Mendelova univerzita v Brně
Provozně ekonomická fakulta

Multi-tenant portál pro evidenci a správu osobních údajů a souhlasů o jejich zpracování

Diplomová práce

Vedoucí práce:
doc. Ing. Oldřich Trenz, Ph.D.

Bc. Petr Kalas

Brno 2019

Poděkování

Rád bych touto cestou poděkoval doc. Ing. Oldřichu Trenzovi, Ph.D. za vedení diplomové práce, společnosti K-net Technical International Group s.r.o. za možnost pracovat na zajímavém projektu a také rodině a přátelům za jejich podporu.

Čestné prohlášení

Prohlašuji, že jsem práci **Multi-tenant portál pro evidenci a správu osobních údajů a souhlasů o jejich zpracování** vypracoval samostatně a veškeré použité prameny a informace uvádím v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a v souladu s platnou Směrnicí o zveřejňování závěrečných prací.

Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 17. května 2019 .....

podpis

Abstract

Kalas, P., *Multitenant personal data and consents to the processing of personal data management portal.* Diploma thesis. Brno, 2019.

This thesis deals with design and implementation of personal data and consents to the processing of personal data management portal with multitenancy support. The work deals with theoretical basis for the portal design in the field of personal data protection and cloud software development. The final design is implemented and deployed in production environment.

Key words: GDPR, cloud computing, information system, Spring Framework, Angular, Docker

Abstrakt

Kalas, P., *Multi-tenant portál pro evidenci a správu osobních údajů a souhlasů o jejich zpracování.* Diplomová práce. Brno, 2019.

Tato práce se zabývá návrhem a implementací portálu pro evidenci osobních údajů a evidenci souhlasů s jejich zpracováním. V práci je pojednáváno o teoretickém základu pro návrh portálu, a to jak v oblasti ochrany osobních údajů, tak v oblasti vývoje software pro provoz na bázi cloudu. Výsledný návrh je implementován a otestován v produkčním prostředí.

Klíčová slova: GDPR, cloud computing, informační systém, Spring Framework, Angular, Docker

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Osobní údaje a všeobecně soukromí jsou chráněné již desítky, ne-li stovky let. Tato práce vzniká v době a prostředí, kdy je ochraně osobních údajů věnována mimořádná pozornost.

Z širšího hlediska a časového pojetí je důvodem rozvoj společnosti a využití techniky, konkrétněji především informačních technologií. Hodnota osobních údajů roste a stejně tak jejich zneužitelnost. Nejde totiž zdaleka pouze o problém vyzrazení důvěrných informací nežádoucím subjektům, ale zneužitelnost tkví také v páchání nezákonných činů. Příkladem je možnost ukradení identity. Informační technologie jsou prostředkem, který umožňuje osobní údaje velice snadno shromažďovat a stejně tak i zneužívat.

Pozornost v posledních letech však ochraně osobních údajů stoupla ze zcela konkrétního důvodu. Tím je nařízení Evropského parlamentu a Rady č. 2016\679 o ochraně fyzických osob v souvislosti se zpracováním osobních údajů a o volném pohybu těchto údajů, známého také jako GDPR.

Nařízení schválené 27. dubna 2016 a vstupující v platnost 25. května 2018 v rámci členských států Evropské unie je evolucí a sjednocením právního rámce ochrany osobních údajů. Přináší zvýšení ochrany dat s osobními údaji, jednotná pravidla s jejich zacházením a také lepší kontrolu a možnost sankcí případných prohřešků. Výrazné zvýšení rozsahu práv osob v této oblasti přináší i změny a nové povinnosti pro organizace osobní údaje zpracovávající. U takových organizací s nástupem GDPR vzniká často potřeba úpravy zavedených procesů, či také vytvoření procesů zcela nových. Zde se informační technologie dostávají do pozice, ve které organizacím s nakládáním s osobními údaji nabízí pomoc.

1.1 Motivace

Autor práce je zaměstnancem organizace, ve které se otázka úprav zavedených procesů týkajících se nařízení Evropského parlamentu a Rady č. 2016\679 o ochraně fyzických osob v souvislosti se zpracováním osobních údajů a o volném pohybu těchto údajů (dále již jen GDPR) řešila a řeší ve všech jejích aspektech. Tato společnost se také rozhodla být součástí partnerství poskytující právní poradenství, procesní a analytickou podporu a technické řešení z hlediska GDPR pro další organizace.

Motivací pro vývoj softwarového produktu, jehož návrh a implementace je cíl této práce se stala potřeba sjednocení a zefektivnění daných procesů ve vlastní organizaci. Teprve další analýza a průzkum trhu ukázal, že by se takové řešení mohlo stát produktem, který by obsáhl potřeby i dalších organizací. Tato skutečnost se stala základem rozhodnutí vytvořit takový produkt, jenž by mohl být, v souladu s moderním přístupem k poskytování IT služeb, nabízen ve formě cloudové služby.

Další, autorovou osobní, motivací se stala možnost obsáhnutí celého procesu vývoje takového softwarového produktu. Tato možnost nabízí vyzkoušení mnoha rolí. Nejedná se pouze o backend a frontend programátora, testera a UX designéra,

ale také o softwarového architekta a DevOps inženýra – vytvořením aplikační vrstvy vývoj totiž nekončí. Vzhledem k povaze projektu, která se ideově blíží k vývoji software na zakázku, jsou to také role konzultanta, byznys analytika a v neposlední řadě projektového manažera.

1.2 Cíl a obsah práce

Cílem práce je návrh a implementace systému pro evidenci osobních údajů a evidenci souhlasů s jejich zpracováním. Zásadním požadavkem na řešení je provoz na úrovni cloud realizace a s ním spjatá multi-tenantnost a konfigurovatelnost.

Pro celistvý popis dosažení tohoto cíle je čtenář v této práci obeznámen s celým procesem tvorby systému od prvotního konceptu po aktuální stav práce a využití jejího výstupu v praxi. Nutné je také čtenáře seznámit s problematikou zpracování osobních údajů a problematikou vývoje softwarového produktu pro provoz na úrovni cloud realizace.

2 Konceptuální návrh

Tato kapitola pojednává o prvotní vizi a na ní založeném konceptu. Koncept je popsán obecně definovaným rámcem požadavků na systém ve třech kategoriích. Definování konceptu, které probíhalo společně s vedením společnosti se stalo důležitým krokem k vypracování metodiky a následnému zahájení studia dotčených oblastí tak, aby mohl být kvalitně a odborně vypracován návrh řešení.

2.1 Zpracování osobních údajů

Základním potřebou je zavedení centralizovaného systému pro práci s osobními údaji. Obecně by měl systém zjednodušit a automatizovat některé kroky procesů týkajících se zpracování osobních údajů. Dále by měl systém poskytnout funkcionalitu, která je nutná pro plnění některých práv a povinností subjektu osobních dat a také organizací osobní data zpracovávající. Veškerý návrh by měl být v co největší míře v souladu s nařízením GDPR.

- Systém umožňuje přístup subjektu k osobním údajům o něm zpracovávaných a manipulaci s nimi.
- Systém umožňuje organizaci správu a manipulaci s osobními údaji.
- Manipulace s daty v systému je evidována.
- Systém dokáže subjekty zpracovávaných dat i organizaci notifikovat o nastalých událostech.
- Systém umožňuje exportování a importování dat.
- Systém nabízí možnost integrace s dalšími systémy.
- Systém nabízí přívětivé uživatelské prostředí.

2.2 Cloudová služba

V rámci vize poskytovat výše popsaný systém jako službu na bázi cloudu budou muset být v návrhu provedeny některé zásadní architektonické rozhodnutí týkající se provozu aplikace. Vznikají také další dvě skupiny požadavků. Tou první je *konfigurovatelnost systému*. Jedná se o možnost vlastního nastavení systému pro novou organizaci bez nutnosti zásahu do zdrojového kódu, tedy z grafického uživatelského prostředí v následujících oblastech:

- **Systémové proměnné**. Je možné nastavit parametry jako název společnosti, logo, doménu nebo email.
- **Datový model**. Systém umožňuje alespoň částečnou konfiguraci vlastního datového modelu z grafického prostředí. Například konfigurace typů zpracovávaných osobních dat.

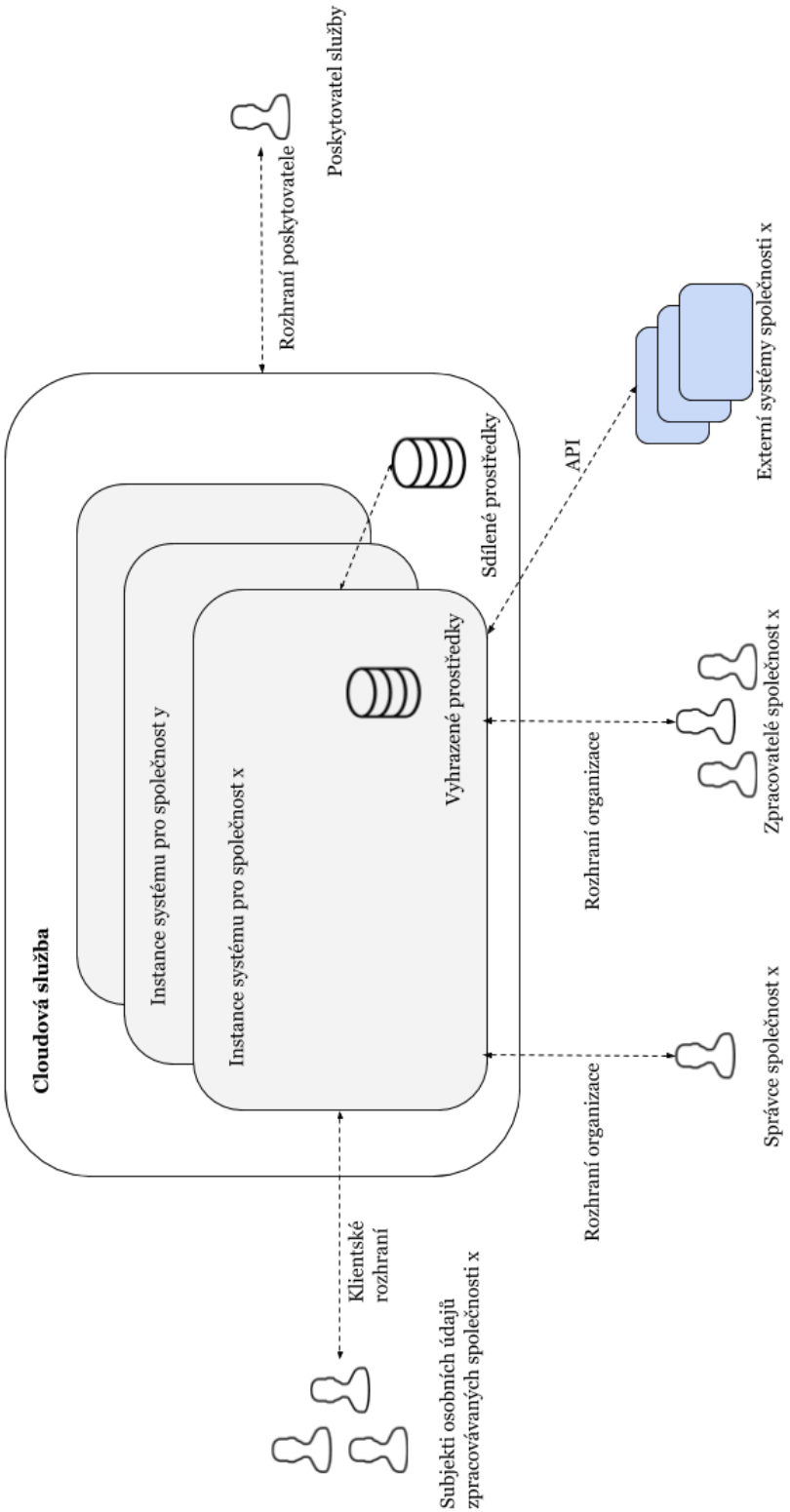
Druhou skupinou požadavků na systém, která vzniká s ideou provozu systému na bázi cloudu, jsou požadavky na běhové prostředí. Dále také označované jako *infrastruktura*. Infrastruktura by měla z velmi obecného hlediska zajistit následující záležitosti:

- **Automatizovanost**. Procesy jako nasazení aplikace pro novou společnost nebo aktualizace systému by měly probíhat automatizovaně bez nutnosti zásahu odborníka.
- **Spolehlivost**. Systém je odolný proti chybám, je schopný samostatného zotavení. Automatizované procesy jsou v případě chyby schopny jejich ohlášení a jejich operace jsou zaznamenávány.
- **Možnost správy a monitoringu**. Systém je možné spravovat a měnit jeho konfiguraci. Je umožněn snadný přístup k provozním informacím.

2.3 Bezpečnost

Dalším důležitým aspektem pro systém nakládající s citlivými údaji je bezpečnost. V popisovaném konceptu je možné mluvit o zabezpečení na třech úrovních.

- **Zabezpečení z pohledu designu systému**. Jedná se například o absenci citlivých údajů v tělech odchozích zpráv.
- **Aplikační zabezpečení**. Tedy záležitosti jako je autorizace a autentizace uživatelů nebo šifrování přenášených souborů.
- **Zabezpečení infrastruktury**. Zde se jedná o zajištění záležitostí jako zálohování dat, nakládání se systémovými přihlašovacími údaji nebo také řízení přístupu na síťové vrstvě.



Obr. 1: Konceptuální návrh - diagram

3 Ochrana osobních údajů

Kapitola se zabývá problematikou zpracování osobních údajů v informačních technologiích, GDPR a jeho zaváděním do podniků. Čtenář by měl dostat základní přehled v dané oblasti, vhodné pro porozumění dalším částem práce.

Osobní údaj je dle zákona o ochraně osobních údajů č. 101/2000 Sb. definován jako jakákoli informace týkající se určeného nebo určitelného subjektu údajů. Tento právní předpis byl v dubnu 2019 zrušen a v účinnost vstoupila adaptační legislativa k GDPR. Dle GDPR jsou osobní údaje veškeré informace o identifikované nebo identifikovatelné fyzické osobě.

3.1 Osobní údaje

Hodnota osobních údajů

Osobní údaje vytvářejí v digitální trhu ekonomickou hodnotu. V rámci dvoustranného tržního mechanismu fungují online platformy jako zprostředkovatelé, kteří shromažďují data od spotřebitelů a prodávají nasbírané informace zejména reklamním firmám (Nezmar, 2017). Množství uložených osobních údajů roste a s nimi výnosy z online reklamy. Velmi dobrým příkladem je společnost Facebook, pravděpodobně největší zpracovatel osobních dat současnosti, jejíž příjmy jsou tvořeny téměř výhradně reklamou. V roce 2018 překročily příjmy Facebooku 55 miliard dolarů (Statista, 2019). V roce 2017 to bylo 27 miliard dolarů a v roce 2016 to bylo 27 miliard dolarů.

Proč chránit osobní údaje

Na otázku proč vůbec osobní údaje chránit se lze podívat z etického respektive i filozofického úhlu pohledu. Soukromí je určitým hodnotným statkem a osobní údaje jsou bezpochyby součástí soukromí každé osoby. Dostáváme se tedy k otázce ochrany soukromí a lidské osobnosti obecně. Ačkoli významné osoby v oblasti zpracování osobních údajů v posledních letech, jako například bývalý CEO společnosti Google Eric Schmidt nebo současný CEO společnosti Facebook Mark Zuckerberg někdy tvrdili jinak, soukromí je důležitou, nezbytnou a přirozenou potřebou každé lidské bytosti (Greenwald, 2014).

Z jiného úhlu pohledu je pak nutné chránit osobní údaje z důvodu zneužitelnosti. Tento problém byl již nastíněn v úvodu práce. Navrátil (Navrátil, 2018) uvádí, že kromě ukradení identity může jít například o průmyslovou špionáž. Pokud má někdo k dispozici nějaké důvěrné informace, jejich hodnota pro něho prudce vzroste v případě, že má i informace o osobách, z jejichž výpočetní techniky byly takové důvěrné informace získány.

3.2 Zavádění GDPR

V dubnu roku 2016 bylo GDPR schváleno Evropskou radou. Dne 25. května 2018 vešlo v účinnost. Vzhledem k tomu, že se jedná o evropské nařízení a nikoli směrnici, je GDPR platné ve všech státech EU.

V období od dubna 2016 do května 2018 měly členské státy definovat a schválit své adaptační zákony. Tam, kde se tak stalo, došlo k zmírnění podmínek. České republice adaptační zákon chyběl, a tak v den účinnosti na našem území začalo nařízeni platit v základní, nejtvrdší podobě (BusinessInfo, 2018). V dubnu roku 2019 vstupuje konečně adaptační zákon nařízení GDPR České republiky v platnost.

Tato sekce se dále bude zabývat výběrem pojmů a definic GDPR. Obsahem výběru jsou některé základní definice a především záležitosti související s definovaným konceptem.

Cíl GDPR

Navrátil (Navrátil, 2018, převzato) uvádí "Cílem GDPR je:

- přizpůsobení právní regulace ochrany osobních údajů poměrům dnešní doby,
- sjednocení práva ochrany osobních údajů ve všech zemích Evropské unie a dalších zemích, na které dopadá,
- posílení práv v oblasti ochrany osobních údajů všech osob, které jsou subjekty údajů a dosáhnout výkladu GDPR dozorovými úřady jednotlivých zemí Evropské unie,
- posílit důvěryhodnost Evropské unie a jejích členských zemí (i dalších zemí, které pod GDPR spadají) pro jiné země, které mají zájem na rozvoji obchodu s Evropskou unií a s tím souvisejícím předáváním osobních údajů mezi zeměmi."

Přístupy GDPR

Škorničková (Škorničková, 2017) uvádí, že lze hovořit o dvou nových přístupech, na kterých je GDPR založeno:

- **Princip odpovědnosti**. Jedná se o odpovědnost správce dodržet zásady zpracování osobních údajů a zároveň musí správce být schopen tento soulad doložit. Jedním ze způsobu doložení jsou záznamy o činnostech zpracování, uvedeny níže v textu stejně jako zásady zpracování osobních údajů.
- **Přístup založený na riziku**. Převzato z webových stránek Úřadu pro ochranu osobních údajů (Úřad pro ochranu osobních údajů, 2019): *"Přístup založený na riziku v širším slova smyslu znamená, že správce již od počátku koncipování zpracování osobních údajů musí brát v potaz povahu, rozsah, kontext a účel zpracování a přihlédnout k pravděpodobným rizikům pro práva a svobody fyzických osob a tomu musí přizpůsobit i zabezpečení osobních údajů. V pojetí*

obecného nařízení tento přístup navíc znamená aplikaci dodatečných povinností pro některé správce, kdy zpracování osobních údajů či porušení zabezpečení (bezpečnostní incident) představuje riziko či vysoké riziko pro práva a svobody fyzické osoby a je tedy důvodné aplikovat tyto povinnosti".

Hlavní zásady zpracování údajů

Správce v rámci *principu odpovědnosti* zodpovídá za dodržení následujících zásad (Navrátil, 2018, převzato): "Osobní údaje musí být:

- zpracovávány korektně a zákonným a transparentním způsobem (zákonnost, korektnost a transparentnost) čili musí být při jejich zpracování dodržovány všechny právní předpisy, které se jich dotýkají, a musí být zpracovávány tak, aby bylo možné identifikovat, jaké osobní údaje a jak byly zpracovány,
- shromažďovány pro určité, výslovně vyjádřené a legitimní účely a nesmějí být dále zpracovávány způsobem, který je s těmito účely neslučitelný; další zpracovávání pro účely archivace ve veřejném zájmu, pro účely vědeckého či historického výzkumu nebo pro statistické účely se nepovažuje za porušení tohoto principu (tzv. účelové omezení),
- přiměřené, relevantní a omezené na nezbytný rozsah ve vztahu k účelu, pro které jsou zpracovávány (minimalizace údajů), což znamená, že nesmějí být shromažďovány osobní údaje, které nejsou skutečně potřebné,
- přesné a v případě potřeby aktualizovatelné, musí být přijata veškerá rozumná opatření, aby se osobní údaje, které jsou nepřesné s přihlédnutím k účelům, pro které se zpracovávají, byly bezodkladně vymazány nebo opraveny (přesnost), osobní údaje musí být zpracovávány bez chyb (například chyba psaní ve jméně určité fyzické osoby ji může způsobit dalekosáhlé následky, například může být určitý velmi důvěryhodný dokument doručen jiné osobě, který pak podle něj v dobré víře jedná,
- uloženy ve formě umožňující identifikaci subjektů údajů po dobu ne delší, než je nezbytné pro účely, pro které jsou zpracovány, čili musí být zajištěna jejich včasná a kvalifikovaná skartace; osobní údaje lze uložit po delší dobu, pokud se zpracovávají výhradně pro účely archivace ve veřejném zájmu, pro účely vědeckého či historického výzkumu nebo pro statistické účely podle čl. 89 odst. 1 GDPR, a to za předpokladu provedení příslušných technických a organizačních opatření požadovaných tímto nařízením s cílem zaručit práva a svobody subjektu údajů (omezení uložení),
- zpracovávány způsobem, který zajistí náležité zabezpečení osobních údajů, včetně jejich ochrany pomocí vhodných technických nebo organizačních opatření před neoprávněným či protiprávním zpracováním a před náhodnou ztrátou, zničením nebo poškozením (integrita a důvěrnost)."

(Navrátil, 2018)

Pojmy

- **Správce**. Dle definice GDPR je správcem osobních údajů každý subjekt, nerozhoduje jaké právní formy, který určuje účel a prostředky zpracování osobních údajů, provádí za jím stanoveným účelem jejich shromažďování, zpracování a uchování. Správce primárně odpovídá za zpracování osobních údajů. (Škorničková, 2017)
- **Zpracovatel**. Zpracovatel, dle definice GDPR *Zpracovatel osobních údajů* je fyzická nebo právnická osoba, orgán veřejné moci, agentura nebo jiný subjekt, který jménem správce zpracovává osobní údaje. Od správce se zpracovatel liší tím, že v rámci činnosti pro správce může provádět jen takové zpracovatelské operace, kterými jej správce pověří nebo vyplývají z činnosti, pro kterou byl zpracovatel správcem pověřen (Škorničková, 2017).
- **Účel zpracování**. Jedná se o účel, ke kterému se zpracování osobních údajů váže a na základě kterého je určen právní důvod.
- **Právní důvod**. Dle serveru gdpr.cz (Škorničková, 2017) je právní důvod nezbytným předpokladem, aby bylo zpracování legální. Osobní údaje jenž nejsou zpracovány bez řádného právního důvodu jsou tedy zpracovávány nezákonně a měly by být zlikvidovány. Typicky je právním důvodem udělení souhlasu subjektem osobních dat. Existují i další právní důvody z nichž je tu skupina pod pojmem *oprávněný zájem*. Oprávněným zájmem může být například zpracování nezbytné pro splnění smlouvy.

Právo na přístup

Subjekt údajů má právo, na základě své žádosti, získat od správce potvrzení, zda osobní údaje, které se ho týkají, jsou či nejsou zpracovávány. Pokud je tomu tak, má právo na přístup k těmto osobním údajům (Navrátil, 2018) a také k mnoha souvisejícím informacím, například pro jaké účely jsou osobní údaje zpracovávány.

Důležitou skutečnosti v kontextu této práce je povinnost správce poskytnout elektronickou kopii zpracovávaných údajů.

Právo na přenositelnost

S předchozím právem souvisí právo na přenositelnost. Navrátil uvádí (Navrátil, 2018), že subjekt údajů má právo získat osobní údaje, které se ho týkají, jež poskytl správci, ve strukturovaném, běžně používaném a strojově čitelném formátu a právo předat tyto údaje jinému správci, aniž by tomu správce, kterému byly osobní údaje poskytnuty, bránil a to v případě, že je zpracování založeno na souhlasu nebo smlouvě a zpracování se provádí automatizovaně.

Právo na opravu

Subjekt osobních údajů má právo na opravu svých osobních údajů. Škorničková (Škorničková, 2017) uvádí, že by měl správce zajistit podmínky pro to, aby žádosti na opravu mohly být podávány online, zejména v případě zpracování osobních údajů elektronickými prostředky.

Právo být zapomenut

Nebo také *právo na výmaz*. Škorničková (Škorničková, 2017) uvádí, že spočívá v provedení přiměřených kroků, včetně technických opatření, vedoucí k vymazání veškerých odkazů na osobní údaje žadatele a jejich kopie. GDPR definuje mnoho výjimek a v praxi tak není snadné toto právo uplatnit. Důvodem je, že podnik je povinen provedené smazání prokázat. Dle Navrátila (Navrátil, 2018) je k tomuto účelu nutné vytvořit proces se všemi technickými záležitostmi pro odstranění dat ze všech systémů. Proces by měl počítat také s anonymizací dat a s odstraněním dat na základě vypršení souhlasu s jejich zpracováním.

Důležitou skutečností je, že pokud jsou osobní údaje, které organizace spravuje, zpracovávány třetí stranou, je povinností organizace zajistit smazání a anonymizování těchto údajů i na jejich straně.

Záznamy o činnostech zpracování

Každý správce a jeho případný zástupce je povinen vést záznamy o činnostech zpracování, za něž odpovídá. Správce je povinen je na žádost zpřístupnit dozorovému orgánu. Dle Škorničkové (Škorničková, 2017) jsou tyto záznamy náhradou za oznamovací povinnost, která s příchodem GDPR zanikla. Dále GDPR vymezuje, jaké konkrétní údaje musí být součástí dokumentace o zpracování osobních údajů. Jedná se o jméno a kontaktní údaje správce, účely zpracování, rozsah zpracovávaných osobních údajů, informace o příjemcích daných osobních údajů, o předávání údajů do třetích zemí, lhůtách pro výmaz jednotlivých kategorií údajů a popis přijatých technických a organizačních opatření k zajištění bezpečnosti údajů.

Dostupná softwarová řešení

Dle GDPR neexistuje žádná povinnost použití nějakých konkrétních technologií pro zabezpečení osobních údajů. Co je definováno jako povinnost pro zpracování je pouze podmínka zabezpečení. GDPR k tomu uvádí některé vhodné přístupy jako je šifrování. Co nařízení požaduje je, aby veškeré nově realizované systémy již ve fázi návrhu s ochranou osobních dat počítaly. Tento koncept je pojmenován jako *"Privacy by design"* a také *"Privacy by default"* (Navrátil, 2018).

Přestože jsou povinnosti dle nařízení definovány jen velmi obecně, IT řešení pro podporu GDPR v podniku jsou mnohdy velmi konkrétní. Jsou jimi softwarová

řešení pro zavádění a provoz GDPR. Na ty se lze dívat z několika úhlů pohledů. Uveďme některé kategorie:

1. Software, jehož účelem je uchovávání, přenášení nebo jiné zpracování osobních údajů v rámci plnění vlastní agendy. Mohou to být ERP systémy, CRM systémy, účetní software, software na řízení lidských zdrojů a další. V rámci nové legislativy implementovaly některé nové funkce nebo prvky zabezpečení, což vedlo k tomu, že se nyní označují jako "GDPR ready" nebo "podporující procesy "GDPR". V některých případech nedochází přímo k implementaci změn v systému, ale je podobné známky dosaženo tím, že autoři software poskytují v oblasti GDPR pro svůj produkt konzultace, nebo novou dokumentaci.

2. Bezpečnostní software jako jsou antiviry, zálohovací software, šifrovací software, monitorovací software a další. V této kategorii výrobci uvádí, že jejich software napomáhá podniku plnit některé povinnosti dle GDPR jako je zajištění bezpečného zacházení s daty, včetně ochrany před neautorizovaným nebo nezákonným procesem. Dále pak plnit doporučení na šifrování a pseudoanonymizaci dat.

3. Software pro analýzu podniku z hlediska zpracování osobních údajů a závádění změn s nařízením GDPR. Jedná se o aplikace, které začaly vznikat s přechodem Obecného nařízení v platnost a poskytují určitou konzultantskou funkci. Jejich účelem je provedení organizace procesem zavádění GDPR v podniku od počátečních analýz jako analýza rizik, provedení interního auditu až po dokumentaci změn (nabízí šablony dokumentů) a řízení opatření ke snížení rizik. Časté jsou k těmto řešení nabídky odborných konzultací.

4. GDPR organizéry jsou další skupinou software vzniknuvší jako reakce na platnost GDPR a především potřebu sjednocení agendy nakládání s osobními údaji a jejich souhlasy. Tento typ software, který bývá označován jako *GDPR compliance software*, nabízí evidenci souhlasů o zpracování osobních údajů, zpracovatelů, žádostí o opravu, žádostí o výmaz, žádostí o výpis, různých smluv jako jsou smlouvy mezi správcem a zpracovatelem, právních důvodu pro zpracování a mnoho dalších agend. V některých případech nabízí i evidenci samotných osobních údajů, není to ale pravidlem. Tato kategorie software se zaměřuje na skutečnost, že je nejen důležité shody s GDPR dosáhnout, ale i v případě kontroly tuto skutečnost dokázat.

Důležitým faktem je, že konkrétní aplikace nejsou často jen jednou z uvedených kategorií, ale jejich funkce se prolínají napříč kategoriemi. To se týká především posledních dvou zmíněných.

Také jejich forma je různá. V mnoha případech se jedná o klasickou *standalone* aplikaci s modelem jednorázového zakoupení a nasazení v podniku. Dále se lze setkat s modelem aplikace nabízené jako služby, tedy aplikace v cloudu, s financováním na bázi průběžných plateb. Některé řešení mají podobu rozšíření, respektive modulu,

do zavedeného software. Speciálně pak se software třetí uvedené kategorie lze setkat v podobě webové stránky implementované formou průvodce a dotazníkových formulářů.

Příkladem první kategorie s přesahem do funkcí kategorie čtvrté je podnikový informační software **Helios**. Ten nabízí v rámci údržbového poplatku základní funkcionalitu pro nakládání s osobními údaji požadovanou zákonem a pak specializovaný modul GDPR Profi za licenční poplatek (Koupilová, 2018). Řešení nabízí agendu jak osobních údajů, tak jejich souhlasů. Dále nabízí funkce pro podporu mnoha dalších povinností a zásad GDPR, jako funkce pro export údajů, výpis hodnot subjektu údajů, anonymizace, logování změn a další.

Příkladem druhé kategorie je produkt od společnosti **Eset**. V základní verzi nabízí antivir, v rozšířené pak i antimalware a šifrovací nástroj. Právě šifrování je oblastí, ve které se přednostně snaží společnost zájemce o zvýšení bezpečnosti v podniku v rámci GDPR zaujmout. Společnost Eset na svých stránkách například uvádí následující (ESET): "Pokud dojde k úniku osobních dat, který představuje pro zákazníka velké nebezpečí, musí firma o úniku dat jednoduše a srozumitelně informovat. Pokud ale jsou uniklá data zabezpečena šifrováním, tudíž se k nim útočník nemůže bez znalostí šifrovacího klíče dostat, pak tato povinnost odpadá."

Aplikace **Ochranou** je příkladem třetí kategorie se zmíněnou podobou online služby na bázi dotazníku. Účelem je usnadnění implementace GDPR v podniku formou komplexního dotazníku pro analýzu společnosti zákazníka. Výrobce sází především na uživatelskou přívětivost řešení. Výstupem aplikace je dokumentace odpovídající provedenému auditu.

Jako zástupce poslední kategorie, GDPR organizéry, lze uvést systém **eDPO**. Jedná se o systém nabízený na bázi cloudu za nabízející širokou škálu funkcí a agend, které zahrnují i funkce třetí zmíněné kategorie, tedy analýzu podniku, průvodce implementací a konzultační možnosti.

4 Vývoj cloudového software

Tato kapitola čtenáři poskytuje základní přehled v oblasti vývoje software a dalších oblastí s touto disciplínou spjatých. Také jsou zde popsány některé konkrétní technologie a přístupy. Obsah této kapitoly odpovídá minimálnímu rozsahu toho, co je nutné pro vypracování návrhu řešení diplomové práce nastudovat.

4.1 Vývoj software

Softwarové inženýrství

Softwarové inženýrství je vědecká disciplína, která se zabývá specifikací, vývojem, řízením a údržbou informačních systémů. Jinými slovy, softwarové inženýrství se soustřeďuje na všechny aspekty výroby softwaru. (Rábová, 2008) V reakci na špatnou situaci v oblasti vývoje software před rokem 1968, která byla označena jako *softwarová krize*, bylo přistoupeno k formalizování přístupů a vytvoření sad pracovních postupů jak by měl vývoj software probíhat. Tyto sady jsou nazvány metodiky a lze je dělit na dvě kategorie.

- **Tradiční metodiky**. Zde je kladen velký důraz na formalizaci, procesní orientovanost, sběr požadavků a analýzu. Jsou také označované jako metodiky *rigorózní*.
- **Agilní metodiky.** Tyto metodiky jsou odezvou na nevýhody metodik tradičních a jejich základem je velká míra flexibility. V dnešní době jsou agilní metodiky trendem stávající se standardem a vývojář software se s uvedením agilních metodik vývoje software setkává u většiny inzerovaných pracovních pozic.

Objevuje se zde pojem *životní cyklus vývoje softwarového produktu*. Pod tím se rozumí definice jednotlivých stádií, kterými software při vývoji musí projít. Životní cykly vývoje softwaru je možné rozdělit na tři kategorie.

- **Lineární**. Tato kategorie je reprezentována modelem zvaným **vodopád**. Základní myšlenkou je rozdělení velkého projektu na malé, na sebe navazující kroky dle plánu vypracovaného na úplném počátku. Plán je následně striktně dodržován. Metodika navazujících kroků je jednoduchou a účinnou strategií, která je prověřená mnoha jinými, nejen inženýrskými, odvětvími. Přestože, jak uvádí Rábová, má vodopádový model naprosto zásadní postavení v softwarovém inženýrství z důvodu jeho prvenství a tedy řízeného procesu a ukončení neřízeného a chaotického vývoje software s velkým rizikem neúspěchu (Rábová, 2008), je tento model dnes již považován za překonaný. Důvodem je především rapidní posun v trendech vývoje software a tento model nenabízí dostatečnou flexibilitu.
- **Iterativní**. Jak uvádí Popelák (Popelák, 2015) Iterativní životní cyklus je v tomto ohledu naprosto opačný, protože nevyžaduje rozsáhlé plánování celého

projektu již při jeho startu. Důvodem je, že v realitě se požadavky mohou měnit denně a je tak výhodné mít možnost reagovat na změny a podněty od všech zainteresovaných stran projektu. Iterační přístup je základem pro agilní metodiky. Velmi rozšířenou konkretní metodikou je **SCRUM**.

- **Kombinované**. Typickým zástupcem kombinované kategorie je **spirálový model** životního cyklu vývoje software. Jak uvádí Rábová (Rábová, 2008), do procesu vývoje se zařazuje iterativní přístup s opakovanou analýzou rizik. V každé otáčce se vytváří prototypové řešení a na základě jeho vyhodnocení se stanoví další postup.

Jaké jsou konkrétní etapy životního cyklu vývoje softwarového produktu popisuje následující výčet (Rábová, 2008):

- **Specifikace problému**. V této etapě je cílem určit realizovatelnost projektu a stanovení základního konceptu.
- **Analýza**. Zde je nutné provést studium problematiky a konkrétního problému.
- **Návrh**. Vytvoření abstraktního modelu systému, technologicky konkrétní.
- **Implementace**. Realizace návrhu v implementačním prostředí.
- **Zavedení a testování**. Instalace technického a programového vybavení.
- **Provoz, údržba a rozvoj produktu**. Posledním bodem je zajištění provozu vytvořeného systému.

4.2 Využití UML

Požadavky

Termín *požadavek* lze definovat jako popis toho, co by měl systém dělat. Součástí požadavku není to, jak by to měl systém dělat. Definice a specifikace požadavků probíhá v prvních etapách životního cyklu vývoje softwarového produktu, jak jsou uvedeny v sekci 4.1. Skutečnost je to zřejmá, protože jen stěží lze implementovat věc, o které není známo, co se od ní požaduje. Tvoření specifikace toho, co má systém dělat je dokonce známo jako vlastní disciplína - *inženýrství požadavků*. Dle Arlowa (Arlow, 2007) je tento termín používaný k popisu aktivit zapojených do zjišťování, dokumentování a údržby množiny požadavků na softwarový systém. Přesto, že samotná specifikace požadavků není součástí UML (Hajník, 2010), UML nám nabízí nástroje k jejich modelování.

Arlow (Arlow, 2007) přistupuje k inženýrství požadavků za pomocí modelu, který dělí specifikaci softwarových požadavků na *model požadavků* a *model případu užití*. Tyto modely jsou různými, ale zároveň vzájemně se doplňujícími způsoby, jak zachytit požadavky na systém.

- **Model požadavků**. Model se dále dělí na *funkční požadavky* (functional requirements) a *nefunkční požadavky* (non-functional requirements).
- **Model případů užití**. Obsahuje balíčky s *případy užití* (use case) a jejich konkrétní případy užití.

Výše uvedené dělení požadavků je pouze jedna z mnoha možností jejich kategorizace. Lze však tvrdit, že se jedná o tu naprosto základní. Je tedy vhodné je podrobněji definovat.

- **Funkční požadavky**. Funkční požadavek je formulace toho, co by měl systém dělat - popisuje požadovanou funkci systému. (Arlow, 2007). Příklad může být u specifikace požadavků na webový prohlížeč možnost instalovat rozšíření.
- **Nefunkční požadavky**. Tento typ požadavků lze chápat jako vlastnosti nebo omezení systému. Příkladem je požadavek naprogramování webového prohlížeče pouze za pomocí jazyku Java (omezení), nebo požadavek na odezvu uživatelského grafického rozhraní prohlížeče kratší než jedna sekunda v jakýchkoli situacích (vlastnost).

Případ užití je metodou pro zachycení funkčních požadavků na systém. Případy užití popisují typické interakce mezi uživateli a samotným systémem (Fowler, 2009). Uživatel je nazýván *aktérem*, nebo také *rolí*. Tělo případu užití je tvořeno *scénářem* skládájícím se z kroků. Scénář by měl skončit úspěšným provedením zamýšlené akce - cíle případu užití. Existují také alternativní scénáře a další volitelné informace jako jsou vstupní/výstupní podmínky, spouštěče nebo sekundární aktéři.

Dle Fowlera (Fowler, 2009) neexistuje žádný standardní způsob, jak zapisovat obsah případu užití, a v různých případech jsou dobře použitelné různé formáty, situace je ale jiná u diagramu pro sadu (balíček) případů použití. Tam je standardní formát specifikován. Diagram případů užití zobrazuje aktéry, případy užití a vztahy mezi nimi - kteří aktéři provádí které případy užití a které případy užití zahrnují další případy užití. Poslední popisovaný vztah je značený třemi způsoby. Vztah *includes* znamená zahrnutí jiného případu užití bezpodmínečně. Vztah *extends* naznačuje, že daný případ užití zahrnuje za daných podmínek chování jiného, od include se liší umístěním závislosti. Posledním možným vztahem je vztah dědičnosti - specializace všeobecnějšího případu užití. Používá se pro snížení složitosti modelu. Posledním prvkem diagramu případu užití jsou *hranice (boundary)*. Je to oblast kolem případů užití, která určuje co je a co není součástí systému.

4.3 Architektura software

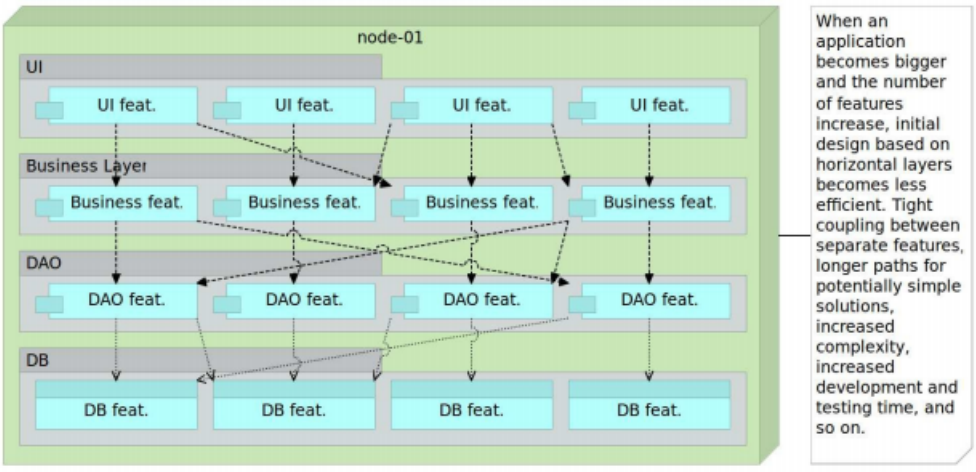
Sommervile (Sommerville, 2016) uvádí, že architektura software se zabývá tím, jak je systém organizován, jak jeho jednotlivé komponenty komunikují a jakou má celkovou strukturu. Také uvádí, že se jedná o důležité spojení mezi inženýrstvím požadavků software a návrhem software.

Existují dva hlavní pilíře dobré softwarové architektury (Kainulainen, 2014). Princip oddělení zodpovědnosti (z anglického *Separation of Concerns - SoC*), který řeší rozdělení programu z funkčního hlediska na části, které by se měly co nejméně překrývat. Druhým principem je KISS (z anglického *Keep It Simple*), jenž se dá aplikovat na návrh řešení v mnoha různých odvětvích a má své nezastupitelné místo v architektuře software.

Architekturu software lze popsat a dělit dle mnoha různých kritérií. Pro účely této práce je vhodné vybrané z nich krátce představit.

Monolitická architektura

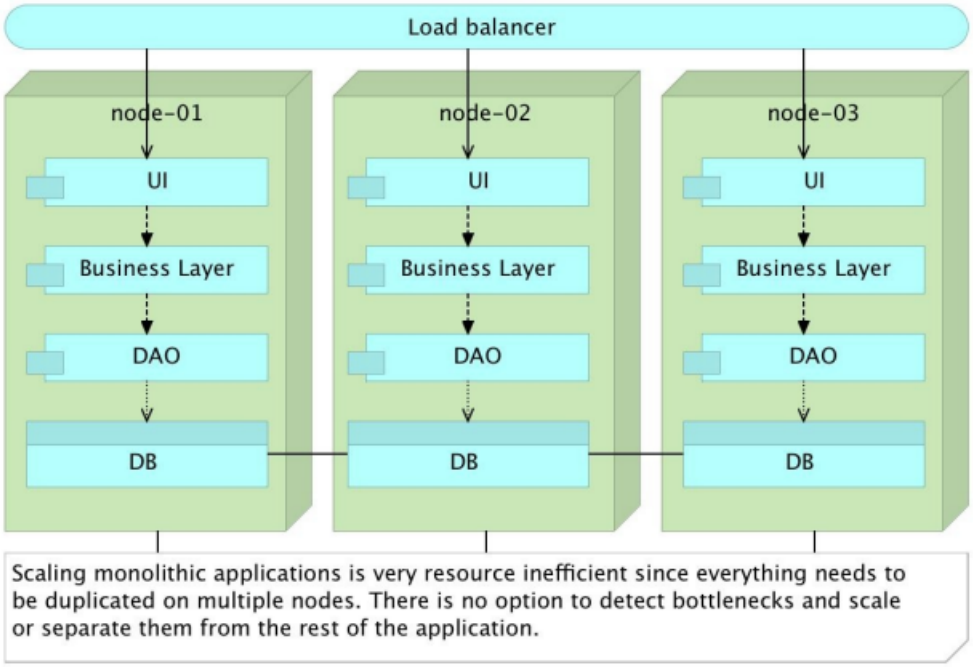
Dle monolitické architektury jsou aplikace vyvíjeny jako jedna celistvá jednotka. Jak uvádí Farcic (Farcic, 2016), většinu doby se v krátké historii vývoje softwaru vykazuje postupným zvětšováním toho, co je vyvíjeno. Jak ubíhá čas, komplexita a velikost aplikací se zvětšuje a rychlost vývoje, testování a nasazování se zmenšuje. Ukázka na obrázku 2. Velká složitost aplikace se pak stává problémem i při malé změně kódu, protože je často nutné upravit stovky závislých funkcí a vrstev. Přidání i té nejmenší funkce vyžaduje otestování a nasazení celku. U velké podnikové aplikace může proces testování a nasazení trvat i celé hodiny. Škálování aplikací na monolitické architektuře se stává velkým problémem. Jediným řešením je totiž duplikace celé aplikace a předřazení systému pro rozložení zátěže (load balancer), viz obrázek 3 Dochází tak často k velmi neefektivnímu využití zdrojů a dalším problémům.



Obr. 2: Monolitická architektura
Zdroj: (Farcic, 2016)

Mikroservisní architektura

Tato architektura se vyznačuje tím, že je jeden funkční celek rozložen na více samostatných jednotek - servisů. Samostatnost je stěžejním principem tohoto přístupu.



Obr. 3: Monolitická architektura - rozdělení zátěže
Zdroj: (Farcic, 2016)

Servisy jsou samostatně vyvíjeny, testovány a také jsou separátně nasazovány do prostředí (Farcic, 2016). Mikroservisní architektura vychází z principů *architektury orientovaná na služby* (Service Oriented Architecture - SOA), v které jsou služby definované jako volně vázané, znovupoužitelné komponenty, které mají plnit jen jednu úlohu či určitou příbuznou oblast úloh. (Sommerville, 2016). SOA ale neřeší nezávislé nasazení a jedná se tak spíše o architekturu uvnitř monolitu.

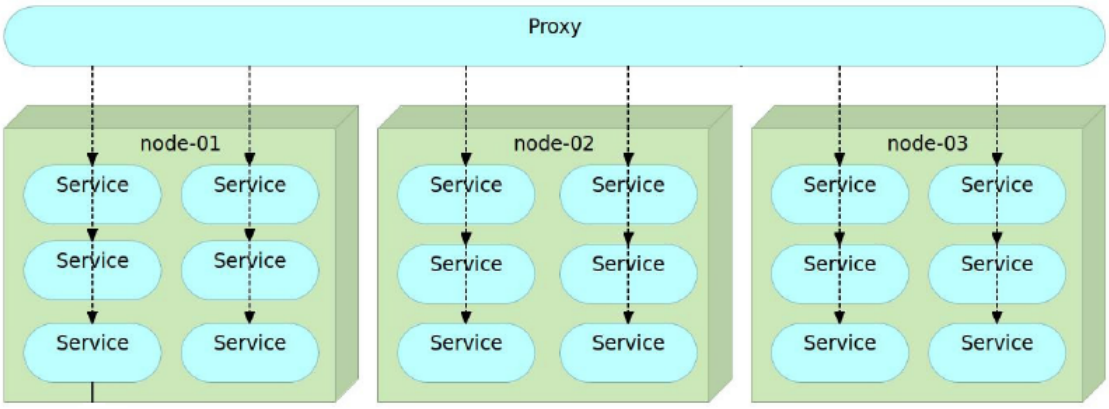
Srovnání mikroservisní a monolitické architektury

Hlavní nevýhodou mikroservisní architektury je zvýšená režijní složitost a složitost nasazení. (Farcic, 2016) Problémem je také rozdělení projektu na malé celky. Zvláště v počátečních fázích vývoje rozdělení nemusí být zřejmé a taky se může zdát, že rozdělováním řešíme problém, který (v danou chvíli) vůbec neexistuje. Monolitní aplikace jsou obecně rychlejší, protože volání metod vně aplikace je vždy méně náročné než volání vzdálených procedur. Také množství kódu při vytváření rozhraní, který je potřeba napsat, aby vůbec volání vzdálených procedur bylo možné, je nezanedbatelné.

Farcic (Farcic, 2016) uvádí pět oblastí, v kterých je mikroservisní architektura výhodnější monolitická:

- **Škálování**. Narozdíl od duplikace celé aplikace v případě monolitu, u mikroservis můžeme škálovat jen tu část, která to opravdu potřebuje.

- **Inovace**. Pokud je aplikace rozdělená na malé celky, je snadné jeden z nich předělat na novější technologii. Stačí pouze respektovat rozhraní. Přepisovat velkou monolitickou aplikaci je obtížně a drahé.
- **Velikost**. Díky tomu, že jsou mikroservisy malé, jsou mnohem snadnější k pochopení. S malými celky je i rychlejší práce, od rychlejšího pochopení byznys logiky až k rychlejšímu běhu vývojového prostředí.
- **Nasazení**. Nasazování malých celků je vždy rychlejší, než nasazování všeobsáhlého monolitu. Pokud se nasazení nepodaří, nedojde k vyřazení celého systému.
- **Závazek**. Jedním z velkých problému monolitu je závazek, který vzniká při výběru implementačních technologií. Pokud jsme jednu z mikroservis navrhli od začátku špatně, není to takový problém. Lze ji snadno přepracovat.



Obr. 4: Mikroservisní architektura - mikroservisy na více uzlech
Zdroj: (Farcic, 2016)

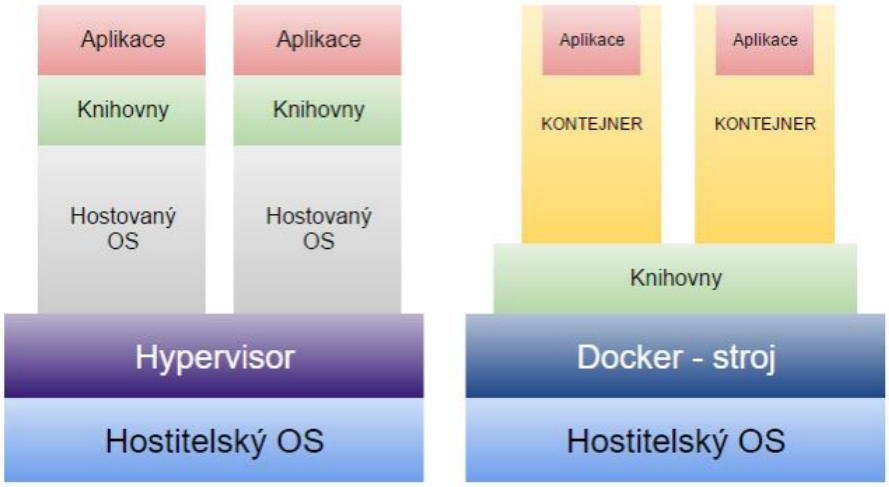
4.4 Vrstvená architektura

Tento architektonický přístup odpovídá zmíněnému principu oddělení zodpovědnosti. Funkcionalita je dle Sommervile (Sommerville, 2016) rozdělená do vrstev z nichž každá využívá služeb nabízených vrstvou pod ní prostřednictvím rozhraní a má určitou odpovědnost. Vrstvená architektura podporuje inkrementální přístup k vývoji software. Problémem těchto architektur v praxi se, jak uvádí Kainulainen (Kainulainen, 2014), stává nedodržení principu KISS a vytváření návrhů s příliš mnoha vrstvami. Přidávání nových funkcí do takového systému se stává obtížné z důvodu nutnosti transformace dat mezi vrstvami a také se kód stává nepřehledným a špatně udržovatelným.

Populární zástupcem vrstvené architektury je MVC (*Model-view-controller*). Kainulainen tvrdí, že pro webové aplikace jsou tři vrstvy optimální počet.

4.5 Kontejnerizace

Kontejnery je typ virtuálního stroje, který sdílí více prostředků než v případě klasické virtualizace založené na hypervisoru. Tam, kde běžně přistupuje k hardwaru počítače více operačních systémů přes hypervisora, který je nejvyšším arbitrem, v případě kontejnerizace běží kontejnery nad kernelem operačního systému. Virtualizace pomocí kontejnerů je tak často nazývaná virtualizací na úrovni operačního systému. Velmi dobře je tato architektura popsaná na obrázku 5. Technologie kontejnerů má v IT prostředí dlouhou historii, ale pro některé svoje nevýhody byla vždy ve stínu klasické virtualizace. Tato situace se v posledních letech mění s příchodem technologie Docker.



Obr. 5: Virtualizace a kontejnerizace
Zdroj: (Mračko, 2016)

4.6 Webové aplikace

Typy webových aplikací

Z hlediska typů webových aplikací je v této práci vhodné rozlišit webové aplikace jednostránkové a vícestránkové.

- **Vícestránkové webové aplikace**. Tradiční přístup k tvorbě webových aplikací. Interakce uživatele s aplikací způsobí její obnovení / načtení. Důvodem je to, že požadavek je zaslán na server, tam je zpracován a následně odeslán ke klientovi, kde je obsah vyrenderován v prohlížeči. Výhodou těchto aplikací je dobrá a nekomplikovaná SEO optimalizace, jednoduchost vývoje (z určitého pohledu) a velká spousta existujících a prověřených frameworků. Hlavní nevýhodou je pomalý běh. Dále také svázání frontendu a backendu do jednoho celku.

- **Jednostránkové webové aplikace (SPA)**. Lze ho označit za moderní přístup. Jednostránková aplikace je aplikace, která běží uvnitř prohlížeče a uživatelské interakce stránku znovu nenačítají. Po prvním načtením stránky jsou už pouze na server a zpět zasílány data. SPA implementovány jazykem Javascript případně jsou použity transpilery jako CoffeScript nebo TypeScript. Hlavními výhodami je podstatně rychlejší běh, než u vícestránkových webových aplikací a oddělení frontendu a backendu.

4.7 Testování software

Testování software je nedílnou součástí jeho vývoje. Obecně je testování definováno jako *Proces získání důvěry v to, že program nebo systém dělá to, co se od něj očekává* (Hetzel 1973), nebo také *Testování je úsilí vyvinuté ke zjištění toho, zda systém pracuje tak, jak je popsáno v jeho návrhu.* (Zelinka 2007). Důležitý je jeho cíl, a to nalézt chyby, které lidský faktor do vývoje softwaru přináší. Testování také přináší další benefity, které s hlavním cílem více či méně souvisí. U software, který je dobře pokrytý testy, existuje zvýšená jistota, že prováděním změn nedojde k nežádoucím vedlejším efektům a tím pádem k chybám. Stoupá udržovatelnost zdrojového kódu a tedy i jeho celková kvalita. Testování dále může být pro vývojáře velmi motivující záležitostí z důvodu poskytování okamžité zpětné vazby k jeho práci. Tato vazba není vždy zcela zřejmá, především v případech vývoje aplikační logiky složitých systémů.

Testy nejsou ale prováděny pouze vývojáři. Kromě nich se na testování podílí také role přímo k testování určená - tester. Dále je také nutné do procesu testování zahrnout zákazníka, specificky u akceptačního testování - jednoho z druhu testování. Kritérií, podle kterých se dá testování dělit je celá řada. Jejich základní přehled je uveden na následujících řádcích.
Rozdělení dle znalosti testovaného software:

- **Bílá skříňka**. V této metodě má tester kompletní znalost testovaného software a také přístup k jeho kódu. Díky tomu je testerovi umožněno pokrýt veškeré potřebné kombinace zadáváním vhodných parametrů a jejich porovnávání s výstupy. Je také možné sledovat průběh testu. Primární výhoda tohoto přístupu je pokrytí i okrajových případů, protože tester je schopen je ze znalosti kódu odvodit a upravit dle nich vstupy.
- **Černá skříňka** Metoda černé skříňky je založena na absenci jakékoli znalosti testera o systému. Jsou mu pouze poskytnuty požadované vstupy a k nim požadované výstupy. Výhodou této metody je, že testerovo chování není nijak ovlivněno znalostí systému, a tak není fixován na zkoušení scénářů, které pravděpodobně uvažoval i programátor a systém na něj připravil. Šance na objevení chyby v chování systému ve velmi neočekávaném stavu je zde tedy vysoká.

Rozdělení dle způsobu realizace testů:

- **Manuální testování**. Testy jsou prováděné dle předem definovaných testovacích scénářů testerem. Tento způsob testování je postupně nahrazován testy automatickými. Důvody pro aplikování tohoto způsobu testování mohou být ekonomické - platit brigádníka neznalého domény ani technologií za manuální testování může vycházet levněji než platit specialistu za vytvoření testů automatických. Dalším důvodem může být speciální případ manuálních testů bez předem definovaných scénářů - průzkumné testy (*Exploratory tests*).
- **Automatizované testování**. Neboli kód kontrolující kód. Hlavním cíli jsou časová úspora a zefektivnění provádění testů. Je eliminována možnost chyby lidského faktoru. Pro tento druh testování existuje velká řada nástrojů pomocí, kterých lze testovat prakticky libovolný software na jakékoli vrstvě a rozsahu. Běžné je vytváření prostředí speciálně pro prováděný automatizovaných testů, testovacích sad dat a také strukturované výstupy vyhodnocování testů. Velkou výhodou je možnost zařazení automatizovaného testování jako krok do dalších automatizovaných procesů jako je proces nasazení software na cílové prostředí.

Rozdělení dle úrovně (nebo také stupně):

- **Testování programátorem**. Jedná se o první fázi testování kódu, která probíhá obvykle bezprostředně po jeho napsání jeho spuštěním. Provádí ho zpravidla autor kódu, ale je možné jej provádět i jiným programátorem (v rámci párového programování).
- **Jednotkové testování**. Tento typ testů se soustřeďuje na malé části softwaru - třídu nebo funkci. Jedná se o automatizované testy. Jejich tvůrci jsou obvykle vývojáři software. Tyto testy by měly být rychlé a jednoduché.
- **Integrační testování**. Testování jehož úkolem je ověřit komunikaci mezi různými části software - jde tedy o testování rozhraní modulů a volání mezi nimi.
- **Systémové testování**. Testování systému složeného z modulů jako funkčního celku z pohledu zákazníka. Jde o to určit, zda funkcionalita odpovídá návrhu. V této úrovni testování se také dále rozlišuje *funkční a nefunkční testování*, kde funkční určuje, zda je možné danou operaci v software provést a nefunkční se zabývá záležitostmi jako je uživatelská přívětivost nebo výkonnost software.
- **Akceptační testování**. Tato fáze testování probíhá po předání software zákazníkovi, který jej validuje oproti akceptačním kritériím.

4.8 Uživatelské rozhraní

Uživatelské rozhraní je důležitým tématem z oblasti *Human-Computer Interaction* (HCI), který se zabývá na vztahem mezi uživatelem a informačním systémem. Je to oblast na pomezí informatiky, psychologie, sociologie a dalších (Procházka, 2014). Důležitým pojmem je UX - *user experience*, česky uživatelská zkušenost. Jedná se o

pocity a reakce, které mají uživatelé při interakci s produktem. Uživatelská zkušenost je velmi důležitý faktor při návrhu uživatelského rozhraní.

V oblasti uživatelských prostředí existuje celá řada vzorů nejrůznějšího typu, které designerovi mohou pomoci při vytváření návrhu software. Jsou to například vzory chování uživatelů v aplikacích, vzory pro orientaci nebo vzory pro identifikaci pozice v aplikacích. Stěžejní je ale celkový přístup k designu rozhraní. Procházka (Procházka, 2014) uvádí následující přístupy, které lze kombinovat:

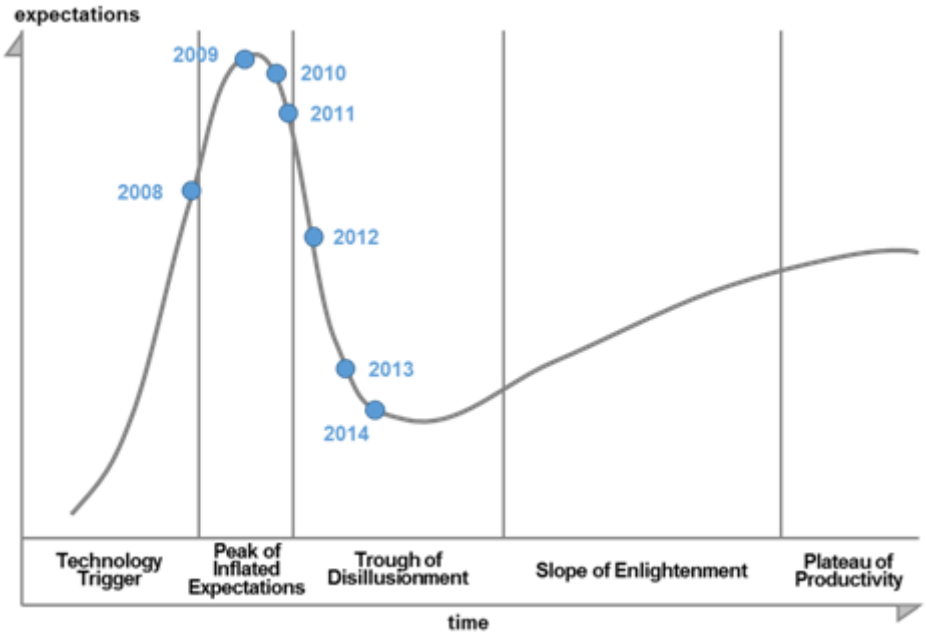
- **Design orientovaný na uživatele** (User centered design).
- **Design zaměřený provedení aktivit** (Activity centered design).
- **Design zaměřený na komponenty systému** (System centered design).
- **Design zkušeného návrháře** (Genius design).

4.9 Cloud computing

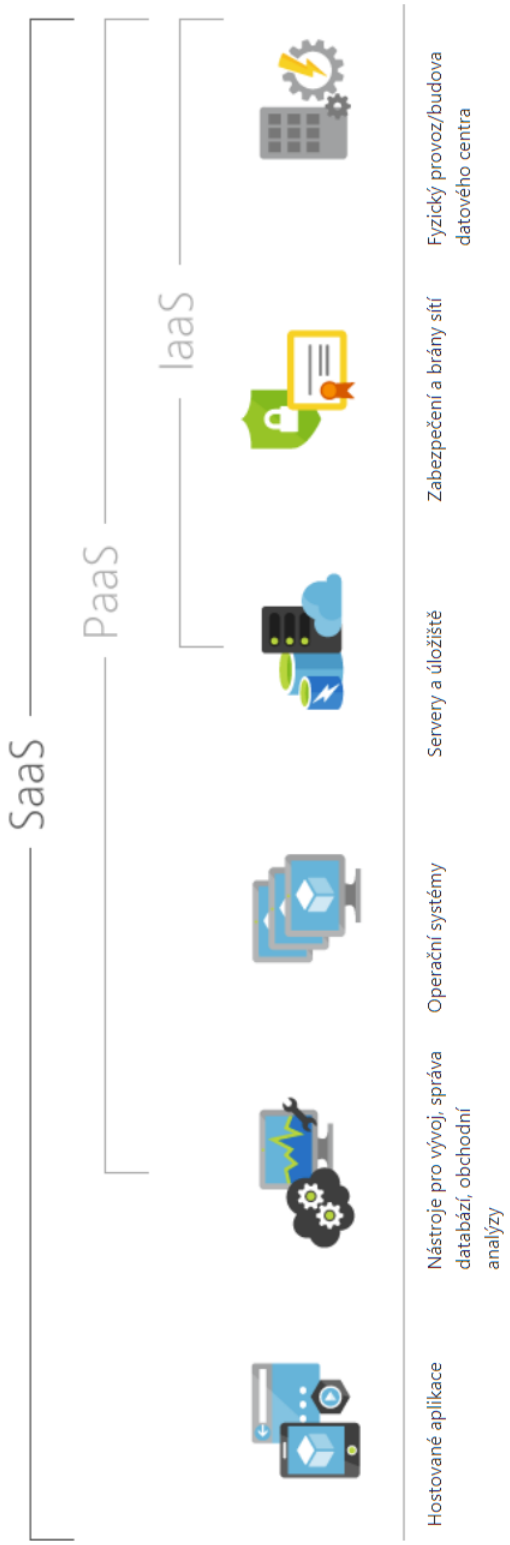
Kolem pojmu *cloud computing* počínaje rokem 2008 proběhl v IT sféře velký mediální humbuk (*hype*). Jeho průběh a také kam se technologie ubírala dál do dnešní doby je dobře vidět na obrázku průzkumu agentury Gartner (Smith, 2013) popisující životní cyklus technologie.

Název cloud computing vznikl dle symbolu obláčku, který často znázorňuje v diagramech internet, respektive "všechny ty další komponenty", za které zodpovídá někdo jiný a jenž není potřeba konkretizovat. A takový je i koncept cloudu (Velte, Elsenpeter, 2011). Prostředky cloudu jsou nabízeny jako služba dle modelů níže (Velte, Elsenpeter, 2011). Názorně pak na obrázku 7.

- **Software jako služba** (Software as a Sevice - SaaS). Je model, v kterém jsou aplikace hostovány a nabízeny jako služba zákazníkům. Ti k nim přistupují prostřednictvím Internetu. Zákazník se nemusí starat o opravy, aktualizace a servis. Také pořizovací náklady jsou menší. Z hlediska poskytovatele je výhodou snadnější ochrana duševního vlastnictví a trvalý zdroj příjmů.
- **Platforma jako služba** (Platform as a Sevice - PaaS). Tento model je velmi podobný s modelem SaaS. PaaS poskytuje všechny prostředky nutné k vytváření aplikací a služeb, tedy infrastrukturu k vývoji. Konkrétně k službám PaaS patří návrh aplikací, vývoj, testování, implementace a hostování. Nabízeny mohou být i prostředí operačních systémů. Platby jsou často vyměřovány dle využitých kapacit.
- **Infrastruktura jako služba** (Infrastructure as a Sevice - IaaS). Někdy také označovaný jako *Hardware jako služba (Hardware as a Service - HaaS)*. Nabízen je hardware, který je použitelný libovolným způsobem. Platby jsou v tomto modelu často vyměřovány dle množství uložených dat, množství přenesených dat nebo za čas procesoru.



Obr. 6: Životní cyklus technologie Cloud computing
Zdroj: (Smith, 2013)



Obr. 7: Cloud computing - modely Saas, PaaS a Iaas
Zdroj: (Smith, 2013)

5 Metodika

Kapitola pojednává o metodách vybraných k dosažení stanoveného cíle práce. Základem k vypracování metodiky bylo studium oblasti ochrany osobních údajů a zavádění GDPR v podnicích v kapitole 3 a také vytvoření přehledu v problematice vývoje cloudového software v kapitole 4.

5.1 Metodika řešení

- **Analýza požadavků a návrh řešení aplikační vrstvy systému**. Na základě znalostí z teoretické části bude provedena analýza požadavků na aplikační vrstvu portálu. Proběhne modelování systémů pomocí nástrojů UML. Stanoveny by měly být případy užití, entity, architektura ale také grafické uživatelské prostředí.
- **Analýza požadavků a návrh řešení infrastruktury**. Analýza požadavků na a vytvoření návrhu prostředí pro běh aplikační vrstvy na bázi cloudové služby. Stěžejní je definovat případy užití a architekturu infrastruktury. Při návrhu bude brán v potaz vytvořený návrh pro aplikační vrstvu.
- **Implementace systému**. Realizace vypracovaných návrhů konkrétními softwarovými nástroji. Bude popsán výběr těchto nástrojů a postup implementace. Prezentovány budou výsledky implementace.
- **Testování systému**. Systém bude testován v průběhu procesu implementace na úrovni automatizovaných jednotkových a integračních testů. Další testování bude probíhat na úrovni systémového a akceptačního testování za spolupráce vedení společnosti a budoucích uživatelů aplikace. Důležitým prvkem testování bude zavedení systému do provozu v produkčním prostředí společnosti.
- **Zhodnocení řešení a návrh rozšíření**. Vypracovaný návrh a implementace bude zhodnocen a budou stanoveny další rozšíření a směr vývoje.

5.2 Agilní přístup

Během vývoje aplikace je postup pravidelně konzultován s vedením společnosti, které je autor práce zaměstnancem a vývoj software v ní probíhá. Účelem těchto pravidelných konzultací je kromě verifikace požadovaného postupu a testování také další upřesňování a vytváření nových požadavků na aplikaci na základě nově vznikajících potřeb.

Vývoj software se tedy neřídí dle vodopádového modelu, ale jedná se o iterativní přístup. Uvedená metodika je rámec kroků, jenž jsou iterovány. V průběhu vývoje jsou například dále prohlubovány znalosti v oblasti ochrany osobních dat a nařízení GDPR a na jejich základě upravovány a zpřesňovány některé aspekty

vyvíjeného software. Dále jsou také na základě nových požadavků zvažovány a zahrnovány nové technologie a knihovny a dle nich je existující kód refaktorován a zlepšován. Směrem, kterým se vývoj tohoto softwaru vydává je tedy více vytvoření pružného a snadno modifikovatelného produktu s možností reagovat na nové potřeby než splnění konkrétních přesně definovaných požadavků.

6 Analýza a návrh aplikace

Byl definován základní koncept, na základě studia vytvořen přehled v oblasti problémové domény a souvisejících technologií a nakonec vypracována metodika. Nyní je možné přejít k analýze požadavků, vytvoření modelu a samotného návrhu řešení.

6.1 Terminologie

Na úvod této kapitoly je vhodné definovat definovat několik základních termínů, které budou dále používány nejen v této kapitole, ale i kapitole nadcházející zabývající se návrhem infrastruktury.

- **Tenant**. Termín tenant (v překladu *nájemník*, *nájemce* nebo také *uživatel*) označuje společnost pro kterou je aplikace provozována. Nejedná se tedy nutně o provozovatele aplikace, tím je poskytovatel služby, ačkoli i ten může být tenantem a v době psaní této práce tomu také tak je.
- **Poskytoval služby**. Poskytovatelem služby je společnost, která aplikaci nabízí jako službu dalším společnostem. Na serverech poskytovatele služby je nasazena infrastruktura a všechny běžící instance aplikace.
- **Kontakt**. Kontakt je označení subjektu zpracovávaných osobních údajů. Tedy jedná se o jednu specifickou unikátní fyzickou osobu, jejíž osobní údaje a souhlasy mohou být v systému.
- **Atribut**. Atributem se dále rozumí typ osobního údaje, který je zpracováván. Jako příklad může být atributem emailová adresa.
- **Záznam**. Záznam je konkrétního hodnota daného atributu. Záznamem tak může například konkretní emailová adresa "petr.novak@email.cz" vyplněná ve formulářovém poli pro atribut emailová adresa. Je tedy zřejmé, že kombinace atributu a záznamu tvoří osobní údaj.
- **Typ souhlasu**. Jedná se o typ udělovaného souhlasu, který je v systému definován. Kromě souhlasu se zpracováním osobních dat se může jednat například o souhlas se zasíláním emailových nabídek.
- **Souhlas**. Termín souhlas je dále v textu použit jako konkrétní vyjádření hodnoty pro určitý typ souhlasu. Může nabývat hodnot ano (resp. *souhlas udělen*) nebo ne (resp. *souhlas neudělen*), který je také vyjádřením zamítnutí.
- **Systém zpracování**. Systémem zpracování se rozumí externí (tedy v rámci vyvíjeného software, nikoli nezbytně v rámci podniku) systém, v kterém jsou osobní údaje evidovány. Příkladem může být konkrétní emailingový software, který automatizovaně rozesílá obchodní nabídky.

6.2 Specifikace požadavků

Funkční požadavky

- **Přihlášení a registrace kontaktů**. Přihlášení do rozhraní klienta. Přihlášení by mělo být umožněno jakýmkoli zvoleným unikátním identifikátorem.
- **Přístup kontaktů k vlastním osobním údajům a souhlasům**. Tedy zobrazení veškerých dat, které o kontaktu daná společnost udržuje.
- **Modifikace vlastních osobních údajů a souhlasů kontaktů**. Tedy editace již existujících dat, které o subjektu společnost udržuje, možnost jejich smazání nebo i přidání osobních údajů. Modifikovat by mělo být také umožněno souhlasy, a to nejen vyjádření souhlasu/nesouhlasu, ale také změny expirace souhlasu nebo i možnosti nahrání přílohy k souhlasu.
- **Zobrazení za jakým účelem jsou vlastní osobní data zpracovávána** Subjekt osobních údajů by měl mít možnost zobrazit za jakým účelem jsou jeho údaje společností zpracovávány.
- **Export vlastních osobní údajů kontaktu**. V souladu s *právem na přenositelnost* dle GDPR. Export by měl být umožněn do strojově čitelného formátu.
- **Možnost být zapomenut**. V souladu s *právem na zapomenutí* dle GDPR. Při využití této možnosti by měl být spuštěn proces na straně správce, který povede k vymazání/anonymizování všech údajů o kontaktu a potvrzení této skutečnosti na adresu kontaktu.
- **Přihlášení/odhlášení správce**. Přihlášení do rozhraní organizace zcela odděleného od klientského rozhraní pod rolí správce.
- **Přihlášení/odhlášení zpracovatele**. Stejné přihlášení do rozhraní organizace zcela odděleného od klientského rozhraní pod rolí zpracovatele.
- **Import kontaktů správcem**. Importování kontaktů z prostředí grafického uživatelského prostředí prostřednictvím souboru ve strojově čitelného formátu.
- **Export kontaktů správcem**. Ve stejném formátu jako při exportu, který má jako možnost kontakt samotný. Správce by měl mít možnost exportovat kontakty hromadně a dle filtrů.
- **Modifikace osobních údajů a souhlasů kontaktů správcem**. Správce by měl mít možnost modifikovat veškeré data v systémech. Taková modifikace je ale vždy oznámena danému kontaktu - subjektu modifikovaných osobních údajů.
- **Přístup zpracovatele k dotčeným osobním údajům a souhlasům**. Zpracovatel by měl mít přístup k osobním údajům a souhlasům, které zpracovává. Přístup se týká také historických údajů.

- **Administrace atributů a typů souhlasů správcem**. Důležitým požadavkem je možnost správce nakonfigurovat vlastní schéma atributů a typů souhlasů. Více o této konfiguraci je napsáno v sekci o konfigurovatelnosti - 6.2
- **Administrace účelů a systému zpracování, zpracovatelů a právních důvodů správcem**. S předchozím požadavkem také úzce souvisí požadavek na možnost nakonfigurování účelů zpracování a dalších náležitostí, jenž se z logického hlediska i právního hlediska k účelu zpracování vážou a tím jsou systémy zpracování, zpracovatelé a právní důvody ke zpracování. I o této konfiguraci je více informací napsáno v sekci o konfigurovatelnosti - 6.2
- **Administrace proměnných systému**. Jedná o proměnné, které dané instanci aplikace dávají identitu daného tenanta. Jedná se například o název společnosti, který se dále může vyskytovat v klientském rozhraní nebo emailech. Dále třeba logo, znění odchozích emailů a podobně.
- **Zasílání notifikací**. V reakci na různé události v systému jsou zasílány emailové notifikace. Příkladem může být změna záznamu určitého atributu provedená kontaktem a následná notifikace zpracovatele (nalezeného dle účelu zpracování) daného atributu o provedené změně.
- **Přívětivé uživatelské prostředí**. Systém by měl poskytnout uživatelům jasnou navigaci a jednotný grafický styl.
- **Evidování akcí**. Provedené akce v systému jako jsou změny dat, nebo odeslaných notifikací jsou evidovány. Taková evidence je žádoucí pro případný GDPR audit.
- **Poskytnutí API**. Velmi důležitý požadavek je možnost ovládání celého systému přes rozhraní, tedy API. Důležitost tohoto požadavku tkví v požadavcích na centralizovanost řešení a plánované systémové integraci.

Nefunkční požadavky

- **Důraz na uživatelskou zkušenost**. Rozhraní aplikace by se mělo držet zásad v oblasti uživatelské zkušenosti (UX) a poskytnout snadné a rychle nalezení požadovaných informací či jasné a důvěryhodné provedení akcí.
- **Oddělení byznys logiky a prezentační vrstvy**. Byznys logika by měla být v maximální míře oddělená od grafického uživatelského prostředí. Důvodem je konsolidace byznys logiky na jedno místo a také izolace klientů od přímého přístupu k ní a také od databáze.
- **Výkonnost**. Systém je připraven operovat s objemy dat v řádu desetitisíců uložených záznamů dat. Při zpracování velkého objemu dat by nemělo nejen docházet k žádnému selhání systému, ale také by nemělo docházet ke zhoršení uživatelské zkušenosti.

- **Řízení uživatelských rolí**. V systému operuje určitá množina uživatelských rolí na základě kterých je řízen přístup k různým funkcím a datům.
- **Použití vhodných nástrojů**. Při volbě programovacích jazyků, knihoven a frameworků by mělo být přihlíženo na to, zda jejich volbou dojde nejen k vytvoření řešení, ale také na na další záležitosti související s udržovatelnosti a rozšiřovatelnosti vývoje. Například tedy zda je programovací jazyk nebo framework standardem v dané oblasti nebo zda je daná technologie *živá* - myšleno jestli je dále vyvíjena a podporována ať už ze strany komunity nebo zaštiťující společnosti.
- **Testovatelnost**. Řešení by mělo být snadno testovatelné, a to na různých úrovních testování. Tomuto je potřeba přizpůsobit volbu technologií.
- **Multiplatformnost**. Požadavek na multiplatformnost zahrnuje možnost nasazení aplikace na různé platformy. Například nevázání se na ekosystém jednoho výrobce (Microsoft) a volbu technologií, které umožní snadnou výměnu různých dalších částí řešení. Druhý požadavek na multiplatformnost je z pohledu klientského rozhraní. Klienti musí mít svobodnou volbu prostředí, ze kterého do aplikace mohou vstoupit. V případě implementace klientského rozhraní pomocí webové technologie by to pak byla široká škála podporovaných prohlížečů.

Konfigurovatelnost

Požadavek na konfigurovatelnost je požadavkem zcela zásadním. Vychází ze skutečnosti, že různé společnosti mohou mít velmi rozdílné schéma zpracování osobních údajů. Tento požadavek byl zaznamenán již při definování konceptu dle několika skutečností.

Tou první byl meeting v kterém zástupce společnosti Y (jiné společnosti než té v které je autor práce zaměstnancem - dále společnost X) projevil o zamýšlený portál zájem a bylo s ním projednáno jaká a jaké množství osobních dat zpracovávají. Některé atributy, které daná společnost zpracovává, nebyly zahrnuty v zamýšlené množině atributů vycházející z potřeb společnosti X a naopak většina atributů společnosti X by byla pro společnost Y v systému zcela zbytečně. K tomu se navíc přidal velmi zajímavý požadavek ze strany společnost Y a to možnost přihlašování klientů do systému na základě atributu, který nebyl společností X vůbec uvažován - konkrétně šlo o IČO. To se stalo základem idee mít nejen přidávatelné/odebíratelné atributy, ale také možnosti volit parametry atributů. Takovým parametrem by pak byla možnost zvolit u určitého atributu, že jeho hodnoty mají být unikátní a na jeho základě se mohou klienti identifikovat a přihlásit.

Další událostí bylo vytvoření velmi jednoduchého prototypu aplikace pro testování a pilotního běhu ve společnosti X. V této verzi nebyla možnost konfigurace vůbec zahrnuta. V krátkém časovém intervalu bylo po představení aplikace zjištěno, že i přes úvodní analýzu nebyla zcela přesně definována množina atributů, které ve skutečnosti je potřeba evidovat. Ta samá situace nastala u typů souhlasů, kde došlo

k nutnosti hned několika úprav. Všechny tyto akce vyžadovaly drobné úpravy zdrojového kódu. Při správném návrhu datových struktur a vytvoření odpovídajícího uživatelského prostředí by ale takové změny mohly být provedeny přímo správcem aplikace. V některých případech šlo totiž jen o prostou změnu popisku atributu, nebo změnu v odkaze, odkazující na dokument s podrobnějšími informaci o daném typu souhlasu.

V předchozích odstavcích byly uvedeny záležitosti ohledně potřeby konfigurovatelnosti atributů a typů souhlasů. Velmi důležitou, ne-li důležitější, potřebou v rámci konfigurovatelnosti je ale potřeba konfigurovat účely zpracování. Tam, kde by se u atributů dal problém s různými atributy, které různé společnosti evidují, řešit snahou o vytvoření velkého výčtu unifikovaných atributů, je potřeba individuálního nastavení účelů zpracování zcela zřetelná. Různé společnosti nejenže osobní data zpracovávají pro různé účely, ale také ve většině případů ve zcela různých systémech. Naprosto individuální jsou pak pro různé společnosti zpracovatelé. Také účely zpracování mohou být určeny na základě různých právních důvodů.

Na základě uvedených skutečností a případů bylo tedy rozhodnuto navrhnout aplikaci tak, aby jeho datové schéma bylo jednoduše konfigurovatelné z prostředí rozhraní organizace správcem.

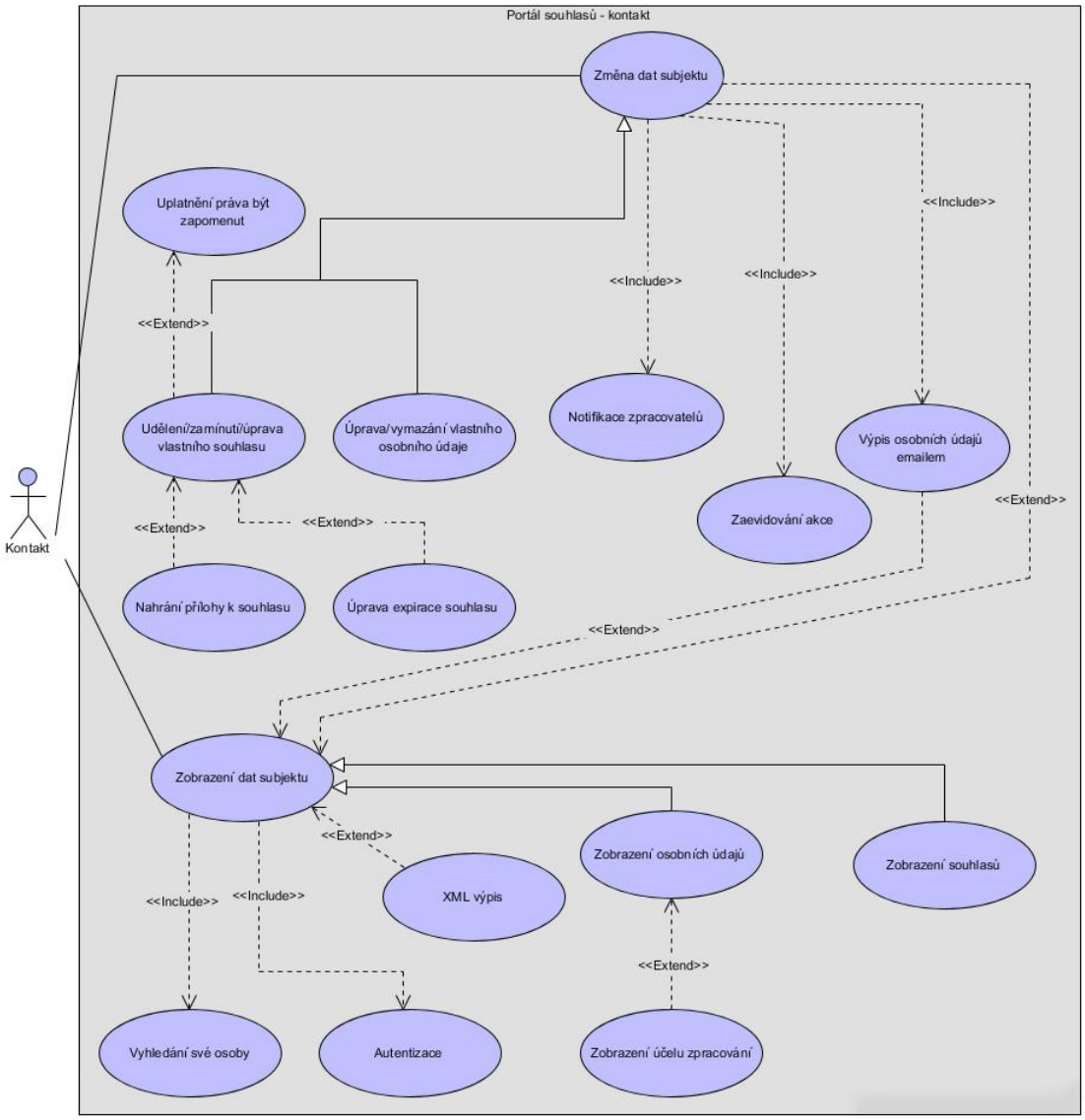
6.3 Případy užití a chování systému

Na základě výše definovaných požadavků a entit lze přejít k modelování chování systému. K podpoře textového popisu byly vybrány nástroje jazyka UML diagram případů užití (use case diagram), případy užití (use case), diagram stavů (state machine diagram) a diagramy aktivit (activity diagram).

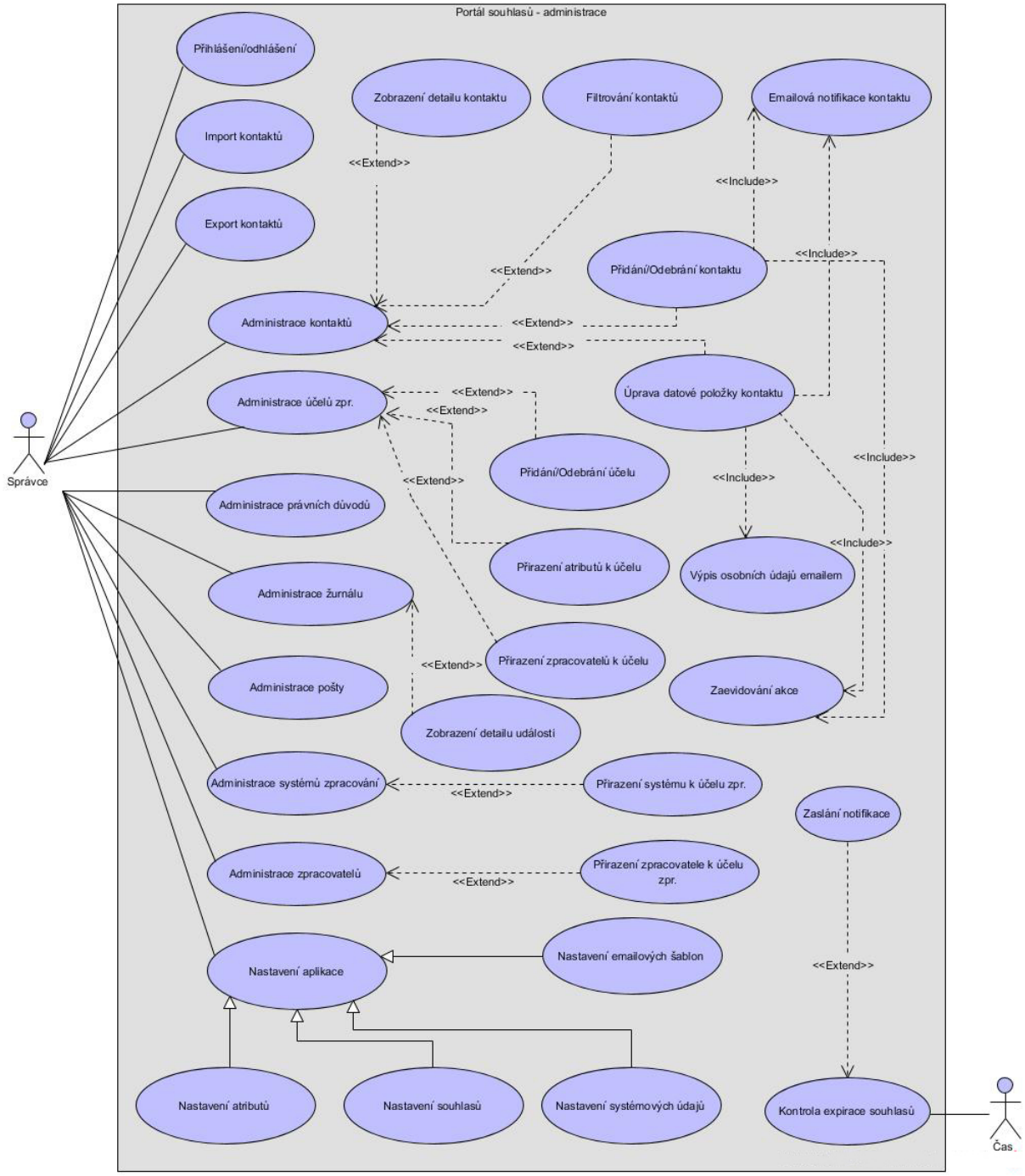
Diagramy případů užití

V rámci návrhu jsou pro aplikační vrstvu vypracovány tři diagramy případů užití. Důvodem rozdělení do tří diagramů je velká komplexnost celého systému a velmi špatná přehlednost diagramu v případě, že by byly veškeré případy užití zahrnuty do jednoho. Rozdělení je také provedeno z logického hlediska na použití systému z pohledu klienta (diagram případů užití na obrázku 8), použití systému z pohledu správce (diagram případů užití na obrázku 9) a použití systému z pohledu zpracovatele (diagram případů užití na obrázku 10).

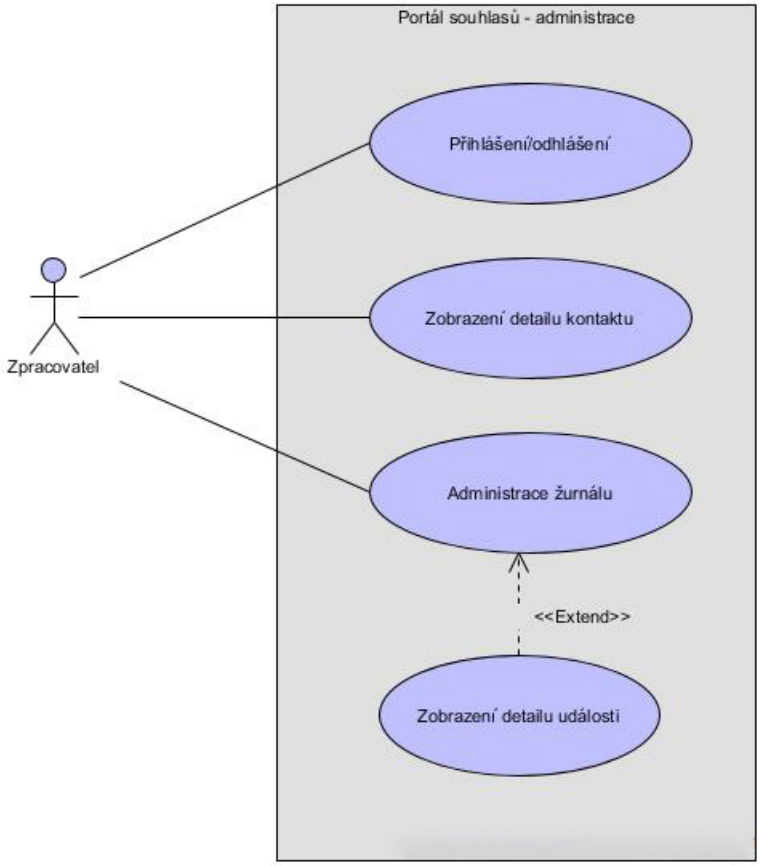
Rozpoznanými aktory v interakcích se systémem jsou tedy: *správce*, *zpracovatel*, *kontakt* (představující subjekt osobních údajů) a také speciální aktor *čas*. Ten je uveden z důvodu akcí, na které systém reaguje a jsou vyvolané na základě času, nikoli tedy akcí konkrétní osoby.



Obr. 8: Diagram případů užití - pohled subjektu zpracovávaných osobních údajů



Obr. 9: Diagram případů užití - pohled správce



Obr. 10: Diagram případů užití - pohled zpracovatele

Stavy kontaktu a vyřazení z evidence

Kontakt se v systému může nacházet v několika stavech. Stavy a přechody mezi nimi jsou vyobrazeny na diagramu stavů na obrázku 11 Stav *platný kontakt* je stavem výchozím, který vzniká zavedením kontaktu do evidence. Dalšími stavy jsou stav *kontakt v procesu zapomenutí* a *smazaný kontakt*.

Případ užití UCA001: Uplatnění práva být zapomenut (tabulka 1) popisuje první přechod stavů. K tomu dochází pokud kontakt, resp. subjekt zpracovávaných dat po přihlášení do klientského rozhraní zamítne souhlas, který byl definován jako souhlas primární. V takovém případě je spuštěn proces zapomenutí, který odpovídá právu na zapomenutí dle GDPR. Kontakt se dostává do stavu *v procesu zapomenutí* a správce dostává stanovenou lhůtu na to jej odstranit. Kontakt z tohoto stavu může přejít do stavu *smazaný kontakt* a nebo se na základě stornování procesu zapomenutí může dostat zpět na původní stav. Lhůta na smazání kontaktu slouží k tomu, aby měla společnost čas na požadavek reagovat formou vymazání osobních údajů nejen z portálu, ale také z ostatních systémů v podniku. Akce, které jsou provedeny po potvrzení smazání kontaktu, jsou popsány na diagramu aktivit na obrázku 13. Tím se kontakt dostává do stavu *smazaný kontakt*. Fyzicky v takovém stavu již kontakt neexistuje, ale vzhledem k požadavku evidování akcí k němu mohou zůstat v systému navázány nějaké jiné data. Vše, co není smazáno, musí být anonymizováno tak, aby nebyl subjekt údajů rozeznatelný.

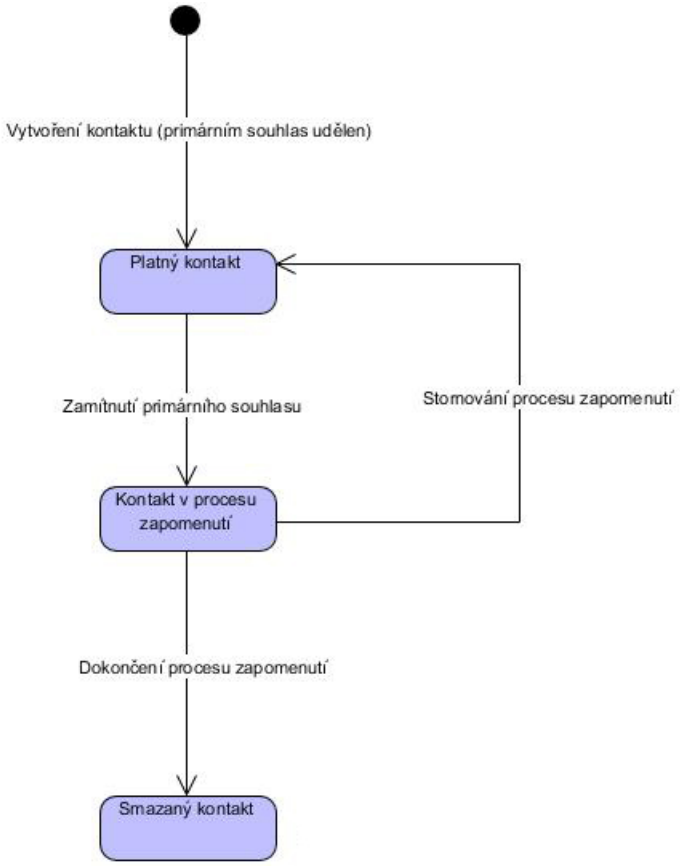
Přihlášení a registrace

Přihlášení správce a zpracovatele do systému probíhá přes jejich přihlašovací údaje zavedené v systému. Jiná situace je u rozhraní klienta. Ten má možnost se do rozhraní klienta přihlásit pomocí identifikátoru. Identifikátor je jakýkoli atribut, který je v systému nastavený jako přihlašovací atribut. Samozřejmě, že hodnoty takového atributu v systému musí být unikátní. Defaultně je v systému jako přihlašovací atribut emailová adresa. Emailová adresa je atribut, který je u kontaktu nutný udržovat vždy, aby s ním vůbec mohl systém pracovat. Kromě notifikací je totiž email používán k autorizaci. Po zadání identifikátoru je klientovi na jeho emailovou adresu odeslán náhodně generovaný přihlašovací kód pomocí kterého se v druhém kroku přihlášení musí prokázat. Tento kód je následně i použitý jako heslo k výpisu osobních údajů a souhlasů, který klient obdrží. Kód je v systému platný po dobu půl hodiny a následně v systému zanikne. Jak přesně probíhá přihlášení do systému popisuje diagram aktivit na obrázku 12.

Proces registrace je s přihlášením totožný s tím rozdílem, že aby mohlo k registraci dojít, je nutné zadat emailovou adresu, na kterou bude zaslán aktivační kód. Následně je klient přesměrován na stejnou obrazovku, jako klient po přihlášení pouze s rozdílem, že jediným evidovaným údajem je právě zadaná adresa. Veškeré souhlasy jsou po prvotním přihlášením nastaveny jako zamítnuté (dle definice podle zákona)

Tab. 1: Případ užití - Uplatnění práva být zapomenut

Název případu užití	Uplatnění práva být zapomenut
Identifikátor případu užití	UCA001
Cíl případu užití	Odstranění či anonymizování osobních údajů
Primární aktér	Klient
Sekundární aktér	Zpracovatelé
Vstupní podmínky	Klient je přihlášen v klientském portálu, odpovídající kontakt ještě není v procesu zapomenutí, odpovídající primární souhlas je udělen
Výstupní podmínky	Je spuštěn proces zapomenutí, akce je evidována v žurnálu, jsou odeslané příslušné upozornění
Popis scénáře	
1. Klient na prvku indikující zamítnutí/udělení souhlasu se souhlasem, který je nastavený jako primární zvolí možnost zamítnout 2. Systém zobrazí klientovi upozornění v dialogovém okně 3. Klient upozornění potvrdí 4. Systém zaznamená změnu hodnoty primárního souhlasu 5. Klient klikne na tlačítko pro uložení změn 6. Klient je systémem přesměrován na odhlášení 7. Systém spustí proces zapomenutí u daného kontaktu 8. Systém zaeviduje požadavek do žurnálu 9. Systém odešle výpis kontaktu na emailovou adresu klienta 10. Systém odešle potvrzení o uplatnění práva být zapomenut na email klienta 11. Systém odešle upozornění provázaným zpracovatelům na jejich email	



Obr. 11: Diagram stavů - Stavy kontaktu

a nově registrovaný klient je má možnost udělit. Pokud tak neučiní, je následně jeho kontakt označen k vymazání.

Administrace pošty

Správce má možnost v rozhraní organizace administrovat poštu. Poštou se myslí pošta elektronická, tedy e-mail. To zahrnuje kromě procházení odchozích e-mailů také úpravu jejich šablon. V systému se nachází množina typů e-mailů, jenž ze systému odchází při určitých událostech (v textu také jako *notifikace*) a jejich obsah lze měnit. V obsahu by měly být použitelné proměnné typu název společnosti nebo adresát. Pokud by tedy společnost chtěla například v těle emailu s aktivačním klíčem poskytnout klientovi odkaz na jejich stránky, nebo na třeba návod a podmínky použití jejich rozhraní, lze to provést v administraci pošty.

Současný návrh nepočítá s možností definovaní a přiřazování vlastních typů e-mailových notifikací k různým událostem.

Úprava záznamu nebo souhlasu

Základní funkcí systému je upravení záznamu nějakého atributu, nebo souhlasu (respektive vyjádření své vůle) k nějakému typu souhlasů. Upravením se myslí i zadání nové hodnoty, tam kde ještě nebyla, nebo odstranění hodnoty. Obvyklým případem je, že tuto akci provede subjekt zpracovávaných dat, tedy kontakt. V systému ale existuje i možnost úpravy dat správcem. Ten má práva upravit libovolnou položku libovolného kontaktu. Tato možnost je správci poskytnuta pro takové případy, kdy ke změně dojde jinou cestou, než je cesta klientského rozhraní. Může to být například telefonát, v kterém se daná osoba kladně vyjádří k určitému typu souhlasu. Aby taková změna správcem nezůstala klientovi neznámá, při upravení je možnost klientovi zaslat o úpravě notifikaci. Samozřejmostí je notifikování zpracovatelů.

Nastavení systému

Případ užití UCA004: Přiřazení atributu k účelu zpracování (tabulka 4) je popis jedné z několika možnosti vytváření vazeb mezi entitamy. Přiřazení zpracovatelů k účelu zpracování, přiřazení systému zpracování k účelu zpracování a přiřazení právního důvodu k účelu zpracování jsou obdobné akce. Analogické je samozřejmě i odebrání těchto vazeb. Zmíněné případy užití a **případ užití UCA003: Nastavení systémových proměnných** (tabulka 3) tvoří konfigurovatelnost systému.

Kromě vytváření vazeb je také možné nastavení vlastností atributů a typů souhlasů. Příkladem je **případ užití UCA005: Přiřazení atributu k účelu zpracování** (tabulka 5).

Další možností, s kterou se v rámci konfigurace počítá, je přidávání, upravování a mazání entit. Příkladem je **případ užití UCA006: Odstranění atributu** (tabulka 6), kde se předpokládá úspěšný scénář. Smazání atributu je také popsáno na diagramu aktivit na obrázku 13, kde krok *Odstranění atributu* je ještě rozveden na obrázku 14. Operace s entitamy jsou evidovány a také je při některých z nich vhodné dotčené osoby (ať už zpracovatele nebo subjekty dat) notifikovat.

Zabezpečení

V rámci určitého zabezpečení systému z pohledu jeho designu je k výše popsaným případům užití nutné dodat některé další prvky návrhu. V žádné z emailových notifikací se nevyskytují osobní údaje přímo v těle zprávy. Pokud je například zpracovatel notifikován o změně osobního údaje u kontaktu X, v těle zprávy je zmíněna pouze událost změny údaje a odkaz do systému. V systému se lze dozvědět konkrétní změnu hodnoty z přímo detailu daného kontaktu, nebo z evidence akcí. V evidenci je také zapsána předchozí hodnota. Důvodem tohoto návrhu je především anonymizace v případě odstranění kontaktu. Po vyhovění žádosti uplatnit právo na zapomenutí, tedy odstranění kontaktu ze systému, by nemělo v systému (a i mimo něj) zbýt nic, z čeho by byl klient určitelný. Toho je dosaženo tím, že neodstraněné entity, které se ke klientovi vážou, jsou anonymizovány.

Tab. 2: Případ užití - Úprava záznamu správcem

Název případu užití	Úprava záznamu správcem
Identifikátor případu užití	UCA007
Cíl případu užití	Upravení daného záznamu kontaktu
Primární aktér	Správce
Sekundární aktér	-
Vstupní podmínky	Správce je přihlášen v portálu správce, existuje daný kontakt
Výstupní podmínky	Daný záznam je upraven, je odeslána notifikace
Popis scénáře	
1. Správce otevře detail daného kontaktu 2. Systém vypíše všechny souhlasy a záznamy daného kontaktu 3. Správce upraví daný záznam a změnu uloží 4. Systém změnu zavede 5. Systém nabídne možnost notifikování klienta 6. Správce notifikaci potvrdí 7. Systém vygeneruje výpis osobních údajů a souhlasů klienta 8. Systém odešle notifikaci s výpisem klientovi	

Dalším prvkem zabezpečení osobních údajů je šifrování souborů, které jsou přílohami e-mailových zpráv. Jak bylo zmíněno, v těle zpráv se žádné osobní údaje nevyskytují, ale mohou se objevit ve výpisu, který je zasílán prostřednictvím e-mailu.

Tab. 3: Případ užití - Nastavení systémových proměnných

Název případu užití	Nastavení systémových proměnných
Identifikátor případu užití	UCA003
Cíl případu užití	Upravení systémových proměnných portálu
Primární aktér	Správce
Sekundární aktér	-
Vstupní podmínky	Správce je přihlášen v portálu správce
Výstupní podmínky	Provedené úpravy jsou potvrzeny, klientská aplikace i aplikace organizace reflektují dané změny
Popis scénáře	
1. Správce otevře obrazovku s nastavením systémových údajů 2. Správce změní systémové údaje 3. Správce uloží provedené změny 4. Systém uloží nastavenou konfiguraci do databáze 5. Systém upraví své parametry dle uložených změn	

Tab. 4: Případ užití - Přiřazení atributu k účelu zpracování

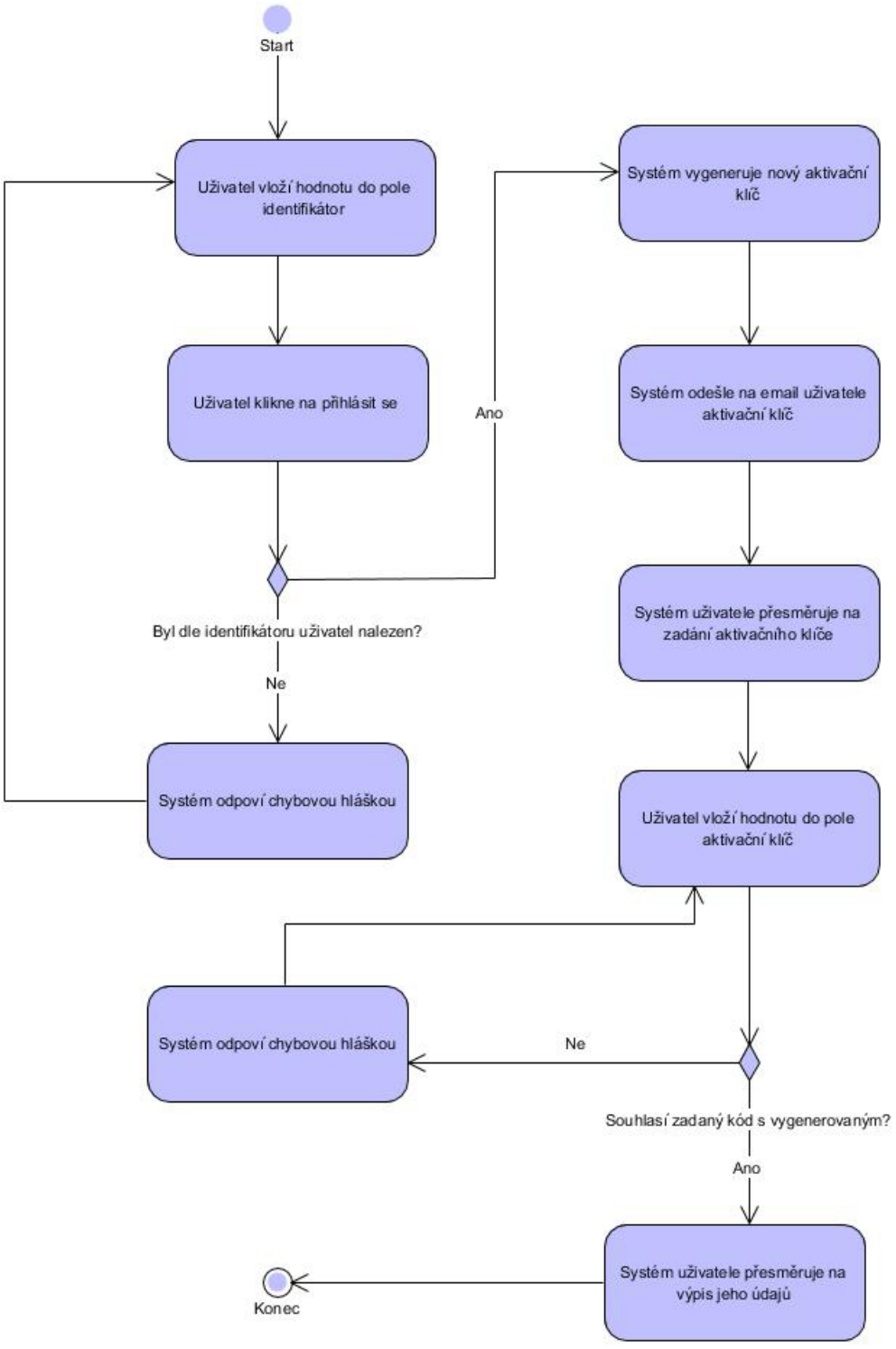
Název případu užití	Přiřazení atributu k účelu zpracování
Identifikátor případu užití	UCA004
Cíl případu užití	Mezi atributem a účelem zpracování je vytvořena vazba
Primární aktér	Správce
Sekundární aktér	-
Vstupní podmínky	Správce je přihlášen v portálu správce, existuje daný atribut, existuje daný účel zpracování, neexistuje daná vazba
Výstupní podmínky	Mezi atributem a účelem zpracování je vytvořena vazba
Popis scénáře	
1. Správce otevře obrazovku s nastavením účelů zpracování 2. Systém vrátí seznam definovaných účelů zpracování 3. Správce vybere daný účel zpracování 4. Systém nabídne možnosti akcí s daným účelem zpracování 5. Správce zvolí možnost přiřazení atributů 6. Systém seznam definovaných atributů 7. Správce zvolí daný atribut a změny potvrdí 8. Systém provedené změny uloží do databáze 9. Systém promítne nově uloženou vazbu ve všech svých částech	

Tab. 5: Případ užití - Nastavení přihlašovacího atributu

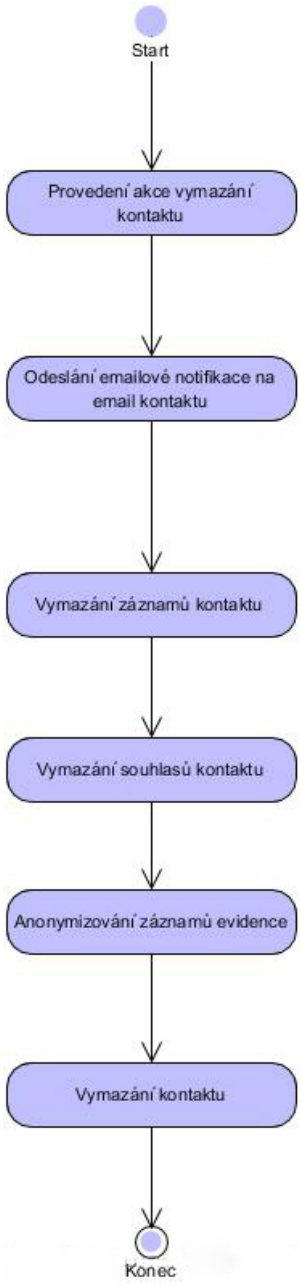
Název případu užití	Nastavení přihlašovacího atributu
Identifikátor případu užití	UCA005
Cíl případu užití	Nastavit zvolený atribut jako přihlašovací atribut
Primární aktér	Správce
Sekundární aktér	-
Vstupní podmínky	Správce je přihlášen v portálu správce, existuje daný atribut, daný atribut není nastavený jako přihlašovací
Výstupní podmínky	V klientském rozhraní je pomocí daného atributu možné se přihlásit
Popis scénáře	
1. Správce otevře obrazovku se seznamem definovaných atributů 2. Systém vrátí seznam definovaných atributů 3. Správce vybere daný atribut 4. Systém nabídne možnosti akcí s daným atributem 5. Správce zvolí možnost označit atribut jako přihlašovací 6. Systém zkontroluje, zda jsou všechny záznamy daného atributu unikátní 7. Systém otevře dialogové okno s potvrzením akce 8. Správce potvrdí svoji volbu 9. Systém nastaví daný atribut jako přihlašovací 9. Systém promítne změnu vlastnosti atributu ve všech svých částech	

Tab. 6: Případ užití - Odstranění atributu

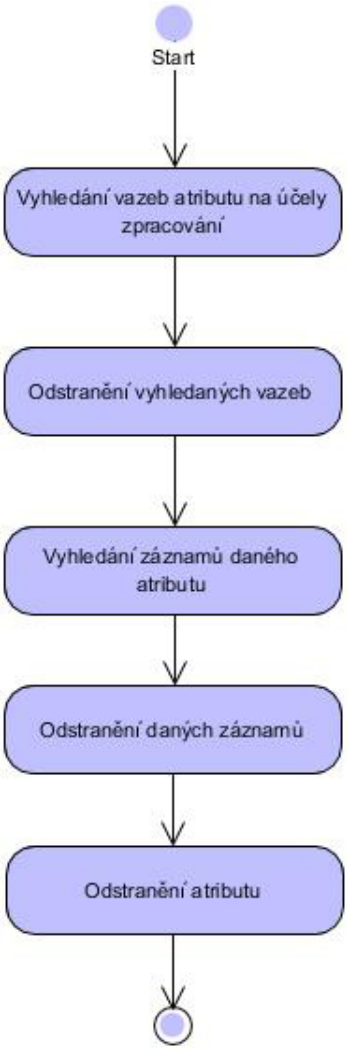
Název případu užití	Odstranění atributu
Identifikátor případu užití	UCA006
Cíl případu užití	Vymazat zvolený atribut ze systému
Primární aktér	Správce
Sekundární aktér	-
Vstupní podmínky	Správce je přihlášen v portálu správce, existuje daný atribut, daný atribut není nastavený jako jediný přihlašovací
Výstupní podmínky	Atribut je smazán ze systému, všechny záznamy a vazby daného atributu jsou smazány ze systému, zpracovatelé jsou notifikování, kontakty jsou notifikovány
Popis scénáře	
1. Správce otevře obrazovku se seznamem definovaných atributů 2. Systém vrátí seznam definovaných atributů 3. Správce vybere daný atribut 4. Systém nabídne možnosti akcí s daným atributem 5. Správce zvolí možnost odstranění atributu 6. Systém zkontroluje, zda není atribut jediným přihlašovacím atributem 7. Systém spočítá počet záznamů daného atributu v systému 8. Systém otevře dialogové okno s potvrzením akce a varováním, že smazáním dojde k odstranění spočteného počtu záznamů 9. Správce potvrdí svoji volbu 10. Systém nastaví smaže daný atribut 11. Systém zaeviduje akci zaeviduje 12. Systém dle vazeb notifikuje dotčené zpracovatele 13. Systém otevře dialogové okno s možností notifikovat dotčené kontakty 14. Správce zvolí potvrdí okno 15. Systém dle vazeb notifikuje dotčené kontakty	



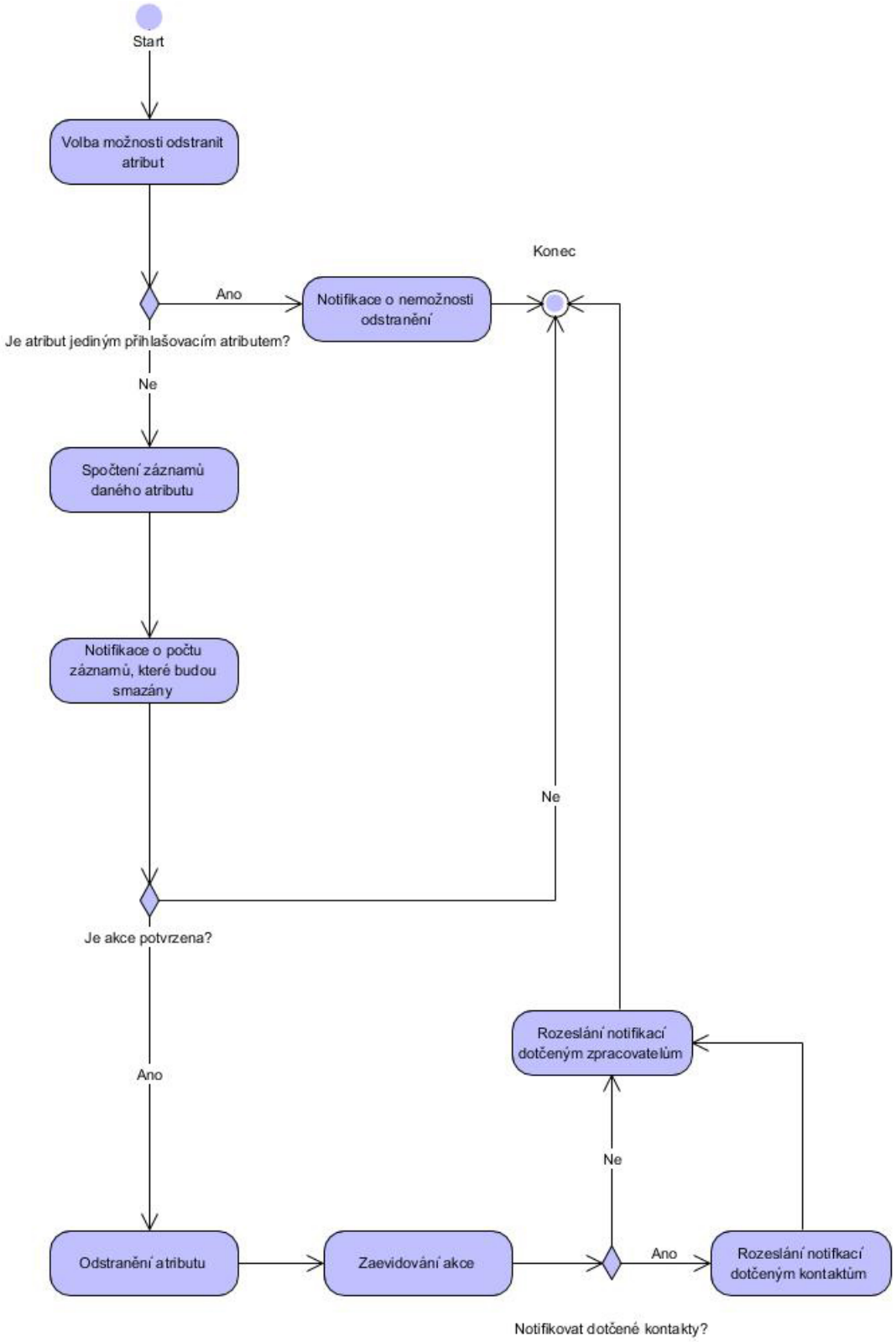
Obr. 12: Diagram aktivit - Přihlášení kontaktu



Obr. 13: Diagram aktivit - Provedení smazání kontaktu



Obr. 14: Diagram aktivit - Odstranění atributu, detail



Obr. 15: Diagram aktivit - Odstranění atributu, obecně

Tab. 7: Případ užití - Export kontaktů

Název případu užití	Export kontaktů
Identifikátor případu užití	UCA002
Cíl případu užití	Vytvoření a stažení strojově čitelného souboru se všemi informacemi o daných kontaktech
Primární aktér	Správce
Sekundární aktér	-
Vstupní podmínky	Správce je přihlášen v portálu správce, v aplikaci jsou instance kontaktů a jejich záznamů, v aplikaci jsou instance atributů
Výstupní podmínky	Je vytvořen soubor exportu
Popis scénáře	
1. Správce otevře seznam kontaktů 2. Správce klikne na možnost export kontaktů 3. Systém vytvoří soubor obsahující informace o daných kontaktech 4. Systém odešle soubor správci 5. Správce soubor obdrží a uloží si jej	

6.4 Entity a jejich vazby

Důležitým krokem je určení entit v sytému a vazeb mezi nimi. Vychází se zde z některých pojmů definovaných v GDPR (více v sekci pojmy kapitoly o GDPR - 3.2) a také ze zavedené terminologie na začátku této kapitoly - 6.1. Základní entity v systému je popisuje následující množina. Jejich definice odpovídá definicím ve uvedených sekcích práce.

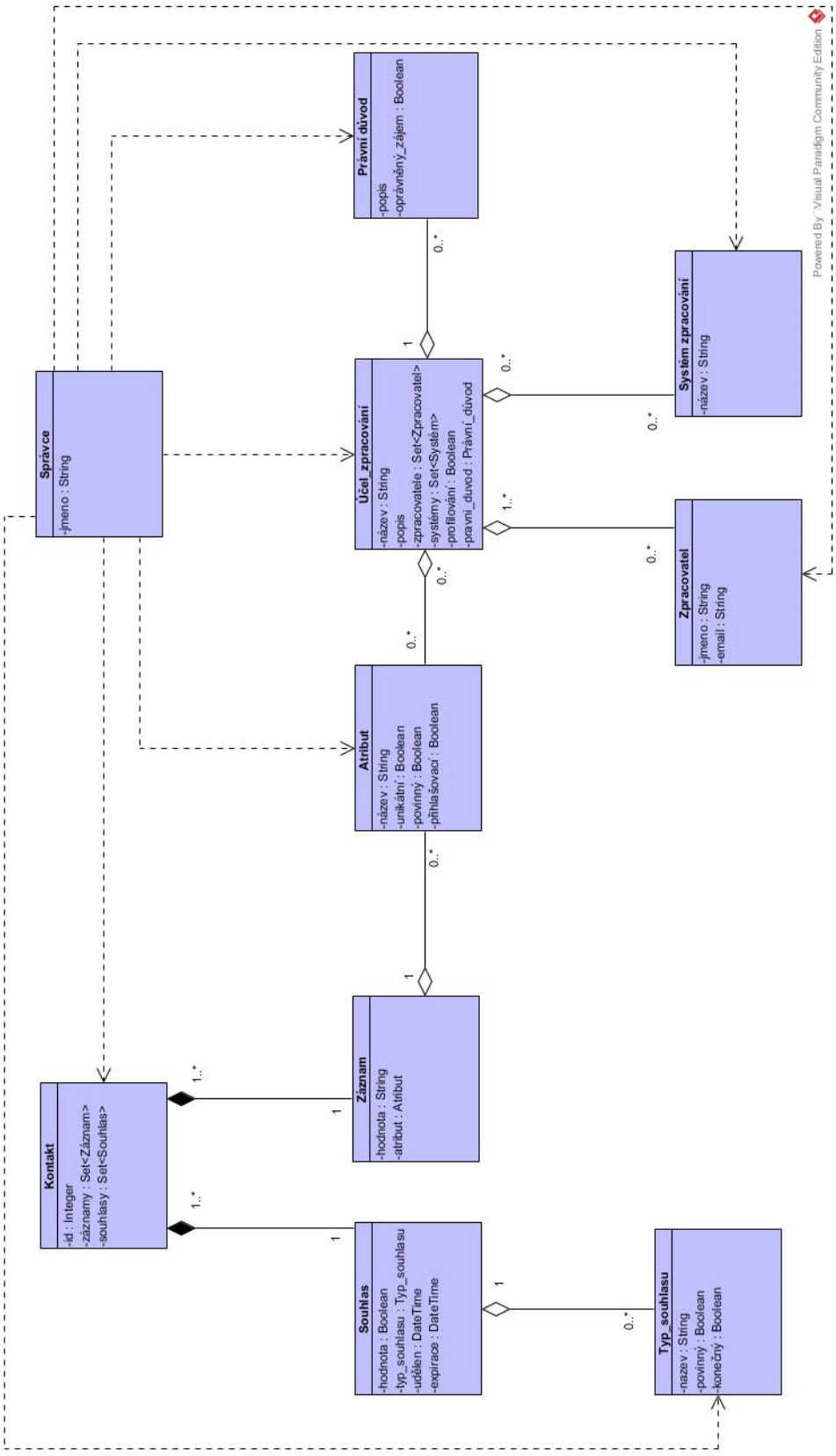
- **Správce**. Termín správce v návrhu aplikace je svázaný s pojmem správce v terminologii GDPR a je k ní navázána uživatelská role správce aplikace ve smyslu uživatele aplikace s administrátorskými právy. Každý tenant má určeného správce. Správce má přístup do rozhraní organizace.
- **Kontakt**. Jedná se o uživatelskou roli klienta, fyzické osoby jejíž účet vzniká existencí jejích osobních dat v systému.
- **Souhlas**. Entita odpovídající popisu v sekci terminologie - 6.1. Váže se ke kontaktu vztahem kompozice.
- **Typ souhlasu**. Entita odpovídající popisu v sekci terminologie - 6.1. Má vlastnosti jako povinnost (bez vyjádření se k tomuto typu souhlasu není možné daný kontakt evidovat) nebo konečnost (zda se jedná o souhlas jehož platnost je ohraničená nějakým datem). Váže se k souhlasu vztahem agregace.
- **Záznam**. Entita odpovídající popisu v sekci terminologie - 6.1. Váže se ke kontaktu vztahem kompozice.
- **Atribut**. Entita odpovídající popisu v sekci terminologie - 6.1. Má vlastnosti jako unikátnosti (v rámci všech kontaktů) nebo zda se jedná o přihlašovací atribut (klient se pomocí něj identifikovat a přihlásit). Váže se k záznamu vztahem agregace.
- **Účel zpracování**. Entita dle pojmu *účel zpracování* v kontextu GDPR. Účel zpracování je vázán k atributům.
- **Právní důvod**. Entita dle pojmu *právní důvod* v kontextu GDPR. Váže se k účelu zpracování vztahem agregace.
- **Zpracovatel**. Podobně jako u správce, je v návrhu definován zpracovatel jako konkrétní uživatelská role a odvíjí se od pojmu zpracovatel v kontextu GDPR. Zpracovatelé jsou přiřazováni ke konkrétním účelům zpracování. Příkladem zpracovatele může být osoba zodpovídající za emailing pro daný podnik. Entita je vázána k účelu zpracování vztahem agregace. Zpracovatelé mají přístup do rozhraní organizace.
- **Systém zpracování**. Entita odpovídající popisu v sekci terminologie - 6.1. Váže se k účelu zpracování vztahem agregace.

Definování entit a vazeb mezi nimi je zásadně ovlivněno požadavkem na konfigurovatelnost. V případě množiny pevně stanovené množiny atributů, které by u kontaktu byly evidovány, byly by tyto atributy realizovány jako *atributy entity*. Aby byl systém konfigurovatelný, je ale nutné atributy rozlišit jako samostatnou entitu. Obdobná situace je tomu u souhlasů. Vzniká tak řada vazeb typu N:M. Entity, některé jejich atributy a veškeré důležité vazby mezi nimi jsou zobrazeny na obrázku 16.

Dále je také nutné uvést, že se jedná jen o základní množinu entit, důležitou pro pochopení logiky systému. V žádném případě se nejedná o kompletní výčet všech entit (a odvozených tříd), které se při implementaci vyskytnou. Již v této fázi návrhu je zřejmé, že dalšímu entitamy a z nich odvozenými třídami budou například e-mail, záznam akce, záznam odeslání emailu, soubor, příloha souhlasu, šablona e-mailu, nastavení tenanta, žádost o zapomenutí a mnoho dalších.

V tomto bodě návrhu je vhodné uvést naprosto **konkrétní příklad použití entit**, z kterého bude mnohem jasnější jak jejich význam, tak význam vazeb.

Řekněme tedy, že ***správce*** společnosti X, fyzická osoba jménem David, udržuje informace o deseti ***kontaktech***. O těchto kontaktech chce evidovat tří ***atributy***: emailovou adresu, příjmení a poštovní adresu. David má dále v systému definované dva ***typy souhlasů***: souhlas se zpracováním osobních dat (tedy ten hlavní) a souhlas se zasíláním časopisu, který vydává. První ze souhlasů je vázán ke všem třem ***atributům***, druhý pouze k příjmení a poštovní adrese (protože nic víc k zaslání časopisu není třeba). Zasílání časopisu je ***účelem zpracování dat*** a ***právní důvod***, který k němu má je udělení souhlasu (nikoli třeba právní důvod na základě oprávněného zájmu). Zasílání časopisu neřeší konkrétně Petr. Ve společnosti X je za to odpovědná jiná osoba, fyzická osoba jménem Marie, která je ***zpracovatelem*** osobních dat, který se váže ke zmíněnému účelu zasílání časopisu. Marie pro svoji práci využívá software s názvem PrintSender. Tento software udržuje svoji vlastní databázi příjmení a adres. Jedná se o ***systém zpracování***. Nyní řekněme, že jeden z ***kontaktů***, tedy klient - osoba jménem Petr se rozhodla, že chce dostávat od společnosti X pravidelně časopis. K tomu, aby se tak mohlo stát potřebuje společnost X jeho ***souhlas***. Petr se tedy přihlásí za pomocí své emailové adresy do systému. ***Záznam*** jeho emailové adresy (***atribut*** emailová adresa) již v systému existuje, takže to není žádný problém. Následně udělí ***souhlas*** u ***typu souhlasu*** s názvem zasílání časopisu a také vyplní svoji poštovní adresu a příjmení. Po odeslání do systému je na základě vazby mezi ***atributem*** a ***účelem zpracování*** a vazby mezi ***účelem zpracování*** a ***zpracovatelem*** notifikována Marie, že Petr provedl dané změny, které se týkají daného ***účelu zpracování*** a tudíž odpovídajícího ***systému zpracování*** PrintSender (ale systému by to mohlo být klidně více). Marie tak ví, že může do seznamu odesílaných časopisů Petra zařadit.



Obr. 16: Diagram tříd - Základní entity

6.5 Architektura

Při návrhu architektury aplikace je snaha o rozdělení projektu na menší části, samostatné aplikace. Využita je tedy architektura mikroservisů. O microservices architektuře je možné více se dozvědět v sekci jí vyhrazené - 4.3.

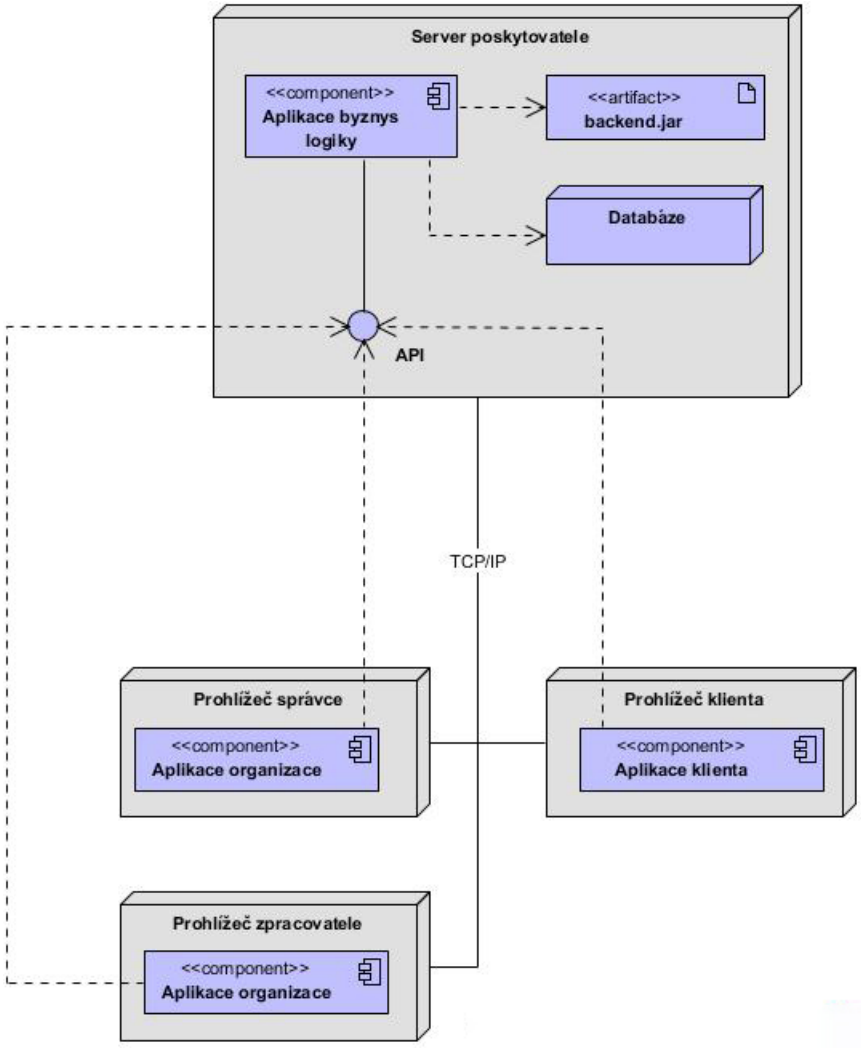
Snahou je využít všechny výhody této architektury. Záležitosti, které jsou často považovány za nevýhody mikroservis jsou v tomto případě vyřešené samou povahou projektu. Prvním bodem je nutnost navrhování a implementace API (Mirtes, 2015). Vytvoření API je v případě vyvíjené aplikace přímo požadavkem z důvodu možností systémové integrace. Dalším důvodem dle Fowlera (Mirtes, 2015) je nebezpečí plynoucí z nevědomosti, jak projekt správně na mikroservisy rozdělit. A to především v počáteční fázi návrhu. Tato záležitost není v případě této práce natolik obtížná, protože rozdělení analýzou požadavků krystalizuje samo. Také je toto rozdělení provedeno v duchu třívrstvé architektury. Rozdělení je následující:

- **Aplikace klienta (Frontend aplikace klienta)**. Zástupce *prezentační vrstvy* třívrstvé architektury. Aplikace je grafickým uživatelským rozhraním celého systému pro kontakty - subjekty zpracovávaných dat. Funkcionalita, kterou aplikace nabízí odpovídá diagramu užití na obrázku 8.
- **Aplikace organizace (Frontend aplikace organizace)**. Další zástupce *prezentační vrstvy* třívrstvé architektury. Jedná se o rozhraní pro správce a zpracovatele a analogicky pokrývá funkcionalitu dle obrázku 9 a 10.
- **Aplikace byznys logiky (Backend aplikace)**. Jádro aplikace v které se řeší logika a operace s daty. V třívrstvé architektuře nazývána *aplikační vrstva* nebo také *vrstva byznys logiky*. Součástí je rozhraní - API, pomocí kterého komunikuje s frontend aplikacemi. Reálně se veškerá funkcionalita uvedená výše provádí na této vrstvě a frontend nabízí pouze uživatelské prostředí. Komunikuje s databází.

Architektura čítající tři samostatné aplikace je vyobrazena na obrázku 17. Jedná se o logické rozdělení, které je vzdálené od skutečné architektury nasazení aplikací v infrastruktuře. Ta je více popsána v kapitole zabývající se návrhem infrastruktury.

Rozdělení frontend aplikací na dvě samostatné aplikace dává smysl z několika hledisek. Obě aplikace poskytují jinou, na sobě nezávislou, funkcionalitu. Je tu možnost používat celý portál jen jako samostatnou backend aplikaci a jeho prezentační vrstvu vůbec nevyužít. Je také možné, že některá společnost nemá zájem portál vystavovat veřejně do internetu a chtěla by ho využít pouze kombinaci backend aplikace a frontend aplikace organizace. Neobvyklý by nebyl ani požadavek, při využití všech třech aplikací portálu, na přístup do aplikace organizace pouze z interní sítě. Pokud se již nyní v návrhu na aplikace podíváme z více implementačního hlediska, tak nejpravděpodobnější implementace frontend aplikací bude použití webu. Rozdělení pak dává smysl i z bezpečnostních hledisek. Pokud veřejně dostupná aplikace klienta je zcela oddělená od aplikace organizace (jiný zdrojový kód, jiná doména),

je to pasivní ochranou před různými potencionálními narušeními bezpečnosti.



Obr. 17: Diagram nasazení - Architektura tří aplikací

6.6 Grafické uživatelské rozhraní

Významnou součástí návrhu je návrh grafického uživatelského rozhraní. Tato část návrhu bývá v různých informačních systémech často opomíjená a vznikají systémy, které sice plní svůj účel, ale jejich uživatelé je jednoduše nemají rádi a práce v nich je zdlouhavá a neefektivní. V této práci je na přívětivé rozhraní a celkovou lepší uživatelskou zkušenost kladen nemalý důraz.

Na návrh v této oblasti bude nahlíženo z pohledu struktury aplikace a navigace. V rámci návrhu byly vypracovány modely designu uživatelského rozhraní, známé pod anglickým slovem *mockup*. Dále byly vypracovány myšlenkové mapy pro které

se používá výraz *mindmap*. Tyto návrhy jsou součástí práce a nachází se v příloze A na straně 98.

Struktura aplikace

Pří návrhu designu rozhraní jsou v případě aplikace klienta uplatněny přístupy User centered design a Activity centered design. Volba byla provedena na základě potřeby co nejvíce vyjít vstříc uživateli (veřejnost) a zaměření na co nejjednodušší vykonání aktivity, která by se obecně dala shrnout jako zobrazení/úpravy svých osobních dat (a souhlasů). V případě aplikace organizace jsou uplatňovány přístupů User centered design a System centered design. Pro rozložení aplikace byl zvolen typ aplikace s jedním oknem měnícím obsah. Ani v aplikaci organizace se nepředpokládá nutnost paralelní práce s více položkami na jedné obrazovce. Jak dodává k těmto typům aplikace Procházka (Procházka, 2014), je nutné pečlivě definovat cesty aplikací. Tím se zabývá následující sekce o navigaci.

Navigace

Důležitým prvkem v oblasti uživatelské zkušenosti je navigace. Jak uvádí Procházka (Procházka, 2014) jedná se o problém, jak uživatele provést aplikací, aby dosáhl cíle, ke kterému byla vytvořena a také aby v každém okamžiku věděl, kde se v aplikaci nachází. Existují tu dva protichůdné směry. Zatímco pokročilí uživatelé upřednostňují mít veškerou funkcionalitu na dosah, tak začátečníky velká paleta možnosti mate. V případě navrhovaného portálu je tu výhodou, že je aplikace dělená na rozhraní organizace a klienta. Klientskou část je vhodné přizpůsobit tak, aby ani amatér neměl problém s navigací. U rozhraní organizace naopak lze předpokládat zkušeného uživatele. Přesto je v obou případech nutné dodržovat minimum a návrh této aplikace jej bude splňovat. Procházka (Procházka, 2014) uvádí toto minimum jako následující trojici pravidel.

1. **Stále vím, kde jsem**
2. **Vím, kam jít**
3. **Je to blízko**

Dále v oblasti navigace existují určité vzory. Ty se dělí na vzory, které řeší přechod ze stavu do stavu a vzory které identifikují pozici v aplikaci. Některých z těchto vzorů je pří návrhu aplikace využito.

V rámci návrhu navigace byly vypracovány myšlenkové mapy pro obě Frontend aplikace. Obrázek 21 je myšlenková mapa pro rozhraní organizace a obrázek 22 pro rozhraní klienta. Uzly v těchto grafech představují obrazovky a hrany přechody mezi nimi. Pro přehlednost jsou vynechány přechody značící akci vrácení se o krok zpět. Uzel označen šedou barvou je uzel startovací.

Při návrhu klientské aplikace byla zmíněná tři pravidla aplikována tak, že výsledkem je průchod aplikací skládající se pouze ze čtyř obrazovek a přechody mezi

nimi jsou lineární a v jasném pořadí. Klient se tedy nemá možnost jakkoli ztratit a vždy ví kde se nachází. Dále byl aplikován vzor Clear Entry Points. Ten Procházka (Procházka, 2014) definuje jako jasný vstupní a zvýrazněný bod evokující úlohu obrazovky. Na všech obrazovkách bude vstup zvýrazněn a především bude pouze jeden (formulář pro přihlášení / formulář pro aktivační klíč / formulář osobních údajů). To lze vidět na mockupech na obrázcích 23 a 24. Dalším použitým vzorem je vzor Modal Panel, vyskakovací okno v kterém je vyžadováno rozhodnutí uživatele. Takové okno je použito u některých akcí, kde je potřeba klientovo rozhodnutí potvrdit. Příklad je na mockupu na obrázku 25.

Jiná situace je u aplikace organizace. Zde se obrazovek nachází dle myšlenkové mapy osmnáct a navigaci je tak nutné řešit intenzivněji. Vhodným vzorem se nabízí Global Navigation. Procházka vzor definuje jako část obrazovky, která vždy ukazuje konzistentní ovládání pro přepnutí mezi částmi aplikace (Procházka, 2014). V myšlenkové mapě 21 jsou obrazovky, které jsou obsahem této navigace uzly označeny zelenou barvou. Využitím tohoto vzoru je tak i v takovém množství obrazovek možné dosáhnout úrovně zanoření maximálně tři. Aby se minimalizoval počet prvků na obrazovce je globální navigace skrývací. Pro podporu prvního pravidla uvedeného výše je také na každé obrazovce záhlaví s názvem obrazovky. Stejně jako v klientské aplikaci je i zde využit vzor Modal panel a snaha aplikaci vzoru Clear Entry Points. Popsané prvky jsou zřejmé z mockupů na obrázcích 26 a 27.

7 Analýza a návrh infrastruktury

Dokončením návrhu řešení aplikační vrstvy je možné přejít k analýze požadavků, modelování a vytvoření návrhu řešení pro infrastrukturu.

7.1 Specifikace požadavků

Funkční požadavky

- **Automatizované vytvoření a nasazení aplikace pro nového tenanta**. Proces nasazení aplikace pro nového tenanta by měl být jednoduchý a spolehlivý a uskutečnitelný bez zásahu programátora. Průběh procesu by se měl sestávat z jasně definovaných kroků a měla by být také dostupná možnost jeho kontroly. Nasazení nového tenanta by nemělo nijak ohrozit ani jinak ovlivnit již běžící instance vázané k jiným tenantům.
- **Možnost aktualizovat tenantské aplikace**. Vzhledem k probíhajícímu vývoji aplikace je žádoucí možnost spuštění automatizované aktualizace poskytované aplikace. Na proces aktualizace jsou kladeny stejné požadavky jako na proces vytvoření a nasazení aplikace pro nového tenanta.
- **Možnost zobrazení provozních informací**. Infrastruktura by měla mít možnost zobrazení jejich provozních informací pomocí grafického uživatelského prostředí. Takový přístup je důležitý pro možnost okamžité kontroly například helpdesk operátorem. Dostupné by měly být informace jako jsou logy aplikací, adresy nebo provozní statistiky.
- **Možnost ovládání infrastruktury**. Infrastruktura, tedy jak instance nasazených aplikací tak instance zajišťující její samotný běh, by měla být ovládatelná pomocí grafického uživatelského prostředí bez nutnosti hlubších vývojářských znalostí. Vyškolený personál by měl mít možnost provádět manuálně některé základní úkony jako je restartování aplikací, nebo diagnostika problému.

Nefunkční požadavky

- **Bezpečnost**. Bezpečnost je důležitým požadavkem na infrastrukturu. V případě narušení bezpečnosti infrastruktury vzniká vysoké riziko z podstaty toho, že práce pojednává o portálu pro skladování osobních údajů. Přístupy je tedy nutné zabezpečit více mechanismy, nikoli pouze například pomocí uživatelského přihlášení. Prvky zabezpečení infrastruktury také zasahují do zabezpečení samotné aplikace.
- **Spolehlivost a dostupnost**. Narušení jakékoli části infrastruktury by nemělo ohrozit běh jiných částí. Jednotlivé prvky infrastruktury by na sobě měly být v co nejvyšší míře nezávislé. Infrastruktura je schopna se pokusit samostatně obnovit narušenou část.

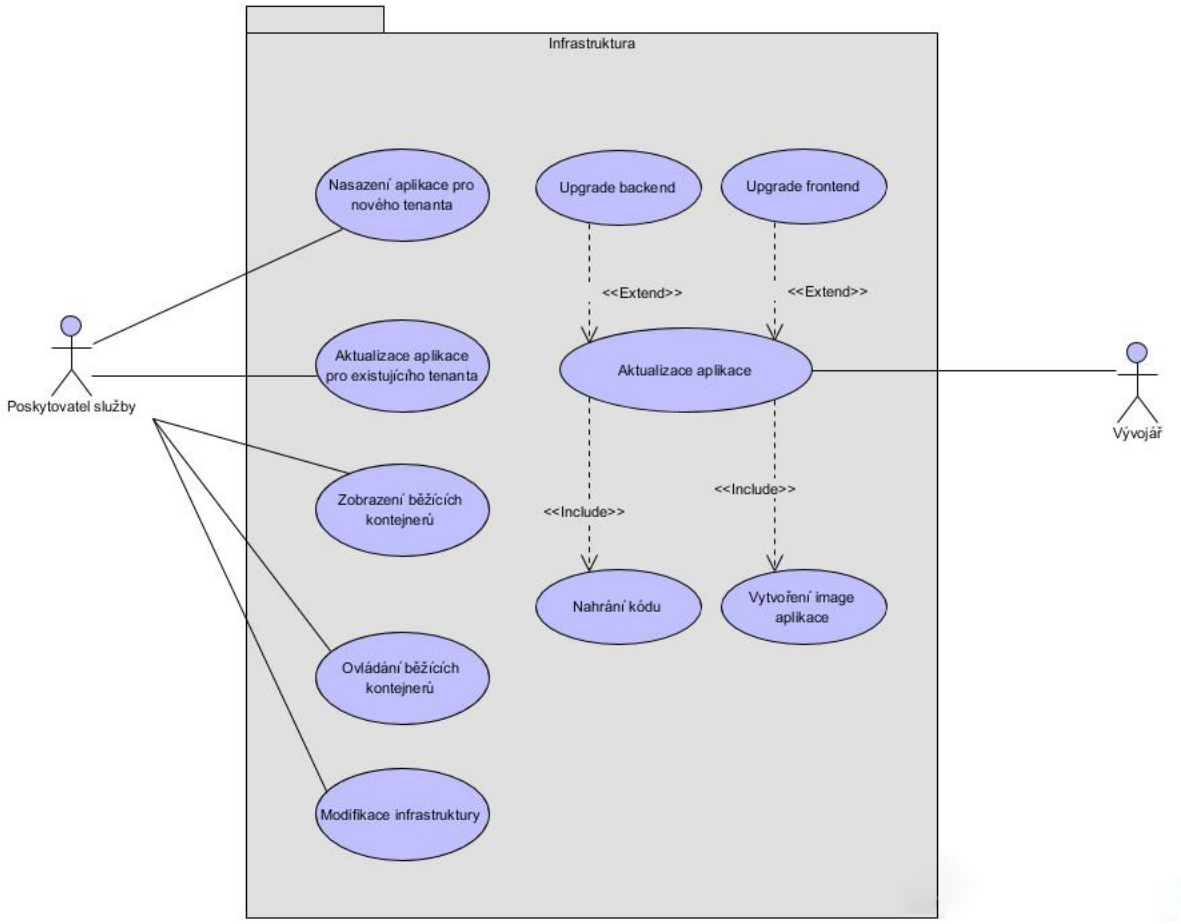
- **Možnost modifikace infrastruktury**. Jednotlivé části infrastruktury by mělo být možné modifikovat, například restartovat a aktualizovat bez narušení funkčnosti celku.
- **Zálohování dat**. Veškerá data by měla být zálohovaná. Narušení infrastruktury jako například její vyřazení z provozu by neměla data ohrozit. Zálohovaná by měla být jak data provozní, tak data infrastruktury tak data nasazených aplikací.

7.2 Specifikace požadavků

Pro uvedené funkční požadavky byl vytvořen diagram případů užití, který je na obrázku 18. Pro některé případy užití byly vypracovány jejich specifikace, které jsou popsány níže. V diagramu se vyskytují dvě role. Role *poskytovatele služby* a role *vývojáře*. Za rolí poskytovatele služby stojí osoba (osoby) z hostující společnosti, která cloudovou aplikaci nabízí. Tato role má v rámci infrastruktury veškeré možnosti a práva. V tomto je opakem role vývojáře, jenž nemá žádné práva a fakticky s infrastrukturou nepracuje.

Jediný případ použití, který se týká vývojářské role je **případ užití UCI01: Aktualizace aplikace** (tabulka 8) (případně některé její části, v závislosti na architektuře projektu). Je specifický v tom, že jeho vyvolání je způsobeno nahráním aktualizace zdrojového kódu do repositáře a spuštěno automaticky. Důležité je uvést, že tato aktualizace neprovede automaticky aktualizaci již běžících instancí aplikací v infrastruktuře. Takový postup se ukázal být v nynější fázi zamýšlené podoby portálu nežádoucí z více důvodů. Více o této záležitosti v sekci 10.3. Výsledek aktualizace je možnost poskytovatele služby založit aplikaci na aktualizované verzi pro nového tenanta, nebo spustit proces aktualizace aplikace pro existujícího tenanta.

Případ užití UCI02: Nasazení aplikace pro nového tenanta (tabulka 9) popisuje zavádění aplikace pro novou organizaci - tenanta, která si přeje portál přístupem SaaS používat. **Případ užití UCI03: Aktualizace existujícího tenanta** (tabulka 10) popisuje aktualizaci aplikace pro existujícího tenanta. Zmíněné parametry tenanta jsou údaje potřebné k založení aplikace tenanta v cloudu. Jedná se například identifikátor, doménu nebo heslo k databázi.



Obr. 18: Diagram případů užití - Infrastruktura

Tab. 8: Případ užití - Aktualizace aplikace

Název případu užití	Aktualizace aplikace
Identifikátor případu užití	UCI01
Cíl případu užití	Aktualizace aplikace v infrastruktuře
Primární aktér	Vývojář
Sekundární aktér	-
Vstupní podmínky	Vývojář má práva zaslat kód do repositáře v systému, nová verze má nepoužité číslo verze
Výstupní podmínky	V systému je možnost vytvořit aplikaci na nové verzi, je možnost existující aplikaci aktualizovat na novou verzi
Popis scénáře	
1. Vývojář zašle nový kód do systému 2. Systém zkontroluje číslo verze 3. Systém provede sestavení a testování nové verze 4. Systém zavede novou verzi do seznamu použitelných verzí	

Tab. 9: Případ užití - Nasazení aplikace pro nového tenanta

Název případu užití	Nasazení aplikace pro nového tenanta
Identifikátor případu užití	UCI02
Cíl případu užití	Běžící aplikace pro nového tenanta
Primární aktér	Poskytovatel služby
Sekundární aktér	-
Vstupní podmínky	Na zadané doméně ještě neběží jiný tenant, v systému je alespoň jedna verze aplikace
Výstupní podmínky	V systému exisuje běžící aplikace pro nového tenanta, aplikace je dostupná pod zadanou doménou, je vytvořena samostatná databáze tenanta pod zadaným heslem
Popis scénáře	
1. Poskytovatel služby parametry tenanta a volitelně verzi aplikace (při nezvolení verze, je zvolena verze poslední) 2. Systém ověří ověří parametry tenanta 3. Systém ověří doménu 4. Systém založí aplikaci 5. Systém odešle poskytovateli služby notifikaci o založení	

Tab. 10: Případ užití - Aktualizace existujícího tenanta

Název případu užití	Aktializace aplikace existujícímu tenantovi
Identifikátor případu užití	UCI03
Cíl případu užití	Běžící aktualizovaná aplikace pro tenanta X
Primární aktér	Poskytovatel služby
Sekundární aktér	-
Vstupní podmínky	V systému běží aplikace tenanta X na verzi, která není aktuální, V systému se nachází nová verze aplikace
Výstupní podmínky	V sytému běží aplikace tenanta X na zadané verzi
Popis scénáře	
1. Poskytovatel služby zadá parametry a požadovanou novou verzi aplikace 2. Systém zastaví aplikaci tenanta 3. Systém aktualizuje aplikace tenanta 2. Systém spustí aplikaci tenanta na nové verzi 4. Systém odešle poskytovateli služby notifikaci o aktualizaci	

7.3 Architektura

V závislosti na požadavcích stanovených pro infrastrukturu a také v návaznosti na architekturu navrhovanou pro aplikační vrstvu v sekci 6.5 je zvolena architektura infrastruktury založená na **kontejnerizaci**. Ve slovníku cloudových technologií je aplikační vrstva poskytována dle modelu *software as a service (Saas)* - **software jako služba**.

Volba kontejnerizace

Důvody proč je zvolena kontejnerizace vychází z informací, které je možné se dozvědět v kapitole věnující se studiu a přehledu v technologiích a přístupech. V kontextu vyvíjené aplikace je to především fakt, že snadno nasaditelné a spravovatelné kontejnery (se zásadně menší režií než třeba virtuální stroje) odpovídají mikroservisní architektuře aplikační vrstvy. Tři samostatně běžící a mezi sebou komunikující aplikace dávají smysl jako tři samostatně běžící a mezi sebou komunikující kontejnery. Dále kontejnerizace odpovídá požadavku provozovat portál jako cloudovou službu. Pro přidání nového tenanta je snadné a pružné řešení pomocí několika málo příkazů nasadit nové kontejnery než nasazovat celé virtuální stroje, natož pak nevirtualizovat vůbec. Stejně tak i správa kontejnerů je snadná a spolehlivá.

Ačkoli to v nynější verzi vývoje není konkrétním požadavkem, je vhodné myslet na záležitosti jako je distribuovanost a škálovatelnosti. Může například dojít k situaci, kdy extenzivní využití některé části aplikační vrstvy nebude nadále možné řešit optimalizací na hardwarové rovině ani algoritmické přímo v aplikační logice. V případě zajištění bezstavových aplikací je škálovatelnost jednou z největších výhod kontejnerizace. U kontejnerů je totiž jejich bezstavovost a neměnnost (immutable) přímo z jedním vzorů, které jsou doporučovány a spousta nástrojů je na to připravena.

Existuje ještě zcela jiný pohled na problematiku architektury než ten z produkčního prostředí. Jedná se o pohled vývojáře. Kontejnerizace nabízí vývojáři výhody přenositelnosti, reprodukovatelnosti a také je i zde výhodou zásadně nižší celková režie než v případě klasické virtualizace. Pokud chce vývojář například provést některé změny na frontend aplikaci může velmi snadně nasadit backend aplikaci na svém lokálním prostředí bez nutností zajišťování závislostí, proměnných systému nebo alokování velkého množství výpočetních prostředků.

Kontejnerizace

Nyní k popisu konkrétního návrhu kontejnerizace. Návrh jaké aplikace a v jakých kontejnerech se snaží pokrýt všechny stanovené požadavky, být dostatečně škálovatelný a zároveň se nadále držet mikroservisní architektury.

V architektuře operují tři kategorie kontejnerů. První kategorie se bude dále nazývat *statické kontejnery*. Jedná se o kontejnery, které jsou v architektuře zastoupeny vždy jednou instancí, která běží stále od prvního nasazení infrastruktury.

Jejich cílem je provoz infrastruktury jako takové. Druhá kategorie kontejnerů jsou kontejnery, v kterých běží aplikační vrstva a tyto kontejnery jsou vytvářeny v závislosti na počtu tenantů. Dále jsou nazývány jako *tenantské kontejnery*. Poslední kategorií kontejnerů jsou *kontejnery dočasné*. Ty vznikají a zanikají v průběhu různých operací.
Výčet statických kontejnerů je následující:

- **Router kontejner**. Tento kontejner funguje jako reverzní proxy. Jeho úkolem je směrování veškeré komunikace z internetu na další kontejnery v infrastruktuře. Měl by mít schopnost *serivce discovery* a registrování nových tenantů by tak mělo být automatické. Úkolem router kontejneru je také zajišťování hned několika záležitostí ohledně bezpečnosti na síťové vrstvě. Jedná se například o řízení přístupu na základě IP adres nebo zajištění SSL.
- **DB kontejner**. V kontejneru běží RDBMS. Data jsou uložena na hostující počítač, tedy nikoli vně kontejneru. To je důležité například z důvodu nezaniknutí dat při chybě kontejneru a jeho restartování. Spravovány jsou jednak databáze samotné aplikační vrstvy, tedy tenantských kontejnerů, tak případně databáze dalších statických kontejnerů zajišťující chod infrastruktury.
- **Backup kontejner**. Tento kontejner se stará o zálohování dat z ostatních kontejnerů. V nynější fázi návrhu se jedná o provádění záloh databází z DB kontejneru a jejich ukládání na pevný disk. Jeho oddělení od databázového kontejneru je ze dvou důvodů. Tento kontejner by se měl obecně starat o zálohování a to nejen databáze v databázovém kontejneru. Dále také, že provedení zálohy může být výpočetně náročná operace, kterou by nebylo vhodné zatížit přímo databázi. V případě problému zálohování by mohl například, v případě provádění zálohy přímo v DB kontejneru, tento kontejner zhavarovat a byla by databáze nedostupná.
- **Version control kontejner**. Kontejner spravující zdrojové kódy. Jsou zde repositáře aplikací aplikační vrstvy. Tento kontejner komunikuje s Build kontejnerem - je schopen na základě nového kódu spustit sestavení aplikace.
- **Control kontejner**. Úkolem tohoto kontejneru je poskytnout poskytovateli služby možnost kontroly nad celou infrastrukturou. Tedy přístup k informacím o běžících kontejnerech, možnost restartování a dalších operací s nimi.
- **Build kontejner**. Kontejner zaštiťující sestavování aplikací, automatizované nasazování a jejich aktualizaci.

Výčet tenantských kontejnerů je následující:

- **Backend kontejner**. Stěžejní kontejner aplikační vrstvy. Je zde nasazena Backend aplikace, která vystavuje rozhraní pro komunikaci.
- **Frontend kontejner**. V tomto kontejneru běží frontend aplikace - rozhraní

organizace nebo rozhraní klienta. Může mít instanci pro každou aplikaci zvlášť. V diagramu na obrázku 19 je kontejner s aplikací organizace nazván *frontend admin container* a analogicky se tam také nachází *frontend client container*.

Poslední kategorie dočasných kontejnerů obsahuje:

- **Backend builder kontejner**. Kontejner, který je zcela v režii build kontejneru. Vzniká v situaci, kdy build kontejner má sestavit a otestovat backend aplikaci a v případě úspěchu vytvořit jeho Docker image.
- **Frontend builder kontejner**. Analogicky stejné s backend builder, ale pro frontend aplikaci a to jak aplikaci organizace, tak aplikaci klientskou.

7.4 Multi-tenantnost

Navrhovaná kontejnerizace vznikla v průběhu iterací konzultací s vedením společnosti a je evolucí původní architektury, ve které nebyly dynamicky vytvářené kontejnery uvažovány a multi-tenantnost byla řešena pouze na aplikační úrovni. Tedy při nasazení byl pro všechny tenanty jeden společný Frontend kontejner a jeden společný Backend kontejner. Tato původní myšlenka byla v průběhu vývoje nahrazena myšlenkou nasazení trojice aplikačních kontejnerů pro každého tenanta a to z důvodů komplikací popsaných v následujícím seznamu:

- Komplikace při udržování různé verze aplikace pro různé tenanty. Více v sekci o aktualizacích 10.3.
- Nasazování nových verzí vyžaduje výpadek systému pro všechny tenanty.
- Při velkém množství zpracovávaných údajů vzniká velká výpočetní zátěž na Backend kontejneru. Tato zátěž při jediné instanci ovlivňuje rychlost běhu aplikace ostatních tenantů.
- I přes míru konfigurovatelnosti celého řešení se počítá s možností tenant-specifických úprav aplikace ve Frontend a Backend kontejnerech (například úpravy na zakázku). Takové upravené řešení vyžaduje vlastní instanci kontejneru s vlastní verzí upravené aplikace.

Pro zmíněné komplikace existují taková řešení, která by nevyžadovala přesun multi-tenantnosti z aplikační vrstvy na vrstvu infrastruktury. Jedná se například o techniku blue/green deployment pro nasazování nových verzí, nebo řešení výkonnostních problémů distribuováním. Hlavně se jedná o nutnost realizace aplikačních kontejnerů (tedy potažmo aplikace samotné) zcela bezstavově. Bylo rozhodnuto pro současný stav vývoje použít navrhované řešení a další úpravy v rámci multi-tenantnosti nechat případně na další vývoj a rozšíření celého řešení. Důvody rozhodnutí jsou časové, ekonomické a především pak o případném směru vývoji a rozšíření v této oblasti bude rozhodovat míra využití a komerční úspěch celého

řešení.

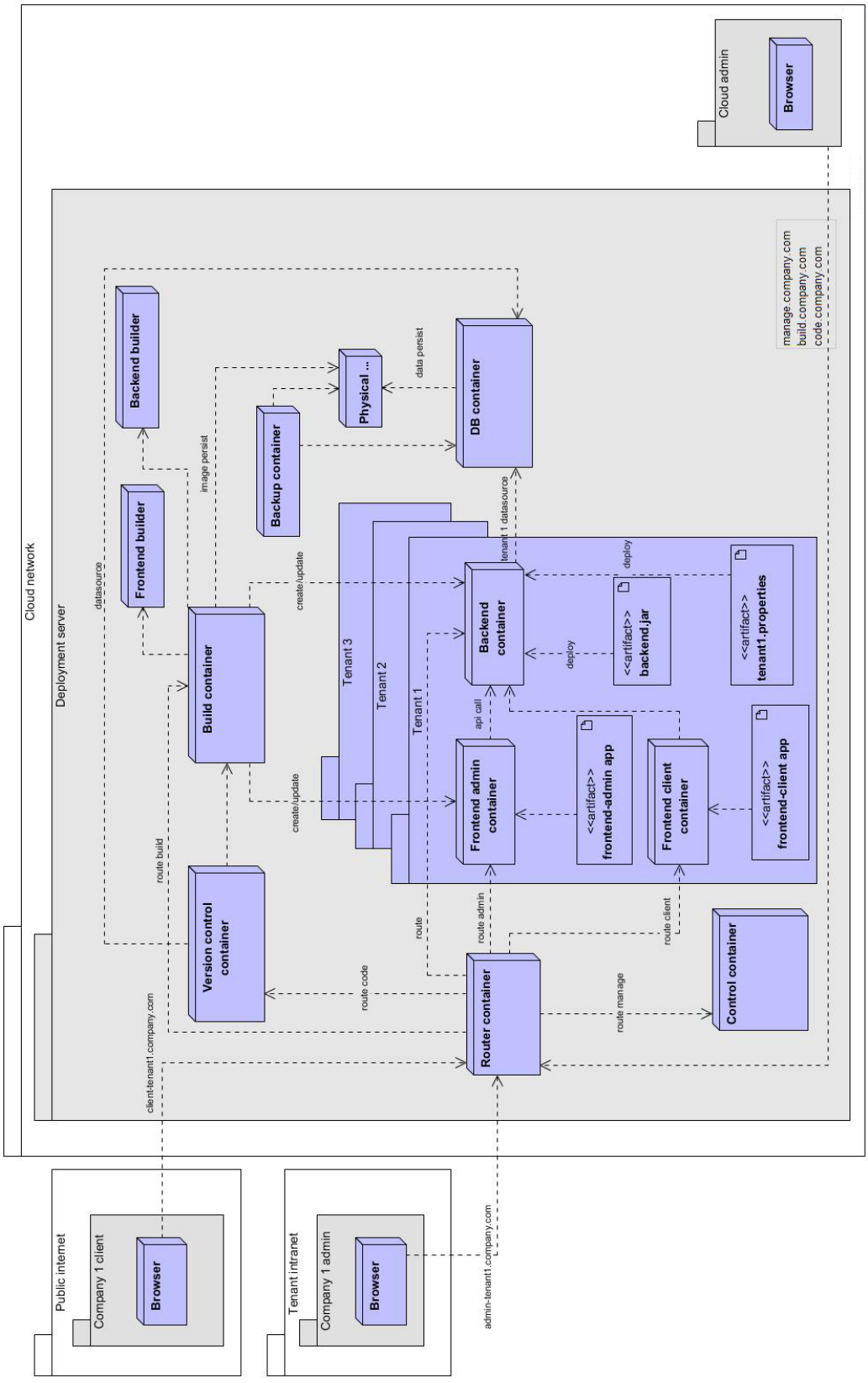
7.5 Bezpečnost

Z hlediska bezpečnosti infrastruktury je nutné se zamyslet nad několika oblastmi.

Tou první je **bezpečnost kontejnerů**. Kontejnery resp. jejich obrazy jsou tvořeny vrstvami souborů. Tou nejdůležitější vrstvou je základní, první vrstva, která je dále rozšiřována o vrstvy další. Nejdůležitější je i z hlediska bezpečnosti. Při volbě základního obrazu je tak důležité se držet několika mála pravidel a technik: základní obrazy by měly pocházet z ověřených zdrojů a být jimi podepsané, neměl by být zastaralý - vrstva operačního systému by měla být *up-to-date*, preferovat by jsme měli minimalistické základní obrazy, vybrané základní obrazy podrobit testu bezpečnosti některým z dostupných nástrojů.

Další oblastí je nakládání s **přístupovými údaji**. Cílem je minimalizovat množství systémových přístupových údajů, které by byly uloženy na serveru, kde infrastruktura běží. Pokud k tomu dojde, měly by být v šifrované podobě. Některé služby mohou pro svoje spuštění vyžadovat defaultní jméno/heslo uložené přímo v konfiguračním souboru. V takových případech bude vždy doporučením takové údaje posléze manuálně změnit.

Důležitou oblastí bezpečnosti je **řízení přístupu**. Statické kontejnery (sekce 7.3) poskytují kritickou funkcionalitu, která by ale měla být dostupná obsluze cloudu přes rozhraní. Aby se předešlo zbytečnému riziku je plánem přístup k těmto kontejnerům řídit kromě zabezpečení přístupovým jménem a heslem také pomocí seznamu povolených zařízení na bázi *whitelisting*. Implementačně se uvažuje zpřístupnit tyto věci jen z interní sítě společnosti pomoci povoleného rozsahu IP adres.



Obr. 19: Diagram nasazení - Infrastruktura

8 Implementace

Po získání přehledu v dotčených oblastech, fázi analýzy a vytvoření návrhu je možné přejít k popisu implementace. Ta se skládá z výběru konkrétních nástrojů a za jejich pomoci naprogramování funkčního řešení.

Návrh je dělen na návrh aplikační vrstvy a návrh infrastruktury. Výstupem návrhu aplikační vrstvy je architektura skládající se ze tří aplikací - backend aplikace, frontend aplikace klienta a frontend aplikace organizace. Tohoto dělení bude využito i pro popis implementace. Implementace obou frontend aplikací jsou velmi obdobné a budou tudíž sjednoceny a popsány případně budou pouze rozdíly.

Určitým strategickým rozhodnutím je **volba webových technologií** jako způsob implementace návrhu. Ačkoli se v návrhu o konkrétně webových technologiích nezmiňuje a návrh je, tak jak by měl být, nezávislý na konkrétní technologii, s využitím webu se počítalo již při definování konceptu. Důvod je jasný - pokud mají klienti mít přístup do portálu, nelze po nich vyžadovat například instalaci desktopového klienta. Respektive lze, ale rozhodně se nejedná o dobrý design. Stejně tak pokud je navrhované řešení na bázi cloudu, resp. modelu SaaS, tak je využití webových technologií velmi preferovanou cestou.

Nyní ještě k stavbě tohoto dokumentu. Většina obrázků na které je v následujících odstavcích odkazováno je součástí příloh této práce, které začínají na straně 97. Konkrétně ukázky implementovaných aplikací, resp. jejich grafického uživatelského rozhraní, jsou na obrázcích v příloze B začínající na straně 104.

8.1 Backend aplikace

V návrhu byla backend aplikace definována jako jeden samostatný spustitelný modul udržující byznys logiku, vystavující rozhraní a komunikující s databází. Takto byla také implementována.

Volba technologií

Pro implementaci byl zvolen jazyk **Java 11**. Projekt je postaven na systému pro správu a sestavování aplikací **Maven** a frameworku **Spring Boot**. Hlavní motivací pro volbu těchto technologií jsou autorovy vlastní zkušenosti s budováním podnikového (*enterprise*) software. Java v kombinaci se Spring frameworkem umožňuje vytváření rychlých, robustních a spolehlivých aplikací. Vývoj je díky rozsáhlému ekosystému Javy a velkému množství oblastí, které se u podobných aplikací typicky řeší a Spring Framework je pokrývá, velmi urychlen. Pro práci s daty je využit ORM nástroj **Hibernate** dle specifikace JPA.

Spring Boot umožňuje vytvářet stand-alone aplikace s integrovaným webovým serverem (Tomcat), které poskytují základní konfiguraci Spring Frameworku a které díky svým *starter* závislostem (jako například *spring-boot-starter-web*) poskytují balíčky známých Java knihoven dle tématických celků.

Architektura

Samotná aplikace byla postavena podle tradiční architektury webových aplikací Spring. Jedná se o rozdělení na tři vrstvy. Je nevhodné mít více než tři vrstvy jak uvádí Kainulainen (Kainulainen, 2014), protože přidávání nových funkcionalit se stává stejně jako údržba aplikace časově velice náročnou a také programátory velmi nepopulární prací.

- **Webová vrstva (*web layer*)**. Tato vrstva obsahuje REST rozhraní. Pracuje se zde s *data transfer object (DTO)* reprezentací entit, určených pouze k přenosu mezi frontendem a backendem. Příkladem tříd na této vrstvě jsou *DataProcessorEndpoint*, který definuje API endpointy pro práci s entitou Zpracovatel a *DataProcessorDTO*, jenž je její podoba jako *data transfer object*.
- **Servisní vrstva (*service layer*)**. Zde se nachází byznys logika. Pracuje se zde s entitamy. Příkladem tříd na této vrstvě jsou *DataProcessorService*, v které se například řeší funkce vyhledání všech zpracovatelů dle provázaných účelů zpracování a třída *DataProcessor* jenž odpovídá entitě Zpracovatel. Kromě servis, které přímo zpracovávají definované entity se na této vrstvě vyskytují ještě třídy řešící například import a export, nebo také práci se soubory (defaultní šablony pro emaily).
- **Datová vrstva (*repository layer*)**. Odpovídá za práci s databází. Příkladem třídy na této vrstvě je *DataProcessorRepository*, jenž má funkce pro ukládání a vyhledávání entit v databázi.

REST rozhraní

Implementované REST API pokrývá veškerou funkcionalitu backend aplikace. Je vytvořené co nejobecnějším způsobem a není nijak přízpůsobené implementací frontendu. Jedná se tedy o obecně použitelné rozhraní a může sloužit ke komunikací s například jinými alternativními implementacemi frontendu nebo jakýmikoli jinými plánovanými mikroservisy (zamýšlená systémová integrace).

Z pohledu práv jsou endpointy API rozděleny na tří kategorie.

- **Veřejné endpointy (*public*)**. Tyto endpointy jsou dostupné veřejně z internetu a nevyžadují žádnou autentizaci. Důvod jejich existence je možnost přihlášení a v případě klientské frontend aplikace také registrace. Endpointů tohoto typu je minimum.
- **Zabezpečené endpointy (*secured*)**. Tyto endpointy jsou pro přihlášeného klienta v klientské aplikaci. Nutná je tedy autentizace pomocí identifikátoru a tokenu, který obdrží klient při přihlášení (to je v režii frontend aplikace). Je zajištěno, že přes daný endpoint může uživatel modifikovat pouze svoje údaje.
- **Systémové endpointy (*admin*)**. Veškeré ostatní endpointy pokrývající zbylou funkcionalitu. Tyto endpointy jsou volány při práci v aplikaci organizace.

Nutná je autentizace administrátorským účtem.

Konfigurace

Aplikace si udržuje svoji základní konfiguraci v externím souboru *application-prod-container.properties*. Z tohoto souboru je také při prvním spuštění načtena defaultní konfigurace. Objekt konfigurace je uložen do databáze a následně je s konfigurací možno pracovat přes administrátorské rozhraní (nastavení jména společnosti, mailového serveru a další). Při restartu aplikace se již konfigurace načte z databáze.

Žurnál

Žurnál je implementací požadavku na evidování akcí v systému. V současné verzi aplikace se eviduje množina akcí uvedená níže. Backend je ale naprogramován tak, že přidání dalších událostí je velice snadné (stačí implementovat rozhraní *JournalRecord* pro model a generické rozhraní *JournalRecordParentRepository* pro repository třídu).

- **Změna datové položky**
- **Vytvoření kontaktu**
- **Vymazání kontaktu**
- **Žádost o zapomenutí**
- **Storno žádosti o zapomenutí**
- **Změna atributu**
- **Změna typu souhlasu**
- **Odeslání emailu**

Emailové notifikace

Pro odesílání emailových notifikací je nejprve nutné v rozhraní organizace nakonfigurovat poštovní server. Odchozí e-maily ze systému jsou předmětem zaznamenávání v žurnálu. Emaily jsou řazeny do fronty, odesílany jsou asynchronně a jejich odeslání je zaznamenáváno. Pokud by email neodešel, je možné ho z rozhraní organizace odeslat znovu. Ukázky emailových notifikací jsou v příloze C začínají na straně 121.

8.2 Frontend aplikace

Frontend aplikace byly pomocí implementovány jako jednostránkové webové aplikace (SPA). Tato volba SPA byla logickým vyústěním faktu, že je budována cloudová služba typu SaaS a architektura aplikační vrstvy odděluje frontend a backend. SPA aplikace také zlepšují uživatelskou zkušenost díky rychlejšímu běhu.

Volba technologií

Jako primární nástroj pro implementaci byl zvolen framework **Angular**. Uvažovány byly ještě frameworky React a Vue. Pří výběru nenabízel žádný z frameworků podstatnou výhodu oproti ostatním a autor práce neměl v době volby zkušenost s žádným z nich. Na základě několika zdrojů srovnání (Neuhaus, 2017), (Kravchenko, 2018), (Hamedani, 2018), oficiálních dokumentací a vlastního otestování na demo projektech byl vybrán Angular. Důvodem se tak především stal jazyk TypeScript, více objektově orientovaný přístup, dobrá dokumentace a bohatá nabídka komponent grafického uživatelského prostředí - Angular Material. Angular je také více oblíben mezi Java programátory.

Vizuální styl

Aplikace byly vystavěny pomocí designovacího jazyku od společnosti Google s názvem **Material Design**. Angular svojí knihovnou Angular Material specifikaci od Google implementuje. Zmiňovaná knihovna umožnila rychlý vývoj dobře strukturovaného, líbivého a interaktivního designu v jednotném stylu i pro vývojáře, jehož doménou jsou spíše backend aplikace. Nabídka komponent pokrývá jak formulářové prvky tak navigační prvky a komponenty pro strukturu stránky (seznamy, karty, taby).

Dynamicky generované formuláře

Jednou z výzev v oblasti implementace frontendu byly dynamicky generované a validované formuláře. Formuláře totiž musejí odpovídat definovanému schématu zpracovávaných dat, které je uloženo v databázi. Na frontendu tedy došlo k nadefinovaní vlastní komponenty *GenericFormComponent*, která zajišťuje vygenerování formuláře dle schéma. Nejedná se však pouze o prvky html, ale také o validátory. Atribut, jehož odrazem je jedno konkrétní vstupní pole, může být totiž například *unikátní*. Potom je nutné pohlídat, zda vkládaná hodnota daného atributu již v databázi neexistuje. To, při velkém množství kontaktů, atributů a hlavně paralelní práci více uživatelů v systému, není možné u SPA aplikace ověřovat na frontendu. Je tedy nutné komunikovat pro ověření unikátnosti s backendem. Postup při vygenerování formuláře je tedy takový, že frontend aplikace načte z backendu endpointu schéma atributů a na základě něj vygeneruje formulář a validátory. Při psaní do validovaného pole je aktuální hodnota zasílána na backend a tam ověřována, zda je unikátní.

Ukázka nastavení atributu, některých jeho vlastností (atributů) a následně vygenerovaného formuláře s tímto atributem je na obrázcích 34 a 35 v příloze B.

Aplikace klienta

Aplikace klienta byla implementována dle návrhu. Průchod obrazovkami odpovídá mindmapě z návrhu na obrázku 22. Všechny obrazovky klientského portálu tak jak

jdou za sebou lze vidět na obrázcích 28, 29, 30, 31 (kontaktu v procesu zapomenutí) a 32. V uvedeném případě ještě nedošlo ke konfiguraci loga a i název společnosti je v systému defaultní. Jako přihlašovací atributy byly v portálu organizace zvoleny atributy Email a IČO a tak jsou zobrazené na úvodní obrazovce.

Aplikace organizace

Taktéž aplikace organizace byla implementována na návrhu jen s malými odchylkami. Ukázky z ní jsou na obrázcích 33 (implementace globální navigace pomocí komponenty z knihovny Angular Material - *Sidenav*) až 40. Na obrázku 41 lze vidět implementaci žurnálu a konkrétně je otevřena nabídka filtrování dle typu událostí. Detail konkrétní události z žurnálu je na obrázku 39 - změna atributu jména. Jak by takový záznam události vypadal po dokončení procesu zapomenutí daného kontaktu a tedy jeho výmazu a anonymizaci všech souvisejících záznamů lze vidět na obrázku 41.

Dalšími obrazovkami, které v příloze B začínají na straně 112 jsou ty spojené s definováním účelů zpracování a souvisejících entit. Na obrázcích lze vidět příklad, jak by schéma společnosti mohlo vypadat.

8.3 Infrastruktura

Infrastruktura je implementována jako množina vzájemně komunikujících kontejnerů popsaných v jednom strukturovaném souboru formátu YAML spustitelném nástrojem Docker-compose.

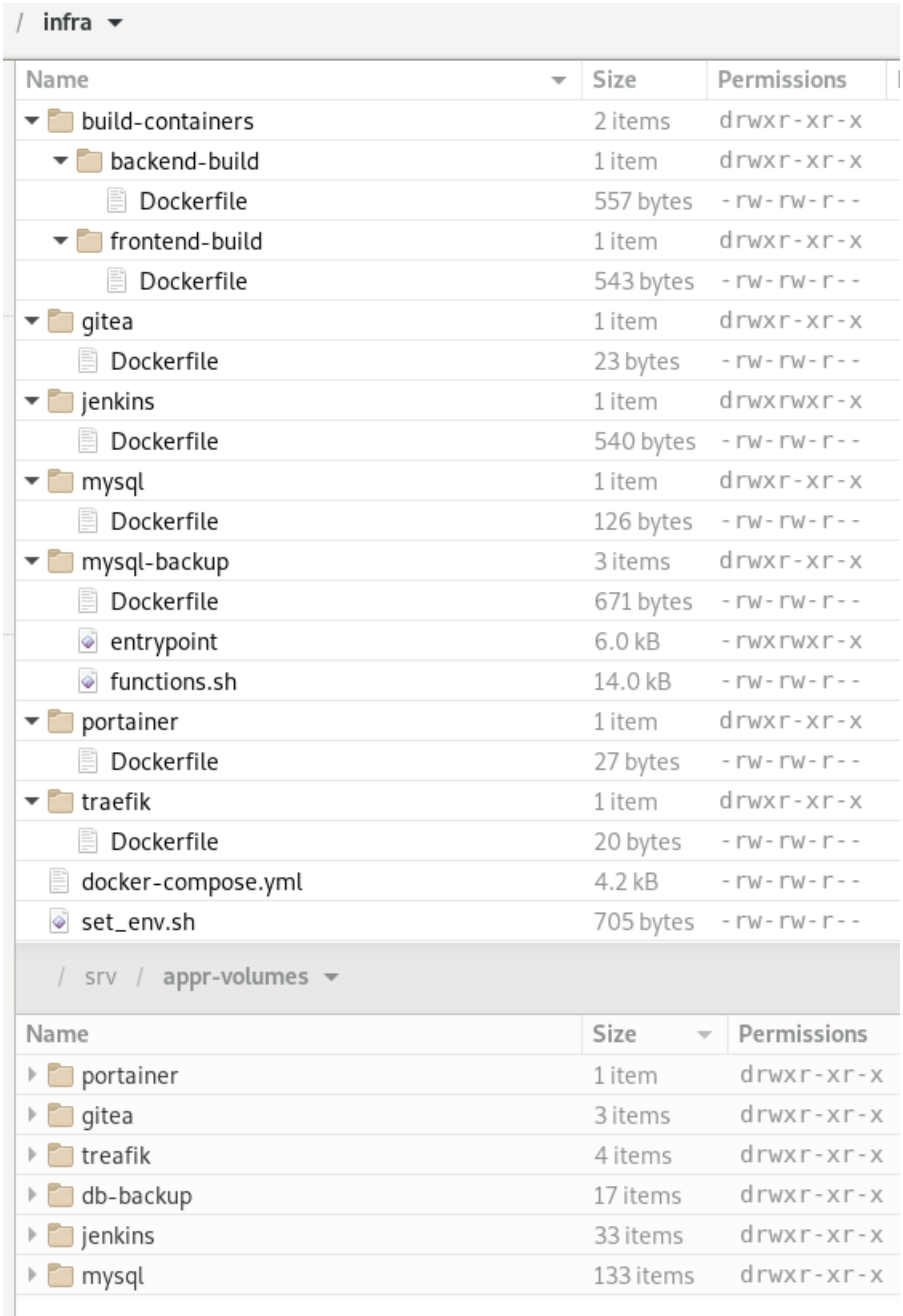
Fyzicky je infrastruktura adresář, jehož obsahem je několik souborů typu Dockerfile, soubor docker-compose.yml a několika dalších skriptů. Je spustitelná na jakémkoli linuxovém operačním systému s přístupem k internetu. Na obrázku 20 je ukázána stromová adresářová struktura a také struktura, která se při prvním spuštěním vytvoří pro persistenci dat na pevný disk počítače. Infrastruktura je verzována systémem Git.

K rozběhnutí infrastruktury na libovolném linuxovém serveru tedy stačí pouze nainstalovat Docker a Docker-compose, naklonovat repositář s infrastrukturou, upravit systémové proměnné v souboru *set_env.sh*, jehož obsah je na ukázce kódu 2, a spustit jej a konečně příkazem *docker-compose up* infrastrukturu zavést. Následně je už pouze potřeba nastavit některé aplikace z jejich uživatelského prostředí.

Kontejnerizace

Každý ze statických kontejnerů je v souboru docker-compose.yml definován jako service, která se sestavuje ze svého Dockerfile. Soubor docker-compose.yml je na ukázce kódu 1 v příloze E. Dockerfile soubory pro všechny typy kontejnerů jsou na ukázkách kódů v příloze F.

Na tomto místě je vhodné popsat jakými konkrétními technologiemi byly jednotlivé kontejnery z návrhu v sekci 7.3 implementovány a popsat jejich konkrétní



Obr. 20: Adresářová struktura infrastruktury

náplň práce. Předem je také třeba dodat, že autor při výběru technologií upřednostňoval nástroje s open-source licencí. Autor sám je open-source modelu fanouškem. Řešení je pouze minimálně závislé na konkrétním vybraném software a změna jakékoli části je velmi snadná.

Router kontejner byl implementován nástrojem **Traefik**. Dalšími zvažovanými možnostmi byly Apache HTTP Server, Nginx a HAProxy, ze kterých se nakonec ukázal Traefik jako nejlepší volba z důvodu jednoduchosti a nabízené funkcionality. Traefik je připojen k Docker Hostovi na hostujícím operačním systému (pomocí Docker volume - viz řádek 19 na ukázce kódu 1) a dle labelů, které jsou zadávány jako labely kontejnerů na ně dokáže routovat HTTP požadavky. Veškerá potřebná konfigurace pro přidání domény k novému kontejneru je pouze zadání labelů při spuštění, jak je například vidět na řádcích 118 - 121 na ukázce kódu 1. Traefik dále umožňuje velkou spoustu dalších nastavení z nichž je využito například omezení přístupu dle určitého rozsahu IP adres. To je znázorněno na řádku 122 na ukázce kódu 1. Jednou z největších výhod je možnost automatického přidělování a obnovování certifikátů pro domény pomocí integrace s Let's Encrypt. Kontejner Treafik si neudržuje žádný stav a jediná jeho konfigurace je uložena pomocí Docker volume na disku hostujícího operačního systému.

Pro **DB kontejner** byla zvolena databáze **MySQL verze 8**. Jeho data jsou také uložena na disku hostujícího operačního systému. Není mu přidělena žádná doména a ostatní kontejnery k němu přistupují dle jeho aliasu v rámci sítí appr_build a appr_tenant.

Backup kontejner má v této fázi vývoje na starost pouze zálohování databází z DB kontejneru. Pravidelnost záloh je nastavitelná systémovými proměnnými, viditelné na řádcích 81 a 82 na ukázce kódu 1. Vzhledem k tomu, že jeho prací je pouze záloha MySQL databáze, pro implementaci bylo zvoleno řešení dle projektu *databacker/mysql-backup* (Deitcher, 2019).

Version control kontejner je implemetován nástrojem **Gitea**. Jedná se o velice jednoduchý a rychlý nástroj pro hostování kódu. Požadavků pro volbu tohoto nástroje nebylo mnoho a splňují je všechny podobné nástroje stejně dobře. Jedná se jen o podporu verzovacího systému Git, grafické uživatelské prostředí a možnost integrace s dalšími nástroji jako je automatizační server. Důvodem je i to, že repositáře v tomto kontejneru jsou využívány pouze pro účely sestavování aplikací a vytváření obrazů uvnitř infrastruktury. Hlavní repositář, kde probíhá vývoj, a na kterém jsou různé vývojové větve a podobně, je odlišný od toho v popisovaném kontejneru. Nachází se na centrálním uložišti zdrojových kódu ve společnosti.

Pro **Control kontejner**, který má být ovládacím panelem celé infrastruktury, byl zvolen nástroj **Portainer**. Nástroj umožňuje veškeré operace nad běžícím Docker enginem na hostujícím operačním systému. Je také umožněn jednoduchý přístup k logům nebo i terminálový přístup do kontejnerů z webového prostředí. Stejně jako Traefik je připojen k Docker hostovi přes Docker volume a je tak schopen automatického zaregistrování vznikajících kontejnerů. Při výběru tohoto nástroje se ukázalo, že není příliš variant a alternativ. Další možností byly v podstatě pouze

nástroje Rancher a Shipyard. Rancher je zdaleka nejrobusnější řešení, které je ale spíše zaměřené na orchestraci kontejnerů při použití technologií jako Kubernetes, Docker Swarm nebo Apache Mesos. Funkcionalita Shipyard a Portainer je velmi podobná. Z této dvojice ale Portainer vychází lépe co se týče grafického rozhraní a také režie při nasazení.

Jako nástroj pro implementaci **Build kontejneru** byl použit **Jenkins**. Na poli CI/CD nástrojů je široká nabídka nástrojů. Z vybrané skupiny nástrojů čítající Jenkins, TeamCity, Travis, Bamboo, CodeShip a GitLab CI byl vybrán Jenkins na základě nejlepší shody požadavků a toho, co daný software nabízí. Dle serveru Altexsoft (AltexSoft, 2019) se jedná o nejrozšířenější software svého druhu jehož hlavní výhody jsou: je zcela zdarma, má aktivní komunitu a nekonečné možnosti integrace. Mezi integrace patří i požadovaná integrace s Dockerem. Velkou výhodou je také kromě dobrého grafického uživatelského prostředí i možnost ovládání z příkazové řádky. Jenkins je také velmi snadné spustit v Docker kontejneru, je připraven oficiální Docker image a ukázal se býti schopný plnit požadovanou funkcionalitu s jen malou potřebou konfigurace a malou spotřebou systémových zdrojů.

Jenkins je v infrastruktuře integrován s Gitou pomocí webhooku. Je stejně jako Traefik a Portainer propojen s Docker Hostem na hostujícím operačním systému. Při sestavování aplikací je tak schopen spustit na hostujícím operačním systému dočasné kontejnery jen pro účel sestavení a testování aplikací. V návrhu jsou tyto kontejnery nazvány jako dočasné kontejnery a jsou to **Backend builder kontejner** a **Frontend builder kontejner**. Po splnění svého účelu jsou tyto kontejnery ukončeny a smazány.

Poslední kategorie kontejnerů jsou kontejnery tenantské. Jejich prací je pouze poskytnout běhové prostředí pro aplikaci. **Backend kontejner** pro běh Java aplikace je založen na **Mavenu**. Vzhledem k implementaci aplikace je to jediná logická volba. **Frontend kontejner** udržuje klientskou nebo správcovskou aplikaci vytvořenou za pomocí frameworku Angular. Podoba takové aplikace po sestavení jsou sobory html a javasript. Roli web serveru zastává ve frontend kontejneru nástroj **Nginx** pro svoji jednoduchost a rychlost pro statický obsah. Dle Leslie (Leslie, 2018) je pro statický obsah Nginx 2.5krát rychlejší než jeho konkurent Apache HTTP Server.

Veškeré kontejnery se jsou schopny, například při výpadku serveru kde infrastruktura běží, samy znovu nastartovat.

Automatizace

Pro dosažení automatizace byly pro potřebné případy užití cloudu navrhnuty *Jenkins pipeline joby*. Jejich použitím je pokryt i požadavek na spolehlivost. Na jejich průběh jsou navázány emailové notifikace a tak například při problému s nasazením nového tenanta je poskytovateli cloudu odeslán e-mail. Jejich průběh je také v nástroji Jenkins vizualizován, viz obrázek 50 v příloze G (strana 132), kde se také nacházejí ukázky kódu k této sekci.

Skripty byly napsány v jazyku Groovy a odpovídající trojice skriptů je součástí repositáře dané aplikace, jsou tedy verzovány. Skripty jsou spouštěny s parametry jak je vidět na ukázce kódu 14. Seznam definovaných skriptů je následujícím seznamu:

- **appr-backend-build**. Sestavení, otestování a vytvoření image backend kontejneru.
- **appr-backend-deploy**. Nasazení backend kontejneru pro nového tenanta.
- **appr-backend-update**. Aktualizace backend kontejneru pro nového tenanta.
- **appr-fe-adminer-build**. Sestavení, otestování a vytvoření image frontend kontejneru s aplikací organizace.
- **appr-fe-adminer-deploy**. Nasazení frontend kontejneru s aplikací organizace pro nového tenanta.
- **appr-fe-adminer-update**. Aktualizace frontend kontejneru s aplikací organizace kontejneru pro nového tenanta.
- **appr-fe-client-build**. Sestavení, otestování a vytvoření image frontend kontejneru s aplikací klienta.
- **appr-fe-client-deploy**. Nasazení frontend kontejneru s aplikací klienta kontejneru pro nového tenanta.
- **appr-fe-client-update**. Aktualizace frontend kontejneru s aplikací klienta kontejneru pro nového tenanta.

Uvedené skripty nejsou součástí této práce, uveden je příklad, skript appr-backend-build, který je na ukázce kódu 13. Spuštění skriptů **appr-backend-build**, **appr-fe-adminer-build** a **appr-fe-client-build** je prováděno automaticky se změnou v repositářích daných projektů. Tím je v infrastruktuře zajištěna kontinuální integrace.

9 Testovaní

V této kapitole jsou popsány veškeré aspekty testování řešení vypracovaného dle návrhu.

9.1 Pokrytí testy

Portál se skládá z více částí, které jsou vhodné otestovat různými metodami. Pro současný stav celého systému byla určena následující forma a rozsah testování.

- **Backend aplikace**. Tato část je kritickou součástí celého systému. Důvodem je fakt, že obsahuje veškerou bussiness logiku a tím pádem se také jedná o nejobsáhlejší část celého portálu ve smyslu objemu zdrojového kódu. Je tedy vhodné vytvořit automatizované testy, které budou také sloužit ke kontrole dalších změn v kódu. Automatizované testy tvořeny programátorem s kompletní znalostí systému byly rozděleny na testy jednotkové a integrační. Integrační jsou pak ještě rozděleny na testy API a testy s načtením defaultní datové struktury a bez načtení. Jednotkové testy se zaměřují na testování jednotlivých tříd, především modelů (v případě, že se v dané třídě nachází logika k testování, metody jako *getter a setter* se netestují) a *repository* tříd, kde je využito anotace *@DataJpaTest*. U jednotkových testů se nenahrává celý aplikační kontext Spring frameworku a jejich průběh je tedy rychlý. Pomalejší je průběh integračních testů, kde se aplikační kontext načítá vždy. Integrační testy testují metody především tříd typu *service*, tedy například třídu *ContactService*, které využívá ke svým operacím mnoho dalších tříd typu *service*. Automatizované testy jsou napsány tak, aby pokrývaly nejdůležitější funkcionalitu a jsou průběžně doplňovány. Seznam existujících automatizovaných testů a jejich vyhodnocení pro stav vývoje aplikace v době psaní těchto odstavců je na obrázku 49.
- **Frontend aplikace**. Pro frontend, tedy klientská a administrátorská webová aplikace, byla zvolena cesta manuálního testování. Toto testování probíhá na třech úrovních - testování programátorem, systémové testování a akceptační testování. Akceptační testování je prováděno vedením společnosti. Testování je prováděno v různých webových prohlížečích.
- **Infrastruktura**. Infrastruktura je testována na úrovní testování programátorem a akceptačním testováním na firemním testovacím serveru.

9.2 Nástroje k testování

Pro automatické testování backend aplikace bylo využito známého testovací frameworku *JUnit* a neméně známého frameworku *Mockito*. Oba frameworky a některé další jsou součástí knihovny *spring-boot-starter-test*. Její výběr je založen na implementaci aplikace za použití frameworku Spring, jednalo se tedy o logickou volbu.

Testy se vykonávají a provádění přímo v IDE - nástroji IntelliJ IDEA, nebo na prostředí infrastruktury v Backend builder kontejneru (návrh infrastruktury, sekce 7.3). V obou případech je spuštění testů řízeno systémem Maven, na kterém je backend aplikace postavena. K manuálnímu testování backendu programátorem, především REST API, byl dále využit nástroj Postman.

K manuální testování frontend aplikací byly využity webové prohlížeče Google Chrome, Mozzila Firefox, Safari, Internet Explorer a Microsoft Edge.

9.3 Nasazení systému pro společnost

Jako jeden z přístupu testování portálu je možno označit nasazení portálu resp. aplikace klienta, správce a backend aplikace (tedy nikoli infrastruktury) do produkčního prostředí pro společnost, ve které autor práce pracuje. Portál byl nasazen v polovině roku 2018 v jedné z jeho prvních verzí. Od té doby je využíván, nachází se v něm několik tisíc kontaktů a jejich údajů a průběžně aktualizován. Ukázka této běžící instance aplikací je na obrázcích 43 a 44. Jedná se o jednu ze starších verzí trojice aplikací s některými individuálními úpravami v oblasti designu.

9.4 Vyhodnocení testů

Vyhodnocení backend testů

K vyhodnocování testů backend aplikace dochází při navržené infrastruktuře pokaždé, když je provedena změna v kódu a ten je nahrán do Version control kontejneru. Krok sestavení aplikace je následován krokem spuštění testů. Pokud testy neprojdou, není vytvořen obraz nové verze a k zavedení změn nedojde. Úspěšné vyhodnocení testů backend aplikace je tedy nutným krokem. Seznam úspěšně provedených testů je pro ilustraci na obrázku 49.

Automatické testy napomohly k lokalizování mnoha chyb a jsou velkou pomocí při každém zásahu do kódu.

Vyhodnocení frontend testů

Při testování frontend aplikací v různých prohlížečích byly výsledky velice uspokojivé. Ve všech prohlížečích byly stránky vyrendrovány jen s minimálními rozdíly. Vděk v tomto případě putuje směrem k vývojářům Angular a Angular Material. Při vývoji nebylo testování programátorem ve více prohlížečích pravidelně prováděno. Jako problémové se ukázaly pouze prohlížeče Internet Explorer a Microsoft Edge, kde se vyskytl problém s některými animacemi komponent z knihovny Angular Material.

Speciálním problémem se ukázal prohlížeč Internet Explorer ve verzi nižší než 11, kde je problém s implementací CORS (Mitchell, 2015) Fakticky tak dochází k tomu, že frontend aplikace se sice načtou, ale již přihlášení stejně jako jakýkoli další požadavek z frontendu na backend selže.

Akceptační testování vedením (resp. každé kolo tohoto testování) bylo velice přínosné hlavně z hlediska zpětné vazby na uživatelské prostředí a verifikace toho, že se vývoj ubírá správným směrem. Při těchto testech také došlo k definování různých dalším požadavků na systém, které byly některé rovnou zakomponovány do návrhu a některé zařazeny jako další rozšíření systému v delším časovém horizontu.

Vyhodnocení testů infrastruktury

Při testování infrastruktury byly zjištěny potíže s automatickým přidělováním certifikátů přes Let's Encrypt integrovaný v Traefik, které na vývojovém prostředí fungovalo bez obtíží. Problém se ukázal být záležitostí firemní sítě.

Dalším problémem při testech infrastruktury bylo selhávání Jenkins jobů při sestavování a testování backend aplikace. Příčinou problému bylo nedostatečné množství paměti RAM na testovacím serveru. Sestavování Java aplikací založených na Mavenu dokáže být velice paměťově náročné.

Za úspěch otestování infrastruktury lze považovat scénář, kdy se po konfiguraci statických kontejnerů (respektive aplikací, které je implementují) a nasazení několika demo tenantů server restartovat a po jeho naběhnutí se všechny požadované kontejnery samy spustily a celá infrastruktura fungovala tak, jak před restartem.

Vyhodnocení nasazení aplikace ve společnosti

Nasazení portálu do produkčního prostředí bylo důležitým krokem. Osoba, která v dané společnosti s osobními údaji pracuje, hlásila hned několik problémů. Jednalo se o mnoho drobných chyb a vylepšení v oblasti navigace a uživatelské zkušenosti. Konkrétně změna mnoha textů v uživatelském prostředí jako jsou popisky tlačítek, nebo názvy sekcí. Dále například o požadavek na přidání potvrzovacích oken při některých akcích (odeslání emailu na adresu kontaktu).

Dalším výstupem testování byly některé záležitosti s výkonem. Příkladem je filtrování velkého množství kontaktů, kdy vlivem použití automaticky generovaných dotazů ORM technologií Hibernate bylo toto filtrování neefektivní a bylo nutné na výsledky čekat dlouho. To ovlivňovalo celkovou uživatelskou zkušenost. Tento problém vznikl nedostatečným testováním programátorem a automatickým testováním při vývoji. Aplikace totiž nebyla otestována pro velké objemy dat. Řešením byla optimalizace algoritmu pro načítání entit z databáze (využití jazyka JPQL - dotazovacího jazyka využívaného v Java jako součástí JPA) a zavedení stránkování výsledků na rozhraní Backend aplikace.

V rámci produkčního nasazení byla také definována celá řada dalších požadavků na aplikaci, které jsou ale nad rámec původního návrhu a této práce. Některým z nich se věnuje kapitola následující kapitola 10 - Rozšíření.

10 Další rozvoj a rozšíření

10.1 Systémová integrace

Rozšířením, které je zcela klíčové a bude náplní práce autora textu a softwaru v nejbližší době po dokončení této práce je systémová integrace.

V situaci, kdy vyvíjený portál poskytuje centralizované místo uložení osobních údajů v podniku vzniká pravé využití jeho potenciálu v provázáním se systémy, které dále osobní údaje zpracovávají a to tak, aby například upravení údajů jejich subjektem bylo automaticky propsáno do integrovaného systému. Případně aby takovou akci jen zpracovatel potvrdil a k upravení by přes API došlo k automaticky. Smyslem je především zbavení se manuálního přepisování dat mezi systémy. Typickými systémy vhodné k integraci jsou podnikové systém CRM nebo mailing systémy. Idea je taková, že pro rozšířené systémy by existovaly přímo moduly v backend aplikaci portálu, které by byly konfigurovatelné z administrátorského uživatelského prostředí (například zadání adres a přístupových údajů k externím systémům). Dalším možným řešením by bylo vytvoření nové mikroservisy za účelem těchto integrací.

Pro tato rozšíření je portál dobře připraven díky REST API, existenci entity pro systémy zpracování a také žurnálování změn dat.

10.2 Orchestrace kontejnerů

S větším komerčním úspěchem portálu, přibývajícími tenanty, zvětšováním vytíženosti a přidávání dalším mikroservisů je možnost, že by bylo vhodné navrhovanou mirkoservisní infrastrukturu rozšířit o využití některého z nástrojů pro orchestraci kontejnerů jako je Docker Swarm, Apache Mesos a nebo nejrozšířenější Kubernetes. Tyto nástroje by mohly pomoci škálovatelnosti celého řešení.

10.3 Automatické aktualizace

V nynější portálu podobě jsou aktualizace aplikací řešeny tak, že sice je automaticky vytvořen Docker image pro každou ze tří částí aplikační vrstvy na základě nového kódu, ale již běžící instance nejsou automaticky aktualizovány. Takovéto řešení je pro současný stav přijatelné, ale zároveň je to řešení kompromisní. Portál se totiž stal projektem natolik rozsáhlým, že není v silách jednoho vývojáře průběžně navrhovat a implementovat řešení na základě nových požadavků a současně vytvářet dostatečné pokrytí celého kódu automatizovanými testy. Dále by se daly napsat automatizované testy i na frontend, čímž by se spolehlivost a důvěra v to, že se chyba nedostane do produkčního prostředí zvětšila. V důsledku popsané situace tak nejsou v současné verzi portálu aplikace aktualizací pro běžící plně automatizované a tento krok je zařazen do dalšího rozšíření. Zároveň jsou automatizované aktualizace jedna ze záležitostí, které by mohla řešit některá z technologií pro orchestraci kontejnerů zmíněných v předchozí sekci.

10.4 Internacionalizace

Dalším z rozšíření, které je plánováno jsou jazykové překlady aplikace. Vzhledem k dokonalému oddělení frontendu a backendu se překlad týká pouze dvou implementovaných frontend aplikací. Frameworku Angular nabízí pro překlady podporu.

10.5 Monitoring a logování

Jedním z plánovaných rozšíření je zavedení komplexního systému pro monitorování a logování. Plánem je logování jak tenantských kontejnerů, tak serveru či serverů, kde infrastruktura běží. Motivací pro zavedení monitoring a externího nástroje pro sběr a analýzu logů je sledování výkonu, dostupnosti, chybových hlášení a mnoha dalších provozních dat. Nástroje, kterým je v plánu toto rozšíření implementovat jsou open source nástroje Kibana, Elasticsearch a Logstash - jako celek také nazývaný jako *ELK stack*. Nástroje jsou nasaditelné jako mikroservisy v Docker kontejnerech. Uvažovaným nástrojem k monitoringu je software Zabbix.

11 Závěr

Tato práce se zabývala návrhem a implementací portálu pro evidenci a správu osobních údajů a souhlasů o jejich zpracování. Důležitou vlastností portálu se stala realizace na bázi cloudově nabízené a konfigurovatelné služby. Aby bylo možné definovaného cíle dosáhnout, bylo nutné nastudovat oblast zpracování osobních údajů v informačních systémech, zvláště pak nařízení GDPR. Další studium bylo nutné provést v oblasti vývoje a architektury software pro cloud. Na základě studia byly stanoveny metody k realizaci cíle a následně vytvořen návrh a jeho implementace.

Z technologického hlediska je výsledkem systém založen na mikroservisní architektuře, který je implementovaný moderními technologiemi (Angular SPA, Spring Boot MVC aplikace v Java 11) a je velmi dobře připraven na další rozšíření a rozvoj. Díky vytvoření REST rozhraní pokrývající veškerou funkcionalitu a naprostého oddělení byznys logiky a uživatelského prostředí je umožněna snadná systémová integrace. Systém byl implementován dle doporučených postupů pro podnikové (enterprise) aplikace a důraz byl také věnován uživatelské zkušenosti (UX). Požadované konfigurovatelnosti řešení bylo dosaženo implementací uživatelského rozhraní, které umožňuje definovat vlastní schéma zpracovávaných dat v systému. Multi-tenantnost je řešena částečně na úrovni aplikační, primárně pak na úrovni infrastruktury, kde bylo využito kontejnerizace pomocí technologie Docker.

Z hlediska naplnění očekávání a realizace konceptu byly požadavky splněny. Portál představuje centralizované místo podniku pro uchovávání a práci s osobními údaji a souhlasy, které kromě přístupu ze strany zpracovatele nabízí i přístup subjektům zpracovávaných osobních dat. Řešení podporuje řadu procesů a požadavků dle nejnovější platné legislativy a nabízí pomocné nástroje jako systém notifikací a žurnálování. Vybudovaná infrastruktura umožňuje portál poskytovat jako službu (SaaS) a pro poskytovatele služby nabízí automatizaci souvisejících procesů a rozhraní pro řízení cloudu.

Pro současný stav portálu bylo již definováno několik plánovaných rozšíření, z nichž hlavním bodem je systémová integrace, která umožní plně využít potenciálu centralizovaného řešení.

Testování řešení při vývoji probíhalo a probíhá na několika úrovních. Nejdůležitějším ověřením řešení je ale nasazení portálu ve společnosti, jenž je autor zaměstnancem. Nasazení v produkčním prostředí se zdařilo a portál je denně využíván. Tím lze považovat řešení za úspěšné a cíl práce splněn.

12 Literatura

Navrátil, J. *GDPR PRO PRAXI* Plzeň: Vydavatelství a nakladatelství Aleš Čeněk, 2018. ISBN 978-80-7380-689-7.

Nezmar, L. *GDPR: Praktický průvodce implementací* PGRADA Publishing, a.s., 2017. ISBN 978-80-271-0921-0.

Statista. *Facebook - revenue and net income* [online] [cit. 3. 3. 2019]. Dostupné z: statista.com/statistics/277229/facebooks-annual-revenue-and-net-income.

Greenwald, G. *Why privacy matters TED Talk* [online] [cit. 3. 3. 2019]. Dostupné z: ted.com/talks/glenn_greenwald_why_privacy_matters.

BusinessInfo. *Česku chybí adaptační zákon ke GDPR. Přesto začne pro tuzemské firmy platit* [online] [cit. 8. 3. 2019]. Dostupné z: businessinfo.cz/cs/clanky/cesku-chybi-adaptacni-zakon-ke-gdpr-presto-zacne-pro-tuzemske-firmy-platit-108545.html.

Škorničková, E. *Obecné nařízení o ochraně osobních údajů — prakticky* [online] [cit. 8. 3. 2019]. Dostupné z: gdpr.cz/gdpr/heslo/zpracovatel-osobnich-udaju.

Šandera, D. *Dopady Obecného nařízení o ochraně osobních údajů (GDPR) na firemní procesy* Diplomová práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, 2017.

AION CS. *101/2000 Sb. Zákon o ochraně osobních údajů.* [online] [cit. 17. 3. 2019]. Dostupné z: zakonyprolidi.cz/cs/2000-101.

Škorničková, E. *Kompletní znění GDPR česky a anglicky* [online] [cit. 19. 3. 2019]. Dostupné z: gdpr.cz/gdpr/kompletni-zneni-gdpr.

Úřad pro ochranu osobních údajů. *Základní příručka k GDPR* [online] [cit. 19. 3. 2019]. Dostupné z: uoou.cz/zakladni-prirucka-k-gdpr/ds-4744/p1=4744.

ESET. *Význam šifrování v nařízení GDPR* [online] [cit. 14. 3. 2019]. Dostupné z: eset.com/cz/sifrovani/.

Koupilová, O. *HELIOS Green GDPR Ready* [online] [cit. 17. 3. 2019]. Dostupné z: helios.eu/systemove-stranky/newsletter/helios-green-gdpr-ready/.

Arlow, J. *UML 2 a unifikovaný proces vývoje aplikací: objektově orientovaná analýza a návrh prakticky.* Brno: Computer Press, 2007. ISBN 978-80-251-1503-9.

Fowler, M. *Destilované UML.* Praha: Grada, 2009. Knihovna programátora (Grada). ISBN 978-80-247-2062-3.

Hajník, J. *Analýza a návrh aplikace s využitím UML*. Bakalářská práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, Katedra informačních technologií, 2010.

Rábová, I. *Podnikové informační systémy a technologie jejich vývoje*. Brno: Tribun EU, 2008. ISBN 978-80-7399-599-7.

Machalický, J. *Tvorba a nasazení firemního informačního systému*. Diplomová práce. Mendelova univerzita v Brně, Provozně ekonomická fakulta, 2015.

Popelák, M. *Využití metodiky SCRUM ve vývojových týmech*. Diplomová práce. České vysoké učení technické v Praze, Masarykův ústav vyšších studií a Vysoká škola ekonomická v Praze, 2015.

Martin, R. *Clean code: a handbook of agile software craftsmanship*. Vupper Saddle River, NJ: Prentice Hall, 2009. ISBN 978-0-13-235088-4.

Stahl, T., Völter, M. *Model-driven software development : technology, engineering, management*. Chichester, England Hoboken, NJ: John Wiley, 2006. ISBN 978-0-470-02570-3.

Turnqueist, G. *Learning Spring Boot : simplify the development of lightning fast applications based on microservices and reactive programming*. Birmingham: Packt, 2017. ISBN 978-1-78646-378-4.

McConnell, S. *Code complete. 2nd ed.* Wash.: Microsoft Press, 2014. ISBN 978-0-7356-1967-8.

Sommerville, L. *Software engineering*. Singapore: Pearson, 2016 ISBN 978-1-292-09613-1.

Loeliger J. *Version control with Git*. Sebastopol, CA: O'Reilly, 2009. ISBN 978-0-596-52012-0.

Heffelfinger, D. *Java EE 8 Application Developement*. Packt Publishing Limited, S.L. ISBN 978-1-78829-367-9.

Duspiva, L. *Framework Spring Data pro řešení přístupu k datům v Java Enterprise aplikacích*. Diplomová práce. Univerzita Hradec Králové, Fakulta informatiky a managementu, Katedra informatiky a kvantitativních metod, 2016.

Fowler, M., Beck, K. *Refactoring: improving the design of existing code*. MA: Addison-Wesley, 1999. ISBN 978-0134757599.

Daschner, S. *Architecting modern Java EE applications*. Packt Publishing Limited, S.L. ISBN 978-1788293679.

JOHNSON, R. *Introduction to the Spring Framework* [online] [cit. 1. 4. 2019]. Dostupné z: `theserverside.com/news/1364527/Introduction-to-the-Spring-Framework`.

ORACLE CORPORATION. *Core J2EE Patterns - Data Access Object* [online] [cit. 1. 4. 2019]. Dostupné z: `oracle.com/technetwork/java/dataaccessobject-138824.html`.

FOWLER, M. [online] [cit. 3. 4. 2019]. *Inversion of Control Containers and the Dependency Injection pattern*. Dostupné z: `martinfowler.com/articles/injection.html`.

NEUHAUS, J. *Angular vs. React vs. Vue: A 2017 comparison* [online] [cit. 26. 3. 2019]. Dostupné z: `medium.com/unicorn-supplies/angular-vs-react-vs-vue-a-2017-comparison-c5c52d620176`.

KRAVCHENKO, I. *Angular or React, Which Is Better for Your Project Development?* [online] [cit. 24. 3. 2019]. Dostupné z: `diceus.com/react-vs-angularjs/`.

HAMEDANI, M. *React vs. Angular: The Complete Comparison* [online] [cit. 24. 3. 2019]. Dostupné z: `programmingwithmosh.com/react/react-vs-angular/`.

MITCHELL, P.. *XDomainRequest and CORS on IE9* [online] [cit. 1. 5. 2019]. Dostupné z: `http://perrymitchell.net/article/xdomainrequest-cors-ie9/`.

KAINULAINEN, P. *Understanding Spring Web Application Architecture: The Classic Way.* [online] [cit. 22. 3. 2019]. Dostupné z: `petrikainulainen.net/software-development/design/understanding-spring-web-application-architecture-the-classic-way/`.

MOUAT, A. *Using Docker*. Boston: O'Reilly, 2015. ISBN 978-1-491-91576-9.

CHACON, S., STRAUB, B. *Pro Git: [everything you need to know about the Git distibuted source control tool]*. New York, NY: Apress, 2014. ISBN 978-1-4842-0076-6.

BUTLER, T. *NGINX cookbook : make the most of your web server*. Birmingham, UK: Packt Publishing, 2017. ISBN 978-1786466174.

OKOSY, M. *Dockerizace Continuous Integration řešení pro aplikace vyvíjené v programovacím jazyce Java*. Diplomová práce. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, 2017.

MRAČKO, A.. *Využití nástroje Docker pro MUFIN*. Diplomová práce. Masarykova Univerzita, Fakulta informatiky, 2016.

Mirtes, O.. *Začněte monolitem* [online] [cit. 17. 5. 2019]. Dostupné z: ondrej.mirtes.cz/zacnete-monolitem.

Fowler, M. *Microservices, a definition of this new architectural term* [online] [cit. 3. 4. 2019]. Dostupné z: martinfowler.com/articles/microservices.html.

Newman, S.. *Building Microservices*. O'Reilly Media, Sebastopol, CA, 2015. ISBN 978-1491950357.

Fowler, M. *Production-ready microservices*. O'Reilly Media, Sebastopol, CA, 2017, ISBN 978-9352134878.

Farcic, V. *The DevOps 2.0 toolkit : automating the continuous deployment pipeline with containerized microservices*. Victoria, BC: Leanpub, 2016. ISBN 978-1523917440.

Kennedy, P.. *Docker Swarm Management: A Quick Overview of Rancher, Portainer and Shipyard* [online] [cit. 12. 4. 2019]. Dostupné z: servethehome.com/docker-swarm-management-a-quick-overview-of-rancher-portainer-and-shipyard.

AltexSoft. *Comparison of Most Popular Continuous Integration Tools: Jenkins, TeamCity, Bamboo, Travis CI and more.* [online] [cit. 14. 4. 2019]. Dostupné z: altexsoft.com/blog/engineering/comparison-of-most-popular-continuous-integration-tools-jenkins-teamcity-bamboo-travis-ci-and-more/.

Deitcher, A. *Mysql-backup* [online] [cit. 17. 5. 2019]. Dostupné z: github.com/databacker/mysql-backup.

Leslie, A. *NGINX vs. Apache (Review, Uses, Hosting for Each)* [online] [cit. 2. 3. 2019]. Dostupné z: hostingadvice.com/how-to/nginx-vs-apache/.

Procházka, D.. *Navigace v aplikaci*. Přednáška předmětu Pokročilá uživatelská rozhraní. Provozně ekonomická fakulta, Mendelova univerzita v Brně..

Smith, D. *Hype Cycle for Cloud Computing* [online] [cit. 11. 5. 2019]. Dostupné z: gartner.com/en/documents/2573318.

Velte, A., Elsenpeter, R. *Cloud Computing - praktický průvodce*. Brno: Computer Press, 2011. ISBN 978-80-251-3333.

Microsoft Corporation. *Co je SaaS? Software jako služba* [online] [cit. 11. 5. 2019]. Dostupné z: azure.microsoft.com/cs-cz/overview/what-is-saas/.

Tidwell, J. *Designing interfaces*. Beijing Sebastopol, CA: O'Reilly, 2006 ISBN 978-1449379704.

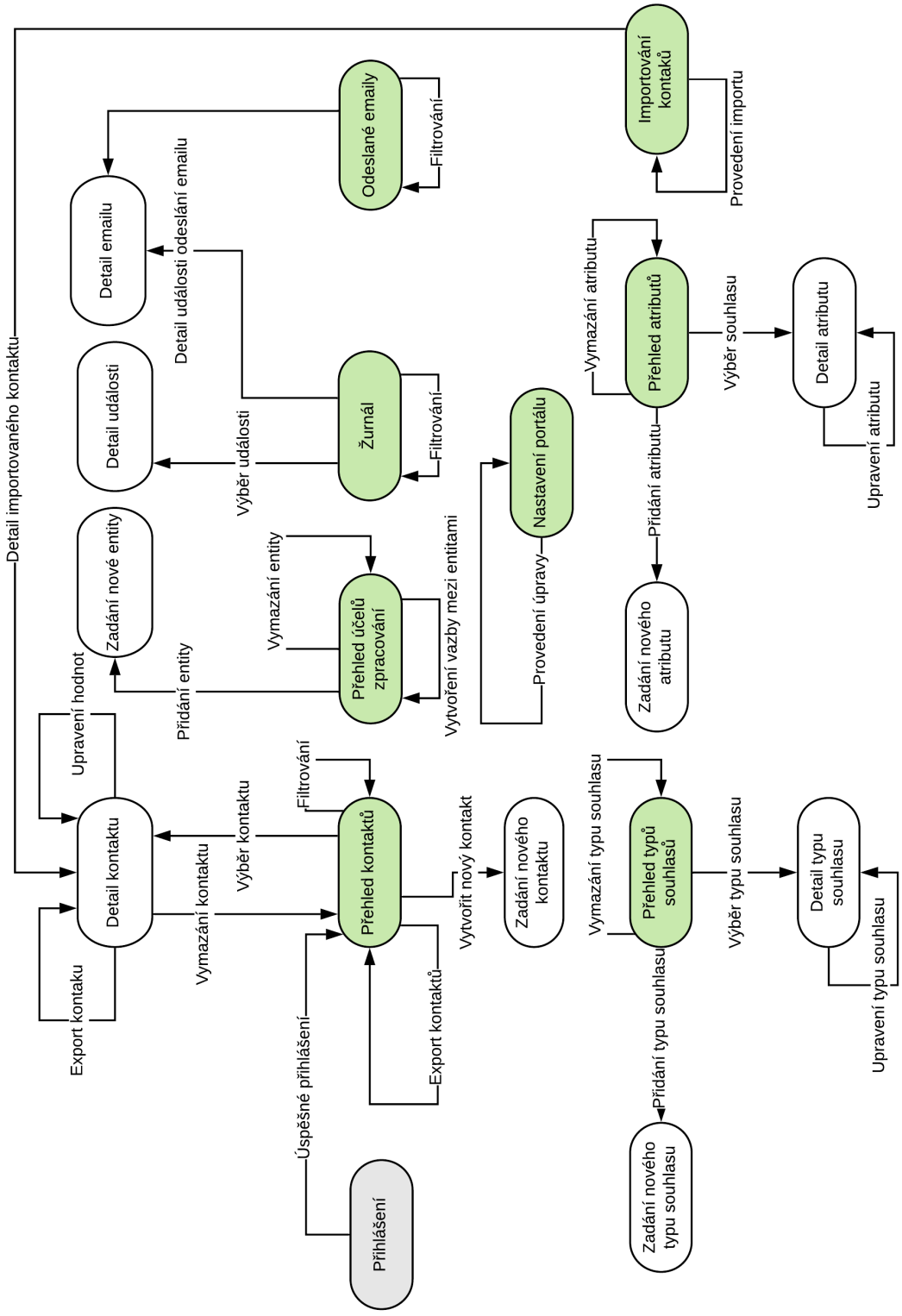
Carroll, J. *HCI models, theories, and frameworks: toward a multidisciplinary science*. San Francisco, Calif: Morgan Kaufmann, 2003. ISBN 978-0080491417.

Saffer, D. *Designing for interaction : creating innovative applications and devices*. CA London: New Riders Pearson Education distrsibutor, 2010. ISBN 978-0321643391.

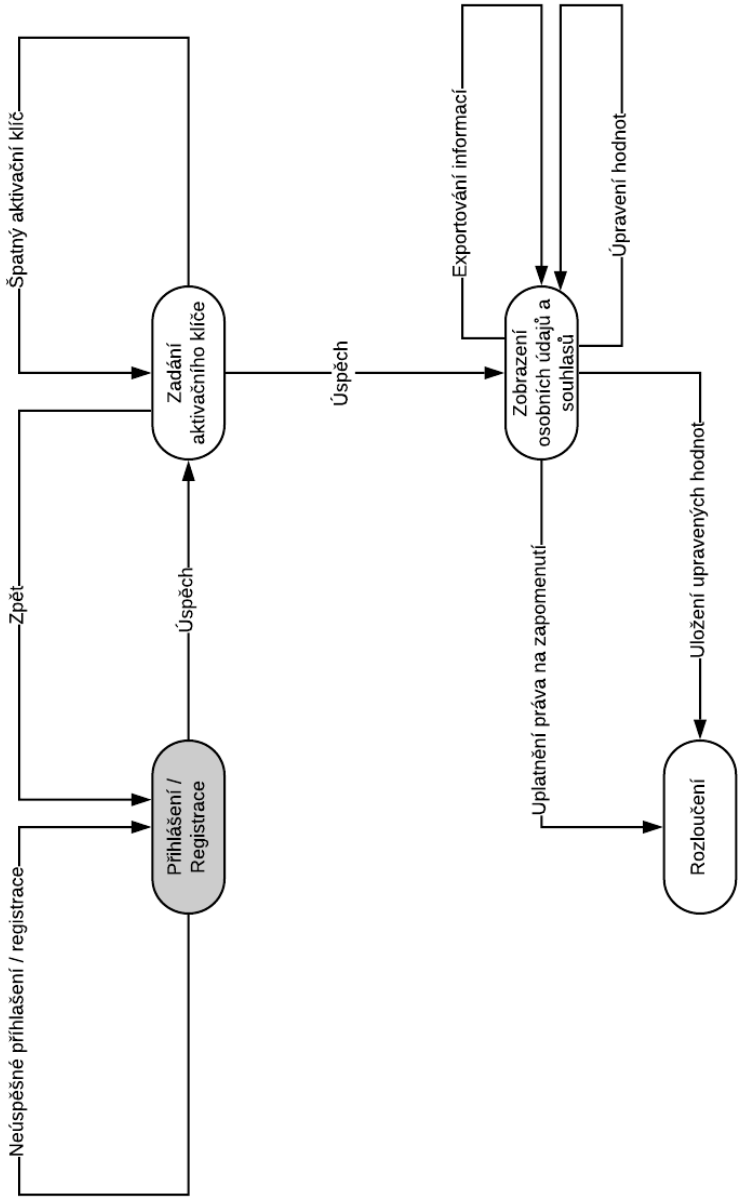
Krug, S. *Don't make me think, revisited: a common sense approach to Web usability*. San Francisco, California: New Riders, Peachpit, Pearson Education, 2014. ISBN 978-0321965516.

Přílohy

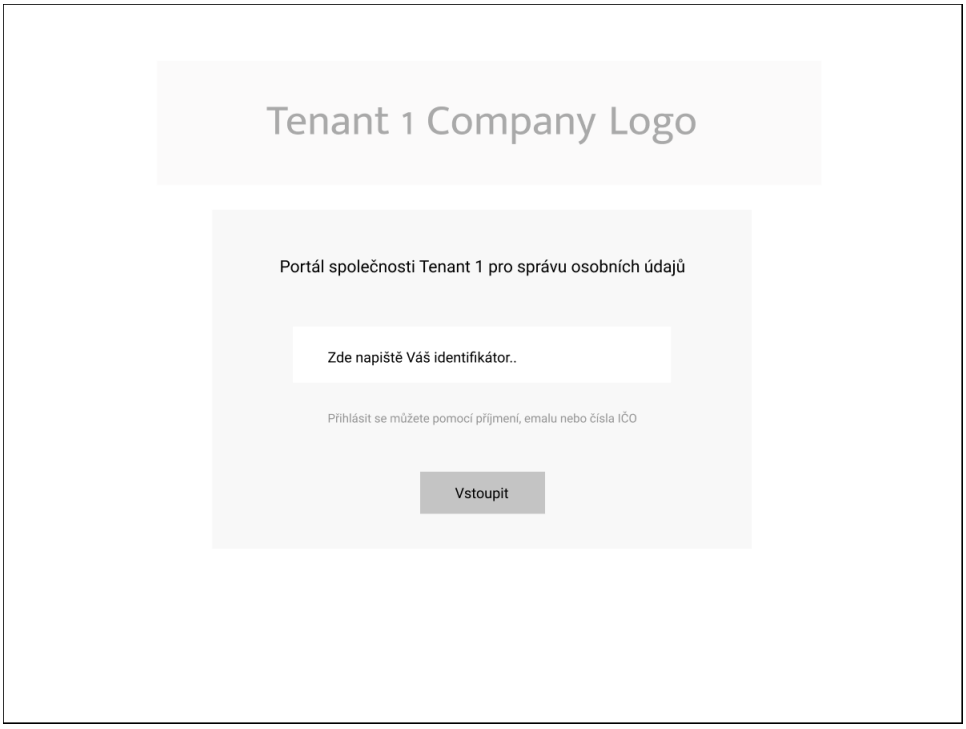
A Příloha - Návrh grafického uživatelského rozhraní



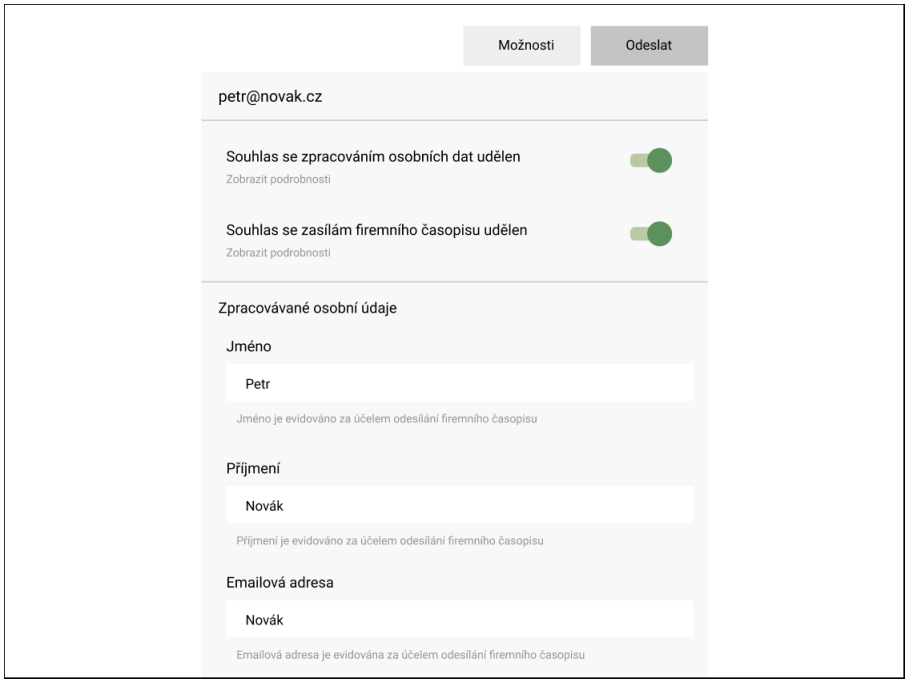
Obr. 21: Navigace - myšlenková mapa, aplikace organizace



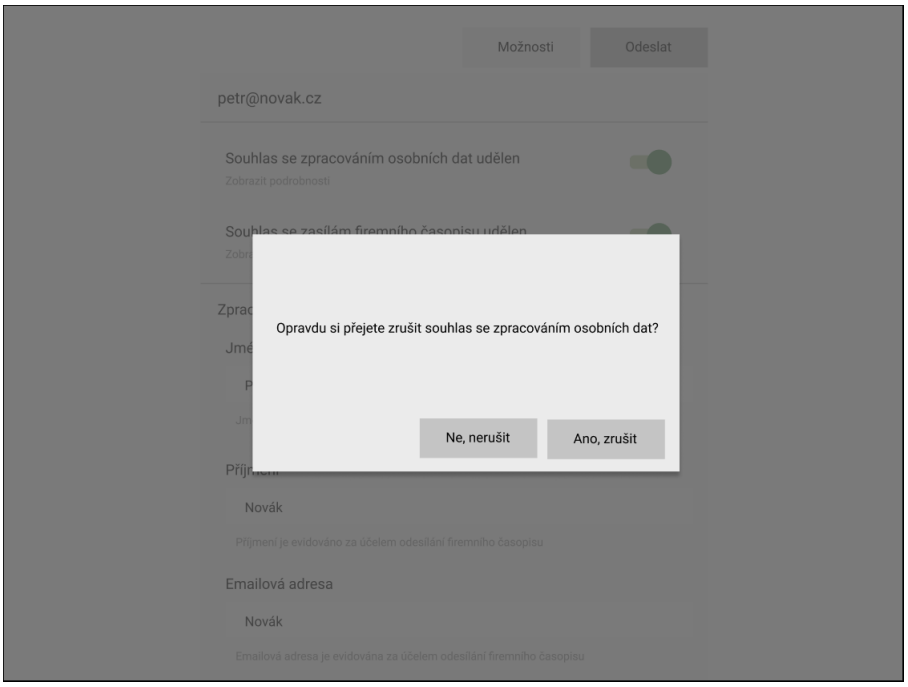
Obr. 22: Navigace - myšlenková mapa, aplikace klienta



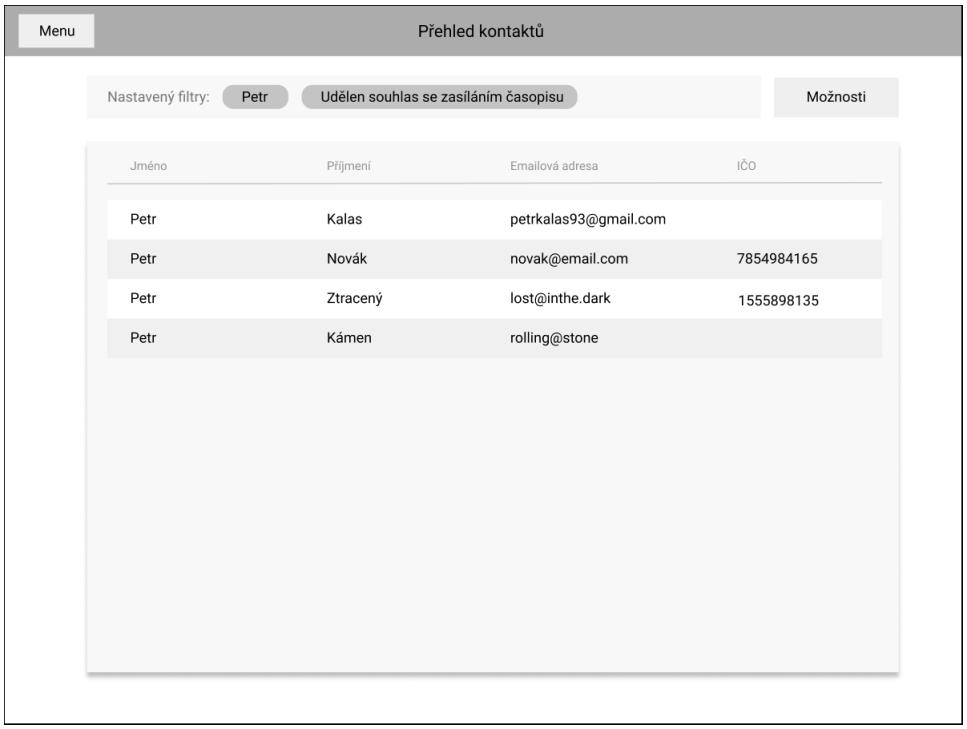
Obr. 23: Mockup - Aplikace klienta, přihlášení



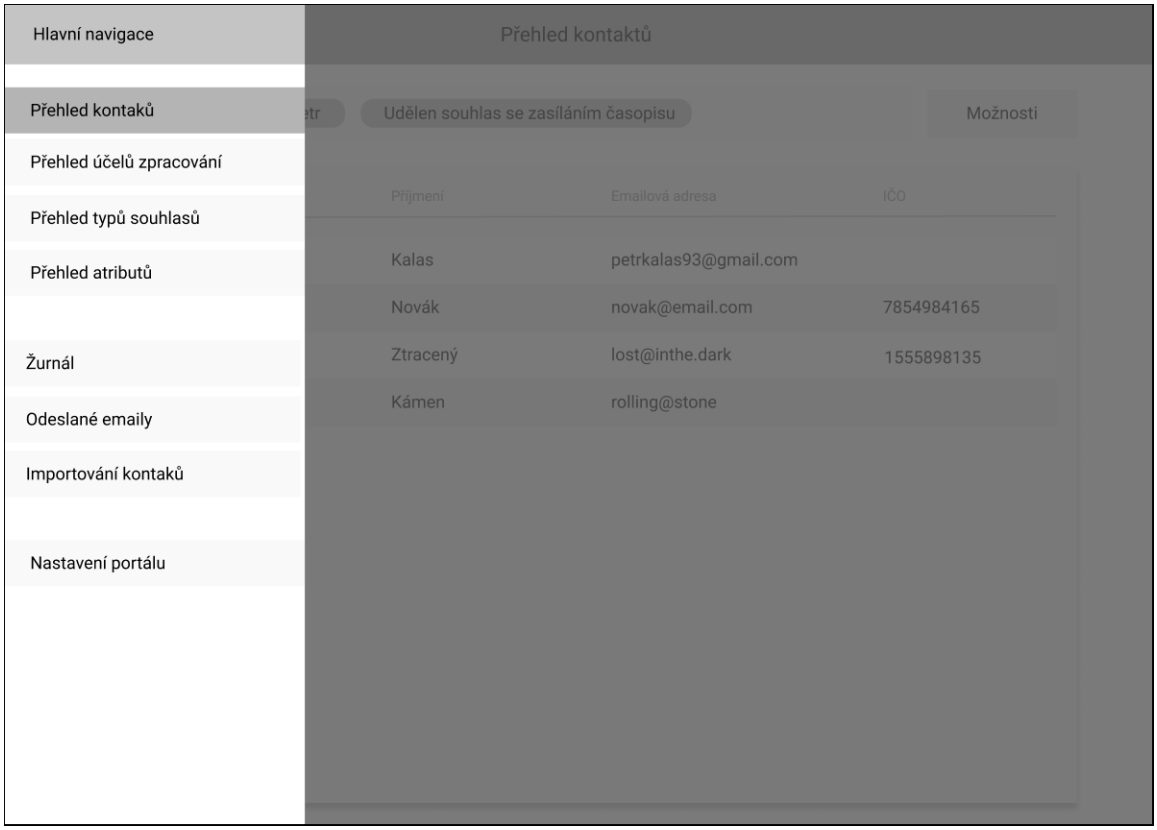
Obr. 24: Mockup - Aplikace klienta, výpis OÚ a souhlasů



Obr. 25: Mockup - Aplikace klienta, Modal Panel

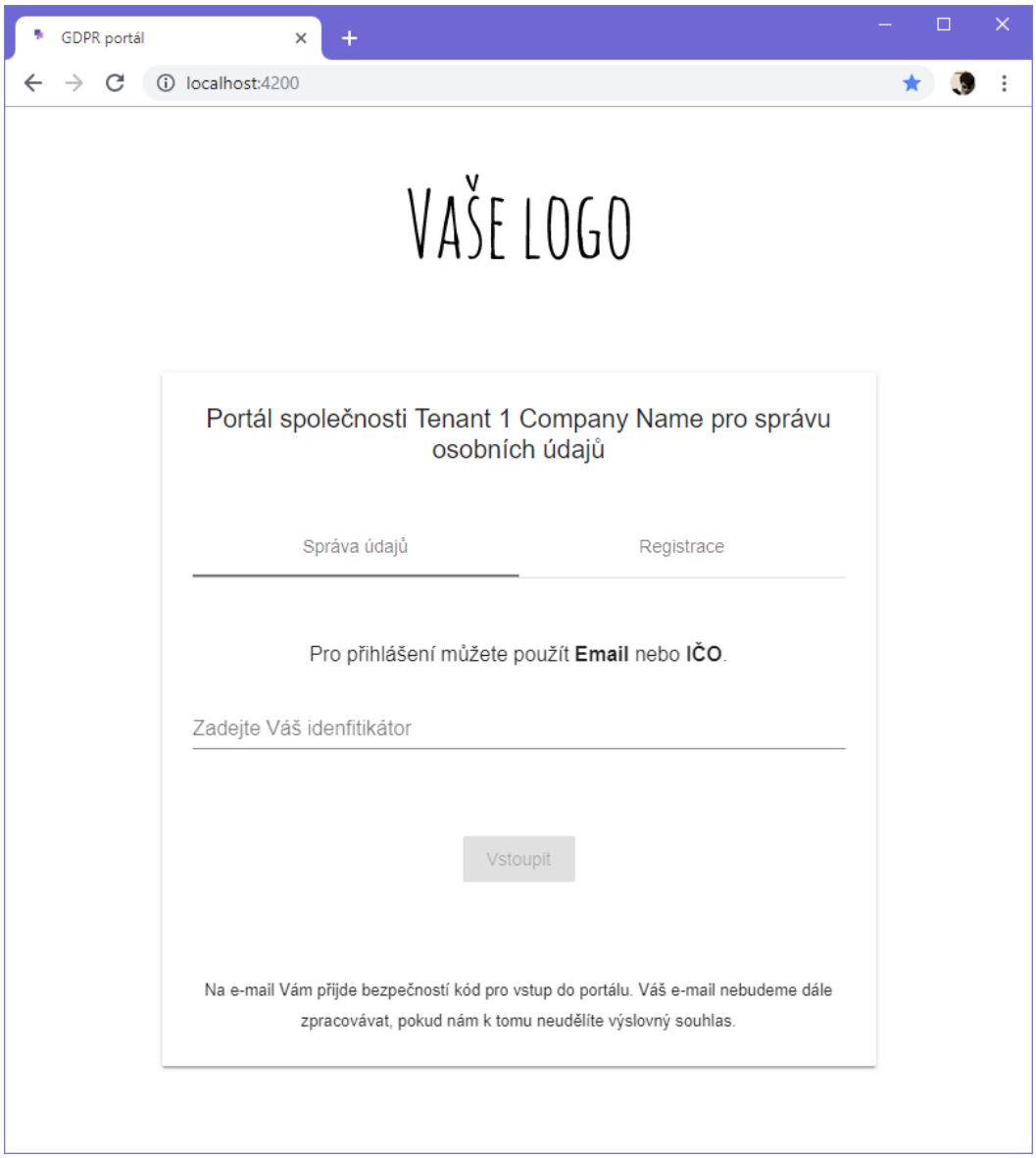


Obr. 26: Mockup - Aplikace organizace, přehled kontaktů

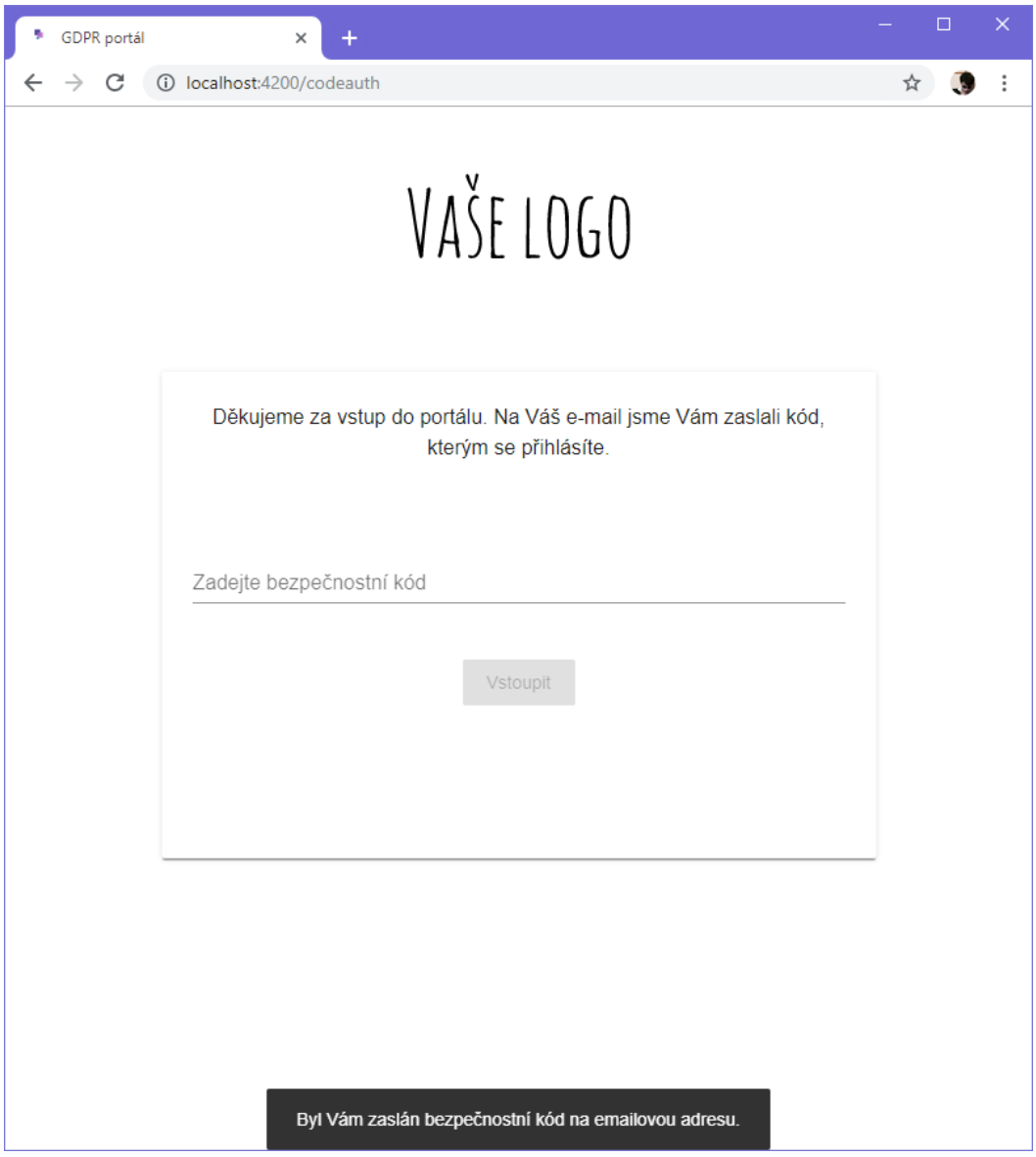


Obr. 27: Mockup - Aplikace organizace, globální navigace

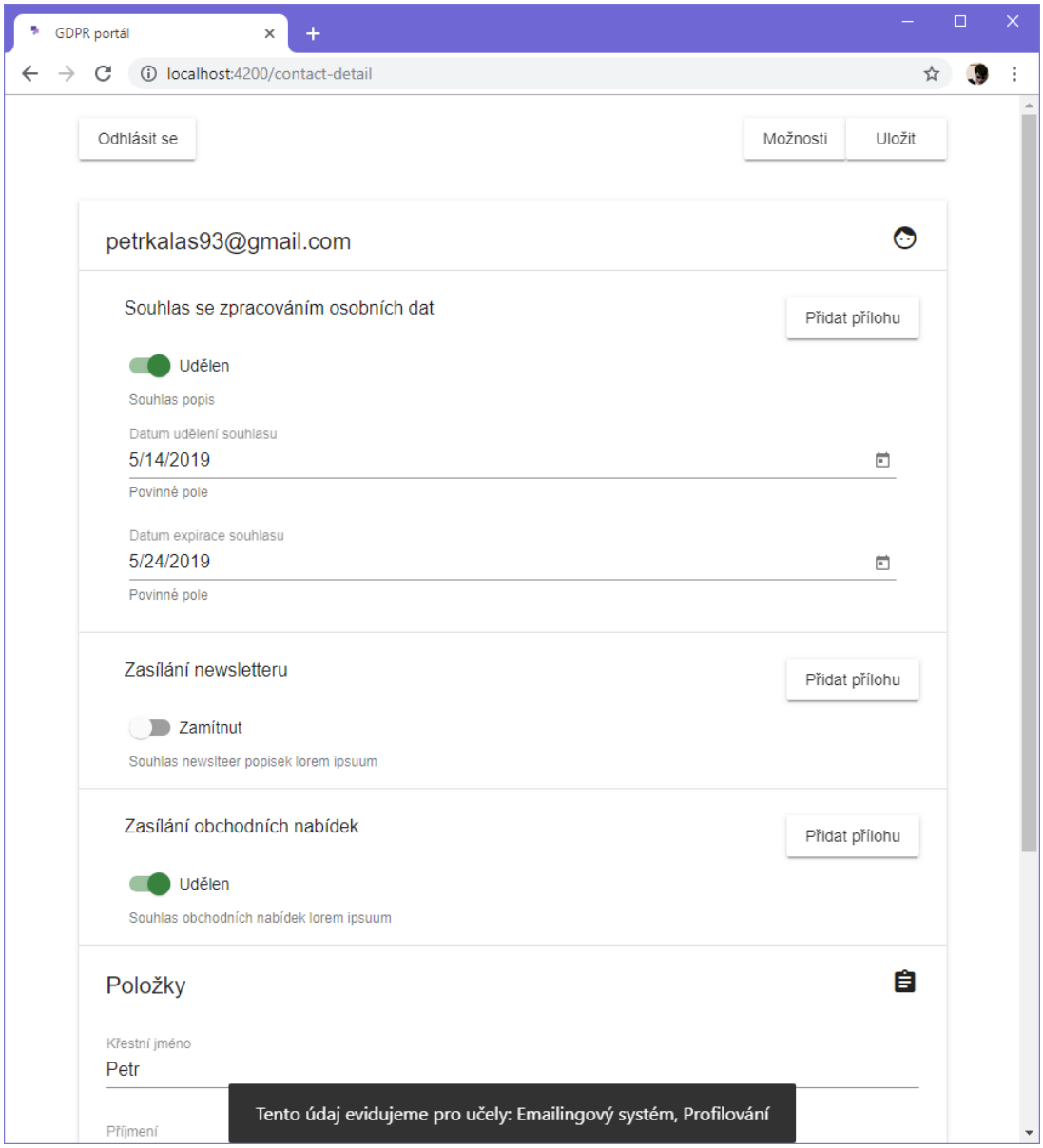
B Příloha - Implementace frontend aplikací



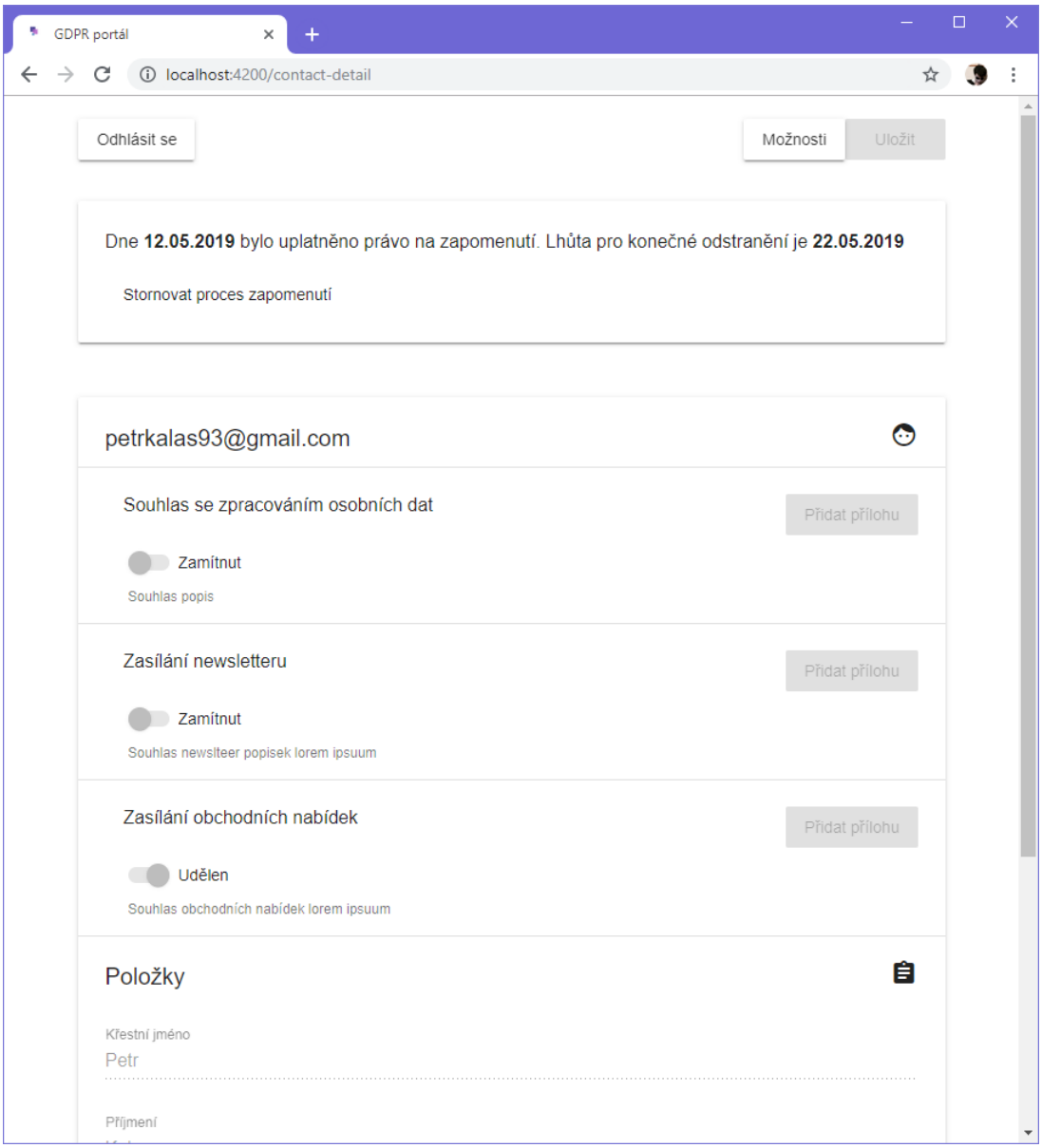
Obr. 28: GUI - Aplikace klienta, přihlášení / registrace



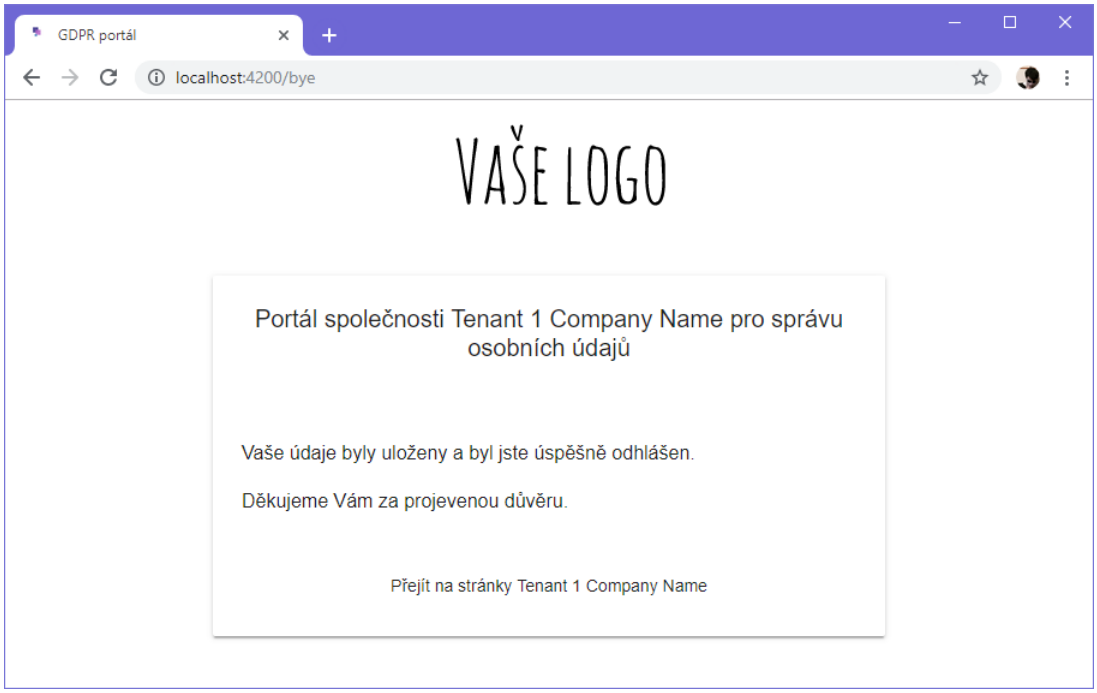
Obr. 29: GUI - Aplikace klienta, přihlašovací kód



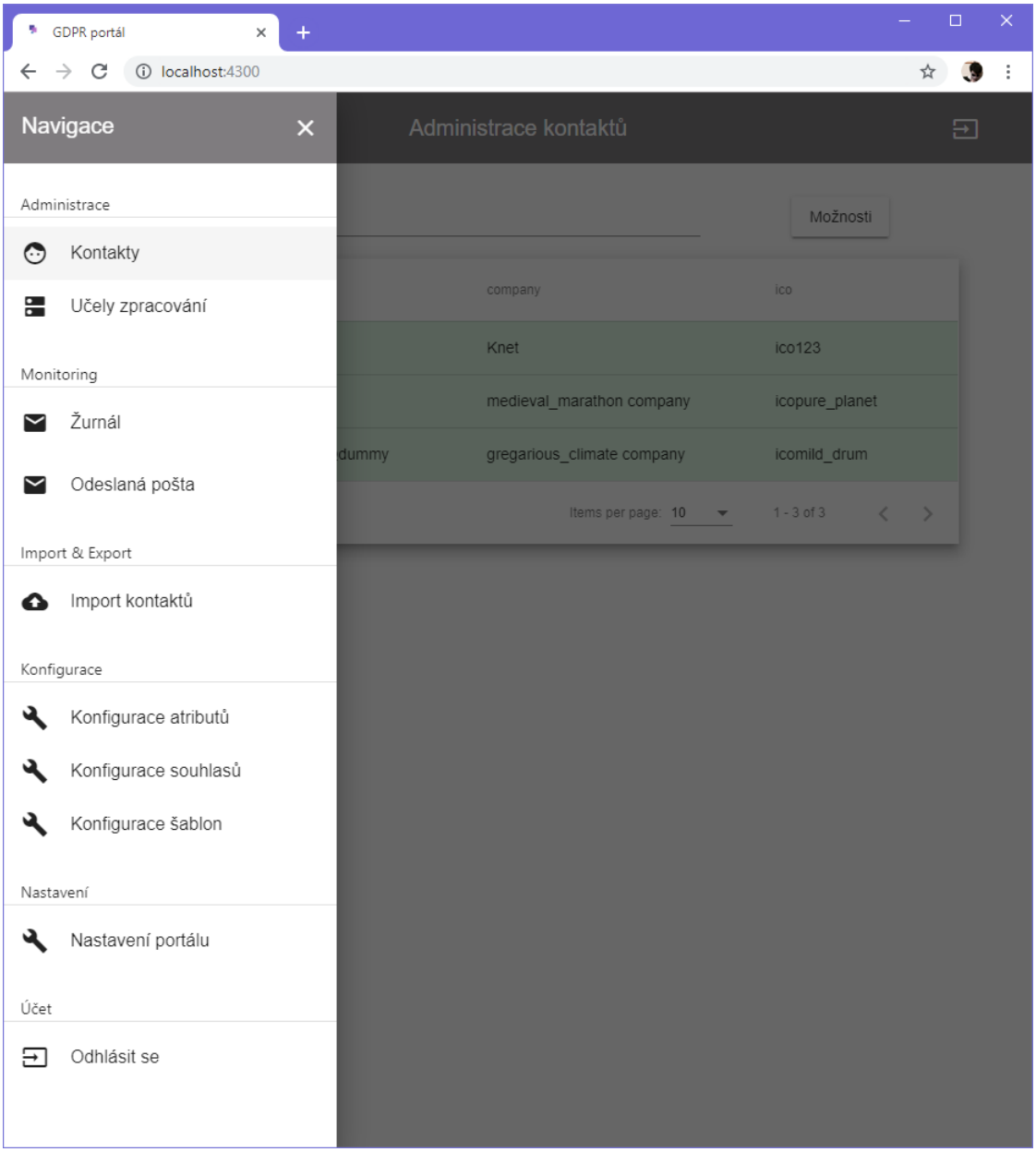
Obr. 30: GUI - Aplikace klienta, výpis OÚ a souhlasů



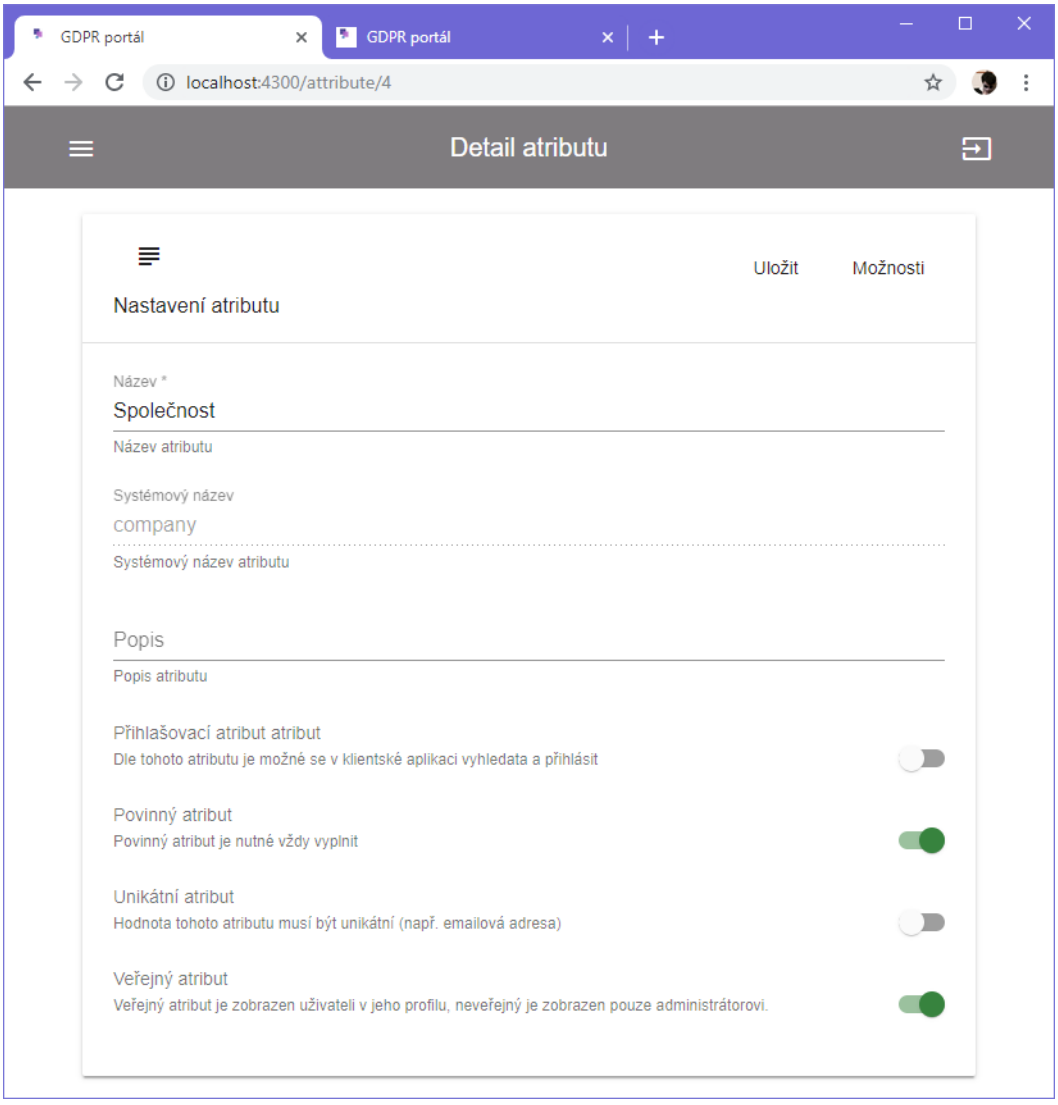
Obr. 31: GUI - Aplikace klienta, kontakt v procesu zapomenutí



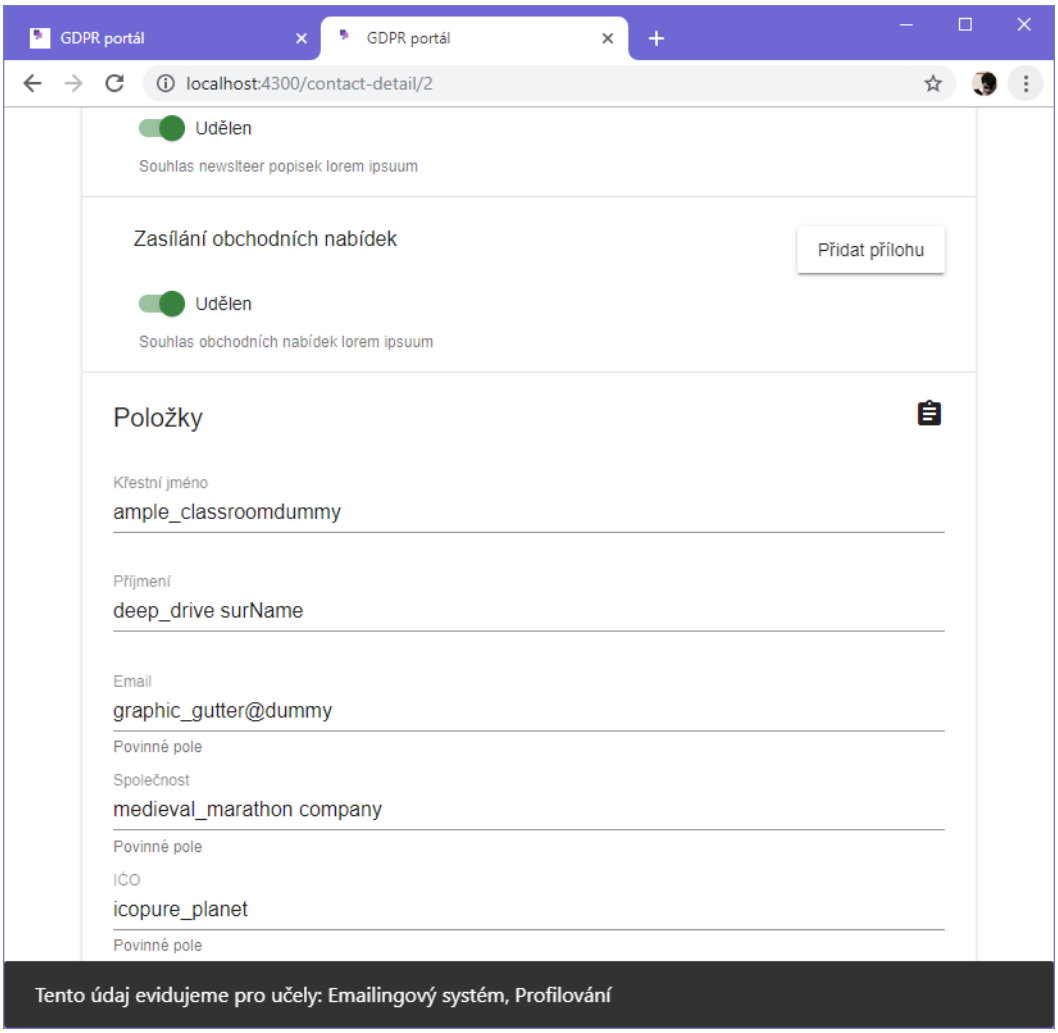
Obr. 32: GUI - Aplikace klienta, odhlášení - rozloučení



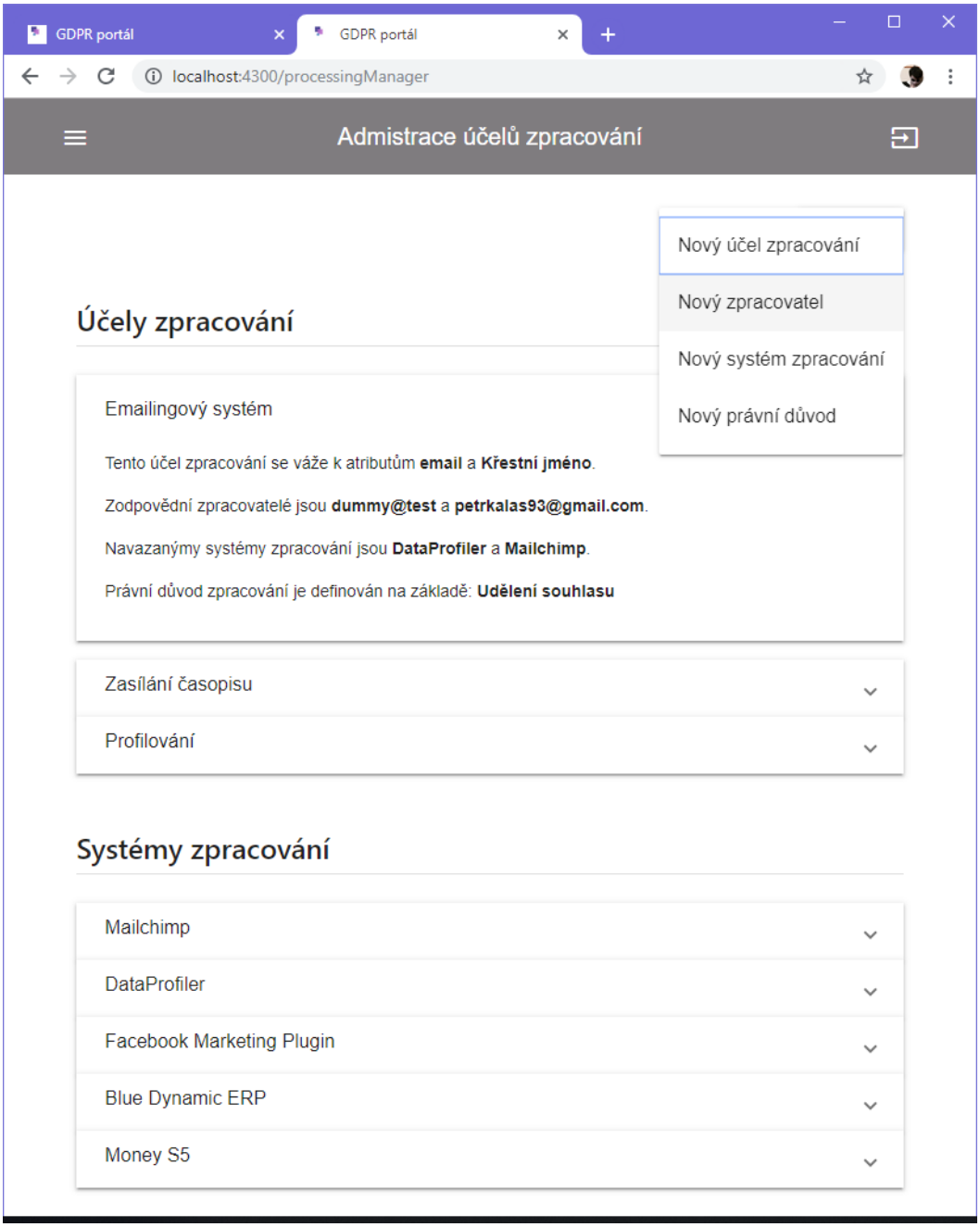
Obr. 33: GUI - Aplikace organizace, hlavní menu



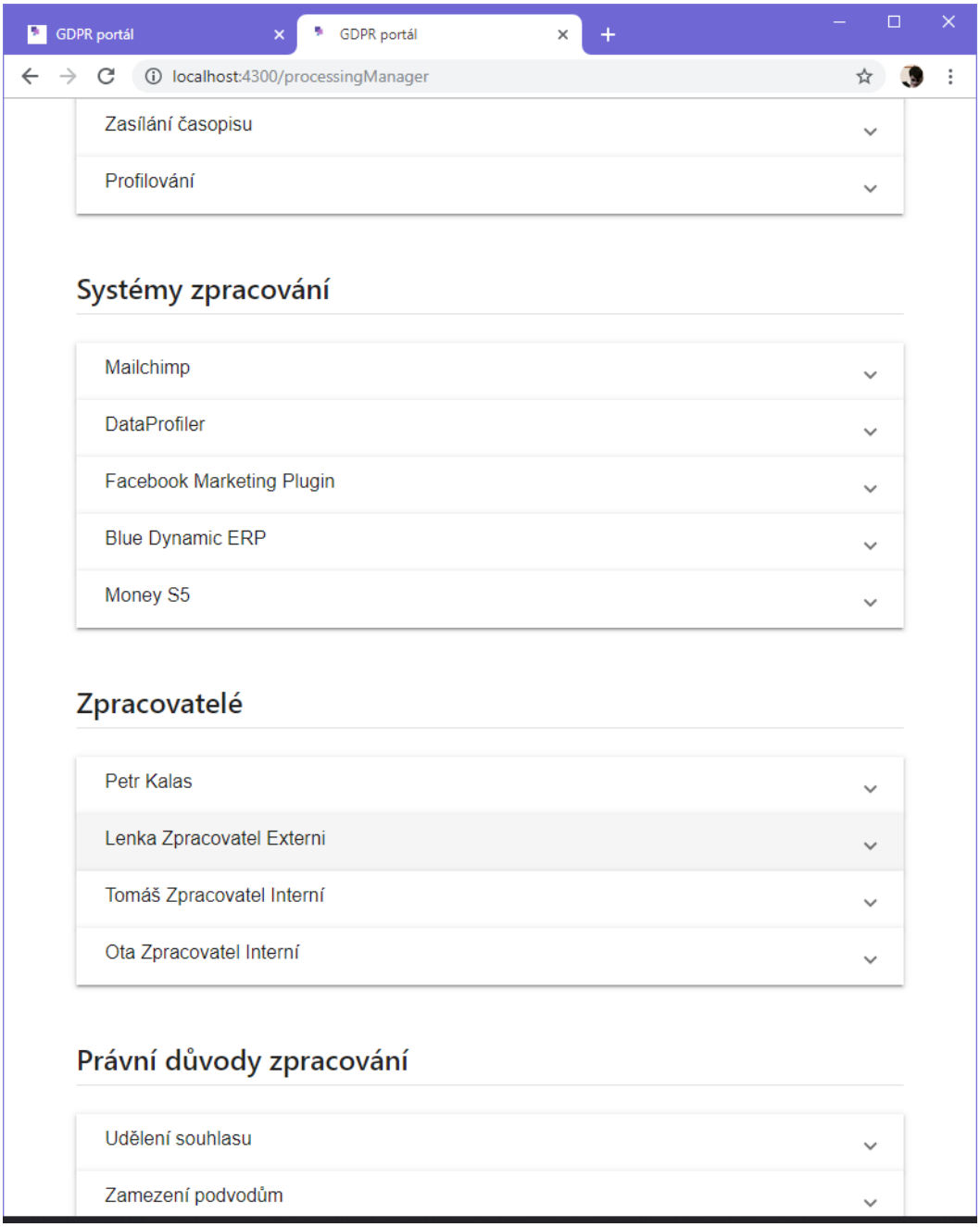
Obr. 34: GUI - Aplikace organizace, nastavení atributu



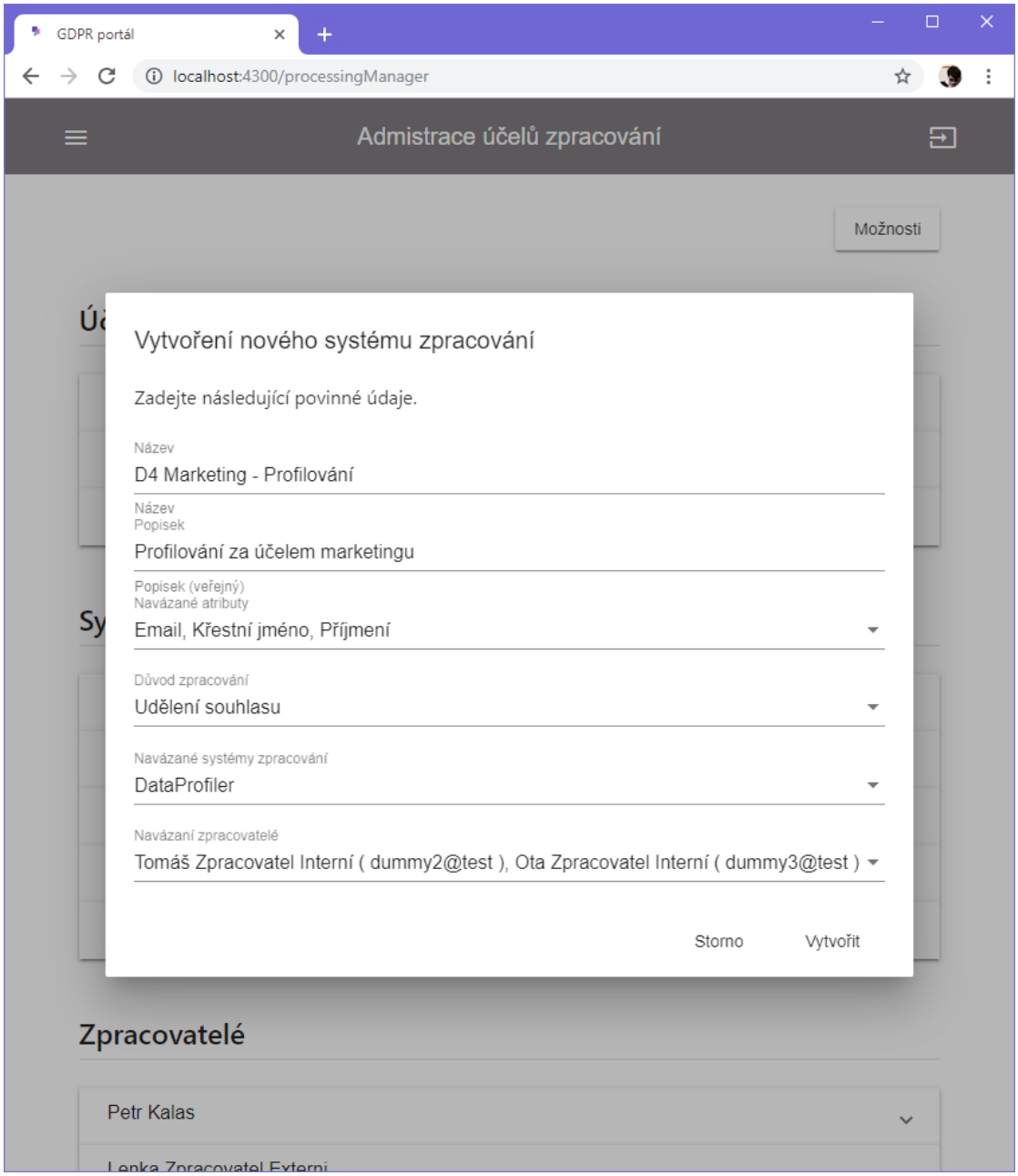
Obr. 35: GUI - Aplikace organizace, detail kontaktu



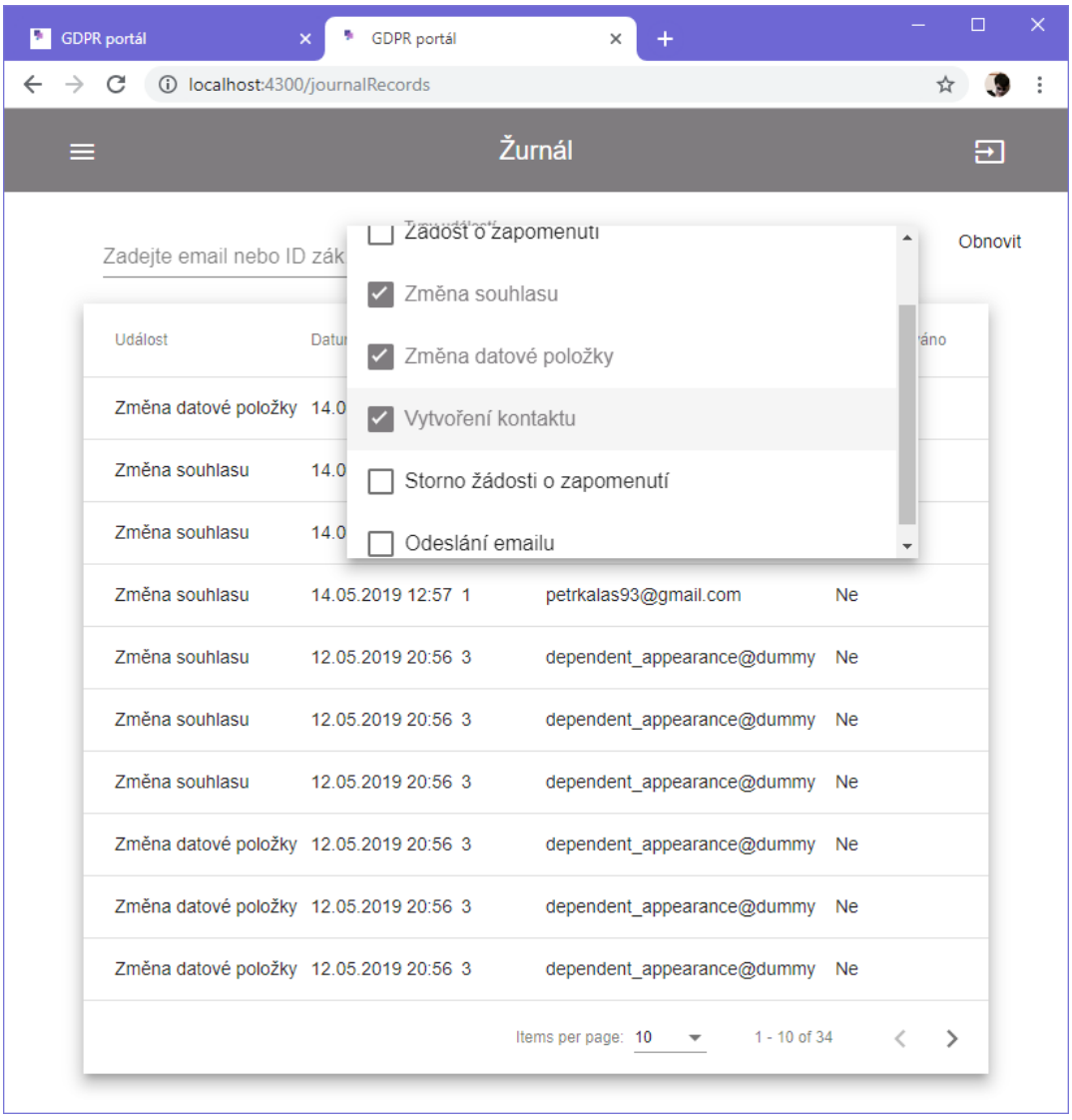
Obr. 36: GUI - Aplikace organizace, účely zpracování



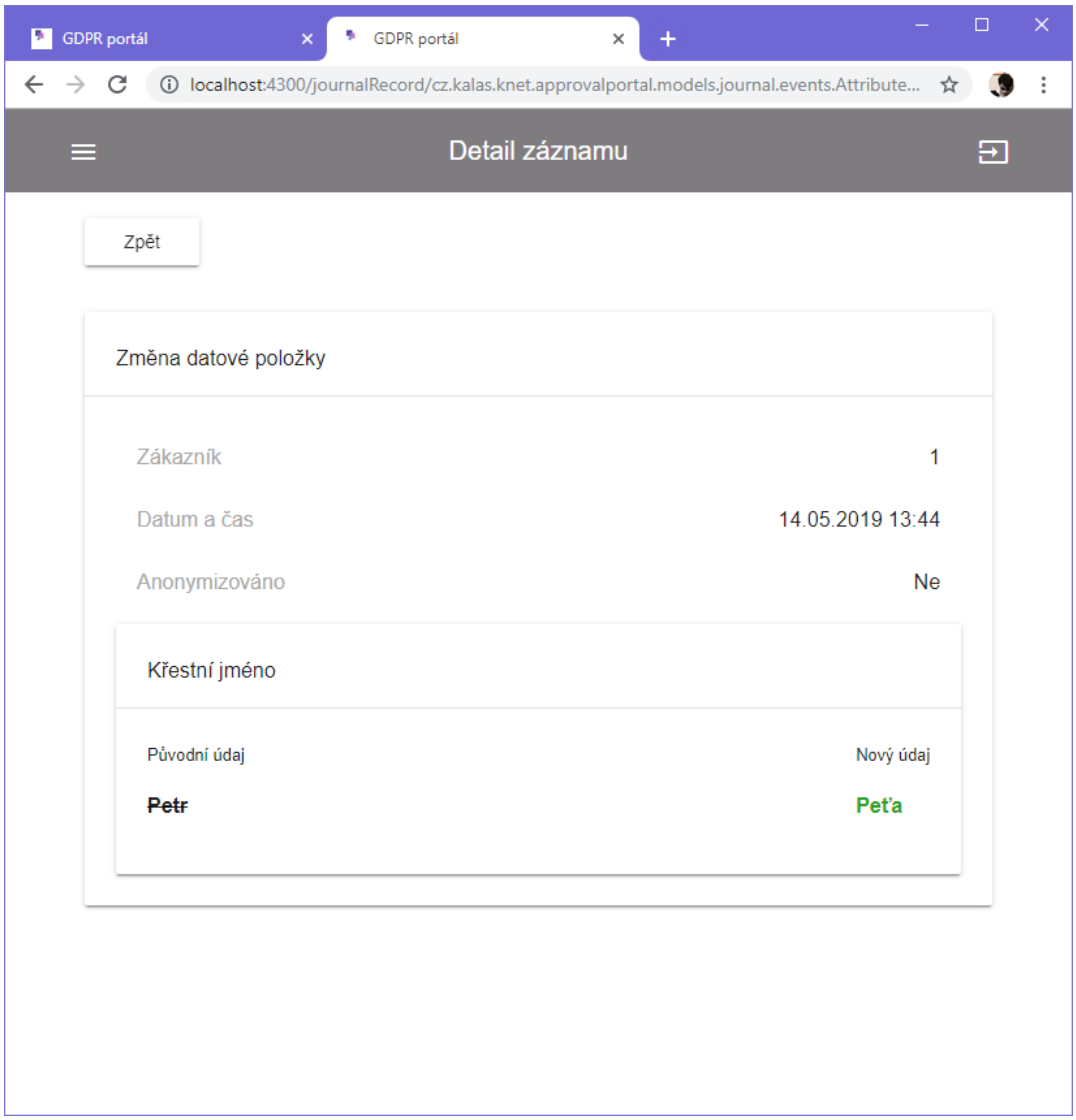
Obr. 37: GUI - Aplikace organizace, účely zpracování



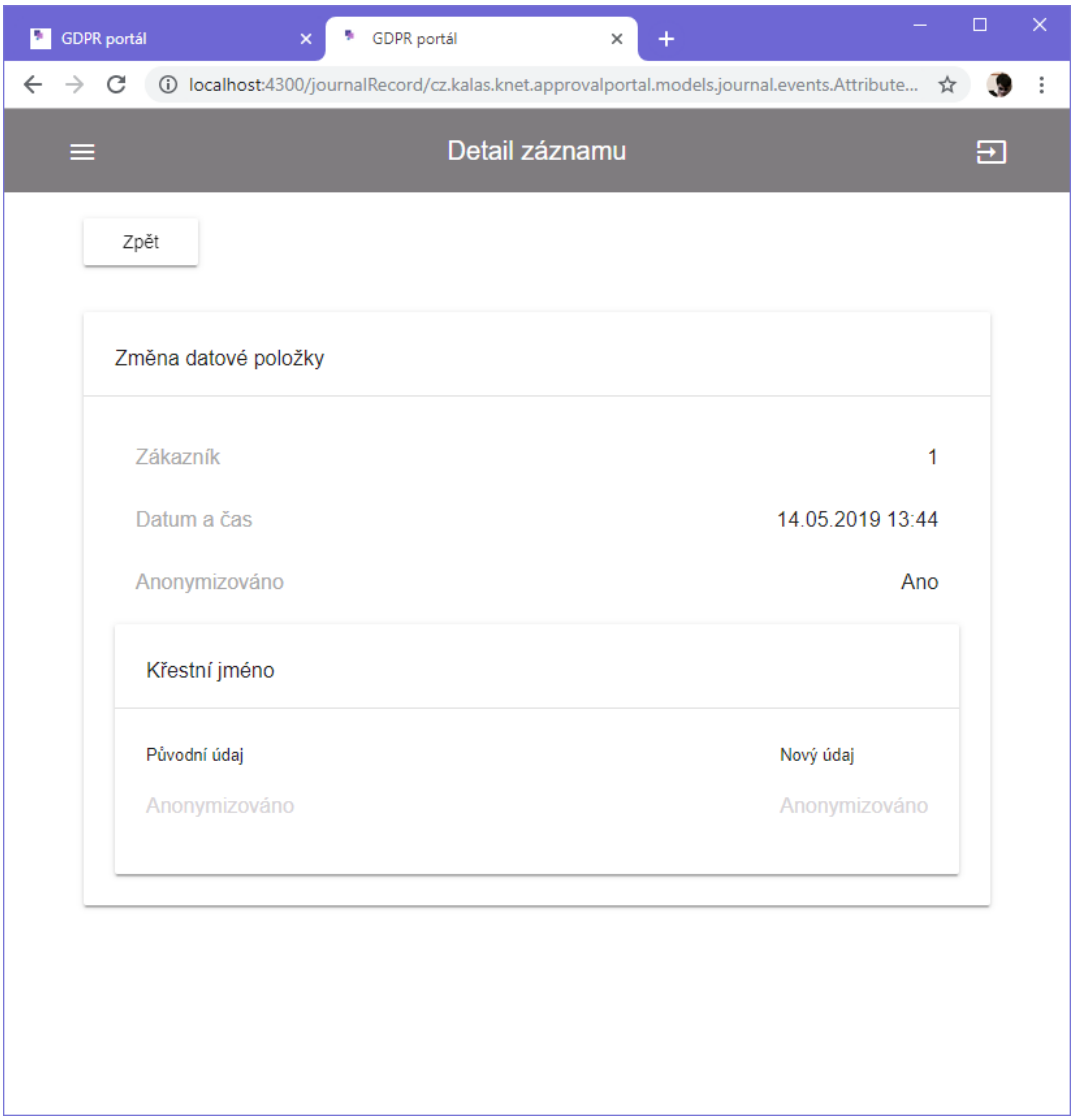
Obr. 38: GUI - Aplikace organizace, nový účel zpracování



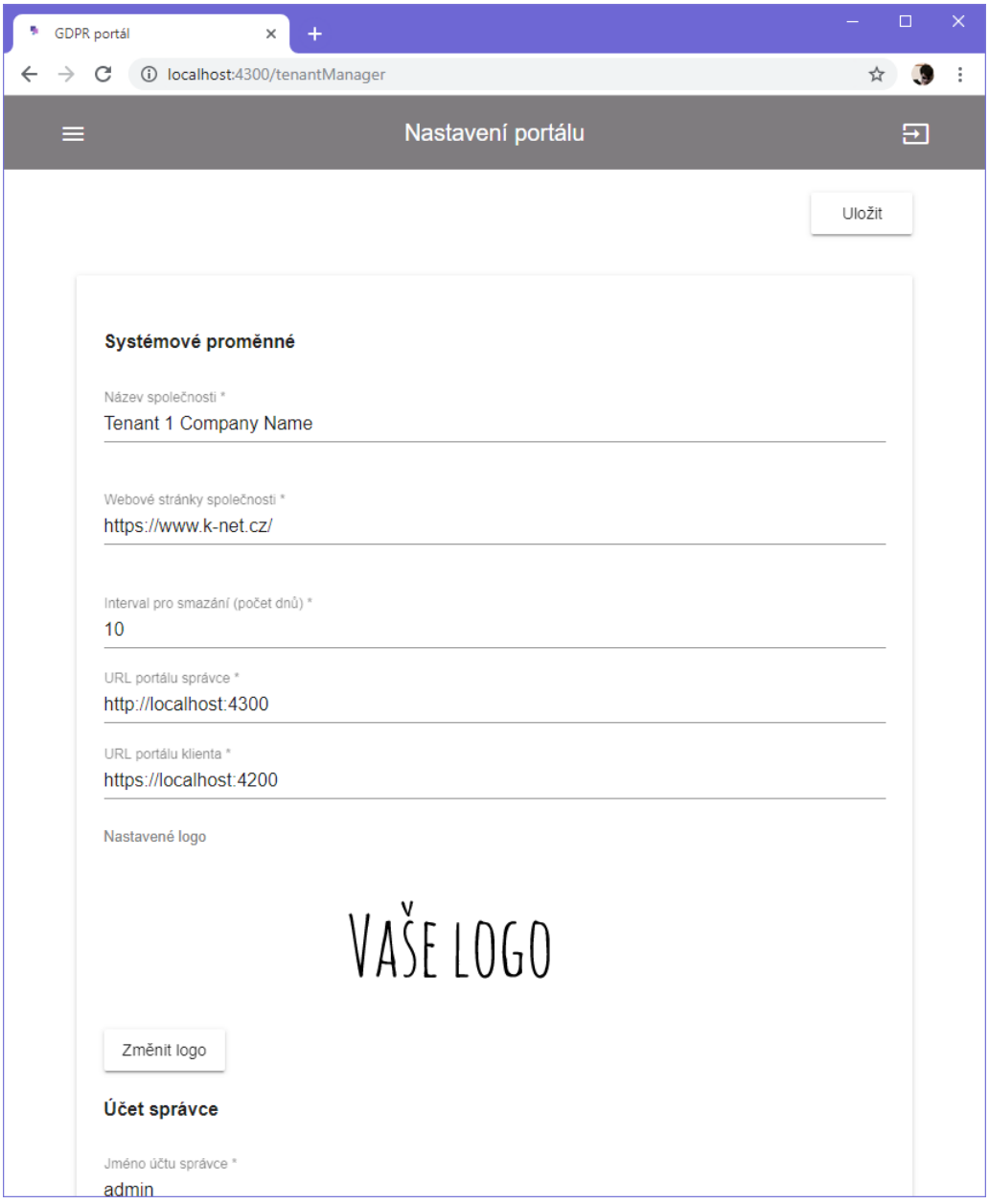
Obr. 39: GUI - Aplikace organizace, žurnál - seznam událostí



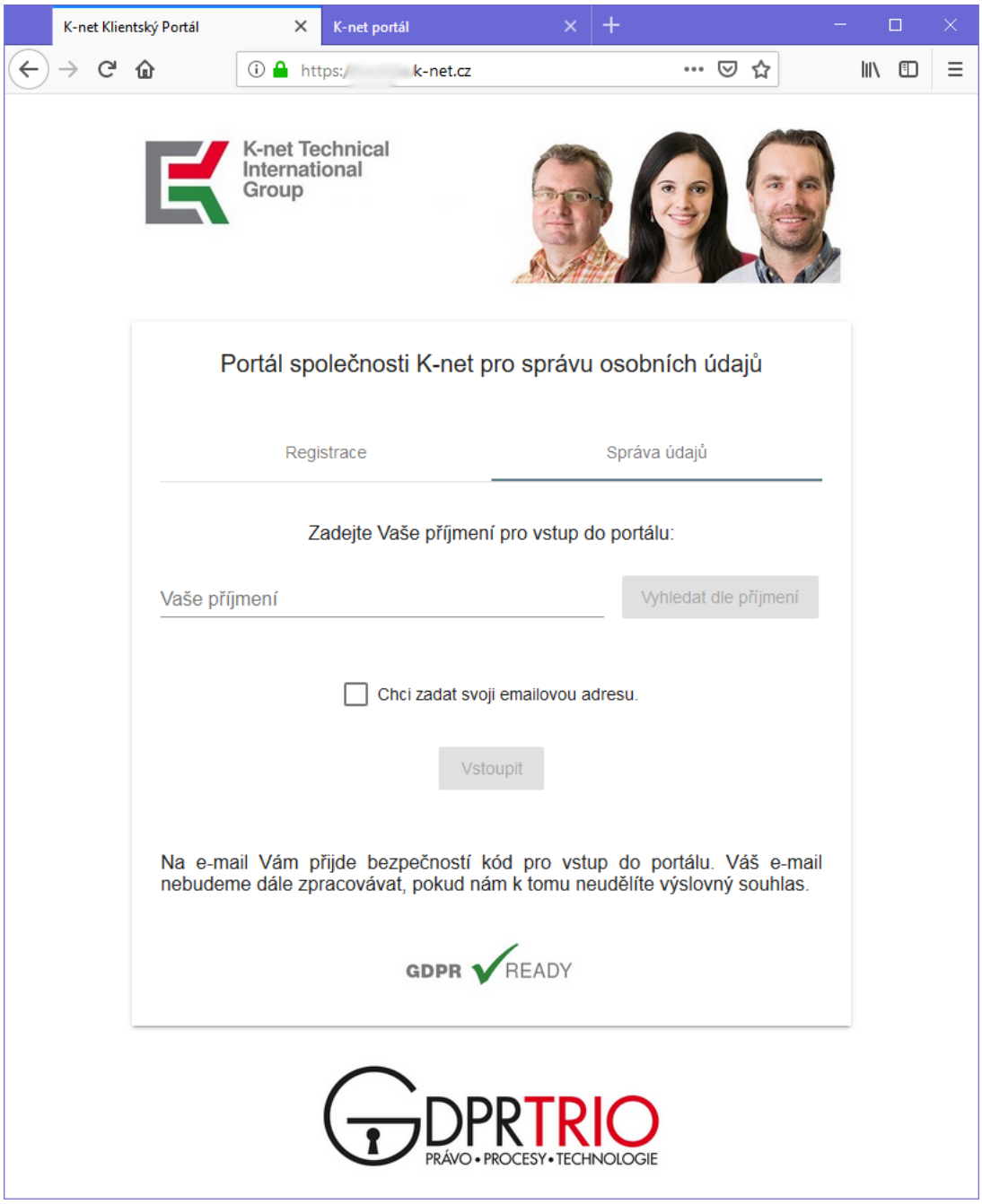
Obr. 40: GUI - Aplikace organizace, žurnál - seznam událostí



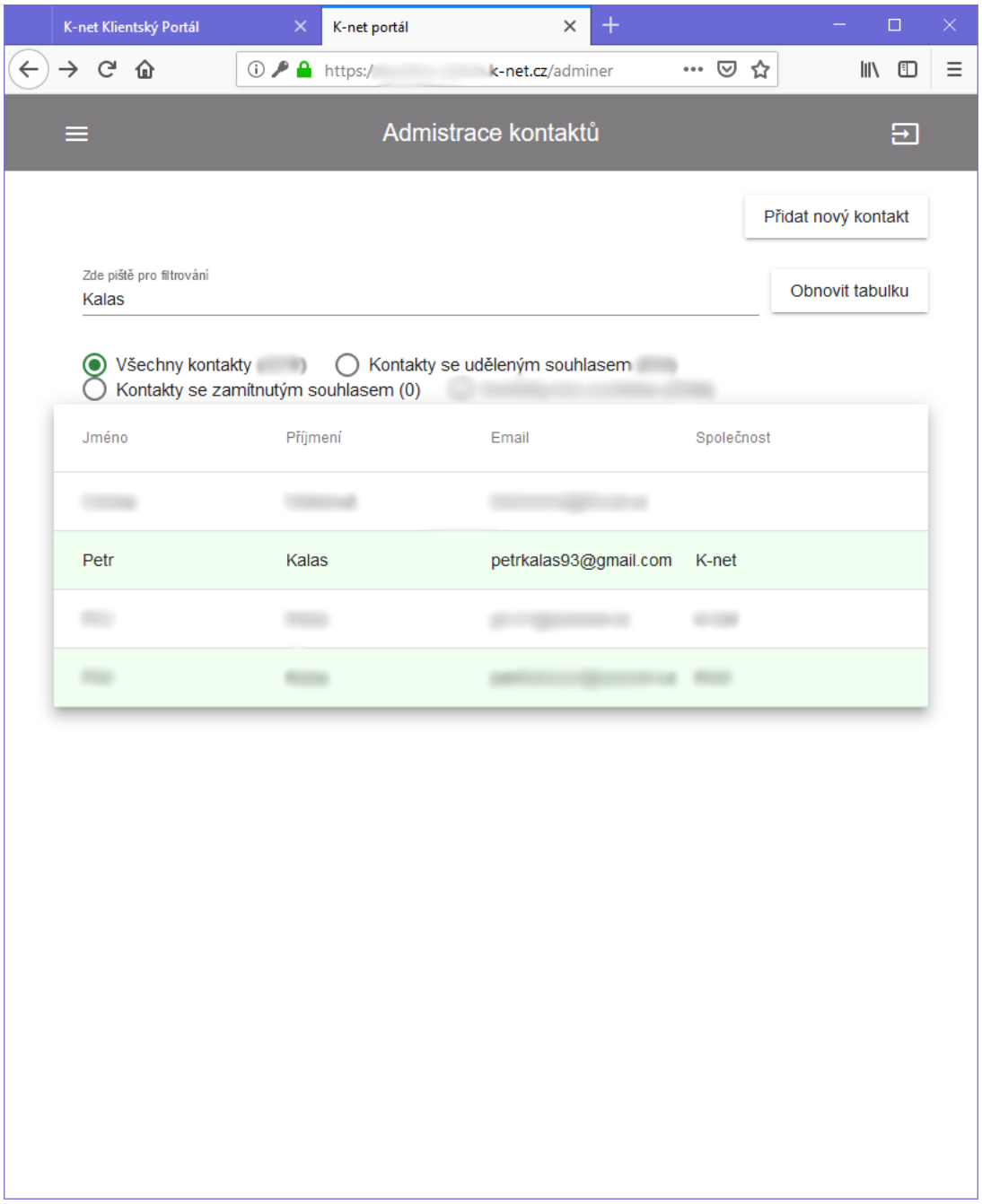
Obr. 41: GUI - Aplikace organizace, žurnál - anonymizovaný detail události



Obr. 42: GUI - Aplikace organizace, nastavení portálu

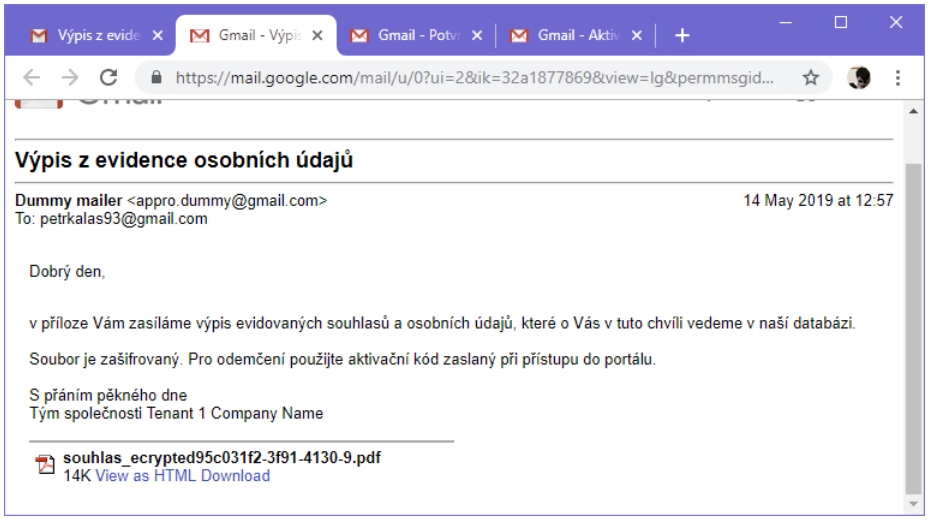


Obr. 43: Testování - nasazení ve společnosti, Aplikace klienta

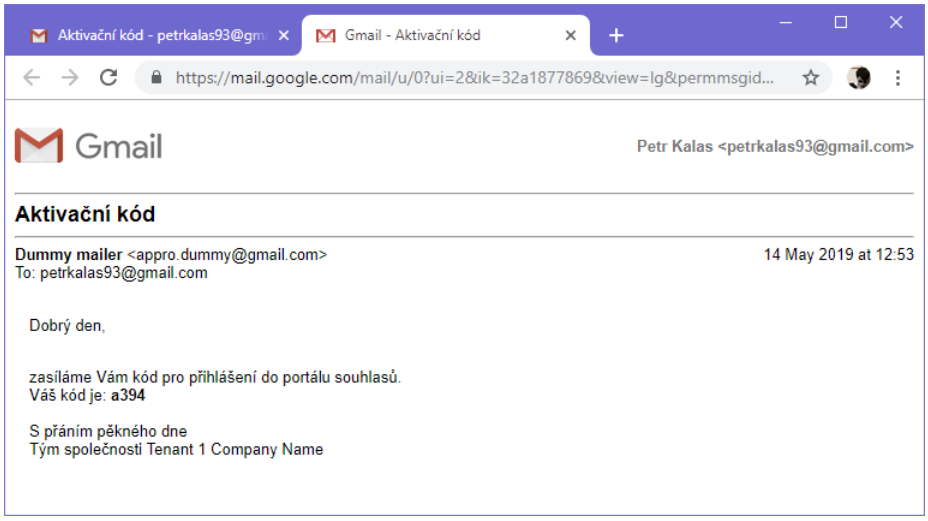


Obr. 44: Testování - nasazení ve společnosti, Aplikace organizace

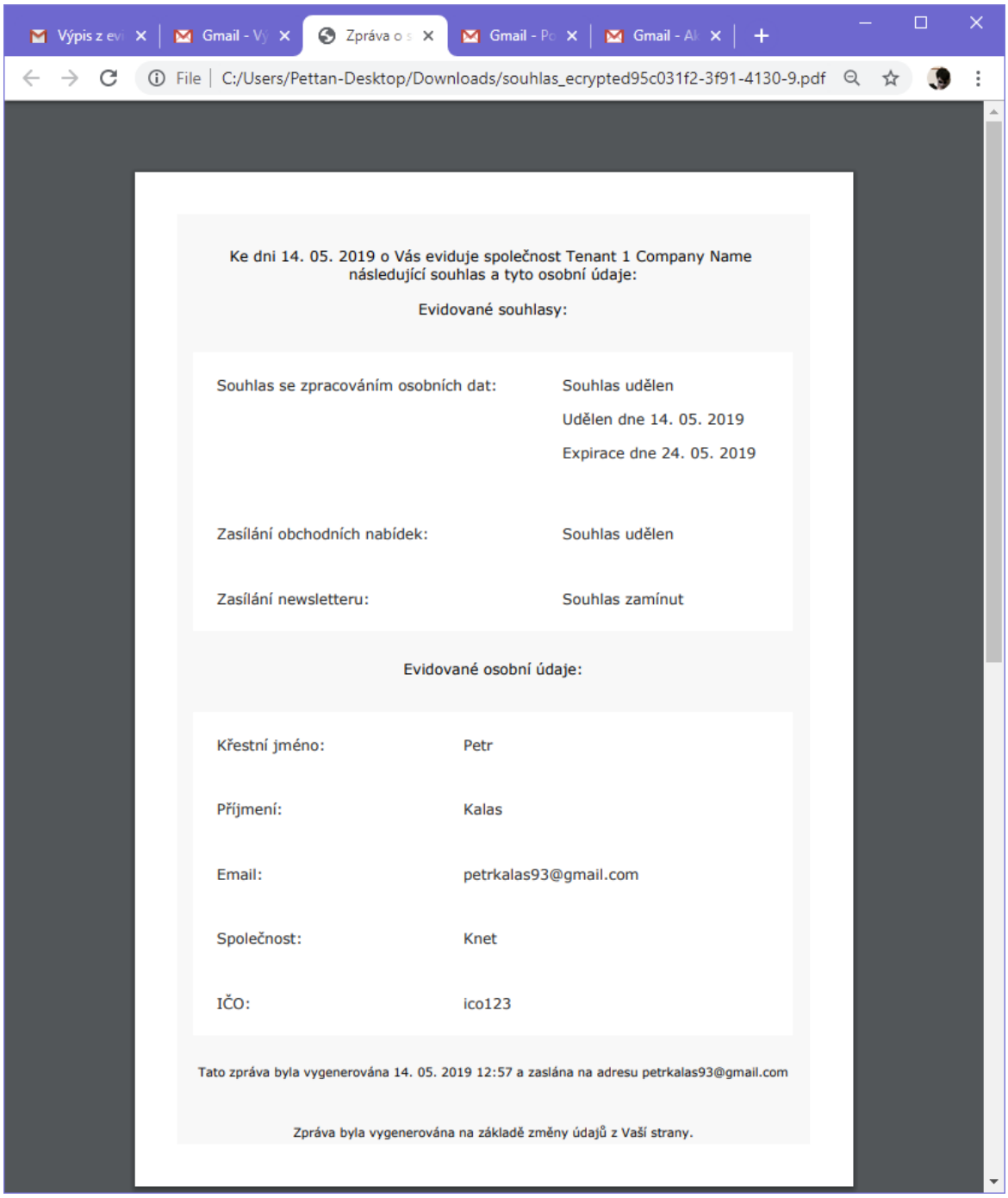
C Příloha - Emailové notifikace



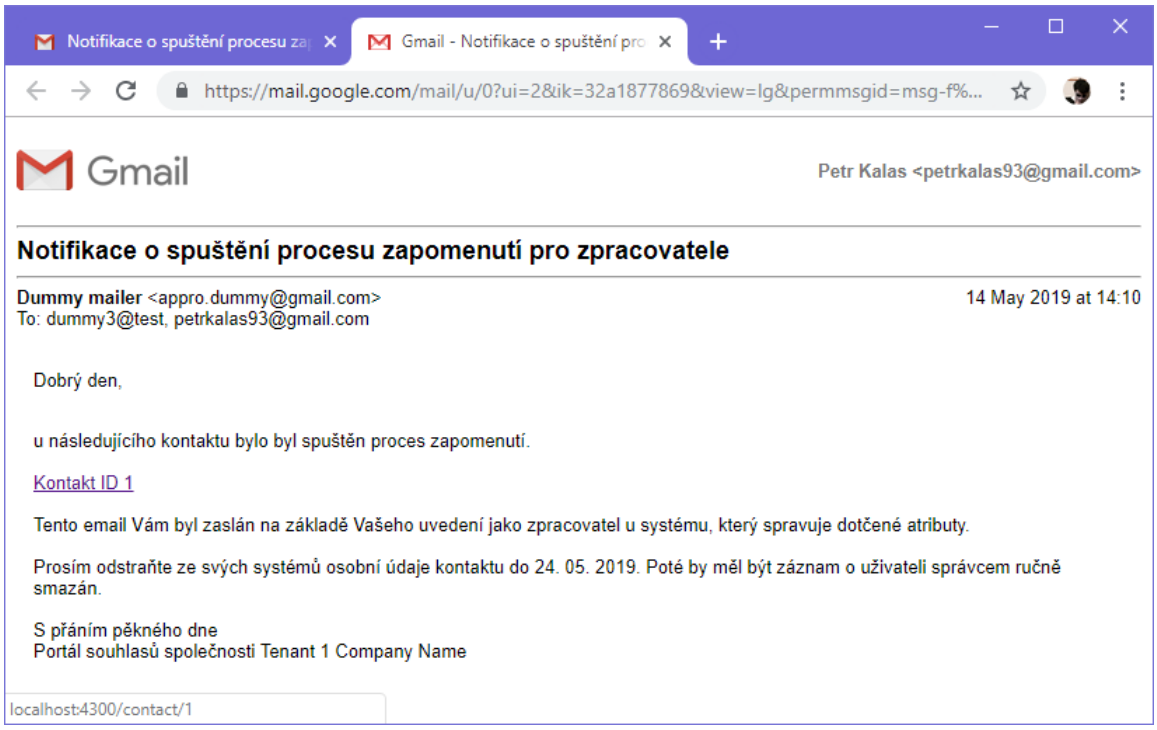
Obr. 45: Emailové notifikace - Výpis z evidence



Obr. 46: Emailové notifikace - Aktivační kód



Obr. 47: Emailové notifikace - Šifrovaná PDF příloha



Obr. 48: Emailové notifikace - Notifikace zpracovatele

D Příloha - Vyhodnocení testů

Obr. 49: Testování - výsledky automatizovaných testů

E Příloha - Infrastruktura: docker-compose.yml a set_env.sh

Infrastruktura. Obsah souboru docker-compose.yml a obsah souboru set_env.sh.

Zdrojový kód 1: Soubor docker-compose.yml

```
1
2  version: '3.2'
3
4  services:
5
6  ### Router container
7    traefik:
8      restart: always
9      build:
10       context: traefik/
11     container_name: appr_traefik
12     networks:
13       - appr_build
14       - appr_tenant
15     ports:
16       - "80:80" # remove to work only using https
17       - "443:443"
18     volumes:
19       - /var/run/docker.sock:/var/run/docker.sock:ro
20       - /srv/appr-volumes/treafik/:/etc/traefik/
21       #- /srv/appr-volumes/treafik/acme:/etc/traefik/acme #Lets encrpyt SSL autoconfig disabled
22     labels:
23       - "traefik.enable=true"
24       - "traefik.backend=traefik"
25       - "traefik.docker.network=infra_appr_build"
26       - "traefik.frontend.rule=Host:traefik.${APPR_DOMAIN}"
27       - "traefik.frontend.whiteList.sourceRange=${APPR_MANAGE_SUBNET}"
28       #- "traefik.frontend.auth.basic=${APPR_MANAGE_BASIC_AUTH}"
29       - "traefik.port=8080"
30
31 ### Control container
32   portainer:
33     restart: always
34     build:
35       context: portainer/
36     container_name: appr_portainer
37     networks:
38       - appr_build
39       - appr_tenant
40     volumes:
41       - /var/run/docker.sock:/var/run/docker.sock:ro
42       - /srv/appr-volumes/portainer/data:/data
43     labels:
44       - "traefik.enable=true"
45       - "traefik.backend=portainer"
46       - "traefik.docker.network=infra_appr_build"
47       - "traefik.frontend.rule=Host:portainer.${APPR_DOMAIN}"
48       - "traefik.frontend.whiteList.sourceRange=${APPR_MANAGE_SUBNET}"
49       #- "traefik.frontend.auth.basic=${APPR_MANAGE_BASIC_AUTH}"
50       - "traefik.port=9000"
51
52 ### DB container
53   mysql:
54     restart: always
```

```
55      build:
56        context: mysql/
57      container_name: appr_mysql
58      networks:
59        - appr_build
60        - appr_tenant
61      ports:
62        - '3306:3306'
63      environment:
64        MYSQL_USER: "${APPR_DB_USER}"
65        MYSQL_ROOT_PASSWORD: "${APPR_DB_PASS}"
66      volumes:
67         - /srv/appr-volumes/mysql:/var/lib/mysql
68         - ./mysql/init/:/docker-entrypoint-initdb.d
69
70
71  ### DB backup container
72    backup:
73      restart: always
74      build:
75        context: mysql-backup/
76      container_name: appr_mysql_backup
77      networks:
78        - appr_build
79      user: "0"
80      environment:
81        DB_DUMP_FREQ: '1440' #every day
82        DB_DUMP_BEGIN: '0415' #4:15 night
83        DB_DUMP_TARGET: '/db'
84        DB_SERVER: 'appr_mysql'
85        DB_USER: "${APPR_DB_USER}"
86        DB_PASS: "${APPR_DB_PASS}"
87      volumes:
88         - /srv/appr-volumes/db-backup:/db
89
90  ### Version control container
91    gitea:
92      restart: always
93      build:
94        context: gitea/
95      container_name: appr_gitea
96      networks:
97        - appr_build
98      environment:
99        APP_NAME: 'Petan Apprv Gitea'
100       RUN_MODE: 'prod'
101       DISABLE_SSH: 'true'
102       DB_TYPE: 'mysql'
103       DB_HOST: 'mysql:3306'
104       DB_NAME: 'gitea'
105       DB_USER: "${APPR_DB_USER}"
106       DB_PASSWD: "${APPR_DB_PASS}"
107       ROOT_URL: 'http://gitea.${APPR_DOMAIN}'
108       SSH_DOMAIN: 'http://gitea.${APPR_DOMAIN}'
109       HTTP_PORT: '3000'
110       DISABLE_REGISTRATION: 'false'
111     volumes:
112       - /srv/appr-volumes/gitea:/data
113     labels:
114       - "traefik.enable=true"
115       - "traefik.backend=gitea"
116       - "traefik.docker.network=infra_appr_build"
```

```
117       - "traefik.frontend.rule=Host:gitea.${APPR_DOMAIN}"
118       - "traefik.frontend.whiteList.sourceRange=${APPR_MANAGE_SUBNET}"
119       #- "traefik.frontend.auth.basic=${APPR_MANAGE_BASIC_AUTH}"
120       - "traefik.port=3000"
121     depends_on:
122      - mysql
123      - traefik
124
125
126 ### Build container
127   jenkins:
128     restart: always
129     build:
130       context: jenkins/
131     container_name: appr_jenkins
132     networks:
133       - appr_build
134     environment:
135       APPR_DOMAIN: "${APPR_DOMAIN}"
136     privileged: true
137     user: root
138     volumes:
139       - /srv/appr-volumes/jenkins:/var/jenkins_home
140       - /var/run/docker.sock:/var/run/docker.sock
141     labels:
142       - "traefik.enable=true"
143       - "traefik.backend=jenkins"
144       - "traefik.docker.network=infra_appr_build"
145       - "traefik.frontend.rule=Host:jenkins.${APPR_DOMAIN}"
146       - "traefik.frontend.whiteList.sourceRange=${APPR_MANAGE_SUBNET}"
147       #- "traefik.frontend.auth.basic=${APPR_MANAGE_BASIC_AUTH}"
148       - "traefik.port=8080"
149
150
151 networks:
152   appr_build: # used in jenkins pipeline
153   appr_tenant: # used in jenkins pipeline
154
```

Zdrojový kód 2: Soubor set_env.sh

```
1  ### Defaults varibles/passwords file.
2  ### Change your credentials on production env
3
4  # Domain
5  DOMAIN=fedoravm93.io
6
7  # Managing accesible only from
8  MANAGE_SUBNET=10.0.0.37/32
9
10 # Managing basic auth
11 # MD5 http://www.htaccesstools.com/htpasswd-generator/
12 # default admin:pass
13 MANAGE_BASIC_AUTH=admin:$$apr1$$EGeT7YGG$9Spy.YjjFzizYHT2nT/BZ.
14
15 # Default database root user
16 # Change it manually
17 DB_ROOT_USER=root
18 DB_ROOT_PASSWORD=password
19
20
21 ## Set enviromental variables
```

```
22 echo -n > /etc/environment
23
24 echo "APPR_DOMAIN=${DOMAIN}" >> /etc/environment
25 echo "APPR_MANAGE_SUBNET=${MANAGE_SUBNET}" >> /etc/environment
26 echo "APPR_MANAGE_BASIC_AUTH=${MANAGE_BASIC_AUTH}" >> /etc/environment
27
28 echo "APPR_DB_USER=${DB_ROOT_USER}" >> /etc/environment
29 echo "APPR_DB_PASS=${DB_ROOT_PASSWORD}" >> /etc/environment
30
31 # Idk
32 source /etc/environment
33
```

F Příloha - Infrastruktura: soubory Dockerfile

Soubory Dockerfile pro všechny typy kontejnerů dle návrhu v sekci 7.3. Jejich implementace je popsaná v kapitole 8.

Zdrojový kód 3: Dockerfile - Backend aplikace

```
FROM openjdk:11-jre-slim

ADD target/approvalportal.jar approvalportal.jar

ADD src/main/resources/application-prod-container.properties application.properties

ADD files files

RUN apt-get update -y && apt-get install curl -y

ENTRYPOINT ["java","-jar","/approvalportal.jar"]

EXPOSE 8080
```

Zdrojový kód 4: Dockerfile - Frontend aplikace

```
FROM nginx:latest

RUN apt-get update -y

RUN apt-get install curl -y

ADD replace.sh replace.sh

COPY default.conf /etc/nginx/conf.d/

ADD dist/* /usr/share/nginx/html/
```

Zdrojový kód 5: Dockerfile - Router kontejner (Traefik) label

```
FROM traefik:alpine
```

Zdrojový kód 6: Dockerfile - DB kontejner (Mysql) label

```
FROM mysql:8
```

Zdrojový kód 7: Dockerfile - Backup kontejner (databacker/mysql-backup) label

```
FROM alpine:3.9
MAINTAINER Avi Deitcher <https://github.com/deitch>

# install the necessary client
RUN apk add --update mysql-client mariadb-connector-c bash python3 samba-client shadow && \
    rm -rf /var/cache/apk/* && \
    touch /etc/samba/smb.conf && \
    pip3 install awscli

# set us up to run as non-root user
RUN groupadd -g 1005 appuser && \
    useradd -r -u 1005 -g appuser appuser
# ensure smb stuff works correctly
RUN mkdir -p /var/cache/samba && chmod 0755 /var/cache/samba && chown appuser /var/cache/samba
USER appuser
```

```
16
17 # install the entrypoint
18 COPY functions.sh /
19 COPY entrypoint /entrypoint
20
21 # start
22 ENTRYPOINT ["/entrypoint"]
```

Zdrojový kód 8: Dockerfile - Version control kontejner (Gitea)

```
1 FROM gitea/gitea:1.7.6
```

Zdrojový kód 9: Dockerfile - Control kontejner (Portainer)

```
1 FROM portainer/portainer
```

Zdrojový kód 10: Dockerfile - Build kontejner (Jenkins)

```
1  FROM jenkins/jenkins:latest
2
3  USER root
4  RUN apt-get update -y && \
5      apt-get install \
6      apt-transport-https \
7      ca-certificates \
8      curl \
9      gnupg2 \
10     software-properties-common -y && \
11     curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add - && \
12     apt-key fingerprint 0EBFCD88 && \
13     add-apt-repository \
14     "deb [arch=amd64] https://download.docker.com/linux/debian \
15     $(lsb_release -cs) \
16     stable" && \
17     apt-get update -y && \
18     apt-get install docker-ce docker-ce-cli containerd.io -y
```

Zdrojový kód 11: Dockerfile - Backend builder kontejner

```
1  FROM maven:3.6.1-jdk-11-slim
2
3  # install docker
4  USER root
5  RUN apt-get update -y && \
6      apt-get install \
7      apt-transport-https \
8      ca-certificates \
9      curl \
10     gnupg2 \
11     software-properties-common -y && \
12     curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add - && \
13     apt-key fingerprint 0EBFCD88 && \
14     add-apt-repository \
15     "deb [arch=amd64] https://download.docker.com/linux/debian \
16     $(lsb_release -cs) \
17     stable" && \
18     apt-get update -y && \
19     apt-get install docker-ce docker-ce-cli containerd.io -y
```

Zdrojový kód 12: Dockerfile - Frontend builder kontejner

```
FROM node:8

# install docker
USER root
RUN apt-get update -y && \
    apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg2 \
    software-properties-common -y && \
    curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add - && \
    apt-key fingerprint 0EBFCD88 && \
    add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/debian \
    $(lsb_release -cs) \
    stable" && \
    apt-get update -y && \
    apt-get install docker-ce docker-ce-cli containerd.io -y
```

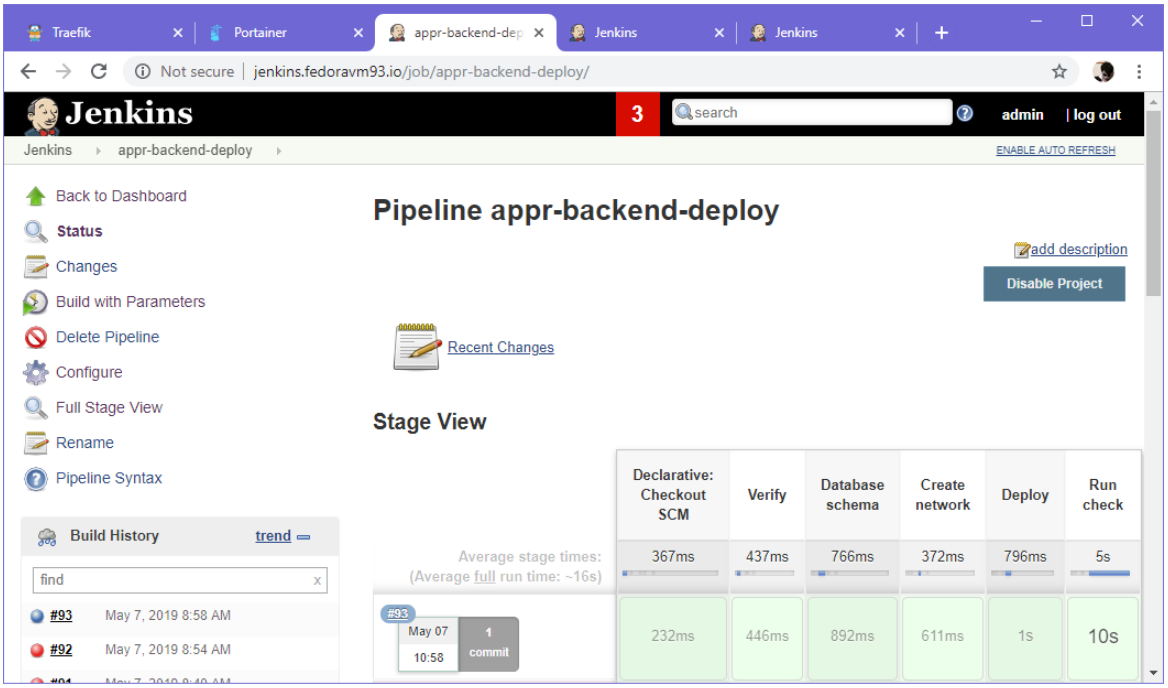
G Příloha - Infrastruktura: Automatizace

Zdrojový kód 13: Soubor Jenkinsfile, backend aplikace - vytvoření image

```
1
2  pipeline {
3
4      agent {
5          docker {
6                  image 'backend-build'
7                  args '-v /root/.m2:/root/.m2 -v /var/run/docker.sock:/var/run/docker.sock'
8          }
9      }
10
11     stages {
12
13         stage('Build') {
14             steps {
15                 sh 'mvn clean package -Dmaven.test.skip=true'
16             }
17         }
18
19         stage('Test') {
20             steps {
21                 sh 'mvn test'
22             }
23             post {
24                 always {
25                     junit 'target/surefire-reports/*.xml'
26                 }
27             }
28         }
29
30         stage('Image') {
31             steps {
32                 script {
33         VERSION = sh (
34         script: 'cat pom.xml | grep "<version>" | head -1 | cut -d ">" -f 2 | cut -d "<" -f 1',
35         returnStdout: true
36                         ).trim()
37                 }
38                                 echo VERSION
39                 echo 'Deploying....'
40                 sh "docker build -t appr-backend:${VERSION} -t appr-backend:latest ."
41             }
42         }
43     }
44 }
```

Zdrojový kód 14: Spuštění Jenkins jobu, příkazová řádka, nasazení backend aplikace

```
1
2  curl -X POST -u admin:password
3          "http://jenkins.fedoravm93.io/job/appr_backend_deploy/
4                  buildWithParameters?
5                  TENANT_ID=Company1
6                  &TENANT_PASS=password1*
7                  &BACKEND_DOMAIN=company-be"
```



Obr. 50: Infrastruktura - proces nasazení tenanta

Zdrojový kód 15: Soubor Jenkinsfile, backend aplikace - nasazení aplikace

```
pipeline {

    agent any

    stages {

        stage('Verify') {
            steps {
                script {
                    /* Test tenant id regex */
                    if ("$TENANT_ID" ==~ /^[a-zA-Z0-9_]*$/) {
                        echo 'TENANT_ID regex OK'
                    } else {
                        error "TENANT_ID don't matches regular expression, terminating."
                    }

                    /* Test tenant container name */
                    EXISTS = sh(
                            script: '''
                            if docker ps -a | grep t_${TENANT_ID}_backend >> /dev/null;
                            then echo "true"; else echo "false"; fi
                            ''',
                            returnStdout: true
                    ).trim()
                    if ("$EXISTS" == "true") {
                        error 'Container for given tenant id is already running!'
                    }
                }
            }
        }

```

```
32
33          stage('Database schema') {
34              steps {
35                  echo "Creating database schema... "
36                  sh '''
37                  docker exec appr_mysql mysql -u ${DB_ROOT_USER} -p${DB_ROOT_PASS}
38                  -e "create database IF NOT EXISTS ${TENANT_ID}_db;"
39                  '''
40                  sh '''
41                  docker exec appr_mysql mysql -u ${DB_ROOT_USER} -p${DB_ROOT_PASS}
42                  -e "create user IF NOT EXISTS '${TENANT_ID}_user'@'%' identified by '${TENANT_PASS}';"
43                   '''
44                  sh '''
45                  docker exec appr_mysql mysql -u ${DB_ROOT_USER} -p${DB_ROOT_PASS}
46                  -e "grant all on ${TENANT_ID}_db.* to ${TENANT_ID}_user@'%';"
47                  '''
48              }
49          }
50
51          stage('Create network') {
52              steps {
53                  echo "Starting backend container"
54                  sh '''
55                      docker network create "t_${TENANT_ID}_net"
56                      '''
57              }
58          }
59
60          stage('Deploy') {
61              steps {
62                  echo "Starting backend container"
63                  sh '''
64                      docker run \
65                      -d \
66                      --name "t_${TENANT_ID}_backend" \
67                      --network=infra_appr_tenant \
68                      --restart always \
69                          -e "COMPANY_NAME=${TENANT_ID}" \
70                          -e "APPR_DOMAIN=${APPR_DOMAIN}" \
71                          -e "JDBC_URL=jdbc:mysql://appr_mysql:3306/${TENANT_ID}_db" \
72                          -e "JDBC_USER=${TENANT_ID}_user" \
73                          -e "JDBC_PASS=${TENANT_PASS}" \
74                          -e "JDBC_DIALECT=org.hibernate.dialect.MySQLDialect" \
75                          -l "traefik.enable=true" \
76                      -l "traefik.backend=${TENANT_ID}_backend" \
77                      -l "traefik.docker.network=infra_appr_tenant" \
78                      -l "traefik.frontend.rule=Host:${BACKEND_DOMAIN}.${APPR_DOMAIN}" \
79                      -l "traefik.port=8080" \
80                      appr-backend:${APPR_BACKEND_VERSION}
81                      '''
82                  sh '''
83                      docker network connect \
84                      --alias="t-${TENANT_ID}-backend" \
85                      "t_${TENANT_ID}_net" "t_${TENANT_ID}_backend"
86                      '''
87              }
88          }
89
90          stage('Run check') {
91              steps {
92                  echo "Check if container is up"
93                  echo "Waiting until container is up..10s"
```

```
94                  sleep 10
95                  script {
96                      RUNNING = sh(
97                              script: '''docker inspect -f '{{.State.Running}}'
98                               t_${TENANT_ID}_backend''',
99                              returnStdout: true
100                     ).trim()
101                     if ("$RUNNING" == "true") {
102                         echo 'Backend container is running.'
103                     } else {
104                         error 'Backend container is not running!'
105                     }
106                 }
107
108             }
109         }
110
111     }
112 }
```