Sem vložte zadání Vaší práce.

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

Bakalářská práce

Příprava integrace IS pro správu zaměstnanců se souvisejícími firemními systémy

Pavel Peroutka

Vedoucí práce: Ing. Pavel Štěpán

10. května 2015

Poděkování

Tímto děkuji Ing. Pavlovi Štěpánovi za jeho ochotu a příjemnou spolupráci při vedení této práce. Dále děkuji Ing. Martinu Brůžkovi, že mi poskytl možnost psát práci na toto téma a že vstřícně přijal nabídku být mým oponentem. Chci také poděkovat mé rodině a kamarádům, kteří mě při psaní této práce podporovali.

Prohlášení

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

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

V Praze dne 10. května 2015 .....

Odkaz na tuto práci

Peroutka, Pavel. *Příprava integrace IS pro správu zaměstnanců se souvisejícími firemními systémy.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Předmětem práce je reimplementace informačního systému pro správu zaměstnanců. Původní řešení je implementováno v databázovém nástroji MS Access, nové je realizováno formou webové aplikace. Webová aplikace je postavena podle vzoru server-klient, vzájemně komunikující prostřednictvím REST rozhraní. Klientská část je single page application, vytvořena za použití frameworku AngularJS. V této práci je popsán návrh i samotná implementace tohoto systému.

Klíčová slova Informační systém, Evidence zaměstnanců, Single page appliacation, REST API, AngularJS

Abstract

The Topic of this thesis is reimplementation of a human resource management system. The Original solution is implemented using the database tool MS Access, and the new one is realised as a web application. The Web application is based on server-client pattern, mutually communicating over a REST interface. The Client side is a single page application created using AngularJS framework. This thesis contains both the design and implementation of this system.

Keywords Information system, Human resource management, Single page application, REST API, AngularJS

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Informační systémy se staly synonymem pro efektivní práci s informacemi. Používání informačních systémů na úřadech, v podnicích, ve školách a dalších organizacích, dokonce i pro účely jednorázových událostí, se stávají standardem. Jejich použití bývá i přes mnohdy vysoké pořizovací náklady výhodné.

Pro účely podpory řízení podniku se ve společnostech často používají informační systémy napomáhající řízení vztahů se zákazníky a veřejností (CRM), lidských zdrojů (HRM), produkce, nákupů a prodejů, skladů, poboček, logistiky, či dalších odvětví, souvisejících s činnostmi podniku. Ideálním stavem je, když spolu tyto systémy vzájemně spolupracují, tedy jsou integrované. V takovém stavu není třeba data nikam zadávat více než jednou, je snazší udržovat vnitřní datovou konzistenci a také aktuálnost dat. V současné době se stávají populárními ERP systémy, které tyto jednotlivé systémy nabízejí jako komponenty do jednoho uceleného komplexního řešení. Pokud ale v již zaběhnutých společnostech existuje větší množství fungujících systémů, je častým řešením jejich integrace nějakým vlastním způsobem.

Tato bakalářská práce se zabývá reimplementací informačního systému pro správu zaměstnanců pro středně velkou společnost čítající řádově stovky zaměstnanců. Nová implementace přináší oproti původnímu řešení několik vylepšení, jak je uvnitř systému navržena struktura dat, která reprezentují skutečnost. Nabízí nové možnosti integrace s ostatními systémy a také poskytuje více prostoru pro další rozšiřování systému. K implementaci je využito moderních technologií a frameworků. V této práci je popsán proces vývoje systému, od sběru požadavků a prozkoumání původního systému, až po implementaci a testování.

Kapitola 1

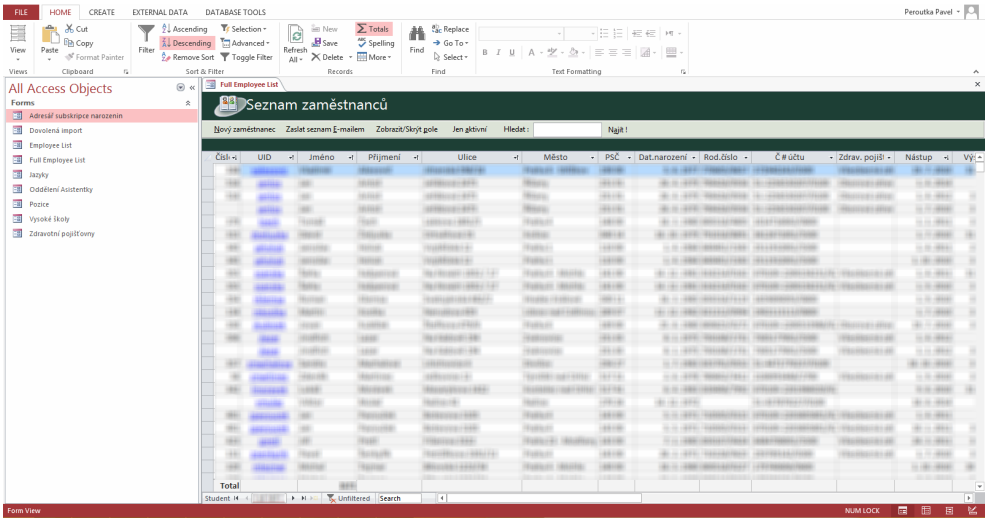
Současný stav

1.1 Současný stav problematiky

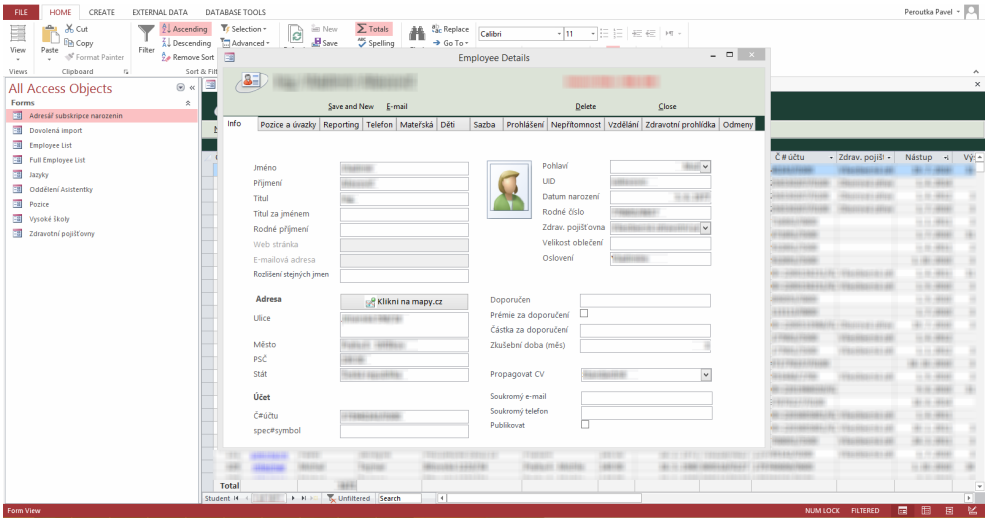
Každý podnik si potřebuje držet informace o svých zaměstnancích, už jen ze zákonné povinnosti. U společností větších než jen několik zaměstnanců začíná evidence zaměstnanců, zejména její struktura a přehlednost, nabývat na důležitosti. Kromě papírové evidence dnes existuje možnost uchovávat evidenci také v sotwarovém systému. Papírová evidence je velmi jednoduchý a levný způsob, vyhovující pro uchovávání relativně malého objemu informací, avšak během růstu společnosti může nastat situace, kdy začne být tento způsob neefektivní. Softwarové systémy pro správu zaměstnanců jsou dnes běžnou součástí i menších společností. Oproti papírově psané formě nabízejí větší přehlednost, vzdálený přístup, centralizaci úložiště, kontrolu jednoty a konzistence zadávaných údajů, upozorňování na termíny a mnoho dalších výhod. V případě přítomnosti dalších systémů, které souvisí se zaměstnanci společnosti, je vhodné systém pro správu zaměstnanců s těmito systémy integrovat. Pokud jsou seznamy zaměstnanců vedeny na více než jednom místě, vytváří se prostor pro možné nekonzistence, vytvořené chybou člověka. Současně vzniká větší pracnost s údržbou těchto seznamů. Příkladem takových systémů může být firemní adresář kontaktů, systém pro vykazování práce či evidence přiřazení firemních zařízení pracovníkovi.

1.2 Současný stav v podniku

Systém, který je předmětem této práce, již existuje a je používaný ve středně velké firmě, čítající řádově stovky zaměstnanců. Slouží pro evidenci pracovníků, pracovních smluv, pozic, mzdových sazeb, nároků na dovolené a mnoho dalších informací potřebných pro chod společnosti (obrázky 1.1 a 1.2). V současné době systém ne zcela vyhovuje požadavkům uživatelů, proto bude v této práci nahrazen efektivnějším řešením.



Obrázek 1.1: Původní systém - přehled pracovníků



Obrázek 1.2: Původní systém - detail pracovníka

Kapitola 2

Analýza

2.1 Popis současného systému

Původní systém sloužící pro správu zaměstnanců použitý ve společnosti je implementován v databázovém nástroji MS Access [2] (součást balíku MS Office) s backendovou databází na MS SQL Serveru. MS Access nabízí práci s databázovými daty prostřednictvím různých definovatelných tabulek a formulářů. Navíc je možné rozšířit standardní funkcionalitu pomocí kódu v jazyce Visual Basic for Applications (VBA). V tomto jazyce je velká část funkcionality tohoto původního řešení napsána.

Systém používají zaměstnanci v oddělení personalistiky a evidují v něm následující informace o zaměstnancích.

- Identifikace, osobní údaje a kontakt
- Pracovní úvazky a smlouvy
- Zkušební doba zaměstnance
- Firemní hierarchie, oddělení a pozice
- Nároky na placené volno
- Daňové úlevy
- Mateřské dovolené
- Vzdělání, jazykové dovednosti

Následně slouží jako zdroj dat pro další firemní systémy.

- Systém dodává seznam zaměstnanců do CRM, kde následovně mohou být dosazováni do různých rolí a funkcí ve vztahu k zákazníkovi.

- Kompletní evidence zaměstnanců se nahrává do datového skladu, kde slouží jako dimenze k propojování entit ve vztahu k zaměstnanci. Dále jsou data využívána například v SQL Server Reporting Services.
- Některé atributy pracovníka (jméno, telefonní číslo, název oddělení apod.) jsou aktualizovány ve službě Active Directory (AD), která slouží, mimo jiné, jako firemní adresář.
- Data dále využívají aplikace pro hodnocení zaměstnanců, evidenci aut, evidenci majetku, tvorbu životopisů a další.

Prostřednictvím systému se také importují a archivují data o čerpání dovolených, které dodává účetní.

Data evidovaná v tomto systému jsem důvěrná, proto je důležité zaměřit se i na zabezpečení. Zabezpečení původního systému je řešeno na úrovni databáze. Tento způsob je možné využít, protože uživatel v případě použití MS Access přistupuje k datům přímo, pod svým vlastním účtem, nikoli přes servisní účet. Druhý způsob zabezpečení, spíše jen preventivní, je poskytnutí samotné aplikace pouze uživatelům, pod které spadá kompetence správy zaměstnanců.

MS Access je „tlustý klient“[1], což přináší nutnost správy jeho verzí u koncových uživatelů a vzniká prostor pro nekonzistence. Další omezení plynoucí z této architektury je závislost na uživatelské platformě, přítomnosti balíku MS Office a také verzi MS Office. Veškeré úpravy systému je možné dělat pouze v rámci nástrojů MS Access a jazyka VBA, což je v některých ohledech limitující. Ve vývojovém prostředí VBA jsou omezené možnosti ladění chyb a také neexistuje možnost automatizovaného testování.

2.2 Sběr požadavků

Jedním z prvních kroků, kterého bylo při tvorbě tohoto nového IS potřeba učinit, byl sběr požadavků na funkcionalitu sytému. Tento krok je důležitý k pochopení, jak má systém vypadat, k čemu bude sloužit, kdo a jak ho bude používat a k celkovému pochopení dané situace. Sběr požadavků byl uskutečněn během několika schůzek s jedním z uživatelů původního systému. Výsledkem je specifikace požadavků, které jsou na dílo kladeny ze strany zadavatele.

[1] program mající většinu své funkcionality či zdrojů dostupné lokálně

2.3 Specifikace

2.3.1 Systémové vlastnosti

- Správa zaměstnanců
  - Vytvoření, úprava, smazání zaměstnance
  - Úprava základních atributů zaměstnance (jméno, identifikace, adresa, kontakt, ...)
  - Evidence pracovních poměrů a úvazků (smluvní vztah, doba, pozice, oddělení, nadřízený, typ úvazku, ...)
  - Evidence mateřských dovolených
  - Evidence mzdových sazeb
  - Evidence daňových prohlášení
  - Evidence nároků na dovolenou
  - Evidence dosaženého vzdělání a jazykových znalostí
  - Evidence zdravotních pojišťoven
  - Evidence zdravotních prohlídek
- Vyhledávání zaměstnanců
- Správa číselníků
  - Jazykových znalostí, pracovních pozic, vysokých škol, zdravotních pojišťoven
- Správa zaměstnaneckých rolí a přiřazování rolí zaměstnancům
- Import dovolených
- Výpočet dovolených
- Upozorňování na termíny

2.3.2 Předpoklady a závislosti

- Úvazky korespondujícího pracovního poměru se nesmí datově překrývat
- Úvazky nesmí datově přesahovat korespondující pracovní poměr
- Nesmí se datově překrývat pracovní poměry stejného typu smluvního vztahu
- Zaměstnanec má jedinečné UID (login)
- Interní číslo zaměstnance v pracovním poměru se může opakovat, ale pouze u jednoho zaměstnance
- Mateřská dovolená nesmí být mimo zaměstnanecký pracovní poměr

2.3.3 Uživatelské role

- Autorizovaný uživatel (úplná práva čtení a zápisu)

2.3.4 Softwarové rozhraní

- MS SQL Server 2012
- Internet Information Services

2.3.5 Operační prostředí

- Internet explorer
- Chrome

2.3.6 Uživatelské rozhraní

- Jednoduché menu do třech úrovní
- Stránky členěny do záložek (tabů)

2.3.7 Komunikační rozhraní

- K autentizaci uživatelů se bude používat Integrated Windows Authentication
- Data číselníku se získávájí ze systému CRM prostřednictvím linkované databáze

2.3.8 Nefunkční požadavky

- Systém bude schopný najednou obsloužit alespoň 10 uživatelů
- Spolehlivé omezení přístupu, aplikace obsahuje citlivá data
- V provozní době je třeba alespoň 90% dostupnost
- Systém bude v českém jazyce

2.4 Porovnání s původním řešením

Oproti původnímu řešení se částečně změnila struktura doménového modelu. Nový model přináší opravu špatně navržené dekompozice vztahů mezi objekty reálného světa a také možnost rozšíření funkcionality systému.

2.4.1 Vztah společnost a pracovník

Vztah pracovníka a společnosti je dán smlouvou, která vymezuje základní právní údaje o spolupráci. Pro interní potřeby firmy je tento vztah dále upřesňován, například je pracovníkovi přiřazen nadřízený, je začleněn do firemního oddělení a je kategorizován dle pracovní pozice či úrovně svých zkušeností. V původním systému je tento vztah reprezentován jednou entitou. To v důsledku znamená, že pokud je například pracovníkovi přiřazen nový nadřízený, je nutné vytvořit nový záznam obsahující i základní údaje o spolupráci, které nejsou změnou nadřízeného nijak ovlivněny. Tento způsob zaviňuje nejen redundanci dat, ale také neefektivní a nespolehlivé třídění těchto záznamů pod jednotlivé období spolupráce (reprezentovány skutečnou smlouvou), kterých může být historicky více. Za tímto účelem je v původním systému u každého záznamu nejen datový rozsah platnosti samotného záznamu, ale také rozsah doby spolupráce. Ten se zbytečně opakuje u záznamů spadající pod stejné období spolupráce a také vzniká prostor pro chyby v datech, protože tyto hodnoty nejsou vzájemně nijak kontrolovány.

Řešením je vztah dekomponovat na dvě entity. Entitu reprezentující samotnou pracovní smlouvu a druhou entitu, zastřešující všechny dílčí změny, které se týkají pouze vnitřního fungování společnosti.

2.4.2 Řazení pracovníků do rolí

V původním systému existují dva seznamy pracovníků, reprezentující dvě role. Zařazením pracovníka do daného seznamu se mu daná role přiřadí. Seznamy jsou definovány v kódu systému a uživatel je nemá možnost upravovat či vytvářet nové. V novém návrhu tvoří role samostatnou entitu, kterou bude možné upravovat či vytvářet.

2.4.3 Paralelní smlouvy

Standardně nemůže mít pracovník více než jednu pracovní smlouvu se společností. Výjimkou je druhý brigádnický pracovní poměr během mateřské dovolené. Tato možnost není v původním systému zohledněna, přestože jsou tyto případy běžné. Nový systém již bude s tímto pravidlem pracovat.

2.5 Srovnání s existujícími produkty

Paleta softwarových produktů pro správu zaměstnanců je široká, ať pod open-source či komerční licencí. Je proto ke zvážení, zda by se vyplatilo nasadit do firmy jedno z hotových řešení, či je třeba implementovat systém nový. Ve společnosti se však za dobu fungovaní vytvořila potřeba evidovat mnoho dat, specifických právě pro tuto společnost, na které musí být systém připraven. Systém také čerpá data z jiných systémů jako zdroj hodnot pro číselníky. Proto

by bylo potřeba možný produkt s požadovaným řešením porovnat a dodatečnou funkcionalitu doplnit. V případě zvolení open-source řešení by bylo možné, po jeho prozkoumání a pochopení, doprogramovat další funkcionalitu za využití vlastních zdrojů a případně spravovat další změny. V případě komerčního softwaru by systém musel mít možnost doprogramovat vlastní modul s funkcionalitou, anebo možnost využít služeb rozšíření systému poskytovaných tvůrcem systému.

2.5.1 BambooHR

BambooHR [3] je komerční HRM systém, umožňující správu prostřednictvím webového prohlížeče či aplikace pro mobilní telefon. Nabízí správu zaměstnanců, dovolených, odměn, zaměstnaneckých rolí a další. Součástí systému je též API pro integraci s dalšími systémy. Výrobce nabízí při nasazování systému do společnosti import původních dat a přizpůsobení systému tamním podmínkám. Bohužel je dostupný pouze v anglickém jazyce.

2.5.2 HR Vema

Komerční informační systém HR Vema [4] nabízí evidenci zaměstnanců, včetně jejich vzdělání, dovolených i organizačních změn. Přístup k systému je prostřednictvím webového prohlížeče. Pro přizpůsobení potřebám společnosti nabízí firma Vema předimplementační studii a následnou úpravu systému.

2.5.3 OrangeHRM

Jedním z open-source HRM systémů je OrangeHRM [5]. Společnost OrangeHRM nabízí mnoho služeb, jako jsou nasazení systému, dlouhodobá podpora a také školení uživatelů. Samotný systém je open-source, zdrojové kódy jsou volně dostupné. Systém eviduje zaměstnance, kvalifikace, dovolené, organizační změny, mzdy a další.

2.6 Zvolené řešení

Jako řešení byla zvolena implementace nového systému. Protože se nebude zakládat na žádné již existující funkcionalitě, systém nebude mít žádné nepotřebné funkce navíc, díky tomu bude systém jednodušší a přehlednější jak z uživatelského, tak vývojářského pohledu. Díky možnosti volného výběru technologií bude možné systém snáze zakomponovat do systémové architektury společnosti a také připravit rozhraní na možnou budoucí integraci s dalšími systémy, současně používanými ve firmě.

Kapitola 3

Návrh

3.1 Případy užití

Případy užití ilustrují kdo a jakým způsobem bude systém používat. [6] Jsou užitečné k vytvoření rychlé a konkrétní představy k čemu bude systém sloužit. Také jsou vhodné ke kontrole splnění požadavků na systém a tedy i k tvorbě testovacích scénářů.

V případě tohoto IS má přístup k aplikaci pouze uživatel, který je členem oprávněné skupiny firemní domény. Má práva prohlížet i upravovat veškerá data, ostatním uživatelům je přístup zakázán. Většina případů užití přímo souvisí s prohlížením či úpravami pracovníka a asociovaných entit (obrázek 3.1).

- Vyhledat pracovníka

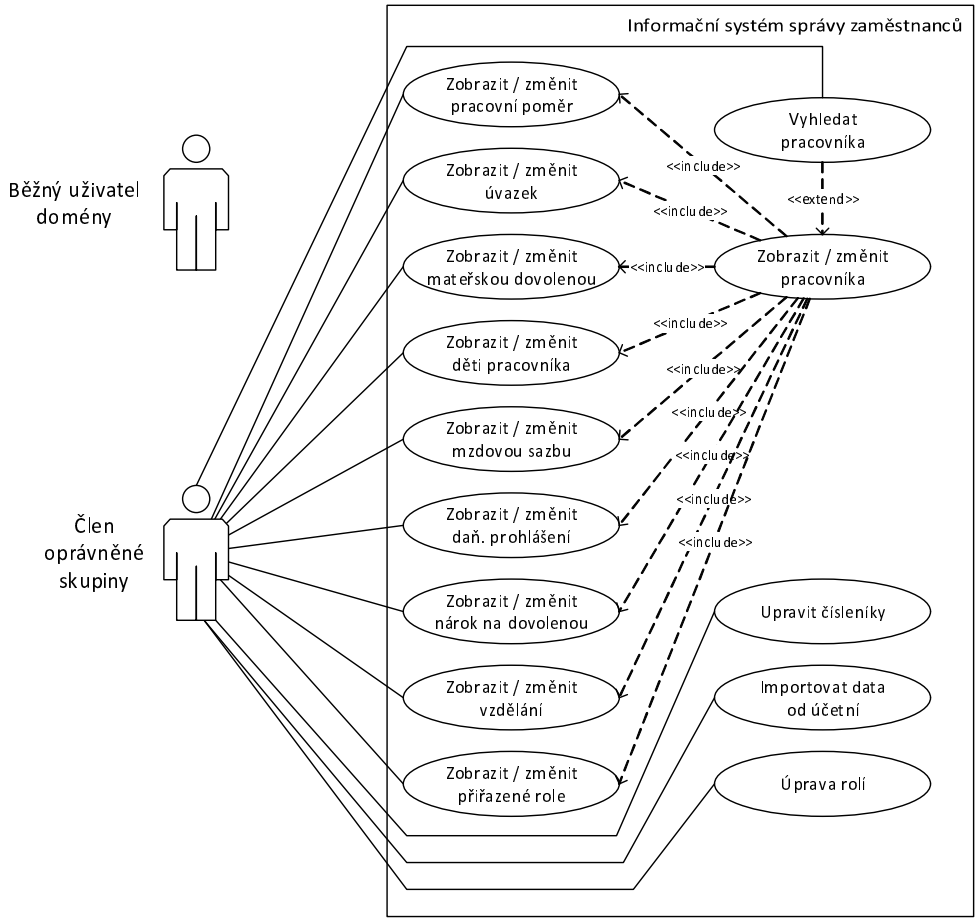
  Uživatel otevře IS a odkáže se na seznam pracovníků. Zde je zobrazena tabulka s pracovníky a základními údaji. Uživatel může zobrazený seznam třídit například dle jména pracovníka, jména nadřízeného, aktuálního oddělení, či typu pracovního poměru. Dále je také možné výsledky řadit podle zobrazených atributů.

- Zobrazit/změnit pracovníka

  Uživatel otevře po vyhledání pracovníka jeho kartu, kde jsou zobrazeny všechny jeho atributy, které je také možné měnit. Dále se zde nachází odkazy na další různé záznamy, které jsou spjaty s entitou pracovníka.

- Zobrazit/změnit pracovní poměr

  Uživatel otevře kartu pracovníka a odkáže se na sekci pracovních poměrů, kde může prohlížet a upravovat existující pracovní poměry, nebo vytvořit nový. Při překrytí datových rozsahů pracovních poměrů systém uživatele na tuto okolnost upozorní.



Obrázek 3.1: Případy užití

- Zobrazit/změnit úvazek

  Uživatel otevře kartu pracovníka a odkáže se na sekci pracovních poměrů, kde jsou pod každým pracovním poměrem zobrazeny jednotlivé úvazky, které se pod něj řadí. Uživatel může tyto úvazky měnit či vytvářet. Při změně datového rozsahu úvazku systém zkontroluje překryv s ostatními úvazky a korespondujícím pracovním poměrem a na případné nesrovnalosti uživatele upozorní.

- Zobrazit/změnit mateřskou dovolenou

  Uživatel otevře kartu pracovníka a odkáže se na sekci mateřských dovolených. Zde se nachází přehled existujících mateřských dovolených, které je možné upravovat, nebo je možné přidat nový záznam. Systém kontroluje případný překryv datových rozsahů a případně uživatele upozorní.

- Zobrazit/změnit děti pracovníka

  Uživatel otevře kartu pracovníka a odkáže se na sekci „děti“, kde se nachází přehled dětí pracovníka. Je možné záznamy upravovat či přidávat nové.

- Zobrazit/změnit mzdovou sazbu

  Uživatel otevře kartu pracovníka a odkáže se na sekci mzdových sazeb. V této sekci se nachází tabulka reálně použitých mzdových sazeb, které je možné upravovat či přidávat, a tabulka plánovaných sazeb, které mají pouze informativní charakter a které je také možné upravovat.

- Zobrazit/změnit daňové prohlášení

  Uživatel otevře kartu pracovníka a odkáže se na sekci daňových přihlášení, kde jsou evidovány fyzické dokumenty daňových prohlášení. Záznamy je možné upravovat, či přidávat nové.

- Zobrazit/změnit nárok na dovolenou

  Uživatel otevře kartu pracovníka a odkáže se na sekci nároků na dovolenou, ve které jsou zobrazeny historické záznamy stavů nároků na dovolenou pracovníka. Ty slouží pouze pro kontrolu a není možné je upravovat. Dále se zde nachází tabulka obsahující záznamy nároků na dovolenou, které je možné měnit či přidávat. Ty slouží pro vnitřní chod společnosti.

- Zobrazit/změnit vzdělání

  Uživatel otevře kartu pracovníka a odkáže se na sekci vzdělání a jazykových znalostí, kde jsou evidovány záznamy absolvovaných škol pracovníka a jeho znalostí cizích jazyků. Všechny záznamy je možné prohlížet a také upravovat nebo přidávat.

- Zobrazit/změnit přiřazené role

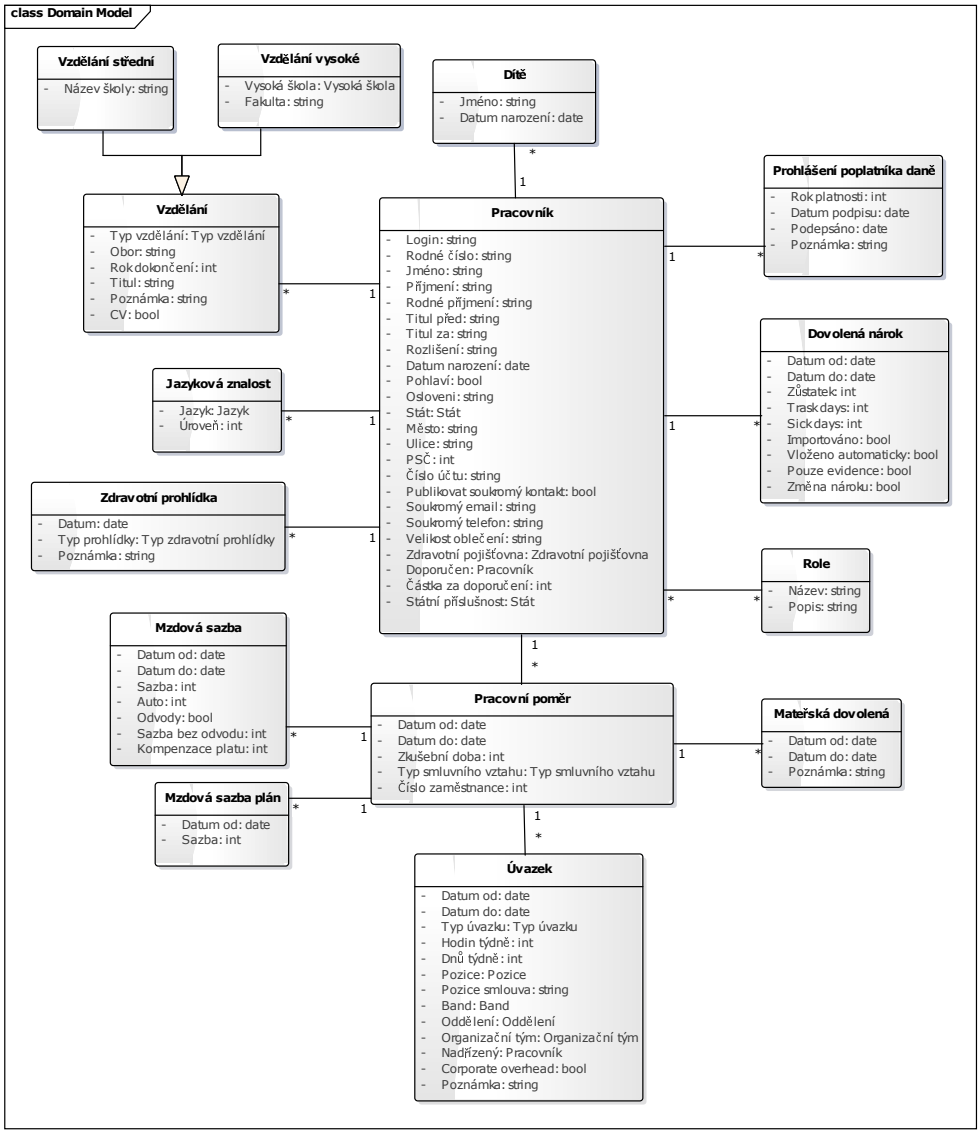
  Uživatel otevře kartu pracovníka, kde je možné vidět, do kterých rolí je pracovník zařazen a případně mu role odebírat, či přidávat nové.

3.2 Doménový model

Doménový model popisuje entity, které jsou součástí řešeného problému, jejich atributy a vzájemné vztahy. V softwarových systémech slouží doménový model jako předloha implementace tříd, se kterými se v systému pracuje.

Navržený doménový model tohoto IS je na obrázku 3.2. Funkce jednotlivých entit i jejich atributů jsou pochopitelné z jejich pojmenování, případně jsou popsány v komentářích zdrojového kódu. Matoucí může být entita Úvazek, která je pojmenována podle entity z původního systému. Tato entita

reprezentuje stav a umístění pracovníka v organizační struktuře společnosti v rámci jednoho pracovního poměru.



Obrázek 3.2: Doménový model

3.3 Zasazení do firemní architektury

Systém pro správu zaměstnanců není ve společnosti jediným systémem. V současnosti se ve společnosti nachází mnoho systémů, které mají na starosti jak evidenci dat, tak i jejich prezentování. Existující systém správy zaměst-

nanců hraje v této struktuře důležitou roli, neboť je zdrojem dat pro některé z těchto systémů. Stejně tak systém správy zaměstnanců využívá dat a služeb ostatních systémů.

3.3.1 Customer relationship management

Customer relationship management (CRM) je ve společnosti, kromě svého primárního účelu správy zákazníků, využíván také jako konzistentní zdroj identifikátorů a atributů (Master data management), které jsou používané nejen v samotném CRM, ale jsou sdíleny i s dalšími systémy. Některá tato data jsou využívána v systému správy zaměstnanců, proto je mezi těmito systémy realizována integrace na datové úrovni.

3.3.2 Active directory

Active directory (AD) je adresářová služba od společnosti Microsoft. Jsou katalogem uživatelů, uživatelských skupin, organizačních jednotek, počítačů a dalších zařízení, které jsou součástí domény (počítačové sítě) společnosti. [7] Slouží především k autentifikaci a autorizaci uživatelů a počítačů ve Windows doméně. Ve společnosti je služba Active directory využívána také jako adresář kontaktů pro službu Microsoft Exchange. Pomocí této služby je pak možné používat adresář kontaktů v emailových klientech či v mobilních zařízeních podporujících protokol Microsoft Exchange.

Adresář obsahuje některé atributy (telefonní číslo, oddělení, nadřízený, pozice, apod.), které jsou zaznamenávány v systému správy zaměstnanců. Proto je ve společnosti implementován automatizovaný proces, který jednou denně porovnává, zda jsou hodnoty atributů uživatelů v Active directory aktuální a případně je aktualizuje. Tento proces je implementován na databázové úrovni.

Active directory je také využíván systémem správy zaměstnanců pro autentizaci uživatelů. Protože bude systém využíván pouze interně ve Windows doméně, odpadá nutnost registrace a přihlašování uživatelů, to je řízeno službami Active directory.

3.3.3 DWH a reporting

Ve společnosti je využíváno robustního řešení MS SQL Server, včetně doplňků Integration services, Analysis services a Reporting services. Pomocí těchto nástrojů lze efektivně shromažďovat data z různých systémů a zpracovávat komplexní datové závislosti.

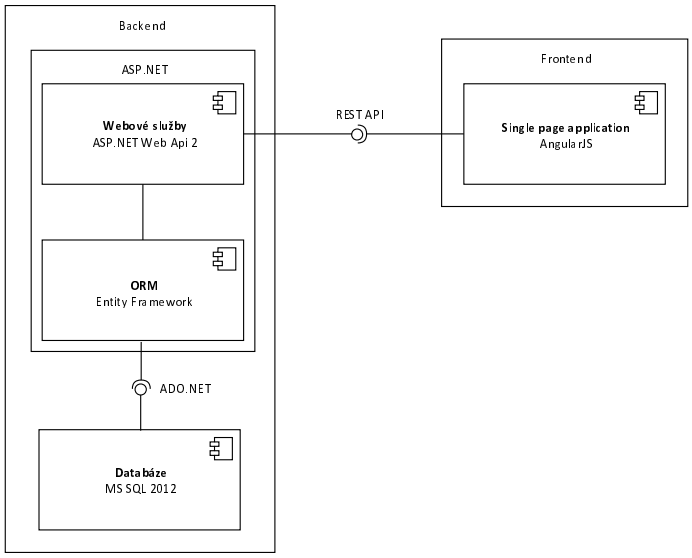
SQL Server Integration Services jsou plaformou pro implementaci ETL (extract, transform, load) procesů, které čtou data z homogenních či heterogenních datových zdrojů, následně je transformují do požadované struktury a formátu k dotazovacím a analytickým účelům. Nakonec data nahrají do společného úložiště – datového skladu.

SQL Server Reporting Services umožňují tvořit strukturované, uživatelsky přívětivé výstupy z databáze, pomocí kterých může uživatel dotazovat a analyzovat velké množství dat.

Množství dat ve firemních systémech je závislé na datech zaměstnanců, proto jsou taktéž zahrnuty do ETL procesu a jsou přítomny v datovém skladu.

3.4 SW architektura

Z hlediska hrubého členění systému na hlavní stavební bloky je systém rozdělen na backendovou a frontendovou část. Backendová část, běžící na serveru, se dále skládá z databázové vrstvy, sloužící k persistenci dat a k provázání dat z jiných systémů, vrstvy obsahující business logiku[2] a vrstvy tvořící rozhraní pro komunikaci s frontendem, případně dalšími systémy. Frontendová část je spouštěna na straně klienta (uživatele) ve webovém prohlížeči a slouží jako uživatelské rozhraní.



Obrázek 3.3: Architektura systému

3.5 Databáze

Nejspodnější vrstva systému je relační databáze. Ta zajišťuje především trvanlivý zápis dat, úpravu a jejich čtení. Zároveň poskytuje prostředí pro konzistentní zápis dat, díky provázanosti ukládaných záznamů pomocí databázových klíčů, které kontrolují přítomnost asociovaných záznamů. Samotná databázová struktura je automaticky generována ORM nástrojem Entity Framework,

[2]definice pravidel popisující jak systém pracuje s daty reprezentující skutečnost

který mapuje objekty programovacího jazyka na záznamy v tabulkách relační databáze.

3.6 Microsoft .NET Framework

Microsoft .NET je platforma pro běh aplikací a knihoven v systému Windows. Tato platforma je velmi rozšířená a používá se pro vývoj jak desktopových aplikací, tak webových aplikací. Pro vývoj webových aplikací existuje framework ASP.NET, který je nadstavbou .NET frameworku. Následně lze použít ještě dalších frameworků, specializovaných dle architektury webové aplikace, například ASP.NET Web Pages, ASP.NET Web Forms, ASP.NET MVC, či ASP.NET Web Api.

Platforma .NET je nezávislá na použitém programovacím jazyce. Jakýkoli podporovaný jazyk, ve kterém je program napsán, je nejprve přeložen do jazyka Common Intermediate Language a až pak je překládán do strojového kódu. Nejčastěji používané jazyky pro vývoj na platformě .NET jsou C# a Visual Basic.

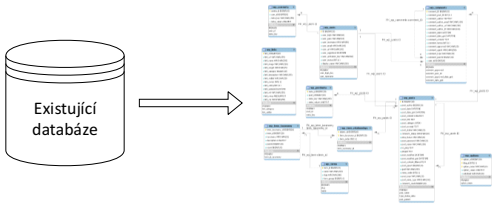
3.7 Objektově relační mapování

Další vrstva, obsahující definici doménového modelu, zprostředkovává jeho namapování do databázových tabulek pomocí ORM nástroje Entity Framework. ORM framework je důležitou součástí persistenčního mechanismu aplikace. Zapouzdřuje všechny běžné operace potřebné pro persistenci objektů, jako je vytváření, čtení, upravování a mazání. Odpadá tím potřeba opakovaného psaní databázových dotazů a transformace sloupců tabulek na atributy objektů programovacího jazyka a opačně. ORM vrstvu je možné implementovat vlastní, pouze pro potřeby daného kontextu. Je ale většinou výhodnější využít již existujících řešení, která jsou obecná a obsahují mechanismy, které efektivně zacházejí se závislostmi mezi objekty a vytvářejí efektivní databázové dotazy.

Entity Framework je open-source ORM nástroj součástí .NET frameworku. Alternativně může být použit framework NHibernate, který je srovnatelně rozsáhlým ORM frameworkem. [8] Entity Framework se opírá o doménový model zvaný Entity data model (EDM) a nabízí tři možné strategie, jak tento model vytvořit. [9]

První strategie, „database first“ (obrázek 3.4), je vytvoření EDM podle struktury existující databáze. Protože struktura databáze je typicky definována dle doménového modelu, je tento způsob v podstatě reverzním inženýrstvím doménového modelu. Vývojové prostředí Visual Studio obsahuje průvodce, ve kterém se lze jednoduše připojit k databázi a zvolit databázové objekty k vytvoření EDM. Těmito objekty mohou být tabulky, databázové pohledy i funkce. Entity Framework následně vygeneruje třídy, včetně jejich

vlastností a také závislostí, které vyčte z definic tabulek a klíčů mezi nimi. V případě komplikovanějších konceptů, například dědění entit, je možné upravit EDM ručně. Pro model tříd programovacího jazyka, model databáze a vztah mezi těmito modely jsou vygenerovány definice v jazycích CSDL, SSDL a MSL a následně uloženy do konfiguračního souboru. [10]



Obrázek 3.4: Entity Framework database first

Další možnou stratagií je „model first“ (obrázek 3.5), tedy vytvoření EDM přímo v designeru, který je součástí Visual Studia. Entity Framework je schopný následně vygenerovat a spustit skripty potřebné k vytvoření databáze. V designeru lze navrhovat entity, vztahy, dědění, standardní hodnoty či některá integritní omezení.

Obrázek 3.5: Entity Framework model first

Třetí podporovanou strategií, „code first“, je napsání tříd programovacího jazyka dle návrhu doménového modelu a automatické vygenerování databáze podle těchto definic. Tato strategie nabízí možnost generování inkrementálních skriptů, pomocí kterých lze upgradovat či downgradovat databázi na určitou verzi, vytvořenou v průběhu vývoje doménového modelu. Při vytvoření nové verze (migrace) vytvoří Entity Framework novou migrační třídu, obsahující metody `Up()` a `Down()`, které obsahují volání metod API Entity Frameworku. Tyto metody generují DDL skripty[3] pro úpravu databáze. Alternativně je možné tuto strategii využít i v případě existující databáze[4]. Vytvoření tříd probíhá obdobným způsobem jako v případě „databse first“, nadále je však možné databázi upravovat prostřednictvím migrací.

[3]DDL (data definition language) je jazyk definující strukturu databáze.

[4]V tomto případě je „code first“ zavádějící název, ale je pro tento způsob také používaný.

Další zajímavou problematikou ORM technik je mapování dědičnosti entit. Existují tři různé způsoby, jak dědičnost řešit: table par hierarchy (TPH), table per type (TPT) a table per concrete class (TPC). [11]

Přístup TPH mapuje celou hierarchii dědičnosti do jedné databázové tabulky, která obsahuje sloupce všech atributů všech tříd hierarchie. Navíc tabulka obsahuje jeden další sloupec, tzv. diskriminátor, jež označuje, ke které třídě hierarchie daný databázový záznam patří. TPH je jednoduchým a výkonnostně efektivním řešením. Všechny záznamy jsou v jedné tabulce a není třeba žádného propojování s asociovanými záznamy. Na druhou stranu vytváří velké tabulky, které jsou z velké části prázdné, protože každý záznam využívá vždy jen některé sloupce. Další nevýhoda je komplikovanější způsob validace dat, protože aby se daly v jedné tabulce uložit záznamy různých tříd, je nutné, aby byly všechny sloupce podtříd nepovinné (mohou obsahovat prázdnou hodnotu), přestože ve skutečnosti jsou tyto atributy povinné.

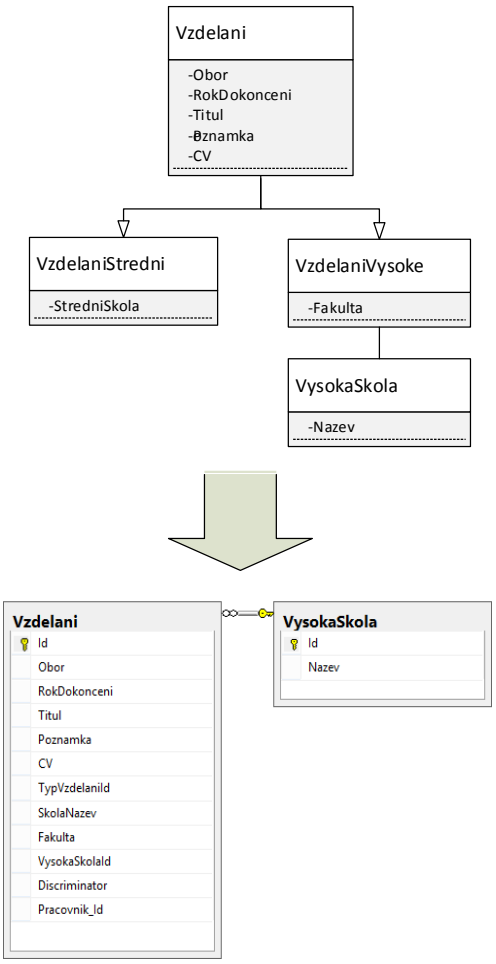
TPT mapuje každou entitu hierarchie dědičnosti do samostatné tabulky. Pro jeden záznam v tabulce nadtřídy existuje právě jeden záznam v jedné z tabulek podtřídy, pokud je nadtřída abstraktní. V opačném případě existuje pro záznam nadtřídy jeden, nebo žádný záznam podtřídy. Tento způsob působí jako správný objektově orientovaný přístup. V případě přidání nové podtřídy není nutné nic měnit, stačí vytvořit jednu novou tabulku. Také je možné kontrolovat integritní omezení atributů standardním způsobem.

Třetí přistup TPC mapuje každou třídu do jedné tabulky, včetně všech jejích vlastních atributů a všech atributů nadtříd. Tento přístup využívá některých technik z obou předešlých přístupů. Problém ale může nastat v případě asociace nadtřídy s dalšími entitami doménového modelu. Pokud je tato nadtřída ve vztahu s jinými entitami nadřazenou entitou, nelze její podtřídy použít jako nadřazené entity, i když pro ně tento vztah platí také.

V modelu tříd tohoto systému je pouze jeden případ dědění. Jedná se o entitu Vzdělání, která představuje absolvovanou školu pracovníka. Vzdělání může být buďto středoškolské, u kterého je volným textem zapsán název školy, nebo vysokoškolské, kde je název vysoké školy vybírán z číselníku. Společnými atributy těchto dvou typů jsou rok dokončení, obor, titul a další. Pro jednoduchost této struktury bylo použito TPH mapování, ukázka je na obrázku 3.6.

Pro práci s entitami je v rámci Entity Frameworku vytvořena třída, potomek třídy `DbContext`, která obsahuje množiny `DbSet<>` záznamů jednotlivých entit. Tyto množiny obsahují metody, díky kterým je možné dotazovat, zapisovat a mazat záznamy. Pro dotazování dat je k dispozici technologie LINQ to entities, která dodává podporu dotazování prostřednictvím dotazovacího jazyka LINQ. Ten je standardně integrovaný v .NET frameworku.

Entity Framework obsahuje mechanismus vytváření tzv. proxy objektů, které jsou odvozené od tříd doménového modelu a vystupují jako jejich zástupci. Tato technika umožňuje rozšiřovat vlastnosti objektu například o sledovaní stavu dané instance nebo podporu „lazy loading“. „Lazy loading“ je



Obrázek 3.6: ORM mapování dědění přístupem TPH

strategie načítání dat až v momentě, kdy jsou skutečně potřeba. Tato strategie je využívána pro plynulejší běh systému, v případě Entity Frameworku je použita například pro načítání asociovaných entit. [12]

3.8 Servisní vrstva

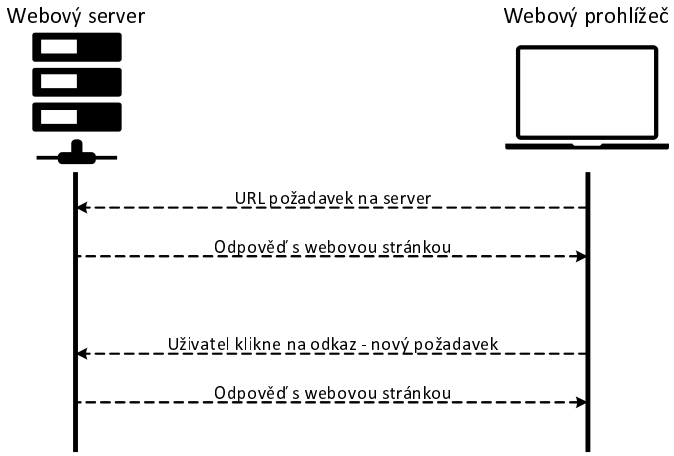
Servisní vrstva zapouzdřuje business logiku a poskytuje služby využívané vrstvou webových služeb. Požadavky z frontendu či jiných systémů, které jsou přijímány vrstvou webových služeb, jsou delegovány do této vrstvy, kde se vykonají náležité kroky k jejich vyřízení. Následně vrátí informaci o výsledku zpět do vrstvy webových služeb, která na požadavek odpoví.

V této aplikaci je využíváno servisní vrstvy především ke kontrole integritních omezení, o kterých je psáno ve specifikaci systému (sekce 2.3), například

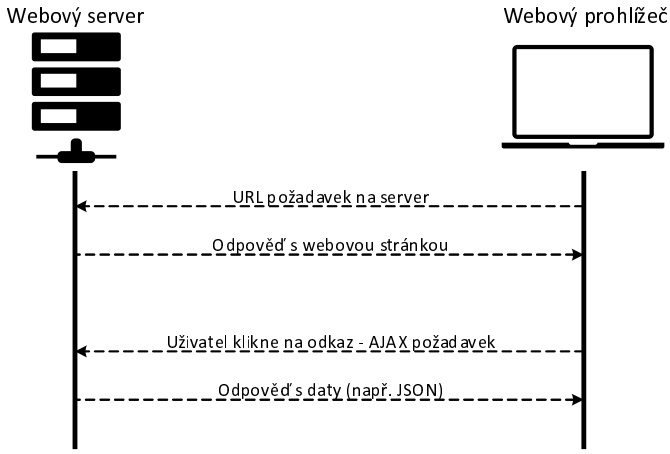
kontrola, zda se navzájem nepřekrývají rozsahy dat jednotlivých pracovních poměrů, či unikátnost některých atributů pracovníka.

3.9 Vrstva webových služeb

Webové služby jsou dnes často používanou technikou pro přenos informací mezi systémy. V případě webové aplikace se používají k poskytování dat a opačně k přijímání dat ke zpracování. Jsou základním stavebním kamenem dynamických webových stránek, které při aktualizaci obsahu nevyžadují zaslání kompletní nové webové stránky (obrázek 3.7), ale pouze data nutná k aktualizaci obsahu (obrázek 3.8).



Obrázek 3.7: Komunikace server-klient u klasických webových stránek



Obrázek 3.8: Komunikace server-klient u dynamických webových stránek

URI	GET	POST	PUT	DELETE
URI KOLEKCE, NAPŘ. HTTP://BASE.URL/API/PRACOVNIK	Vrátí všechny položky kolekce	Vytvoří novou položku	Nahradí kolekci jinou kolekcí, není typicky používáno	Smaže všechny položky kolekce, není typicky používáno
URI POLOŽKY, NAPŘ. HTTP://BASE.URL/API/PRACOVNIK/4	Vrátí položku dle identifikátoru	Není používáno	Nahradí položku daného identifikátoru	Smaže položku daného identifikátoru

Tabulka 3.1: Přehled použití metod REST rozhraní

Pro tento způsob vývoje webových aplikací se používá JavaScriptového kódu, který zasílá asynchronní požadavky na server a zpracovává odpovědi. Požadavky a odpovědi serveru nesou často data, která bývají typicky v JSON či XML zápisu. Proto se tato technika označuje jako AJAX (Asynchronous JavaScript and XML).

Ustálenou strukturou webových služeb pro účely obsluhy webové aplikace se stalo REST (Representational State Transfer) rozhraní. Využívá protokolu HTTP, tedy stejného, jako se používá pro komunikaci serveru a klienta v případě klasických webových stránek. REST využívá HTTP metod GET, PUT, POST a DELETE, které zprostředkovávají tzv. CRUD[5] operace, tedy vytvoření, čtení, úpravu a mazání dat. [13] Příklad použití REST rozhraní je uvedený v tabulce 3.9.

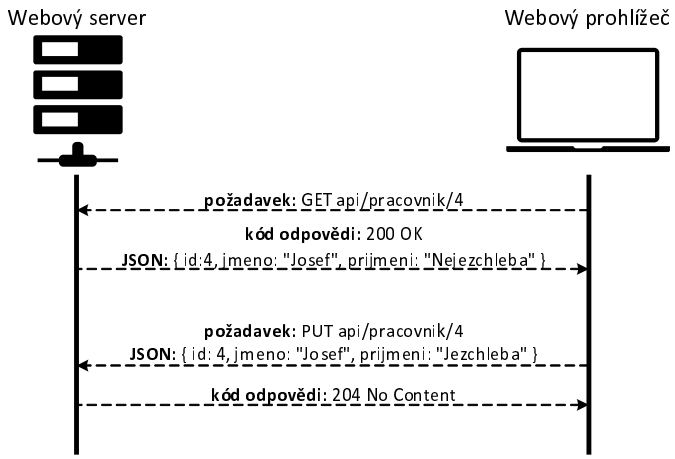
Komunikace prostřednictvím protokolu HTTP je realizována zasíláním zpráv, majících definovanou strukturu. Zpráva obsahuje, mimo jiné, metodu požadavku (například GET, POST), v případě zprávy posílané na server, či kód odpovědi, v případě zprávy posílané směrem ke klientu, a dále tělo zprávy, obsahující data. Příklad komunikace je znázorněný v obrázku 3.9.

Framework ASP.NET Web Api 2 podporuje tvorbu webových služeb na platformě ASP.NET. Obsahuje knihovny obalující obsluhu HTTP komunikace, tvorbu HTTP odpovědí, serializaci a deserializaci objektů a další. Web Api také zjednodušují vývoj použitím techniky „convention over configuration“, která předpokládá použití definovaných konvencí v kódu, například jmenné konvence, a využití těchto konvencí k automatické konfiguraci programu.

Konkrétním příkladem konfigurace dle jmenné konvence ve frameworku ASP.NET Web Api 2 je směrování HTTP volání. Webové rozhraní je v tomto frameworku programováno prostřednictvím speciálních tříd, tzv. controllerů, které zpracovávají přijatá volání, prostřednictvím servisní vrstvy vykonávají požadované operace, a vracejí odpověď. Konvencí je v tomto případě uložit tyto třídy do adresáře „Controllers“ a pojmenovat je s postfixem „Controller“. Třídy následně obsahují veřejné metody, které obsluhují jednotlivé metody HTTP volání. Tyto metody se dle konvence pojmenovávají s prefixem názvu dané HTTP metody, tady například s prefixem Get, Post atd. [14]

Například struktura controlleru pro operování se záznamy pracovníků podniku vypadá následovně.

[5] create, read, update, delete



Obrázek 3.9: Příklad komunikace prostřednictvím REST rozhraní při změně příjmení pracovníka s identifikátorem 4

```
public class PracovnikController : HRMController {
        private PracovnikService pracovnikService = new PracovnikService();

        // GET api/Pracovnik
        public IEnumerable<PracovnikDTO> GetPracovnici()
        { ... }

        // GET api/Pracovnik/5
        [ResponseType(typeof(PracovnikDTO))]
        public IHttpActionResult GetPracovnik(int id)
        { ... }

        // PUT api/Pracovnik/5
        public IHttpActionResult PutPracovnik(int id, PracovnikDTO pracovnikDTO)
        { ... }

        // POST api/Pracovnik
        //[ResponseType(typeof(PracovnikDTO))]
        public IHttpActionResult
            PostPracovnik(PracovnikDTO pracovnikDTO)
        { ... }

        // DELETE api/Pracovnik/5
        [ResponseType(typeof(PracovnikDTO))]
        public IHttpActionResult DeletePracovnik(int id)
        { ... }

}
```

Požadavky jsou směrovány dle URL a HTTP metody na metody třídy PracovnikController způsobem popsaném v tabulce 3.9.

K přenosu dat prostřednictvím REST rozhraní je využito návrhového vzoru Data transfer object (DTO). Tento návrhový vzor využívá k přenosu dat mezi systémy jednoduché objekty zkonstruované dle tříd, které slouží pouze k tomuto účelu. Použitím těchto objektů se zabrání odhalení objektů pou-

URI	GET	POST	PUT	DELETE
HTTP://BASE.URL/API/PRACOVNIK	GetPracovnici()	PostPracovnik(...)		
HTTP://BASE.URL/API/PRACOVNIK/3	GetPracovnik(...)		PutPracovnik(...)	DeletePracovnik()

Tabulka 3.2: Přehled REST metod

žívaných uvnitř systému a také jasně definují strukturu dat, které je třeba přenést. Serializace složitých zástupných proxy objektů Entity Frameworku působí často komplikace. Využitím návrhového vzoru DTO tento problém řeší.

V kontextu webových aplikací, jejichž součástí bývá větší množství souborů, typicky JavaScriptového kódu nebo kaskádových stylů, je vhodné využít technik bundlingu a minifikace, které jsou součástí ASP.NET Web Api frameworku. Velké množství souborů, které je třeba při načítání stránky stáhnout, může způsobit větší prodlevu při iniciálním načítání SPA. Proto se využívá bundling, který vytvoří předem definované balíčky těchto souborů a zredukuje tím počet požadavků na server. Proces minifikace dokáže současně zmenšit velikost těchto souborů řádově o desítky procent. Zmenšení velikosti funguje na principu zjednodušování kódu na takovou úroveň, při které je zdrojový soubor pro programátora již prakticky nečitelný, ale jeho logický význam zůstává nezměněný. Během minifikace se odstraní všechny komentáře, bílé znaky (whitespaces) a názvy proměnných se nahradí krátkými řetězci.

3.10 Single page application

Single page application (SPA) je klientská část webová aplikace, která je z vývojářského pohledu pouze jedna webová stránka, jejíž viditelný obsah je dynamicky měněn a řízen kódem, který je součástí aplikace na straně klienta. Při prvotním načtení aplikace je serverem poskytnuta kompletní webová aplikace, včetně veškerého obsahu (HTML, JavaScript, CSS) potřebného pro její běh. Webová stránka se v žádném momentu nenačítá znovu a veškerý obsah je načítán prostřednictvím asynchronních požadavků na server. Koncept SPA vytváří u webové aplikace dojem desktopové aplikace, díky uživatelsky přívětivějšímu a interaktivnějšímu prostředí s rychlými odezvami. Pro vývoj SPA lze použít jeden z mnoha frameworků, mezi největší patří AngularJS, Backbone.js a Ember.js.V tomto projektu je použitý framework AngularJS.

3.10.1 Představení AngularJS

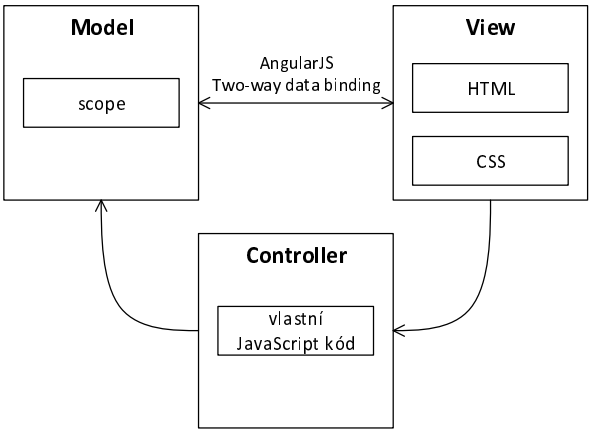
Framework AngularJS [15] je open-source projekt od společnosti Google a je jedním z nepoužívanějších frameworků pro tvorbu single page applications. Jeho přímými konkurenty jsou frameworky Backbone.js a Ember.js. AngularJS má z nich mezi vývojářskými komunitami nejsilnější pozici, je nejpopulárnější na GitHubu, známém hostingu open-source projektů, má největší

Metrika	AngularJS	Backbone.js	Ember.js
Hvězd na Githubu	27.2k	18.8k	11k
Modulů třetích stran	800	236	21
StackOverflow otázek	49.5k	15.9k	11.2k
YouTube výsledků	~75k	~16k	~6k
GitHub přispěvatelů	928	230	393

Tabulka 3.3: Srovnání popularity SPA frameworků [1]

množství různých rozšíření a nejrozsáhlejší komunitní podporu na vývojářských fórech. [1]

AngularJS je MVC framework napsaný v jazyce JavaScript a je spouštěný ve webovém prohlížeči na straně klienta. Použití není omezeno pouze pro webové prohlížeče, existují například různé Windows 8 aplikace, napsané za použití tohoto frameworku, ale nejčastější je vývoj SPA.



Obrázek 3.10: AngularJS MVC

MVC (model-view-controller) je návrhový vzor, logicky oddělující tři základní části programu. Jednou úrovní návrhového vzoru MVC je tzv. model, který spravuje entity a data, která jsou v aplikaci používána. V AngularJS je model reprezentován objektem scope, který se pevně váže k view a je spravován controllerem.

Vrstva view je část aplikace, kterou vidí uživatel. Prezentuje aplikaci uživateli a zprostředkovává interakci. V AngularJS je tato část tvořena HTML stránkou, do jejíž struktury jsou zakomponovány ovládací prvky aplikace, prostřednictvím AngularJS direktiv. Direktivy jsou značky (HTML elementy, atributy, CSS třídy či komentáře), kterými se označují části webové stránky a tím udávají AngularJS HTML kompilátoru místo, kam vložit definované chování. Mezi modelem a view zajišťuje framework „two-way data binding“, což je obousměrné provázání hodnot uložených ve scope a zobrazených ve

view. To znamená, že kdykoli je hodnota ve view změněna, je tato změna automaticky promítnuta do scope, a opačně. Při vývoji tak odpadá nutnost psát poměrně objemné části kódu, který by toto řešil.

Poslední vrstva je controller, který obsahuje business logiku a přímo reaguje na akce uživatele. V tomto frameworku tvoří controller JavaScript kód, který, mimo jiné, definuje scope a manipuluje s daty v něm obsaženými. Zároveň zajišťuje komunikaci s rozhraním backendu.

Veškerý kód napsaný v tomto frameworku je zapouzdřován do modulů, ze kterých je výsledně celá aplikace sestavena. Moduly jsou samostatné, znovupoužitelné celky kódu, které jsou odpovědné za konkrétní funkcionalitu. V konfiguraci hotové aplikace jsou deklarovány všechny použité moduly a jejich závislosti. Tento přístup podporuje nízkou provázanost kódu, díky které je aplikace snadněji udržovatelná a lépe čitelná. Modularitu kódu využívá návrhový vzor Dependency injection, pro který je ve frameworku vestavěná podpora právě pomocí konfigurace závislostí. AngularJS dynamicky vkládá (injektuje) objekty zdrojových modulů do cílových modulů, mezi kterými je definovaná závislost. Kdykoli jsou tyto objekty v cílovém modulu použity, framework je automaticky dosadí na základě názvu, pokud je totožný s názvem objektu ve zdrojovém modulu.

3.10.2 Návrh single page application

V tomto projektu je SPA koncovou komponentou, která má na starosti komunikaci s REST rozhraním backendového řešení a konečně interakci s uživatelem.

Důležitým stavebním kamenem je modul využívající HTTP protokolu pro komunikaci s backendem. AngularJS nabízí vestavěnou službu pro obsluhu HTTP, prostřednictvím které je možné operovat s protokolem na nízké úrovni. Pro komunikaci s REST rozhraním lze tuto službu využít, protože je však REST rozhraní velmi rozšířenou technikou, existují moduly, které s tímto protokolem zacházejí pravě takovým způsobem, který je pro komunikaci s REST rozhraním typický. Takovým modulem je například ngResource, který je též součástí frameworku. Ještě širší nadstavbou je modul třetí strany Restangular, který je použitý v tomto projektu. Restangular skládá požadavky posílané na server podle volaných metod, díky tomu nemusí vývojář vytvářet požadavky ručně. Pokud na straně serveru existuje výjimka, která porušuje obecně uznávanou strukturu REST rozhraní, lze definovat i vlastní požadavky.

S daty následně operuje vrstva controller. Tato vrstva se skládá ze souboru controllerů, každý z nich spravuje nějakou konkrétní oblast aplikace. Často jsou odpovědné za operace s nějakou konkrétní entitou a jsou navázány na konkrétní view či direktivy. V controllerech se také manipuluje s daty v objektu scope, který je navázán na view pomocí “two-way binding”.

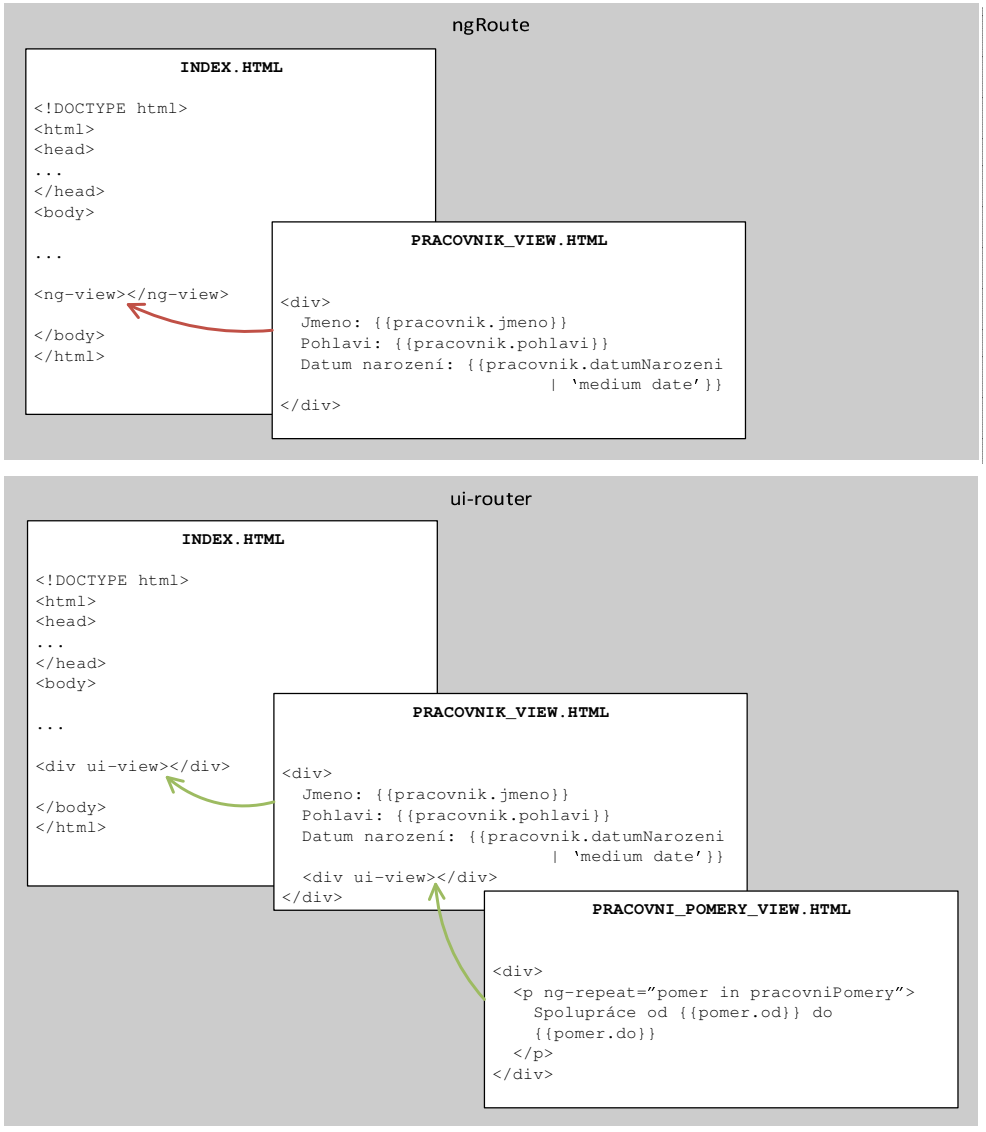
Velmi důležitou komponentou aplikace je systém směrování. V případě statických webových stránek je směrování na jednotlivé stránky přímočaré,

vychází z adresářové struktury, ve které jsou soubory představující jednotlivé webové stránky uložené. V kontextu SPA je potřeba tuto směrovací strukturu uměle vytvořit, protože aplikaci tvoří jediná webová stránka. Pokud by směrovací struktura chyběla, URL adresa v prohlížeči by během průchodu aplikace uživatelem zůstávala nezměněna. Pokud by se uživatel například pokusil aktualizovat stránku, na které se právě nachází, ocitl by se zpátky na základní stránce. Nebylo by také možné používat navigačních tlačítek prohlížeče „zpět“ a „vpřed“, ani by nebylo možné URL adresu dané stránky ukládat či někomu posílat. AngularJS obsahuje modul ngRoute, který tento problém řeší. V konfiguraci tohoto modulu je možné definovat hierarchii směrování a pevně svázat konkrétní view a konkrétní controllery s relativní URL. URL může obsahovat parametry, které se následně předají do controlleru. Například URL „base.url/Pracovnik/32“ obsahuje parametr „32“, který identifikuje konkrétního pracovníka. Controller následně vyhledá pracovníka dle tohoto identifikátoru a zobrazí jeho informace ve view.

View jsou při průchodu webovou aplikací dynamicky dosazovány do hlavní webové stránky, té, která je iniciálně načtena při spuštění aplikace. Ta obvykle obsahuje záhlaví, zápatí, menu, případně další obecné prvky. Při přechodu mezi různými view (stránkami) je nahrazován obsah předchozího view obsahem nového view, zbytek stránky zůstává stejný. AngularJS v základu podporuje pouze takto ploché navigování skrz view.

Modul třetí strany ui-router, součást open-source knihovny AngularUI, je jednou z možných alternativ k ngRoute. Velkým rozdílem tohoto řešení oproti ngRoute je organizování navigační struktury okolo stavů, ne okolo adres URL. Ke stavům se mohou a nemusí vázat URL, stejně tak se ke stavům mohou a nemusí vázat i view. Dalším velkým přínosem je podpora vnořených view. [16] Rozdíl noření view oproti ngRoute je ilustrován na obrázku 3.11. Díky této funkcionalitě je možné se pomocí URL odkazovat na stav, kdy je uživatel zanořen hlouběji ve struktuře aktuálně zobrazené stránky. Modul ui-router je v tomto IS použitý. Při nahrazování modulu ngRoute tímto modulem se projeví síla návrhového vzoru Dependency injection, díky kterému stačí pouze přepsat název modulu v konfiguraci aplikace.

K tvorbě uživatelského prostředí je použito frameworku Bootstrap. Bootstrap je velmi populární open-source framework, obsahující množství předdefinovaných stylů v jazyce CSS, které lze aplikovat na HTML elementy. Obsahuje také hotové komponenty, které nejsou součástí definice HTML, například záložky (taby), modální okna, navigační lišty apod. Součástí frameworku jsou také skripty, které řídí zobrazování webové aplikace na různých zařízeních, například notebooku, tabletu či mobilním telefonu, které mají různě velké zobrazovací plochy a také různé způsoby ovládání. [17] Uživatelské prostředí aplikace dále tvoří modul ui-bootstrap, další z knihovny AngularUI. Tento modul obsahuje definice mnoha prvků uživatelského prostředí (např. tlačítka, kalendář, záložky) a korespondujících AngularJS direktiv, pomocí kterých je možné tyto prvky snadno použít.



Obrázek 3.11: Rozdíl noření view za použití modulů ngRoute a ui-router

Kapitola 4

Implementace

4.1 IDE a další nástroje

4.1.1 Visual Studio

Pro vývoj systémů na platformě .NET poskytuje Microsoft vývojové prostředí Visual Studio. Poskytuje podporu ASP.NET Web Api 2 i Entity Frameworku, frameworků použitých při vývoji backendové části této webové aplikace.

Vývoji webových služeb prostřednictvím ASP.NET Web Api napomáhá Visual Studio automatickým generováním controllerů strukturovaným dle šablon odpovídajících konvencím REST rozhraní. Je možné specifikovat konkrétní entitu, spravovanou Entity Frameworkem a nechat si vygenerovat všechny metody pro čtení, vytváření, úpravy a mazání této entity, včetně obsluhy persistenčních metod Entity Frameworku.

Visual studio nabízí velmi užitečnou funkci umožnující spuštění backendové části aplikace na integrovaném webovém serveru IIS Express a zároveň, během používání SPA, kód ladit. Pro použití IIS Express není třeba nic konfigurovat a Visual Studio se o inicializaci postará samo. K dispozici je během ladění také Immediate window, ve kterém je možné v kontextu aktuálně spouštěného kódu spouštět příkazy a vyhodnocovat výrazy.

Během vývoje je velmi nápomocný Intellisense, nástroj pro automatické napovídání a doplňování kódu. Napovídání plně funguje i v případě složitějších syntaktických konstrukcí, například lambda výrazů či LINQ dotazů. Intellisense zrychluje psaní kódu a redukuje počet překlepů.

Visual Studio dále vývojáři pomáhá například upozorňováním na některé chybné konstrukce ještě dříve, než je program zkompilován, automatickým doplňováním použitých namespaces, generováním deklarací nových metod a dalšími způsoby.

Součástí Visual Studia je správce knihoven NuGet. Pomocí tohoto nástroje se výrazně zjednodušuje instalace a udržování různých rozšíření aplikace, kromě samotného stažení a začlenění do projektové struktury zajišťuje

v některých případech také configuraci projektu.

4.1.2 Sublime text

Vývoj SPA byl psán v pokročilém textovém editoru Sublime text. Tento editor je zaměřený na psaní kódu a stal se oblíbeným nástrojem mnoha vývojářů. Je velmi svižný, nenáročný a poskytuje množství užitečných funkcí úpravy textu, zvláště hromadných úprav, které nejsou často k vidění ani v IDE. Pohyb mezi soubory a uvnitř dlouhých bloků kódu je velmi rychlý, díky vyhledávacím oknům, které dokáží vyhledávat i mezi jednotlivými metodami v kódu. Sublime text je snadno rozšířitelný, k dispozici je velký výběr rozšíření třetích stran. Pro účely tohoto projektu bylo využito rozšíření pro napovídání AngularJS direktiv do HTML kódu, či metod AngularJS knihovny.

4.1.3 Chrome

Webový prohlížeč Chrome obsahuje kvalitní nástroje pro vývojáře webových aplikací, mezi které patří pokročilý prohlížeč a editor zdrojového HTML kódu stránky a souvisejících CSS stylů, JavaScript debugger, prohlížeč HTTP požadavků a odpovědí a další. Pro ladění aplikace napsané v AngularJS je možné Chrome rozšířit například o nástroje AngularJS Batarang, nebo ng-inspector for AngularJS, které umí zobrazovat hierarchii scope a kód controllerů asociovaných s aktuálně zobrazenou stránkou. Pro ladění a testování backendového REST rozhraní existují také rozšíření do Chrome, pomocí kterých lze zasílat ručně psané požadavky na rozhraní a prohlížet odpovědi. Při vývoji tohoto projektu bylo použito rozšíření Advenced Rest Client.

4.1.4 Firefox

Prohlížeč Firefox má integrovaný velmi vyspělý editor CSS stylů. Disponuje funkcionalitou pro napovídání názvů elementů a tříd při psaní selektorů, i napovídání při definici samotných stylů. Obsahuje 3D prohlížeč elementů webové stránky, kde lze velmi dobře pozorovat strukturu kontejnerů nesoucích obsah stránky a po kliknutí na konkrétní kontejner se v editoru vyhledá odpovídající HTML kód. Dále nabízí testování responzivního designu stránky, při kterém lze upravovat velikost zobrazovací plochy a také simulovat klepnutí na displej či otáčení mobilního zařízení.

4.2 Použité servery

4.2.1 MS SQL Server

MS SQL Server je pokročilý relační databázový server vyvinutý společností Microsoft, schopný efektivně uchovávat a zpracovávat i velké objemy dat. K databázi je v tomto systému přistupováno prostřednictvím ADO.NET.

Použití této databáze je dáno přítomností MS SQL Serveru ve společnosti. Pro účely vyvíjeného systému by stačil i méně náročný databázový server, například MySQL.

4.2.2 Internet Information Services

Internet Information Services (IIS) je webový server vyvinutý společností Microsoft. Kromě protokolů HTTP a HTTPS podporuje i další běžné protokoly pro internetovou komunikaci, například FTP, FTPS, SMTP a NNTP.

4.3 Verzování

Verzování je jedna z podpůrných aktivit při vývoji software. Odpovídá za správu všech softwarových položek[6], ze kterých se kompletní produkt skládá. [18] Verzovací systém je řízeným úložištěm těchto položek a všech jejich historických změn. Jeho použití je při vývoji v kolektivu vývojářů prakticky nezbytné, ale i v případě tohoto projektu, kde je pouze jeden vývojář, je verzovací systém důležitý. Slouží k zálohování historických stavů vývoje a také jednotlivých hotových sestav, ke kterým může vzniknout potřeba se vrátit.

V tomto projektu je využito systému Team Foundation Server. Tento systém je rozsáhlou platformou pro vývoj softwaru, obsahuje nástroje pro verzování (Team Foundation Version Control nebo Git), automatizované sestavování software, testování, a další. Obsahuje také podporu pro řízení projektu, řízení změn či správu dokumentů. Tím pokrývá všechny důležité aktivity, běžně používané pro správu životního cyklu software.

4.4 Problémy při implementaci

4.4.1 Směrování AJAX požadavků

Původní záměrem bylo nasazení frontendu a backendu odděleně na dvě různá umístění. Při otevření aplikace uživatelem by byl z jednoho zdroje stažen kód frontendu, který by komunikoval s druhým zdrojem, jehož adresu by měl v sobě zapsanou.

V oblasti webových aplikací však existuje bezpečnostní opatření same-origin policy, které je standardně vynucováno webovými prohlížeči. Toto opatření vyžaduje, aby webová aplikace zasílala AJAX požadavky na stejné místo (kombinace URI, hostname a číslo portu), z nějž byla načtena. Důvodem tohoto omezení je zabránění vyřizování požadavků potencionálně nebezpečných skriptů třetích stran. [19]

Proto bylo potřeba nastavit webové rozhraní backendové části, aby na konkrétní požadavek odpovědělo poskytnutím single page appliacion a na ostatní

[6] identifikovatelná část softwarového produktu

požadavky reagovalo prostřednictvím REST rozhraní. Tím se zajistí stejný původ samotné aplikace i zdrojů dat.

Problém byl vyřešen pomocí frameworku ASP.NET MVC, prostřednictvím kterého lze z controllerů odesílat HTML stránky vygenerované podle předdefinovaných šablon. Bylo tedy třeba vytvořit HTML šablonu obsahující úvodní stránku aplikace a do ní vložit zdrojové soubory single page application. Následně bylo potřeba nastavit v konfiguraci backendového rozhraní standardní přesměrování na controller, poskytující tuto stránku, a ostatní požadavky směrovat na controllery REST rozhraní. O směrování požadavků na zdrojové soubory (JS, CSS, ...) se není třeba starat, k těm webový server standardně poskytuje přímý přístup.

```
public static void RegisterRoutes(RouteCollection routes)
{
    //Website routes
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        name: "Default",
        url: "{controller}/{action}/{id}",
        defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
    );
}

public static void Register(HttpConfiguration config)
{
  // Web API routes
  config.MapHttpAttributeRoutes();

  config.Routes.MapHttpRoute(
      name: "DefaultApi",
      routeTemplate: "api/{controller}/{id}",
      defaults: new { id = RouteParameter.Optional }
  );
}
```

4.4.2 Reprezentace časových hodnot

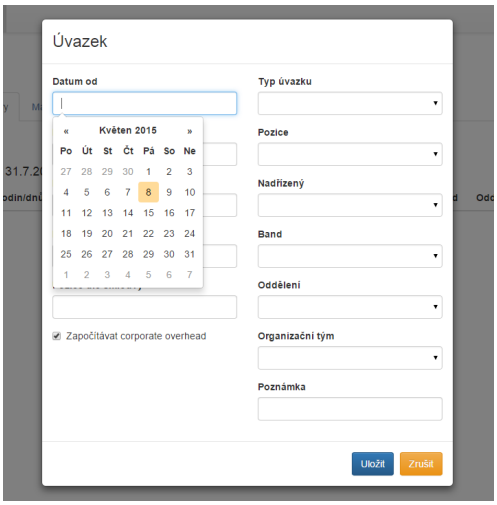
IS pracuje s časovými údaji, proto bylo potřeba se zamyslet nad způsobem ukládání a reprezentace těchto hodnot. Zvláště v tomto případě, kdy jsou hodnoty předávány mezi nehomogenními systémy[7]. Časové údaje mohou být reprezentovány různými způsoby, které se liší národnostními zvyklostmi a také místem, kde je časová značka pořízena. Proto byl vytvořen mezinárodní standard ISO 8601 [20], který definuje formát zápisu času, včetně údaje o časové zóně. Zápis může vypadat například následovně: „2015-04-30T00:00:00Z“. Hodnoty jsou seřazeny od nejvýznamnější po nejméně významnou. Na konci řetězce je informace o časové zóně, ke které se časový údaj vztahuje. V uvedeném příkladu je to hodnota „Z“, která značí základní časové pásmo UTC.

Tato aplikace není koncipována pro použití v zemích různých časových pásem, proto je vhodné systém nekomplikovat a zaznamenávat čas lokální časové

[7] backendem napsaným v jazyce C# a frontendem napsaným v JavaScriptu

zóny, bez údaje o časové zóně. Doménový model navíc nevyžaduje zaznamenávat čas během dne, důležité je pouze datum, protože ale jazyk C# i JavaScript obsahují datové typy reprezentující datum i čas, je třeba použít ty.

Problém nastal při použití knihovny bootstrap-datepicker, která obsahuje komponentu uživatelského rozhraní pro snadný výběr data (obrázek 4.1). Při provázání této komponenty s textovým polem obsahujícím datum, které je zobrazeno v českém formátu, je toto datum uloženo do vnitřní proměnné (JavaScript Date). Komponenta předpokládá, že datum je zapsáno v lokální formě a do vnitřní proměnné ho uloží v UTC formě. Když se následně tato hodnota odešle na backend, je odeslána v UTC formátu. Například při uložení data 30. 4. 2015 je hodnota serializována v UTC formě na řetězec „2015-05-29T22:00:00“. Backend tuto hodnotu následně uloží do databáze. Když si uživatel znovu nechá zobrazit uloženou hodnotu, systém zobrazí 29. 4. 2015 namísto 30. 4. 2015.



Obrázek 4.1: Kalendář knihovny bootstrap-datepicker

Problém byl vyřešen vytvořením vlastní AngularJS direktivy Hrmdatepicker. Direktiva definuje vlastní HTML značku, která vyžaduje atribut obsahující controller direktivy ngModel. Ta odpovídá za „two-way binding“ hodnot mezi view a scope. Uvnitř direktivy Hrmdatepicker se v průběhu vykreslování webové stránky vytvoří nové textové pole, na které se naváže komponenta bootstrap-datepicker a vloží se do ní datum z controlleru ngModel. Po upravení data uživatelem se nově zvolená hodnota uloží zpátky do controlleru ngModel, tentokrát bez časového posunu, což je zajištěno vlastním JavaScriptovým kódem zapsaném v definici direktivy.

```
angular.module('directives').directive('hrmdatepicker', function () {
    return {
        restrict: 'E',
        require: 'ngModel',
        scope: {inputClass: '@inputClass'},
        template: '<input type="text" class="{{inputClass}}" ng-model="datum">',
        link: function (scope, element, attrs, ngModelCtrl) {

            element.find("input").datepicker({
                language: "cs",
                autoclose: true,
                todayHighlight: true
            }).on('changeDate', function (e) {
                scope.$apply(function () {
                    var year = e.date.getFullYear();
                    var month = e.date.getMonth(); //months are zero-based
                    var day = e.date.getDate();
                    var utcdate = new Date(Date.UTC(year,month,day));
                    ngModelCtrl.$setViewValue(utcdate.toISOString());
                });
            });

            ngModelCtrl.$render = function () {
                if (ngModelCtrl.$viewValue) {
                    var dt = new Date(ngModelCtrl.$viewValue);
                    element.find("input").datepicker('update', dt);
                }
            };
        }
    }
});
```

4.4.3 Mapování databázových pohledů

V systému správy zaměstnanců je použito několika číselníků, které mají původ v jiných systémech. Jejich hodnoty jsou zpřístupněny v databázi systému správy zaměstnanců prostřednictvím databázových pohledů, linkovaných do databází jiných systémů. Tyto pohledy bylo potřeba namapovat do systému prostřednictvím Entity Frameworku. Při postupu „database first“ nabízí Entity Framework standardně možnost namapovat databázové pohledy v designeru a následně s nimi uvnitř systému pracovat stejným způsobem jako s tabulkami. Při postupu „code first“, využitým v tomto systému, tato možnost není.

Problém se dá snadno obejít vytvořením všech tříd doménového modelu, včetně těch, které bude v databázi reprezentovat pohled. Při promítnutí těchto definic do databáze jsou pro všechny třídy vytvořeny tabulky. Následně je třeba smazat tabulky, které bude nahrazovat pohled, zrušit jejich asociace s ostatními tabulkami a nahradit je pohledy. Název pohledu, jeho sloupců i datových typů se musí shodovat s původní tabulkou, jinak by mohlo docházet k chybám a případně k pádům systému. Zrušením asociací mezi tabulkami narušíme kontrolu datové integrity, protože asociace mezi pohledem a tabulkou nemůže z principu existovat. Protože ale zdroje těchto pohledů zachovávají historická data, dají se považovat za dostatečně spolehlivé.

4.4.4 Poskytování číselníkových hodnot

K zapisování informací do systému je často použito číselníkových hodnot, které jsou uživateli nabízeny prostřednictvím rolovacích nabídek. Tyto hodnoty obsahují popisek, který je viditelný uživateli, a identifikátor, který se následně ukládá od databáze. Hodnoty jsou buďto definovány v kódu backendu pomocí struktury enum, nebo v databázi. V případě zadávání informací do systému je potřeba tyto hodnoty zaslat na frontend. Protože jsou ale tyto hodnoty poměrně stálé a nemění se často, je zbytečné, aby se při opakovaném použití zasílaly na frontend znovu.

Pro tento účel byl na backendu vytvořen resource, který přijímá parametr specifikující požadované hodnoty a vrací serializované číselníkové hodnoty. K zamezení opakovaných zasílání hodnot při opakovaných žádostech je na frontendu využito HTTP cachování. Během vyřizování prvního požadavku se odpověď serveru uloží do mezipaměti webového prohlížeče a všechny další požadavky jsou automaticky odpovídány daty z mezipaměti.

Kapitola 5

Testování

Testování je důležitá součást vývoje a následného udržování softwarového díla. Testováním lze docílit vyšší kvality výsledného produktu a také předejít zanesení chyb do existujícího systému během jeho upravování či rozšiřování.

Typy testů se dají kategorizovat do třech dimenzí. První dimenze je úroveň, na které se software testuje. Testuje se od nejnižší úrovně, ve které se testují třídy a metody, až po nejvyšší, která testuje splnění všech požadavků na software a schopnost provozu. Další dimenze je aspekt, který se testuje. Zahrnuje například funkční testy, výkonové testy, bezpečnostní testy a další. Poslední dimenze je účel testování, kam spadají kvalifikační a regresní testy. Kvalifikační testy ověřují, zda software poskytuje žádanou funkcionalitu. Regresní testy kontrolují, zda během úprav softwaru nebyla původní funkcionalita poškozena. [21]

5.1 Způsoby testování

5.1.1 Funkční testování REST rozhraní

Funkční testování REST rozhraní proběhlo na platformě .NET za použití nástrojů Testing Tools formou přímého testování metod controllerů tvořící rozhraní, nikoli testováním rozhraní prostřednictvím HTTP požadavků. Testy se kromě testování základní persistenční funkcionality zaměřily především na testování dodržování integritních omezení dat.

Při spuštění testovacího scénáře je před zahájením samotných testů smazána prostřednictvím Entity Frameworku celá testovací databáze, pokud existuje, a je znovu vytvořena. Následně je naplněna testovacími daty, jejichž existence se při vykonávání testů předpokládá. Tím se zároveň otestuje funkčnost vygenerovaných DDL skriptů, které by využity při nasazení systému do nového prostředí či upgradu mezi verzemi systému.

Jako příklad je níže uveden kód metody testující vytvoření záznamu pracovníka a následně přítomnost tohoto nového záznamu v systému.

```
[TestMethod]
public void PostPracovnik()
{
        var pracovnik = new PracovnikDTO() {
                login = "kokoman",
                jmeno = "Koko",
                prijmeni = "Man",
                pohlaviId = Pohlavi.Muz,
                publikovatKontakt = true,
                velikostObleceni = Velikost.XXL
        };
        var result = controller.PostPracovnik(pracovnik);

        Assert.IsInstanceOfType(result, typeof(CreatedAtRouteNegotiatedContentResult<Pracovnik
              >));

        int pracovnikId = Int32.Parse(((CreatedAtRouteNegotiatedContentResult<Pracovnik>)
              result).RouteValues["id"].ToString());
        var result2 = controller.GetPracovnik(pracovnikId);

        Assert.IsInstanceOfType(result2, typeof(OkNegotiatedContentResult<PracovnikDTO>));
}
```

5.1.2 Zátěžové testování REST rozhraní

Zátěžové testování systému bylo realizováno zasíláním mnoha souběžných HTTP požadavků na REST rozhraní, simulujících aktivitu uživatelů. K definici testovacího scénáře i k jeho samotnému běhu byl použit nástroj Apache JMeter. V tomto nástroji lze vytvářet komplexní testy, které dokáží zpracovávat odpovědi serveru a získávat z nich parametry, které jsou použity pro kontrolu správnosti odpovědí, nebo se podle nich mohou testy například dále větvit, či mohou být použity jako parametry dalších požadavků na server. Testy mohou být spouštěny postupně, jako jeden požadavek za druhým, anebo lze vytvořit skupinu vláken, která reprezentují skupinu uživatelů, kde každé vlákno zasílá požadavky nezávisle na ostatních. Takovýmto nastavením se z testů stávají zátěžové testy.

Realizovaný testovací scénář obsahuje jednoduchý test, který na server zašle POST požadavek na vytvoření pracovníka. Po vytvoření server zašle odpověď s daty nového záznamu, ze kterých test získá identifikátor pracovníka. Poté zašle DELETE požadavek s daným identifikátorem a počká na odpověď. Scénář je vykonáván naráz 30 vlákny a každé vlákno spouští 10 těchto testů po sobě.

5.2 Výsledky testování

Průběžné funkční testování REST rozhraní napomohlo k odhalení chyb v kontrole integritních omezení. Nejpřínosnější bylo testování mezních hodnot. Chyby byly opraveny a testy mohou nadále fungovat jako regresní.

Během zátěžového testování dokázal systém obsloužit naráz 30 aktivních uživatelů a tím splnil nároky ze strany zadavatele.

Závěr

V rámci této práce byl úspěšně navržen a reimplementován informační systém pro správu zaměstnanců. Systém byl navržen dle požadavků kladených na systém ze strany zadavatele a dle vzoru původního systému, který se tímto systémem nahrazuje.

Během návrhu bylo třeba upravit původní doménový model systému, který nevhodně reflektoval některé aspekty řešeného problému. Tyto změny jsou v práci vysvětleny a popsány. Součástí práce je také návrh a popis architektury, vrstvení systému a komunikace mezi jednotlivými komponentami. Při výběru technologií použitých pro implementaci byly technologie prozkoumány a popsány, včetně použitých frameworků a knihoven a také využití některých návrhových vzorů. V průběhu implementace bylo třeba řešit několik nečekaných problémů, které jsou v této práci vyřešeny a zdokumentovány. Zdrojový kód implementace je v příloze této práce.

K finálnímu nasazení systému do produkčního prostředí je potřeba navrhnout proces migrace dat z původního systému a transformovat data do formy odpovídající upravenému doménovému modelu. Následně je třeba systém otestovat z uživatelského hlediska. Odhalené chyby, nedokonalosti či přání je třeba v systému zohlednit. Protože původní a nový systém nejsou schopny souběžného používání, je nutné nový systém nasadit až při jistotě úplné kompletnosti řešení.

Literatura

[1] Shaked, U.: AngularJS vs. Backbone.js vs. Ember.js [online]. [Citováno 8.4.2015]. Dostupné z: https://www.airpair.com/js/javascript-framework-comparison

[2] Microsoft: Access [online]. 2015, [Citováno 10.4.2015]. Dostupné z: https://products.office.com/cs-cz/access

[3] BambooHR: BambooHR [online]. 2015, [Citováno 25.3.2015]. Dostupné z: http://www.bamboohr.com/

[4] Vema: Informační systém HR Vema [online]. 2015, [Citováno 25.3.2015]. Dostupné z: http://www.vema.cz/default.aspx?categoryID=Produkty.2

[5] OrangeHRM: OrangeHRM Open Source [online]. 2015, [Citováno 25.3.2015]. Dostupné z: http://www.orangehrm.com/OpenSource.php/

[6] Jacobson, I.; Spence, I.; Bittner, K.: *USE-CASE 2.0*. Englewood Cliffs: Ivar Jacobson International, Prosinec 2011, 4 s.

[7] Microsoft: Active Directory Domain Services [online]. 2015, [Citováno 21.4.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/aa362244(v=vs.85).aspx

[8] Kucinskas, D.: Entity Framework 6 vs NHibernate 4 [online]. Leden 2014, [Citováno 1.4.2015]. Dostupné z: https://www.devbridge.com/articles/entity-framework-6-vs-nhibernate-4/

[9] Lerman, J.: Demystifying Entity Framework Strategies: Model Creation Workflow [online]. Dostupné z: https://msdn.microsoft.com/en-us/magazine/hh148150.aspx

[10] Microsoft: CSDL, SSDL, and MSL Specifications [online]. Srpen 2011, [Citováno 3.5.2015]. Dostupné z: https://msdn.microsoft.com/en-us/library/vstudio/bb399604(v=vs.100).aspx

[11] James, A. D.: How to choose an Inheritance Strategy [online]. Duben 2009, [Citováno 9.5.2015]. Dostupné z: http://blogs.msdn.com/b/alexj/archive/2009/04/15/tip-12-choosing-an-inheritance-strategy.aspx

[12] Microsoft: CSDL, SSDL, and MSL Specifications [online]. 2015, [Citováno 17.4.2015]. Dostupné z: https://msdn.microsoft.com/en-us/data/jj574232.aspx

[13] Webber, J.; Parastatidis, S.; Robinson, I.: *REST in Practice*. O'Reilly Media, 2010, 57 s.

[14] Wasson, M.: Routing in ASP.NET Web API [online]. Únor 2012, [Citováno 8.4.2015]. Dostupné z: http://www.asp.net/web-api/overview/web-api-routing-and-actions/routing-in-aspnet-web-api

[15] Google: AngularJS [online]. 2015, [Citováno 12.2.2015]. Dostupné z: https://angularjs.org/

[16] Sevilleja, C.: AngularJS Routing Using UI-Router [online]. Duben 2014, [Citováno 4.5.2015]. Dostupné z: https://scotch.io/tutorials/angular-routing-using-ui-router

[17] Otto, M.; Thornton, J.; Rebert, C.; aj.: Bootstrap: Getting started [online]. [Citováno 4.4.2015]. Dostupné z: http://getbootstrap.com/getting-started/

[18] Krátký, T.: Configuration Management [online]. [Citováno 5.5.2015]. Dostupné z: http://www.profinit.eu/fileadmin/Content/profinit.eu/Academy/NSWI129/prednasky/07_CM.pdf

[19] Ruderman, J.: Same-origin policy [online]. [Citováno 9.4.2015]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy

[20] ISO: Date and time format - ISO 8601 [online]. [Citováno 20.4.2015]. Dostupné z: http://www.iso.org/iso/home/standards/iso8601.htm

[21] Krátký, T.: Software testing [online]. [Citováno 5.5.2015]. Dostupné z: http://www.profinit.eu/fileadmin/Content/profinit.eu/Academy/NSWI129/prednasky/05_Testing.pdf

Příloha **A**

Seznam použitých zkratek

AD Active Directory

AJAX Asynchronous JavaScript and XML

API Application Programming Interface

ASP Active Server Pages

CRM Customer Relationship Management

CRUD Create, Read, Update, Delete

CSDL Common Schema Definition Language

CSS Cascading Style Sheets

DDL Data Definition Language

DML Data Manipulation Language

DTO Data Transfer Object

DWH Data Warehouse

EDM Entity Data Model

ERP Enterprise Resource Planning

ETL Extract, Transform, Load

FTP File Transfer Protocol

FTPS Kombinace File Transfer Protocol (FTP) a Transport Layer Security (TLS)

HRM Human Resource Management

HTML Hyper Text Markup Language

HTTP Hypertext Transfer Protocol

IDE Integrated Development Environment

IIS Internet Information Services

IS Informační Systém (Information system)

ISO International Organization for Standardization

JSON JavaScript Object Notation

LINQ Language Integrated Query

MS Microsoft

MSL Mapping Specification Language

MVC Model View Controller

NNTP Network News Transfer Protocol

ORM Objektově relační mapování (Object Relational Mapping)

REST Representational State Transfer

SMTP Simple Mail Transfer Protocol

SPA Single Page Application

SQL Structured Query Language

SSDL Store Schema Definition Language

TPC Table Per Concrete Class

TPH Table Per Hierarchy

TPT Table Per Type

URI Uniform Resource Identifier

URL Uniform Resource Locator

UTC Coordinated Universal Time

VBA Visual Basic for Applications

XML Extensible Markup Language

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `BP_Pavel_Peroutka_2015` .... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Pavel_Peroutka_2015.pdf` ..... text práce ve formátu PDF