České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Heger Branislav

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: Systém pro správu životního cyklu zaměstnance

Pokyny pro vypracování:

Systém umožní vytvářet elektronické žádosti, jejichž prostřednictvím se pro zaměstnance žádá a schvaluje výpočetní technika a přístupová oprávnění do systémů. Systém bude notifikovat příslušné uživatele pomocí emailu a service desku. Analyzujte všechny požadavky na systém. Navrhnete a implementujte systém v technologii JAVA EE. Prozkoumejte možnosti, vyberte a implementujte efektivní řešení, které systému umožní komunikaci s aplikacemi třetích stran různých platforem. Vytvořený systém otestujte na reálných datech.

Seznam odborné literatury:

[1] Ian SUMMERVILLE, Software Engineering: 9th Edition. USA:Addison-Wesley Publishing Company, 2010. ISBN 9780137035151/0137035152.
[2] Jerome LOUVEL, Thierry TEMPLIER, and Thierry BOILEAU, Developing RESTful web APIs in Java. NY:Manning Publications, September 2012. 464 s. ISBN 9781935182344.
[3] Willie WHEELER with Joshua WHITE, Spring in Practice. NY:Manning Publications,May 2013. 560 s. ISBN 9781935182054.
[4] Rod COLLEDGE, SQL Server 2008 Administration in Action. NY:Manning Publications,August 2009. 464 s. ISBN 9781933988726.

Vedoucí: Ing. Jiří Husák

Platnost zadání do konce letního semestru 2017/2018

prof. Dr. Michal Pěchouček, MSc.
vedoucí katedry

L.S.

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 6.2.2017

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů

Diplomová práca

Systém pre správu životného cyklu zamestnanca

Bc. Branislav Heger

Vedúci práce: Ing. Jiří Husák

23. mája 2017

Poďakovanie

Ďakujem vedúcemu tejto diplomovej práce Ing. Jiřímu Husákovi za odborné vedenie mojej práce.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 23. mája 2017 .....

Odkaz na túto prácu

Heger, Branislav. *Systém pre správu životného cyklu zamestnanca.* Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta elektrotechnická, 2017.

Abstrakt

Cieľom tejto práce je navrhnúť a implementovať systém, ktorý umožní vytvárať elektronické žiadosti, pomocou ktorých sa pre zamestnancov žiada a schvaľuje výpočetná technika a prístupy do informačných systémov. Práca popisuje celý vývojový cyklus od analýzy požiadavkov, návrhu, implementácie a testovania systému.

Klíčová slova životný cyklus zamestnanca, online formulár, oprávnenia, prístupy do systému

Abstract

This thesis focuses on the design and implementation of a system, for creating online application forms. These forms will be used for requesting and granting of equipment and access to various information systems for employees. The thesis describes whole development cycle starting at analysing requirements, system design, implementation and running tests on the system.

Keywords employee's life cycle, online application form, permissions, system access

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Životný cyklus zamestnanca

Ak má používateľ v spoločnosti prístup k viac informačným systémom, má spravidla aj viac prístupových účtov. Počet aplikácií a systémov stále narastá a schopnosť riadiť veľké množstvo účtov v dostatočnom čase môže byť pre organizáciu problém. Aby organizácia mala jasný prehľad o tom, kto a kedy má k akej informácii prístup, je potrebné používateľské účty spravovať z jedného miesta. Odpovede na tieto potreby ponúka software pre centralizovanú správu užívateľov, zvaný identity management. Základným procesom identity manažmentu je životný cyklus identity užívateľa. Cyklus spravidla začína nástupom zamestnanca do organizácie, kedy sú užívateľovi zriadené základné prístupy, napríklad do firemnej pošty a intranetu. V priebehu času užívateľ získava ďalšie nové účty a oprávnenia, alebo o oprávnení naopak prichádza. Organizácie tiež pomerne často zažívajú zmenu organizačného zaradenia pracovníka a s ňou súvisiace zmeny v oprávneniach.

Motivácia

Potreba systému pre životný cyklus identity zamestnanca vyplýva z odporučenia auditora. Akonáhle má používateľ prístup k ľubovolnému systému, je potrebné vedieť presne určiť, z akého dôvodu o tento prístup žiadal, resp. niekto žiadal pre neho. Ďalej je potrebné vedieť, kto tento prístup schválil, v prípade zamietnutého prístupu je potrebné vedieť vysvetlenie. Dôvodom pre vlastnú implementáciu takéhoto systému je skutočnosť, že užívatelia sú zvyktnutí na súčastný systém, ktorý však nepodporuje všetku požadovanú funkcionalitu. Rozšírenie pôvodného systému nepripadalo v úvahu z dôvodu nekompatibility technológií s technológiami, ktoré pre interné aplikácie firma používa momentálne. Vedenie sa teda rozhodlo o implementáciu nového systému. Aby nasadenie nového systému nespôsobilo chaos a nespomalilo zaužívané procesy, bude

user experience v novom systéme inšpirovaný pôvodným systémom.

Štruktúra práce

Práca je rozdelená do 5 kapitol: Analýza, Návrh aplikácie, Implementácia, Testovanie, Nasadenie systému a budúci rozvoj. Toto delenie odpovedá modelu softwarového procesu waterfall [1]. Kapitola Analýza sa zaoberá súčastným riešením a problémami, ktoré s nim súvisia. Následne zberom a spracovaním požiadavkov na nové riešenie. Kapitola Návrh aplikácie popisuje zvolené technológie, pomocou diagramov definuje architektúru systému, dátový model a komunikáciu jednotlivých vrstiev systému. V tejto kapitole je ďalej zadefinovaný návrh užívateľského rozhrania prostredníctvom wireframe. Kapitola Implementácia je rozdelená podľa jednotlivých vrstiev systému a pri každej popisuje riešenie určitého problému vyplývajúceho z požiadaviek na systém. Kapitola Testovanie popisuje a vyhodnocuje testy, ktoré nad aplikáciou prebehli. Posledná kapitola Nasadenie systému a budúci rozvoj definuje požiadavky a prostredie, v ktorom je systém nasadený a nakoniec popisuje budúci rozvoj systému.

Kapitola 1

Analýza

1.1 Súčasné riešenie, popis problému a špecifikácia cieľa

Riešenie životného cyklu zamestnanca je súčastne riešené systémom, ktorý podporuje iba časť procesu potrebného preto, aby bolo žiadanie o prístupy do informačných systémov v súlade s odporučením auditora. Momentálne je proces nastavený tak, že o všetkých oprávneniach, ktoré užívateľ má alebo nemá dostať, rozhoduje jediný človek. Je ním nadriadený daného užívateľa. Problém je v tom, že o prístupoch do informačných systémov by správne nemal rozhodovať iba nadriadený zamestnanca. Majú o tom rozhodovať vlastníci užívateľských rolí do týchto konkrétnych systémov, do ktorých daný užívateľ žiada prístup. Cieľom nového systému nebude zaviesť odlišný proces schvaľovania, to by mohlo spôsobiť, že by sa systém u užívateľov neujal a komplikoval by im prácu. Cieľom bude nadviazať na predchádzajúci proces. Schválenie žiadosti nadriadeným bude po novom súhlasom s tým, že jeho podriadený požiada o vybrané prístupy. Následne príde fáza schvaľovania prístupov jednotlivými vlastníkmi užívateľských rolí.

1.2 Požiadavky na systém

1.2.1 Funkčné požiadavky

1.2.1.1 Interaktívne formuláre

Systém umožní podávať žiadosti prostredníctvom interaktívnych formulárov, slúžiacich k rýchlemu a efektívnemu vkladaniu dát a ich schvaľovaniu. Formuláre budú fungovať ako alternatíva papierových formulárov. Systém rozlišuje nasledujúce typy formulárov:

- Nástup zamestnanca

- Prestup zamestnanca
- Zmena v technických štruktúrach
- Výstup zamestnanca

Žiadosť reprezentovaná takýmto formulárom sa môže nachádzať v nasledujúcich stavoch:

- Nová
- Čaká na schválenie
- Schválená
- Zamietnutá

Formulár sa skladá z troch častí. Prvá časť je vyhradená vkladaniu informácií o zaradení zamestnanca. Táto časť sa v každom type formulára mierne líši, vo všeobecnosti ale vždy pôjde o vyplnenie nasledujúcich údajov:

- Výber zamestnanca zo zoznamu užívateľov, v prípade formuláru "Nástup zamestnanca" pôjde o ručné zadanie mena, priezviska a titulov pred aj za menom. Ide o užívateľa, ktorý je subjektom žiadosti, pre tohoto užívateľa je žiadosť určená, preto sa tento užívateľ v rámci systému nazýva subjekt.
- Výber dátumu, od kedy majú schválené údaje a oprávnenia nadobudnúť platnosť.
- Výber oddelenia, na ktorom daný zamestnanec pracuje, alebo pracovať bude. Jednotlivé oddelenia majú určitú hierarchiu. Na základe tejto hierarchie systém vytvorí strom oddelení tak, aby mal užívateľ možnosť jednoducho vybrať to správne oddelenie. Strom oddelení sa podľa hierarchie nazýva organizačná štruktúra a jedna položka takéhoto stromu je organizačná jednotka.
- Výber pozície, na ktorej bude zamestnanec pracovať. Pozície budú vyberané zo zoznamu pozícií príslušných k vopred vybratému oddeleniu.
- Výber pobočky, na ktorej sa zamestnanec nachádza. Vo formulári sa nazýva lokalita.
- Výber nadriadeného zo zoznamu užívateľov.

Druhá časť obsahuje zoznam všetkých vecí, o ktoré je v aplikácii možné žiadať. Pôjde o vybavenie, oprávnenia pre prístupy do informačných systémov a mnoho ďalšieho, pre zjednodušenie ďalej len oprávnenia. Každá položka bude obsahovať checkbox, pomocou ktorého užívateľ zo zoznamu vyberá. Po zaškrtnutí checkboxu sa pri položke môžu zobraziť textové polia vyžadujúce

rôzne údaje, typicky pôjde o business zdôvodnenie, ktoré schvaľovateľom pomôže pri rozhodovaní o schválení alebo zamietnutí daného oprávnenia. To, či sa takéto textové pole zobrazí, alebo nezobrazí a či je jeho vyplnenie povinné, bude konfigurovateľné na úrovní danej entity v databáze. Tým pádom bude editovateľné prostredníctvom administrácie priamo v aplikácii.

Tretia a posledná časť obsahuje 5 možností reprezentovaných pomocou tlačítok, ktoré sa zobrazujú v závislosti na aktuálnom stave žiadosti. Určujú, čo sa s vyplneným formulárom stane ďalej, teda predurčujú budúci stav žiadosti. Ide o nasledujúce možnosti:

- Schváliť - Možnosť sa zobrazuje ak je žiadosť v stave „Nová“ a „Čaká na schválenie“, táto možnosť prepne žiadosť do stavu „Schválená“. To však systém umožní iba užívateľovi, ktorý je v žiadosti vybraný ako nadriadený, respektíve pracovníkom personálneho oddelenia, ktorí sú ku schvaľovaniu žiadosti za nadriadeného oprávnení.

- Zamietnuť - Možnosť sa zobrazuje iba vtedy, keď je žiadosť v stave „Čaká na schválenie“, táto možnosť prepne žiadosť do stavu „Zamietnutá“. Rovnako ako v možnosti „Schváliť“, umožní systém zamietnuť žiadosť iba užívateľovi, ktorý je v žiadosti vybraný ako nadriadený.

- Odoslať na schválenie - Možnosť sa zobrazuje, ak je žiadosť v stave „Nová“, žiadosť sa následne prepne do stavu „Čaká na schválenie“. Túto možnosť môže vybrať akýkoľvek užívateľ.

- Uložiť - Možnosť sa zobrazuje, ak je žiadosť v stave „Nová“, možnosť „Uložiť“ nemení stav žiadosti. Žiadosť sa teda uloží do databázy a zostane v stave „Nová“

- Rozhodnúť oprávnenia - Táto možnosť sa zobrazí na žiadosti, ktorá je v stave „Schválená“ a len vtedy, keď si formulár rozklikne schvaľovateľ nejakého z oprávnení, ktoré je vo formulári vybrané. Pri vybraných oprávneniach, o ktorých môže rozhodovať, sa mu zobrazia možnosti schváliť alebo zamietnuť. Možnosť „Rozhodnúť oprávnenia“ uloží žiadosť do databázy s údajmi o tom, ktoré oprávnenia boli schválené, a ktoré boli zamietnuté.

1.2.1.2 Vytvorenie užívateľa s identifikačným číslom

Jedným z výstupov formuláru typu „Nástup zamestnanca“ bude vytvorenie užívateľa s validným a jedinečným identifikačným číslom. V prípade obnovenia spolupráce s bývalým zamestnancom bude formulár „Nástup zamestnanca“ umožňovať zadať pôvodné identifikačné číslo.

1.2.1.3 Prehľad žiadostí

Systém bude obsahovať stránku so zoznamom žiadostí. Do tohoto zoznamu sa dotiahnu iba tie dáta, do ktorých ma konkrétny užívateľ možnosť nahliadnuť. Ide o nasledujúce prípady:

- Užívateľ, ktorý podáva žiadosť pre samého seba.
- Užívateľ, ktorý podáva žiadosť pre iného užívateľa.
- Nadriadený užívateľa, pre ktorého bola žiadosť podaná. Tento užívateľ žiadosť zároveň schvaľuje. Znamená to, že schváli alebo zamietne zaradenie daného zamestnanca, vybavenie jeho pracovného miesta a tie prístupy do informačných systémov, ktoré nepotrebujú byť samostatne schválené ich schvaľovateľom.
- Schvaľovatelia oprávnení, o ktoré sa pre užívateľa v danej žiadosti žiada.
- Zástupcovia schvaľovateľov oprávnení.

Nemôže sa stať, že sa v tejto tabuľke užívateľovi zobrazí vlastný výstup. V riadku tabuľky budú uvedené základné údaje žiadosti. Konkrétna žiadosť v plnom rozsahu sa užívateľovi zobrazí až po kliknutí na odkaz uvedený v riadku tabuľky. V prípade, že sa k takémuto odkazu dostane osoba, ktorá k danej žiadosti nemá prístup, žiadosť sa jej nezobrazí. Rovnako sa žiadosť nezobrazí, ak by odkaz smeroval na žiadosť s výstupom daného užívateľa.

1.2.1.4 Administrácia systému

Systém umožní administráciu prostredníctvom na to určených stránok. Každá entita v databáze bude mať vlastnú stránku. Stránky patriace verziovaným entitám umožnia pridávanie nových záznamov a editáciu pôvodných. Prostredníctvom administrácie bude možné konfigurovať celú aplikáciu. Do administrácie systému bude mať prístup iba užívateľ v roli ROLE_ADMIN.

1.2.1.5 Notifikácie užívateľov

Pri zmene stavu žiadosti systém notifikuje príslušných užívateľov prostredníctvom emailu alebo požiadavku v Service desku[1]. Oba spôsoby notifikácie budú obsahovať všetky potrebné informácie, ktoré daný formulár obsahuje. V prípade emailu bude obsah formátovaný pomocou HTML a CSS. V prípade požiadavky do Service desku, pôjde iba o text. Prípady, v ktorých systém rozosiela notifikácie:

[1] Jednotný kontaktný bod medzi poskytovateľom služieb a používateľom služieb. Typický service desk riadi incidenty a žiadosti o službu a tiež rieši komunikáciu s používateľmi.

- Výber možnosti „Odoslať na schválenie“. Žiadosť sa prepne zo stavu „Nová“ do stavu „Čaká na schválenie“. V tomto prípade systém rozosiela emaily na subjekt a nadriadeného.

- Výber možnosti „Schváliť“. Žiadosť sa prepne zo stavu „Nová“ alebo „Čaká na schválenie“ do stavu „Schválená“. Schválenie žiadosti znamená, že nadriadený súhlasí s informáciami o zaradení zamestnanca, ktoré sa nachádzajú v prvej časti formulára. Ďalej súhlasí s tým, že vybavenie a oprávnenia vybraté v danom formulári dávajú zmyslel v súvislosti s pozíciou, na ktorej subjekt pracuje. Schválením žiadosti zároveň schvaľuje to vybavenie a oprávnenia, ktoré nevyžaduje samostatné schvaľovanie schvaľovateľom daného oprávnenia alebo vybavenia. Tým pádom musí systém prostredníctvom emailu notifikovať subjekt a kvôli kontrole aj nadriadeného. Ďalej systém rozošle email na schvaľovateľov neschváleného vybavenia a oprávnení vybratých vo formulári. Keďže niektoré oprávnenia v tejto fáze už môžu byť schválené, systém zároveň notifikuje osoby zodpovedné za prideľovanie týchto oprávnení. To, či má systém týchto užívateľov notifikovať pomocou emailu alebo Service desku, záleží od jednotlivého oprávnenia. V prípade, že žiadosť obsahuje oprávnenia, ktoré majú ako spôsob notifikácie nastavený Service desk, sa postupuje nasledovne. Keďže každé jedno oprávnenie môže mať na starosti niekto iný, tak sa v Service desku vytvorí samostatná úloha pre každé jedno takéto oprávnenie. Názov oprávnenia bude súčasťou názvu úlohy a informácie o kompletnej žiadosti budú v popise tejto úlohy.

- Výber možnosti „Zamietnuť“. Žiadosť sa prepne zo stavu „Nová“ alebo „Čaká na schválenie“ do stavu „Zamietnutá“.

- Výber možnosti „Rozhodnúť oprávnenia“. Žiadosť zostane v stave „Schválená“. Informácia o schválení alebo zamietnutí požadovaných oprávnení je prostredníctvom emailu odoslaná užívateľovi, ktorý vo formulári figuruje ako subjekt a jeho nadriadenému. Systém zároveň notifikuje osoby zodpovedné za prideľovanie týchto oprávnení. Rovnako ako v predchádzajúcom prípade je to, či sú títo ľudia notifikovaný pomocou emailu alebo Service desku, závislé od jednotlivého oprávnenia.

1.2.2 Nefunkčné požiadavky

1.2.2.1 Technológie

- Systém bude vytvorený ako webová aplikácia prístupná cez prehliadač

- Systém bude implementovaný v jazyku Java. Webová časť aplikácie bude implementovaná v technológii Java Server Faces

- Systém bude pracovať s databázou Microsoft SQL Server.

1.2.2.2 Rozhranie

- Aplikácia bude tvorená oddelenou prezentačnou a aplikačnou vrstvou. Viac prezentačných vrstiev používa len jednu aplikačnú vrstvu a žiadna časť aplikačnej logiky nie je súčasťou databázovej vrstvy.

- Aplikácia umožní integráciu s aplikáciami tretích strán. Táto funkcionalita bude riešená prostredníctvom vystavených REST rozhraní. Vyvolávať ich bude servlet mapovaný na špecifické url adresy. Tretím stranám aplikácia poskytne vždy len aktuálne dáta, teda bez histórie.

1.2.2.3 Prostredie

- Aplikácia musí byť schopná prevádzky na pracovných staniciach zadávateľa

- Aplikácia a ďalší súvisiaci software bude inštalovateľný a aktualizovateľný pracovníkmi zadávateľa

- Aplikácia bude podporovať prehliadače : Internet Explorer 8, Firefox 30, Safari 5,6 a 7 na Mac OS X 10.7 a 10.9 , Google Chrome 35 , Internet Explorer na Windows Phone 8 , Safari na Apple iPhone 4S , 5C , 5S, 6, 6 plus, 6S , 6S plus, 7, 7plus.

1.2.2.4 Autentifikácia užívateľov

Systém bude zabezpečený proti neoprávneným prístupom. Prístup bude umožnený iba interným užívateľom prostredníctvom integrovanej autorizácie systému Windows. Užívatelia z intranetu budú môcť pracovať so systémom pod rovnakým menom a heslom, s akým sa prihlasujú do podnikovej siete. Prístup k dátam bude povolený na základe užívateľských rolí. Aplikácia na stranu klienta prenáša len tie dáta, ku ktorým má užívateľ prístupové oprávnenia.

1.2.2.5 Výkon

Systém musí užívateľovi odpovedať maximálne do 2 sekúnd. Aj napriek tomu, že databáza obsahuje nemalé množstvo údajov, očakáva sa od systému rýchla odozva.

1.2.2.6 Aktualizácia dát

Systém bude pripojený k vlastnej databáze. Táto databáza bude v pravidelných intervaloch dopĺňaná o aktuálne dáta z Intranetu. Keďže pôjde o úplne iné databázy obsahujúce inak štruktúrované dáta, bude potrebné naprogramovať špecifický migračný skript v jazyku SQL.

1.2.2.7 História dát

Všetky zmeny v databáze majú plnú históriu na úrovni business entít. Pôjde o verziovanie dát na úrovni databázy. Systém ani jeho administratíva nesmú umožniť vymazanie záznamu z databázy. Alternatíva vymazania záznamu bude jeho zneplatnenie, neplatný záznam bude stále viditeľný v administrácii.

1.2.2.8 Užívateľské rozhranie

- Pre vývoj užívateľského rozhrania bude použitá technológia Java Server Faces, grafické prvky budú primárne z knižnice RichFaces.
- Všetky zoznamy v aplikácii sú triedené tak, aby nedošlo k náhodnému zobrazovaniu údajov, užívateľ môže nastaviť vlastné triedenia v hlavičke zoznamov.
- Stránky a obrazovky aplikácie sú v dynamickom rozložení a budú sa prispôsobovať veľkosti okna.
- Stránky a obrazovky aplikácie, vrátane formulárov je možné tlačiť tak, aby obsahovali len všetky požadované dáta, do tlače nebudú zaradené prvky navigácie. Pre stránku alebo obrazovku, ktorá obsahuje textové pole, ktorého obsah nie je viditeľný pri tlači, bude vytvorená tlačová verzia tak, aby bol zobrazený celý obsah textového poľa.

1.2.2.9 Správa aplikácie

- Aplikácia bude monitorovať aktivity užívateľov tak, aby ich bolo možné spätne vyhľadať.
- Dátový model uchováva plnú históriu zmien a aplikácia umožňuje náhľad do tejto histórie.
- Aplikácia bude ošetrená tak, aby nedochádzalo k vzájomnému prepisovanie dát medzi užívateľmi.
- Pokiaľ je pre jednu „session“ otvorených viac zobrazení, tak sa v „session“ uchovávajú iba dáta, ktoré nemôžu ovplyvniť správanie aplikácie.

1.2.2.10 Podpora

Všetka činnosť užívateľov bude evidovaná vo forme logov a významných udalostí v databáze. Vzniknuté chyby bude systém ukladať vo forme logov.

1.2.2.11 Dokumentácia

Súčasťou systému budú aj referencie zdrojového kódu v HTML generovaná prostredníctvom programu doxygen.

1.3 Užívateľské role

Systém bude rozlišovať nasledujúce užívateľské role:

1. ROLE_EMPLOYEE Jedná sa o rolu bežného zamestnanca, ktorý má v systéme minimálne oprávnenia. Medzi jeho oprávnenia patrí možnosť využívať formulár „Zmena v technických štruktúrach“a možnosť zobrazenia žiadosti, ktorá sa ho akýmkoľvek spôsobom týka s výnimkou vlastného výstupu.

2. ROLE_IT_EMPLOYEE Ide o rolu zamestnanca IT, ktorého hlavnou úlohou je prideľovať užívateľom oprávnenia alebo inštalovať potrebný software. Tieto úkony vytvára na základe požiadavku, ktorý mu systém vytvorí v Service desku. V rámci tohoto požiadavku je k dispozícii textové pole, ktoré zobrazuje stručný popis žiadosti. V prípade výpadku softwaru tretej strany Service desk je potrebné, aby mal užívateľ v roli ROLE_IT_EMPLOYEE možnosť zobrazenia akejkoľvek žiadosti. Ďalej má všetky oprávnenia role ROLE_EMPLOYEE.

3. ROLE_HR_EMPLOYEE Túto rolu budú mať pracovníci HR, ktorí budú pracovať so všetkými typmi formulárov. Navyše od role ROLE_IT_EMPLOYEE majú možnosť využívať formuláre „Nástup zamestnanca“,„Prestup zamestnanca“ a „Výstup zamestnanca“.

4. ROLE_ADMIN Jedná sa o rolu administrátora systému. Užívateľ s touto rolou bude mať ako jediný prístup k administrácii aplikácie. V rámci administrácie bude vytvorená stránka, ktorá užívateľovi v roli ROLE_ADMIN umožní prideľovať užívateľom systému jednotlivé užívateľské role.

1.4 Aktéri procesu spracovania žiadosti

Najdôležitejšou časťou business logiky systému bude proces spracovania žiadosti. Na základe funkčných požiadavkov je možné rozlíšiť jednotlivých aktérov podieľajúcich sa na tomto procese.

1.4.1 Žiadateľ

Žiadateľom je akýkoľvek užívateľ systému, ktorý prostredníctvom jedného z formulárov, či už pre seba, alebo pre iného zamestnanca, o čokoľvek zažiada. Inak povedané vyplní formulár a zvolí jednu z možností na jeho spracovanie. Vo väčšine prípadov je žiadateľom pracovník personálneho oddelenia, ktorý takéto žiadosti vypĺňa pre zamestnancov meniacich pracovné zaradenie.

1.4.2 Subjekt

Subjektom žiadosti je užívateľ, pre ktorého sa vo formulári žiada. Tomuto užívateľovi budú vo výsledku pridelené požadované oprávnenia.

1.4.3 Nadriadený

Ide o priameho nadriadeného subjektu.

1.4.4 Schvaľovateľ žiadosti

Vyplnený formulár je odoslaný na schválenie a doplnenie nadriadeným. Nadriadený ma teda možnosť schváliť žiadosť. Schvaľovateľom žiadosti môže byť aj užívateľ v roli ROLE_HR_EMPLYEE, ktorý má vďaka tejto roli oprávnenie schvaľovať žiadosť. Opäť pôjde o pracovníka personálneho oddelenia, ktorý vie na základe pracovnej pozície určiť, aké základné oprávnenia subjekt k práci potrebuje.

1.4.5 Schvaľovateľ oprávnenia

Akonáhle je formulár doplnený a schválený nadriadeným, príde na radu schvaľovanie jednotlivých oprávnení. Oprávnenie môže a nemusí mať nastaveného schvaľovateľa. V prípade, že oprávnenie nastaveného schvaľovateľa má, bude toto oprávnenie pridelené až po schválení týmto schvaľovateľom. V opačnom prípade je oprávnenie možné prideliť už na základe schválenej žiadosti nadriadeným.

1.4.6 Správca oprávnenia

Správca oprávnenia je zodpovedný za prideľovanie oprávnení užívateľom. Rôzne oprávnenia majú rôznych správcov. Systém notifikuje správcu oprávnenia, akonáhle dôjde ku schváleniu jeho oprávnenia.

1.5 Prípady použitia

1.5.1 Výber formuláru podľa typu

- Popis: Užívateľ si kliknutím na odkaz vyberie jeden z nasledujúcich typov formulárov:
  - Zmena v technických štruktúrach
  - Nástup zamestnanca
  - Prestup zamestnanca
  - Výstup zamestnanca

- Hlavní aktéri: Žiadateľ
- Vedľajší aktéri: Žiadni
- Vstupné podmienky: Žiadateľ vyberie taký typ formuláru, aký mu jeho užívateľská rola v systéme umožňuje používať.
- Hlavný scenár: Systém užívateľovi zobrazí formulár, obsahujúci údaje popísané vo funkčnom požiadavku 1.2.1.1 Interaktívne formuláre
- Výstupné podmienky: Systém zobrazí formulár reprezentujúci žiadosť v stave „Nová“
- Alternatívny scenár: Užívateľ vybral formulár, do ktorého nemá mať prístup. Systém mu ukáže hlásenie s odôvodnením a na stránku formuláru ho nepustí.

1.5.2 Vyplnenie formuláru

- Popis: Žiadateľ vyplní potrebné údaje vo formulári a označí požadované oprávnenia.
- Hlavní aktéri: Žiadateľ
- Vedľajší aktéri: Žiadni
- Vstupné podmienky: Úspešný priebeh prípadu použitia 1.5.1 Výber formuláru podľa typu
- Hlavný scenár:

  1. Žiadateľ má k dispozícii prázdny formulár. Typicky začína vyplňovaním informácií o zaradení zamestnanca.
     - Subjekt
     - Dátum platnosti
     - Jednotka
     - Pozícia
     - Lokalita
     - Nadriadený

     Zoznam zamestnancov sa bude dynamicky doťahovať už podľa prvých pár písmen priezviska zamestnanca. V zozname bude uvedené celé meno zamestnanca a jeho identifikačné číslo v rámci firmy. Akonáhle žiadateľ vyberie subjekt zo zoznamu zamestnancov, ostatné známe údaje o tomto zamestnancovi sa automaticky predvyplnia. V prípade, že ide o zamestnanca, ktorý je vo firme

krátko a ešte nestihol prebehnúť proces pridania užívateľa do Intranetu a aktualizácia dát, tento proces je popísaný vo funkčnom požiadavku 1.2.2.6 Aktualizácia dát, žiadateľ chýbajúce údaje doplní. Aby systém neobmedzoval užívateľov, sú z vyššie uvedených údajov povinné iba vyplnenie subjektu a nadriadeného.

2. Ďalej žiadateľ prejde na časť formuláru zameranú na výber oprávnení. Oprávnenia sú zaradené pod informačné systémy a tie sú ďalej zaradené pod typy informačných systémov. Žiadateľovi sa teda zobrazujú formou vnoreného zoznamu. Žiadateľ si vo formulári zaškrtne checkboxy reprezentujúce výber požadovaného oprávnenia a v prípade oprávnení obsahujúcich povinné údaje, tieto údaje vyplní.
3. V poslednej časti formuláru žiadateľ zvolí, čo s vyplneným formulárom chce ďalej urobiť.

- Výstupné podmienky: Vyplnený formulár pripravený k schváleniu nadriadeným.
- Alternatívny scenár: Žiadny

1.5.3 Uloženie rozpracovaného formuláru

- Popis: Žiadateľ formulár nestihol dokončiť a plánuje sa k nemu vrátiť neskôr.
- Hlavní aktéri: Žiadateľ
- Vedľajší aktéri: Žiadni
- Vstupné podmienky: Úspešný priebeh prípadu použitia 1.5.2 Vyplnenie formuláru.
- Hlavný scenár: Žiadateľ klikne na tlačítko „Uložiť“, čím sa formulár uloží do databázy. Takáto žiadosť zostáva naďalej v stave nová.
- Výstupné podmienky: Formulár sa nachádza v zozname žiadostí a žiadateľ má kedykoľvek možnosť sa k nemu vrátiť.
- Alternatívny scenár: Žiadny

1.5.4 Odoslanie formuláru na doplnenie a schválenie nadriadeným

- Popis: Žiadateľ za seba vyplnil všetky potrebné údaje a formulár posiela na schválenie a doplnenie nadriadeným.
- Hlavní aktéri: Žiadateľ

- Vedľajší aktéri: Žiadni
- Vstupné podmienky: Úspešný priebeh prípadu použitia 1.5.2 Vyplnenie formuláru.
- Hlavný scenár: Žiadateľ klikne na tlačítko „Odoslať“ na schválenie, formulár zmení stav na „Čaká na schválenie“. V tejto chvíli môže žiadosť editovať iba vybraný nadriadený. Systém rozošle notifikácie prostredníctvom emailu na subjekt a nadriadeného.
- Výstupné podmienky: Nadriadený dostane email popisujúci žiadosť. Email obsahuje všetky potrebné informácie, ktoré daný formulár obsahuje, vrátane odkazu na stránku s konkrétnou žiadosťou. Nadriadený klikne na odkaz a v prehliadači sa mu zobrazí stránka s formulárom čakajúcim na doplnenie a schválenie.
- Alternatívny scenár: Nadriadený email prehliadne.

1.5.5 Schválenie formuláru nadriadeným

- Popis: Nadriadený žiadosť skontroluje, doplní oprávnenia, ktoré podla neho subjekt pre prácu potrebuje a schváli žiadosť.
- Hlavní aktéri: Nadriadený
- Vedľajší aktéri: Žiadni
- Vstupné podmienky: Úspešný priebeh prípadu použitia 1.5.4 Odoslanie formuláru na doplnenie a schválenie nadriadeným. V prípade, že je žiadateľ zároveň schvaľovateľom žiadosti, stačí úspešný priebeh prípadu použitia 1.5.2 Vyplnenie formuláru.
- Hlavný scenár: Nadriadený skontroluje informácie o zaradení zamestnanca. Skontroluje, prípadne upresní odôvodnenie pri vybratých oprávneniach. Doplní formulár o oprávnenia, ktoré sú pre prácu subjektu nevyhnutné a žiadateľ ich neoznačil. Nakoniec zvolí možnosť schváliť žiadosť.
- Výstupné podmienky: Žiadosť zmení stav na schválená. Systém rozošle notifikácie na schvaľovateľov vybraných oprávnení. Akonáhle formulár obsahuje oprávnenia, ktoré nepotrebujú dodatočné schválenie, schválením nadriadeným sa tieto oprávnenia automaticky dostanú zo stavu schválené. Na základe schválených oprávnení systém rozošle notifikácie na osoby, zodpovedné za prideľovanie týchto oprávnení.
- Alternatívny scenár: Nadriadený sa rozhodne žiadosť zamietnuť, čím definitívne ukončí proces spracovania žiadosti. Systém následne pošle notifikácie vo forme emailu. Túto notifikáciu dostane žiadateľ, subjekt a pre

kontrolu aj nadriadený. Alebo nastane prípad, že nadriadený sa ku žiadosti nevyjadrí a žiadosť zostane v stave „Čaká na schválenie“.

1.5.6 Schválenie alebo zamietnutie vybraných oprávnení

- Popis: „Schvaľovateľ oprávnenia“ za seba rozhodne, či daný subjekt má alebo nemá dostať vybrané oprávnenia.
- Hlavní aktéri: Schvaľovateľ oprávnenia
- Vedľajší aktéri: Žiadni
- Vstupné podmienky: Úspešný priebeh prípadu použitia 1.5.5 Schválenie formuláru nadriadeným. Alebo špeciálny prípad, keď je nadriadený zároveň žiadateľ a označí oprávnenia, u ktorých figuruje ako schvaľovateľ. Akonáhle v takomto prípade nadriadený schváli žiadosť, rovno schváli aj tieto oprávnenia.
- Hlavný scenár: „Schvaľovateľ oprávnenia“ si otvorí email, ktorý mu systém poslal. Klikne na odkaz s názvom „detail žiadosti“, následne sa mu v prehliadači otvori stránka obsahujúca daný formulár. Schvaľovateľ pozorne prejde celý formulár. Oprávnenia, ktoré môže schvaľovať, obsahujú jeho meno a radio button s možnosťami schváliť a zamietnuť. Keď prejde celý formulár, tak na konci formuláru klikne na tlačítko s názvom „rozhodnúť oprávnenia“. Týmto systém formulár spracuje a pre všetky novo schválené oprávnenia rozošle notifikácie užívateľom, ktorí sú zodpovední za prideľovanie oprávnení.
- Výstupné podmienky: Oprávnenia zmenia stav na schválené alebo zamietnuté, systém rozošle notifikácie príslušným užívateľom.
- Alternatívny scenár: „Schvaľovateľ oprávnenia“ email prehliadne a tým pádom o oprávnenia nerozhodne. Ďalší prípad je ten, keď „schvaľovateľ oprávnenia“ nejaké z vybraných oprávnení vo formulári prehliadne. Ako kontrola slúži tlačítko na konci formulára. Ak sa vo formulári stále nachádza nerozhodnuté oprávnenie, na konci formulára sa zobrazuje tlačítko rozhodnúť oprávnenia, toto tlačítko zmizne, akonáhle boli všetky oprávnenia rozhodnuté.

1.5.7 Zobrazenie záznamov v administrácii aplikácie

- Popis: Užívateľ v roli administrátora použije administráciu aplikácie pre zobrazenie záznamov z databázy
- Hlavní aktéri: Užívateľ v roli ROLE_ADMIN
- Vedľajší aktéri: Žiadni

- Vstupné podmienky: Žiadne

- Hlavný scenár: Užívateľ v menu aplikácie príde kurzorom myši na políčko administrácia, následne na to sa mu zobrazí drop-down list, z ktorého si vyberie ľubovolnú položku. Každá položka reprezentuje entitu v databáze, ale nie každá entita je zastúpená v tomto zozname. Je to pre to, že nie každá entita môže byť zobrazená v aplikácii. Po kliknutí na položku sa užívateľovi zobrazí stránka zobrazujúca dáta. Táto stránka štruktúrou odpovedá tabuľke, každý stĺpec má svoju hlavičku, ktorá obsahuje názov stĺpca, políčko pre vloženie textu na filtrovanie a možnosti vzostupného a zostupného zoradenia. Každý riadok obsahuje jednotlivé parametre a tlačítko pre možnosť editovania. Stránka obsahuje aj tlačítko pre pridanie záznamu do databázy.

- Výstupné podmienky: Užívateľ má k dispozícii prehľad databázovej entity v podobe tabuľky.

- Alternatívny scenár: Žiadny

1.5.8 Pridanie záznamu v administrácii aplikácie

- Popis: Užívateľ v roli administrátora použije administráciu aplikácie pre pridanie záznamu do databázy.

- Hlavní aktéri: Užívateľ v roli ROLE_ADMIN

- Vedľajší aktéri: Žiadni

- Vstupné podmienky: Úspešný priebeh prípadu použitia 1.5.7 Zobrazenie záznamov v administrácii aplikácie

- Hlavný scenár: Užívateľ klikne na tlačítko „pridať záznam“ v ľavom hornom rohu obrazovky. Následne sa mu zobrazí okno obsahujúce názvy a vstupné polia všetkých parametrov danej entity. Povinné vstupné polia sú označené hviezdičkou. Po vyplnení všetkých povinných polí systém užívateľovi, kliknutím na tlačítko „uložiť“, umožní pridať záznam do databázy.

- Výstupné podmienky: Systém obsahuje nový záznam v databáze.

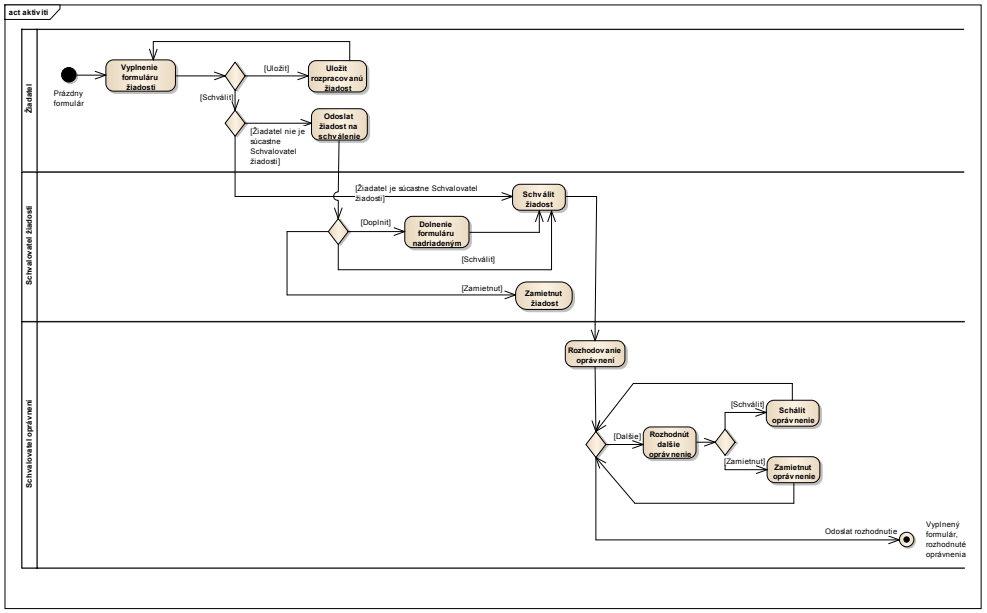
- Alternatívny scenár: Užívateľ nevyplní povinné pole a pokúsi sa pridať záznam. Následne ho systém upozorní chybovou hláškou pri nevyplnenom políčku.

1.5.9 Editácia záznamu v administrácii aplikácie

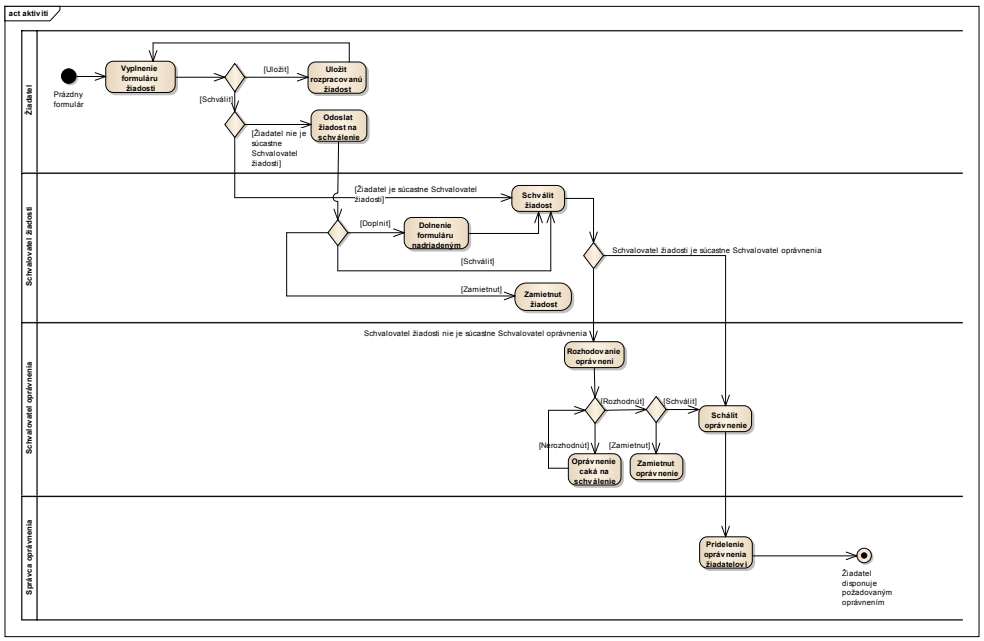
- Popis: Užívateľ v roli administrátora použije administráciu aplikácie pre editáciu záznamu do databázy.
- Hlavní aktéri: Užívateľ v roli ROLE_ADMIN
- Vedľajší aktéri: Žiadni
- Vstupné podmienky: Úspešný priebeh prípadu použitia 1.5.7 Zobrazenie záznamov v administrácii aplikácie
- Hlavný scenár: Užívateľ si zo zoznamu záznamov vyberie jeden, ktorý chce editovať. Vo vybranom riadku tabuľky kline na tlačítko „editovať záznam“. Následne sa mu zobrazí okno obsahujúce názvy a vyplnené vstupné polia všetkých parametrov danej entity. Povinné vstupné polia sú označené hviezdičkou. Jednou z editovateľných položiek je dátum konca logickej platnosti záznamu. Nastavenie tohoto dátumu slúži pre určenie, do kedy má záznam platnosť, zároveň je to jediný spôsob, akým administrácia umožní „vymazať“ záznam. Po editovaní vybraných polí systém užívateľovi, kliknutím na tlačítko „aktualizovať“, umožní aktualizovať záznam v databáze. Reálne ale ide o pridanie nového záznamu a invalidovanie predchádzajúceho, všetky zmeny sú takýmto spôsobom uchované v databáze.
- Výstupné podmienky: Editovaný záznam sa v databáze nachádza s aktualizovanými hodnotami.
- Alternatívny scenár: Užívateľ premaže povinné pole a pokúsi sa aktualizovať záznam. Následne ho systém upozorní chybovou hláškou pri nevyplnenom políčku.

1.6 Proces schvaľovania a prideľovania oprávnenia

Proces je reprezentovaný diagramom aktivít. Tento diagram je rozdelený do častí, ktoré sa nazývajú swimlanes. V prípade použitých diagramov reprezentuje každá časť iného príslušníka procesu schvaľovania. Diagram aktivít zároveň reprezentuje jednotlivé stavy procesu, a to v logickom poradí, v akom po sebe nasledujú. Diagram 1.1 reprezentuje proces, v ktorom sa spracúva formulár ako celok. A teda o jednotlivých oprávneniach je rozhodované individuálne. Diagram 1.2 reprezentuje proces od požiadania o oprávnenie, až ku jeho prideleniu.



Obr. 1.1: Diagram aktivít popisujúci proces schvaľovania oprávnení z pohľadu formuláru



Obr. 1.2: Diagram aktivít popisujúci proces schvaľovania oprávnení z pohľadu jedného oprávnenia

Kapitola 2

Návrh aplikácie

2.1 Zvolené technológie

2.1.1 Java

Technológia Java je kombináciou programovacieho jazyka a platformy.

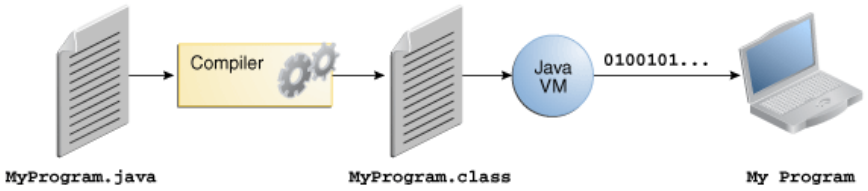
2.1.1.1 Programovací jazyk Java

Jazyk Java sa radí medzi jazyky vyššej abstrakcie a je charakteristický týmito vlastnosťami:

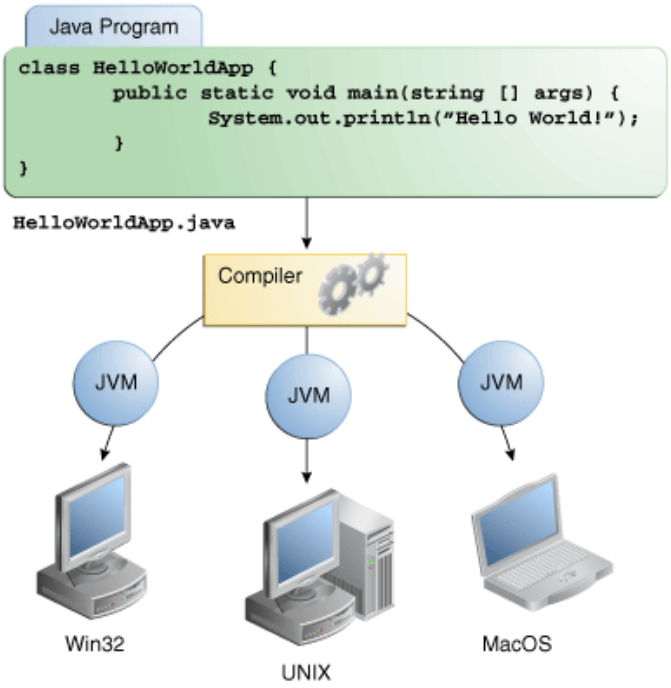
- architektonicky neutrálny
- objektovo orientovaný
- distribuovaný
- viacvláknový
- dynamický
- prenosný
- výkoný
- robustný
- bezpečný

Zdrojový kód je písaný v súboroch s koncovkou `.java`, tieto súbory sú preložené prekladačom `javac` do `.class` súborov, a tie sú následne interpretované pomocou Java Virtual Machine - JVM.

JVM je k dispozícii pre rôzne operačné systémy, možno teda použiť rovnaké `.class` súbory pre beh na Microsoft Windows, Solaris ™ Operating System (Solaris OS), Linux alebo Mac OS.



Obr. 2.1: Proces vývoju v jazyku Java. Prevzaté z [2]



Obr. 2.2: JVM umožňuje chod aplikácie na rôznych platformách. Prevzaté z [2]

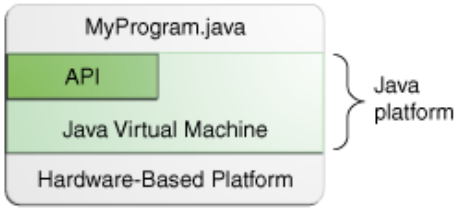
2.1.1.2 Platforma Java

Platforma je hardwarové alebo softwarové prostredie, v ktorom program beží. Vyššie sú spomenuté niektoré z najpopulárnejších platforiem, ako sú Microsoft Windows, Linux, Solaris a Mac OS. Väčšina platforiem môže byť označených ako kombinácia operačného systému a hardware. Platforma Java sa líši od väčšiny ostatných platforiem v tom, že ide o čisto softwarovú platformou, ktorá beží na iných hardwarových platformách. Rozlišujú sa rôzne platformy Javy. Základ tvorí platforma Java SE (standart edition) určená pre desktopové zariadenia a servery. Ďalej java poskytuje platformy ako napríklad platforma JavaCard bežiaca na čipových kartách, platforma Java ME (mobile edition)

bežiaca na mobilných zariadeniach, platforma Java DB, ktorá poskytuje open source databázu. V neposlednom rade Java poskytuje platformu Java EE, ktorá umožňuje vytvárať rozsiahle webové aplikácie. Súčasťou Javy je technológia Enterprise JavaBeans (EJB), jedná sa o dôležitú serverovú komponentu platformy Java, Enterprise Edition (Java EE). EJB umožňuje jednoduchší vývoj distribuovaných aplikácií. EJB sa používa ako komponent pre vytvorenie rôznych vrstiev, ktoré majú rozličnú funkcionalitu, obsahujú business logiku, pristupujú k databáze, sú prístupné z viacerých klientov.

Platforma Java má dve zložky:

- Java Virtual Machine: Je to základná komponenta platformy Java, ktorá je vytvorená v osobitných variantách pre rôzne hardwarové platformy.
- Java Application Programming Interface (API): Ide o veľkú kolekciu pripravených softwarových komponent, ktoré poskytujú mnoho užitočných funkcií. Program je rozdelený do knižníc, súvisiacich tried a rozhraní. Tieto knižnice sú známe ako balíčky.



Obr. 2.3: Platforma Java v kombinácii s hardwarovou platformou. Prevzaté z [2]

Nezávislosť technológie Java od platformy spôsobuje to, že táto technológia môže byť pomalšia ako natívny kód. Avšak zlepšovanie úrovne kompilátorov a technológii používaných v JVM tento rozdiel postupne znižuje.

2.1.2 JavaServer Faces

JavaServer Faces (JSF) je štandard Java frameworku pre tvorbu webových aplikácií, ktorý zjednodušuje stavbu užívateľských rozhraní (UI) pre serverové aplikácie [3]. JSF je podobné iným Java technológiám založeným na báze webovej aplikácie. Aplikácia beží na serveri v kontajneri Java servletu, klient-server architektúra zabezpečuje, že všetká funkcionalita ja vykonávaná na serverovej strane a výsledok následne prenesený na stranu klienta [4]. JSF aplikácia obsahuje:

- JavaBeans komponenty ako modely, ktoré obsahujú funkčnosť a dáta pre konkrétnu aplikáciu. Ide o `.java` triedy, ktoré pomocou anotácie

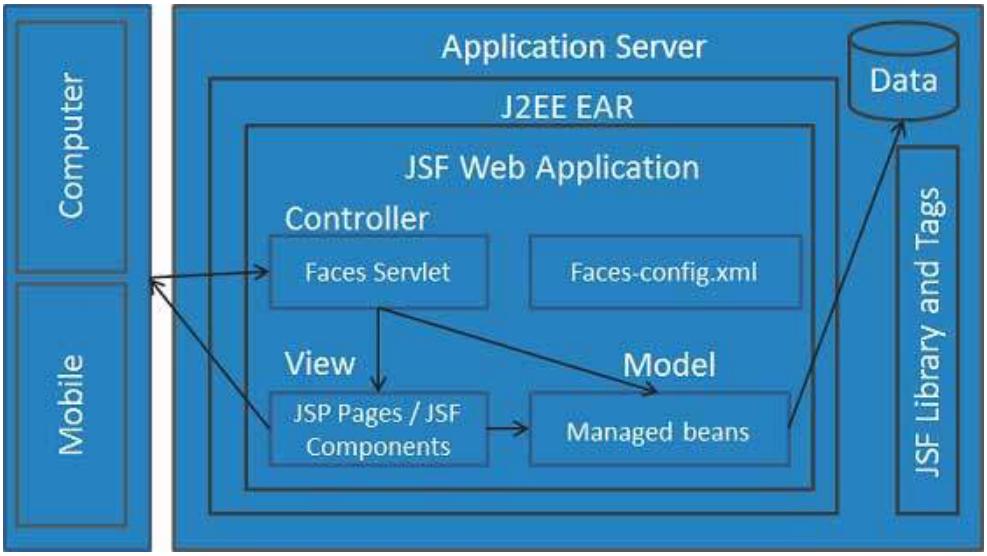
alebo konfiguračného súboru FacesConfig.xml, označíme ako Managed-Beans, tieto triedy sú teraz prístupné z JSF komponent v `.xhtml` stránkach a podliehajý životnému cyklu JSF.

- Jednoduchý prenos dát medzi jednotlivými UI komponentami.
- Široká škála UI komponent a možnosť implementácie vlastných komponent.
- Komponenty pre spracovávanie udalostí.
- Prepojenie udalosti na strane klienta s kódom na strane serveru.
- Komponenty pre validáciu vstupných dát a možnosť implementácie vlastného validátora.
- Poskytuje funkcionalitu pre pripojenie UI komponent k databáze priamo v `.xhtml` stránke.
- UI komponenty reprezentujúce stavové objekty na serveri.
- Pomocné triedy na strane servera.
- Vlastné spracovanie navigácie medzi jednotlivými `.xhtml` stránkami.
- FacesConfig.xml aplikačný konfiguračný súbor pre správu a konfiguráciu prostriedkov.

2.1.2.1 Návrhový vzor MVC

JSF zaisťuje, že aplikácie sú dobre navrhnuté vďaka integrácii s dobre zavedeným Model-View-Controller (MVC) návrhovým vzorom priamo do ich architektúry [5]. Návrhový vzor MVC rozdeľuje aplikáciu na tri oddelené moduly:

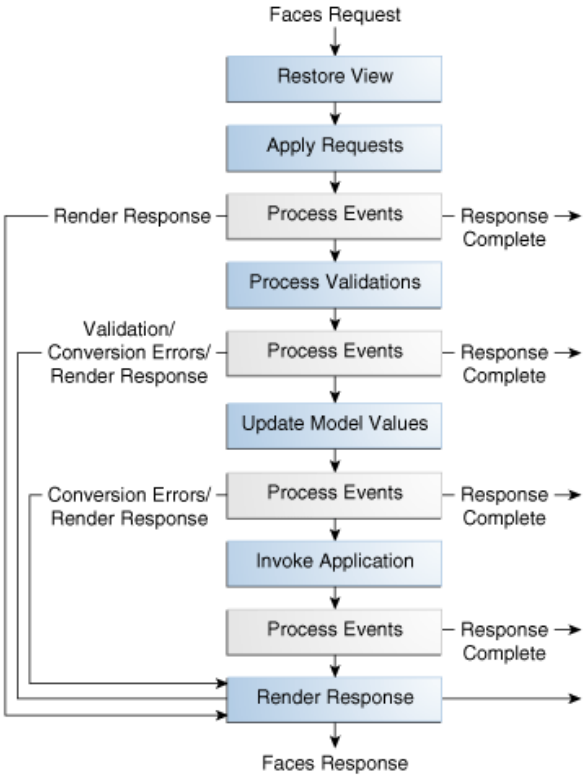
- Model: Obsahuje aplikačnú a business logiku rozdelenú do rôznych vrstiev.
- View: Užívateľské rozhranie aplikácie a teda `.xhtml` stránky alebo `.jsp` obsahujúce JSF komponenty
- Controller: Ovláda spracovanie aplikácie, ide o FasesServlet, ktorý obsluhuje requesty na server.



Obr. 2.4: Návrhový vzor MVC v JSF architektúre. Prevzaté z [6].

2.1.2.2 Životný cyklus JavaServer Faces stránky

Životný cyklus JavaServer Faces stránky sa skladá z nasledujúcich fáz:



Obr. 2.5: Životný cyklus Java Server Faces. Prevzaté z [7]

1. **Restore view** Vo chvíli keď užívateľ klikne na tlačítko alebo odkaz JSF, dostane požiadavok a začína túto fázu. Počas tejto fázy sa JSF vytvorí `view`, zadrôtuje ovládače udalosti a validátory do komponent užívateľského rozhrania `view` do inštancie FacesContext. Inštancia FacesContext bude teraz obsahuje všetky informácie potrebné na spracovanie požiadavku.

2. **Apply request values** Po vytvorení alebo obnovení stromu komponent, každá zložka v tomto strome použije spôsob dekódovania pre extrakciu jeho novej hodnoty z parametrov požiadavku. Táto hodnota je držaná v komponente. Ak prevod zlyhá, vygeneruje sa chybová hláška, ktorá sa zaradí do fronty vo FacesConfig. Táto správa sa objaví počas fázy render response, spolu so všetkými chybami validácie. Ak niektorá z metód „eventListener“ volala renderResponse nad aktuálnou inštanciou FacesContext, JSF preskočí do fázy `render response`.

3. **Process validations** Počas tejto fázy JSF spracováva všetky validátory definované v použitých komponentách. Porovnáva hodnotu komponenty s parametrami validátora. Ak hodnota nie je validná, JSF pridáva chybovú hlášku do fronty v inštancii FacesContext a životný cyklus po-

stúpi fázy `render response` a znovu zobrazí pôvodnú stránku s chybovým hlásením.

4. **Update model values** Po tom čo JSF skontroluje, či sú údaje validné, prechádza strom komponentov a nastavuje ich hodnoty odpovedajúcim objektom na serverovej strane. JSF aktualizuje hodnoty parametrov nachádzajúce sa v ManagedBean. V prípade, že akákoľvek updateModels metóda volala renderResponse nad aktuálnou inštanciou FacesContext, JSF preskočí do fázy `render response`.

5. **Invoke application** Počas tejto fázy JSF spracováva všetky udalosti na úrovni aplikácie. Ide o udalosti, ako je napríklad odoslanie požiadavku alebo presmerovanie na inú stránku.

6. **Render response** v prípade, že aplikácie používa JSP stránky, JSF v tejto fázy nechá spracovať stránku JSP kontajnerom. Ak sa jedná o prvý požiadavok, tak počas spracúvania stránky JSP kontajnerom budú komponenty zo stránky pridané do stromu komponent. Ak sa nejedná o prvý požiadavok, tak komponenty už v strome komponent pridané sú, a preto nie je potreba pridávať ich tam znovu. V oboch prípadoch sa komponenty zobrazia, akonáhle JSP kontajner poprechádza značky na stránke

   V prípade, že počas fáz request values, process validations alebo update model values dôjde k chybe, zobrazí sa pôvodná stránka. Ak kód stránky obsahoval značky pre zobrazenie chybových hlášok, zobrazia sa v tejto fáze. Akonáhle sa zobrazí obsah view, stav odpovede požiadavku sa uloží, aby bol prístupný pre ďalšie požiadavky a bol pripravený na fázu restore view.

2.1.2.3 JSF - Managed Beans

Managed Bean je regulárna Java Bean trieda registrovaná pomocou JSF. Teda Managed Beana je Java Beana riadená frameworkom JSF. Managed bean obsahuje gettre, settre a metódy zabezpečujúce business logiku potrebnú pre front end. Managed Bean funguje ako model pre UI komponenty frameworku JSF, k Managed Beane je možné pristupovať z JSF stránky. Registrácia Managed Bean je možná pomocou konfiguračného súboru frameworku JSF `faces-config.xml` alebo pomocou anotácie priamo v konkrétnej triede. Registrácia znamená nastavenie mena, pomocou ktorého budú k Managed Beane ostatné komponenty pristupovať. Ďalej je potrebné nastaviť scope, ktorý určuje životný cyklus Managed Beany:

- **@RequestScoped** - Managed Beana žije iba medzi HTML požiadavkom a odpoveďou. Vytvorí sa na počiatku HTTP požiadavok a skončí,

pri dokončení odpovede spojenej s daným HTTP požiadavkom. Tento scope je vhodný pre bez stavový systém.

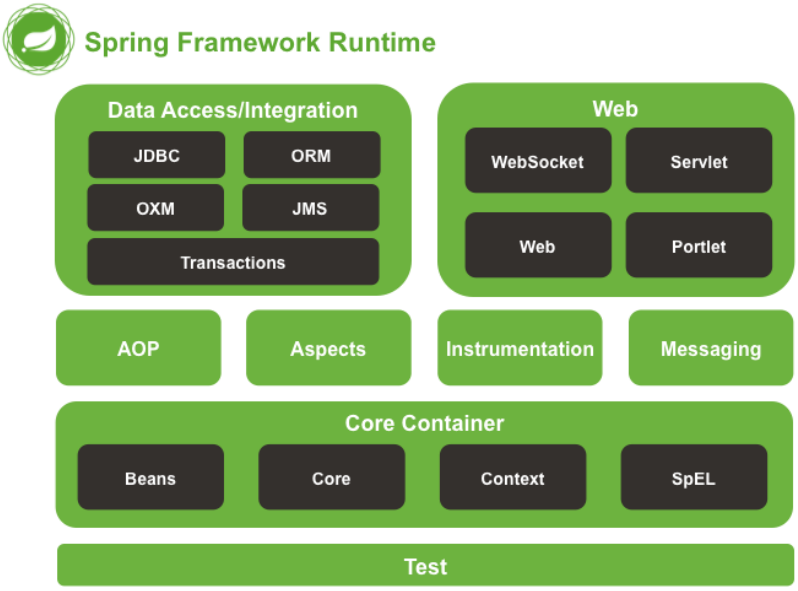
- **@SessionScoped** - Managed Beana žije tak dlho, ako HTTP session. Vytvorí sa pri prvom HTTP požiadavku a ukončí sa až s ukončením HTTP session. Tento scope je vhodné použiť pre Mannaged Beanu, ktorá v sebe drží informácie o tom, v akom jazyku si užívateľ praje systém používať.

Managed Beany použité v systéme budú pri používať práve tieto dva typy scope.

2.1.3 Spring

Spring framework umožňuje komplexné programovanie a konfiguráciu modelu pre moderné JavaEE aplikácie bežiace na ľubovolnej platforme [8]. Spring môže byť vnímaný ako alternatíva k EJB, ktorá je súčasťou JavaEE. Kľúčovým prvkom Spingu je podpora infraštruktúry na úrovni aplikácie, ktorá uľahčuje a zrýchľuje vývoj, a tým dáva vývojárovi viac priestoru pre prácu na business logike. Uľahčenie vývoja znamená predovšetkým:

- Pomoc pri odstránení tesných programových väzieb jednotlivých POJO objektov a vrstiev, za pomoci návrhového vzoru Inversion of Control
- Možnosť voľby implementácie (EJB, POJO) business vrstvy pre aplikačnú architektúru.
- Riešenie rôznych aplikačných domén bez nutnosti použitia EJB, napríklad transakčné spracovanie, podpora pre vzdialené business vrstvy formou webových služieb alebo RMI.
- Podpora implementácie komponentov pre prístup k dátam, či už formou priameho JDBC alebo ORM (Object Relation Mapping) technológií a nástrojov ako napríklad Hibernate.
- Zjednotenie konfiguračných súborov.
- Abstrakcia, ktorá vedie k zjednodušenému používanie ďalších častí Java EE, ako napríklad JMS, Java-Mail, JDBC alebo JNDI.
- Zjednodušenie používania a písanie unit testov.



Obr. 2.6: Kompletná architektúra frameworku Spring. Prevzaté z [9]

Spring framework je rozdelený do 20 rôznych modulov zobrazených na obrázku 2.6. Tieto moduly sú ďalej zoskupené do hlavných modulov AOP, DAO, ORM, Context, Web, MVC a Core modul. Niektoré z modulov sú na sebe závislé. Závislosť medzi dvoma modulmi znamená, že jeden modul používa triedy druhého modulu. Prípad závislosti modelov je uvedený na diagramoch architektúry springu, ktoré ukazujú napríklad závislosť ostatných modulov na module Core.

2.1.3.1 Spring Core

Modul Spring Core tvorí základ frameworku Spring a poskytuje funkcionalitu pre riadenie celého kontajneru. Spring Core poskytuje základné časti frameworku, medzi ktoré patria aj Inversion of Control a Dependency Injection.

Bean modul poskytuje BeanFactory, jedná sa o vyspelú implementáciu Factory Design Pattern. BeanFactory má na starosti životný cyklus POJO objektov.

2.1.3.2 Spring Context

Context modul stavia na základoch tvorených modulmi Core a Beans a umožňuje prístup ku všetkým rovno zadefinovaným a nastaveným objektom. Jadrom modulu Context je rozhranie ApplicationContext.

ApplicationContext je pokročilejší kontajner frameworku Spring definovaný rozhraním `org.springframework.context.ApplicationContext`. Rozširuje implementáciu BeanFactory, vie načítať definície bean, prepájať beany medzi sebou a na požiadanie dodať požadované beany. Ďalej pridáva enterprise funkcionalitu ako napríklad možnosť pristupovať k textovému obsahu `.properties` súborov. Prostredníctvom ApplicationContext sú jednotlivé udalosti v aplikácii publikované k nim priradeným eventListener funkciám.

2.1.3.3 Spring DAO

Tento modul sa používa pre komunikáciu s databázovým serverom pomocou JDBC. Pri použití Spring DAO modulu, nie je potreba písať zdĺhavý JDBC kód. o pripojenie k databáze sa postará JDBC šablóna.

Modul Transaction infrastructure podporuje programovú a deklaratívnu správu transakcií pre triedy, ktoré implementujú špeciálne rozhrania a pre všetky POJO (Plain Old Java objekty). Komplexná podpora transakcií je jedným z najväčších dôvodov používania frameworku Spring.

Programová správa transakcií znamená, mať kód pre správu transakcie okolo špecifického business kódu. To metóda dodáva extrémnu flexibilitu.

Deklaratívna správa znamená, oddelení správu transakcií business logiky. Používa sa anotácia alebo konfigurácie v `.xml` súbore. Táto metóda patrí do skupiny aspektového programovania a je plne pokrytá v module Spring AOP. Deklaratívny spôsob sa používa pre jednoduchšie prípady, programový pre tie komplexnejšie.

2.1.3.4 Spring ORM

Modul ORM poskytuje integračné vrstvy pre populárne objektovo-relačné mapovacie API, vrátane JPA, JDO a Hibernate. Spring ORM umožňuje použiť všetky objektovo relačné mapovacie frameworky v kombinácii so všetkými ostatnými funkciami, ktoré Spring ponúka.

2.1.3.5 Spring Web

Spring Web je určený na podporu integrácie s frameworkami pre tvorbu webových grafických používateľských rozhraní ako sú Struts, JSF alebo Wicket. Ďalej poskytuje podporné triedy, napríklad pre spracovanie nahrávania a sťahovania súborov, lokalizáciu textov alebo prácu s cookies. Spring Web modul poskytuje základnú integráciu s funkciami orientovanými na web ako inicializáciu IoC kontajneru pomocou Servletu a aplikačného kontextu zameraného pre web. Spring Web obsahuje aj HTTP klienta a časti súvisiace s webom z funkcie Spring remote.

2.1.3.6 Spring Web MVC

Spring Web MVC, tiež nazývaný Web-Servlet modul pozostáva zo Spring model-view-controller (MVC) a implementácie REST Web Services pre webové aplikácie. Spring MVC framework poskytuje čisté oddelenie medzi doménovým modelom, kódom a webovou časťou.

2.1.3.7 Spring AOP

Spring AOP implementuje podporu pre aspektovo orientované programovanie. Umožňuje separovať časti kódu naprieč celou aplikáciou, ktoré majú spoločný príznak napríklad autorizácia, logovanie alebo už spomenuté transakcie (jedná sa o deklaratívny spôsob použitia transakcií) a použiť ich na akýkoľvek POJO objekt.

2.1.4 Inversion of Control

(IOC) rieši väzby medzi aplikačnými objektami, ktoré sú tesne zviazané v zdrojovom kóde. IOC, alebo tiež obrátené riadenie, umožňuje uvoľniť vzťahy medzi inak tesne zviazanými komponentmi. Pri klasickom programovaní sa vytvára trieda, ktorá potom využíva ďalšie triedy a tak ďalej. Tým sú potom triedy veľmi pevne zviazané a zmena používanej triedy za inú vyžaduje zásah do kódu. IOC umožňuje uvoľniť túto väzbu. V podstate to znamená, že trieda nevytvára sama inštancie dalších tried, ktoré potrebuje, ale sú jej dodané určitým spôsobom z vonku. Týchto spôsobov existuje niekoľko, najznámejšie sú nasledujúce:

2.1.4.1 Constructor Injection

Triedy, do ktorých je vkladaná inštancia ďalšie triedy, ktorú potrebujú, musia mať vytvorený konštruktor, ktorý vie prijímať potrebné typy objektov.

2.1.4.2 Setter Injection

Triedy, do ktorých sú vkladané inštancie iných tried, musia mať pre tieto triedy zadefinované potrebné settre. Spring Framework používa práve tento spôsob injektáže. Všetky triedy je treba najprv v konfiguračnom súbore springu zadefinovať ako beany. Inštancie vkladaných tried sú následne vkladané ako parametre takýchto bean.

2.1.4.3 Interface Injection

Najprv je definované rozhranie, ktoré určuje metódy, pomocou ktorých budú nastavované inštancie objektov, ktoré príslušná trieda potrebuje. Každá trieda, ktorá chce tieto inštancie využívať, musí implementovať príslušné rozhrania.

2.1.5 Apache maven

Apche maven je nástroj pre správu, riadenie a automatizáciu buildov aplikácií a agendy s tým spojené. Hoci je možné použiť tento nástroj pre projekty písané v rôznych programovacích jazykoch, podporovaný je prevažne jazyk Java. Pokrýva päť základných oblasti:

- Uľahčenie procesu buildovania.
- Jednotný systém buildovania.
- Poskytovanie kvalitných informácií o projekte.
- Poskytovanie direktív pre „best practices“.
- Poskytnutie transparentného pridávanie nových funkcií.

Základným princípom fungovania Maven je popísanie projektu pomocou súboru `pom.xml` (Project Object Model) obsahujúceho model, ktorý popisuje softwarový projekt nielen z pohľadu jeho zdrojového kódu, ale vrátane závislosti na externých knižniciach, popisu procesu buildu a rôznych funkcií s tým spojených (ako je spúšťanie testov, zbieranie informácií o zdrojových kódoch a podobne). Súbor býva umiestnený v koreňovom adresári projektu. Ak má projekt viac podprojektov (sub-modulov), má každý z nich vlastný pom.xml, ktorý dedí vlastnosti z pom súboru nadriadeného modulu a pridáva ďalšie, vlastné. Tento princíp umožňuje build celého projektu jedným príkazom. Maven sám je postavený na modulárnej architektúre a funguje na princípe volania jednotlivých pluginov. Sám iba obstaráva dodanie a spustenie definovaných pluginov. Maven nemá žiadne vlastné grafické užívateľské rozhranie a ovláda sa iba pomocou príkazov príkazového riadku. Pluginy tak môžu využívať všetky nástroje, ktoré dokážu komunikovať pomocou štandardných vstupov [10].

2.1.6 HIbernate ORM

ORM znamená Object/Relational Mapping jedná sa teda o framework ktorý mapuje objekty na relačnú databázu. Java Persistence API(JPA), je špecifikácia toho, ako ukladať, získavať a spravovať dáta medzi Java objektami a relačnou databázou. Hibernate je konkrétnou implementáciou tejto špecifikácie. Hibernate je schopný generovať SQL príkazy pre rôzne typy databáz, s akým typom databázy pracuje je špecifikované pomocou parametru s názvom „dialect“. Pre interakciu s databázovým serverom používa Hibernate JDBC driver.

2.1.6.1 Microsoft SQL Server

Pre správu dát sa bude využívať Microsoft SQL Server, v kombinácii s operačným systémom Microsoft Server. Jedná sa o systém pre správu relačnej databázy. Pri návrhu dátového modelu, je použitá výhoda tejto technológie, ktorou je dĺžka jednotlivých tabuliek a stĺpcov. V dátovom modely sú tabuľky a stĺpce pomenované podľa ich business funkcie, vo väčšine prípadov takéto pomenovanie prekračuje 30 znakov, čo je pri niektorých databázach maximálna dĺžka. SQL server používa pre uloženie takýchto názvov dátový typ `nvarchar(128)` a teda umožňuje uložiť až 128 znakov. Ako nástroj pre administráciu databázy sa ponúka SQL Server Management Studio, jedná sa o integrované prostredie pre prístup, konfiguráciu, riadenie a správu všetky súčasti SQL Serveru. Management Studio kombinuje širokú škálu grafických nástrojov s piatimi bohatými editormi pre písanie skriptov v závislosti na úrovni schopností vývojárov a administrátorov [11].

2.1.7 HTML 5

HTML (HyperText Markup Language) je značkovací jazyk používaný na webových stránkach . O štandardizáciu jazyka sa stará W3C (World Wide Web Consorcium). Dňa 28.10.2014 bola štandardizovaná verzia HTML 5 [12]. Avšak mnoho častí HTML 5 bolo podporované väčšinou popredných webových prehliadačov už pár rokov pred štandardizáciou. Čo nepriamo ukazuje na skutočnosť, že štandardizácia je pomerne v sklze za skutočne používanými technológiami a verziami.

2.1.8 CSS 3

CSS (Cascading Style Sheet) je skratka pre jazyk, ktorý umožňuje vizuálne formátovanie dokumentov v jazyku HTML a všeobecne všetkých jazykoch, ktoré vychádzajú z XML (Extensible Markup Language). Kaskádové štýly umožňujú rozmiestnenie elementov, ich ohraničenie, farbu a mnoho ďalších možností, ako zmeniť vzhľad HTML a to všetko bez nutnosti meniť HTML obsah. Jedná sa teda o oddelenie obsahovej časti od grafického formátovania. Ďalšou výhodou kaskádových štýlov je možnosť zadefinovať zmenu vzhľadu pre rôzne zariadenia. Ide o takzvaný responzívny design, ktorý je v dnešnej dobe nevyhnutnou súčasťou každej webovej aplikácie či statickej stránky. Užívatelia si aplikácie zobrazujú nie len na monitoroch rôznych rozmerov, čo design a rozloženie stránky tak zásadne nezmení, ale hlavne na tabletoch a telefónoch, ktorých rozmery sa z roka na rok menia.

2.1.9 RESTové Webové služby a WSDL 2.0

Termín Webové služby je obvykle spájaný s operation-based a action-based službami, ktoré používajú SOAP a WS * štandardy, ako je napríklad WS-

Addresing a WS-Security. Termín RESTové Webové služby sa všeobecne odkazuje na Webové služby s resource-based architektúrou, ktorá používa HTTP a XML. Každá z týchto architektonických štýlov pre Webové služby má svoje uplatnenie, avšak až do nedávna neboli ekvivalentne podporované štandardom WSDL. WSDL 1.1 HTTP väzba nebola dostatočná, aby mohla popísať komunikáciu s HTTP a XML, takže neexistoval spôsob, ako formálne opísať RESTové Webové služby s WSDL. Zmena nastala zavedením verzie WSDL 2.0, ktorý bol navrhnutý s dôrazom na RESTové Webové služby, ako odporúčanie World Wide Web Consortium (W3C) [13].

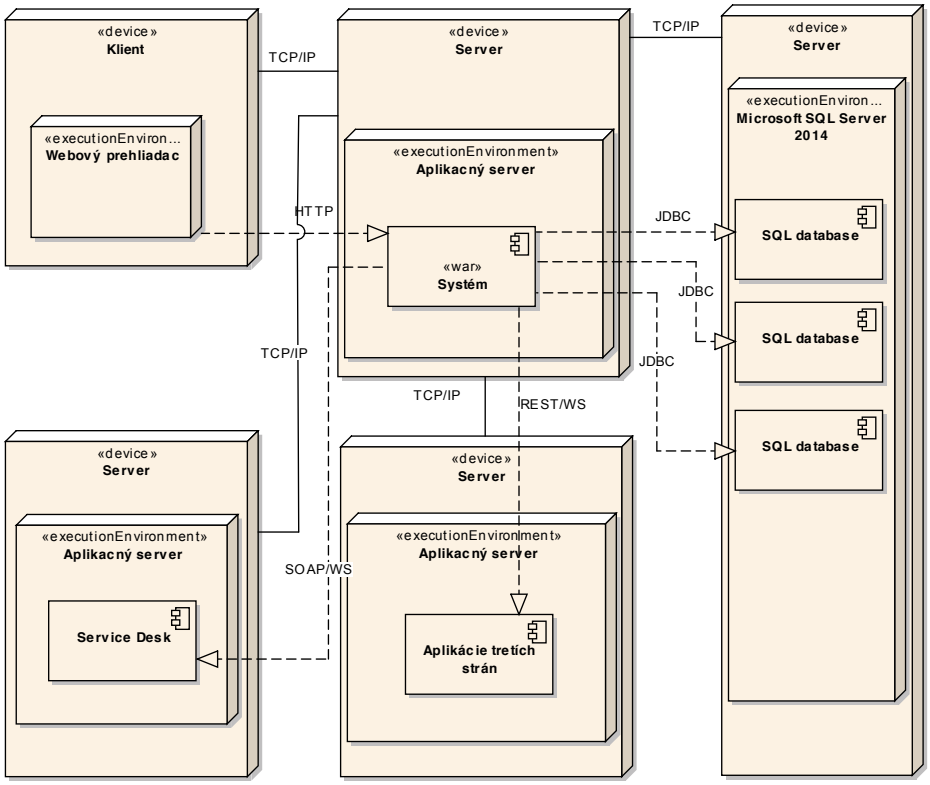
2.1.10 Zhodnotenie výberu technológií

Výber platformy Java EE je daný požiadavkami na systém, rovnako tak technológia JavaServer Faces. Jedným z výhod použitia JSF je trojvrstvová architektúra, ktorá oddeluje business logiku od prezenčnej vrstvy. Požiadavok je spracovaný priamo na serveri, rozhodovanie čo sa užívateľovi v závislosti na jeho užívateľskej roli zobrazí a čo nie prebieha už na serveri. Na klienta sa následne posiela stránka s HTML a JavaScript obsahom, ktorá bola na serveri vygenerovaná na základe Java kódu. Pre implementáciu systému je použitý projekt vytvorený z Maven archetypu, ktorý v sebe obsahuje komponenty, ktoré využívajú technológie Spring a Hibernate. Microsoft SQL Server je použitý z dôvodu už existujúcej infraštruktúry, ktorá zabezpečuje dostupnosť 24/7, zálohovanie a obnovu dát. Túto technológiu rovnako používa aj databáza `intranet_service`, podľa ktorej budú pravidelne aktualizované dáta týkajúce sa organizačnej štruktúry. Technológie HTML a CSS sú dané, keďže sa jedná o webovú aplikáciu. Výber technológie pre komunikáciu s aplikáciami tretích strán bol síce určený požiadavkami ale formát, v ktorom sa dáta budú prenášať v týchto požiadavkoch určený nebol. Použitý bude JSON, tento výber je založený na skutočnosti, že aplikácie tretích strán, s ktorými bude systém komunikovať budú slúžiť na zobrazovanie dát a to primárne prostredníctvom klientov písaných v jazyku JavaScript. Kód Java objektu reprezentovaný vo formáte Json, je totiž syntakticky identický s kódom objektu jazyku JavaScript.

2.2 Návrh architektúry

Všetky komponenty systému komunikujú nad TCP/IP. Systém bude prístupný cez webový prehliadač. Klient a server budú komunikovať cez HTTP protokol. Nosnou platformou pre systém bude server, ktorý bude používať operačný systém Windows Server. Na tomto serveri bude nainštalovaný aplikačný server Apache Tomcat verzia minimálne 8.x.x, pre vývoj bola použitá verzia 8.0.42. Systém bude vo formáte WAR nahratý do kontajneru servletu aplikačného serveru. Systém bude pracovať s relačnými databázami bežiacimi na Microsoft SQL Server 2014. Komunikácia s týmito databázami prebehne pomocou

JDBC. So systémom Service Desk sa bude komunikovať prostredníctvom webovej služby a protokolu SOAP. Komunikácia s ostatnými aplikáciami prebehne pomocou RESTových rozhraní.



Obr. 2.7: Diagram nasadenia znázorňujúci architektúru systému

2.3 Dátový model

Dátový model systému je popísaný z pohľadu jednotlivých tabuliek, ktoré pomocou relácií tvoria súvislý celok. Pre účely ukážky spôsobu tvorby modelu bolo vybraných iba niekoľko diagramov, na ktorých je možné vysvetliť použité postupy modelovania. Model je navrhnutý v nástroji Enterprise Architect (EA). Jednou z funkcií EA je funkcia generate Data definition code (DDL), táto funkcia slúži pre vygenerovanie SQL skriptu pre vytvorenie databázy. Takýto skript bude použitý pre vytvorenie a každé prípadné rozšírenie databázy. V databáze nikdy nebude dochádzať ku zmenám bez toho, aby k nim došlo už v dátovom modele.

2.3.1 Verziovanie dát

Jeden z požiadavkov na systém bola plná história dát v databáze. Pod týmto požiadavkom sa rozumie možnosť uchovať dáta tak, ako vyzerali pred zmenou a následne vyrobiť nové dáta už so zapracovanou zmenou. Pre návrh systému s takýmito vlastnosťami je treba začať už v dátovom modele. V modele sa vyskytujú aj tabuľky reprezentujúce objekty, ktoré verziované nie sú, jedná sa o tabuľky, ktorých dáta sa v rámci business logiky systému needitujú. Pôjde napríklad o tabuľky reprezentujúce typy iných tabuliek, prístupové práva do systému, odoslaný email alebo udalosť reprezentujúcu neobvyklé chovanie systému. Verziované objekty budú v modele reprezentované hlavnou tabuľkou ďalej Master tabuľka a dátovou tabuľkou ďalej Data tabuľka. Master tabuľka obsahuje vždy iba `id` objektu, zároveň pôjde o primárny kľúč tejto tabuľky. Data tabuľka obsahuje `id` objektu Data, teda primárny kľúč tabuľky Data, ďalej cudzí kľúč reprezentujúci `id` Master tabuľky. Väzba medzi týmito tabuľkami je 1..N, teda Master tabuľka má viacero Data tabuliek a Data tabuľka má práve jednu Master tabuľku. Každá Data tabuľka ďalej obsahuje položku `inserted_by`, táto položka reprezentuje užívateľa, ktorý vložil záznam do databázy respektíve urobil posledný zmenu. A nakoniec každá Data tabuľka obsahuje štyri položky typu datetime: `vf` (valid from), `vt` (valid to), `ef` (effective from), `et` (effective to).

Systém bude naprogramovaný tak, že každé vloženie nového záznamu do databázy znamená vytvorenie Master tabuľky a Data tabuľky s nastavenými parametrami `inserted_by` na `id` užívateľa, ktorý záznam vkladá, nastavené `vf` na aktuálny dátum a nastavené `vt` na hodnotu `null`. Hodnota `null` u `vt` znamená, že sa jedná o platný záznam. Pri editácii záznamu bude uloženiu dát do databázy predchádzať zneplatnenie aktuálneho záznamu data nastavením `vt` na aktuálny dátum a pridanie nového záznamu data nastaveným `inserted_by`, `vf` na aktuálny dátum a `vt` na `null`. Aplikácia bude naprogramovaná tak, aby neumožňovala do databázy pridať záznam tabuľky Data bez toho, aby bol predchádzajúca verzia invalidovaná nastavením `vt` na aktuálny dátum. Pri vyťahovaní záznamov z databázy pre účely business logiky budú použité modifikované hibernate criteria, ktoré k Master tabuľke dotiahnu vždy iba validnú Data tabuľku. V prípade viac alebo žiadnych validných dát systém užívateľa upozorní vyhodením výnimky. Pre účely náhľadu do histórie cez administráciu budú dotiahnuté aj invalidné teda už historické dáta.

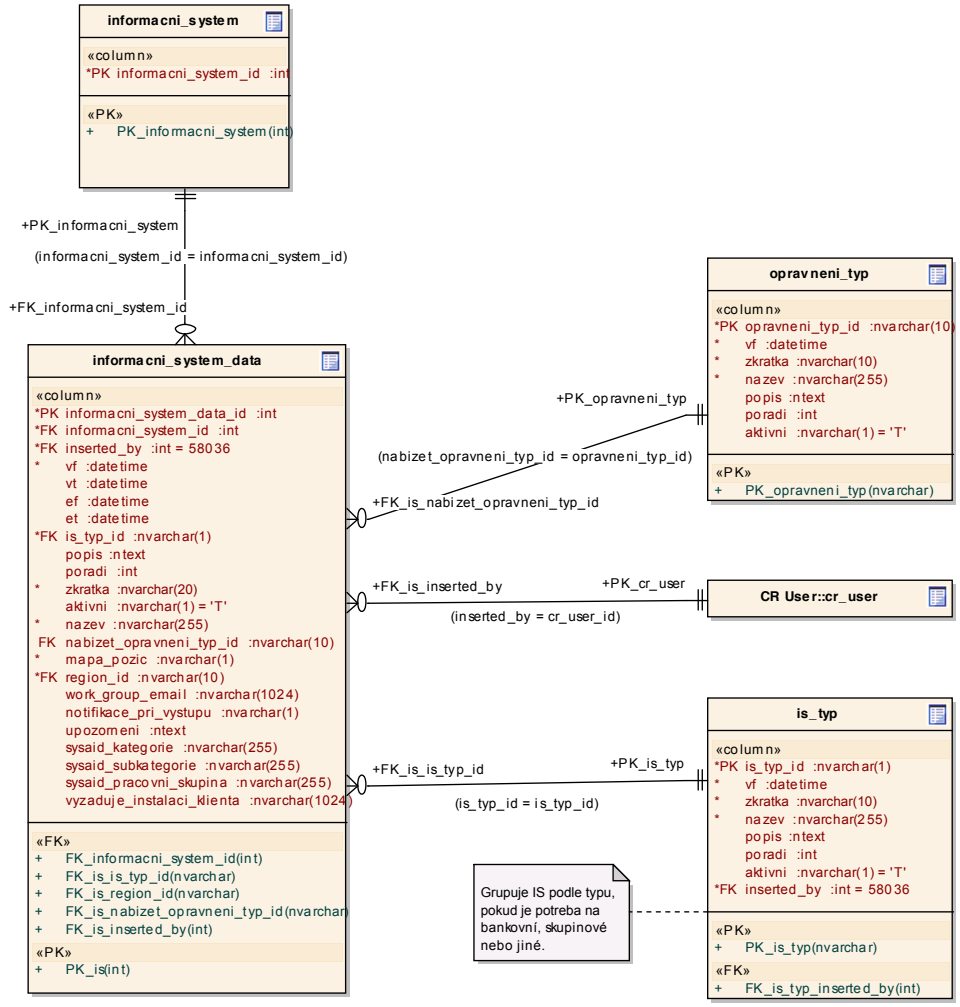
Parametre `ef` a `et` plnia podobnú, funkciu s tým rozdielom, že tieto parametre ukazujú na to, či je záznam aktívny z pohľadu business logiky. Táto vlastnosť sa nazýva efektívna platnosť, tým pádom je záznam aktivovaný iba vtedy, ak je aktuálny dátum v rozmedzí `ef` a `et`. Nastavovanie parametrov bude narozdiel od `vf` a `vt` možné prostredníctvom administrácie. Typický prípad neefektívneho záznamu je deaktivácia užívateľa počas rodičovskej dovolenky. To, ako bude s neaktívnou hodnotou naložené, závisí od konkrétneho prípadu použitia. Ak majú dáta neefektívnu platnosť, neznamená to, že sa

dáta z databázy do systému nedostanú. Tabuľky v databáze obsahujú položku s názvom `aktivni`, ktorá nadobúda hodnotu `T` a `F` ako `true` a `false`, položka má rovnaký účel ako `ef` a `et`. Používa sa už len z historických dôvodov pre označenie neaktívnych dát vzniknutých migráciou z iných systémov. V prípade dát vzniknutých v tomto systéme táto položka nemusí byť použitá.

Tabuľky reprezentujúce neverziované objekty obsahujú `id` objektu ako primárny kľúč, ďalej môžu obsahovať `vf`, ktoré slúži pre určenie dátumu vytvorenia záznamu, položku `aktivni` a nakoniec položku `inserted_by`. Neverziované objekty nemusia mať primárny kľúč typu `integer`. V systéme sa nachádzajú aj neverziované objekty, ako napríklad systémové práva, ktorých jediným parametrom je ich `id`, teda primárny kľúč. Takýto kľúč je typu `nvarchar` a priamo reprezentuje pomenovanie práva, napríklad `CAN_READ_ALL_ZADOST` je oprávnenie pre príslušníkov personálneho oddelenia, ktoré im umožňuje vidieť všetky žiadosti užívateľov.

Väčšina tabuliek obsahuje business parametre ako `nazev`, `zkratka`, `popis` a `poradi`. Parameter `poradi` určuje ako sa dáta na frontende radia, bude ho možné jednoducho nastaviť v administrácii systému. Ostatné parametre tabuliek už závisia na individuálnej funkcii daného objektu v systéme.

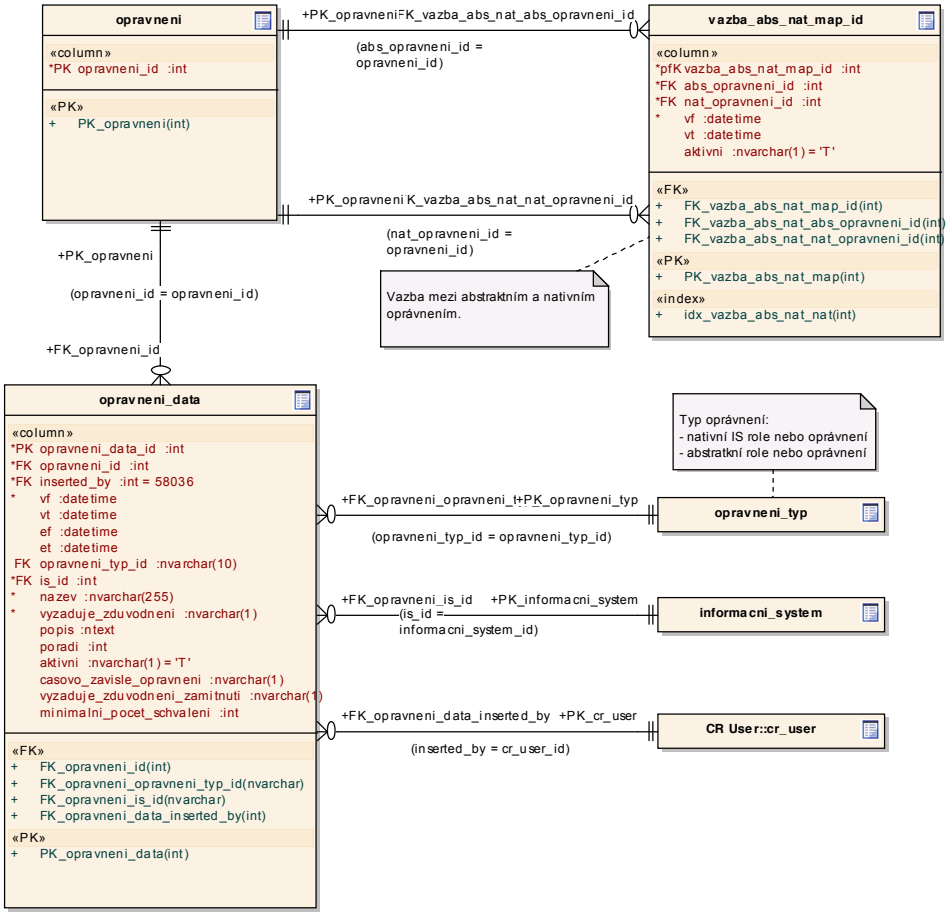
2.3.2 Informačné systémy a oprávnenia



Obr. 2.8: informacni_system

2.3.2.1 Informačný systém

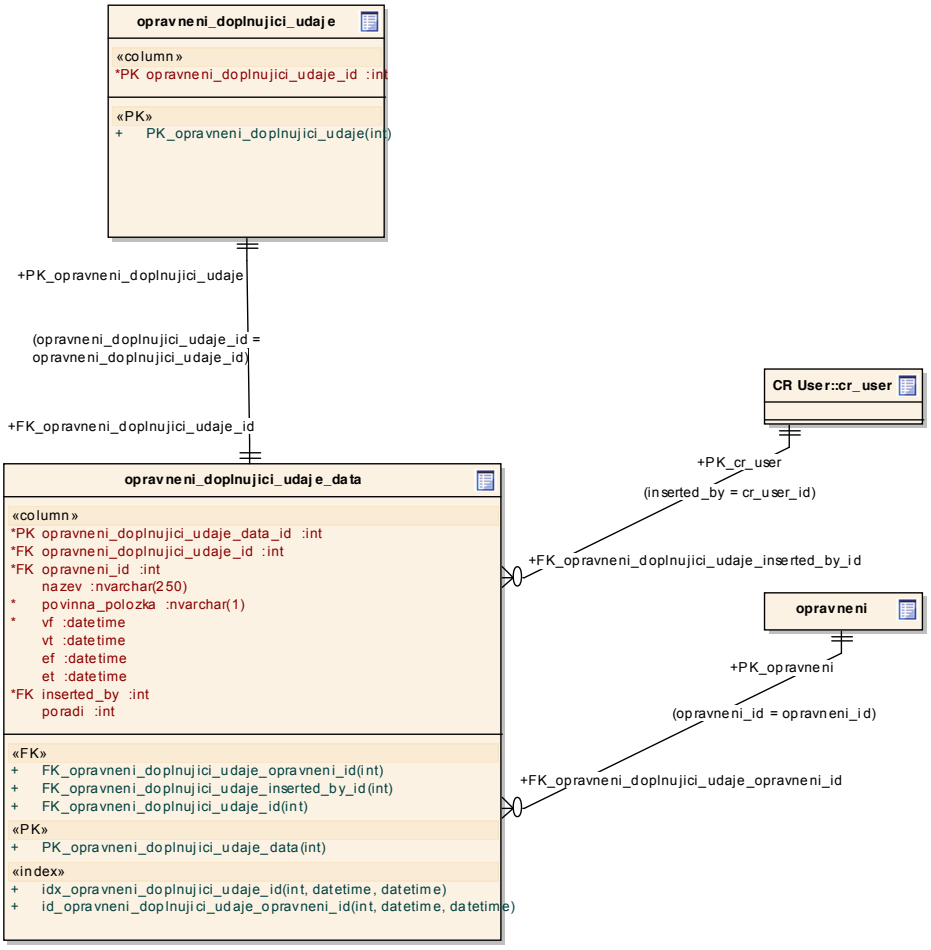
Verziovaná tabuľka 2.8 `informacni_system_data` v sebe obsahuje referenciu na neverziovanú tabuľku `is_typ`, je použitá väzba `1..N`. Vďaka tejto väzbe bude vygenerovaný objekt `InformacniSystemData` obsahovať položku `IsTyp` a objekt `IsTyp` bude naopak obsahovať kolekciu objektov `InformacniSystemData`.



Obr. 2.9: opravneni

2.3.2.2 Oprávnenie

Verziovaná tabuľka 2.9 `opravneni_data` zase obsahuje referenciu na tabuľku `informacni_system`, princíp je rovnaký ako v predchádzajúcom prípade. Vygenerovaný objekt `OprávneniData` bude obsahovať položku `InformacniSystem` a objekt `InformacniSystem` bude obsahovať kolekciu objektov `OprávneníData`.

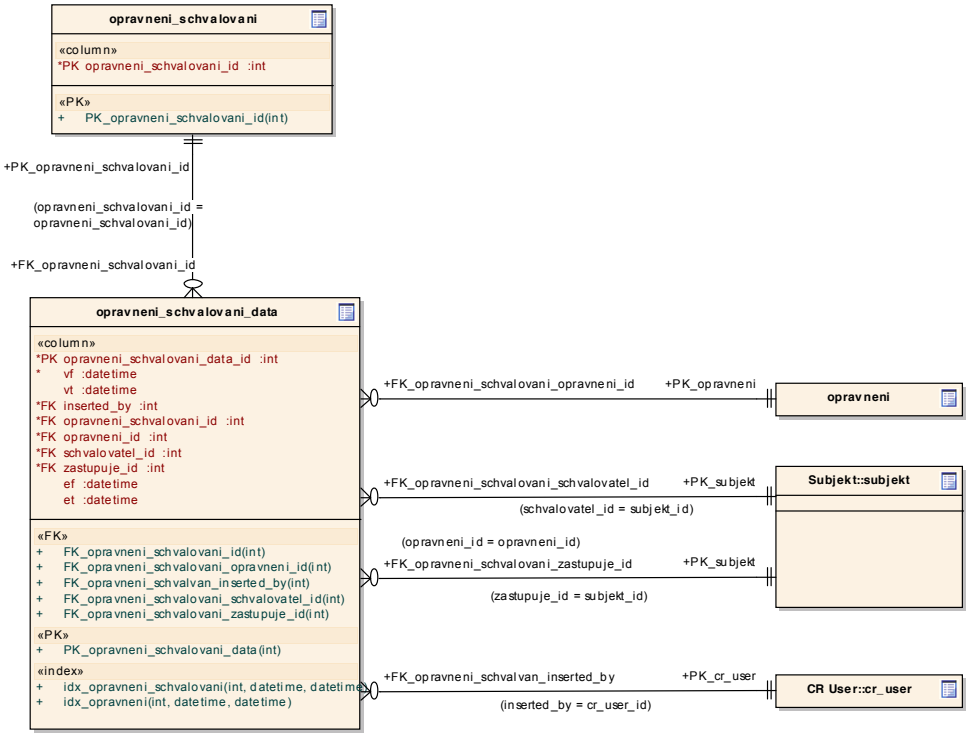


Obr. 2.10: opravneni_doplnujici_udaje

2.3.2.3 Doplňujúce údaje oprávnení

Pri žiadaní o oprávnenie bude v niektorých prípadoch potreba okrem odôvodnenia vyplniť aj rôzne doplňujúce údaje. Tieto doplňujúce údaje sú závislé na individuálnych oprávneniach. Napríklad oprávnenie s názvom „Inštalácia neštandardného softwaru“ požaduje odkaz na neštandardný software. z tohoto dôvodu bolo potrebné zaviesť pre doplňujúce údaje samostatnú tabuľku, vďaka ktorej bude prostredníctvom administrácie, možné dynamicky pridávať a odoberať doplňujúce údaje ľubovolného oprávnenia. Jednoducho sa do databázy uloží nová inštancia objektu `OpravnniDoplnujiciUdaje`, ktorému sa ako `Opravneni\verb` nastaví oprávnenie s názvom „Inštalácia neštandardného softwaru“ a do položky `nazev\verb` vloží text „Odkaz na neštandardný software“. Pôjde o verziovaný objekt, závislý na objekte `Opravneni` a teda

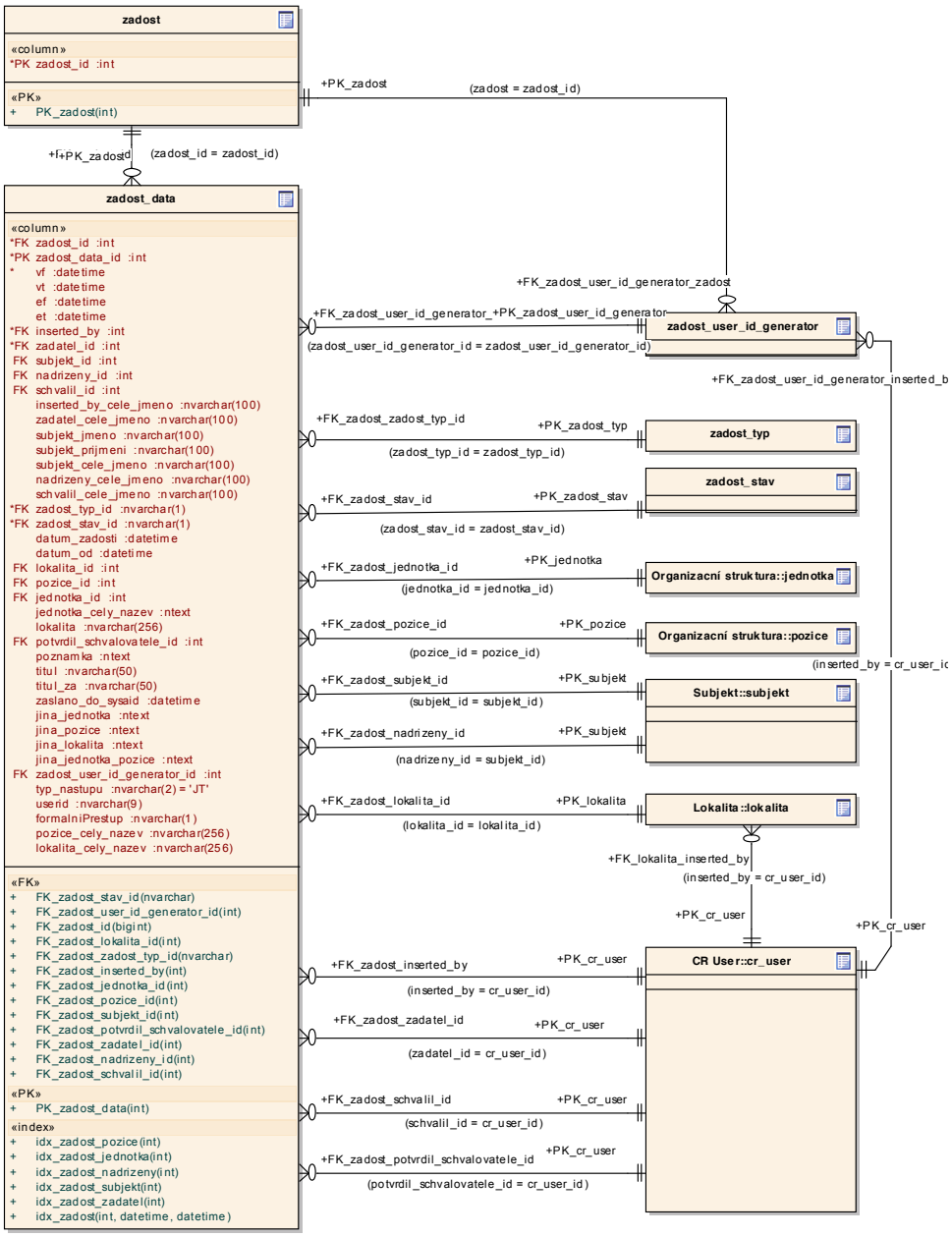
tabuľka 2.10 `opravneni_doplnujici_udaje_data` obsahuje väzbu na tabuľku `opravneni`. Vygenerovaný objekt `OpravneniDoplnujiciUdajeData` bude obsahovať položku `Opravneni`, a objekt `Opravneni` bude obsahovať kolekciu objektov `OpravneniDoplnujiciUdajeData`.



Obr. 2.11: opravneni_schvalovani

2.3.2.4 Schvaľovatelia oprávnení

O väčšine oprávnení budú rozhodovať rôzni schvaľovateľia. Každé oprávnenie môže a nemusí mať schvaľovateľa, schvaľovateľov môže byť zároveň viac. z tohoto dôvodu je zase vhodné použiť samostatnú tabuľku. Použitý je rovnaký princíp ako v predchádzajúcom prípade, tabuľka 2.11 `opravneni_schvalovani_data` obsahuje referenciu na tabuľku `opravneni`. Vygenerovaný objekt `OpravneniSchvalovaniData` bude obsahovať položku `Opravneni` a objekt `Opravneni` bude obsahovať kolekciu objektov `OpravneniSchvalovaniData`.

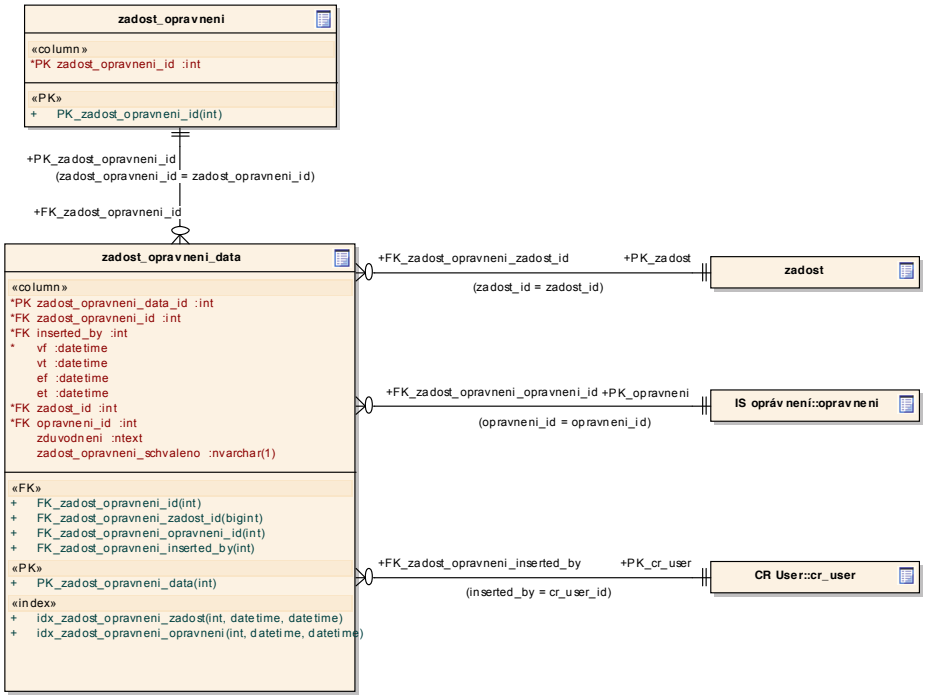


Obr. 2.12: zadost

2.3.2.5 Žiadosť

Tabuľka 2.12 `zadost_data` obsahuje všetky parametre potrebné k podaniu žiadosti. Pri niektorých dátach tabuľka obsahuje cudzí kľúč a následne ešte aj položku typu `nvarchar` rozšírenú o `_cele_jmeno` alebo `_cely_nazev`. Duplicitné

parametre typu `nvarchar` umožnia do systému migrovať žiadosti vytvorené pôvodným systémom, ktoré môžu obsahovať referencie na dáta, ktoré v novom systéme existovať nebudú. Pre nové žiadosti bude zase platiť, že pri ukladaní žiadosti sa parametre typu `nvarchar` nastavia z referencovaných objektov. Pôjde o užívateľov, pozície, lokality a jednotky. Špecifická je referencia na tabuľku `zadost_user_id_generator`. Primárne kľúče takýchto tabuliek, budú použité v identifikačných číslach nových užívateľov, ktorých systém automaticky vytvorí pri schválení formuláru typu nástup.

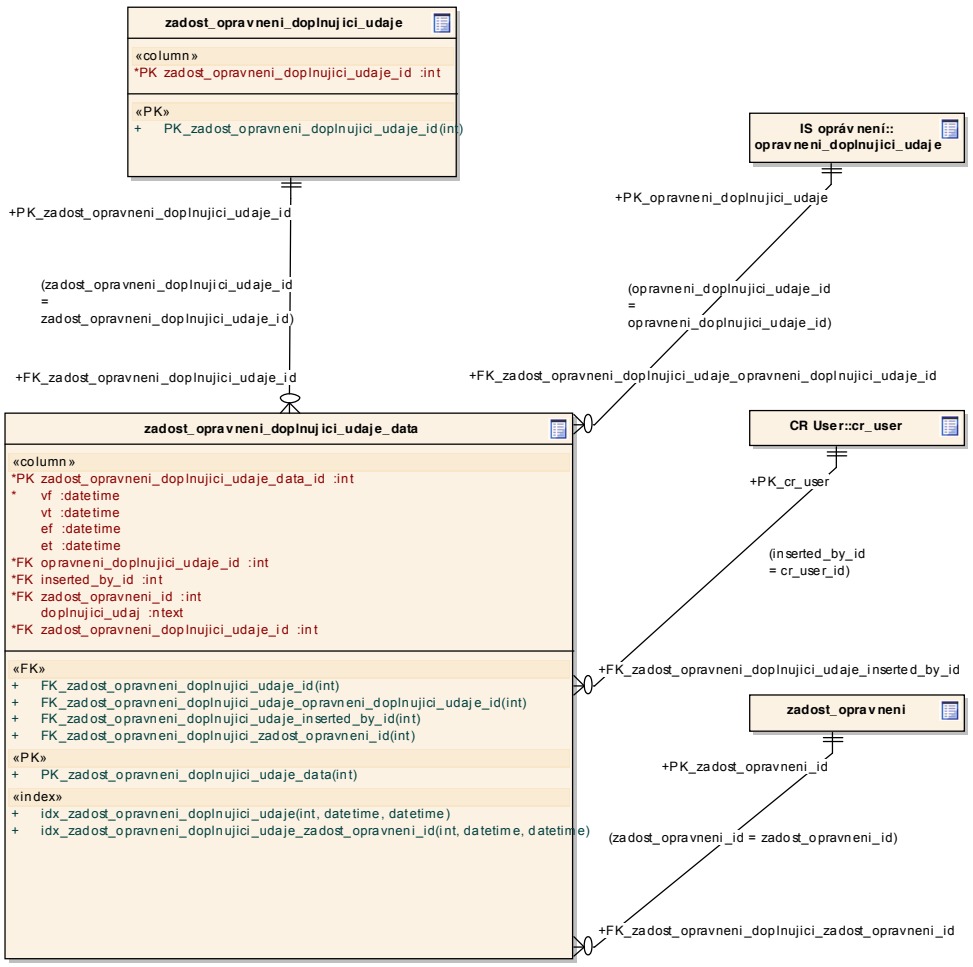


Obr. 2.13: zadost_opravneni

2.3.2.6 Uloženie oprávnenia

Tabuľka 2.13 `zadost_opravneni_data` obsahuje referenciu na tabuľku `zadost` a tabuľku `opravneni`. Takýto objekt vznikne označením oprávnenia vo formulári žiadosti. Tabuľka okrem štandardných položiek obsahuje položku `zduvodnni` a `zadost_opravneni_schvaleno`. Hodnoty oboch položiek budú závisieť na položkách tabuľky `opravneni_data`. V prípade, že má dané oprávnenie nastavenú položku `vyzaduje_zduvodeni` na hodnotu „T“, vo formulári sa zobrazí vstupné pole pre vloženie zdôvodnenia, ktoré sa uloží do tabuľky `zadost_opravneni_data`. Systém nastaví hodnotu parametru

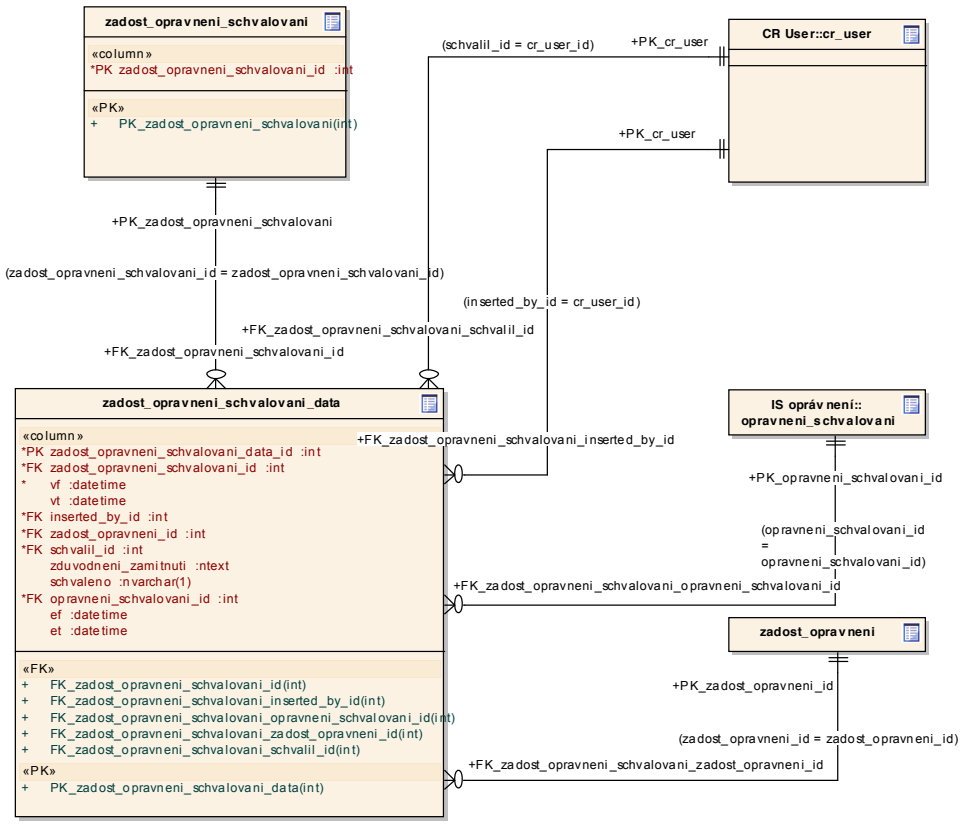
zadost_opraveni_schvaleno na „T“, až keď počet schválení dosiahne číslo nastavené v tabuľke opravneni_data ako minimalni_pocet_schvaleni.



Obr. 2.14: zadost_opravneni_doplnujici_udaje

2.3.2.7 Uloženie doplňujúcich údajov oprávnenia

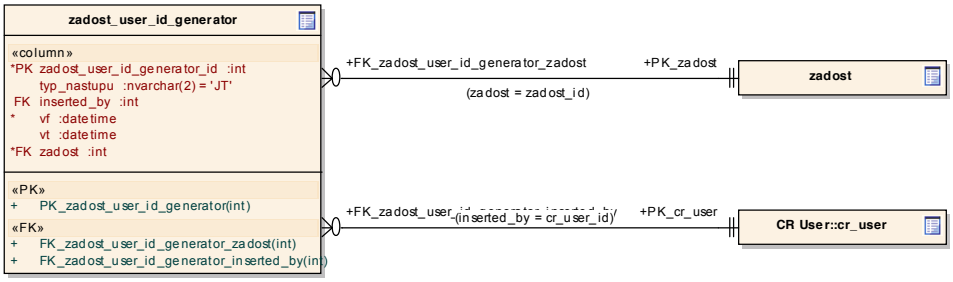
Tabuľka 2.14 zadost_opravneni_doplnujici_udaje_data referencuje tabuľky zadost_opravneni a opravneni_doplnujici_udaje. Rovnako ako v predchádzajúcom príklade sa jedná o spôsob uloženia dát. Teraz, sa konkrétne jedná o dáta, požadované tabuľkou opravneni_doplnujici_udaje_data. Ak oprávnenie „Inštalácia neštandardného softwaru“ pomocou tabuľky opravneni_doplnujici_udaje_data požaduje „odkaz na neštandardný software“, do parametru doplnujici_udaj v tabuľke zadost_opravneni_doplnujici_udaje_data sa uloží požadovaný odkaz.



Obr. 2.15: zadost_opravneni_schvalovani

2.3.2.8 Uloženie schvaľovania oprávnenia

Tabuľka 2.15 `zadost_opravneni_schvalovani_data` je odpoveďou na tabuľku `opravneni_schvalovani_data` a teda ukladá rozhodnutia schvaľovateľov oprávnení, poprípade ukladá zdôvodnenie zamietnutia. Preto referencuje tabuľky `zadost_opravneni` a `opravneni_schvalovani`.



Obr. 2.16: zadost_user_id_generator

2.3.2.9 Generovanie identifikačných čísel

Jedná sa o neverziovanú tabuľku 2.16 `zadost_user_id_generator`, ktorej primárny kľúč slúži pre generovanie unikátnych identifikačných čísel nových zamestnancov.

2.3.2.10 Využitie väzieb pri vyťahovaní z databázy

Vďaka takto zadefinovanej štruktúre bude možné pri inicializácii vstupných dát prázdneho formulára vytiahnuť všetky potrebné dáta naraz s objektom `IsTyp`. Pre doťahovanie vnorených objektov bude využité hibernate criterium include, ktoré na pozadí vykonáva klasický `SQL LEFT OUTER JOIN`. z databázy sa dotiahne:

1. kolekcia objektov `IsTyp`, tento objekt obsahuje potrebné kolekcie `IsTypData` a `InformacniSystemData`
   a) k objektom `InformacniSystemData` je potreba ich Master objekty `InformacniSystem`
2. `InformacniSystem` obsahuje potrebné kolekcie `InformacniSystemData` a `OpravneniData`
   a) k objektom `OpravneniData` je potreba ich Master objekty `Opravneni`
3. `Opravneni` obsahuje potrebné kolekcie `OpravneniData`, `OpravneniSchvalovaniData` a `OpravneniDuplnujiciUdajeData`
   a) k objektom `OpravneniDuplnujiciUdajeData` je potreba ich Master objekty `OpravneniDuplnujiciUdaje`
   b) k objektom `OpravneniSchvalovaniData` je potreba ich Master objekty `OpravneniSchvalovani`

V prípade zobrazenia stránky obsahujúcej už existujúci formulár sa navyše z databázy dotiahnu ďalšie potrebné dáta pomocou objektu Zadost. z databázy sa dotiahne:

1. objekt `Zadost` podľa parametru `zadostId`, tento objekt obsahuje potrebné kolekcie `ZadostData` a `ZadostOpravneniData`
   a) k objektom `ZadostOpravneniData` je potreba ich Master objekty `ZadostOpravneni`
2. `ZadostOpravneni` obsahuje potrebné kolekcie `ZadostOpravneniData`, `ZadostOpravneniDoplnujiciUdajeData` a `ZadostOpravneniSchvalovaniData`
   a) k objektom `ZadostOpravneniDoplnujiciUdajeData` je potreba ich Master objekty `ZadostOpravneniDoplnujiciUdaje`
   b) k objektom `ZadostOpravneniSchvalovaniData` je potreba ich Master objekty `ZadostOpravneniSchvalovani`

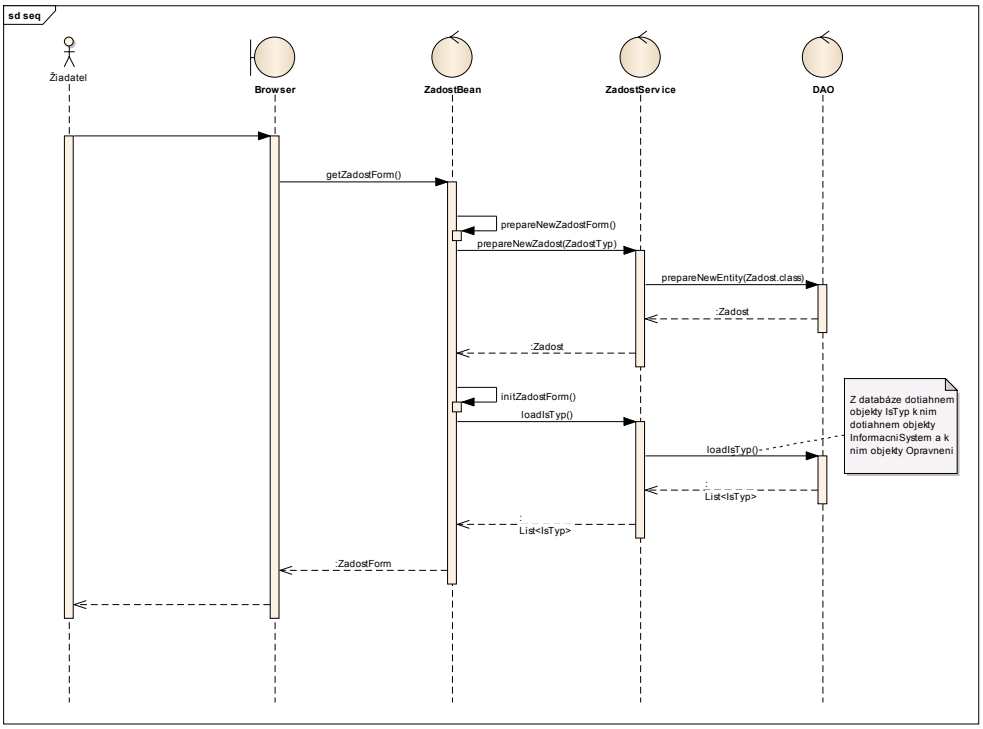
Pri žiadosti typu „Čaká na schválenie“, bude mať nadriadený možnosť požiadať o viac alebo menej oprávnení ako je vo formulári už vybraté, poprípade pozmeniť doplňujúce údaje. Preto sa pri takomto prípade dotiahnu všetky dáta s objektom `IsTyp` a všetky dáta s objektom `Zadost`. Tieto dáta sa následne spárujú tak, aby nadriadenému vo formulári zobrazili vybraté oprávnenia, spolu s doplňujúcimi údajmi a schvaľovateľmi.

2.4 Návrh komunikácie

V kapitole sú uvedené sekvenčné diagramy reprezentujúce návrh komunikácie medzi jednotlivými vrstvami systému. Vrstvy Browser a ZadostBean je v takto navrhnutom systéme možné označiť za Frontend. Triedy ZadostService a DAO zase reprezentujú Backend. Trieda ZadostService implementuje rozhranie IZadostService, toto rozhranie obsahuje všetky verejné metódy, ku ktorým trieda ZadostBean pristupuje. V prípade potreby prechodu na inú technológiu, ako JSF, by bolo možné pristupovať k týmto metódam napríklad z rozhrania webovej služby, tým pádom by Frontend mohol byť napísaný čisto v HTML a JavaScripte.

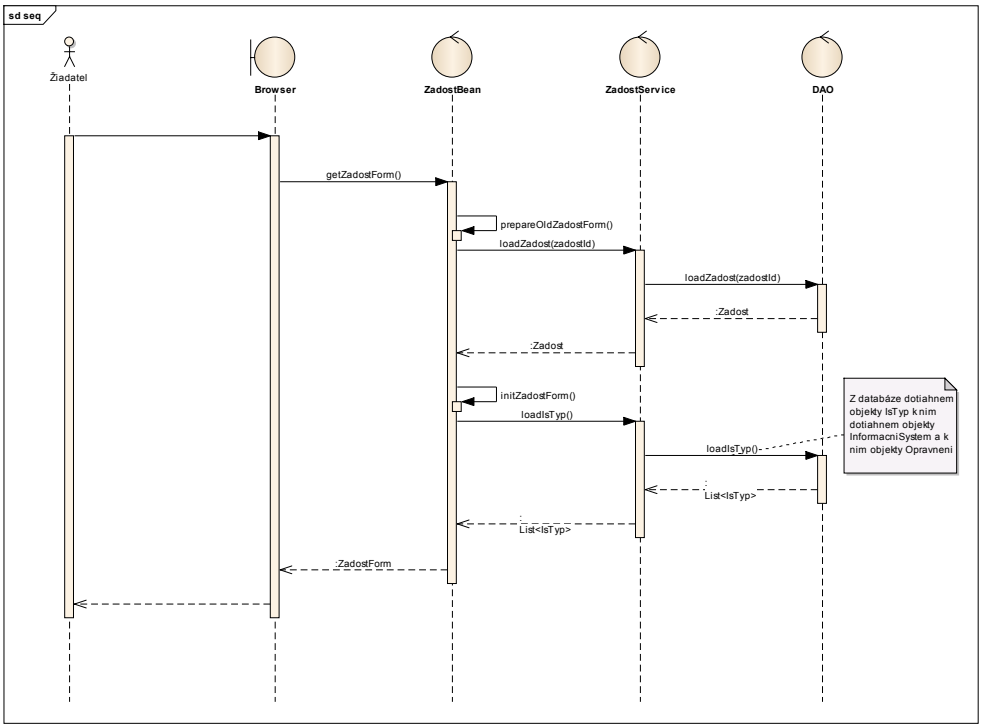
2.4.1 Zobrazenie formuláru pre novú žiadosť

V sekvenčnom diagrame 2.17 reprezentuje vrstva Browser `.xhtml` stránky obsahujúce HTML a JavaScript vygenerované z použitých JSF komponent. Táto vrstva pristupuje k JSF Managed Bean s názvom ZadostBean. Pre dosiahnutie bezstavovej architektúry používa ZadostBean request scope 2.1.2.3. Formulár bude obsahovať výber z množsta dát skladajúcich sa z rôznych objektov, a preto je jednoduchšie si takýto formulár poskladať už na úrovni



Obr. 2.17: Sekvenčný diagram popisujúci zobrazenie formuláru pre novú žiadosť

ManagedBeany a preniesť ho na stránku ako jeden objekt. Stránka v prehliadači požaduje obsah premennej s názvom `zadostForm`, ku ktorej pristupuje pomocou get metódy `getZadostForm()`. Táto premenná reprezentuje objekt `ZadostForm`, nejedná sa o objekt s väzbou k databáze, ide len o pomocný objekt poskladaný z iných pomocných a databázových objektov, ktorý reprezentuje všetky dáta nachádzajúce sa vo formulári. Kvôli tomu, že sa jedná o Managed Beanu v request scope, ktorej platnosť vykonaním HTTP odpovede skončí, je možné použiť lazy inicializáciu. To znamená, že ak premenná nie je inicializovaná, prebehne proces inicializácie, a ak inicializovaná je proces inicializácie znovu neprebehne a get metóda vráti rovno hodnotu premennej. Inicializácia začína volaním metódy `prepareNewZadostForm()`, v tejto metóde sa volá metóda `prepareNewZadost(zadostTyp)`, ktorej parametrom je inštancia objektu ZadostTyp. o aký typ žiadosti ide, rozpozná Managed Beana podľa Enum objektu `ZadostTypKategorie`, ktorý sa nastaví akonáhle užívateľ klikne na nejakú zo stránok žiadostí. Metóda sa nachádza v triede `ZadostService`, ide o Spring Beanu, `ZadostBean` môže k beane `ZadostService` pristupovať kvôli IOC za používať Setter Injection 2.1.4. `ZadostService` je servisná vrstva a je na nej implementovaná všetka business logika sys-



Obr. 2.18: Sekvenčný diagram popisujúci zobrazenie formuláru pre existujúcu žiadosť

tému, táto vrstva môže pristupovať k databáze, avšak komplikovanejšie dotazy sú presunuté na `DAO` vrstvu, ktorá slúži ako databázová vrstva. Následne je volaná metóda `prepareNewEntity(Zadost.class)` na triede `DAO`, je to taktiež Spring Bean zavedená do `ZadostService` pomocou Setter Injection 2.1.4. Metóda vracia inicializovanú entitu `Zadost`, ktorá má nastavené všetky potrebné parametre. Pokračuje sa volaním metódy `initZadostForm()`, ktorá najprv dotiahne `List<IsTyp>`. Jedná sa o kolekciu, v ktorej sa ku každému objektu `IsTyp`, dotiahne kolekcia objektov `InformacniSystem` a k nim kolekcia objektov `Opravneni`. Objekt `Oprávnení` môže obsahovať kolekciu objektov `OpravneniSchvalovani` a `OpravneniDoplnujiciUdaje`.Táto štruktúra bude v objekte `ZadostForm` reprezentovaná stromom, pomocou ktorého bude vo formulári možné rozkliknúť jednotlivé informčné systémy a dostávať sa k oprávneniam ktoré poskytujú. Po tomto kroku je objekt `ZadostForm` inicializovaný a server ho vracia klientovi. Pomocou tohoto objektu dokáže prehliadač zobraziť všetky potrebné dáta pre zobrazenie formulára klientovi.

2.4.2 Zobrazenie formuláru pre existujúcu žiadosť

V sekvenčnom diagrame 2.18 prebieha komunikácia rovnako ako v prípade novej žiadosti, menia sa iba volané metódy. Id existujúcej žiadosti je podané ako URL parameter a .xhtml stránka obsahuje komponent, ktorý toto Id prevezme a predá Managed Beane. Tým sa nastaví parameter `zadostId`, na základe toho je volaná metóda `prepareOldZadostForm()`. Následne je volaná metóda `loadZadost(zadostId)`, ktorá z databázy dotiahne objekt `Zadost` spolu s kolekciou objektov `ZadostOpravneni`. V metóde `initZadostForm()` sa kolekcia objektov `ZadostOpravneni` naparuje na príslušné objekty `Opravneni` a tým pádom sa oprávnenia, o ktoré už bolo žiadané, vo formulári zobrazia so zaškrtnutým chcekboxom.

2.5 Návrh grafického užívateľského rozhrania

Grafické užívateľské rozhranie (GUI) slúži k prezentácii informácií pomocou vizuálnych prvkov. Táto forma prezentovania informácií uľahčuje užívateľovi prácu so systémom. Medzi často využívané vizuálne prvky patria rôzne tlačítka, kolonky určené k vloženiu informácií a mnoho iných jednoduchých, ale aj zložitejších prvkov. Návrh GUI má byť na toľko intuitívny, aby užívateľovi umožnil rýchle pochopenie funkcií, ktoré od aplikácie požaduje. So správnym návrhom GUI by mal užívateľ dokázať využiť funkcie aplikácie bez akýchkoľvek dodatočných návodov alebo memorovania priechodu jednotlivých scenárov používania. Ako dokáže správny návrh GUI aplikácii pridať na kvalite, tak dokáže nesprávny návrh GUI aplikácii uškodiť. Neprehľadný GUI užívateľovi často neodhalí všetku funkcionalitu systému.

2.5.1 Využitie Nielsenovej heuristiky pri návrhu

Pri návrhu GUI je postupované podľa pravidiel Neilsenovej heuristiky [14].

2.5.1.1 Viditeľnosť stavu systému

Pri každom úkone užívateľ vidí, kde sa nachádza. Názvy jednotlivých stránok sú viditeľné v menu, ktoré sa nachádza na každej stránke.

2.5.1.2 Systém zodpovedá reálnemu svetu

Všetky ovládacie prvky aplikácie sa zobrazujú v logickom poradí. Rozmiestnenie dôležitých prvkov je zobrazené v hornej alebo ľavej časti stránky.

2.5.1.3 Užívateľská sloboda

Systém vždy umožňuje návrat na predchádzajúcu stránku. Návrat na hlavnú stránku je možný z akejkoľvek stránky prostredníctvom menu v hornej časti

stránky.

2.5.1.4 Konzistencia a štandardy

Názov stránky odkazuje na hlavnú stránku.
V systéme sa nenachádzajú rôzne veci, ktoré by boli rovnako pomenované.

2.5.1.5 Prevencia chýb

Návrh systému predchádza vzniku chýb. Pri zadávaní dát systém využíva výber z pred pripravených možností. V prípade, že užívateľ potrebné políčko zabudne vyplniť, systém ho upozorní prostredníctvom validácie.

2.5.1.6 Ľahšie rozpoznať ako spomínať

Nie je potrebné pamätať si špecifické postupy. Systém je intuitívny a všetky dôležité informácie má užívateľ k dispozícii na jednej stránke.

2.5.1.7 Flexibilita a efektivita použitia

Systém je jednoduchý a prehľadný, nie sú v ňom navrhnuté žiadne klávesové skratky.

2.5.1.8 Estetický a minimalistický návrh

Systém nezaťažuje užívateľa zbytočnými informáciami. Estetický a jednoduchý design je riešený pomocou zaužívaných elementov.

2.5.1.9 Pomoc užívateľom rozpoznať a napraviť chyby

Systém je navrhnutý tak, aby minimalizoval chyby, ktoré môže užívateľ spraviť. Chybové hlášky sú navrhnuté jednoducho a zrozumiteľne.

2.5.1.10 Nápoveda a dokumentácia

Pri zadávaní informácií do systému užívateľ väčšinou vyberá zo zoznamu, v tomto prípade mu napovedá samotný zoznam možností. Alebo zadáva informáciu do vopred pripraveného políčka, kde mu tak tiež napovedá zoznam možností formou okna, ktoré sa po kliknutí na dané políčko zobrazí.

2.5.2 Wireframe

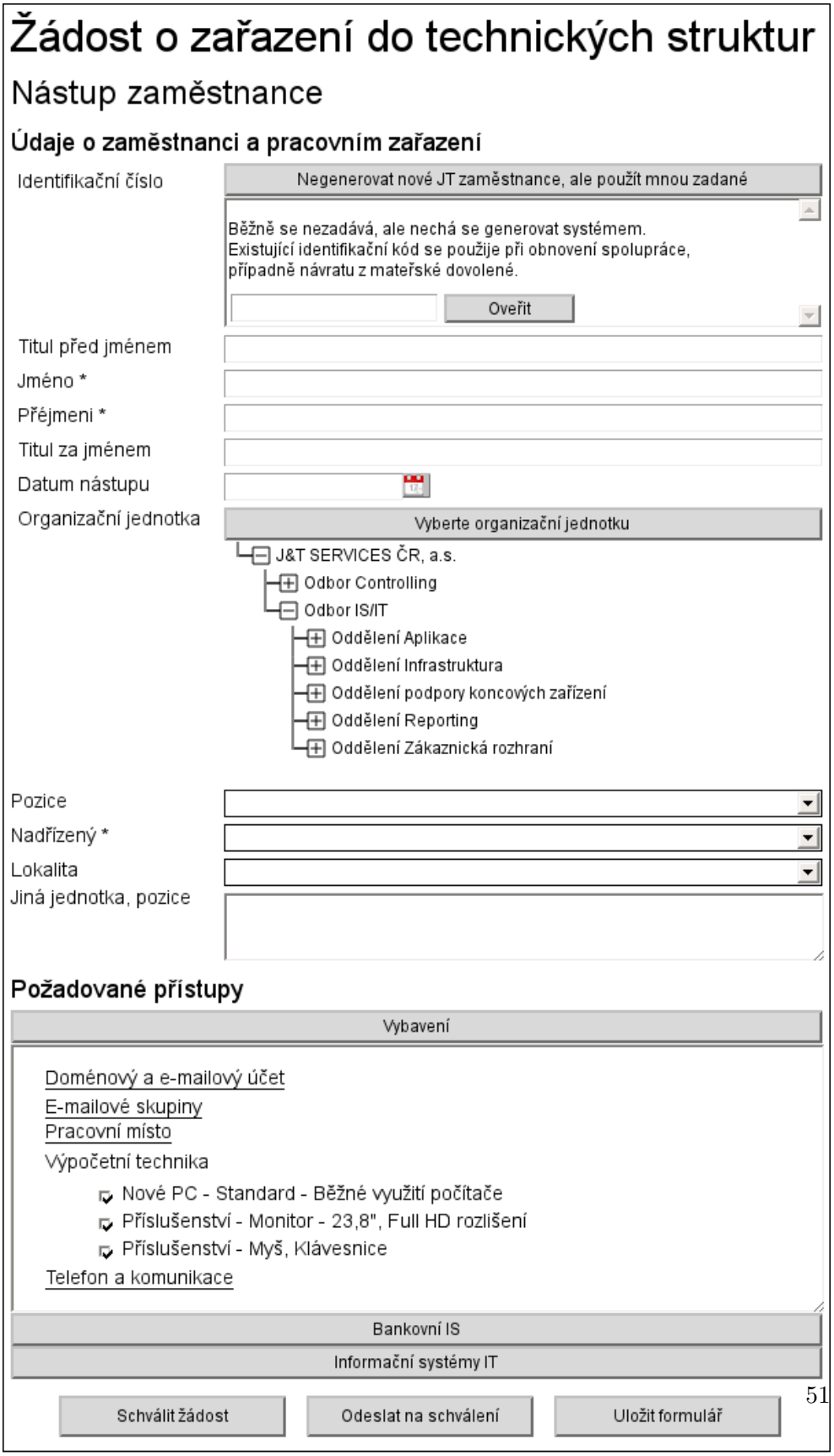
Drôtené modely, anglicky wireframes, sú dôležitým krokom návrhu grafického užívateľského rozhrania. Sú potrebným prostriedkom k navrhnutiu informačnej hierarchie. Teda rozvrhnutia spôsobu, akým bude užívateľ spracúvať informácie. Nejedná sa o úplne detailný návrh stránky. Cieľom je ukázať, ktoré

operácie a informácie budú dostupné na vybranej stránke. Drôtené modely sa navrhujú v rôznych nástrojoch prostredníctvom širokej škály grafických prvkov.

2.5.3 Proces návrhu

Pri návrhu užívateľského rozhrania sa vychádza z funkčných požiadavkov na systém, ktoré sú detailne rozpísané v sekcii 1.2.1.1 Požiadavky na systém. Pre ukážku návrhu užívateľského rozhrania pomocou wireframe je použitý funkčný požiadavok na vytvorenie interaktívnych formulárov, konkrétne formuláru „Nástup zamestnanca“. Proces začína rozborom konkrétneho funkčného požiadavku. Výstupom je zoznam operácií, ktoré má navrhovaná stránka užívateľovi umožniť, a logické poradie, v akom sa operácie majú vykonávať. Wireframe pre formulár reprezentujúci „Nástup zamestnanca“ v sebe musí zahrňať nasledujúce operácie:

- možnosť použiť, už existujúce identifikačné číslo
- overiť, či sa naozaj jedná o existujúce identifikačné číslo
- vloženie mena, priezviska a titulov nastupujúceho zamestnanca pomocou textových polí
- výber dátumu nástupu a teda aj dátumu, od kedy majú schválené údaje a oprávnenia nadobudnúť platnosť
- výber oddelenia, na ktorom daný zamestnanec pracuje
- výber pozície, na ktorej bude zamestnanec pracovať
- výber pobočky, na ktorej sa zamestnanec nachádza. Vo formulári sa nazýva lokalita
- výber nadriadeného zo zoznamu užívateľov
- možnosť zadať oddelenie alebo pozíciu, ktorá sa v systéme ešte nenachádza, pomocou textového pola
- možnosť vyberať oprávnenia zo zoznamu deleného podľa systémov a typov systémov
- schváliť žiadosť
- zamietnuť žiadosť
- odoslať žiadosť na doplnenie a schválenie nadriadeným
- uložiť žiadosť



Obr. 2.19: Wireframe reprezentujúci nástupný formulár

Kapitola 3

Implementácia

Kapitola je rozdelená podľa jednotlivých vrstiev systému. Pri každej vrstve je stručne opísané jej využitie a funkcia v systéme. Ďalej táto kapitola špecifikuje jednotlivé problémy vyplývajúce z požiadavkov na systém a stručne popisuje spôsob, akým boli tieto problémy vyriešené.

3.1 Konfigurácia projektu

3.1.1 Generovanie business objektov

Prvým krokom implementácie je vytvorenie tabuliek, kľúčov, a indexov databázy pomocou vygenerovaného SQL, ktorý odpovedá dátovému modelu. Systém bude mať k dispozícii tri rôzne databázy: `intranet_idm_dev`, `intranet_idm_prod` a `intranet_idm_test`. Pôjde o vývojovú, produkčnú a testovaciu databázu. Ďalším krokom je vytvorenie Maven projektu z archetypu. Aby bolo možné vo vytvorenom projekte vygenerovať `.hbm.xml` a `POJO` objekty podľa databázy, je potreba zadefinovať pripojenie pomocou súborov `hibernate.cfg.xml` a `hibernate.properties`. Pre konfiguráciu generovania `.hbm.xml` a `POJO` je použitý súbor `hibernate.reveng.xml`, v tomto súbore je možné napríklad zadefinovať mapovanie Microsoft SQL Server dátových typov na Java dátové typy alebo vopred zadefinovať, aby vygenerovaný objekt dedil alebo implementoval ľubovolný objekt. Každá databáza má v projekte vlastnú trojicu takýchto súborov. V súbore pom.xml je pre každú databázu vytvorený profil, ktorý obsahuje cestu ku konkrétnym súborom. Ďalšie profily sú vytvorené aj pre jednotlivé pluginy alebo skupiny pluginov, ktoré reprezentujú rovnakú funkciu. Pre vygenerovanie `hbm.xml` a `POJO` je použitý profil genModel, ktorý obsahuje plugin `maven-antrun-plugin` [15] a knižnice, ktoré potrebuje fungovaniu prostredníctvom „maven dependencies“[2]. Použitím príkazu `mvn generate-sources -PgenModel,dev`, maven použije schéma vývojo-

[2]https://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html

vej databázy, podľa ktorého vygeneruje `.hbm.xml` a `POJO`. Ak by sa požadoval aj build projektu, nahradí sa fáza generate-sources fázou install. Pri použití príkazu `mvn install` dôjde k buildu projektu a vytvoreniu war súboru, je to tým, že fáze install predchádzajú fáze compile a package. Jedná sa o fáze životného cyklu maven.

3.1.2 Pristupovanie k jednotlivým vrstvám

Vrstvy pristupujú ku verejným metódam iných vrstiev pomocou modelu Inversion of Control 2.1.4. Vrstvy systému sú triedy, ktoré boli pomocou konfiguračného súboru Springu `spring-application.xml` deklarované za Spring beany. Použitie Setter Injection vyžaduje implementáciu getter a setter metód. Tým pádom bude každá vrstva, ktorá pristupuje ku iným vrstvám, obsahovať množstvo týchto metód. Zjednodušením je vytvorenie ďalšej vrstvy, ktorá pristupuje ku všetkým potrebným vrstvám a použitie tejto vrstvy ako kontajneru, ktorý umožňuje prístup ku potrebným vrstvám. V systéme je táto vrstva pomenovaná `FormIdmFacadeImpl` . Ukážka deklarácie triedy `FormIdmFacadeImpl` za Spring Beanu a pridanie parametrov iných Spring Bean pomocou IOC:

```
<bean id="formIdmFacade"
   class="com.jtfg.itservice.core.FormIdmFacadeImpl">
        <property name="coreFacade" ref="coreFacade" />
        <property name="formIdmPersisterService"
           ref="formIdmPersisterService" />
        <property name="zadostService"
           ref="zadostService" />
</bean>
```

V prípade JSF Managed Bean táto funkcionalita použitá nie je, to je dané tým, že pre deklaráciu Managed Bean sú použité anotácie a nie deklarácia cez konfiguračný súbor `faces-config.xml`. Ak by boli deklarované cez konfiguračný súbor, bolo by im možné nastaviť triedu `FormIdmFacadeImpl` ako parameter a fungovalo by to rovnako ako pri Spring Beanach. Dôvodom pre použitie anotácií namiesto deklarácie v konfiguračnom súbore je to, že počet Managed Bean môže v takomto systéme výrazne prevyšovať počet Spring Bean a tým pádom by konfiguračný súbor `faces-config.xml` obsahoval zbytočne veľa záznamov. Managed Beany teda dedia od abstraktnej triedy `AbstractFormIdmManagedBean`, táto trieda nie je JSF Managed Beana, ktorá implementuje metódu getFormIdmFacade(), ktorá pomocou názvu triedy a knižníc `FacesContext` a `FacesContextUtils` získa inštanciu triedy `FormIdmFacadeImpl`.

```
public IFormIdmFacade getFormIdmFacade() {
   if (formIdmFacade == null) {
       FacesContext currentInstance =
          FacesContext.getCurrentInstance();
```

```
        WebApplicationContext webApplicationContext =
           FacesContextUtils
        .getWebApplicationContext(currentInstance);
         formIdmFacade = (IFormIdmFacade)
         webApplicationContext.getBean("formIdmFacade");
        }
         return formIdmFacade;
}
```

3.2 Dátová vrstva

Dátová vrstva v tomto systéme slúži pre vyťahovanie dát z databázy, sú na nej implementované iba komplikovanejšie dotazy. Jednoduché vyťahovanie alebo ukladanie dát podľa niekoľko málo parametrov sa v systéme vykonáva už na servisnej vrstve. Jedná sa o Spring Beanu so štandardnou konfiguráciou, z toho vyplýva, že pôjde o singleton triedu. Pri takomto type triedy je potrebné dbať na to, aby trieda neobsahovala žiadne globálne premenné, pretože takéto premenné by držali rovnakú hodnotu pre všetky inštancie aplikácie. Trieda teda obsahuje iba metódy.

Jedným z problémov vychádzajúceho z požiadavkov bolo vymyslieť spôsob, akým užívateľom zobraziť iba tie žiadosti, ktoré sa ich konkrétne týkajú, ale s výnimkou vlastného výstupu. Zároveň bolo potrebné umožniť zamestnancom z HR a administrátorom systému vidieť všetky žiadosti. Systém umožňuje zobrazenie žiadostí prostredníctvom stránky „Zoznam žiadostí“ a následne umožňuje rozkliknúť detail žiadosti, na základe čoho sa otvori stránka žiadosti s id žiadosti ako parametrom URL. Je potrebné, aby sa v oboch týchto spôsoboch zobrazenia žiadosti zobrazili iba žiadosti, ktoré sa zobraziť majú. Univerzálne riešenie takéhoto problému, je rozšírenie hibernate criteria o volanie vlastnej funkcie, ktorá bude implementovať rôzne pravidlá pre doťahovanie rôznych objektov. Tým pádom sa problém vyrieši už na úrovni databázy a zároveň bude riešenie rozšíriteľné aj pre iné typy objektov.

```
@Override
public void modify(ExtendedDetachedCriteria criteria) {
        ISecurityController securityController =
        SecurityManagerUtil.getSecurityManager().
        getSecurityController();
        Class<?> entityClass =
           criteria.getEntityClass();
        if (Zadost.class.isAssignableFrom(entityClass))
           {
                boolean canViewAllZadost =
                   securityController
                .hasPermission("CAN_VIEW_ALL_ZADOST");
```

```
                if (canViewAllZadost) {
                        return;
                }
                modifyZadostCriteria(criteria);
        }
}
```

Pri akomkoľvek volaní dotazu do databázy pomocou hibernate criterií sa zavolá metóda public `modify(criteria)`. Metóda z parametru získa objekt entityClass, ktorý hibernate criteria popisujú. Zistí, či ide o objekt `Zadost` pomocou `isAssignableFrom()`. Ak áno, najprv sa vyrieši prípad, ak ide o pracovníka HR s rolou ROLE_HR_EMPLOYEE, alebo administrátora v roli ROLE_ADMIN, pre ktorých nie je potreba criteria modifikovať. Tento prípad je riešený pomocou oprávnenia `CAN_VIEW_ALL_ZADOST`, ktoré majú iba tieto role. V opačnom prípade sa volá metóda, ktorá criteria pre vyťahovanie objektu alebo objektov typu žiadosť zmodifikuje. Pre modifikáciu kritérii je použitý nasledujúci postup:

```
Set<String> aliases = criteria.getAliases();
```

Množina aliases obsahuje všetky zavedené aliasy z objektu criteria. Jednotlivé aliasy budú potrebné pri určovaní, ktoré žiadosti sa pre súčastného užívateľa môžu dotiahnuť z databázy.

```
// nadrizenyData
boolean containsNadrizenyData =
   aliases.contains("nadrizenyData");
if (!containsNadrizenyData) {
        criteria.includeValid("nadrizeny.subjektDatas",
           "nadrizenyData", true);
}
Criterion nadrizeny =
   Restrictions.eq("nadrizenyData.userid", userid);
```

Najprv je potrebné určiť, či kritérium obsahuje potrebný alias. Ak nie, alias sa zavedie. V tejto chvíli je možné s istotou povedať, že potrebný alias sa naozaj nachádza v objekte `criteria`, a to práve raz. Obmedzenie sa vytvorí podľa userid užívateľa, toto obmedzenie hovorí, že sa žiadosť dotiahne, iba ak aktuálne prihlásený užívateľ v žiadosti figuruje ako nadriadený.

```
Criterion subjekVystup = Restrictions.and(subjekt,
   vystup);
criteria.add(Restrictions.not(subjekVystup));
```

Vytvorí sa nové obmedzenie, ktoré vyberá iba také žiadosti, v ktorých aktuálne prihlásený užívateľ figuruje ako subjekt a zároveň sa jedná o žiadosti typu výstup. Ku existujúcemu `criteria` sa pridá negácia tohoto obmedzenia.

```
criteria.add(Restrictions.or(subjekt, nadrizeny,
   schvalil, insertedBy, schvalovatelOpravneni));
```

Nakoniec sa pridá obmedzenie, ktoré hovorí, že sa majú dotiahnuť také žiadosti, kde aktuálne prihlásený užívateľ figuruje ako subjekt, nadriadený, ten kto schválil žiadosť, ten kto ju vložil alebo schvaľovateľ nejakého z oprávnení.

3.3 Servisná vrstva

Táto vrstva umožňuje pristupovať ku databáze a je na nej implementovaná všetka business logika aplikácie s výnimkou logiky, ktorá sa vyslovene týka zobrazenia dát. Rovnako, ako pri dátovej vrstve, sa jedná o Spring Beanu so štandardnou konfiguráciou, takže platia rovnaké pravidlá ako pre dátovú vrstvu.

3.3.1 Notifikácie pomocou emailu a Service desku

3.3.1.1 Webová služba

Systém Service desk umožňuje pridávať jednotlivé úlohy prostredníctvom metód na to určených, ktoré poskytuje v rozhraní webovej služby. Service desk poskytuje WSDL, ktoré tieto triedy a metódy definuje. Pre vygenerovanie objektov z wsdl je použitý maven plugin `jaxws-maven-plugin` [16]. Ten je umiestnený v profile `genWsClient`, takže generovanie tried podľa WSDL prebehne iba po zavolaní príkazu `mvn generate-sources -P,dev,genWsClient`. Plugin ponúka možnosť vygenerovať objekty priamo z URL, na ktorom sa WSDL nachádza, druhou možnosťou je uložiť toto WSDL do súboru a na generovanie použiť tento súbor. Druhá varianta je spoľahlivejšia, pretože nie je isté či nenastane situácia, kedy odkaz s WSDL nebude fungovať.

3.3.1.2 Spôsob vyplnenia ticketu

Ticket do Service desku sa zadáva pomocou vygenerovaného objektu ApiServiceRequest. Tento objekt má množstvo rôznych parametrov, ktoré pre túto úlohu nie sú podstatné, je ich však potreba správne vyplniť. Preto sa v systéme nebude vytvárať nová inštancia tohoto objektu, ale nechá sa vytvoriť Service deskom pomocou na to určenej metódy. Tým pádom je pripravená inštancia objektu ApiServiceRequest, ktorej stačí nastaviť iba potrebné parametre. Primárne pôjde o parametre kategória, podkategória a riešiteľská skupina, tieto parametre sú uložené v objekte `InformacniSystem`.

3.3.1.3 Kedy notifikovať

Koho, kedy a ako má systém notifikovať je závislé na veľa parametroch, mať tieto údaje pevne uložené v kóde, by bolo neprehľadné. Preto vznikla nová

tabuľka v databáze s názvom `zadost_notifikace`, podľa dát z tejto tabuľky systém určuje, koho a ako má notifikovať. Tento objekt sa z databázy dotiahne na základe parametrov ako sú typ žiadosti, stav žiadosti, či schvaľovateľ oprávnenia rozhodol o oprávnení alebo či o oprávnení bolo rozhodnuté automaticky. z tohoto objektu systém zistí, či treba notifikovať subjekt, nadriadeného, schvaľovateľov oprávnení alebo Service desk. Ďalším problémom, ktorý pri riešení notifikácií nastal bolo, aby sa notifikácie neposielali viackrát ako je potrebné. Takže ďalším rozšírením bola implementácie podmienky, ktorá rozlišuje či žiadosť práve zmenila stav na schválenú. Rovnaké opatrenie bolo treba zaviesť aj pre jednotlivé oprávnenia, aby sa tickety do Service desku posielali iba raz, a to vtedy, keď objekt `ZadostOpravneni` zmení stav schvaľovania. Akonáhle je určený spôsob notifikácie, je treba určiť, koho notifikovať. V prípade vybraného oprávnenia, ktoré má svojich schvaľovateľov, systém po schválení žiadosti notifikuje týchto schvaľovateľov. Akonáhle je oprávnenie schválené, systém musí notifikovať osoby zodpovedné za prideľovanie oprávnenia. V prípade oprávnení typu vybavenie, sú tieto notifikácie závislé od pobočky, na ktorej subjekt pracuje. Preto existuje možnosť použiť pre určenie príjemcu notifikácie objekt `OpravneniNotifikce`, ktorý na základe oprávnenia a lokality určuje, koho notifikovať. Dáta tohoto objektu obsahujú parametre pre notifikáciu prostredníctvom Service desku, ale aj parameter pre zadanie emailových adries. Algoritmus pre určovanie notifikácií najprv skúsi vytiahnuť dáta objektu `OpravneniNotifikace`, ak ho nenajde, použije dáta z objektu `InformacniSystem`. ktorý obsahuje tie isté parametre. Je isté, že každý objekt `InformacniSytem` má tieto parametre vyplnené.

3.3.1.4 Aktuálne oprávnenia

Pri výstupe alebo prestupe je potreba obsah ticketu rozšíriť o súčasné oprávnenia vystupujúceho alebo prestupujúceho užívateľa, aby mu mohli byť odobrané. Výnimkou je, ak nadriadený pri vyplňovaní formuláru zaškrtne položku Formálny prestup, tým pádom prestupujúci užívateľ o súčastné oprávnenia nepríde. Tieto oprávnenia sa získavajú z databázy `intranet_service`, z tejto databázy je možné získať jedine názvy systémov, v ktorých užívateľ figuruje. Tým pádom nastáva problém, koho ohľadom týchto systémov notifikovať a je potrebné získané prístupy do informačných systémov spárovať so systémami v databáze systému `intranet_idm_prod`. Keďže databázy na sebe nie sú závislé, môže nastávať prípad, že sa informačné systémy v týchto databázach líšia. Prípad, kedy by databáza `intranet_idm_prod` neobsahovala nejaký zo systémov z `intranet_service` znamená, že daný informačný systém už nie je aktuálny a teda nie je možné získať informáciu o tom, ako riešiť notifikáciu, tým pádom sa ním program nebude zaoberať. Problémom je opačný prípad.

Riešenie tohoto problému je algoritmus, ktorý získa z databázy `intranet_service` dve množiny, prvá obsahuje všetky systémy nachádzajúce sa v tejto databáze a druhá iba systémy, v ktorých figuruje súčastný užíva-

teľ. Tieto dve množiny sa spoja do HashMap<String,Boolean>, táto mapa obsahuje všetky systémy a v prípade, že v nich figuruje súčastný užívateľ, vracia hodnotu true. Následne sa z databázy `intranet_idm_prod` vytiahne zoznam všetkých informačných systémov a prechádza sa po jednom. Každý informačný systém v tomto zozname sa použije ako kľúč v mape. V prípade, že mapa vráti true, informačný systém sa pridá do zoznamu, podľa ktorého sa bude notifikovať Service desk. V prípade, že mapa vráti false sa ide ďalej. Tento algoritmus je zavedený kvôli tomu, aby ošetril tretí prípad. Prípad, keď mapa vráti null, čo znamená, že o informačnom systéme neexistujú informácie v databáze `intranet_service`, a teda program nemá ako určiť, či užívateľ v tomto systéme figuruje alebo nie. Najbezpečnejšie riešenie je tento systém zaradiť do zoznamu, podľa, ktorého sa bude notifikovať Service desk, ako keby v tom systéme užívateľ figuroval.

3.3.2 Univerzálne REST rozhranie

Štandardne sa pre každý objekt implementuje samostatné trieda, ktorá obsahuje implementáciu všetkých možných HTTP metód [17]:

HTTP	**CRUD**	**Celá kolekcia (napr. /customers)**
POST	Create	201 (Created), vytvorí nový objekt.
GET	Read	200 (OK), Vráti zoznam objektov. Pri veľkých zoznamoch sa používa stránkovanie, radenie a filtrovanie.
PUT	Update /Replace	404 (Not Found)
PATCH	Update /Modify	404 (Not Found)
DELETE	Delete	404 (Not Found)

Tabuľka 3.1: HTTP metódy celá kolekcia

3.3.2.1 Definícia problémov

Štandardné riešenie by bolo pre poskytovanie všetkých aplikačných dát príliš pracné a neefektívne.

Prvým problémom, ktorý treba vyriešiť, je automatizácia implementácie takýchto rozhraní. Keďže je aplikácia implementovaná tak, aby sa POJO objekty generovali na základe tabuliek v databáze bez nutnosti písania kódu, bolo by dobré RESTové rozhrania tak isto generovať. A tým pádom sa aplikácia môže dynamicky rozširovať bez nutnosti písania kódu.

Druhým problémom je serializácia. Pri výbere prostriedku serializácie sa rozhodovalo medzi JSON a XML. Kvôli tomu, že RESTové rozhrania budú pravdepodobne použité klientom implementovaným v JavaScripte, bude ako

HTTP	CRUD	Jedna položka (napr. /customers/id)
POST	Create	404 (Not Found), 409 (Conflict) ak objekt už existuje.
GET	Read	200 (OK), Vráti konkrétny objekt. 404 (Not Found), ak neexistuje objekt s danýnm ID alebo ID nie je validné.
PUT	Update /Replace	200 (OK) or 204 (No Content). 404 (Not Found), ak neexistuje objekt s danýnm ID alebo ID nie je validné.
PATCH	Update /Modify	200 (OK) or 204 (No Content). 404 (Not Found), ak neexistuje objekt s danýnm ID alebo ID nie je validné.
DELETE	Delete	200 (OK). 404 (Not Found), ak neexistuje objekt s danýnm ID alebo ID nie je validné.

Tabuľka 3.2: HTTP metódy jedna položka

prostriedok serializácie použitý štandard, ktorý z JavaScriptu vychádza, zároveň sa označuje sa ako viac čitateľný, je ním JSON. Druhým problémom je spôsob serializácie. Objekty sú navrhnuté tak, že ich nie je možné priamočiaro serializovať použitím štandardných knižníc. Je to dané tým, že sú zložené z Master a Data objektu. Riešenie teda musí zahŕňať vytvorenie ďalších objektov, ktoré poslúžia ako kontajnery pre serilizáciu, klient príde do styku iba s týmito objektami.

3.3.2.2 Riešenie problémov

Na základe týchto problémov, je pre komunikáciu s aplikáciami tretích strán vystavené iba jedno rozhranie, ktoré obsahuje všetky HTTP metódy a dokáže spracovávať akýkoľvek objekt. Oba problémy sú riešené súčastne, implementovaním univerzálneho rozhrania, ktoré spracúva všetky objekty tým, že ich serializuje do formátu JSON rovno vo forme pomocných objektov slúžiacich na prenos dát. Takéto riešenie vôbec nepotrebuje generovanie pomocných POJO objektov, objekty vzniknú rovno vo formáte `JsonObject`. Všetky metódy rozhrania sú mapované na špecifickú cestu, ktorá sa líši názvom metódy a obsahuje parameter názov objektu, prípadne ďalší parameter id objektu. Názov objektu je použitý ako parameter metódy `Java.lang.Class.forName()`, ktorá vracia konkrétny Class objekt. Vďaka takémuto riešeniu odpadá potreba generovať samostatné RESTové rozhranie pre každý objekt.

```
// Read one
// Read multiple
@Post
@Produces({ MediaType.APPLICATION_JSON })
```

```
@Path("{objectName}")
public Response get(@PathParam("objectName") String
    objectName, @RequestBody String jsonStr)
```

Metóda get je implementovaná pomocou POST, je to kvôli tomu, aby bolo možné preberať parametre filtrov a sortov z @RequestBody a nie z parametrov URL, ktoré má obmedzenú dĺžku. Na požiadavok get je vždy poslaná odpoveď, ktorá obsahuje iba aktuálne dáta, keďže objekty sú zložené z Master objektu a kolekcie objektov Data, serializácia do JSON potrebuje špecifickú implementáciu. JSON objekty vzniknuté serializáciou v sebe obsahujú všetky parametre aktuálneho objektu Data okrem id objektu Data. Ako id vzniknutého JSON objektu je použité id objektu Master. Zobrazenie neverziovaných objektov je jednoduchšie, výsledné JSON objekty obsahujú všetky parametre pôvodných objektov. Pri serializácii vnorených objektov sa do JSON serializuje iba Master objekt, ktorý bude vo výsledku obsahovať iba položku id. Okrem id obsahuje Master kolekcie všetkých Data objektov, ktoré ho referencujú, takéto kolekcie serializované nebudú.

```
// Create
// Only versioned entity
@Post
@Consumes({ MediaType.APPLICATION_JSON })
@Path("post/{objectName}")
public Response post(@PathParam("objectName") String
    objectName, @RequestBody String jsonStr)
```

Metóda post v @RequestBody prijma JSON objekt vo formáte popísanom vyššie. Serializácia z JSON je tak isto špecificky implementovaná. Pripraví sa prázdna inštancia verziovaného objektu, to znamená Master objekt, ktorý obsahuje kolekciu objektov Data. JSON objekt prijatý metódou post sa pomocou knižnice Gson serializuje na Data objekt. Tento objekt sa vloží do kolekcie Data objektov a táto kolekcia sa nastaví Master objektu. Vytvorenému Data objektu sa nastaví posledný nenastavený parameter, a tým je Master objekt. Keďže ide o univerzálne rozhranie, objekt Master aj objekt Data je typu lang.pojo.Objekt a je potrebné vyvolať settre, ktoré potrebné parametre nastavia.V ukážke nižšie je použitý postup, ktorý vyvoláva setter na základe názvu nastavovanej položky pomocou Spring triedy `PropertyAccessorFactory`. Ide o prípad nastavenia parametru Master objekt, objektu Data, „objectName“ je názov objektu a „jsonStr“ je hodnota z parametru @RequestBody:

```
Class<?> classDataFromParam =
    Class.forName("com.jtfg.itservice.pojo." +
    objectName + "Data");
Gson gson = new GsonBuilder().create();
```

```
Object fromJsonObjectData = gson.fromJson(jsonStr,
    classDataFromParam);
PropertyAccessor myAccessorForObjectData =
    PropertyAccessorFactory
.forBeanPropertyAccess(fromJsonObjectData);
Object object = getPersistenceService()
.prepareNewEntity(classFromParam);
String loCaseName = objectName.substring(0, 1)
.toLowerCase() + objectName.substring(1);
myAccessorForObjectData.setPropertyValue(loCaseName,
    object);

// Put
// Update/Replace
// Only versioned entity
@PUT
@Consumes({ MediaType.APPLICATION_JSON })
@Path("put/{objectName}/{objectId}")
public Response put(@PathParam("objectName") String
    objectName, @PathParam("objectId") String id,
    @RequestBody String jsonStr)
```

Metóda put je iplementovaná podobne ako metóda post, s tým rozdielom, že sa nevytvára nový Master objekt ale vytihne sa z databázy podľa id v URL.

```
// Patch
// Update/Modify
// Only versioned entity
@Post
@Consumes({ MediaType.APPLICATION_JSON })
@Path("patch/{objectName}/{objectId}")
public Response patch(@PathParam("objectName") String
    objectName, @PathParam("objectId") String id,
    @RequestBody String jsonStr)
```

Metóda patch, vytiahne Master a Data objekty z databázy podľa id v URL. Vytvorí sa kópia aktuálneho Data objektu, a pôvodný sa znevalidní. Pomocou PropertyAccessorFactory sa novému data objektu nastaví položka vf na aktuálny dátum a insertedBy na užívateľa ktorý akciu inicializoval a všetky ďalšie položky, ktoré prišli v JSON objekte v @RequestBody.

```
// Delete
// Only versioned entity
@DELETE
@Produces({ MediaType.APPLICATION_JSON })
@Path("delete/{objectName}/{objectId}")
```

```
public Response delete(@PathParam("objectName") String
    objectName, @PathParam("objectId") String id)
```

Metóda delete sa správa úplne rovnako ako metóda patch s tým, že sa patchuje iba položka vt. Zavolá sa metóda `invalidateEntity`, ktorá slúži na invalidovanie verziovaného objektu. Táto metóda vytvorí kópiu aktuálneho data objektu, pôvodný invaliduje, novej verzii nastaví položku insertedBy a tiež ju invaliduje. Je teda možné vyhľadať, kto Data vytvoril a aj kto ich invalidoval.

3.3.2.3 Zhodnotenie

Systém poskytuje rozhranie, ktoré aplikácii umožňuje komunikáciu s aplikáciami tretích strán pomocou RESTových služieb. Táto funkcionalita bude primárne použitá pre poskytovaní dát iným interným aplikáciam. Ďalej by mohla byť použitá pri prechode na iný typ architektúry, ktorý by implementáciu business logiky preniesol na klienta. Implementácia týchto rozhraní bude spracovaná ako knižnica, ktorá sa bude používať aj v ďalších interných aplikáciách spoločnosti.

3.4 Prezenčná vrstva

Prezenčná vrstva sa skladá za JSF Managed Bean a z `.xhtml` stránok, ktoré ku týmto Beanam môžu pristupovať. Narozdiel od Spring Bean sa Managed Beany chovajú štandardne a umožňujú vytvárať inštancie. Tým pádom je v Managed Beanach možné využívať globálne premenné, a keďže sa k týmto premenným pristupuje často a ešte v každej JSF fáze, je dobré používať lazy inicializáciu. Príklad použitia lazy inicializácie objektu, prevzatý z [18]:

```
public final class MyFrame extends Frame
{
  private MessageBox mb_ ; //null, implicit
  //private helper used by this class
  private void showMessage(String message)
  {
    if(mb_==null)//first call to this method
      mb_=new MessageBox();
    //set the message text
    mb_.setMessage( message );
    mb_.pack();
    mb_.show();
  }
}
```

3.4.1 Bezstavovosť

Požiadavkom na systém bola bezstavovosť, pri práci s JSF sa bezstavovosť dosahuje nastavením scope na Request u konkrétnej Managed Beany. Každý request od klienta príde na server, kde nájde úplne novú Managed Beanu a všetky objekty sa inicializujú odznova [4]. Užívateľ má k dispozícii stránku, ktorá obsahuje rôzne komba a vstupné políčka pre vyplňovanie hodnôt, akonáhle užívateľ nejaké políčko vyplní, pri dalšom requeste na server sa zavolá setter danej položky a jej hodnota sa nastaví podľa hodnoty vo formulári. Takže jednotlivé komba a políčka nachádzajúce sa vo formulári slúžia ako úložisko zadaných hodnôt po celý čas práce s formulárom. Problém nastane akonáhle je potreba udržať údaje, ktoré sa do formuláru nezadávajú ručne. V prípade formuláru žiadosti ide o údaje z parametru URL, id žiadosti. Ďalej typ žiadosti, ktorý sa nastavuje v závislosti na tom, na akej stránke, sa užívateľ nachádza. A nakoniec dáta, ktoré sa doťahujú z databázy v závislosti na iných zadaných dátach. Pre každú takúto položku je potrebné vytvoriť si vstupné pole, ktoré vo formulári síce nebude viditeľné, ale bude udržovať požadovanú hodnotu. Takéto políčko potrebuje mať nastavené tri parametre. Prvým parametrom je `value`, mapuje vstupné políčko s premennou v Managed Beane. Druhým parametrom je `converter`, ten slúži na prevádzanie objektu na string a opačne, pod string hodnotou sa bude nachádzať id objektu. Tretím parametrom je `binding`, ktorý mapuje políčko na objekt typu UIInput nachádzajúci sa v Managed Beane, akonáhle je políčko mapované cez UIInput, dá sa ku hodnote dostať už vo fáze RESTORE_VIEW. Táto vlastnosť je zásadná pri získavaní dát potrebných pre inicializáciu objektov, ktorým sa následne nastavujú položky z formulára, ako napríklad id žiadosti. Najprv je potrebné dotiahnuť žiadosť z databázy podľa kľúča zadostId, až potom je možné nastavovať jednotlivé položky žiadosti pomocou formulára.

Pri hodnote z URL parametru je pomocné textové políčko nesmierne dôležité, pretože vykonaním metódy POST na stránke, URL automaticky stratí parametre, a keďže napríklad tlačítka typu <h:commandButton> sú implementované pomocou POST, je potrebné dáta z parametru uchovať ináč. Implementácia je nasledovná, getter pre daný parameter použije lazy inicializáciu, v ktorej sa najprv pokúsi získať hodnotu z parametru URL, ak sa akcia nepodarí, získa hodnotu z UIInput objektu. V prvom requeste sa hodnota nastaví podľa parametru, takže hodnota vstupného políčka sa nenastavuje cez formulár, ale cez server. V každom ďalšom requeste sa už parameter v URL nenachádza, ale hodnota je stále uložená v políčku, z ktorého sa ku nej dá dostať pomocou UIInput.

3.4.2 Užívateľské rozhranie

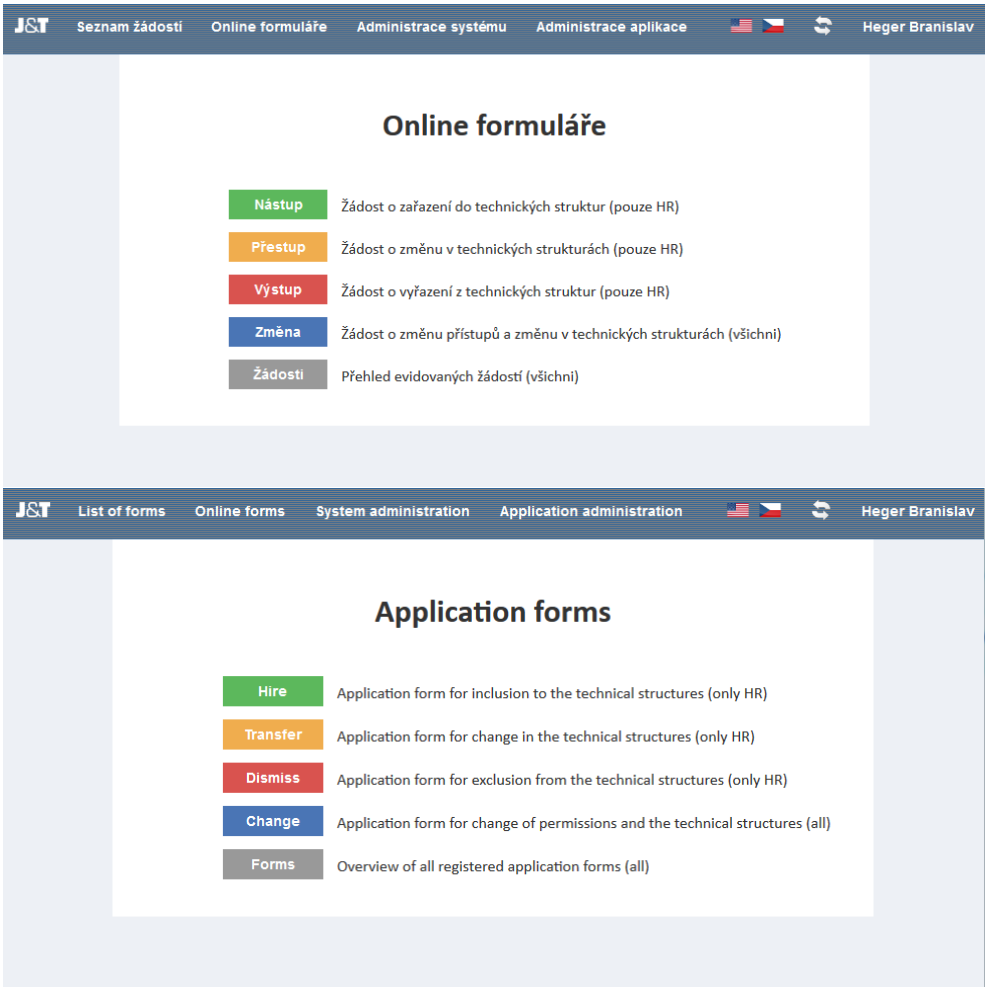
Hlavnou úlohou systému je evidovať aktivitu pomocou formulárov žiadostí, preto domovská stránka systému obsahuje rovno presmerovania na stránky

formulárov podľa typu 3.1. Tlačítka sú farebne odlíšené, čo prispieva k rýchlejšiemu výberu typu formulára. Pretože užívateľ, ktorý využíva systém často, má spojený typ formulára s farbou tlačítka.

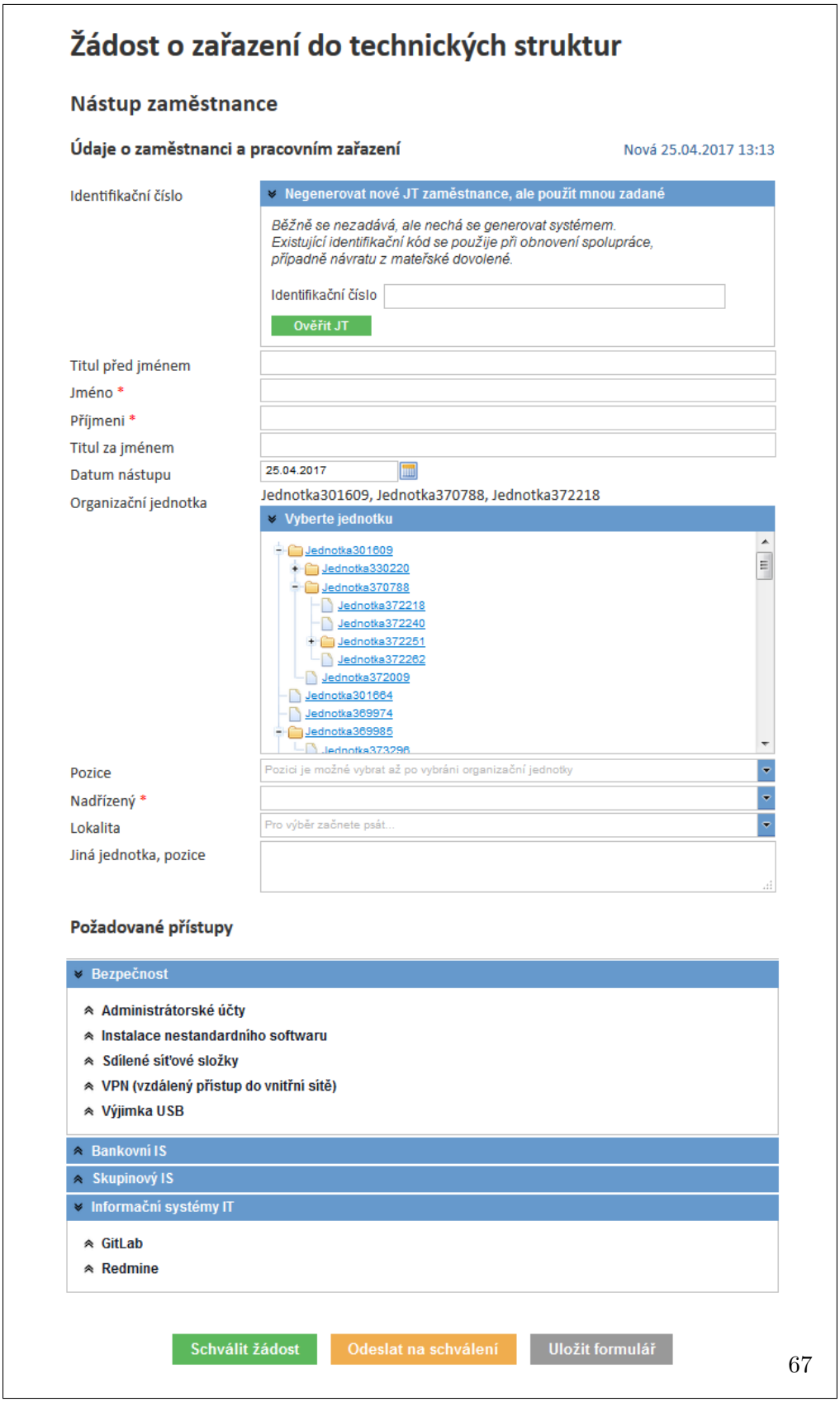
Systém je k dispozícii v českom a anglickom jazyku. Všetky texty, ktoré sa v aplikácii nachádzajú sú do kódu importované z `.properties` súborov. Tieto súbory sú vždy k dispozícii v oboch jazykoch, systém určuje, ktorý jazyk použiť podľa jazyku prehliadača, v ktorom aplikácia beží. V prípade, že je jazyk prehliadača iný, ako angličtina alebo čeština, použije sa hlavný jazyk, ktorým je angličtina. Výber jazyka je ďalej možný kliknutím na ikony v menu aplikácie, ktoré sú k dispozícii na každej stránke systému. Informácia, aký jazyk bol vybraný je držaná v ManagedBeane s názvom LanguageBean, ktorá je ako jediná v Session Scope, tým pádom sa informácia o vybranom jazyku drží v session.

Ukážka 5.2 reprezentuje nástupný formulár. Výber jednotky pomocou adresárovej štruktúry vyvolá akciu, ktorá z databázy vytiahne všetky pozície, ktoré vybraná jednotka poskytuje. Tieto pozície je následne možné vybrať prostredníctvom komba s popisom Pozice. Nadriadený sa vyplní podľa vybratej pozície, v prípade potreby zmeny kombo umožňuje vybrať iného užívateľa. Kombo pre výber užívateľa vyťahuje užívateľov z databázy dynamicky na základe zadaných znakov priezviska užívateľa.

V prípade ostatných typov formuláru sa subjekt vyberá prostredníctvom dynamického komba rovnako ako nadriadený. Po výbere subjektu sa ostatné údaje vyplnia automaticky na základe väzby, ktorú poskytuje objekt Subjekt-Pozice.



Obr. 3.1: Domovská stránka



Obr. 3.2: Nástupný formulár

Kapitola 4

Testovanie

Testovanie slúži k odhaleniu chýb a dosiahnutiu a udržaniu dostatočnej kvality produktu. Každý softwarový produkt je nutné podrobiť testovaniu. Testovanie softwaru možno deliť podľa niekoľkých kritérií. z hľadiska obsluhy sa delí na manuálne a automatizované. Vo všetkých fázach vývoja je vhodné kombinovať a priebežne vykonávať oba tieto spôsoby. Manuálne testovanie slúži predovšetkým pre detekciu nezhody s funkčnou špecifikáciou a odchýlok od smeru k dosiahnutiu cieleného výsledku. z pohľadu softwarového systému sa rozlišujú izolované (jednotkové) testovania, kedy sa testujú jednotlivé funkčné časti, ktorými môžu byť buď celé funkčné moduly alebo jednotlivé operácie (metódy). Ďalším druhom v tomto členení je integračné testovanie. Pri integračnom testovaní sa testuje vzájomná interakcia medzi operáciami, modulmi a funkčnými blokmi. Pri integračnom testovaní je potreba vykonávať testy naprieč systémom, teda zadávať príkazy na užívateľskom rozhraní a sledovať priechod dát naprieč prezenčnou, servisnou a databázovou vrstvou.

4.1 Unit testy

Jednotkové testovanie je základná časť testovania, za jednotku je považovaná časť programu, ktorú možno samostatne testovať. Typicky sa jedná o jednu metódu triedy. U jednoduchších tried môže byť jednotkou celá trieda. Pri testovaní je dôležité izolovať jednotku od ostatných častí programu. Tieto testy sa vďaka použití maven spúštajú automaticky vo fáze maven test, ktorá predchádza fáze maven install. A teda poskytujú vývojárom najrýchlejšiu odozvu výsledkov testu. V ideálnom prípade by pri každom pridaní novej metódy do programu mal vzniknúť aj unit test, ktorý bude testovať funkcionalitu novej metódy. Konfigurácia takéhoto testu zaberie veľa času, preto sú v projekte implementované unit testy, ktoré testujú takú funkcionalitu, ktorá je spoločná pre rôzne systémy a testy bude možné použiť aj v budúcich projektoch. V adresáre src/test/java sú implementované Unit testy pre testovanie funkcionality:

- CRUDTest - Testuje operácie create a read nad databázou.
- Odoslanie emailu - Testuje sa vytvorenie objektu `Cr_Email` jeho vloženie do databázy a následné odoslanie na emailovú adresu užívateľa pod ktorým je test pustený.
- Spracovanie notifikácií žiadosti - Tento test umožňuje vytiahnuť z databázy žiadosť podľa id, nastaviť jej parametre tak aby ju systém bral ako práve schválenú a zavolať metódu, ktorá spracúva notifikácie.
- Pripravenie parametrov pre email - Testuje funkcionalotu, ktorá z kolekcie príjemcov emailov vytvorí jeden reťazec v ktorom sú jednotlivé emailové adresy oddelené pomocou znaku „;“. Zároveň testuje aj opačný postup.

Adresár src/test/resources obsahuje vlastné konfiguračné súbory, ktoré napríklad umožňujú, aby testy prebiehali nad databázou `intranet_idm_test`, aj keď samotná aplikácia používa databázu `intranet_idm_prod`.

4.1.1 Vyhodnotenie Unit testov

Možnosť spúšťať a testovať jednotlivé komponenty systému samostatne bola potrebná pri ladení a kontrole odosielania notifikácií do service desku. Pomocou testu „Odoslanie ticketu do Service Desku“ bol objavený prípad kedy odosielanie notifikácií neprebehlo a proces padol na `NullPointerException`. Po vyladení tejto funkcionality prebehli všetky testy bez chyby.

4.1.2 Výsledky testovania

Unit test	**true/false**
Pripravenie parametrov pre email	true
Odoslanie emailu	true
CRUDTest	true
Spracovanie notifikácií žiadosti	true

Tabuľka 4.1: Výsledky Unit testov

4.2 Výkonnostné testy

Aplikácia bola testovaná na výkon. Pri výkonových testoch je potreba stanoviť výkonové parametre a ich limity. Na základe týchto parametrov bývajú definovaná tzv. SLA (Service Level Agreement) kritériá, ktoré musia systém spĺňať. SLA limity bývajú zakotvené priamo v zmluve o dodaní produktu medzi dodávateľom a zákazníkom. Medzi parametre takéhoto testovania patrí

doba odozvy jednotlivých modulov, systémov ale aj operácií. Ďalej potom dátová priepustnosť alebo veľkosť správ na rozhraniach komunikačných kanálov. Parametrom testovania výkonu bol nefunkčný požiadavok na výkon 1.2.2.5.

4.2.1 Výsledky výkonnostných testov

Tabuľka 4.2 zobrazuje, koľko sekúnd trvalo zobraziť príslušnú stránku.

Názov stránky	**čas pred**	**čas po**
Dashboard.xhtml	0.506s	0.489s
Nastup.xhtml	2.640s	0.836s
Prestup.xhtml	2.461s	0.917s
Vystup.xhtml	0.785s	0.608s
Zmena.xhtml	2.640s	0.968s
SeznamZadosti.xhtml	0.892s	0.902s
InformacniSystem.xhtml	0.957s	0.941s
Opravneni.xhtml	0.874s	0.884s

Tabuľka 4.2: Výsledky výkonnostných testov pred a po optimalizáciách

Testovanie ukázalo, že okrem stránok Nastup.xhtml, Prestup.xhtml, a Zmena.xhtml sa stránky zobrazujú do 1 sekundy. Problém je z časti v tom, že tieto stránky potrebujú z databázy vytiahnuť zoznam všetkých oprávnení. Čas ktorý aplikácia potrebovala pre vytiahnutie dát z databázy bol približne 1.5 sekundy. Tento čas bol porovnaný s časom potrebným na vytiahnutie tých istých dát z databázy prostredníctvom SQL skriptu. Rozdiel časov viedol k zisteniu, že hibernate kritérium, pomocou ktorého systém vyťahuje dáta z databázy je potreba optimalizovať.

Ďalším zrýchlením bolo odstránenie zakomentovaného kódu v `.xhtml` stránke, ktorý sa kvôli cyklom v kóde stránky na pozadí zobrazoval pri každom oprávnení. Tým sa zmenšila veľkosť stránky prenášanej zo serveru na klinenta z 2,4 MB na 1,3MB.

Pri vybraní oprávnenia zaškrtnutím checkboxu, čo má následne zobraziť doplňujúce údaje a schvaľovateľov tohoto oprávnenia dochádzalo k obnovovaniu stránky čo spôsobovalo zdržanie. To zdržanie je primárne zapríčinené tým, že formulár je na pozadí reprezentovaný objektom, ktorý v sebe obsahuje strom oprávnení a pri každej operácii klienta sa znovu inicializuje. Na základe tohoto zistenia boli vykonané optimalizácie. Aby táto akcia nepotrebovala obnoviť formulár, bola funkcia zobrazenia týchto položiek implementovaná pomocou JavaScriptu priamo v stránke, tým pádom nie je potreba vykonávať request na server a údaje sa zobrazia okamžite.

Ďalšou optimalizáciu z pohľadu výkonu ale aj využiteľnosti, bola úprava spôsobu zobrazenia existujúcej schválenej žiadosti. Ku takejto žiadosti sa po

novom z databázy dotiahnu iba vybrané oprávnenia a tým pádom sa stránka takéhoto formuláru zobrazí do jednej sekundy.

4.2.2 Výsledky výkonnostných testov po optimalizácii

Po optimalizáciách sa podarilo znížiť čas potrebný pre zobrazenie stránok Nastup.xhtml, Prestup.xhtml, a Zmena.xhtml pod 1 sekundu. Výsledky dosiahnuté vo výkonnostných testoch pred aj po optimalizáciách boli dosiahnuté pri spustení aplikácie na serveri, ktorý je výkonom ekvivalentý produkčnému serveru.

4.3 Integračné testy

Pred odovzdaním softwarového produktu k akceptácii zákazníkom je vykonávané systémové integračné testovanie (SIT). Tento druh patrí medzi manuálne testy a výsledkom je identifikácia nezhôd so špecifikáciou produktu vzniknutých buď z dôvodu chyby programátora, alebo zlým pochopením zadania. Testy SIT sú vykonávané ľudmi s nezaujatým pohľadom na problém. Pre SIT vznikli testovacie prípady (test cases), podľa ktorých bola aplikácia testovaná. Testovacie prípady sú zhodné s prípadmi použitia 1.5.1. Ak systém umožní vykonať daný test case, považuje sa za splnený a bude označený hodnotou true.

4.3.1 Výsledky testovania

Test case	**true/false**
Výber formuláru podľa typu	true
Vyplnenie formuláru	true
Uloženie rozpracovaného formuláru	true
Odoslanie formuláru na doplnenie a schválenie nadriadeným	false
Schválenie formuláru nadriadeným	true
Schválenie alebo zamietnutie vybraných oprávnení	false
Zobrazenie záznamov v administrácii aplikácie	true
Pridanie záznamu v administrácii aplikácie	true
Editácia záznamu v administrácii aplikácie	true

Tabuľka 4.3: Výsledky integračných testov

Testovanie odhalilo nasledujúce chyby systému:

- V niektorých prípadoch sa do databázy neukladali doplňujúce údaje žiadosti.

- Schvaľovateľia oprávnení nemali prístup ku žiadosti, v ktorej mali schváliť oprávnenie.

Všetky chyby boli následne opravené.

4.4 Užívateľské akceptačné testy

Užívateľské akceptačné testy (UAT) vykonávajú pracovníci určení zákazníkom. UAT slúži pre overenie zo strany zákazníka, či systém spĺňa ním kladené požiadavky na funkcionalitu, vzhľad a ovládateľnosť. Pre UAT môžu byť použité rovnaké testovacích prípady ako pre SIT. Testovanie UAT vykonali pracovníci oddelenia HR, ktorí budú aplikáciu používať každodenne a presne vedia, čo od takéhoto systému potrebujú. V prípade interného vývoja tohoto systému zastávajú pracovníci HR úlohu pracovníkov zákazníka. Testy vykonávali 4 na sebe nezávislí užívatelia, ktorí boli vopred oboznámený s novým procesom schvaľovania oprávnení. Každý z týchto užívateľov vykonal každý test case aspon raz. Ak systém umožní vykonať daný test case všetkým užívateľom test case sa považuje za splnený a bude označený hodnotou true.

4.4.1 Výsledky testovania

Test case	**true/false**
Výber formuláru podľa typu	true
Vyplnenie formuláru	true
Uloženie rozpracovaného formuláru	true
Odoslanie formuláru na doplnenie a schválenie nadriadeným	true
Schválenie formuláru nadriadeným	true
Schválenie alebo zamietnutie vybraných oprávnení	true
Zobrazenie záznamov v administrácii aplikácie	true
Pridanie záznamu v administrácii aplikácie	true
Editácia záznamu v administrácii aplikácie	true

Tabuľka 4.4: Výsledky akceptačných testov

Testovanie neodhalilo žiadne chyby systému ale poukázalo na funkcionalitu, ktorá by užívateľom zjednodušila prácu so systémom. Jedná sa o zmeny vzhľadom na zadanie systému, preto sú označené ako Change Requesty:

- Vo schválenej žiadosti zobraziť iba oprávnenia, o ktoré bolo žiadané.
- Emaily pre viac príjemcov sa posielali jednotlivo, tým pádom príjemcovia nevideli ostatných príjemcov. Posielať tieto maily radšej skupinovo.

- V prípade, že nadriadený schváli žiadosť a je zároveň schvaľovateľom jedného alebo viac oprávnení, o ktoré je prostredníctvom formuláru žiadané, tieto oprávnenia budú automaticky schválené. Táto funkcia bola pôvodne implementovaná iba pre novú žiadosť. Po novom sa systém bude takto správať aj pre žiadosti, ktoré čakajú na schválenie nadriadeným.
- Užívateľ v roli HR bude môcť schvaľovať aj žiadosti, ktoré čakajú na schválenie nadriadeným. Podobne ako v predchádzajúcom prípade, bola táto funkcionalita pôvodne implementované iba pre nové žiadosti.

Kapitola 5

Nasadenie systému a budúci rozvoj

Kapitola obsahuje časť, ktorá popisuje nasadenie systému, požadovaný software pre spustenie aplikácie a postup inštalácie. Diagram nasadenia 2.7 sa nachádza už v kapitole Návrh aplikácie, preto nie je nutné uvádzať ho znovu. Druhá časť popisuje požiadavky pre budúci rozvoj aplikácie a ukážky ich realizácie.

5.1 Nasadenie systému

Systém je nasadený v rámci infraštruktúry holdingovej spoločnosti, v ktorej je dostupný pre každý jeden z podnikov spoločnosti. Počas prvých troch mesiacov fungovania systému bol systém použitý pobočkami na Slovensku, v Česku a v Rusku. Za tento čas bolo do systému vložených 370 žiadostí, z toho 316 bolo schválených. Prostredníctvom týchto žiadostí bolo požiadané o 844 oprávnení, z toho bolo 730 schválených. Žiadosti do systému vložilo 103 rôznych ľudí, bolo žiadané pre 232 rôznych ľudí, na schvaľovaní žiadosti nadriadeným sa podielalo 90 rôznych ľudí vrátane personalistov a na schvaľovaní oprávnení sa zatiaľ podielalo 92 rôznych schvaľovatelov.

Jedná sa o enterprise aplikáciu, ktorá ku spusteniu potrebuje iba správnu konfiguráciu a pobeží na akomkoľvek serveri, ktorý bude spĺňať určité softwarové požiadavky.

5.1.1 Softwarové požiadavky

- Operačný systém s podporou Javy
- Verziu Javy minimálne JavaEE 7 a Java 7

- Aplikačný server, pre vývoj aplikácie bol použitý Tomcat 8.0[3]
- Zdroj dát, aplikácia je nakonfigurovaná na Microsoft SQL Server, názov databázy je `intranet_idm_prod`, dodatočné dáta získava z databázy `intranet_service`.

5.1.2 Inštalácia

Akonáhle server splňuje softwarové požiadavky, je inštalácia priamočiara.

- Skompilovaný war súbor je treba nahrať do servlet kontajnera na aplikačnom serveri.

5.2 Budúci rozvoj

Okrem hlavného účelu systému, ktorým je monitorovanie prideľovania a odoberania oprávnení, systém momentálne slúži aj na zakladanie nových užívateľov a administráciu oprávnení. Systém pracuje nad svojou databázou, ktorá je každodenne dopĺňaná o dáta z intranetu. Ide iba o dáta týkajúce sa organizačnej štruktúry, užívateľov a pracovných pozícií. Ďalšie rozšírenie systému vyplýva z požiadavku na výraznú zmenu v organizačnej štruktúre. Vkladanie dát do intranetu je robené cez jednoduché rozhranie ktoré, neumožňuje hromadne upravovať organizačnú štruktúru.

Ďalší rozvoj spočíva v implementácii modulu, ktorý umožní efektívnu administráciu takýchto dát. Tým pádom bude možné vkladať všetky dáta priamo do tohoto systému, čo umožní zmenu v dátovom toku. Úloha systému sa týmto rozšíri o nástroj na administráciu organizačnej štruktúry pre intranet. Tým pádom sa systém stane hlavným zdrojom dát.

V súčastnosti je tento modul vo fáze implementácie. Ukážka 5.2 znázorňuje užívateľský profil. Táto stránka zobrazuje všetky osobné údaje, ktoré je u užívateľa potrebné držať. Keďže užívateľ môže formálne pôsobiť na viacej pozíciách, stránka umožňuje zobraziť viacej pozícií. Užívateľ v roli ROLE_HR_EMPLOYEE má možnosť užívateľský profil editovať, to znamená editovať osobné údaje a pridávať alebo odoberať pozície. Stránka umožňuje pridávanie pozícií podľa organizačnej jednotky, ak ide o zavádzanie novej pozície, systém umožní vytvorenie novej pozície priamo v tejto stránke. Tým, že bude systém hlavným zdrojom dát, bude funkcionalita pre pridávanie resp. odoberanie pozícií spustená aj pri schválení nástupného resp. výstupného formuláru.

Administrácia organizačnej štruktúry je riešená pomocou stránky v ukážke 5.3. Jednotky sú reprezentované pomocou adresárovej štruktúry, rozhranie

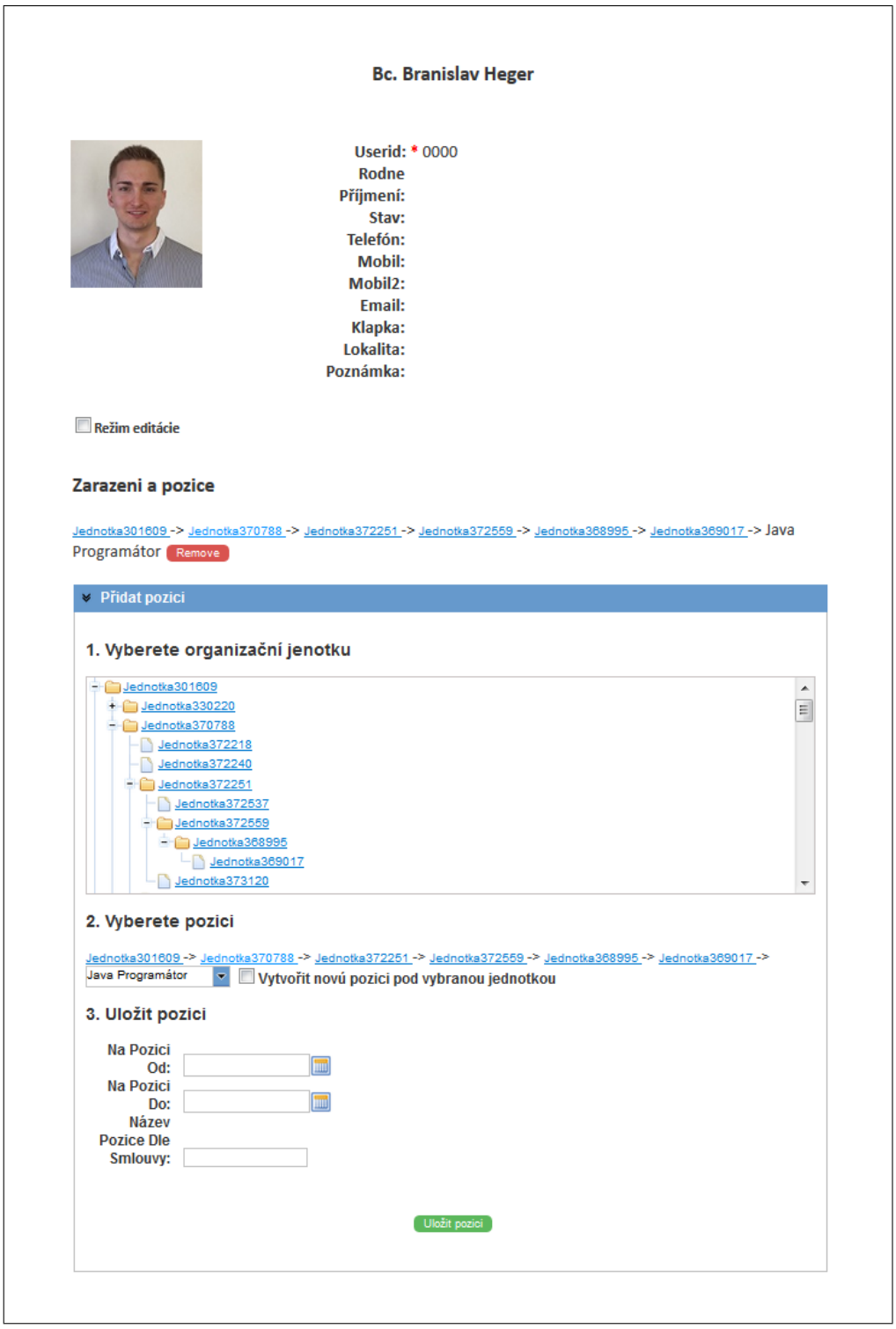
[3] https://tomcat.apache.org/

stránky je inšpirované systémom TotalComander [4], ktorý obsahuje dva adresárové okná vedľa seba. Kliknutím na jednotku v adresárovej štruktúre ju užívateľ vyberie, tým sa pod týmto oknom zobrazí celá cesta ku vybranej jednotke a pod ňou sa zobrazí zoznam jednotiek, ktorým je táto jednotka nadriadená. Pri vybranej jednotke sa zobrazujú tlačítka reprezentujúce operácie:

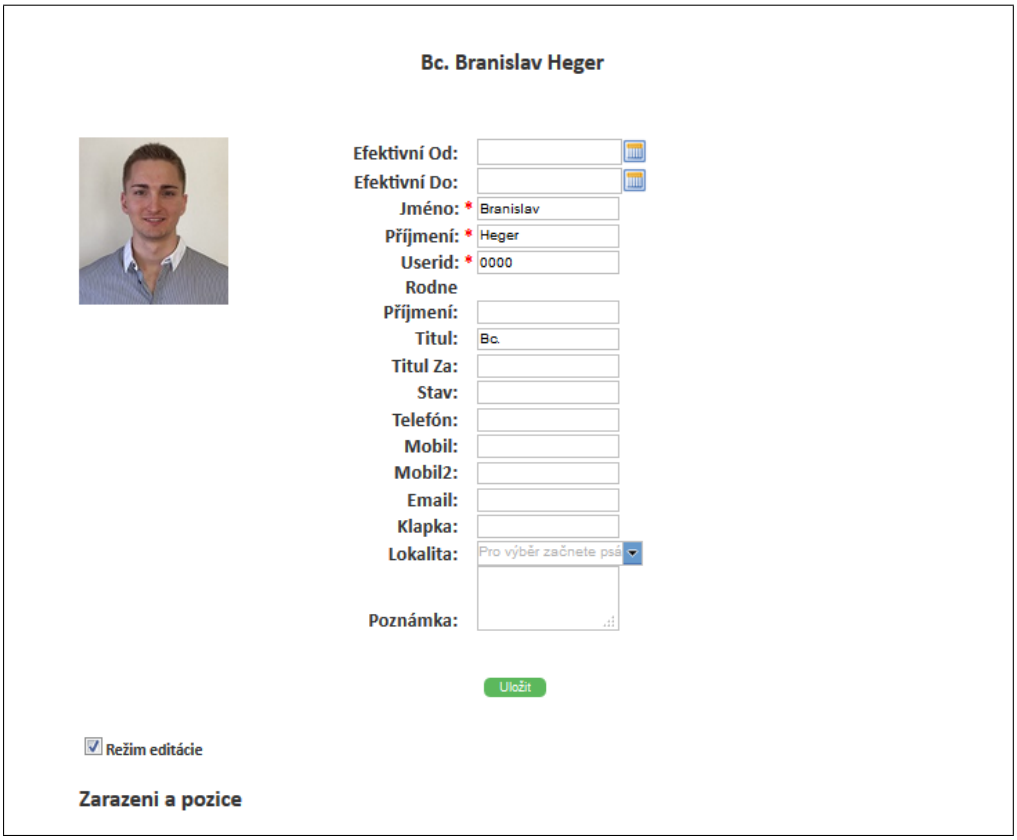
- Vymazať: táto operácia znamená zrušiť efektívnu platnosť záznamu. Vymazaná jednotka zo zoznamu zmizne. Vybraním checkboxu „Zobraziť neplatné záznamy“ sa neplatné záznamy zobrazia, ale sú preškrtnuté, tlačítko vymazať sa nahradí tlačítkom obnoviť, ktoré efektívnu platnosť obnovuje.
- Vytvoriť novú jednotku: otvori popup okno pre vytvorenie novej jednotky, vybraná jednotka slúži ako nadriadená jednotka novo vytváranej jednotky.
- Editovať: otvori popup okno pre editáciu vybranej jednotky.
- Domov: tlačítko zobrazí zoznam jednotiek, ktoré nemajú nadriadenú jednotku

Jednotky, ktoré sa zobrazujú v zoznamoch, obsahujú navyše tlačítko „Presunúť doprava“ resp. „Presunúť doľava“. Tieto tlačítka umožňujú presúvať jednotky a tým pádom meniť organizačnú štruktúru prostredníctvom jedného kliknutia. Presunutej jednotke sa zmení nadriadená jednotka na tú, korá bola vybraná v druhom okne.

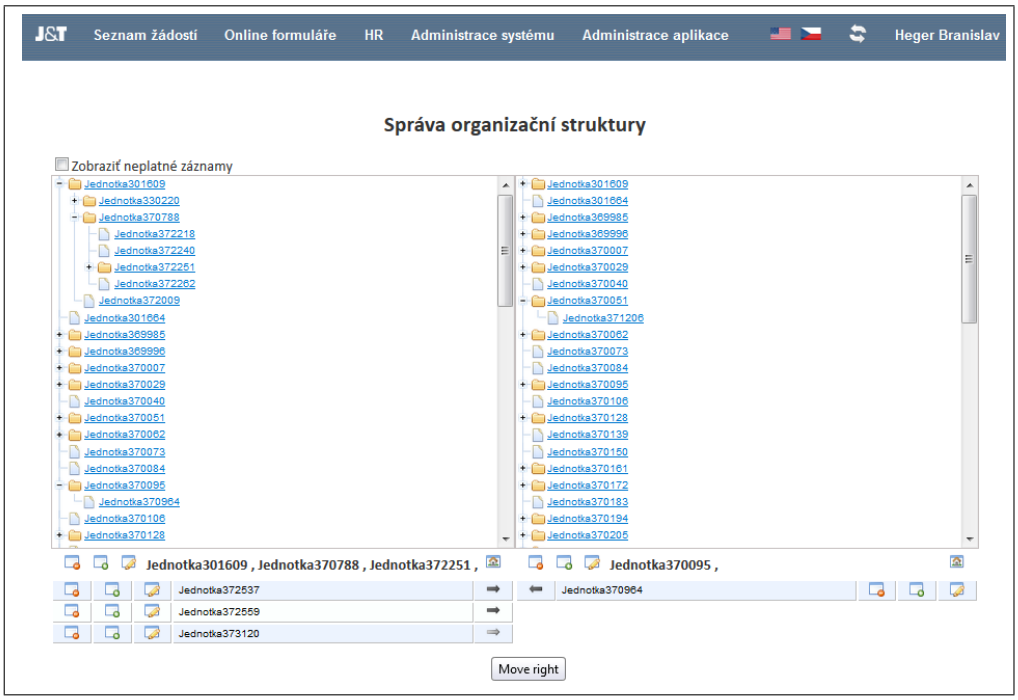
[4] https://www.ghisler.com/



Obr. 5.1: Detail užívateľa



Obr. 5.2: Nastup



Obr. 5.3: Správa organizačnej štruktúry

Záver

Cieľom práce bol návrh a funkčná implementácia systému pre správu životného cyklu zamestnanca. V časti analýza práce bol kladený dôraz na požiadavky, ktoré od systému očakáva zadávateľ. Rovnako tak na požiadavky, ktoré od systému požadujú potencionálni užívatelia. Na základe všetkých požiadavkov boli vypracované prípady použitia. V časti návrh systému boli dôkladne opísané technológie, pomocou ktorých bol systém implementovaný. V tejto časti vznikol dátový model, ktorý umožní požadovanú funkciu na verziovanie dát. Architektúra systému je zadefinovaná pomocou diagramu nasadenia a komunikácia medzi jednotlivými časťami systému pomocou sekvenčných diagramov. Užívateľské rozhranie bolo navrhnuté, aby pokrývalo všetky tieto požiadavky. Prechody jednotlivých scenárov použitia pokrývajúcich požiadavky na systém boli detailne navrhnuté prostredníctvom drôtených modelov. Pri návrhu bol kladený dôraz na jednoduchosť a prehľadnosť, cieľom bolo navrhnúť rozhranie, ktoré budú vedieť užívatelia ovládať intuitívne.

Systém je implementovaný pomocou platformy Java EE, pre implementáciu grafického užívateľského rozhrania bol využitý JSF framework Richfaces. Úložisko dát poskytuje Microsoft SQL Server 2014.

Systém bol testovaný pomocou na to určených prostriedkov. Prebehlo automatizované a manuálne testovanie.

Výsledný software je možné jednoducho nasadiť v rámci infraštruktúry spoločnosti. Návod a softwarové požiadavky sú uvedené v kapitole Nasadenie systému a budúci rozvoj, táto kapitola sa naviac zaoberá ďalším rozvojom systému.

Systém je plne funkčný a používa sa každodenne, je dostupný na všetkých pobočkách spoločnosti. Počas prvých troch mesiacov fungovania systému bol systém použitý pobočkami na Slovensku, v Česku a v Rusku. Za tento čas bolo do systému vložených 370 žiadostí, z toho 316 bolo schválených. Prostredníctovm týchto žiadostí bolo požiadané o 844 oprávnení, z toho bolo 730 schválených. Žiadosti do systému vložilo 103 rôznych ľudí, bolo žiadané pre 232 rôznych ľudí, na schvaľovaní žiadosti nadriadeným sa podielalo 90 rôznych

ľudí vrátane personalistov a na schvaľovaní oprávnení sa zatial podielala 92 rôznych schvaľovateľov.

Literatúra

[1] SUMMERVILLE, I.: *Software Engineering*. USA: Addison-Wesley Publishing Company, 2010, ISBN 9780137035151/0137035152.

[2] Oracle: The Java™ Tutorials: About the Java Technology. 2015 [cit. 2017-03-01]. Dostupné z: http://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html

[3] MANN, K. D.: *JavaServer Faces in Action*. Greenwich: Manning Publications Co., 2004, ISBN 1-932394-11-7.

[4] NEWARD, T.: *Server-Based Java Programming*. NY: Manning Publications, 2000, ISBN 9781884777714.

[5] FIELDS, D. K.; KOLB, M. A.; BAYERN, S.: *Web Development with JavaServer Pages*. NY: Manning Publications, second edition vydání, 2001, ISBN 9781930110120.

[6] JSF - Architecture: What is MVC Design Pattern? [online]. [cit. 2017-03-01]. Dostupné z: https://www.tutorialspoint.com/jsf/jsf_architecture.htm

[7] Oracle: The Lifecycle of a JavaServer Faces Application. [online], 2014 [cit. 2017-03-01]. Dostupné z: https://docs.oracle.com/javaee/7/tutorial/jsf-intro006.htm

[8] WHEELER, W.; WHITE, J.: *Spring in Practice*. NY: Manning Publications, 2013, ISBN 9781935182054.

[9] Spring: Introduction to the Spring Framework [online]. [cit. 2017-03-01]. Dostupné z: https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/overview.html

[10] Zyl, J. V.: *Maven: The Definitive Guide*. O'Reilly Media, 2009, ISBN 978-0-596-10302-6.

[11] COLLEDGE, R.: *SQL Server 2008 Administration in Action*. NY: Manning Publications, 2009, ISBN 9781933988726.

[12] Hickson, I.; Berjonand, R.; Faulkner, S.; aj.: HTML5 [online]. 2014, [cit. 2017-04-01]. Dostupné z: http://www.w3.org/TR/2014/REC-html5-20141028/

[13] Mandel, L.: Describe REST Web services with WSDL 2.0 [online]. 2008 [cit. 2017-04-10]. Dostupné z: https://www.ibm.com/developerworks/webservices/library/ws-restwsdl/

[14] Nielsen, J.: Enhancing the Explanatory Power of Usability Heuristics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, New York, NY, USA: ACM, 1994, ISBN 0-89791-650-6, s. 152–158, doi:10.1145/191666.191729. Dostupné z: http://doi.acm.org/10.1145/191666.191729

[15] Apache: Maven AntRun Plugin [online]. [cit. 2017-03-01]. Dostupné z: http://maven.apache.org/plugins/maven-antrun-plugin/

[16] MojoHaus: JAX-WS Maven Plugin [online]. [cit. 2017-04-10]. Dostupné z: http://www.mojohaus.org/jaxws-maven-plugin/

[17] LOUVEL, J.; TEMPLIER, T.; BOILEAU, T.: *Developing RESTful web APIs in Java*. NY: Manning Publications, 2012, ISBN 9781935182344.

[18] Bishop, P.; Warren, N.: Java Tips: Java Tip 67: Lazy instantiation [online]. 1999 [cit. 2017-04-07]. Dostupné z: http://www.javaworld.com/article/2077568/learn-java/java-tip-67--lazy-instantiation.html

Dodatok A

Zoznam použitých skratiek

GUI Graphical user interface

XML Extensible markup language

EJB Enterprise JavaBeans

API Java Application Programming Interface

JVM Java Virtual Machine

JSF JavaServer Faces

UI User interface

MVC Model-View-Controller

JSP JavaServer Pages

POJO Plain Old Java Object

IOC Inversion of Control

ORM Object/Relational Mapping