, syntaxe je relativně snadná, postačí naučit se jednotky prvků pro vytváření formátovaného textu. Pro ne zcela pohodlné, ale přesto funkční zadávání textu, postačí

standardní HTML `<textarea>`. Pro větší kontrolu nad zadaným textem je ovšem třeba umožnit i náhled výsledného zformátovaného textu. Tato varianta je použita například k editaci příspěvků na portálech Wikipedia, GitHub a dalších. Problémem je nutnost znalosti daného jazyka. Současná situace, kdy jsou využívány jazyky s různou syntaxí, navíc pro běžného uživatele neznámé, není vhodná pro výběr zadávání textu tímto způsobem.

Třetí možností je využití editoru typu WYSIWYM („What You See Is What You Mean“, v překladu „Vidíš, co máš na mysli“). Cílem této metody je zaměřit se na sémantiku, strukturu a význam dokumentu. WYSIWYM editor je podobný editoru WYSIWYG, ale nezobrazuje výslednou strukturu, ale jednotlivé části typu nadpis, číslovaný seznam, odstavec. Tato varianta není příliš používaná, existuje málo knihoven, které tento model implementují a není tak snadno uchopitelná jako WYSIWYG.

Ideálním editorem pro webovou aplikaci určenou široké škále uživatelů byl zvolen WYSIWYG. Uživatelům nabízí známé prostředí k editaci textů a s textem je zároveň tvořena okamžitě viditelná struktura. Z široké škály dostupných editorů byl zvolen TinyMCE v podobě zdarma dostupné knihovny. Přestože nenabízí konkrétní typy pro jazyk TypeScript, ani přímou integraci s frameworkem Angular, je použití velmi jednoduché. Pro znovupoužitelnost v rámci celé aplikace byla vytvořena Angular komponenta, kterou je možné použít v rámci šablony stránky pomocí selektoru `<tiny-mce>`. Pomocí vstupního parametru `elementId` přiřazujeme editoru unikátní identifikátor v rámci stránky. Důvodem je identifikace v rámci DOM a také limit existence pouze jednoho editoru s daným identifikátorem v rámci jedné stránky. Výstupním parametrem onEditorKeyUp lze zaregistrovat funkci, která bude volána po změně textu uvnitř editoru.

```
<tiny-mce [elementId]="'commentInput'"
          (onEditorKeyUp)="onTextChanged($event)"></tiny-mce>
```

Ukázka 3.10: Použití komponenty TinyMCE

3.3 Nasazení

Webový server lze spustit jak pro testování, tak pro nasazení skrze `jar` balíček, který obsahuje vestavěný server Tomcat. Další možností je spuštění pomocí systému gradle, který před spuštěním kompiluje zdrojové soubory. Pro nasazení na serveru je však nejčastěji použit samostatný servletový server. K tomu stačí definovat gradle úkol `war`. Ten je následně předložen servletovému serveru, který ho zpřístupní. Na serveru je třeba mít nainstalovaný Java Runtime Environment a PostgreSQL. Tabulky databáze jsou vytvořeny Hibernate frameworkem při prvním spuštění aplikačního serveru.

Ke spuštění webové aplikace je v testovacím prostředí použit vestavěný server Angular CLI + Webpack s automatickou aktualizací. Ke spuštění i kompilaci pomocí tohoto způsobu je třeba mít nainstalovaný správce balíčků npm. Pro produkční prostředí je možné použít v podstatě jakýkoliv webový server. Je možné nasadit ji na server spolu s předkreslením na serveru, podporovaný je například server Node.js. Pro účely tohoto projektu je ideální nasazení na server Tomcat, který následně pracuje jak s API, tak s webovou aplikací. Před nasazením je třeba spustit předzpracování, které provede kompilaci jazyka TypeScript do ECMAScript5 podporovaného prohlížeči. Poté provede minifikaci (složení balíčků do jednoho souboru) a uglifikaci (znepřehlednění kódu, přejmenování proměnných, odstranění komentářů). Je možné použít dva způsoby kompilace, JIT (Just-in-time) a AOT (Ahead-of-time). JIT je vhodný zejména pro testovací účely. Počáteční kompilace je rychlejší, ale každý klient musí provést náročnější kompilaci na vlastním stroji. AOT na druhou stranu déle provádí kompilaci, nicméně ta je provedena jen jednou na vývojářském stroji. Navíc zachytí více chyb kódu již v průběhu kompilace. Následně již klient aplikaci kompilovat nemusí, aplikace je ve výsledku rychlejší. K tomu přispívá i obvykle menší výsledná velikost aplikace, docílená právě absencí kompilátoru a také strategií tzv. tree shakingu. Ten prochází reálně používané závislosti a ostatní neumístí do výsledného balíčku. Ověření rozdílů mezi JIT a AOT se nachází v sekci 4.3.3. Pro spuštění zkompilované aplikace již stačí použít běžně dostupný webový prohlížeč.

3.4 Současný stav a budoucí vývoj

Současný stav systému odpovídá počáteční vizi projektu. Jedná se o komplexní systém složený z aplikačního serveru a webové aplikace. Hlavním přínosem je dle mého názoru aplikační server s datovým modelem, který s drobnými úpravami může být použit k projektům podobného typu, případně k dalším rozšířením. Webová aplikace lehce utrpěla tím, že se jedná o opravdu novou technologii. Vývoji webové aplikaci z toho důvodu byly kladeny větší překážky než vývoji aplikačního serveru na zavedené, v korporátních systémech užívané, technologii. Problémy nevedly nutně ke zhoršení aplikace jako takové, ale zpomalovaly tempo vývoje. Způsobené byly několika vlivy. Jedním z nich je často chybějící, případně neúplná, dokumentace. Použití knihovny následně závisí na procházení zdrojového kódu, případně na odhadech chování dle názvů tříd/metod. Využití jazyka TypeScript s typovou inferencí a zejména přímým použitím typů naštěstí pomáhá. Nicméně i přesto se stávalo, že bylo využito zbytečně složité vlastní řešení na úkor snadno použitelné, ale špatně zdokumentované, funkčnosti knihovny. Dalším problémem zastihujícím celý ekosystém JavaScriptových knihoven je jejich neuvěřitelně rychlý vývoj. To může s aktualizací přinést dříve neobsažené chyby, případně nutnost změny práce s knihovnou. Zejména při zásadních změnách rozhraní neumožňujících dále

používat knihovnu stejným stylem, případně změnách doporučujících změnu v používání knihovny, přináší náklady na jejich udržování. Stav dokumentace se s největší pravděpodobností zlepší, posledně jmenované problémy ale budou další vývoj zastihovat i nadále, minimálně v horizontu měsíců.

Hlavní funkčnost systému nyní spočívá v příjmu e-mailů zákazníků, na základě kterých se vytvářejí tickety, případně rozšiřují již existující. Ty následně může zpracovávat zákaznická podpora, technici a další.

K výhodám webové aplikace patří její svižný chod bez rušivých přechodů viditelných u většiny webů renderovaných na serveru. Také je díky použitému frameworku přehledná a udržitelná, na rozdíl od kombinace běžného webu s asynchronními voláními. Opačnou stranou mince je ale opět náročnější vývoj. Běžná webová aplikace, kde spolu komunikují komponenty databáze – aplikační server, v rámci kterého je komunikace mezi službou (modelem), controllerem a view mnohem přímočařejší. V našem případě používáme model databáze – API – webový klient, uvnitř kterého komunikuje opět služba – controller, ale také HTTP pro přenos dat a dále služba – komponenta – view na webovém klientu. Takový přístup je i v takto zjednodušeně popsané formě zřetelně náročnější na vývoj. Je třeba definovat reprezentaci přenášených dat, metody pro přenos v controlleru serveru, služby pro příjem dat a následné zpracování na webovém klientu. Notná míra práce je tedy do jisté míry vykonávána dvakrát. A za toto zvýšení objemu práce dostáváme možnost využít obrovské množství klientských knihoven kompilovaných do jazyka JavaScript a uživatelský komfort v podobě emulace chování místní, tedy desktopové či mobilní aplikace. Jsem velmi přesvědčený, že v tomto projektu šlo o správnou volbou, ale u každého projektu je nutné zvážit pro a proti použití takové architektury.

Systém je vystavěn se záměrem kontinuální údržby, vylepšení a rozšíření. Vzhledem k architektuře, fungující jako běžné cloudové aplikace, by tento záměr měl být jednoduše splněn. Framework Angular, využitý při vývoji webové aplikace, je pro takové rozšiřování připraven. Modularita je jedním z jeho klíčových vlastností. Pozitivním zjištěním je, že vývojáři frameworku opravdu berou vážně kompatibilitu verzí. V průběhu vývoje proběhl přechod z verze 2 na verzi 4 bez sebemenších potíží, aktualizace byla záležitostí jednotek minut. Aplikační server je také vytvořen s ohledem na modularitu. Provoz na centrálním serveru a nikoliv nutnost aktualizovat verzi zvlášť všem klientům také podporuje kontinuální vývoj. Nad současnými daty je možné začít zpracovávat statistiky, kteří zákazníci často reklamují, které produkty jsou často reklamované a podobně. Také je záměr vytvořit systém notifikací, spolu s jejich nastaveními. Například upozornění na uživateli přiřazený ticket, nově přijaté tickety, změna na sledovaném ticketu a další. Těmito vylepšeními se ještě zvýší přínosnost systému. Další vylepšení mohou být aplikována na základě statistik provozu systému. Ať už se jedná o možnost korektně aplikovat indexy v databázi, nebo optimalizaci přednačtených částí webové aplikace.

Je možné představit si i další rozšíření napojitelná na existující API. Ať

už doplněk pro MS Outlook, které by umožňoval vytvořit z právě otevřených e-mailů ticket, či mobilní aplikaci. Vzhledem k povaze dat, konkrétně důrazu na jejich aktuálnost, není rozsáhlá mobilní aplikace příliš žádaná. Z tohoto pohledu je dostačující aktuální responzivní webová aplikace, která by byla ještě lépe optimalizovaná pro mobilní zařízení. Mobilní aplikace by spíše mohla sloužit k rychlému zaznamenání stráveného času, například technikem řešícího reklamaci. Případně pro sběr uskutečněných hovorů se zákazníky/klienty pro účely statistik, nebo i fakturace. Také by mohly být podporovány integrace se systémy třetích stran. Jak s interními systémy zákazníků, tak se systémy dopravců. Následně by pracovník zákaznické podpory mohl zákazníkovi přímo z prostředí webové aplikace objednat svoz.

Kapitola 4

Testování

4.1 Automatizované testování

4.1.1 Integrační testování

Automatizované integrační testy RESTful API se nachází v rámci architektury tohoto systému mezi jednotkovými a end-to-end testy. Automatizované end-to-end testy nejsou použity, jelikož ve fázi prototypu webové aplikace je stále možnost zásadních změn rozhraní, což by znamenalo rozbití testů. Integrační testy ovšem testují rozhraní jako celek. Stabilita REST rozhraní je silně žádaná, zejména při rozdělení vývoje mezi více vývojářů či při využití API třetími stranami. Integrační testy ale zároveň dokáží testovat funkčnost, ta je samozřejmě žádaná také. Bez funkčního rozhraní, respektive bez fungujících jednotlivých zdrojů, nemohou klienti rozhraní bezproblémově využívat. Tyto testy jsou také použitelné jako regresní testy. Je pomocí nich možné otestovat, zda nové změny negativně nezasáhly ostatní části systému. Tyto testy by měly být spuštěny před zasláním nových změn do systému správy verzí (anglicky Verson Control System – VCS; v tomto projektu git). [65]

Integrační test předpokládá sestavení celé aplikace. S pomocí pomocných tříd v rámci frameworku je spuštěn celý aplikační server a databázové transakce jsou nastaveny na automatické navrácení po proběhnuté metodě. Příprava před všemi testy zahrnuje žádost o uživatelův token, který je používán k zabezpečení požadavků. Testovaný uživatel, se všemi systémovými oprávněními, je předem připraven v datech k testování. Následně jsou zasílány požadavky jako při běžném používání API. K tomu je použita třída `MockMvc`, která využívá nahrazení reálné implementace Java Servletu za vlastní. Nejedná se tedy o reálně HTTP požadavky, ale chování je shodné. Na objektu typu `MockMvc` je zavolán požadavek na server se zvolenými parametry a server ho zpracuje shodně, jako by šlo o požadavek zvenčí. Je s ním tedy možné testovat pouze na vývojářském stroji. Pro testování vzdáleného serveru by bylo nutné použít instanci třídy `RestTemplate`, která umožňuje vykonávání skuteč-

ných HTTP požadavků. Chování `MockMvc` je ale pro tento typ testů vhodnější, je možné snadno vracet databázové transakce, před testem je možné nahrát testovací data a podobně.

```
CompanyContact companyObject = new CompanyContact()
        .setName("Company");
String company = this.mapper
        .writerWithView(CompanyView.SentInfo.class)
        .writeValueAsString(companyObject);
this.mockMvc.perform(post("/clients/{clientId}/companies",
        applicationClient.getId())
        .contentType(MediaType.APPLICATION_JSON)
        .content(company))
        .andDo(this.documentationHandler
            .document(
                requestFields(fieldWithPath("name")
                    .description("Name of the company")),
                pathParameters(parameterWithName("clientId")
                    .description("Application client's id"))))
        .andExpect(status().isCreated())
        .andExpect(jsonPath("$.name", is(single.getName())));
```

Ukázka 4.1: Příklad integračního testu

K ukládání testovacích dat slouží databáze H2 [66]. Ta umožňuje uchování databáze v paměti, testy poté probíhají rychleji než s plnohodnotnou databází. Před spuštěním testů probíhá inicializace databáze základními systémovými daty, které jsou potřebné i k běhu aplikace v běžném režimu. Jedná se například o základní systémové role, moduly, stavy a typy ticketů. K těmto nezbytným datům je přidán testovací uživatel a klientská firma systému. Všechny testy rozšiřují třídu `DocumentedTestBase` s potřebnou konfigurací testů. Třída je označena anotací `@ActiveProfiles("test")`, což umožňuje načtení nastavení testovací konfigurace uložené v souboru *application-test.properties*. Ten obsahuje například konfiguraci zmíněné databáze H2 namísto v běhu používané databáze PostgreSQL. Použit je kompatibilní mód, který se snaží kopírovat chování databáze PostgreSQL. Dále je vykonán požadavek na server, který umožní stažení bearer tokenu používaného v následujících požadavcích. Další testy k inicializaci dat využívají aplikační repozitáře, do kterých vkládají potřebná data. Je snadnější data následně testovat, není třeba je nejdříve získat z databáze pro možnost porovnávání. Testy jsou také lépe udržitelné při změnách modelu, jelikož zpětně nekompatibilní změna modelu vyvolá chybu již při kompilaci, nikoliv za běhu. Následně jsou volány metody serveru a testovány odpovědi. Zjednodušený integrační test je možné zhlédnout na ukázce 4.1. Nejprve je vytvořen objekt k odeslání na server. Následně je připraven k ode-

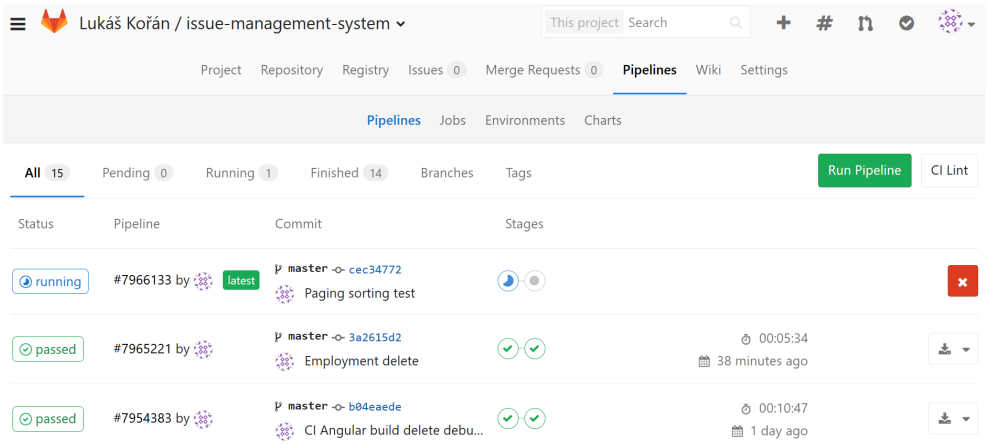
slání, tzn. serializován do formátu JSON. Poté je odeslán metodou POST na server, v těle požadavku je přenesen právě vytvořený JSON objekt. Navíc je použita knihovna Spring REST Docs [67] k dokumentaci požadavků a jejich odpovědí. V ukázce je zdokumentováno pole jméno v těle požadavku a parametr cesty obsahující id klientské firmy, u které je firemní kontakt zakládán. Metodou `andExpect()` jsou prováděny aserce, tedy ověření, zda byly obdrženy očekávané údaje. Knihovna umožňuje velmi kompaktní notaci. Na ukázce je ověřován HTTP stavový kód, očekáván je kód 201 (Created). Výraz v `jsonPath()` ověřuje přítomnost a zároveň obsah vlastnosti name proti původně vytvořenému objektu, respektive jeho vlastnosti name.

4.1.2 Průběžná integrace

Průběžná integrace (anglicky Continuous Integration – CI) je souborem několika navazujících procesů. Prvním je potvrzení změn a zaslání kódu do systému řízení změn vývojářem. Další procesy již probíhají na integračním serveru. Nejprve proběhne kompilace, následně testy a nakonec nasazení. Pro tuto práci není využito automatické nasazení. V případě problému v některém ze zmíněných bodů je chyba oznámena vývojářům, způsob záleží na konkrétním řešení. Tento projekt k CI používá vestavěnou podporu na portálu GitLab. Jak bylo popsáno u integračních testů, bylo by vhodné, aby je vývojáři spustili před odesláním do VCS. Zavedením průběžné integrace je tento důležitý článek procesu vývoje automatizován. Dále již nezáleží na tom, zda si programátor vzpomněl na spuštění testů, ani na tom, že testy mohou běžet příliš dlouho pro pohodlné používání na vývojářském počítači. Chyba v testech či kompilaci je oznámena e-mailem. Detaily a historii úloh je možné prohlížet ve webovém prostředí. Také je možné stáhnout výsledek kompilace. Další výhodou užití integračního serveru je ověření funkčnosti na dalším OS (systém byl vyvíjen na OS Windows, integrační server používá OS Linux). Části jednotlivých fází probíhají paralelně, v tomto případě je zároveň kompilováno API i webový klient. Následně probíhají integrační testy webového API. Ukázku webového prostředí viz na obrázku 4.1. Nastavení se nachází v souboru *.gitlab-ci.yml*, specifikuje používané Docker obrazy systému, prováděné skripty, které soubory mají být exportovány po kompilaci, ... [68]

4.2 Heuristické vyhodnocení

Na prototypu webové aplikace bylo provedeno heuristické vyhodnocení. Jedná se o běžně používané Nielsenovy heuristiky [69], užívané k návrhu a testování rozhraní. České názvy bodů pochází z [70]. Bylo snahou řídit se těmito heuristikami již v průběhu návrhu a implementace rozhraní, vyhodnocení slouží zejména jako kontrola splnění daných bodů.



Obrázek 4.1: Ukázka webového rozhraní Gitlab CI [68]

1. **Viditelnost stavu systému** *Je uživatel informován o právě probíhajících akcích vhodnou zpětnou vazbou v rozumném čase?*

   Pro dlouho trvající akce je zobrazen ukazatel průběhu odpovídající Material vzhledu. Vzhledem k příliš rychlým reakcím serveru je pro testování použito simulované zpomalování připojení pomocí Chrome DevTools. Uživatel je informován o stavu akce po jejím vykonání prostřednictvím takzvané toast notifikace. V případě korektních stavů akcí na seznamech položek se nová položka přidá do seznamu, v případě chyby přidána není a uživateli je oznámena chyba.

2. **Shoda mezi systémem a realitou** *Zachovává systém názvy, popisky, fráze běžně užívané jeho uživateli? Objevují se informace v přirozeném a logickém pořadí?*

   Jsou využity ikony korespondující s danou akcí. Systém nevyužívá specializovaných názvů. Tabulky a historie akcí jsou ve výchozím zobrazení seřazeny od nejnovějších (poslední změny).

3. **Minimální zodpovědnost** *Je zřetelně označený odchod z obrazovek, do kterých se uživatel dostal nechtěně? Jsou podporovány funkce vrátit a znovu vykonat?*

   Uživatel je varován před provedením nevratné akce typu smazání. Z podstaty návrhu systému uživatel není varován před odchodem z editace pomocí navigace zpět. K uložení je určeno speciální tlačítko viditelné v každém stavu aplikace. Jedná se o tzv. „Floating action button“, které je pro takové akce definované v rámci Material Design doporučení. Potvrzují se akce typu editace osoby, firmy. V rámci práce s tickety změny potvrzovány nejsou. Jedná se o designové rozhodnutí, změny jsou

ukládány automaticky po časové prodlevě určené například pro změnu rozhodnutí uživatele. Automatické potvrzení neklade stres na uživatele v podobě nutnosti potvrdit změny a zrychluje práci s tickety. Změny navíc lze vyjma přidaných komentářů a příloh změnit na jinou hodnotu, což v podstatě změnu ruší.

4. **Shoda s použitou platformou a obecnými standardy** *Nejsou použita různá slova, ikony, akce k totožným výsledkům? Jsou dodrženy konvence platformy?*

   Webová aplikace se řídí doporučeními Material Design. V důsledku se nejvíce blíží vzhledu aplikace pro Android. Webové aplikace na platformě PC nemají obecná doporučení pro vzhled, proto jsou doporučení Material společně s přizpůsobeními pro webový prohlížeč dostačující. Určité prvky, například ikony a fonty neodpovídají platformě iOS, nicméně doporučení dané platformy se vážou zejména na instalované aplikace, nikoliv webové aplikace v rámci prohlížeče.

5. **Prevence chyb** *Brání návrh chybám uživatele?*

   Povinné položky formulářů jsou zvýrazněny dle doporučení Material. V rámci možností je zabráněno nesprávným vstupům. Náročné validace typu kontroly zadání PSČ pro různé státy v prototypu implementovány nejsou.

6. **Kouknu a vidím** *Jsou viditelné důležité možnosti, objekty a akce? Nemusí si uživatel pamatovat informace z jiných obrazovek?*

   Menu vždy zobrazuje aktuální sekci. Vzhledem k pouhým dvěma úrovním zanoření není zobrazena podrobnější pozice ve stromové struktuře.

7. **Flexibilita a efektivita** *Jsou dostupné možnosti k urychlení práce expertním uživatelům?*

   Rozdíl je pouze pro různé typy uživatelů, zejména administrátory, kteří mají přístup do více sekcí. Expertní funkce/zjednodušení ale nabízeny nejsou.

8. **Minimalita** *Neobsahují obrazovky/dialogy nepodstatné či zřídka užívané informace?*

   Splněno, šlo o jeden z hlavních záměrů při vytváření webové aplikace.

9. **Smysluplné chybové hlášky** *Užívají chybové hlášky běžný jazyk k indikaci problému a jeho možného řešení?*

   Chybové hlášky jsou pro přehlednost ve formulářích zobrazeny u polí obsahujících chybu, se vzhledem dle doporučení Material design.

10. **Nápověda a dokumentace** *Pokud je to nutné, obsahuje systém nápovědu a dokumentaci?*

    Složitější ikony zobrazují kontextovou nápovědu po najetí kurzorem. Globální nápověda pro celý systém není v rámci prototypu implementována.

Heuristické vyhodnocení nezaznamenalo zásadní problémy uživatelského rozhraní. Důležité je ovšem na výše jmenované body dbát i při dalším rozšiřování systému, případně průchod rozhraním opakovat.

4.3 Testování programátorem

4.3.1 Testování aplikací

V průběhu vývoje bylo využito několik typů testování programátorem jako doplněk k automatizovaným testům. Prováděny byly z důvodu jejich snadnějšího provedení oproti pokrytí automatizovanými testy, případně kvůli povaze testovaného subjektu. K testování komunikačního rozhraní byl použit nástroj Postman [71], který umožňuje rychlé otestování právě implementované funkcionality. Používá HTTP požadavky na server. Odpovědi serveru mohou být následně prohlédnuty programátorem ke zjištění, zda odpověď odpovídá kýženému výsledku. Případně mohou být odpovědi serveru testovány na očekávaný výsledek pomocí vestavěného testovacího rozhraní, používajícího jazyk JavaScript. Další testování programátorem se zaměřovalo na zdrojový kód. Přezkoumání kódu dokáže odhalit například opomenuté nepoužívané kusy kódu, chyby ve stylu kódu s pomocí nápověd IDE, ať už se jedná o chyby stylu kódu webové aplikace testované nástrojem TSLint, nebo přímo vývojovým prostředím doporučené změny kódu směřující k použití funkcionálních vlastností jazyka Java 8. Automatické testy uživatelského rozhraní, například frameworkem doporučeným nástrojem Protractor [72], by nebyly příliš stabilní při očekávaném doladění rozhraní pro produkční prostředí, proto bylo místo nich využito testování programátorem. Testován byl průchod scénáři uvedenými v sekci *Případy užití*. Jeho průchodem bylo také ověřeno splnění funkčních požadavků, vyjma nedostupných uživatelských skupin a propojení ticketů v rámci webové aplikace. Nebyl vytvořen samostatný dashboard, ale zjednodušené statistiky a filtry přímo v seznamu ticketů. Anonymní přístup do systému je umožněn skrze povolené vytváření ticketů ve webovém API.

4.3.2 Testování komprese

K ověření funkčnosti a užitečnosti komprese gzip při komunikaci mezi klientem a serverem byl proveden jednoduchý test. Bylo zvoleno několik zdrojů a byly otestovány se zapnutou a vypnutou kompresí na serveru. K testu byl použit prototyp webového klienta, k ověření velikosti byl použit nástroj

Tabulka 4.1: Srovnání velikosti odpovědí serveru

Zdroj \ Komprese	Žádná	Gzip
`/clients`	933 B	778 B
`/roles/master`	527 B	603 B
`/users/{userId}`	3 KB	1,1 KB
`/modules/users/{userId}`	4,1 KB	1 KB
`/tickets/{ticketId}/activities`	3,2 KB	1 KB

Tabulka 4.2: Srovnání velikosti webové aplikace

Soubor \ Typ kompilace	JIT	AOT	JIT optim.	AOT optim.
Vendor	496 KB	406 KB	444 KB	381 KB
Main	51,6 KB	164 KB	51,6 KB	164 KB

Chrome DevTools [73]. Z tabulky 4.1 vyplývá, že gzip komprese je opravdu funkční a navíc užitečná. Ve všech odpovědích kromě jedné je komprimovaná verze znatelně menší. Jedna odpověď je menší ve variantě bez komprese. To je způsobeno velmi malou velikostí těla obsahujícího pouze jednu hodnotu. V tu chvíli převáží velikost hlaviček kvůli navíc obsažené hlavičce `Content-Encoding:gzip`.

4.3.3 Testování kompilace

K ověření hypotézy o vhodnosti využití AOT místo JIT kompilace, byl také proveden test. Webová aplikace byla zkompilována pomocí Angular CLI s možností AOT, respektive JIT. Při kompilaci bez určení pro produkční prostředí má celá aplikace velikost 16 MB, protože neprochází potřebným předzpracováním a obsahuje soubory s mapováním pro účely vývoje. Při kompilaci s korektním určením pro produkční prostředí a možností JIT má celý aplikační balík velikost 4,1 MB, při volbě AOT 4,59 MB. Stěžejní jsou ovšem data reálně přenášená klientovi. Souhrn klíčových stažených souborů, ve kterých dochází ke změně velikosti, je k nahlédnutí v tabulce 4.2. Velikosti byly opět zjištěny nástrojem Chrome DevTools. Většina přenášených souborů je shodná, například styly, polyfills pro doplnění funkcí do nepodporovaných prohlížečů, ... Rozdíl je ve velikosti souboru *vendor*. U JIT kompilace je větší. Jedná se právě o kompilátor, který u AOT verze není třeba. Soubor *main* je ovšem větší u verze s AOT kompilací. S největší pravděpodobností se jedná o zkompilované verze souborů, které JIT překládá až u klienta. Ve výsledku je tedy verze po JIT kompilaci o něco menší, než aplikace zkompilovaná metodou AOT. Ovšem i přesto nadále zůstávají důvody k použití AOT kompilace. Při dalším načtení aplikace totiž server vrací prohlížeči odpovědi se stavovým kódem 304 (Not

Modified – zdroj nebyl změněn). V důsledku lehce vyšší velikost nijak nevadí. Hlavním rozdílem obou verzí je totiž rychlost načtení aplikace. AOT verze pouze čeká na soubory ze serveru. V případě návratové hodnoty 304 aplikace nabíhá do jedné vteřiny. JIT kompilovaná aplikace ovšem vždy čeká na kompilaci u klienta, ta na testovacím stroji trvala několik vteřin. Na závěr byl ještě proveden pokus o optimalizaci velikosti aplikace, konkrétně optimalizace importů knihovny RxJS. Import pouze používaných operátorů uspořil další desítky kilobajtů v souboru *vendor*, jak u JIT, tak u AOT kompilace. I do budoucna je tedy nutné s každou novou rozsáhlou knihovnou optimalizovat její využití.

Závěr

Cílem této práce bylo navrhnout a implementovat systém pro práci s reklamacemi, který by umožňoval zákaznické podpoře snadno zpracovávat probíhající reklamace.

Výsledkem práce je funkční webové API a prototyp webového klienta. Aplikace umožňuje práci s reklamacemi, údaji o zákaznicích a správu systému. Využitelná je pracovníky zákaznické podpory, techniky pracujícími na opravě a správci systému. K práci s webovou aplikací nejsou kladeny vysoké nároky na uživatele. Prostředí bylo vyvíjeno v souladu s doporučeními Material Design. To by mělo zajistit příjemný vzhled a prostředí k práci, které neklade uživatelům překážky, ale provází prováděnými úkony. Hlavní funkcionalitou prototypu je právě vyřizování reklamací, které nabízí výhody v přehlednosti a snadnějším zpracováním dat oproti e-mailům.

Systém byl vyvíjen s důrazem na další rozvoj. Vzhledem ke zvoleným platformám a architektuře je možné další vývoj zajistit. Hlavní sílou systému by následně byly integrace na pozadí, které by automatizovaly nyní náročné procesy. Zaměstnanci zabývající se reklamacemi by však nadále byli vystaveni práci s jednoduchým rozhraním, které by sdružovalo procesy kolem reklamací.

Jak vyplývalo již ze zadání samotné práce, potýkáme se se systémem, který má potenciál dále se vyvíjet na základě aktuálních požadavků. Výsledek této práce dle mého názoru splňuje její cíl a je vhodným stavebním kamenem pro další vývoj. Při něm se může systém dále zdokonalovat a zvyšovat užitnou hodnotu, která v důsledku zefektivní práci podniku využívajícího tento systém.

Použité zdroje

[1] JetBrains s.r.o.: YouTrack [online]. [cit. 3.4.2017]. Dostupné z: https://www.jetbrains.com/youtrack/

[2] Zendesk, Inc.: Zendesk [online]. [cit. 3.4.2017]. Dostupné z: https://www.zendesk.com/

[3] Digita s.r.l.: Tustena CRM [online]. [cit. 3.4.2017]. Dostupné z: http://www.tustena.com/en/

[4] Digita s.r.l.: Asp.net CRM [online]. [cit. 3.4.2017]. Dostupné z: https://crm.codeplex.com/

[5] SugarCRM, Inc.: SugarCRM [online]. [cit. 3.4.2017]. Dostupné z: http://www.sugarcrm.com/

[6] Oram, C.: SugarCRM in the Next 10 Years [online]. [cit. 3.4.2017]. Dostupné z: https://community.sugarcrm.com/thread/18434

[7] Free Software Foundation, Inc.: GNU Affero General Public License, Version 3 [online]. 2007, [cit. 3.4.2017]. Dostupné z: https://www.gnu.org/licenses/agpl-3.0.html

[8] Agile CRM, Inc.: Agile CRM [online]. [cit. 3.4.2017]. Dostupné z: https://www.agilecrm.com/

[9] Mueller, J.: Understanding SOAP and REST Basics And Differences [online]. 2013, [cit. 4.4.2017]. Dostupné z: http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/

[10] JSON: The Fat-Free Alternative to XML [online]. [cit. 4.4.2017]. Dostupné z: http://www.json.org/xml.html

[11] Pivotal Software, Inc.: Spring IO platform [software]. [cit. 5.4.2017]. Dostupné z: http://platform.spring.io/platform/

[12] Maple, S.: Java Tools and Technologies Landscape Report 2016: Trends and Historical data [online]. 2016, [cit. 4.4.2017]. Dostupné z: https://zeroturnaround.com/rebellabs/java-tools-and-technologies-landscape-2016-trends/

[13] Pivotal Software, Inc.: Spring Boot [software]. [cit. 5.4.2017]. Dostupné z: http://projects.spring.io/spring-boot/

[14] Foundation, T. A. S.: Apache Tomcat [software]. [cit. 4.4.2017]. Dostupné z: https://tomcat.apache.org/

[15] Q-Success DI Gelbmann GmbH: Usage of web servers for websites [online]. 2017, [cit. 4.4.2017]. Dostupné z: https://w3techs.com/technologies/overview/web_server/all

[16] Gradle, Inc.: Gradle [software]. [cit. 5.4.2017]. Dostupné z: https://gradle.org/overview

[17] Gradle, Inc.: Maven vs Gradle: Feature Comparison Chart [online]. [cit. 4.4.2017]. Dostupné z: https://gradle.org/maven-vs-gradle

[18] Pivotal Software, Inc.: Spring Data JPA [software]. [cit. 4.4.2017]. Dostupné z: http://projects.spring.io/spring-data-jpa/

[19] Community, H.: Hibernate ORM [software]. [cit. 5.4.2017]. Dostupné z: http://hibernate.org/orm/

[20] OAuth 2.0 [online]. [cit. 5.4.2017]. Dostupné z: https://oauth.net/2/

[21] Parecki, A.: OAuth 2 Simplified [online]. [cit. 5.4.2017]. Dostupné z: https://aaronparecki.com/oauth-2-simplified/

[22] Pivotal Software, Inc.: Spring Security OAuth [software]. [cit. 5.4.2017]. Dostupné z: http://projects.spring.io/spring-security-oauth/

[23] World Wide Web Consortium (W3C): *Securing the Web*. 2015. Dostupné z: https://www.w3.org/2001/tag/doc/web-https

[24] PostgreSQL Global Development Group: PostgreSQL [software]. [cit. 5.4.2017]. Dostupné z: http://www.postgresql.org/

[25] Facebook Inc.: React [software]. [cit. 17.4.2017]. Dostupné z: https://facebook.github.io/react/

[26] Google, Inc.: AngularJS [software]. [cit. 17.4.2017]. Dostupné z: https://angularjs.org/

[27] Google, Inc.: Angular [software]. [cit. 17.4.2017]. Dostupné z: https://angular.io/

[28] Microsoft Corporation: TypeScript [online]. [cit. 17.4.2017]. Dostupné z: https://www.typescriptlang.org/

[29] Google, Inc.: Browser support [online]. [cit. 19.4.2017]. Dostupné z: https://angular.io/docs/ts/latest/guide/browser-support.html

[30] Bauer, C.; King, G.: *Java Persistence with Hibernate*. Greenwich, CT, USA: Manning Publications Co., 2006, ISBN 1932394885.

[31] Berners-Lee, T.: *Universal Resource Identifiers in WWW*. Network Working Group, 1994. Dostupné z: https://tools.ietf.org/html/rfc1630

[32] World Wide Web Consortium (W3C): *Date and Time Formats*. 1997. Dostupné z: https://www.w3.org/TR/NOTE-datetime

[33] Internet Engineering Task Force: *Hypertext Transfer Protocol – HTTP/1.1*. 1999. Dostupné z: https://tools.ietf.org/html/rfc2616

[34] Balsamiq Studios, LLC: Balsamiq [software]. [cit. 6.4.2017]. Dostupné z: https://balsamiq.com/

[35] LePage, P.: Responsive Web Design Patterns [online]. [cit. 6.4.2017]. Dostupné z: https://developers.google.com/web/fundamentals/design-and-ui/responsive/patterns

[36] Pivotal Software, Inc.: Developer Tools [online]. [cit. 17.4.2017]. Dostupné z: http://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-devtools.html

[37] Npm, Inc.: Npm [software]. [cit. 17.4.2017]. Dostupné z: https://www.npmjs.com/

[38] Google, Inc.: Angular CLI [software]. [cit. 17.4.2017]. Dostupné z: https://cli.angular.io/

[39] Webpack [software]. [cit. 17.4.2017]. Dostupné z: https://webpack.js.org/

[40] Git Community: Git [software]. [cit. 21.4.2017]. Dostupné z: http://git-scm.com/

[41] GitLab Inc.: GitLab [online]. [cit. 21.4.2017]. Dostupné z: https://gitlab.com/

[42] Pivotal Software, Inc.: The IoC container [online]. [cit. 16.4.2017]. Dostupné z: https://docs.spring.io/spring/docs/current/spring-framework-reference/html/beans.html

[43] Google, Inc.: Dependency Injection [online]. [cit. 16.4.2017]. Dostupné z: https://angular.io/docs/ts/latest/guide/dependency-injection.html

[44] Oracle Corporation: Java Persistence API [online]. [cit. 4.4.2017]. Dostupné z: http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

[45] Pivotal Software, Inc.: Spring Integration [software]. [cit. 12.4.2017]. Dostupné z: https://projects.spring.io/spring-integration/

[46] Oracle Corporation: JavaMail API [online]. [cit. 12.4.2017]. Dostupné z: https://java.net/projects/javamail/pages/Home

[47] Google, Inc.: Architecture Overview [online]. [cit. 13.4.2017]. Dostupné z: https://angular.io/docs/ts/latest/guide/architecture.html

[48] Google, Inc.: Routing & navigation [online]. [cit. 10.4.2017]. Dostupné z: https://angular.io/docs/ts/latest/guide/router.html

[49] Google, Inc.: Forms [online]. [cit. 10.4.2017]. Dostupné z: https://angular.io/docs/ts/latest/guide/forms.html

[50] Google, Inc.: Reactive Forms [online]. [cit. 10.4.2017]. Dostupné z: https://angular.io/docs/ts/latest/guide/reactive-forms.html

[51] Mozilla Developer Network: HTTP access control (CORS) [online]. [cit. 12.4.2017]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

[52] Google, Inc.: Material Design for Angular [online]. [cit. 13.4.2017]. Dostupné z: https://github.com/angular/material2

[53] Google, Inc.: Angular Animations [online]. [cit. 7.4.2017]. Dostupné z: https://angular.io/docs/ts/latest/guide/animations.html

[54] Google, Inc.: Angular Flex Layout [software]. [cit. 8.4.2017]. Dostupné z: https://github.com/angular/flex-layout

[55] Refsnes Data: CSS3 @media Rule [online]. [cit. 8.4.2017]. Dostupné z: https://www.w3schools.com/cssref/css3_pr_mediaquery.asp

[56] Google, Inc.: Internalization (i18n) [online]. [cit. 8.4.2017]. Dostupné z: https://angular.io/docs/ts/latest/cookbook/i18n.html

[57] NGX–Translate: Ngx–translate [software]. [cit. 8.4.2017]. Dostupné z: http://www.ngx-translate.com/

[58] Swimlane LLC: Ngx–datatable [software]. [cit. 8.4.2017]. Dostupné z: https://github.com/swimlane/ngx-datatable

[59] Brown, C.: Ic Datepicker [software]. [cit. 7.4.2017]. Dostupné z: https://github.com/IckleChris/ic-datepicker

[60] Lorenz, T.: Parse–link–header [software]. [cit. 7.4.2017]. Dostupné z: https://github.com/thlorenz/parse-link-header

[61] Internet Engineering Task Force: *Web Linking*. 2010. Dostupné z: https://tools.ietf.org/html/rfc5988

[62] Vitvar, T.: Web 2.0, přednáška HATEOAS, Scalability and Description. 2017, [cit. 7.4.2017].

[63] Nemeth, G.: The Evolution of Asynchronous JavaScript [online]. [cit. 7.4.2017]. Dostupné z: https://blog.risingstack.com/asynchronous-javascript/

[64] RxJS 5 [software]. [cit. 16.4.2017]. Dostupné z: https://github.com/ReactiveX/software

[65] Kaczanowski, T.: *Practical Unit Testing with JUnit and Mockito*. Cracow: Tomasz Kaczanowski, 2013, ISBN 8393489393, 978-8393489398.

[66] Müller, T.: H2 Database Engine [software]. [cit. 24.4.2017]. Dostupné z: http://www.h2database.com/html/main.html

[67] Pivotal Software, Inc.: Spring REST Docs [online]. [cit. 24.4.2017]. Dostupné z: https://projects.spring.io/spring-restdocs/

[68] Gitlab Inc.: GitLab Continuous Integration [online]. [cit. 29.4.2017]. Dostupné z: https://docs.gitlab.com/ce/ci/

[69] Nielsen, J.: 10 Usability Heuristics for User Interface Design [online]. 1995, [cit. 14.4.2017]. Dostupné z: http://www.nngroup.com/articles/ten-usability-heuristics/

[70] Žikovský, P.: Návrh uživatelských rozhraní, přednáška Testování bez uživatelů. [cit. 14.4.2017]. Dostupné z: https://edux.fit.cvut.cz/courses/MI-NUR/_media/lectures/x03-semestralka_testing_without_users.pdf

[71] Postdot Technologies, Inc.: Postman [software]. [cit. 20.4.2017]. Dostupné z: https://www.getpostman.com/

[72] Google, Inc.: Protractor [software]. [cit. 20.4.2017]. Dostupné z: http://www.protractortest.org

[73] Google, Inc.: Chrome DevTools [software]. [cit. 24.4.2017]. Dostupné z: https://developers.google.com/web/tools/chrome-devtools/

Příloha **A**

Seznam použitých zkratek

ACL Access Control List

AOT Ahead-of-time

API Application Programming Interface

CI Continuous Integration

CORS Cross-origin Resource Sharing

CRM Customer Relationship Management

DI Dependency Injection

DOM Document Object Model

GUI Graphical User Interface

HQL Hibernate Query Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IDE Integrated Development Environment

JDBC Java Database Connectivity

JIT Just-in-time

JPA Java Persistance API

JPQL Java Persistence Query Language

JSON JavaScript Object Notation

ORM Object-relational mapping

OS Operating System

REST Representational State Transfer

SOAP Simple Object Access protocol

SPA Single-page application

SQL Structured Query Language

TLS Transport Layer Security

URI Uniform Resource Identifier

URL Uniform Resource Locator

WYSIWYG What You See Is What You Get

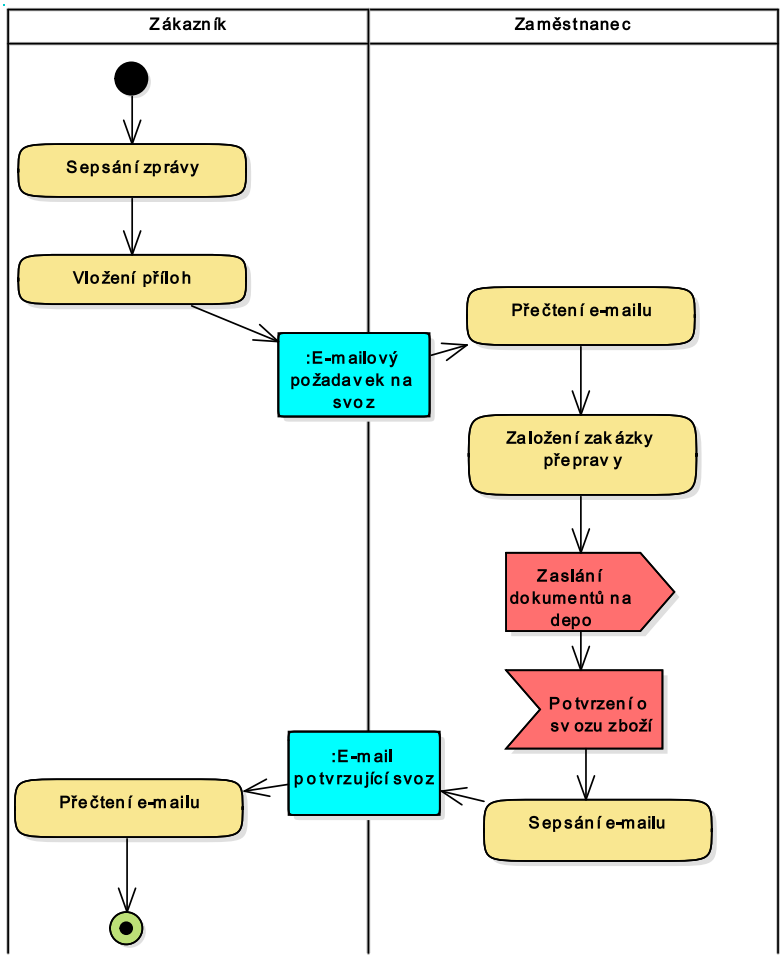
WYSIWYM What You See Is What You Mean

XHR XMLHttpRequest

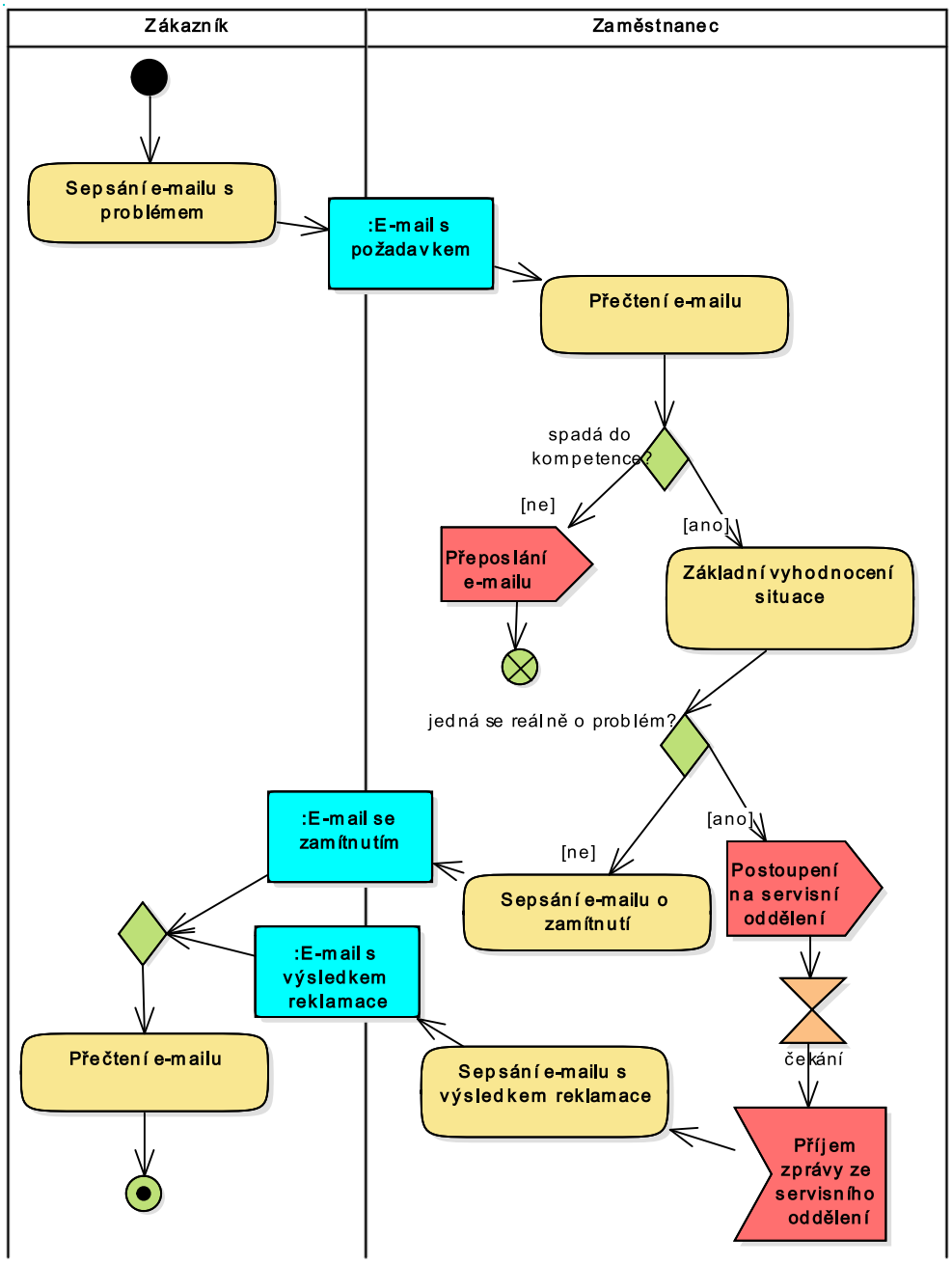
XML Extensible Markup Language

Příloha B

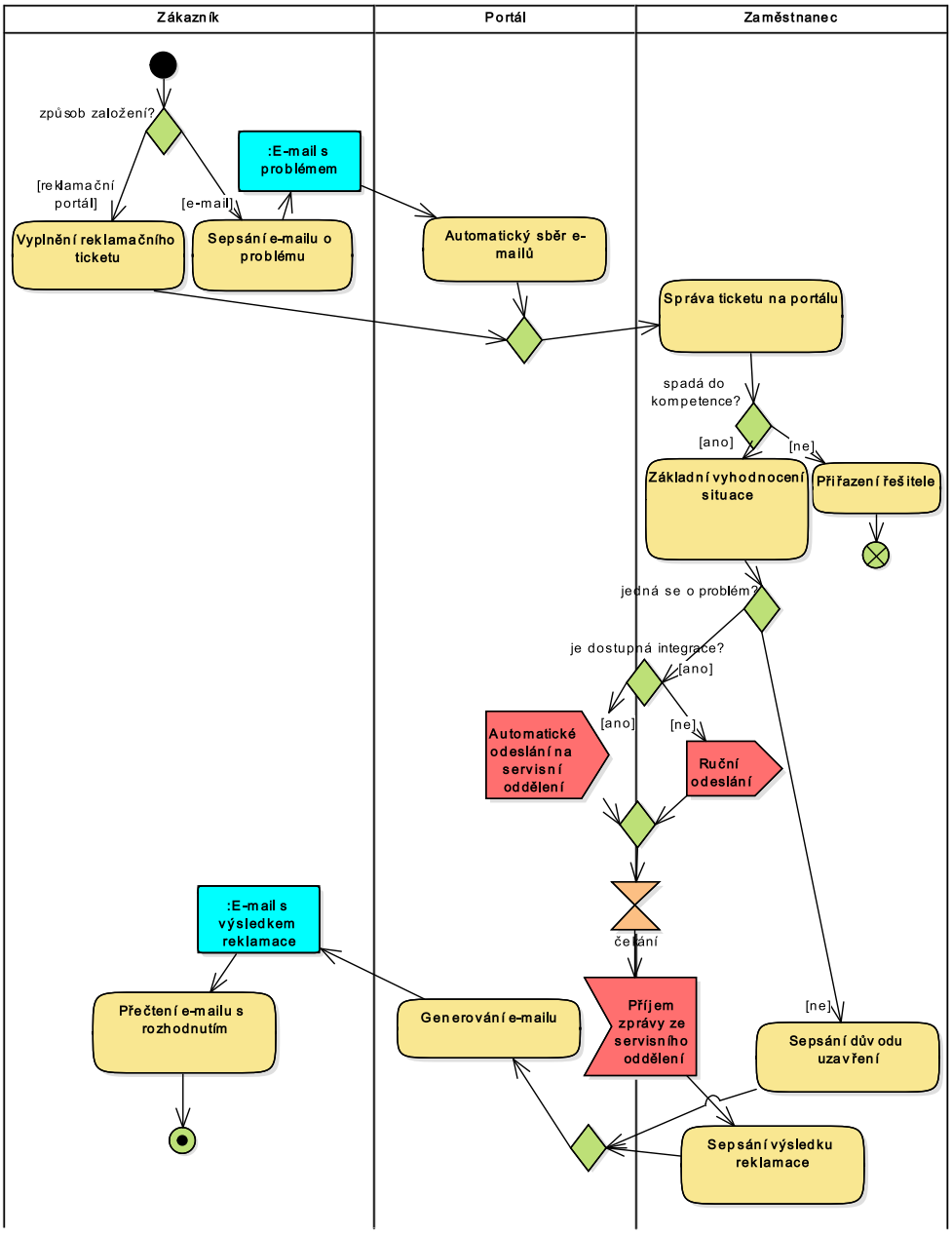
Diagramy aktivit



Obrázek B.1: Aktivity integrace se službami třetích stran – bez portálu



Obrázek B.2: Aktivity správy problému – bez portálu

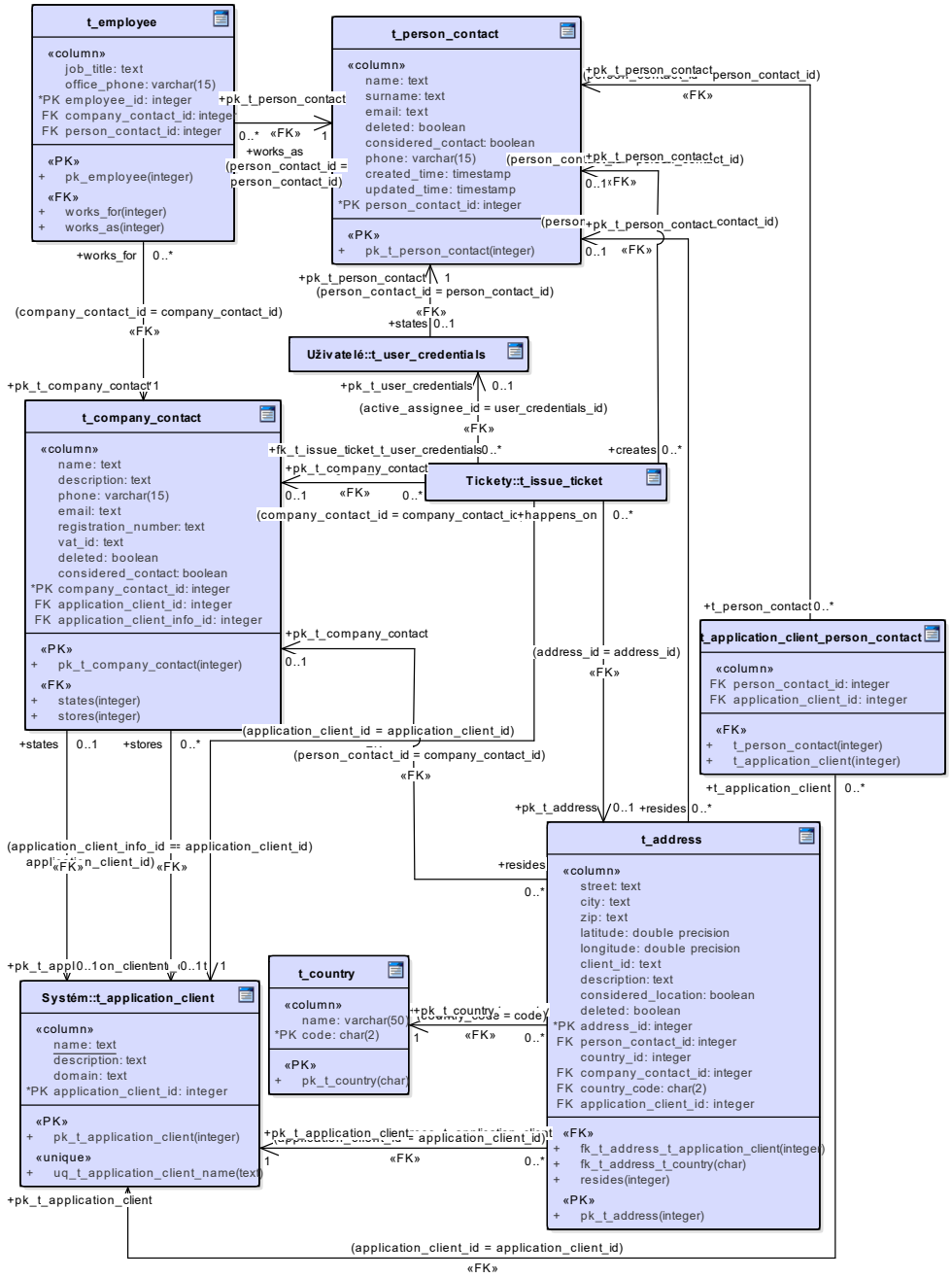


Obrázek B.3: Aktivity správy problému – s portálem

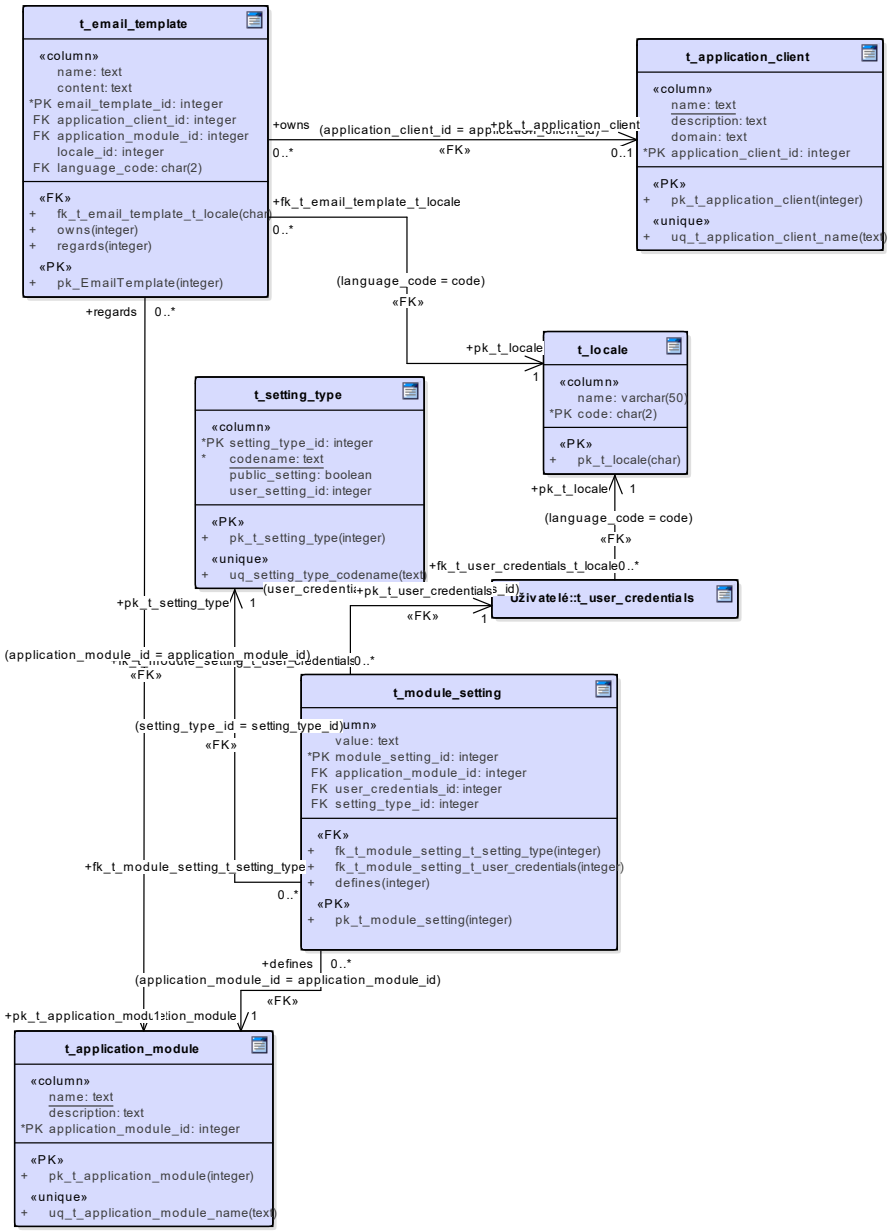
Příloha C

Databázový model

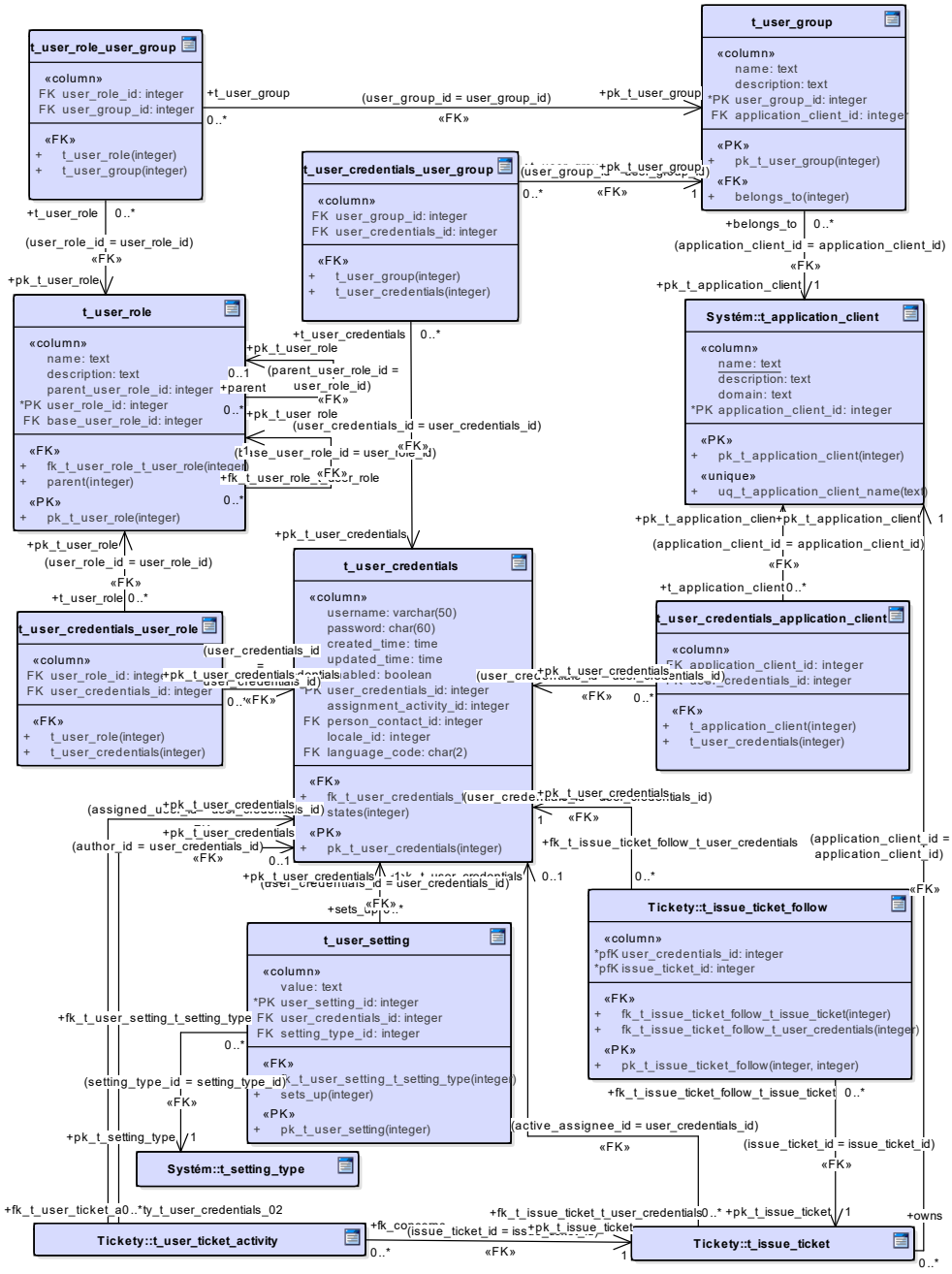
Na následujících stranách se pro úplnost nachází databázový model. I přes snahu o jeho zmenšení je opravdu rozměrný. Vzhledem k tomu je doporučeno jeho zhlédnutí buď v kompletní formě na přiloženém CD nebo prostřednictvím mapování ve zdrojových kódech, případně viz sekci *Doménový model*, ze které tento model vychází.



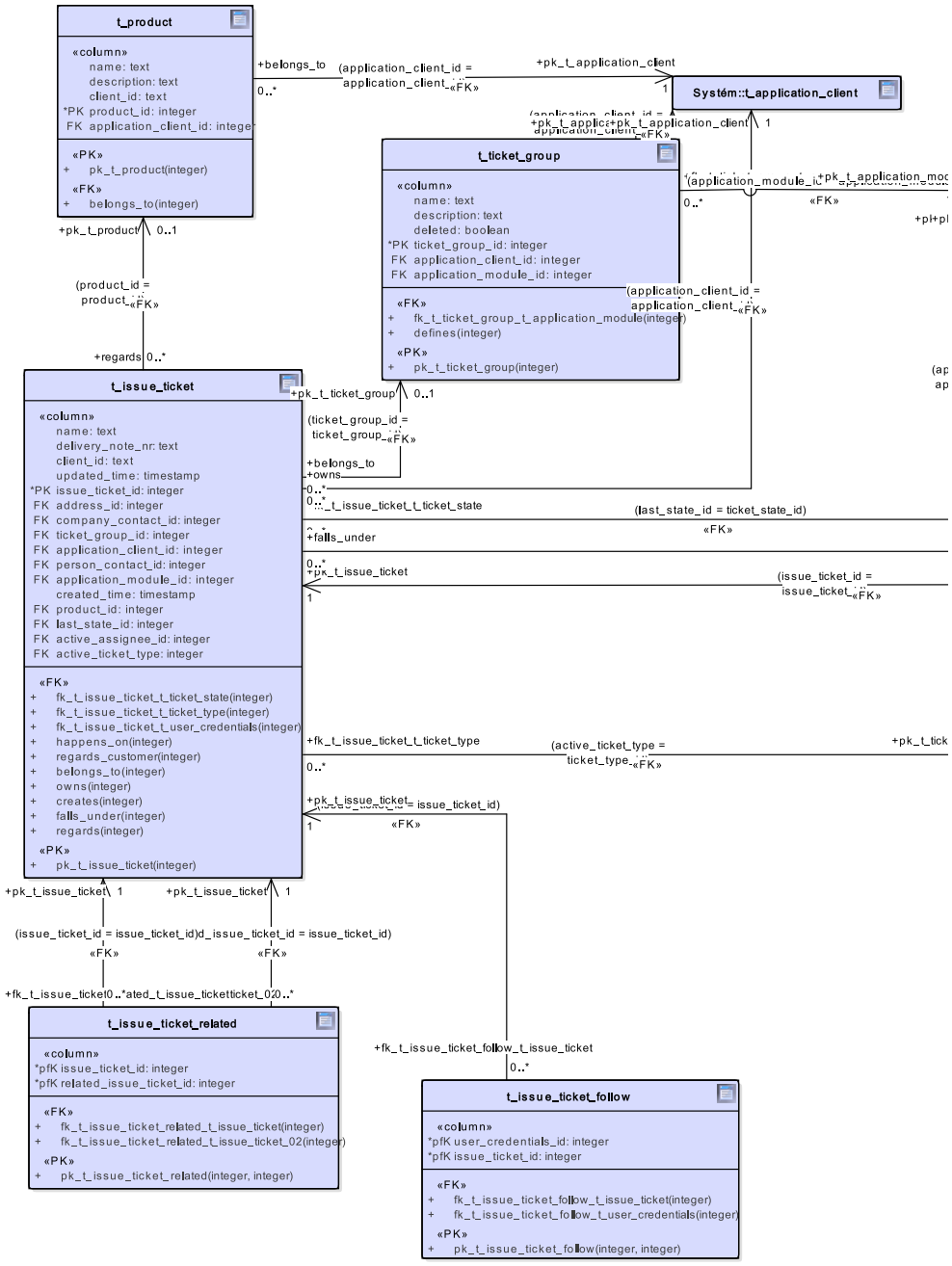
Obrázek C.1: Datový model – kontakty



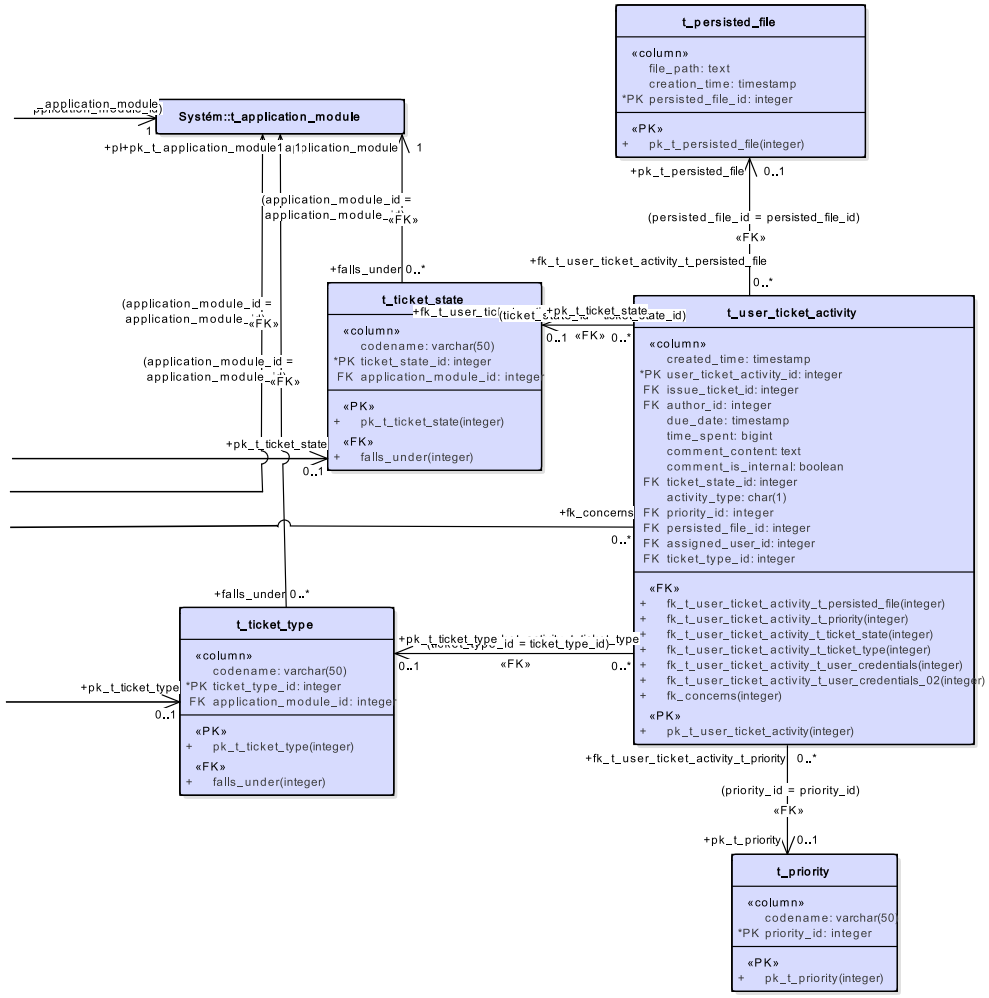
Obrázek C.2: Datový model – systém



Obrázek C.3: Datový model – uživatelé



Obrázek C.4: Datový model – tickety, 1. část

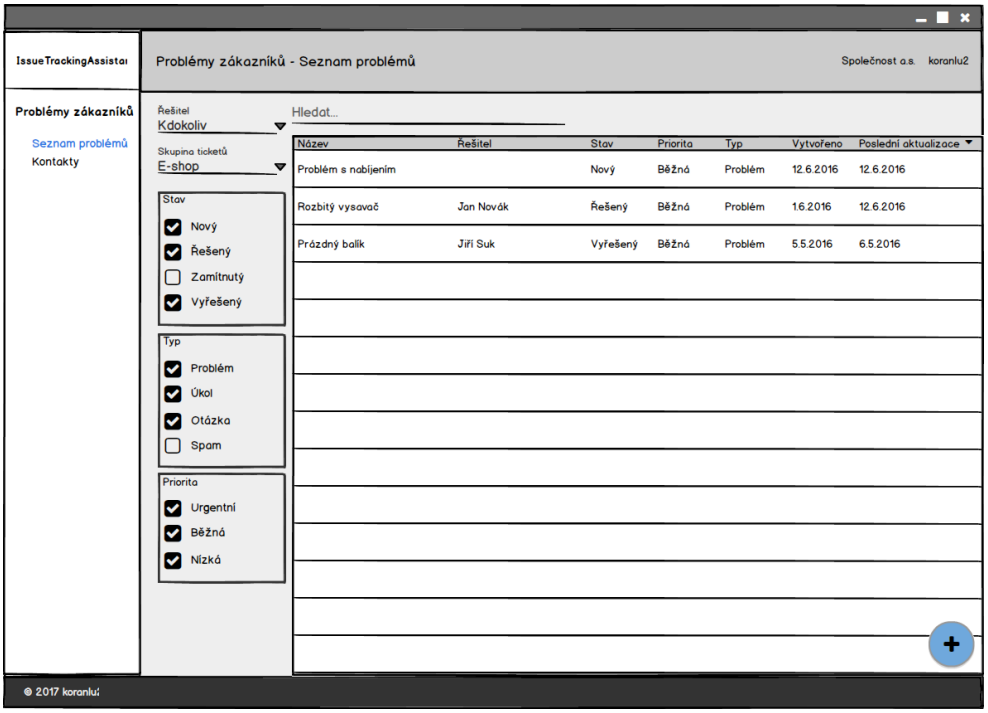


Obrázek C.5: Datový model – tickety, 2. část

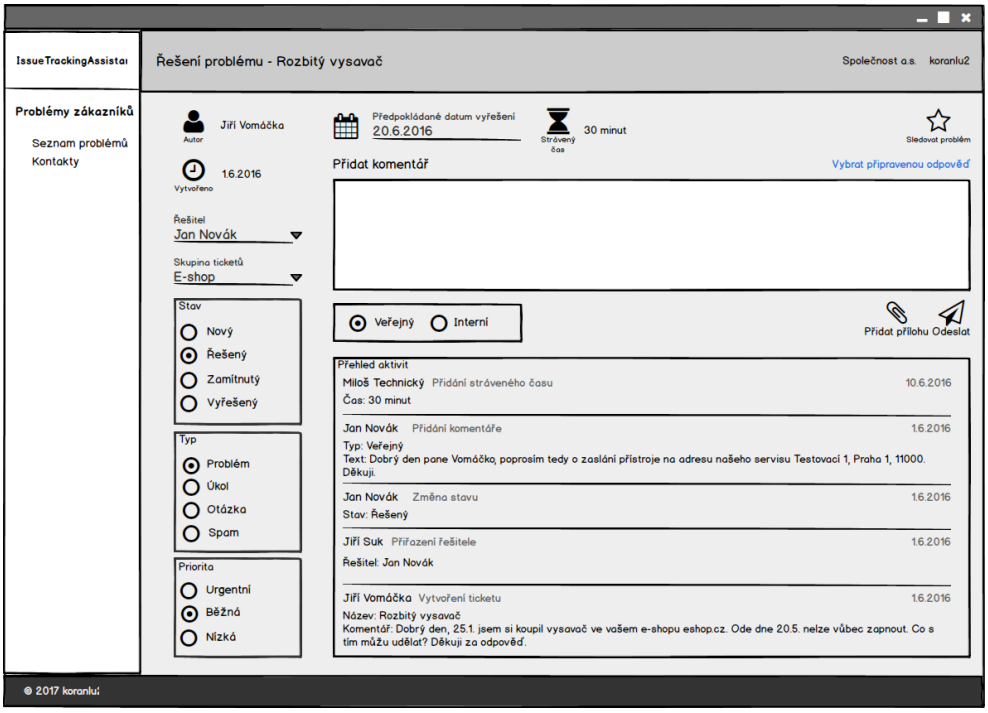
Příloha D

Návrh uživatelského rozhraní

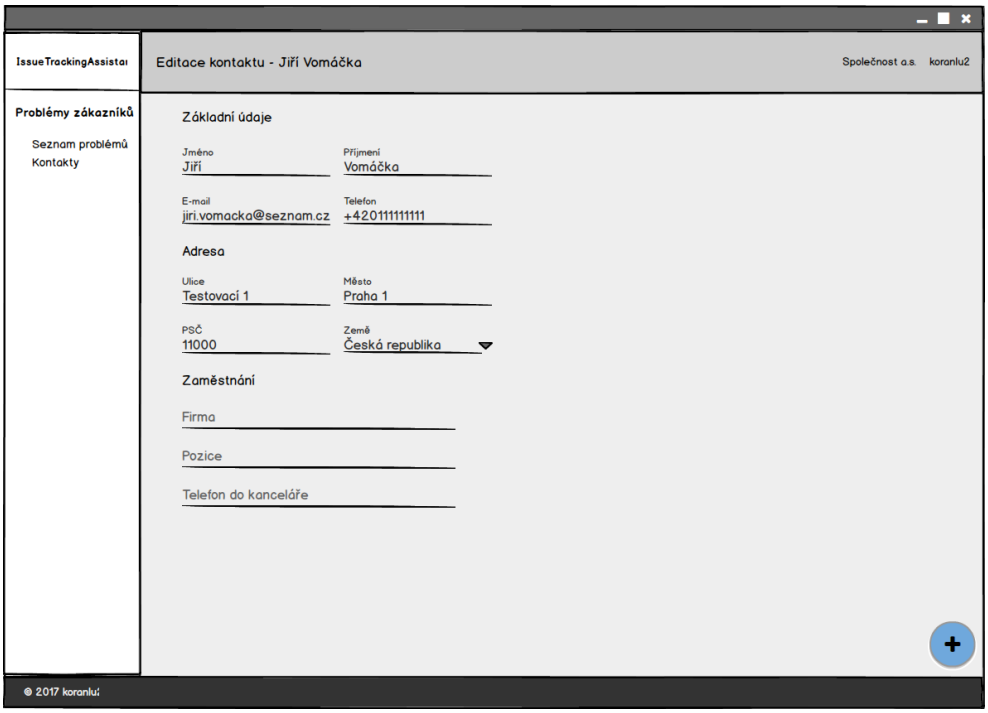
V této příloze se nacházejí návrhy uživatelského rozhraní, pro více informací přejděte do sekce *Uživatelské rozhraní*.



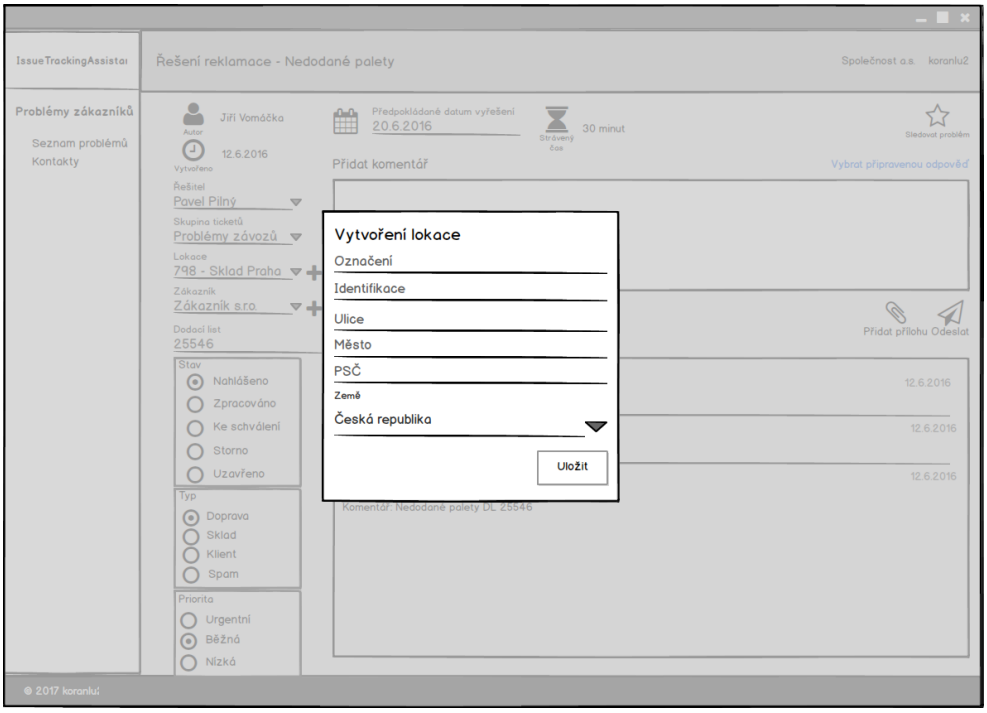
Obrázek D.1: Návrh seznamu problémů



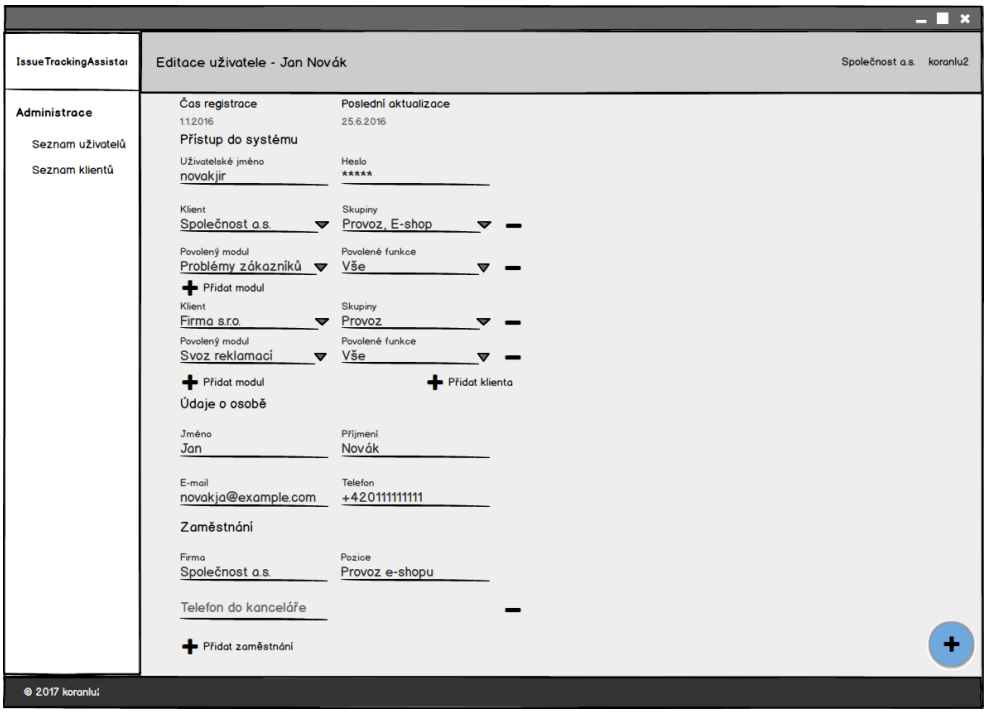
Obrázek D.2: Návrh editace problému



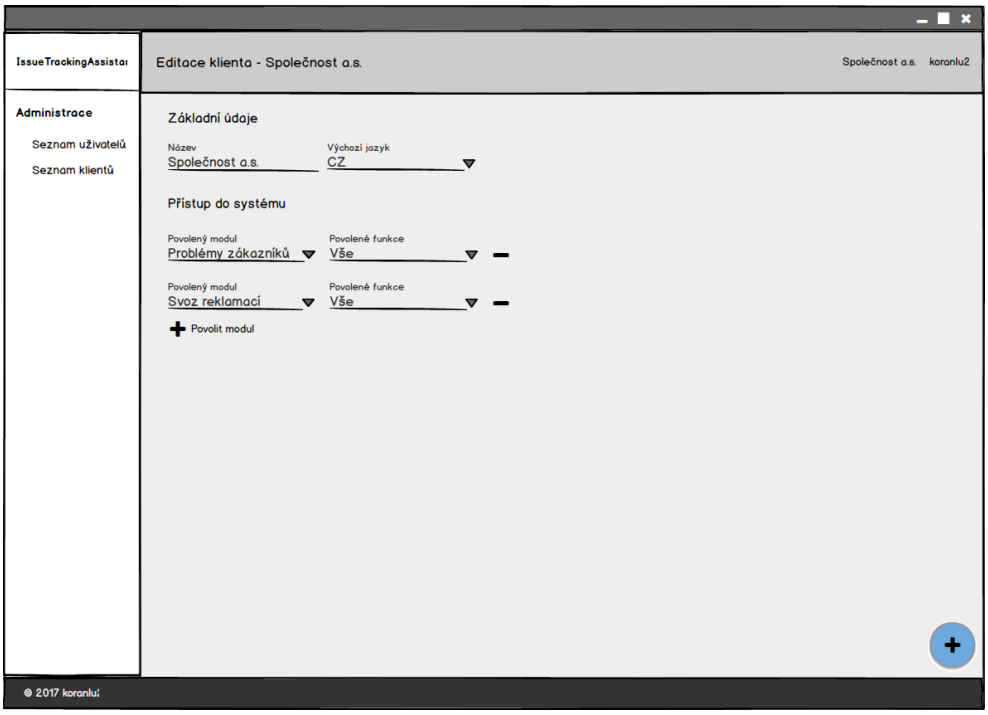
Obrázek D.3: Návrh editace kontaktu



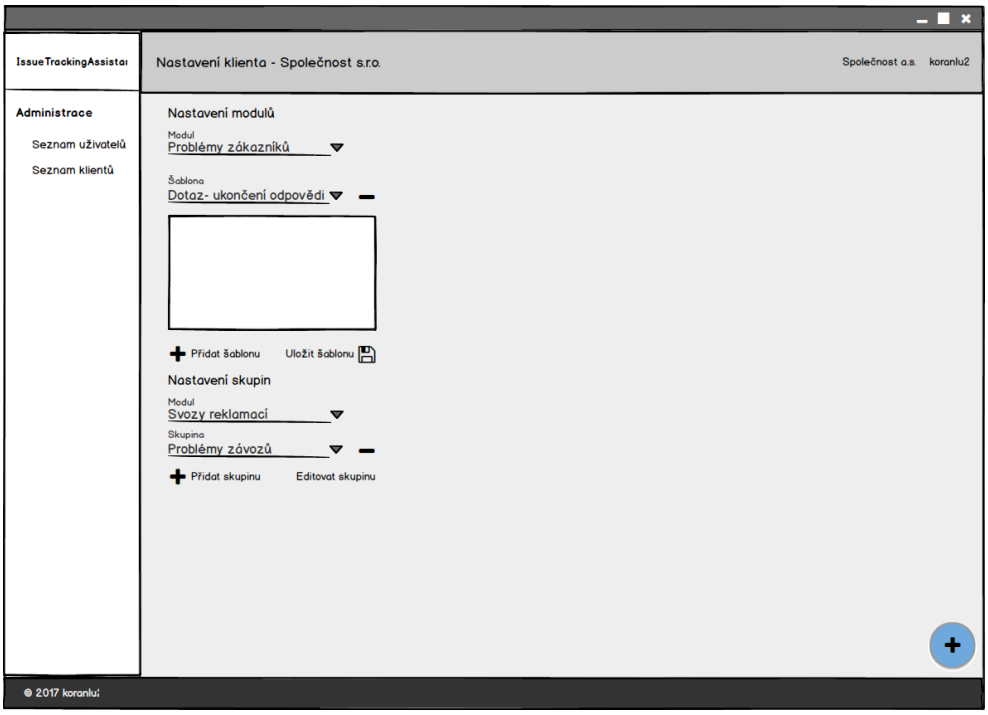
Obrázek D.4: Návrh vytvoření lokace



Obrázek D.5: Návrh editace uživatele



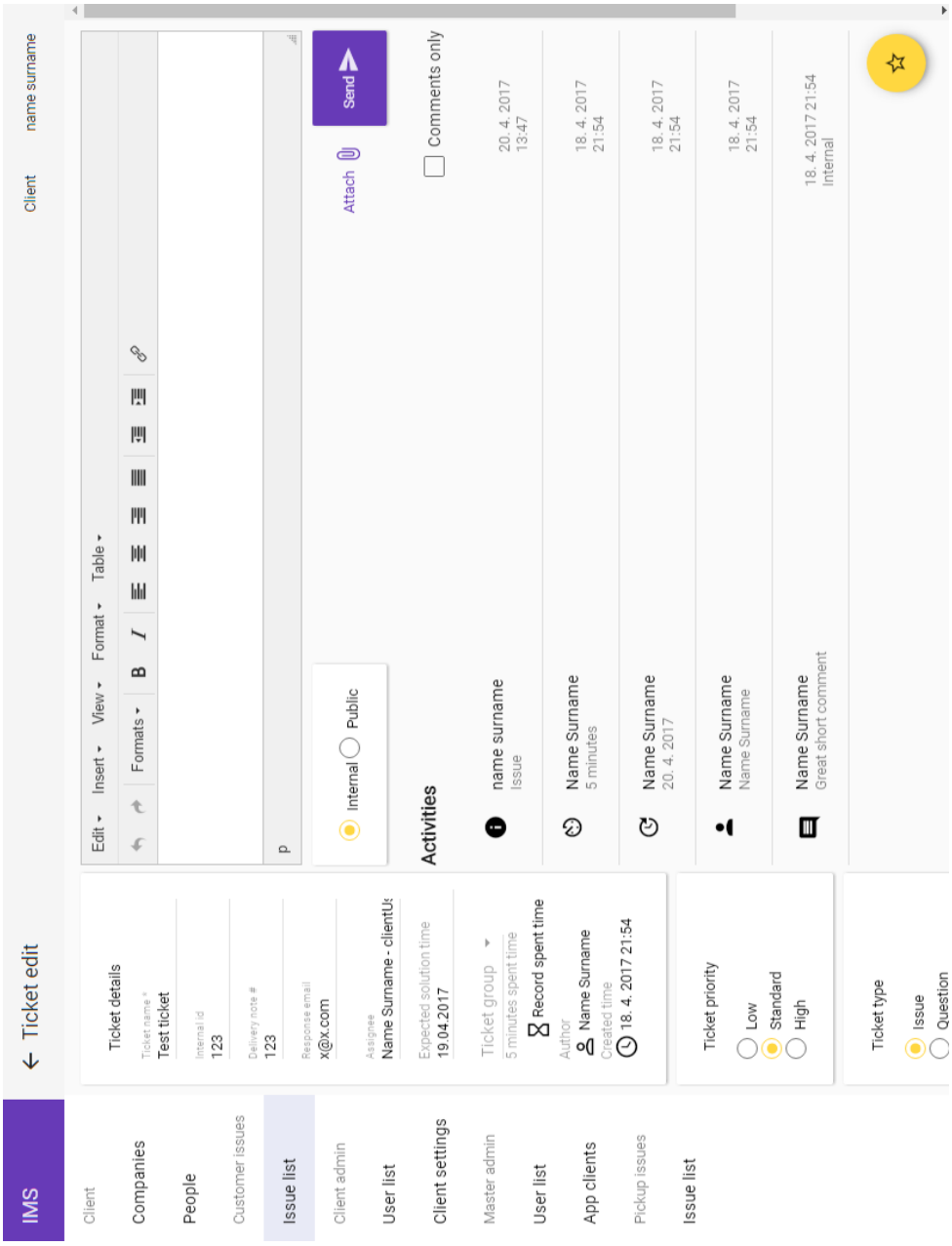
Obrázek D.6: Návrh editace klienta



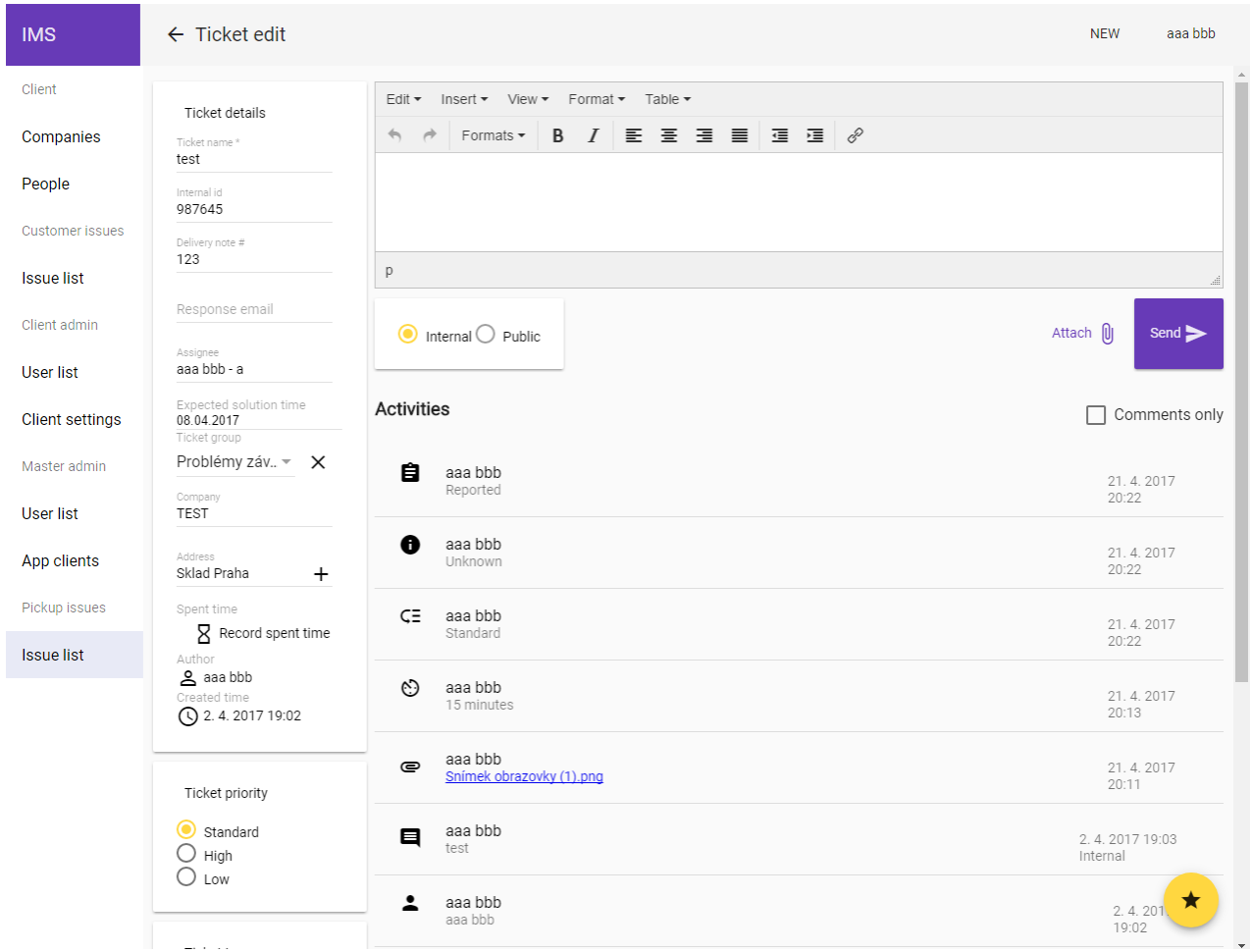
Obrázek D.7: Návrh nastavení klienta

Příloha E

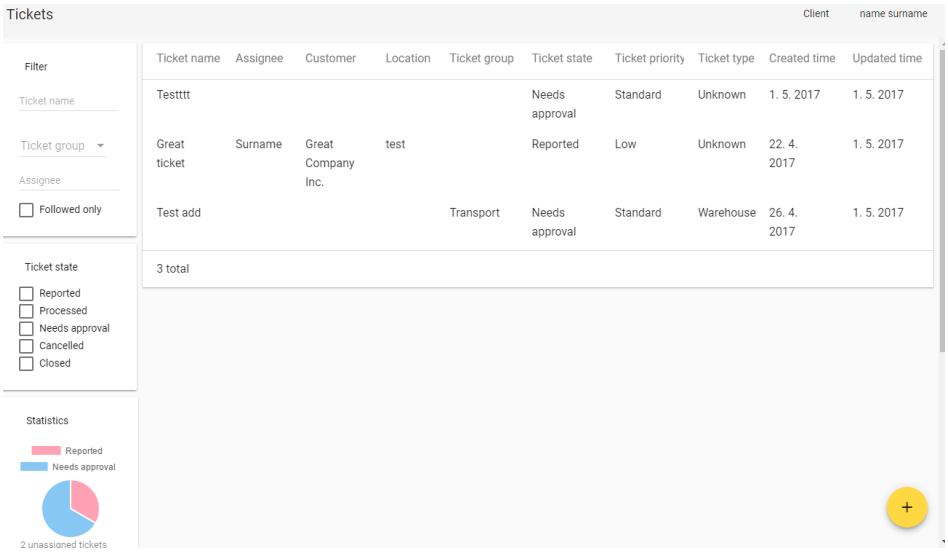
Ukázka prototypu aplikace



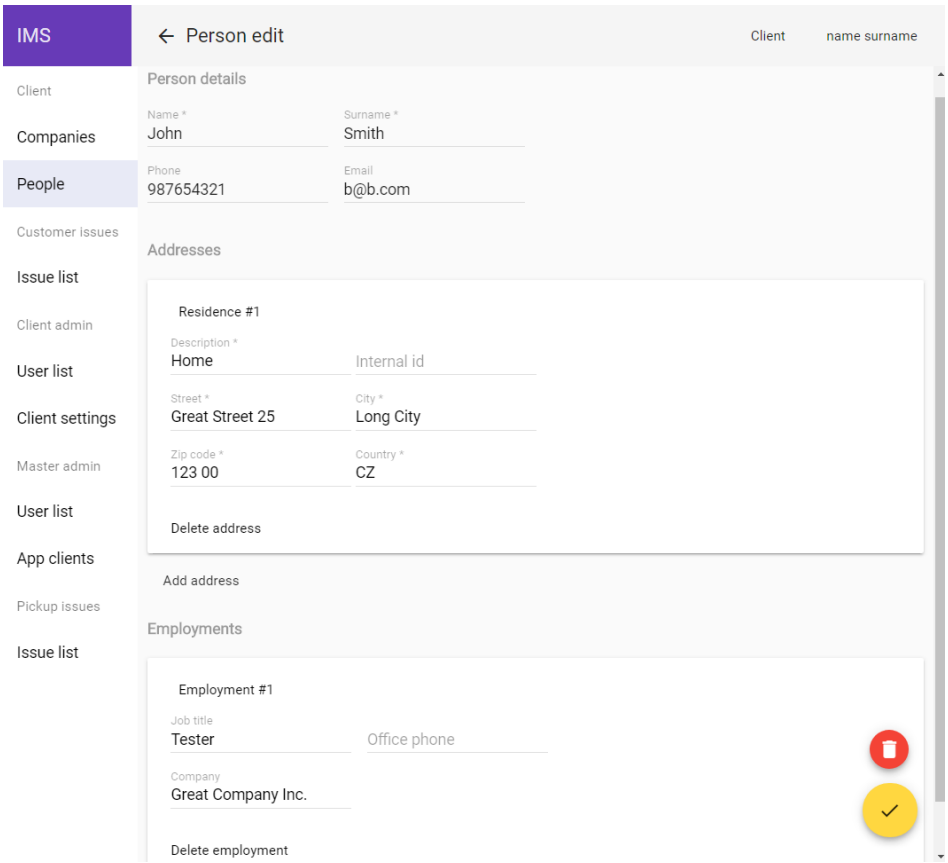
Obrázek E.1: Prototyp – editace problému



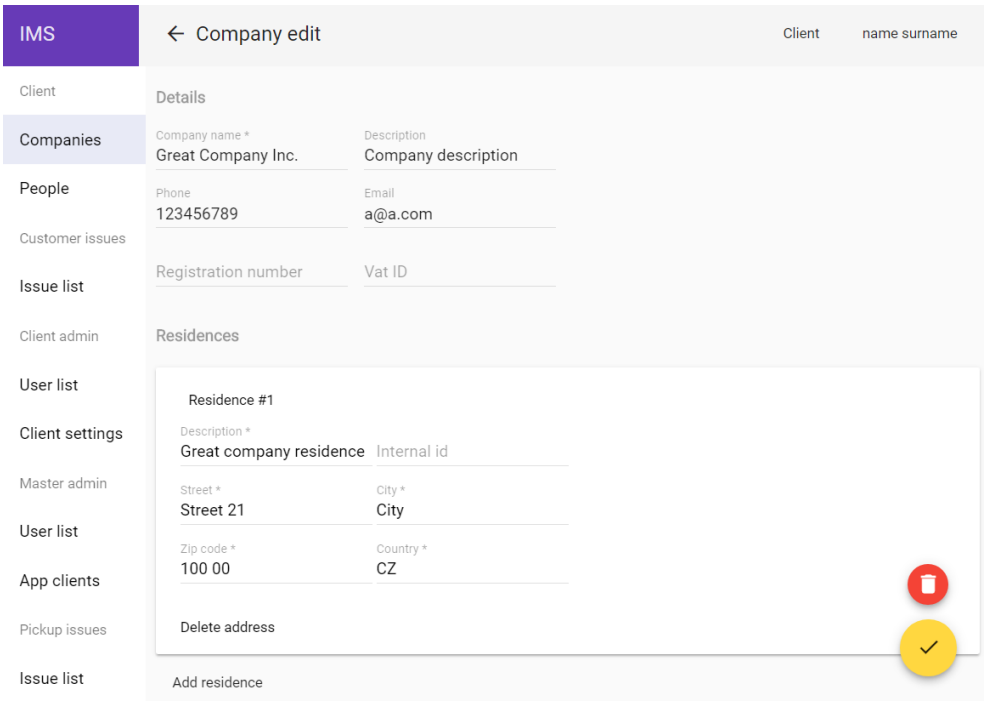
Obrázek E.2: Prototyp – editace reklamace



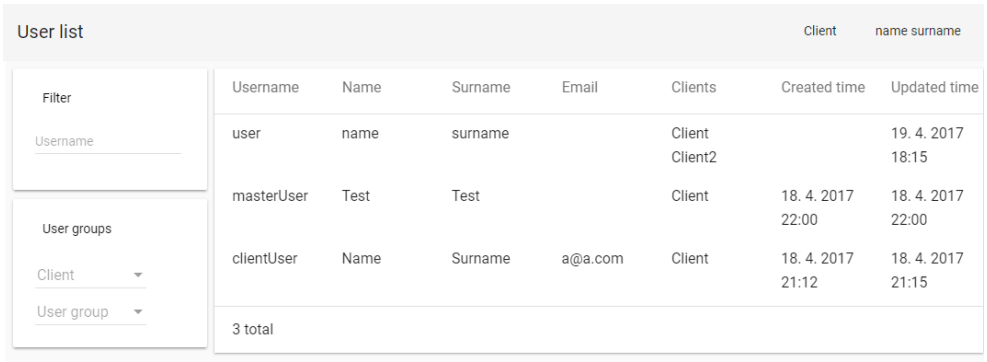
Obrázek E.3: Prototyp – seznam reklamací



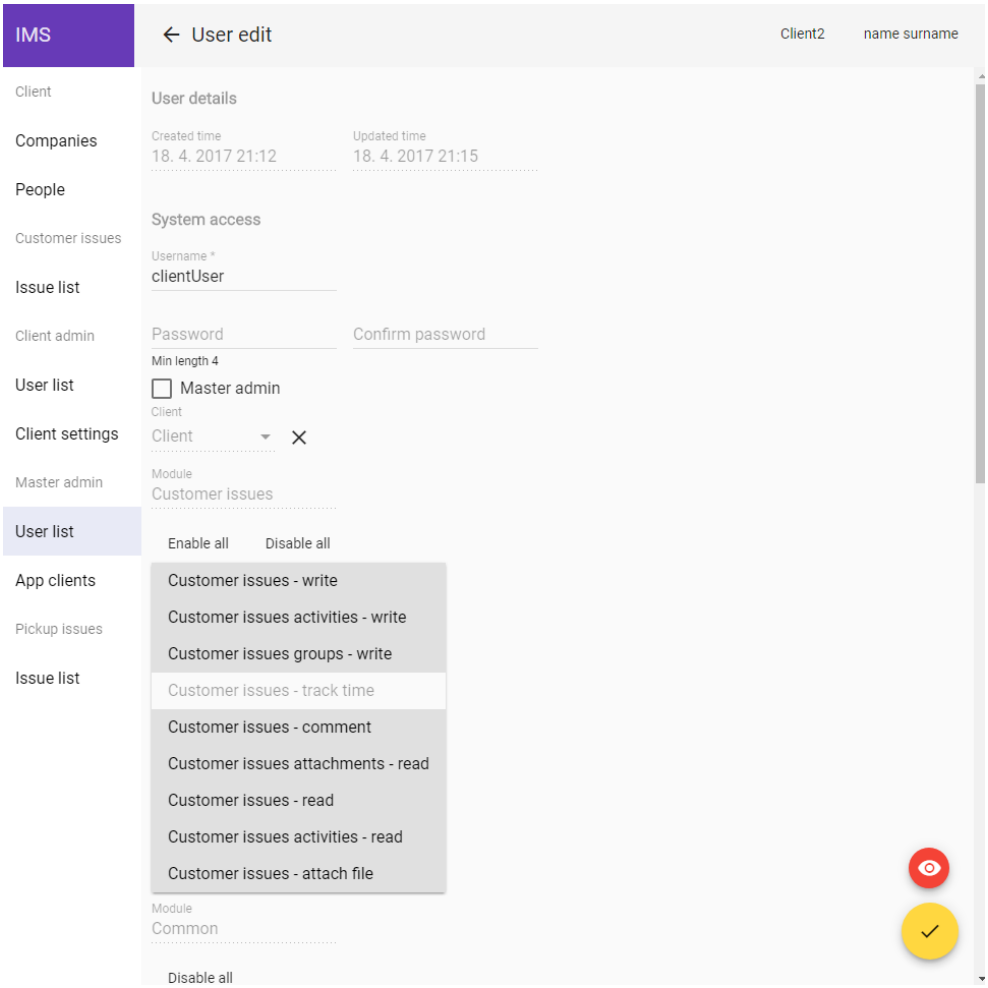
Obrázek E.4: Prototyp – editace osoby



Obrázek E.5: Prototyp – editace firmy



Obrázek E.6: Prototyp – seznam uživatelů



Obrázek E.7: Prototyp – editace uživatele

Příloha F

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `additional` ..... diagramy v původním formátu a velikosti
- `exe` ..... adresář se spustitelnou formou implementace
  - `ims-1.0.0.jar` ..... spustitelné REST API
  - `ims-web` ..... zkompilované soubory webové aplikace
  - `inst.md` ..... instalační příručka
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF