VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

ZAMĚSTNANECKÝ A VÝPLATOVÝ INFORMAČNÍ SYSTÉM

INFORMATION SYSTEM FOR EMPLOYEES AND SALARIES RECORDS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JOZEF ONDRIA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Ondria Jozef**
Program: Informační technologie
Název: **Zaměstnanecký a výplatový informační systém**
Information System For Employees and Salaries Records
Kategorie: Informační systémy

Zadání:

1. Seznamte se s programovacím jazykem Java, tvorbou mobilních aplikací a dostupnými prostředími. Dále prostudujte legislativu ohledně výpočtu mezd na Slovensku.
2. Analyzujte požadavky na informační systém, který bude evidovat docházku zaměstnanců, provádět výpočet hrubé i čisté mzdy, evidenci dovolené, tisk výplatních pásek. Svou docházku bude moci zaměstnanec zadávat pomocí mobilní aplikace.
3. Navrhněte informační systém dle požadavků.
4. Navržený informační systém implementujte a otestujte jeho funkčnost.
5. Zhodnoťte dosažené výsledky a další možné pokračování tohoto projektu.

Literatura:

- KEOGH, James Edward. Java bez předchozích znalostí: průvodce pro samouky. Brno: CP Books, 2005. ISBN 80-251-0839-2.
- Act 311/2001 Coll.: Labour code = Zakon c. 311/2001 Z.z. : zakonník prace (ucinny k 1.1.2012). Bratislava: Wolters Kluwer, 2011. ISBN 9788080784393

Pro udělení zápočtu za první semestr je požadováno:

- Body 1-3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Bartík Vladimír, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 16. října 2019

Abstrakt

Cieľom tejto práce bolo vytvorenie zamestnaneckého, výplatného informačného systému určeného pre zamestnávateľov. Informačný systém je postavený na dvoj-vrstvovej architektúre, pričom pre implementáciu bol zvolený MySQL databázový systém, jazyk PHP pre REST rozhranie serveru s použitím frameworku CodeIgniter, jazyk Java pre klientskú desktopovú aplikáciu a klientskú mobilnú aplikáciu na platforme Android. Najdôležitejšími funkciami systému sú výpočet miezd zamestnancov, s ktorou priamo súvisia tiež funkcie pre správu legislatívnych a firemných informácií, evidenciu dochádzky a absencií zamestnancov, správu zamestnancov a ich užívateľských kont. Informačný systém bol úspešne vytvorený a otestovaný potenciálnymi používateľmi.

Abstract

This thesis aimed to design and create an information system for employees and salary records. The system is based on a client-server architecture and uses multiple technologies. First, the MySQL system was chosen for the database implementation, PHP language and CodeIgniter framework for the REST server interface, and lastly Java for both the desktop and also Android mobile client application. The implemented system offers the ability to calculate employees' salaries, record their attendance, manage their accounts along with their personal and work data, and manage the legislative and company data as well.

Kľúčové slová

informačný systém, PHP, CodeIgniter, MySQL, Java, JavaFX, Android, zamestnávateľ, zamestnanci, mzdy, výplaty, dochádzka

Keywords

information system, PHP, CodeIgniter, MySQL, Java, JavaFX, Android, employer, employee, salaries, work attendance

Citácia

ONDRIA, Jozef. *Zaměstnanecký a výplatový informační systém.* Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Zaměstnanecký a výplatový informační systém

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pana Ing. Vladimíra Bartíka, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Jozef Ondria
28. mája 2020

Poďakovanie

Týmto ďakujem vedúcemu práce Ing. Vladimírovi Bartíkovi , Ph.D. za odborné vedenie a pomoc pri tvorbe tejto bakalárskej práce.

Obsah

Kapitola 1

Úvod

Cieľom tejto bakalárskej práce je vytvoriť informačný systém pre ľubovoľného stredného či malého podnikateľa, resp. zamestnávateľa, ktorý by pomohol a najmä uľahčil vypočítať výplaty zamestnancov a to podľa legislatívy Slovenskej republiky.

Zamestnávateľ by mal mať prístup k informačnému systému prostredníctvom desktopovej aplikácie implementovanej v jazyku Java. Okrem toho by mal systém ponúknuť zamestnancovi možnosť zaznamenať svoj odpracovaný výkon do systému pomocou mobilnej aplikácie.

Obsah tejto práce je logicky usporiadaný v niekoľkých kapitolách. V kapitole 2 sa čitateľ dozvie niečo o základných princípoch tvorby informačných systémoch a technológiách, ktoré sa k vývoju týchto systémov využívajú.

Kapitola 3 je tiež teoreticky zameraná. Jej úlohou je čitateľa oboznámiť so základnými právnymi normami a povinnosťami, ktoré vplývajú na výpočet miezd a to podľa legislatívy Slovenskej republiky.

Na túto kapitolu nadväzuje kapitola 4, ktorá rozoberá analýzu a špecifikáciu základných požiadaviek informačného systému. Táto analýza je zakončená diagramom prípadov použitia, ktorý poukazuje na základne funkcionality systému z pohľadu aktérov, ktorí sa v tomto systéme vyskytujú.

Nasledujúcou časťou tohto dokumentu, ktorá popisuje ďalšiu etapu praktického vývoja informačného systému, je návrh systému, ktorý je obsiahnutý v kapitole 5. Jej hlavnou časťou je dátový návrh informačného systému, ktorý pozostáva z ER diagramu. Okrem toho je v nej aj stručne popísaný výber architektúry systému a návrh používateľského rozhrania tohto systému.

Najrozsiahlejšia kapitola 6 čitateľovi ponúka informácie o samotnej implementácii informačného systému. Najprv sú vymenované a vysvetlené použité technológie pri vývoji, následne implementácia jednotlivých aplikácií rozdelených podľa použitej architektúry. Nakoniec je stručne popísaná implementácia hlavných častí v rámci celého informačného systému. Na túto kapitolu nadväzuje kapitola 7. Tá pojednáva o testovaní vyvíjaného systému.

Záverečnou časťou dokumentu je kapitola 8. V nej sú zhodnotené výsledky, ktoré boli v tejto bakalárskej práci dosiahnuté.

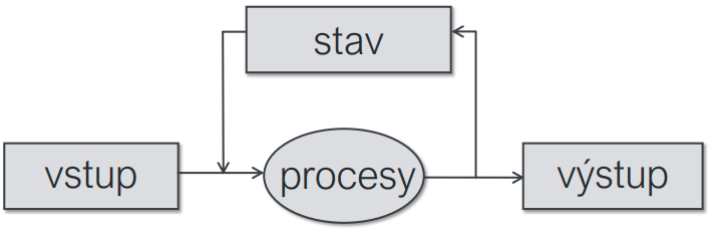
Kapitola 2

Princípy tvorby informačných systémov

V tejto kapitole nadobudne čitateľ základné poznatky o informačných systémoch a ich klasifikácii. Zároveň sú v tejto sekcii zahrnuté aj informácie o najpoužívanejších architektúrach informačných systémov či technológiách, ktoré sa pri vývoji týchto systémov využívajú.

2.1 Informačný systém

Informačný systém môžeme chápať ako systém vzájomne prepojených informácií a procesov, ktoré s týmito informáciami pracujú. Pod pojmom procesy rozumieme funkcie, ktoré spracovávajú vstupné údaje a transformujú ich na výstupné údaje. Zjednodušene je možné povedať, že procesy zabezpečujú zber, prenos, uloženie, spracovanie a distribúciu dát. Pod pojmom informácie zas rozumieme dáta, ktoré slúžia hlavne na rozhodovanie a riadenie v systéme. [33]



Obr. 2.1: Schéma informačného systému (zdroj: [21, sl. 30])

2.2 Klasifikácia informačných systémov podľa úrovne rozhodovania

Klasifikáciu informačných systémov podľa úrovne rozhodovania obecne popisuje pyramídová schéma, ktorú je možné vidieť na obrázku 2.2. Schéma obsahuje štyri základné úrovne, ktoré sú bližšie vysvetlené nižšie.

2.2.1 Systémy OLTP

Primárnym cieľom pri návrhu OLTP systémov, resp. systémov transakčného spracovania je umožniť klientom databázového serveru vykonávať veľké množstvo transakcií online. Príkladom takýchto transakcií je napríklad internet banking a online nakupovanie. Cieľom týchto systémov je automatizácia každodenných činností, ktoré sú predmetom podnikania. V transakčných systémoch pristupuje k zdroju údajov veľké množstvo používateľov, ktorí údaje čítajú, zapisujú, prípadne niektorí nad nimi vykonávajú jednoduché analýzy. Z teórie aj praktických skúseností vyplýva, že údaje v týchto databázach by mali byť uložené v normalizovaných tabuľkách, ktoré by mali vyhovovať druhej alebo tretej normálovej forme. [27, str. 20]

2.2.2 Systémy MIS

Systémy MIS sú systémy pre podporu riadenia, ktorých hlavnou úlohou je poskytovanie kvalitných informácií pre riadiacich pracovníkov. Informácie slúžia ako podklad pre riadenie. Ide o prechod z úrovne okamžitých transakcií na akúsi firemnú operatívno-taktickú úroveň. Tieto systémy poskytujú riadiacim pracovníkom rôzne komplexné prehľady, agregované podľa rôznych hľadísk, napríklad časových, geografických a mnohých ďalších. Do MIS systémov vstupujú údaje z transakčných systémov. [27, str. 20]

2.2.3 Systémy DSS

Ide o systémy pre podporu rozhodovania, ktoré sú nadstavbou systémov MIS. Na rozdiel od systémov MIS, ktoré sa nasadzujú na operatívne-taktickej úrovni, systémy DSS sú na rozhraní taktického a strategického rozhodovania. Poskytujú riadiacim pracovníkom napríklad výsledky pomerne zložitých analýz. Okrem nasadenia DSS na rozhraní taktickej a strategickej úrovne, bol dôležitým aspektom aj nástup osobných počítačov v osemdesiatych rokoch minulého storočia. Tento aspekt umožnil priamy prístup k operačným údajom aj pre obchodných používateľov. [27, str. 21]

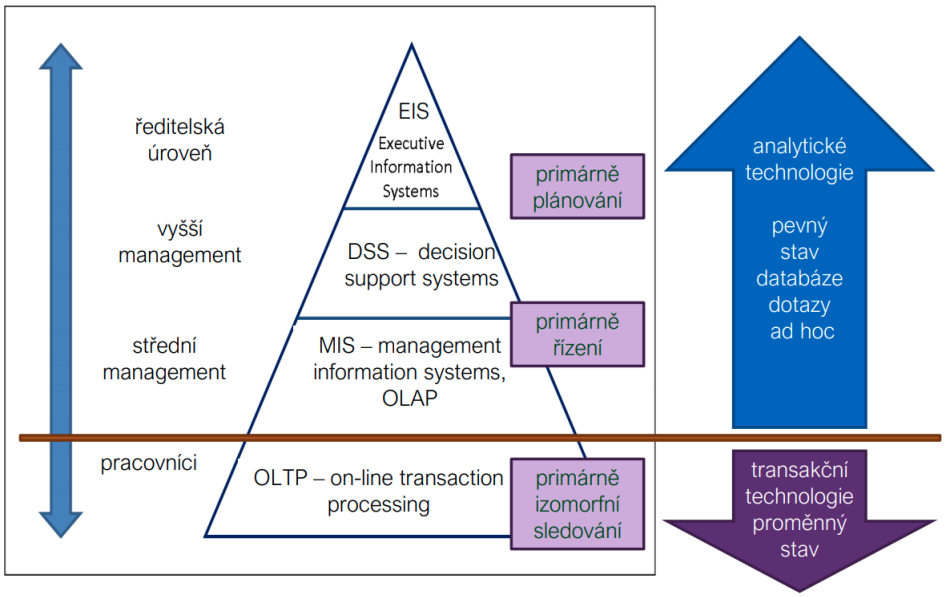
2.2.4 Systémy EIS

Ide o informačné systémy pre vrcholové riadenie nasadzované na strategickej úrovni. Tieto systémy sú však čoraz častejšie nahradzované systémami DSS pre podporu rozhodovania, pričom sú nazývané aj termínom Bussiness Inteligencia. [27, str. 21]

2.2.5 Systémy OLAP

Systémy OLAP sú systémy analytického spracovania určené pre analýzu veľkého množstva údajov. Výsledkom analýzy sú súhrny a reporty, ktoré slúžia manažérom ako podklady pre ich rozhodovanie, či už v oblasti riadenia firmy, riadenia ekonomických, technologických procesov a podobne. Pre výpočet OLAP kociek je potrebné vykonať veľké množstvo výpočtov či agregácii, a to skoro v reálnom čase. [27, str. 21]

OLAP systémy síce nie sú explicitne uvedené v pyramídovej schéme klasifikácie informačných systémov, no môžu sa vyskytovať v jej troch, vyššie spomenutých úrovniach, ktorými sú systémy MIS, DSS a EIS.



Obr. 2.2: Pyramídová schéma klasifikácie informačných systémov (zdroj: [21, sl. 44])

2.3 Architektúry informačných systémov

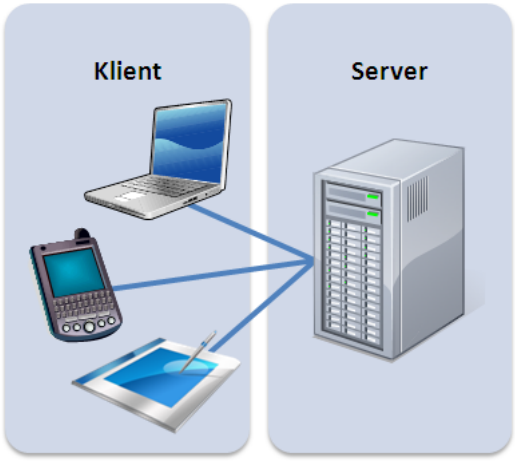
Informačný systém môže byť postavený na rôznych architektúrach. Architektúra popisuje, akým spôsobom je informačný systém rozdelený medzi to, čo používa používateľ a to, čo sa odohráva v pozadí na strane servera. Existuje niekoľko druhov týchto architektúr, z ktorých tie najpoužívanejšie sú vysvetlené v tejto podkapitole.

2.3.1 Klient-server architektúra

Architektúra klient-server (anglický client-server model) je jeden z typov architektúry informačných systémov. Je to tzv. **dvojvrstvová architektúra**, kde klient (čo je aplikácia na koncovom zariadení používateľa) zabezpečuje používateľské rozhranie a aplikačnú logiku, zatiaľ čo na serveri sa nadchádza databáza. [2]

Model klient-server je forma distribuovaného spracovania výpočtového výkonu medzi koncovým zariadením (**klientom**) a **serverom**, ktorí medzi sebou komunikujú a odovzdávajú si navzájom dáta. Klient predkladá používateľovu požiadavku tak, aby bola zrozumiteľná pre server a čaká od neho odpoveď, ktorú prekladá späť do takej formy, aby bola zrozumiteľná pre klienta, ktorý ju na obrazovke prezentuje používateľovi. V architektúre klient-server teda server spracováva dotazy v databáze a klient ich prezentuje, zabezpečuje aplikačnú logiku a sprostredkováva rozhranie pre používateľov. [2]

Tento model je dnes už koncepčne zastaraný. Kvôli tomu, že klient obsahuje väčšinu aplikačnej logiky, tak so zložitosťou aplikácie rastú nároky na klientské počítače či zariadenia. Tiež je v tejto architektúre ťažšie aplikácie na strane klienta aktualizovať. Ďalšou nevýhodou architektúry klient-server je, že je problematické sprístupniť časť aplikácie pomocou webového prehliadača. [2]



Obr. 2.3: Schéma klient-server architektúry informačného systému (zdroj: [2])

2.3.2 Trojvrstvová architektúra

Trojvrstvová architektúra (anglicky Three-tier architecture) označuje jeden z typov architektúry informačných systémov. Ako už z názvu vyplýva, jedna sa o architektúru, v ktorej je informačný systém rozdelený do troch vrstiev – prezentačnej, aplikačnej a dátovej. [15]

Prezentačná vrstva je tá časť, ktorá je viditeľná pre používateľov, zaisťuje vstup požiadaviek a prezentáciu výsledkov. Je závislá na platforme (napr. webová aplikácia, desktopová aplikácia pre windows, android aplikácie atď.). Môže byť teda rôzna pre rôzne zariadenia či platformy. [15]

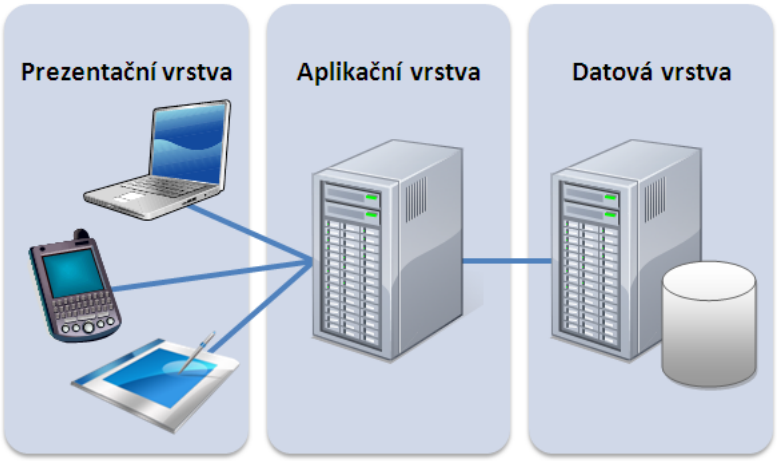
Ďalšou vrstvou architektúry je **aplikačná vrstva** (funkčná vrstva). Je to prostredná vrstva modelu (anglicky middleware), ktorá zabezpečuje výpočty a operácie vykonávané medzi vstupno-výstupnými požiadavkami a dátami. Je tiež nazývaná ako aplikačný server. [15]

Posledná **dátová vrstva** (databázová vrstva) je najnižšia vrstva modelu, zaisťujúca prácu s dátami, teda systém riadenia bázy dát a základné dátovo-funkčné operácie zaisťujúce ukladanie, výber, agregáciu, predspracovanie, integritu či audit dát. [15]

Výhodou architektúry je oddeľovanie jednotlivých vrstiev tak, aby na sebe neboli závislé. Trojvrstvovú architektúru využíva veľké množstvo aplikácií, ktoré pracujú s dátami. Takto je postavená väčšina moderných podnikových aplikácií, niektoré portálové riešenia či webové stránky. Trojvrstvová a viac vrstvová architektúra je v dnešnej dobe trendom a využíva sa pre tvorbu robustnejších systémov. Jej výhodou je rozdelenie výkonu medzi zariadenie používateľa a server. Prezentačná vrstva môže pracovať taktiež na výkonnostne slabších zariadeniach. [15]

2.4 Technológie

V praxi je možné pri vývoji informačných systémov použiť mnoho technológií a programovacích jazykov. V tejto sekcii sú uvedené tie najpoužívanejšie z nich, roztriedené podľa vrstiev trojvrstvovej architektúry.



Obr. 2.4: Schéma trojvrstvovej architektúry informačného systému (zdroj: [15])

2.4.1 Databázový systém

Medzi tri najpopulárnejšie databázové systémy patria Oracle, MySQL a Microsoft SQL Server. Všetky tieto systémy sú relačné databázy. Relačné databázy ukladajú údaje v tabuľkách, ich riadkoch a stĺpcoch. Riadok tabuľky predstavuje záznam, zatiaľ čo stĺpce definujú atribúty jednotlivých záznamov v tabuľke. [9]

Oracle Database

Databázový systém Oracle bol vyvinutý spoločnosťou Oracle Corporation a je najpopulárnejším relačným systémom riadenia bázy dát. Napriek tomu že je to relačný systém poskytuje funkcionalitu napríklad aj pre objektovo-relačný model databázy, cloudové služby, grafové databázy atď. V databáze Oracle je databázová schéma súborom logických dátových štruktúr alebo objektov schémy. Používateľ databázy vlastní databázovú schému s názvom ktorý je rovnaký ako prihlasovacie meno používateľa. [9]

MySQL

MySQL je najpopulárnejší relačný databázový systém riadenia bázy dát, ktorý je voľne dostupný. Získala ho spoločnosť Oracle v roku 2009 a je najbežnejší štandardizovaný jazyk používaný pre prístup k databázam. K jeho výhodám patrí rýchlosť, spoľahlivosť a nízka náročnosť používania. [9]

Microsoft SQL

K najpopulárnejším databázovým systémom patrí tiež systém vyvinutý spoločnosťou Microsoft, ktorý bol uvedený na trh v roku 1989, a ktorý bol vyvinutý prostredníctvom programovacích jazykov C/C++. Podobne ako dva vyššie spomenuté databázové systémy aj tento patrí k relačným databázovým systémom. Je vo veľkej miere používaný významnými spoločnosťami. [9]

2.4.2 Aplikačná vrstva

Aplikačná vrstva alebo aplikačný server môže byť implementovaný radou technológií a programovacích jazykoch, preto tie najpopulárnejšie sú priblížené v nasledujúcom texte.

PHP

PHP je multiplatformový a skriptovací programovací jazyk, ktorý pracuje na strane servera. Je ním možné vyvíjať dynamické webové aplikácie, ktoré komunikujú s databázovými systémami. Pôvodný význam skratky PHP bol Personal Home Page. Vzniklo v roku 1994. Od tej doby prešlo veľkými zmenami a teraz táto skratka znamená PHP: Hypertext Preprocessor. [11]

Java

Java je vysoko-úrovňový programovací jazyk, pôvodne vyvinutý spoločnosťou Sun Microsystems a uvedený na trh v roku 1995. Tento jazyk beží na rade platforiem ako sú Windows, Mac OS a rôzne verzie UNIX-u. Výhodami tohto programovacieho jazyka je objektová orientácia, jednoduchosť používania a robustnosť. [7]

Je to objektovo orientovaný programovací jazyk, ktorého syntax je podobná programovacím jazykom C a C++. Java je interpretovaný jazyk , čo znamená, že program sa neprekladá priamo do strojového kódu, ale do takzvaného bajt-kódu Java. Ten je následne interpretovaný virtuálnym strojom. Je tak programovacím jazykom nezávislým na platforme, čo je jeho najväčšou výhodou. [28, str.17]

ASP.NET

ASP.NET je framework pre webové aplikácie, ktorý bol vyvinutý spoločnosťou Microsoft. Umožňuje vývojárom vytvorenie dynamických webových aplikácií s použitím jazykov ako sú C, VB.NET alebo JavaScript. Pracuje nad protokolom HTTP a používa jeho príkazy pre nastavenie komunikácie či spolupráce medzi prehliadačom a serverom. [3]

2.4.3 Prezentačná vrstva

Výber technológie, resp. programovacieho jazyka pri prezentovaní odpovede servera používateľovi na tejto vrstve, závisí najmä od hrúbky klienta. V prípade, že ide o tenšieho klienta, kedy aplikácia neobsahuje žiadnu plnohodnotnejšiu logiku (najčastejšie pri webovom informačnom systéme), je použitá skupina programovacích jazykov HTML, CSS a JavaSript pre tvorbu webových aplikácií. Pre implementáciu hrubších klientov, kedy aplikácia obsahuje aj logiku, sa najčastejšie vyvíjajú desktopové alebo mobilné aplikácie v programovacích jazykoch ako je Python, C, C/C++ alebo Java, ktorá je popísaná vyššie.

HTML a CSS

HTML je hypertextový značkovací jazyk, používaný hlavne pri vývoji webových aplikácií. Vytvoril ho Berners-Lee v roku 1991, ale prvou uverejnenou štandardizovanou verziou bola verzia 2.0 v roku 1995. Pôvodne bol jazyk HTML vyvíjaný s cieľom definovať štruktúru dokumentov, ako sú nadpisy, odseky a zoznamy pre uľahčenie zdieľania vedeckých údajov medzi vedcami. Momentálne sa jazyk používa na vytváranie štruktúry webových stránok. [6]

CSS je technológia používaná na jednoduché štýlovanie webového dokumentu, ktorý je napísaný prevažne v jazyku HTML. Je teda zodpovedný za vzhľad webovej stránky. Je ním možné ovládať farbu textu, štýl písma, medzery medzi odsekmi, rozloženie, zarovnanie obsahu stránky atď. [5]

C++

C++ je programovací jazyk strednej úrovne, ktorý vyvinul Bell Labs. Jazyk beží na rade operačných systémov ako sú Windows, Mac OS a na rôznych verziách UNIX-u. Je to staticky napísaný, kompilovaný a univerzálny programovací jazyk, ktorý podporuje procedurálne a objektovo-orientované programovanie. [4]

Python

Python je vysoko-úrovňový, interpretovaný a objektovo-orientovaný jazyk, ktorý vytvoril Guido van Rossum v rokoch 1985-1990. Bol navrhnutý tak, aby bol dobre čitateľný. Oproti ostatným programovacím jazykom má menej syntaktických konštrukcií. Jeho hlavnými výhodami sú už spomenuté ľahké čítanie zdrojového kódu, ľahké učenie, ľahko udržiavaný zdrojový kód a multiplatformnosť. [13]

Kapitola 3

Legislatíva ohľadom výpočtu miezd na Slovensku

Ako bolo načrtnuté už v úvode tejto práce, cieľom je vytvoriť informačný systém, ktorého hlavnou funkciou je prepočet miezd, resp. výplat pracujúcich u ľubovoľného zamestnávateľa. S výpočtom miezd však súvisí veľké množstvo legislatívnych zákonov či podmienok, ktoré musí zamestnávateľ splniť. Preto sú v tejto sekcii uvedené tie najpodstatnejšie legislatívne a právne normy, ktoré podstatne ovplyvňujú už spomenutý výpočet miezd na území Slovenska.

V prvej časti tohto textu je čitateľ oboznámený s pracovnoprávnymi vzťahmi, ktoré môže zamestnávateľ uzatvoriť so zamestnancom a ich jednotlivými náležitosťami. V ďalšej časti je priblížené zaznamenávanie odpracovaného výkonu zamestnanca, zatiaľ čo v poslednej časti sú rozobraté jednotlivé náležitosti výplatnej pásky, jej zložky mzdy, a to na čom tieto zložky mzdy závisia.

3.1 Pracovnoprávny vzťah

To najpodstatnejšie pri vyhotovení výplatnej pásky pre zamestnanca je druh pracovného vzťahu, ktorý daný zamestnanec so zamestnávateľom uzatvorí, a jeho dohodnuté podmienky. Závisí od toho napríklad výška minimálnej mzdy zamestnanca a iné záležitosti, ktoré sú bližšie vysvetlené pri konkrétnych druhoch týchto vzťahov.

Pracovnoprávne vzťahy sú právne vzťahy medzi zamestnávateľom a zamestnancom, ktorý vykonáva závislú prácu. Práva a povinnosti vyplývajúce z pracovnoprávnych vzťahov sú regulované legislatívou, najmä Zákonníkom práce. Ten upravuje individuálne ako aj kolektívne pracovnoprávne vzťahy. Závislá práca je práca vykonávaná vo vzťahu nadriadenosti zamestnávateľa a podriadenosti zamestnanca, osobne zamestnancom pre zamestnávateľa, podľa pokynov zamestnávateľa, v jeho mene a v pracovnom čase určenom zamestnávateľom. [12]

Pod pracovnoprávnym vzťahom je potrebné chápať predovšetkým pracovný pomer, ktorý je typickým pracovným vzťahom, a dohody o prácach vykonávaných mimo pracovného pomeru. [12]

Pracujúci môže mať uzatvorených viacero pracovnoprávnych vzťahov s rôznymi zamestnávateľmi. Je ale aj možné, že ich má uzatvorené s jedným zamestnávateľom. Najčastejšie ide o pracovný pomer v kombinácii s ľubovoľnou formou dohody o práci vykonávanej mimo pracovného pomeru.

3.1.1 Pracovný pomer

Pracovný pomer je pracovnoprávny vzťah medzi zamestnancom a zamestnávateľom založený na základe písomnej pracovnej zmluvy. Pracovná zmluva musí obsahovať povinné náležitosti, ktorými sú [25]:

- Druh práce, ktorú bude zamestnanec vykonávať a jej stručná charakteristika,
- Miesto výkonu práce zamestnanca – konkrétna adresa alebo názov obce,
- Deň nástupu do práce – je to zároveň aj deň vzniku pracovného pomeru,
- Mzdové podmienky – ak nie sú dohodnuté v kolektívnej zmluve.

Okrem povinných náležitostí zamestnávateľ uvedie v pracovnej zmluve aj ďalšie pracovné podmienky, napr. výmeru dovolenky, hmotné výhody, výplatné termíny alebo dĺžku výpovednej doby. [25]

Niektoré spomenuté náležitosti pracovnej zmluvy sú vysvetlené v ďalšom texte tejto kapitoly.

Dohodnutý pracovný čas

Existuje niekoľko delení pracovných pomerov. Pre problematiku výpočtu miezd je však dôležité len rozdelenie pracovných pomerov podľa dĺžky pracovného času. Podľa tejto hodnoty pracovného času sa napríklad vypočítava pomerná výška minimálnej mzdy zamestnanca. Na základe dĺžky pracovného času delíme pracovné pomery na:

- **Pracovný pomer na plný úväzok** – vyskytuje sa v 40, 38.75 alebo 37.5 hodinovej forme za týždeň. [17, §85, ods.5]
- **Pracovný pomer na kratší pracovný čas** – v akejkoľvek forme kratšej ako je ustanovený týždenný pracovný čas. [25]

Ďalšími špecifickými druhmi pracovného pomeru sú delené pracovné miesto a domácka práca i telepráca.

Dohodnutý druh práce

Prácu, ktorú má zamestnanec uvedenú v pracovnej zmluve ako druh práce a jej charakteristika, môže ovplyvniť mzdu zamestnanca a to prostredníctvom minimálnej mzdy.

Zákonník práce ustanovuje, že ak nie je odmeňovanie zamestnancov dohodnuté v kolektívnej zmluve, je zamestnávateľ (spravidla malý a stredne veľký podnikateľ), povinný zamestnancovi poskytnúť mzdu najmenej v sume minimálneho mzdového nároku určeného pre stupeň náročnosti práce príslušného pracovného miesta. [31]

Zamestnávateľ musí vždy zhodnotiť každé vytvorené pracovné miesto a súčasne mu priradiť príslušný stupeň náročnosti, a to v súlade s charakteristikami stupňov náročnosti pracovných miest uvedenými v prílohe č. 1 Zákonníka práce – podľa najnáročnejšej pracovnej činnosti, ktorej výkon sa od zamestnanca požaduje. Zákon podľa miery zložitosti, zodpovednosti a namáhavosti práce zamestnanca rozlišuje a charakterizuje šesť stupňov náročnosti pracovných miest. [31]

Dohodnuté mzdové podmienky

V rámci mzdových podmienok sa musia dohodnúť najmä formy odmeňovania zamestnancov, suma základnej zložky mzdy a ďalšie zložky plnení poskytovaných za prácu a podmienky ich poskytovania. [10]

Ďalšími zložkami sa myslí napríklad pohyblivá zložka mzdy ako sú prémie, osobné ohodnotenia alebo odmeny. Čo sa týka základnej zložky mzdy, tento text je zameraný len na tie najpoužívanejšie formy mzdy, ktoré je možné dohodnúť v pracovnom pomere.

Najpoužívanejšie formy mzdy základnej zložky sú:

- **Časová mzda** – Tento typ mzdy sa využíva najčastejšie a výška mzdy závisí od odpracovaného času. Najčastejšie sa časová mzda definuje buď hodinovou alebo mesačnou sadzbou. [30]
- **Úkolová mzda** – V tomto prípade sa využíva tento typ mzdy vtedy, ak má ohodnotenie závisieť od skutočne vykonanej práce, ktorá je merateľná alebo kvantifikovateľná. [30] Napríklad môže ísť o mzdu, kedy sa udá mzdová tarifa na jeden kus vyrobeného výrobku.
- **Podielová mzda** – Tento typ mzdy sa využíva vtedy, ak má byť ohodnotenie práce na základe dosiahnutého hospodárskeho výsledku, najčastejšie obratu alebo tržieb. [30] Dobrým príkladom môže byť čašníčka, ktorej odmena závisí od tržby, ktorú dosiahne.

V praxi sa veľmi často vyskytuje aj kombinácia týchto foriem mzdy prevažne kombinácia časovej a inej formy mzdy. Príkladom môže byť opäť čašníčka pracujúca v reštaurácii, v ktorej tržby nie sú pravidelne rovnako vysoké. V takýchto prípadoch sa často volí mzda kombináciou časovej a podielovej mzdy.

3.1.2 Dohoda o práci vykonávanej mimo pracovného pomeru

Zákonník práce umožňuje zamestnávateľovi okrem pracovných zmlúv uzatvárať aj dohody o prácach vykonávaných mimo pracovného pomeru. Medzi ne sa zaraďujú [19]:

- **Dohoda o vykonaní práce**,
- **Dohoda o pracovnej činnosti**,
- **Dohoda o brigádnickej práci študentov**.

Dohoda by mala byť uzatvorená písomne a musí v nej byť uvedená [19]:

- Práca, ktorú má vykonať zamestnanec,
- Odmena za vykonanú prácu,
- Doba trvania dohody (max. 12 mesiacov),
- Rozsah pracovného času.

V prípade ak ide o dohodu o vykonaní práce alebo o dohodu o pracovnej činnosti, z hľadiska výpočtu mzdy nás zaujíma to, či je dohodár v takýchto dohodách platený pravidelne alebo nepravidelne. Dohodárovi je vyplácaná mzda pravidelne vtedy, ak je vyplácaná pravidelne každý mesiac, čiže tak ako je to aj pri pracovnom pomere. Nepravidelné vyplácanie je vyplácanie na inej ako mesačnej báze, teda môže ísť o kratšie ale aj dlhšie obdobie.

Informácia o pravidelnosti vyplácania mzdy do veľkej miery ovplyvňuje výšku odvodov, ktoré majú byť danému dohodárovi odňaté z hrubej mzdy.

Dohodár má podobne ako zamestnanec nárok na minimálnu mzdu, no nie podľa stupňov náročnosti práce. Minimálna mzda dohodára sa vždy viaže k minimálnej mzde najnižšieho stupňa náročnosti práce. Oproti zamestnancovi nemôže dohodár pracovať nadčas, nemôže vykonávať pracovnú pohotovosť a nemôže mať nárok na dovolenku. [26]

3.2 Evidencia dochádzky a pracovného času zamestnanca

Evidencia pracovného času spolu s evidenciou dochádzky zamestnancov je užitočná z viacerých hľadísk. Slúži napríklad na kontrolu priemerného týždenného pracovného času, na určenie výmery dovolenky, na určenie výšky mzdy a poistného, prípadne na účely zostavovania výkazov pre Sociálnu a zdravotnú poisťovňu. [24]

Evidencia pracovného času musí obsahovať [24]:

- informácie o počte hodín odpracovaných zamestnancom v jednotlivých dňoch, záznam o počte hodín, ktoré zamestnanec odpracoval,
- záznam o opustení pracoviska z pracovných dôvodov – musí byť uvedený čas odchodu a príchodu a miesto výkonu práce, opustení pracoviska zamestnancom z pracovných dôvodov so zaznamenaním času odchodu, príchodu a miesta vykonávania pracovnej činnosti,
- údaj o prerušení pracovného času z osobných dôvodov – taktiež je potrebné uviesť čas odchodu a prekážky v práci na strane zamestnanca s ich uvedením a so zaznamenaním času odchodu, prekážkach v práci na strane zamestnanca a tiež záznam o počte hodín aktívnej a neaktívnej časti pracovnej pohotovosti,
- evidencia dochádzky zamestnanca by mala zaznamenávať napríklad informácie o náhradnom voľne, pracovnom voľne bez náhrady mzdy, čerpaní dovolenky alebo neospravedlnenej neprítomnosti.

Z hľadiska výpočtu mzdy zamestnanca je ale dôležité zaznamenávať predovšetkým odpracovaný čas, resp. odpracovaný výkon a neprítomnosti zamestnanca.

3.2.1 Odpracovaný výkon

Aby bol zamestnávateľ schopný vyčísliť mzdu zamestnanca, a to hlavne pre účely základnej zložky mzdy a zložky mzdových zvýhodnení, ktoré sú vysvetlené v kapitole 3.3.1, je potrebné aby bol výkon zamestnanca určitým spôsobom evidovaný. Evidencia pracovného času je však aj povinnosťou zamestnávateľa, o ktorej hovorí aj Zákonník práce Slovenskej republiky.

Zamestnávateľ je povinný viesť evidenciu pracovného času, práce nadčas, nočnej práce, aktívnej časti a neaktívnej časti pracovnej pohotovosti zamestnanca tak, aby bol zaznamenaný začiatok a koniec časového úseku, v ktorom zamestnanec vykonával prácu alebo mal nariadenú alebo dohodnutú pracovnú pohotovosť. Počas dočasného pridelenia zamestnávateľ vedie evidenciu podľa prvej vety v mieste výkonu práce dočasne prideleného zamestnanca. [17, §99]

V praxi to však často vyzerá tak, že samotný zamestnanec si eviduje svoj opracovaný výkon a to po dohode so zamestnávateľom.

3.2.2 Neprítomnosti zamestnanca

Zaznamenávanie neprítomnosti zamestnanca je dôležité hlavne pre vyčíslenie náhrad mzdy, ktoré sú zložkou mzdy, resp. hrubej mzdy uvedenej v kapitole 3.3.1. Náhrada mzdy sa totiž poskytuje zamestnancovi za dovolenku, pracovnú neschopnosť a iné prekážky v práci.

3.3 Výplatná páska

Pri vyúčtovaní mzdy je zamestnávateľ povinný zamestnancovi vydať doklad obsahujúci najmä údaje o jednotlivých zložkách mzdy, o jednotlivých plneniach poskytovaných v súvislosti so zamestnaním, o stave účtu konta pracovného času, ak je zavedené konto pracovného času, o vykonaných zrážkach zo mzdy a o celkovej cene práce. [17, §130, ods.5]

Výplatnou páskou sa teda v tomto texte rozumie tento doklad o ktorom hovorí Zákonník práce Slovenskej republiky. Výplatná páska je v nasledujúcom texte pre lepšie pochopenie rozdelená na hrubú mzdu a čistú mzdu.

3.3.1 Hrubá mzda

Odmena zamestnanca, ktorá ešte nie je zdanená, sa zvykne nazývať hrubou mzdou. Čo všetko ju tvorí, je postupne zoradené a vysvetlené v nasledujúcom texte.

Základná zložka mzdy

Základná zložka je poskytovaná na základe odpracovaného času alebo dosiahnutého výkonu, pričom ide o pevnú zložku, ktorá sa nedá meniť bez písomnej dohody medzi zamestnancom a zamestnávateľom. [29]

Túto zložku teda chápeme ako peňažnú odmenu za merateľný a skutočne odpracovaný výkon zamestnanca. Táto odmena je vyhodnotená na základe dohodnutých taríf vo vzťahu k dohodnutým formám mzdy, ktoré sú popísané v podkapitole 3.1.1.

Pohyblivé zložky mzdy

Tieto zložky sú už variabilné a nadstavbové a v tomto prípade ich vyplatenie ako aj konkrétna výška závisí od splnenia kvantitatívnych alebo kvalitatívnych parametrov, prípadne na základe pracovnej disciplíny a podobne. Medzi pohyblivé zložky mzdy patrí osobné ohodnotenie, odmena ale aj prémia. [29]

Mzdové zvýhodnenia

Ďalšou zložkou hrubej mzdy sú mzdové zvýhodnenia alebo ľudovo nazývané príplatky. Výšky týchto príplatkov sa zvyknú často meniť – približne raz ročne.

Mzdové zvýhodnenia patria zamestnancovi podľa Zákonníka práce za prácu nadčas, prácu v noci, prácu v sobotu, prácu v nedeľu alebo prácu vo sviatok. [17, podľa §121-§123]

Náhrady mzdy

Náhradu mzdy môže čitateľ rozumieť ako peňažnú náhradu, resp. kompenzáciu za čas, ktorý zamestnanec neodpracoval.

Medzi náhrady mzdy patria najmä náhrady z dôvodu prekážok či už na strane zamestnanca alebo zamestnávateľa, napríklad náhrada za dovolenku, sviatok, pracovnú neschopnosť, čas na vyšetrenie v zdravotníckom zariadení a podobne. [29]

Iné zložky mzdy

Existuje veľa ďalších zložiek mzdy, ktoré môže zamestnávateľ dohodnúť so zamestnancom. Ide napríklad o:

- **Nepeňažný príjem** – v tomto prípade sa jedná o naturálnu mzdu vyjadrenú v peňažnej hodnote, [29]
- **Mimoriadne príjmy** – napríklad cestovné náhrady nad mieru stanovenú zákonom, odstupné, odchodné, odmeny pri životnom jubileu (tieto nie sú zákonom nárokovateľné, keďže ide o niečo „nad“ povinnosti vyplývajúce pre zamestnávateľa), [29]
- **Iné** – finančný príspevok na stravu alebo finančný doplatok aspoň do výšky minimálnej mzdy. [29]

Doplatok do minimálnej mzdy

Zamestnávateľ je povinný zamestnancovi poskytnúť mzdu najmenej v sume minimálneho mzdového nároku určeného pre stupeň náročnosti práce príslušného pracovného miesta. Ak mzda zamestnanca v kalendárnom mesiaci nedosiahne sumu minimálneho mzdového nároku, zamestnávateľ poskytne zamestnancovi doplatok v sume rozdielu medzi dosiahnutou mzdou a sumou minimálneho mzdového nároku ustanoveného pre stupeň patriaci príslušnému pracovnému miestu. [17, §121 ods.1]

3.3.2 Čistá mzda

Čistou mzdou sa rozumie mzda, ktorá je zdanená. Zdanenie mzdy sa vykonáva na základe rôznych zrážok, resp. zrážaním rôznych peňažných čiastok, vyrátaných z vymeriavacieho základu, ktorý predstavuje hrubú mzdu. V tejto sekcii sú vymenované a vysvetlené rôzne typy týchto zrážok.

Na to akú výšku majú jednotlivé zrážky zo mzdy, vplýva veľmi veľa informácií. Ide napríklad o druh pracovnoprávneho vzťahu (vysvetlené v kapitole 3.1), o uplatnenie rôznych výhod ako je odvodová výnimka alebo o poberanie niektorého dôchodku. Po zrazení všetkých potrebných zrážok vzniká čistá mzda, teda mzda ktorú zamestnanec od zamestnávateľa dostane.

Zrážky zo mzdy sa členia na tie, ktoré sa vykonávajú bez písomného súhlasu zamestnanca, a tie, na ktoré je potrebný súhlas zamestnanca. [22]

Prednostné zrážky bez písomného súhlasu zamestnanca

Zrážky, ktoré musí zamestnávateľ vykonať prednostne sú [22]:

- zrážky poistného na sociálne poistenie,
- zrážky preddavkov poistného na verejné zdravotné poistenie,
- zrážky nedoplatku z ročného zúčtovania preddavkov na verejné zdravotné poistenie,

- zrážky príspevku na doplnkové dôchodkové sporenie, ktoré platí zamestnanec podľa osobitného predpisu,
- zrážky preddavku na daň alebo dane, nedoplatku preddavku na daň, daňového nedoplatku, nedoplatku, ktorý vznikol zavinením daňovníka na preddavku na daň a na dani vrátane príslušenstva, a nedoplatku z ročného zúčtovania preddavkov na daň z príjmov zo závislej činnosti.

Následujúce zrážky bez písomného súhlasu zamestnanca

Tieto sumy sa zrážajú až po prednostných zrážkach, no vyskytujú sa veľmi výnimočne. Ide napríklad o zrážky [22]:

- preddavok na mzdu, ktorý je zamestnanec povinný vrátiť preto, že neboli splnené podmienky na priznanie tejto mzdy,
- sumy postihnuté výkonom rozhodnutia nariadeným súdom alebo správnym orgánom,
- peňažné tresty a pokuty, ako aj náhrady uložené zamestnancovi vykonateľným rozhodnutím príslušných orgánov.

Zrážky len s písomným súhlasom zamestnanca

Ďalšie zrážky zo mzdy môže zamestnávateľ vykonávať len [22]:

- Na základe písomnej dohody so zamestnancom o zrážkach zo mzdy – môže ísť napríklad o stravné lístky alebo uhradenie spôsobených škôd,
- na základe dohody o zrážkach zo mzdy a iných príjmov zamestnanca podľa § 551 zákona č. 40/1964 Zb. Občiansky zákonník. V tomto prípade ide o záväzkový právny vzťah medzi veriteľom (treťou osobou – právnickou alebo fyzickou osobou) a dlžníkom (zamestnancom).

Kapitola 4

Analýza a špecifikácia požiadaviek

Výpočet miezd zamestnancov je pre zamestnávateľa náročná a hlavne nárazová činnosť, resp. povinnosť, sprevádzaná mnohými ďalšími legislatívnymi povinnosťami a normami, z ktorých tie najdôležitejšie sú spomenuté v kapitole 3. Pre uľahčenie práce pri tejto povinnosti je dôležitý vznik informačného systému, ktorý dokáže niektoré etapy tohto zložitého procesu zautomatizovať.

V prvej sekcii kapitoly sú popísané časti a s nimi spojené funkcionality, ktoré by mali byť súčasťou systému. V tej ďalšej je uvedený diagram prípadov použitia, ktorý je výsledkom analýzy požiadaviek.

4.1 Časti informačného systému

V tejto podkapitole sú zhrnuté požiadavky na informačný systém, ktoré podrobne popisujú potrebnú funkcionalitu najdôležitejších časti celého informačného systému, a to aj v súvislosti s jednotlivými typmi jeho používateľov. Tieto požiadavky vzišli z ich analýzy a špecifikácie, ktoré patria k najdôležitejším častiam pri návrhu informačného systému.

4.1.1 Používatelia

V praxi sa však často zamestnávateľovi o výpočet miezd pre zamestnancov stará špecifický zamestnanec. Ide často o účtovníka, ktorý má odborné znalosti nie len v tejto legislatívnej problematike. Ten však nemá rovnaké právomoci ako samotný zamestnávateľ, preto je dôležité aj v informačnom systéme oddeliť práva rôznych typov používateľov. Keďže uchovávajúcimi údajmi sú dáta o jednotlivých zamestnancov, ktoré sú značne citlivé, prístup do systému budú mať len registrovaní používatelia.

Preto sa v systéme bude vyskytovať niekoľko rolí používateľov, ktorými sú:

- **admin**,
- **zamestnávateľ (riaditeľ)**,
- **účtovník**,
- **zamestnanec**.

Keďže by sa v systéme malo vyskytovať viacero používateľov, je dôležité aby niekto spravoval ich používateľské účty. O túto správu by sa mal starať používateľ s rolou admin.

Registrácia používateľov bude spadať výlučne len do kompetencií tejto správy, pretože ako je v úvode tejto kapitoly uvedené, neregistrovaný používateľ nemá do systému prístup v žiadnej forme. Tieto účty bude admin priraďovať zamestnancom, ktorí sú v informačnom systéme zaznamenaní. Okrem toho bude mať v kompetencii aj zmenu hesla, aktiváciu a tiež odstránenie používateľského účtu. Prihlasovacie konto používateľa bude možné označiť za neaktívne, aby sa napríklad zamedzilo prístupu bývalým zamestnancom firmy do systému.

4.1.2 Zamestnanci

Na koľko ide o zamestnanecký, výplatný informačný systém, tak je dôležité uchovávať záznamy o jednotlivých zamestnancov. Zamestnanec je základnou stavebnou jednotkou v tomto systéme. S ním sa neskôr uzatvárajú pracovnoprávne vzťahy.

Spravovanie zamestnancov a ich osobných údajov má na starosti **účtovník**. Ten najčastejšie v praxi rieši legislatívne úkony spojené s prijatím zamestnanca, a to na základe podmienok, ktoré zamestnávateľ s daným zamestnancom dohodne. K správe zamestnancov by mal mať prístup samozrejme aj samotný **zamestnávateľ**.

Zamestnancov by malo byť možné v informačnom systéme vytvoriť pomocou ich osobných údajov. K potrebným údajom by mali patriť bežné kontaktné údaje, ale rovnako aj údaje dôležité z hľadiska mzdy, ktorými sú napríklad počet zaopatrených detí či zdravotná poisťovňa, ktorej je zamestnanec poistený. Okrem vytvorenia záznamov o zamestnancoch by ich malo byť možné aj odstraňovať, upravovať a takisto prehľadávať.

4.1.3 Pracovnoprávne vzťahy

Pracovnoprávny vzťah je uzatváraný medzi zamestnávateľom a zamestnancom. Práve na základe tohto vzťahu a jeho dohodnutých podmienok najviac závisí výpočet výplaty zamestnanca. Je preto dôležité uchovať všetky jeho dohodnuté podmienky v informačnom systéme čo najefektívnejšie tak, aby na ich základe bolo možné správne vypočítať mzdu zamestnanca.

Prístup a možnosť spravovania záznamov o pracovnoprávnych vzťahov zamestnanca, by mal mať rovnako aj v tomto prípade **účtovník** i **zamestnávateľ**. V praxi sú to totiž oni, ktorí sa podieľajú na procesoch prijatia zamestnanca a uzatvárania pracovných vzťahov.

Tieto vzťahy je možné v rámci jedného zamestnanca prehľadávať, vytvárať, upravovať alebo ukončovať. U ľubovoľného zamestnanca by malo byť možné vytvoriť viacero pracovnoprávnych vzťahov, a to dokonca aj počas toho istého obdobia. V pracovnoprávnom vzťahu je takisto možné meniť jeho podmienky, pre prípad že sa na tom obe strany dohodnú. Spomenuté podmienky by mali obsahovať všetko podstatné, ako je pracovisko, pracovná pozícia, dohodnuté základne mzdy v spojitosti s mzdovými formami, výmeru dovolenky, informácie o uplatnení rôznych výhod zamestnanca v rámci tohto vzťahu atď.

4.1.4 Odpracovaný výkon

Dôležitou funkcionalitou je aj evidencia odpracovaného výkonu zamestnancov. Tá neskôr pomôže pri výpočte mzdy, určiť mzdu zamestnanca za odpracovaný výkon, pričom tento proces by mal byť plne automatizovaný. Táto funkcionalita by tak dopomohla ušetriť veľa času stráveného sčítavaním, ktoré je pomerne pracné.

Možnosť evidencie výkonu by mal mať každý **zamestnanec**, resp. všetky role používateľov, a to v rámci svojich pracovnoprávnych vzťahov. Okrem toho však **zamestnávateľ** môže spravovať a dohliadať na výkon každého zamestnanca, resp. každého používateľa v informačnom systéme.

Evidovaný výkon by sa u ľubovoľného používateľa mal priraďovať k mzdám, na ktorých sa používateľ, resp. zamestnanec dohodol so zamestnávateľom v rámci podmienok pracovnoprávneho vzťahu, keďže na týchto dohodnutých mzdách závisí, čo všetko musí používateľ v rámci svojho odpracovaného výkonu zaznamenať.

Pri evidencii by malo byť možné zaznamenať odpracovaný čas, odpracovaný výkon výkonovej mzdy, dosiahnutý percentuálny základ podielovej mzdy, prácu nadčas či pracovnú pohotovosť v jej všetkých formách. Chýbať nesmie ani dátum, kedy evidovaný výkon prebehol.

4.1.5 Neprítomnosti

Neprítomnosti zamestnanca v práci sú dôležite z hľadiska vyčíslenia náhrad mzdy pri výplate zamestnanca.

V reále neprítomnosť oznamuje zamestnanec svojmu vedúcemu, no pri niektorých typoch absencií musí zamestnanec priniesť tiež potvrdenie o svojej neprítomnosti účtovníkovi firmy, ktorý túto absenciu zaeviduje. Spravovanie záznamov o neprítomnostiach jednotlivých zamestnancov v systéme by mal mať teda na starosti **zamestnávateľ** aj **účtovník**.

V rámci absencie v práci je potrebné uviesť dňové rozmedzie, kedy táto neprítomnosť prebehla, a taktiež jej dôvod. V prípade ak bol zamestnanec v práci neprítomný len počas jedného dňa, je možné zaevidovať chýbanie len za polovicu daného dňa.

4.1.6 Legislatívne informácie

Z pohľadu legislatívy sa pri výpočte mzdy nachádza veľa informácii, ktoré mzdu zamestnanca ovplyvňujú, a ktoré sa zvyknú pomerne často meniť. Nejde o zmenu procesu výpočtu, ale hlavne o mnoho číselných konštánt a percentuálnych hodnôt. Je preto dôležité aby bolo možné v informačnom systéme takéto často sa meniace konštanty editovať, aby výpočet miezd ostal aktuálny aj pri týchto legislatívnych zmenách.

Používateľ v roli **zamestnávateľ** i **účtovník** bude teda oprávnený spravovať tieto legislatívne nastavenia, resp. údaje.

K týmto často sa meniacim údajom patria napríklad minimálna mzda, stupne náročnosti vykonávanej práce, výška príplatkov alebo rôzne mzdové konštanty, ako je napríklad maximálny denný vymeriavací základ. V rámci správy bude používateľ schopný tieto všetky informácie pridávať, meniť a odstraňovať.

4.1.7 Firemné informácie

Existujú aj informácie spojené s chodom firmy, ktoré síce výpočet mzdy priamo neovplyvňujú, ale vyskytujú sa v podmienkach pracovnej zmluvy. Ide o pracovnú pozíciu resp. druh práce, ktorý zamestnanec vykonáva a takisto pracovisko, na ktorom sa táto práca vykonáva.

Pridávať, upravovať a odstraňovať tieto záznamy by mal **zamestnávateľ** i **účtovník**. Na základe týchto informácií bude možné triediť zamestnancov pri ich prehľadávaní, čo môže používateľom s rolou zamestnávateľa a účtovníka značne pomôcť.

Pracovnej pozícií bude možné priradiť stupeň náročnosti, na základe ktorého sa neskôr pri výpočte mzdy zvolí minimálna mzda.

4.1.8 Výplata

Výpočet výplaty je najdôležitejšou ale tiež najnáročnejšou časťou systému. Informačný systém by mal proces výpočtu mzdy zautomatizovať tak, aby bol pre používateľa čo najjednoduchší a najmenej pracný. Tento proces by sa mal prispôsobiť aj rôznym podmienkam, ktoré má zamestnanec so zamestnávateľom dohodnuté.

Ako prvé by mal systém automaticky spočítať celkový odpracovaný výkon zamestnanca a na základe dohodnutých mzdových podmienok vypočítať základnú zložku mzdy. Následne by mal mať používateľ možnosť zraziť tiež pohyblivú zložku mzdy, ktorou je napríklad odmena. Taktiež by mal systém automaticky prepočítať množstvo času, za ktorý zamestnancovi patria príplatky a obvyklé náhrady mzdy. Vypočítanie hrubej mzdy dokončí automatizovaným prepočtom doplatku do minimálnej mzdy zamestnanca.

V ďalšej etape pre výpočet mzdy najprv systém zrazí z hrubej mzdy prednostné zrážky a ponúkne používateľovi možnosť pridať iné, neobvyklé zrážky zo mzdy. Nakoniec bude ponúknutá rekapitulácia celého prepočtu výplaty a dokončenie výpočtu v podobe uloženia záznamov o tejto výplate.

Okrem výpočtu by mal byť používateľ schopný dostať PDF-súbor s vygenerovanými výplatnými páskami pre zamestnancov.

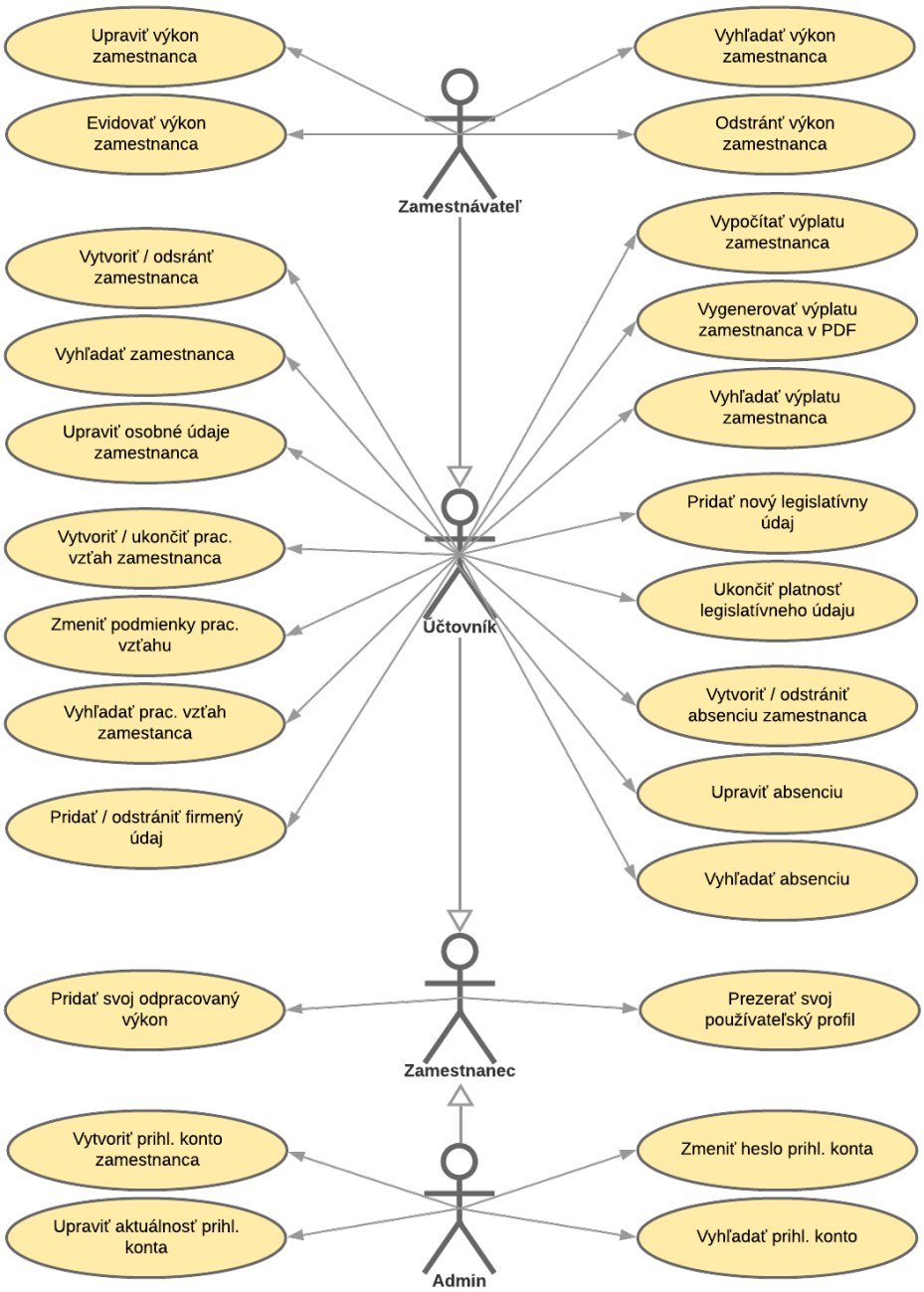
K funkcionalitám spojených s výplatami budú mať užívatelia s rolami **zamestnávateľ** a **účtovník**.

4.2 Diagram prípadov použitia

Diagram prípadov použitia (Use Case Diagram) umožňuje popísať chovanie systému z hľadiska používateľa. V diagrame prípadov použitia sa špecifikuje, aké typy používateľov (ľudia a iné systémy) používajú systém a aké činnosti vykonávajú. Model prípadov použitia sa skladá z diagramov prípadov použitia a špecifikácií (slovných popisov) jednotlivých prípadov použitia. Prípady použitia predstavujú funkčné požiadavky na vyvíjaný systém. [20, str.304]

Prvky diagramu prípadov použitia sú aktér (Actor), prípad použitia (Use case) a vzťah (Relation). Aktér prezentuje prvok okolia systém, ktorý komunikuje so systémom. Aktérom nemusí byť iba rola reprezentovaná živou osobou, ale tiež externý systém, s ktorým modelovaný systém komunikuje. Aktér sa v UML označuje symbolom panáka a názvom. Aktérom môže byť napríklad správca objednávok, študent, učiteľ či bankový systém. Prípad použitia (Use case) špecifikuje časť funkcionality systému, ktorú využíva aktér, a ktorá plní určitý cieľ. Názov prípadu použitia by tento cieľ mal vyjadrovať a používa sa pre neho slovesná väzba - napríklad "Otvoriť účet". [20, str.304]

Na obrázku nižšie (obr.4.1) je možné vidieť diagram prípadov použitia, ktorý vznikol ako výsledok analýzy a špecifikácie požiadaviek na informačný systém, ktorého vývojom sa zaoberá táto práca.



Obr. 4.1: Diagram prípadov použitia vyvíjaného informačného systému (zdroj: autor)

Kapitola 5

Návrh systému

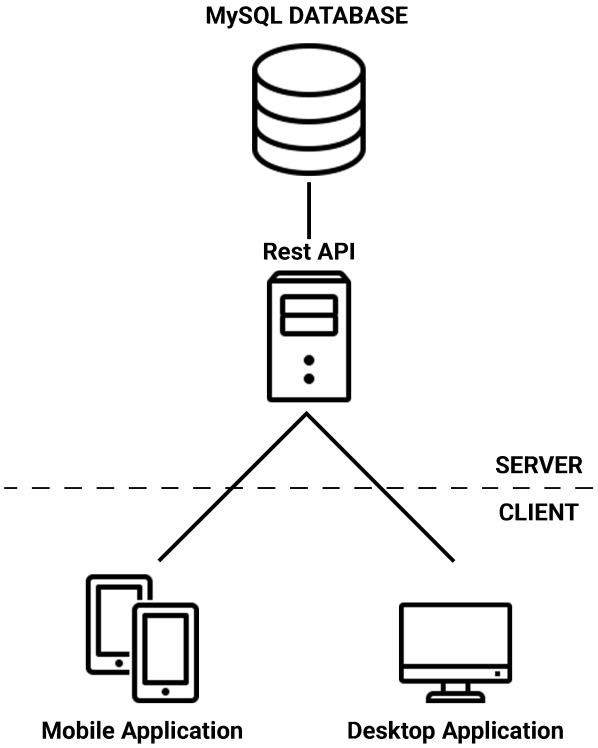
V tejto kapitole je predstavený návrh informačného systému, ktorého súčasťou je zvolenie architektúry systému, dátový návrh systému a tiež návrh používateľského rozhrania klientských aplikácií. Ako dátový návrh je v tejto časti predstavený ER diagram, ktorý bol neskôr prevedený do tabuliek relačnej databázy. Čo sa týka návrhu používateľského rozhrania, popísaný je predovšetkým menu klientských aplikácií a ich najdôležitejšie časti.

5.1 Výber architektúry

V sekcii 2.3 v tejto práci boli predložené a vysvetlené najzákladnejšie a najpoužívanejšie fyzické architektúry informačných systémov. Cieľom tejto práce je vyvinúť informačný systém, ktorý by mal obsahovať klientskú, desktopovú aplikáciu implementovanú v programovacom jazyku Java, určenú na výpočet miezd a zároveň by mal obsahovať klientskú, mobilnú aplikáciu, určenú pre evidenciu dochádzky jednotlivých zamestnancov.

Na základe týchto informácií bola zvolená architektúra **klient-server** s hrubými klientmi. Týmto spôsobom bude využitá vysoká úroveň jazyka Java, a teda základná logika bude umiestnená na klientskej desktopovej aplikácií, k čomu nie je potrebné zavádzať zvlášť aplikačnú a dátovú vrstvu. Architektúru je vhodné použiť aj na základe poznatku, že desktopová klientská aplikácia nebude v rámci firmy zamestnávateľa využívaná na veľkom počte zariadení. Mobilná klientská aplikácia by mala byť využívaná len na ukladanie dát o odpracovanom výkone zamestnanca na databázový server, a takisto na prezentáciu základných údajov o zamestnancovi.

Keďže ale bude k databázovému serveru pristupovať viacero rolí používateľov, ktorí majú rôzne oprávnenia, je potrebné z hľadiska bezpečnosti určitým spôsobom riadiť prístup k databázovému serveru. Tento problém bol vyriešený zavedením rozhrania architektúry REST k databázovému serveru. Celý návrh architektúry je možné vidieť na blokovej schéme tejto architektúry na obrázku 5.1.



Obr. 5.1: Bloková schéma architektúry informačného systému (zdroj: autor)

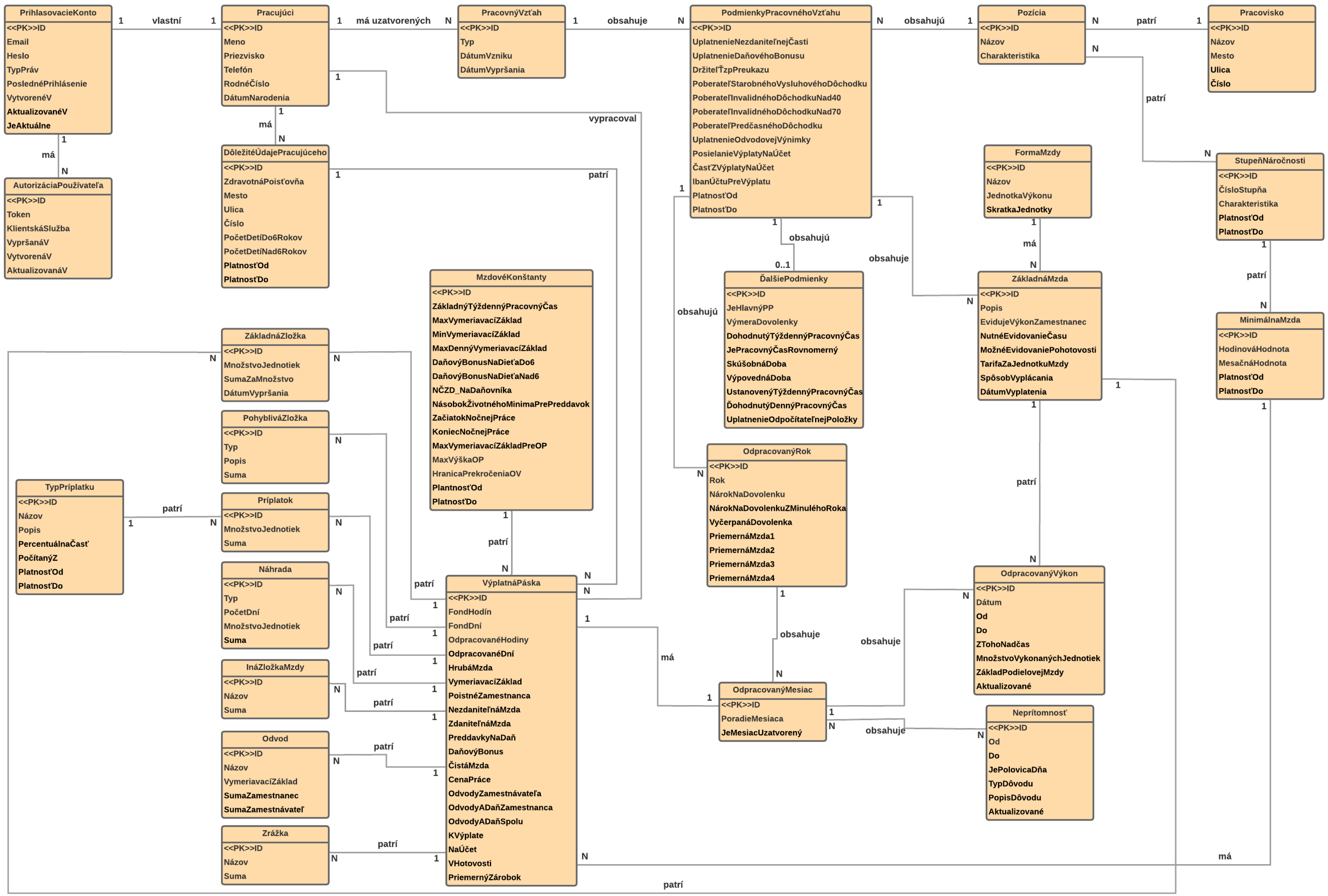
5.2 ER diagram

Diagram entít a vzťahov (ERD) je grafický nástroj, používaný k vyjadreniu dátových objektov - **entít** (Entity), ich podstatných **vzťahov** (Relationship) a podstatných vlastností - **atribútov** (Atribute) týchto objektov a vzťahov. [20]

Entita je vec, ktorá môže byť samostatne identifikovaná. Príkladom entity môžu byť konkrétna spoločnosť, osoba, udalosť a podobne. Entity sa na základe podobností zlučujú do **entitných množín** (typov entít), napríklad zamestnanec, oddelenie. Každá entitná množina musí mať uvedený identifikátor, teda minimálnu množinu atribútov, ktoré zaisťujú jednoznačnú identifikáciu entít v tejto množine. [20]

Atribút je vlastnosť, resp. charakteristika konkrétnej entity alebo vzťahu. Môže ísť napríklad o číslo oddelenia, názov oddelenia alebo meno osoby. Vzťah je nejaké spojenie medzi entitami. Napríklad Karol Novák je členom oddelenia mzdového účtovníctva. Je možné, že niektorí ľudia budú vidieť nejakú vec ako entitu, zatiaľ čo iní to vidia ako vzťah. V takomto prípade ide o dohodu, ako to bude modelované. [20]

V nasledujúcom texte je predstavený dátový návrh systému, resp. ER diagram (obr.5.2) vyvíjaného informačného systému tejto práce a popis jednotlivých entitných množín a vzťahov medzi nimi.



Obr. 5.2: ER diagram vyvíjaného systému (zdroj: autor)

5.2.1 Pracujúci – Dôležité údaje pracujúceho

Entitná množina *pracujúci* by mala uchovávať základné osobné a kontaktné údaje o jednotlivých pracujúcich, ktorí majú alebo mali so zamestnávateľom uzatvorený určitý pracovnoprávny vzťah.

Každý pracujúci má aj dôležité údaje z pohľadu výpočtu mzdy. Tieto údaje sa ale zvyknú v priebehu času meniť, preto bola vytvorená samostatná entitná množina *dôležité údaje pracujúceho*, ktorá tieto údaje obsahuje. Množina obsahuje atribúty *platnosť od* a *platnosť do*, ktoré ohraničujú platnosť údajov.

Keďže *pracujúci* môže mať niekoľko *dôležitých údajov pracujúceho*, vzťah medzi nimi je typu 1:N.

5.2.2 Prihlasovacie konto – Autorizácia používateľa

Každý pracujúci môže vlastniť jedno prihlasovacie konto, preto entitné množiny *prihlasovacie konto* a *pracujúci* sú vo vzťahu typu 1:1.

Entitná množina *prihlasovacie konto* obsahuje atribúty, dôležité pre prihlásenie pracujúceho do systému. Atribút *typ práv* označuje rolu prihlasovacieho konta, zatiaľ čo atribút *je aktuálne* označuje aktuálnosť konta, a teda ponúka informáciu o tom, či sa je možné prostredníctvom daného konta prihlásiť do informačného systému.

Autorizácia používateľa je entitnou množinou určenou pre zaznamenávanie údajov o prihlásení jednotlivých prihlasovacích kont, resp. používateľov. Na základe nej by malo byť možné overiť, či požiadavky prichádzajú už od autorizovaných používateľov. Atribút *token* je jedinečná postupnosť znakov slúžiaca pre overenie prihlásenia používateľa, kým atribút *klientská služba* označuje službu z ktorej sa používateľ prihlásil.

Pretože sa môže používateľ z prihlasovacieho konta prihlásiť viacero krát aj z rôznych služieb je medzi spomenutými entitnými množinami vzťah 1:N.

5.2.3 Pracovisko – Pozícia

Entitná množina *pracovisko* by mala uchovávať údaje o jednotlivých pracoviskách firmy.

Pozícia je entitnou množinou definujúca jednotlivé pracovné pozície v rámci firmy. Jej atribút *charakteristika* by mala obsahovať stručný popis činnosti práce.

Pracovná pozícia patrí pod jedno pracovisko, zatiaľ čo na pracovisku môže byť viacero pracovných pozícií. Preto medzi týmito vzťahovými množinami ide o vzťah 1:N.

5.2.4 Stupeň náročnosti – Minimálna mzda

Ďalšou entitnou množinou je *stupeň náročnosti*. Zobrazuje stupne náročnosti práce vyplývajúce z legislatívy Slovenskej republiky. Pre prípad zmeny tejto legislatívnej normy sú v entitnej množine prítomné atribúty ohraničujúce platnosť jednotlivých stupňov. Aj preto je možné, že stupňov náročnosti bude vo vzťahu k entitnej množine *pozícia* viacero, a teda vzťah medzi týmito množinami je M:N.

Minimálna mzda je entitná množina, ktorá popisuje minimálnu mzdu vychádzajúcu z legislatívy Slovenskej republiky. Keďže sa v praxi minimálna mzda často mení, aj v tomto prípade je ohraničená platnosť údajov entity.

Keďže minimálna mzda sa priraďuje k stupňu náročnosti a tiež sa zvykne často meniť, vzťah medzi týmito entitnými množinami je 1:N.

5.2.5 Forma mzdy – Základná mzda

Entitná množina *forma mzdy* odzrkadľuje formy mzdy, ktorými môže byť zamestnanec platený. Atribút *jednotka výkonu* definuje merateľnú jednotku, v ktorej je meraný odpracovaný výkon zamestnanca.

Ďalšou entitnou množinou v dátovom modele je *základná mzda*. Táto množina zobrazuje základné mzdy dohodnuté v zmluve medzi zamestnávateľom a zamestnancom. Atribút *nutné evidovanie času* definuje, či je pri evidovaní výkonu pri tejto mzde nutné evidovať čas, zatiaľ čo *možné evidovanie pohotovosti* indikuje možnosť evidovania pohotovosti pri odpracovanom výkone zamestnanca. *Tarifa za jednotku mzdy* určuje dohodnutú tarifu, ktorou je zamestnanec platený na jednotku výkonu.

Každá základná mzda obsahuje jednu formu mzdy, zatiaľ čo môže existovať viacero základných miezd s tou istou formou mzdy. Preto je typ vzťahu medzi týmito množinami 1:N.

5.2.6 Pracovný vzťah – Podmienky pracovného vzťahu – Ďalšie podmienky

Pracovný vzťah je entitná množina predstavujúca pracovnoprávne vzťahy, ktoré pracujúci uzavrie so zamestnávateľom. Atribút *typ* pracovného vzťahu vyjadruje druh pracovnoprávneho vzťahu, ktorý obe strany dohodli. Keďže pracovný vzťah uzatvára zamestnávateľ s pracujúcim, tak je táto entitná množina vo vzťahu s entitnou množinou *pracujúci*. Je medzi nimi vzťah 1:N, pretože pracujúci môže mať so zamestnávateľom uzatvorených viacero pracovnoprávnych vzťahov.

Ďalšou entitnou množinou, ktorá súvisí s pracovnoprávnym vzťahom je množina *podmienky pracovného vzťahu*. Táto množina obsahuje atribúty popisujúce dohodnuté podmienky, ktoré sú rovnaké pre všetky typy pracovnoprávnych vzťahov. Aj oni sa ale môžu v rámci tohto vzťahu meniť, a preto majú prostredníctvom atribútov ohraničenú platnosť.

Keďže podmienok môže byť v rámci pracovného vzťahu viacero, je medzi týmito dvomi entitnými množinami vzťah 1:N.

Entitná množina *ďalšie podmienky* zobrazuje dohodnuté podmienky v rámci pracovného vzťahu, ktoré sa uzatvárajú len v prípade pracovného pomeru. Vzťah medzi entitnou množinou *ďalšie podmienky* a množinou *podmienky pracovného vzťahu* je 1:1 s tým, že existencia entity *ďalšie podmienky* nie je nevyhnutná.

5.2.7 Odpracovaný rok – Odpracovaný mesiac

Entitná množina *odpracovaný rok* zobrazuje odpracované kalendárne roky pracujúceho v rámci jeho pracovnoprávneho vzťahu. Atribút *rok* definuje číslo konkrétneho kalendárneho roku. Táto entitná množina je vo vzťahu typu 1:N s množinou *podmienky pracovného vzťahu*, keďže v rámci dohodnutých podmienok pracovného vzťahu môže odpracovať pracujúci aj niekoľko rokov.

Odpracovaný mesiac je entitnou množinou, ktorá znázorňuje odpracované kalendárne mesiace pracujúceho v rámci jeho odpracovaných rokov v pracovnoprávnom vzťahu. Atribút *poradie mesiaca* znázorňuje číslo poradia kalendárneho mesiaca, zatiaľ čo atribút *je mesiac uzatvorený* hovorí o tom, či je pre daný mesiac výplata pracujúceho vypočítaná.

Keďže pracujúci môže v rámci odpracovaného roku odpracovať niekoľko mesiacov, vzťah medzi týmito entitnými množinami je 1:N.

5.2.8 Odpracovaný výkon a Neprítomnosť

Ďalšou entitnou množinou je *odpracovaný výkon*, ktorý predstavuje záznamy odpracovaných výkonov zamestnanca v rámci jeho pracovnoprávneho vzťahu a dohodnutých základných miezd. Atribúty *od* a *do* sú časové hodnoty ohraničujúce výkon v rámci jedného dňa. Atribút *množstvo vykonaných jednotiek* je hodnota výkonu pre výkonovú formu mzdy, zatiaľ čo *základ podielovej mzdy* je dosiahnutá hodnota percentuálneho základu pre podielovú formu mzdy.

Táto entitná množina je vo vzťahu 1:N s množinou *odpracovaný mesiac*, pretože odpracovaný mesiac môže mať niekoľko záznamov odpracovaného výkonu. Takisto je vo vzťahu s množinou *základná mzda* a na základe toho vzťahu je definované, ku ktorej dohodnutej mzde výkon patrí. Tento vzťah je tiež typu 1:N.

Entitná množina *neprítomnosť* by mala slúžiť k ukladaniu záznamov o neprítomnostiach, resp. absenciách pracujúceho v rámci jeho pracovnoprávneho vzťahu. Atribúty *od, do* ohraničujú dňové rozmedzie neprítomnosti, takže ide o rozmedzie dátumov. Atribút *je polovica dňa* zase určuje, či absencia nastala len počas polovice pracovného dňa v prípade, že dňové rozmedzie ohraničuje jeden deň.

Vzťah medzi touto entitnou množinou a množinou odpracovaný mesiac je typu M:N, pretože mesiac môže obsahovať viacero neprítomností a zároveň jedna neprítomnosť môže trvať dlhšie ako jeden kalendárny mesiac.

5.2.9 Mzdové konštanty a Typ príplatku

Mzdové konštanty sú dôležitou entitnou množinou, ktorá zobrazuje tie najdôležitejšie konštanty spojené s výpočtom výplat pracujúcich, vyplývajúcich z legislatívy Slovenskej republiky. Tieto konštanty sa v praxi zvyknú často meniť, preto aj v tejto množine sa nachádzajú atribúty pre ohraničenie platnosti údajov.

Entitná množina *typ príplatku* je tiež množina, ktorá by mala slúžiť k uchovávaniu často sa meniacich legislatívnych údajov. V tomto prípade ide o výšku jednotlivých príplatkov. Atribút *percentuálna časť* definuje výšku percentuálnej časti výpočtu príplatku a atribút *počítaný z* definuje druh sumy (napr. minimálna mzda), ktorá je percentuálnym základom výpočtu výšky príplatku.

5.2.10 Výplatná páska a jej jednotlivé zložky

Výplatná páska je entitná množina, ktorá by mala uchovávať záznamy o výplatných páskach a ich dôležitých sumách a častiach. Ide o veľké množstvo atribútov, z ktorých sa väčšina aj v reále musí nachádzať na výplatnej páske. Táto entitná množina je vo vzťahu s niekoľkými množinami. Vo vzťahu 1:1 je *výplatná páska* s množinou *odpracovaný mesiac*, pretože každá výplatná páska patrí k jednému odpracovanému mesiacu. Ďalej sa každá *výplatná páska* viaže k jednej konkrétnej *minimálnej mzde* a k jedným konkrétnym *mzdovým konštantám*, preto je s ich entitnými množinami vo vzťahu 1:N. S takýmto vzťahom je výplatná páska aj s entitnými množinami *pracujúci* a *dôležité údaje pracujúceho*. Vzťah s touto prvou spomenutou entitnou množinou označuje pracujúceho, ktorý výplatu vypracoval, zatiaľ čo vzťah s druhou spomenutou množinou označuje pracujúceho, ktorému výplatná páska patrí.

Výplatná páska je vo vzťahu aj s ďalšími entitnými množinami, ktoré predstavujú jednotlivé zložky mzdy. Aj tie by mali slúžiť na uchovávanie údajov, ktoré sa vyskytujú na výplatnej páske. Medzi tieto entitné množiny patria *základná zložka, pohyblivá zložka, príplatok, náhrada, iná zložka mzdy, odvod* a *zrážka*. Všetky tieto množiny obsahujú atribút

suma, vyjadrujúci hodnotu, ktorá vplýva na mzdu pracujúceho. K jednej výplatnej páske sa môže viazať viacero konkrétnych zložiek mzdy, preto sú tieto entitné množiny vo vzťahu 1:N s množinou *výplatná páska*.

5.3 Schéma relačnej databázy

Aby bolo možné dátový model ER diagramu aj neskôr implementovať, je ho nutné najprv previesť do schémy relačnej databázy. Je tomu tak preto, že ER diagram slúži hlavne pre komunikáciu so zákazníkom, a teda zobrazuje dáta v ľudsky zrozumiteľnom tvare.

Pre prevedenie ER diagramu do schémy relačnej databázy, je nutné správne transformovať jednotlivé typy vzťahov medzi jednotlivými entitami, ktorými sú:

- **Vzťah 1:1** – Do tabuľky ľubovoľne vybranej entitnej množiny sa pridá cudzí kľúč, ktorý odkazuje na primárny kľúč druhej tabuľky.
- **Vzťah 1:N** – Do tabuľky entitnej množiny, ktorej patrí kardinalita N sa vloží cudzí kľúč tabuľky entitnej množiny s kardinalitou 1.
- **Vzťah M:N** – Vyžaduje vznik takzvanej väzobnej tabuľky, ktorý obsahuje väčšinou 2 atribúty. Tieto atribúty sú cudzími kľúčmi, ktoré by mali odkazovať na primárne kľúče tabuliek obidvoch entitných množín.

5.4 Návrh používateľského rozhrania

Jednou fázou pri návrhu aplikácie, resp. informačného systému, je aj návrh používateľského rozhrania. V tomto prípade je veľmi dôležité aby používateľské rozhranie bolo pre používateľa čo najjednoduchšie na používanie.

Najprv je v tejto podkapitole priblížené používateľské rozhranie pre používateľa desktopovej klientskej aplikácie a následne je stručne popísané používateľské rozhranie jednoduchej mobilnej aplikácie.

5.4.1 Desktopová aplikácia

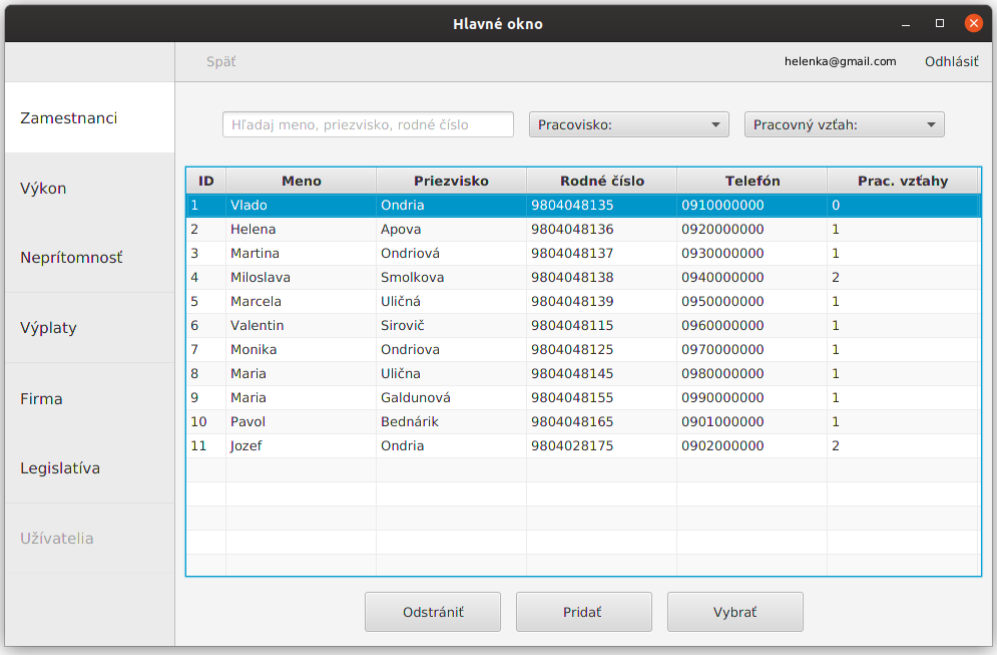
Desktopová aplikácia by mala slúžiť pre používateľov s rolami admin, zamestnávateľ a účtovník. Pre úspešné spustenie hlavnej časti, resp. hlavného okna aplikácie sa títo používatelia musia najprv úspešne prihlásiť.

Ako je vidno nižšie na obrázku 5.3, užívateľské rozhranie tejto aplikácie je riešené jednoducho, použitím farieb – odtieňov šedej. Hlavné okno aplikácie sa skladá z **horného navigačného panelu**, **postranného menu** a **hlavného obsahu**, určeného pre jednotlivé časti aplikácie.

Horný navigačný panel obsahuje tlačidlá pre odhlásenie používateľa a krok späť, ktoré samozrejme nie je dostupné za každých okolností. Rovnako sa v tomto paneli vyskytuje e-mailová adresa používateľa, ktorá slúži ako identifikátor práve prihláseného používateľa.

Postranné menu je umiestené v ľavej časti hlavného okna. Obsahuje tlačidlá, ktorými sa používateľ dostane do rôznych hlavných častí tejto aplikácie.

Čo sa týka hlavného obsahu jednotlivých hlavných častí aplikácie, vo väčšine prípadov pozostávajú z prvkov, ktoré umožňujú prehľadávanie údajov, tabuliek pre zobrazenie údajov a tlačidiel pre možné akcie s údajmi.



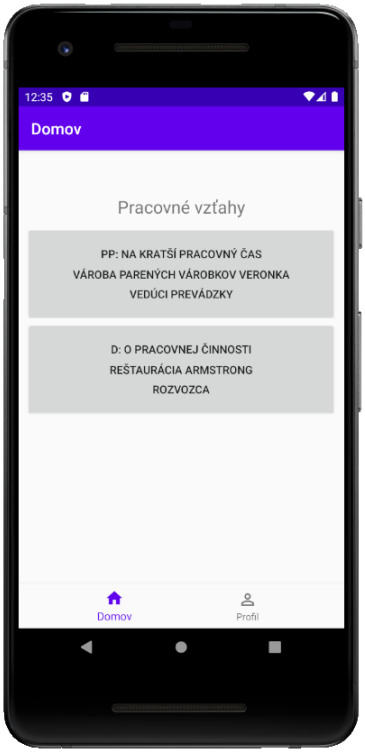
Obr. 5.3: Náhľad hlavného okna desktopovej aplikácie (zdroj: autor)

5.4.2 Mobilná aplikácia

Ďalšia klientská aplikácia vyvíjaná pre mobilné zariadenia je určená pre všetky role používateľov informačného systému pre evidenciu vlastného odpracovaného výkonu.

Na obrázku nižšie 5.4 je možné vidieť, že používateľské rozhranie sa skladá z **dolného navigačného panelu** a **hlavného obsahu**. Navigačný panel obsahuje tlačidlo *profil* pre náhľad používateľského profilu a tlačidlo *domov* pre operácie spojené s evidenciou výkonu zamestnanca.

Po prihlásení sa používateľ dostane k obrazovke, ktorú je vidno na obrázku 5.4. V nej si môže prostredníctvom tlačidiel zvoliť pracovnoprávny vzťah a následne základnú mzdu vzťahu, ku ktorej chce pridať odpracovaný výkon. Ten môže pridať pomocou formulára, ktorý je vidno na obrázku 5.5.



Obr. 5.4: Náhľad obrazovky po prihlásení (zdroj: autor)

Obr. 5.5: Náhľad formulára pre evidenciu výkonu (zdroj: autor)

Kapitola 6

Implementácia

V tejto kapitole sú zobrazené najdôležitejšie etapy implementácie tohto informačného systému. Najprv sú uvedené a vysvetlené vybrané technológie pre vývoj systému a následne je popísaná implementácia systému, či už z pohľadu vrstiev zvolenej, resp. použitej architektúry alebo z pohľadu najdôležitejších častí informačného systému.

6.1 Použité technológie

Jednou z častí implementácie informačného systému je výber technológií, ktoré by vyhovovali účelu a požiadavkám systému. V tejto podkapitole sa nachádza nie len výpis vybraných technológií, ale aj dôvod ich výberu, a takisto ich priblíženie čitateľovi. Technológie sú rozdelené podľa vrstiev použitej architektúry.

6.1.1 Serverová časť

Ako bolo už spomenuté v podkapitole 5.1, na strane serveru by sa malo okrem **databázového systému** nachádzať aj **serverové rozhranie**.

Ako databázový systém bol zvolený **MySQL**, ktorý je relačným databázovým systémom, a ktorý aj v neplatenej, voľne dostupnej verzii ponúka vysokú rýchlosť a širokú funkcionalitu. Tento databázový systém je detailnejšie popísaný v podkapitole 2.4.1.

Najmä kvôli riadeniu prístupu používateľov k databázovému serveru a ich oprávnení bolo nutné zavedenie REST rozhrania databázového serveru. Pre vývoj tejto architektúry bol zvolený programovací jazyk **PHP**, ktorý patrí k najrozšírenejším technológiám používaným na aplikačných serveroch, a ktorý bol popísaný v podkapitole 2.4.2. V rámci tohto programovacieho jazyka bol zvolený aj framework CodeIgniter.

REST architektúra

REST (Representational state transfer) je štýl softvérovej architektúry pôvodne predstavený Royom Fieldingom, ktorý bol jedným z hlavných autorov protokolu HTTP verzií 1.0 a 1.1. Rovnako ako v prípade HTTP aj REST očakáva komunikáciu klienta a servera, resp. komunikáciu postavenú na požiadavkách a odpovediach. Pri zasielaní požiadaviek sa využívajú dotazovacie metódy protokolu HTTP. To je opakom iných systémov, ako je napríklad SOAP, kde používatelia môžu definovať svoje vlastné metódy. Cesta URI sa v REST architektúre používa na prístup k určitým zdrojom alebo ich zmene. Zdrojmi môžu byť napríklad dáta alebo stavy aplikácie. [23, str.94]

REST využíva tieto HTTP požiadavky [23, str.94]:

- **GET** – pre získanie aktuálnych dát od servera.
- **PUT** – pre modifikáciu dát na serveri.
- **POST** – pre vytvorenie nových dát na serveri.
- **DELETE** – na odstránenie potrebných dát na serveri.

Tieto HTTP metódy pokrývaju všetky CRUD (CREATE/READ/UPDATE/DELETE) operácie, ktoré je možné vykonávať nad záznamami na úložisku, ktorým je najčastejšie databáza. REST používa pre prenos dát niekoľko štandardizovaných formátov, ako je napríklad JSON. [23, str.94]

CodeIgniter

CodeIgniter je jedným z PHP frameworkov, ktorý dovoľuje vývojárovi pomerne jednoducho a rýchlo vytvoriť webovú aplikáciu. Tento framework bol vybraný pre vytvorenie REST rozhrania pre databázový server a bol vybraný preto, že má nízku náročnosť používania a preto, že s ním má vývojár najväčšie skúsenosti.

CodeIgniter je vysokovýkonný a open source PHP framework pre tvorbu webových aplikácií, postavený na architektúre MVC (Model-View-Controller). Obsahuje bohatú sadu funkcionality s mnohými knižnicami pre posielanie e-mailov, nahrávanie súborov, pripojenie k databáze a s mnohými pomocnými triedami, ktoré ohromne zvyšujú rýchlosť vývoja webovej aplikácie, čím prispieva k zníženiu ceny a času vývoja. [32, str.1]

Tento framework bol vyvinutý spoločnosťou British Columbia Institute of Technology. Je licencovaný MIT licenciou a dostupný GitHube, čo znamená že je ho možné voľne použiť. [32, str.1]

Čo sa týka MVC architektúry, tak ovládač (controller) spracováva všetky požiadavky od klienta, model sa stará o operácie súvisiace s databázou a pohľad (view) generuje výstup, teda obsah odpovede. Použitie contorllerov je povinné, zatiaľ čo použitie pohľadov (view) a modelov (model) povinné nie je. To dáva vývojárovi veľkú voľnosť pri tvorbe aplikácií. [32, str.1]

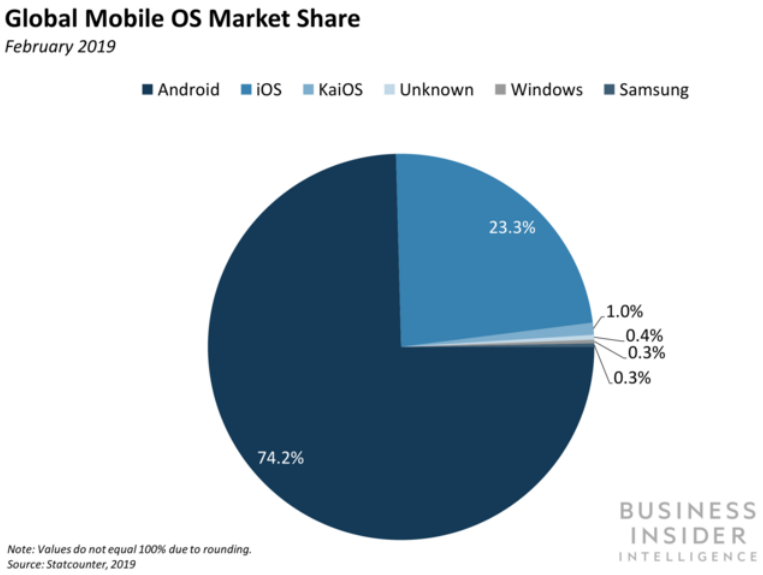
6.1.2 Klientská časť

Na klientskej časti architektúry klient-server by mali pracovať dve samostatné aplikácie. Prvou je desktopová aplikácia určená pre zamestnávateľa a tou druhou je mobilná aplikácia pre každého zamestnanca. Keďže o komunikáciu s klientmi sa bude starať REST serverové rozhranie, klienti by mali predkladať požiadavky serveru prostredníctvom protokolu HTTP a jeho požiadaviek, a tiež by mali byť schopní spracovávať odpovede prostredníctvom tohto protokolu.

Desktopová aplikácia by mala slúžiť hlavne na výpočet miezd zamestnancov, no okrem toho by v nej mal mať zamestnávateľ (alebo je ho poverený pracovník) možnosť upravovať legislatívu, firemné informácie, spravovať zamestnancov a ich pracovnoprávne vzťahy a mnohé ďalšie funkcionality, ktoré majú byť súčasťou informačného systému, a ktoré boli špecifikované v kapitole 4. Ako bolo spomenuté vyššie táto aplikácia by mala byť implementovaná v programovacom jazyku **Java**. Tento programovací jazyk bol priblížený v podkapitole 2.4.2. Dôležitou časťou tejto aplikácie je používateľské rozhranie, pre ktorého

tvorbu bola vybratá technológia JavaFX. JavaFX je softvérová platforma, ktorá dovoľuje vytvoriť Java aplikáciu s moderným používateľským rozhraním.

Čo sa týka **mobilnej aplikácie**, tá by mala slúžiť každému zamestnancovi pre evidenciu svojho vlastného odpracovaného výkonu. Ako je možné vidieť na obrázku 6.1, v mesiaci február v roku 2019 na trhu medzi mobilnými zariadeniami podielom stále jednoznačne víťazí operačný systém Android. Preto bol tento operačný systém, resp. mobilná platforma zvolená pre vývoj mobilnej, klientskej aplikácie pre účely tohto informačného systému.



Obr. 6.1: Podiel mobilných operačných systémov na trhu (zdroj: [18])

Android

Android je v súčasnosti najpopulárnejší operačný systém pre mobilné zariadenia. Preto je táto platforma obrovskou výzvou pre vývojárov aplikácií. Viac ako miliarda majiteľov smartphonov a tabletov s týmto operačným systémom má obrovský trhový potenciál nielen pre platené, ale aj voľné stiahnuteľné aplikácie. [28, str.15]

Na rozdiel od iOS, kde potrebujete na vývoj, aspoň v konečnej fáze, počítač s operačným systémom OS X, na vývoj aplikácie pre Android môžete použiť vývojársky počítač akejkoľvek platformy, teda Windows, Linux aj OS X. [28, str.15]

Problém nepredstavuje ani etapa testovaní, keďže nie je dôležité otestovať aplikácie na "vlajkovej lodi", tam určite nebude mať žiadne problémy s výkonom, ale skôr na lacných, menej výkonných telefónoch, ktoré sú samozrejme medzi ľuďmi oveľa viac rozšírené. [28, str.15]

Pre vývoj mobilnej aplikácie na platforme Android v tomto projekte bol taktiež zvolený programovací jazyk **Java**.

6.1.3 Ďalšie použité technológie

Okrem technológií, prostredníctvom ktorých bol informačný systém priamo vyvíjaný, sa pri vývoji využili aj softvérové aplikácie, ktoré tento vývoj zjednodušili. Tie budú predstavené v nasledujúcom texte.

Apache server

Projekt Apache serveru je úsilie vyvinúť a udržať voľne dostupný HTTP server pre moderné, operačné systémy zahrňujúce UNIX či Windows. Jeho cieľom je poskytnúť bezpečný, efektívny a rozšíriteľný server, ktorý poskytuje služby HTTP v súlade s aktuálnymi štandardmi HTTP protokolu. Apache server vznikol v roku 1995 a bol dlho najpopulárnejším webovým serverom na internete. [16]

V tomto projekte bol použitý ako lokálny server pre vývoj REST rozhrania serveru a pri vývoji celého informačného systému. Lokálny server Apache spoločne s lokálnym MySQL serverom boli použité v rámci softvérovej aplikácie **XAMPP**, ktorá umožňuje tieto lokálne servery.

Postman

Postman je Google Chrome aplikácia na interakciu s HTTP aplikačnými rozhraniami (servermi). Poskytuje používateľovi priaznivé grafické rozhranie na vytváranie žiadosti a čítanie odpovedí HTTP protokolu. [1]

Pri vývoji systému sa táto aplikácia využila pri testovaní správnej funkčnosti REST serverového rozhrania.

Android Studio Emulator

Pri vývoji klientskej aplikácie na platforme Android bolo nutné aplikáciu nejakým spôsobom priebežne testovať bez toho, aby ju bolo nutné inštalovať na mobilné zariadenie. K tomuto účelu bol použitý softvérový nástroj Android Studio Emulator, ktorý je súčasťou vývojového prostredia Android Studio.

Tento nástroj simuluje zariadenia Android na počítačovom zariadení používateľa, vďaka čomu je schopný testovať svoju aplikáciu na rôznych Android zariadeniach, bez toho aby muselo byť zariadenie Android fyzicky použité. [14]

Git

Pre zálohu jednotlivých verzií projektu bol využitý systém správy verzií nazývaný ako Git. Okrem toho, že poskytuje históriu verzií projektov, je Git nápomocný aj pri zdieľaní zdrojového kódu medzi jednotlivými členmi vývojárskeho tímu pri tímových projektoch.

Existuje mnoho takýchto systémov v podobe webových služieb. Pre tento projekt bola využitá webová služba **GitHub**.

6.2 Implementácia serverovej časti

Ako bolo spomenuté v podkapitole 6.1.1, pre implementáciu serverového rozhrania systému bol zvolený PHP framework CodeIgniter, zatiaľ čo pre databázový systém technológia MySQL. V tejto podkapitole sú zhrnuté najzákladnejšie časti implementácie týchto použitých technológií.

6.2.1 Adresárová štruktúra frameworku CodeIgniter

Koreňový adresár frameworku obsahuje viacero pod-adresárov a súborov. Z hľadiska implementácie aplikácie nás v tomto frameworku zaujíma hlavne pod-adresár s názvom

`application`. Tento adresár obsahuje ďalšie pod-adresáre, z ktorých tie najdôležitejšie pre túto serverovú aplikáciu sú:

- `config` – obsahuje konfiguračné súbory aplikácie,
- `logs` – adresár určený pre logovacie súbory,
- `helpers` – adresár určený pre pomocné triedy, resp. funkcie aplikácie,
- `controllers` – adresár určený pre ovládače (controllers), patriace do MVC architektúry,
- `models` – adresár určený pre modely, patriace do MVC architektúry,
- `views` – adresár určený pre pohľady (views), patriace do MVC architektúry.

Adresár `config`

Ako je spomenuté vyššie, adresár `config` zahŕňa konfiguračné súbory pre vývoj serverovej aplikácie. K najpodstatnejším konfiguračným súborom patria:

- `autoload.php` – slúži na automatické načítanie knižníc, datových modelov alebo pomocných funkcií,
- `config.php` – obsahuje rôzne základne nastavenia aplikácie, napr. nastavenie predvolenej URI cesty,
- `database.php` – obsahuje nastavenia prístupu k databáze,
- `routes.php` – slúži na prepisovanie URI odkazov pri požiadavkách HTTP protokolu

6.2.2 Prepisovanie odkazov URI

Ako je uvedené vyššie, na prepisovanie URI odkazov slúži konfiguračný súbor `routes.php`. Keďže v tomto projekte serverové rozhranie ponúka veľké množstvo operácií nad dátami, táto funkcia bola použitá pre zjednodušenie URI odkazov, ktorými klienti žiadajú server o služby, a to prostredníctvom požiadaviek HTTP protokolu.

Príklad prekladu odkazov je možné vidieť na ukážke 6.1. Do prvého indexu dvojrozmerného asociatívneho poľa `$route` sa zadáva odkaz, ktorým by sa klient mal dožadovať konkrétnej služby servera. Voliteľné segmenty odkazu sa zapisujú pomocou zátvoriek, v tomto prípade je použitý zápis `(:num)`, ktorý signalizuje číselnú hodnotu. Do druhého indexu toho poľa sa zadáva HTTP dotazovacia metóda.

Následne je tomuto zápisu priradená iná cesta. V tomto prípade ide o cestu v rámci frameworku CodeIgniter. Prvý segment tejto cesty označuje názov súboru, resp. triedy ovládača (controller) a druhý segment označuje názov metódy v tejto triede, ktorá sa má povolať. Ďalšie segmenty už určujú parametre funkcie, ktorá sa má povolať a to v usporiadanom poradí takom, v akom sú aj pri deklarácii tejto funkcie. Tieto segmenty sa odkazujú na voliteľné segmenty prekladaného URI odkazu. Deklaráciu tejto metódy umiestnenej v triede ovládača je možné vidieť na ukážke 6.2.

```
$route['year/yrs/(:num)/(:num)']['get'] = Year/all_years_of_this_yearnumber/$1/$2';
```

Výpis 6.1: Príklad zdrojového kódu prepisu URI odkazu

6.2.3 Ovládač (controller)

Ovládač slúži ako sprostredkovateľ medzi modelom, pohľadom (view) a inými zdrojmi, ktoré sú potrebné pre spracovanie HTTP požiadavky a pre vygenerovanie webstránky, resp. výsledku. [8]

Ovládač sa skladá len z funkcií, ktoré sa starajú o spracovanie požiadavky klienta. Pri deklarácii triedy ovládača je potrebné dediť z triedy `CI_Controller`, ktorá je súčasťou frameworku a vďaka ktorej bude vytvorenej triede ovládača priradená každá knižnica frameworku.

Keďže ide o architektúru REST serverového rozhrania a úlohou servera je sprístupniť klientom operácie nad údajmi v databáze, tak funkcie všetkých ovládačov majú podobnú implementáciu. Takúto implementáciu je možné vidieť na ukážke 6.2.

V prvej časti, resp. podmienke implementácie sa pomocou PHP super-globálnej premennej `$_SERVER` skontroluje správnosť dotazovacej HTTP metódy klientskej požiadavky, a tiež správnosť formátu parametrov funkcie. V ďalšej časti zdrojového kódu sa pomocou triedy dátového modelu `AuthMod` postupne skontrolujú hlavičky HTTP požiadavky klienta, autorizuje používateľ a preveria sa oprávnenia používateľa pre žiadanú operáciu na základe jeho role v informačnom systéme.

V prípade nesplnenia niektorej z podmienok v týchto častiach sa prostredníctvom pomocnej funkcie `json_output()` odošle klientovi HTTP odpoveď s príslušným chybovým kódom a podrobnejším popisom chyby obsiahnutom v tele tejto HTTP odpovede. V opačnom prípade sa v poslednej časti implementácie funkcie zavolá funkcia triedy prislúchajúceho dátového modelu. Po vrátení výsledku tejto funkcie sa odošle klientovi HTTP odpoveď a to opäť prostredníctvom pomocnej funkcie `json_output()` s kódom signalizujúcim úspech operácie.

```
public function all_years_of_this_yearnumber($year, $relID)
{
    $method = $_SERVER['REQUEST_METHOD'];
    if($method != 'GET' || $this->uri->segment(3) == ''
                || is_numeric($this->uri->segment(3)) == FALSE
                || $this->uri->segment(4) == ''
                || is_numeric($this->uri->segment(4)) == FALSE) {
            json_output(400,array('status' => 400,'message' => 'Bad request.'));
            return;
    }

    $check_auth_client = $this->AuthMod->check_auth_client();
    if($check_auth_client != true) {
            json_output($check_auth_client['status'], $check_auth_client);
            return;
    }
    $response = $this->AuthMod->auth();
    if($response['status'] != 200){
            json_output($response['status'], $response);
            return;
    }
    $usertype = $this->AuthMod->get_user_type();
    if(!($usertype=="riadite" || $usertype=="tovnk"))  {
            json_output(403,array('status' => 403,'message' => 'Forbidden'));
            return;
    }

    $resp = $this->YearMod->get_all_years_of_this_yearnumber($year, $relID);
    json_output(200,$resp);
}
```

Výpis 6.2: Príklad zdrojového kódu funkcie ovládača

6.2.4 Dátový model (model)

Model reprezentuje dátové štruktúry. Zvyčajne obsahuje funkcie, ktoré napomáhajú získať, vložiť, odstrániť alebo aktualizovať údaje v databáze. [8]

Funkcie, ktoré sú v modeli zahrnuté majú rôznu implementáciu. V ďalšom texte je však priblížené, ako sa dajú pomocou frameworku vykonávať operácie nad dátami v databáze pomocou všeobecného príkazu. Zároveň tu je uvedené, akým spôsobom je možné získať výsledok operácie vo forme takej, aby sa s ním dalo pracovať.

Pre prístup k databáze je nutné tento prístup nastaviť, a to pomocou konfiguračného súboru `database.php`, ktorý bol spomenutý vyššie. Pri deklarácií triedy dátového modelu vo frameworku je nutné uviesť dedenie z triedy frameworku `CI_Model`. Následne je veľmi nápomocné pridať názov tejto triedy modelu do konfiguračného súboru `autoload.php`, presnejšie ako parameter funkcie `array()`, ktorá je priradená k položke `$autoload['model']`. Tým bude tento model automatický načítaný počas prevádzky aplikácie a nebude ho nutné explicitne načítavať v ktorejkoľvek triede ovládača.

Získanie dát z databázy

Dáta sa z databázového systému MySQL získavajú prostredníctvom príkazu `SELECT`. CodeIgniter síce ponúka pre túto operáciu vlastné príkazy, no keďže sú tieto operácie v aplikácii často veľmi zložité, používaný bol všeobecný príkaz frameworku, ktorého parametrom je

reťazec tvorený MySQL dotazom. Ide o príkaz `$this->db->query()`, ktorý je pre získanie výsledku nutné priradiť ľubovoľnej premennej.

Z tejto premennej je následne možné získať výsledok. V tomto projekte bol na tento účel využívaný prevažne príkaz `result_array()`. Tento príkaz formátuje výsledok dotazu do dvojrozmerného poľa.

V prípade, že s výsledkom bolo potreba ešte pracovať, s týmto poľom sa dalo k údajom jednoducho pristupovať, a to pomocou indexov, z ktorých prvý definuje poradie riadku výsledku dotazu a ten druhý zasa atribút jednotlivých dotazovaných záznamov. V prípade, že výsledok dotazu bol aj výsledkom operácie, tak funkcia triedy modelu bola ukončená a dvojrozmerné pole obsahujúce výsledok dotazu vrátené triede ovládača, ktorý metódu povolal, prostredníctvom príkazu `return`.

Ostatné operácie s údajmi v databáze

Ostatné operácie využívané pre prácu s databázou sú vkladanie, aktualizovanie a odstraňovanie jednotlivých záznamov. V jazyku MySQL pre vykonanie týchto operácií slúžia príkazy `INSERT, UPDATE` a `DELETE`. Keďže tieto operácie neboli v projekte tak zložité ako dotazovanie nebol využívaný všeobecný príkaz `$this->db->query()`, ale konkrétne príkazy frameworku určené pre tieto operácie.

Pri funkciách `delete()` a `update()` je použitá tiež funkcia `where()`, ktorá nahrádza MySQL kľúčové slovo `WHERE`, a teda prostredníctvom atribútu `'id'`, ktorý je primárnym kľúčom, jednoznačne určuje záznam, ktorý má byť vymazaný, resp. aktualizovaný.

Asociatívne pole `$data` je nutné vytvoriť pre definovanie hodnôt atribútov konkrétneho záznamu, ktorý má byť vložený alebo aktualizovaný. Pri inicializácii tohto poľa reťazec v úvodzovkách označuje atribút, resp. vlastnosť záznamu a priradená premenná zasa hodnotu tohto atribútu. Pole je využité pri volaní funkcií `insert()` a `update()`.

Pri vkladaní alebo aktualizácii údajov v tabuľkách databázy klient v požiadavke uvádza aj parametre HTTP protokolu. Tieto parametre je potrebné spracovať, aby ich bolo možné použiť. Slúži na to super-globálna PHP premenná `$_REQUEST`. Tá je v ukážke 6.3 priradená premennej `params`, z ktorej sa týmto stáva asociatívne pole. Pomocou tohto poľa je možné pristupovať k jednotlivým parametrom HTTP požiadavky, a to tak že sa do jeho indexu zadá reťazec, predstavujúci názov parametru.

```
$this->db->where('id', $id)->delete('odpracovane_hodiny');
$params = $_REQUEST;
$hrs_id = $params['id'];
$data = array('datum' => $hrs_date, 'od' => $hrs_from,'do' => $hrs_to,
    'z_toho_nadcas' => $hrs_overtime);
$this->db->insert('odpracovane_hodiny', $data);
$this->db->where('id', $hrs_id)->update('odpracovane_hodiny', $data);
```

Výpis 6.3: Implementácia funkcie json_output()

6.2.5 Pohľad (view)

Pohľad je informácia, ktorá sa zobrazuje používateľovi. Pohľadom vo frameworku CodeIgniter je väčšinou webová stránka alebo iný typ stránky. [8]

Keďže je toto serverové rozhranie postavené na REST architektúre, jeho výstupom, resp. odpoveďou používateľovi nie je žiadny typ stránky, ale dáta serializované v určitom formáte. Preto v implementácii tohto serverového rozhrania nie je vytvorený ani použitý žiadny pohľad.

6.2.6 Pomocná funkcia `json_output()`

Ako bolo spomenuté v podkapitole 6.2.3 pre odpoveď používateľovi sa využíva pomocná funkcia `json_output()`. Implementáciu tejto funkcie je možné vidieť v ukážke 6.4.

Táto pomocná funkcia sa nachádza v súbore `json_out_helper.php`, ktorý sa nachádza v pod-adresári `helpers`. Funkcia najprv nastaví HTTP hlavičku `Content-Type` odpovede ako serializačný formát `JSON`. Následne nastaví HTTP návratový kód, signalizujúci úspešnosť spracovania používateľovej požiadavky a nakoniec transformuje výsledok spracovania požiadavky do formátu `JSON`, ktorý následne pridá do tela odpovede.

```
function json_output($statusHeader, $response)
{
    $ci =& get_instance();
    $ci->output->set_content_type('application/json');
    $ci->output->set_status_header($statusHeader);
    $ci->output->set_output(json_encode($response));
}
```

Výpis 6.4: Implementácia funkcie json_output()

6.3 Implementácia klientskej desktopovej aplikácie

Jednou z klientských aplikácií je desktopová aplikácia určená pre používateľov s rolami riaditeľ, účtovník a admin. Tak ako bolo spomenuté v podkapitole 6.1.2 pre vývoj tejto aplikácie bol zvolený programovací jazyk Java spolu s technológiou JavaFX pre tvorbu používateľského rozhrania. V tejto sekcii sú priblížené základné prvky implementácie tejto aplikácie informačného systému.

6.3.1 Adresárová štruktúra

V koreňovom adresári tejto aplikácie sa nachádza viacero dôležitých adresárov pre implementáciu. Patria medzi nich:

- `application` – obsahuje súbory zdrojových kódov programovacieho jazyka Java, ktoré tvoria celú aplikáciu,
- `css` – obsahuje súbory zdrojových kódov jazyka CSS, určených pre štylizáciu niektorých prvkov používateľského rozhrania,
- `fonts` – obsahuje použité fonty v aplikácii,
- `png` – obsahuje obrázky alebo ikony použité v aplikácii.

Okrem týchto zložiek obsahuje koreňový adresár aj dôležitý súbor `module-info.java` určený pre deklaráciu modulov projektu. Medzi ne patria externe knižnice programovacieho jazyka Java, ale aj package-priečinky projektu.

Adresár `application`

Ako je spomenuté vyššie v tomto adresári sa nachádzajú zdrojové súbory implementácie aplikácie programovacieho jazyka Java. Tento adresár obsahuje zdrojový súbor `Main.java`, ktorý je hlavným zdrojovým súborom implementácie a stará sa o spustenie aplikácie. Okrem toho adresár obsahuje následujúce pod-adresáre:

- `alerts` – obsahuje zdrojové súbory implementujúce grafické chybové hlásenia prostredníctvom technológie JavaFX,
- `core` – obsahuje zdrojové súbory implementujúce logiku výpočtu výplat, resp. miezd zamestnancov,
- `exceptions` – obsahuje zdrojové súbory pre vlastné výnimky jazyka Java,
- `httpcomunication` – obsahuje zdrojové súbory s príponou `.fxml` tvoriace používateľské rozhranie a zdrojové súbory ovládačov (controllers) tohto používateľského rozhrania implementujúce základnú logiku používateľského rozhrania prostredníctvom technológie JavaFX.
- `models` – obsahuje zdrojové súbory objektov reprezentujúcich tabuľky databázy informačného systému,
- `pdf` – zdrojové súbory pre generovanie výplat do PDF súborov.

6.3.2 Komunikácia HTTP

Klient musí pre komunikáciu so serverovým rozhraním komunikovať prostredníctvom protokolu HTTP, preto bolo potrebné túto komunikáciu zabezpečiť. Na tento účel boli zvolené interné knižnice `HttpClient, HttpRequest` a `HttpResponse`.

Keďže komunikáciu so serverom treba vykonávať v aplikácií pomerne často, bola implementovaná trieda `HttpClientClass`, reprezentujúca objekt, ktorý komunikáciu vykoná a značne zjednoduší. Táto trieda je implementovaná v súbore `HttpClientClass.java`, ktorý sa nachádza v pod-adresári `httpcomunication`.

Klient HTTP sa v aplikácií vytvorí pri vytvorení inštancie, resp. objektu tejto triedy. Ďalej sa už zašle len HTTP požiadavka, vďaka ktorej komunikácia so serverom prebehne, a to prostredníctvom niektorej z nasledujúcich členských metód:

- `sendGet(String uriSufix, String authorization, String userID)` – odošle požiadavku s HTTP metódou GET, a to pomocou prípony predvoleného URI odkazu (`uriSufix`), prideleného používateľského autorizačného tokenu (`authorization`) a jednoznačného identifikátora používateľa (`userID`),
- `sendDelete(String uriSufix, String authorization, String userID)` – odošle požiadavku s HTTP metódou DELETE,
- `sendPost(String uriSufix, String authorization, String userID)` – odošle požiadavku s HTTP metódou POST,
- `sendLoginPost(String email, String password)` – odošle požiadavku s HTTP metódou POST určenú pre prihlásenie používateľa, a to pomocou e-mailu používateľa (`email`) a jeho šifrovaného hesla (`password`).

Pri posielaní požiadavky POST sú často odoslané aj parametre tejto metódy. Pre definovanie týchto parametrov sa pred samotným odoslaním požiadavky používa metóda `addParam(String name, String value)`, ktorý definuje jeden parameter požiadavky na základe jeho návrhu (`name`) a jeho hodnoty (`value`).

Po zaslaní požiadavky je v implementácii možné dostať odpoveď serveru a to prostredníctvom metódy `getResponseStatus()`, ktorá vráti návratový HTTP kód odpovede zo

servera a `getResponseBody()`, ktorá vráti výsledné dáta zo servera zapísaných vo formáte JSON.

6.3.3 Spracovanie dát vo formáte JSON

Rozhranie servera zasiela výsledok požiadavky v tele HTTP odpovedi v serializačnom formáte JSON. Tieto údaje bolo potrebné v klientskej aplikácii spracovať, aby ich bolo možné použiť. Pre toto spracovanie sa využila voľne dostupná externá knižnica `json-simple` vo verzii 1.1.

V pod-adresári `httpcomunication` sa nachádzajú zdrojové súbory reprezentujúce triedy objektov, ktoré túto knižnicu pre spracovanie využívajú. Ide o súbory `JsonArrayClass.java` a `JsonObjectClass.java`. Trieda `JsonObjectClass` reprezentuje objekt jedného JSON objektu, zatiaľ čo trieda `JsonArrayClass` reprezentuje objekt poľa JSON objektov.

Viac používanou triedou bola tá reprezentujúca pole JSON objektov. Táto trieda zahŕňa dôležité funkcie:

- `parseJson(String j)`, ktorá spracuje reťazec `j` zapísaný vo formáte JSON do použiteľného poľa JSON objektov, a ktorá je volaná v konštruktéri triedy,
- `getSize()`, ktorá vráti počet prvkov, resp. JSON objektov poľa,
- `getElement(int index, String elementName)`, ktorá na základe číselného indexu (`index`) označujúceho poradie JSON objektu v poli a názvu atribútu (`elementName`) vráti žiadaný údaj.

6.3.4 Prihlásený používateľ

Po prihlásení používateľa do aplikácie je nutné uchovávať informácie o tomto používateľovi. K tomuto slúži trieda so statickými atribútmi, ku ktorým je možné kedykoľvek pristupovať. Ide o triedu `LoggedInUser`, ktorej zdrojový súbor sa nachádza v pod-adresári `httpcomunication`. Údaje o prihlásenom používateľovi slúžia pre autorizáciu a autentifikáciu používateľa pri zasielaní klientskych požiadaviek serveru. Medzi tieto údaje patria atribúty:

- `id` – jednoznačný identifikátor používateľa v rámci celého informačného systému,
- `token` – používateľov jedinečný reťazec náhodných znakov pridelený serverom pri prihlásení, slúžiaci k autorizácii,
- `email` – prihlasovací e-mail používateľa,
- `role` – rola používateľa v rámci celého informačného systému.

6.4 Implementácia klientskej mobilnej aplikácie

Ďalšou z klientských aplikácií je mobilná aplikácia vyvíjaná na mobilnej platforme Android tiež s využitím programovacieho jazyka Java. V ďalšom texte sú priblížené základné prvky implementácie tejto aplikácie.

6.4.1 Adresárová štruktúra

V koreňovom adresári zdrojových súborov tejto aplikácie sa nachádza adresár `main`, ktorého najdôležitejšími pod-adresármi pre implementáciu tejto aplikácie sú `res` a `java`. Okrem toho sa v tomto adresári nachádza aj dôležitý súbor `AndroidManifest.xml`, ktorý popisuje základné informácie o aplikácii.

Pod-adresár `java` obsahuje zdrojové súbory implementované v jazyku Java, ktoré tvoria logiku aplikácie. Tieto zdrojové súbory sú rozdelené v priečinkoch:

- `data` – obsahuje zdrojové súbory objektov reprezentujúcich tabuľky databázy informačného systému a zdrojový súbor triedy prihláseného používateľa,
- `httpcomunication` – obsahuje zdrojové súbory s implementáciou HTTP komunikácie,
- `mainactivity` – obsahuje logickú časť používateľského rozhrania hlavnej časti aplikácie, do ktorej patria všetky úkony okrem prihlásenia používateľa,
- `login` – obsahuje logickú časť používateľského rozhrania pre prihlasovací formulár.

V pod-adresári `res` obsahuje súbory `.xml`, ktoré prevažne tvoria používateľské rozhranie aplikácie. Najdôležitejšími pod-adresármi pre implementáciu v adresári `res` sú:

- `layout` – obsahuje zdrojové súbory tvoriace používateľské rozhranie,
- `menu` – obsahuje zdrojové súbory implementujúce položky dolného navigačného panelu,
- `navigation` – obsahuje zdrojové súbory implementujúce fragmenty, zobrazené po zvolení položky v dolnom navigačnom paneli.

6.4.2 Komunikácia HTTP

Implementácia HTTP komunikácie so serverom je veľmi podobná tej z podkapitoly 6.3.2. Jediný rozdiel je v použitej knižnici. V tejto aplikácii sa totiž využíva knižnica `HttpURLConnection`.

Názov triedy, ktorá túto implementáciu obsahuje sa nazýva `HttpClient` a jej zdrojový súbor sa nachádza v adresári `httpcomunication`.

6.4.3 Spracovanie dát vo formáte JSON

Aj v tomto prípade ide o takmer totožnú implementáciu ako v prípade desktopovej aplikácie. Opäť spočíva rozdiel len v použitých knižniciach, ktorými pre túto aplikáciu sú `JSONArray` a `JSONObject`.

Pre spracovanie JSON dát sa používa trieda `JsonArrayClass`, ktorá sa nachádza v adresári `httpcomunication`.

6.4.4 Prihlásený používateľ

Pri implementácii prihláseného používateľa ide o implementáciu, ktorá je dokonca úplne zhodná s implementáciou v desktopovej aplikácii spomenutej v podkapitole 6.3.4.

Zdrojový súbor sa rovnako nazýva `LoggedInUser` a nachádza sa v adresári `data`.

6.5 Implementácia základných častí informačného systému

V tejto sekcií sú priblížené implementácie tých najdôležitejších základných častí informačného systému, ktoré vychádzajú aj z analýzy požiadaviek v kapitole 4.

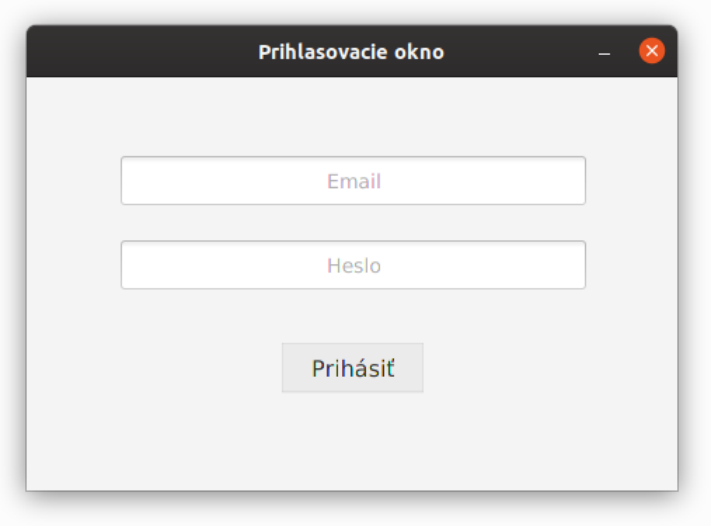
6.5.1 Prihlásenie používateľa

Pre prístup do informačného systému sa musí každý používateľ prihlásiť. Používateľ sa prihlási prostredníctvom prihlasovacieho formulára, ktorý je v obidvoch klientských aplikáciach prístupný hneď po spustení aplikácie. V prihlasovacom formulári sa zadáva prihlasovací e-mail a heslo.

Prostredníctvom členskej metódy `sendLoginPost(String email, String password)` triedy `HttpClientClass` sa po potvrdení prihlasovacích údajov používateľom, odošle HTTP požiadavka pre prihlásenie na server.

Na strane servera prihlasovaciu požiadavku obslúži povolaná funkcia `login()` ovládača (controller) `Auth`, ktorá prostredníctvom prislúchajúceho modelu `AuthMod` a jeho funkcie `login($username, $password)` najprv overí správnosť prihlasovacích údajov a následne právo pre prihlásenie do danej klientskej aplikácie. V prípade nesplnenia kontrolných podmienok zašle server klientskej aplikácii HTTP odpoveď s chybovým návratovým kódom a stručným popisom o neúspechu prihlásenia. V opačnom prípade server vygeneruje reťazec náhodných znakov (token) pre identifikáciu prihlásenia, ktorý spoločne s identifikátorom prihlasovacieho konta používateľa uloženom v databáze zašle klientovi. Následne zaznamená prihlásenie, resp. autorizáciu používateľa do databázy, čím je tento používateľ prihlásený v informačnom systéme.

Klientská aplikácia v prípade neúspešného prihlásenia vypíše popis neúspechu od servera. Naopak pri úspešnom prihlásení uloží údaje spojené s týmto prihlásením do statickej triedy `LoggedInUser` a sprístupní používateľovi hlavnú časť aplikácie.

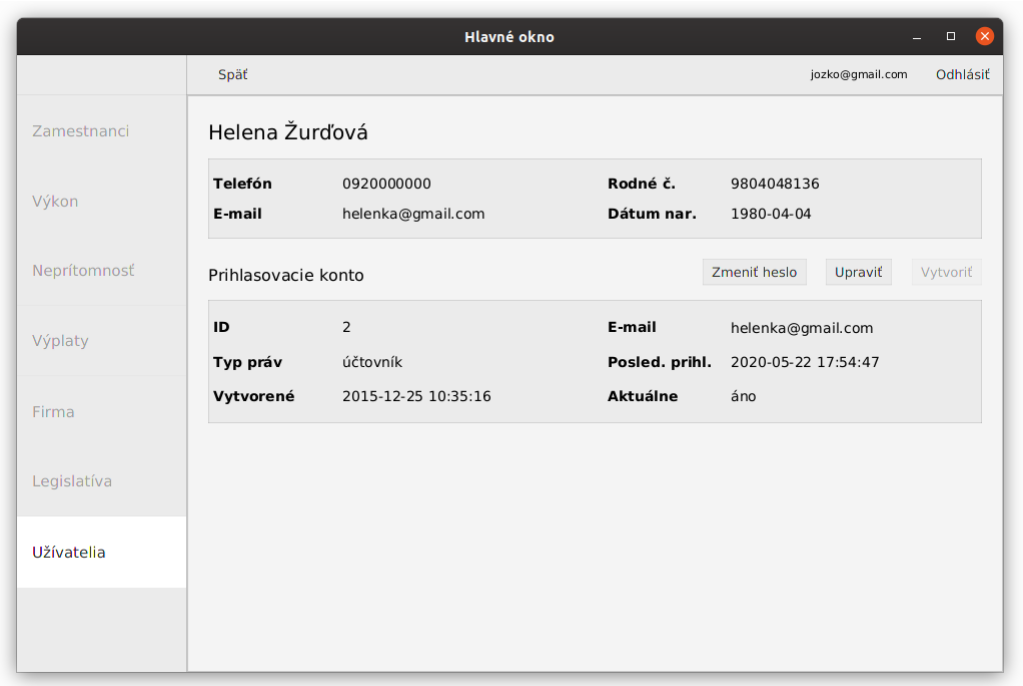


Obr. 6.2: Prihlasovací formulár desktopovej klientskej aplikácie (zdroj: autor)

6.5.2 Správa používateľov

Spravovať prihlasovacie kontá používateľov môže len používateľ s rolou admin, a to len v klientskej desktopovej aplikácii.

Po prihlásení používateľa role admin sa zobrazí hlavný obsah s tabuľkovým zoznamom všetkých zamestnancov s ich prihlasovacími kontami. Po výbere jedného z nich sa v aplikácii zobrazí detail daného zamestnanca (viď obr. 6.5). O zobrazenie prehľadu zamestnancov sa stará zdrojový súbor pohľadu (view) a trieda ovládača (controller) s názvom `PageUsers`, zatiaľ čo o zobrazenie detailu jedného zamestnanca pohľad a ovládač `PageUsersDetail`.



Obr. 6.3: Náhľad používateľského rozhrania pre detail používateľa (zdroj: autor)

Po ich spustení sa zobrazí nové dialógové okno s formulárom pre vykonanie konkrétnej akcie. Formuláre pre tieto akcie sú implementované v zdrojových súboroch s názvami `AddAccount`, `UpdateAccount` a `UpdateAccountPassoword`. V nich je zahrnuté aj odoslanie požiadaviek pre tieto úkony.

V prípade neexistujúceho používateľského konta tohto zamestnanca môže admin toto konto zamestnancovi vytvoriť. V prípade, že zamestnanec vlastní používateľské konto, admin má možnosť zmeniť údaje prihlasovacieho konta, ako je aktívnosť a rola tohto konta, a tiež zmeniť heslo používateľa. O spustenie týchto akcií sa starajú členské metódy `CreateClick()`, `UpdateClick()` a `ChangeClick()`, ktoré sa vykonajú kliknutím na prislúchajúce tlačidlo.

Na strane servera sa o vybavenie požiadaviek súvisiacich s používateľmi stará ovládač `Employee` s funkciami `crt_usr()`, `upd_usr()` a `upd_usr_pas()`.

6.5.3 Pridanie pracovnoprávneho vzťahu zamestnanca

V časti desktopovej aplikácie – `zamestnanci` je možné v detaile jedného pracujúceho pridať pracovnoprávny vzťah tomuto zamestnancovi. Tento úkon môže vykonať iba používateľ s rolou riaditeľa alebo účtovníka.

Po kliknutí na tlačidlo pridania nového pracovnoprávneho vzťahu, aplikácia zobrazí dialógové okno s formulárom pre vytvorenie takéhoto vzťahu. O implementáciu formulára sa starajú ovládač a pohľad s názvom `AddRelation`.

Okrem vyplnenia bežných podmienok pracovnoprávneho vzťahu, musí používateľ zadať aj všetky mzdy, ktorými bude zamestnanec platený a to v dolnej časti formulára.

Po vyplnení a potvrdení formulára používateľom sa povolá členská metóda `create()`, ktorá odchytí kliknutie na tlačidlo potvrdenia formulára. V jej implementácii sa na začiatku povolajú funkcie `checkFormular()` a `checkFormularTypes()`. Prvá spomenutá skontroluje, či boli vyplnené všetky vstupy formulára, ktoré vyplnené mali byť, zatiaľ čo druhá spomenutá skontroluje či vyplnené vstupné dáta majú správny formát.

Následne sa povolá metóda `setModelsFromInputs()`, ktorá zo zadaných vstupných údajov vytvorí dátové modely, resp. objekty, ktoré zodpovedajú tabuľkám databázy informačného systému. Ide o tabuľky, ktoré budú v prípade úspešného pridania pracovnoprávneho vzťahu naplnené novými údajmi. Objekty sa teda vytvoria ako inštancie tried `RelationD`, `ConditionsD`, `NextConditionsD` a `WageD`.

Na koniec sa v metóde `create()` povolá metóda `createRelation()`, ktorá zaobstará vytvorenie HTTP požiadaviek metódy POST s parametrami, ktoré sú tvorené prostredníctvom už naplnených dátových modelov. V prípade úspechu pridania pracovnoprávneho vzťahu sa okno formulára zatvorí, v opačnom prípade sa zobrazí používateľovi len chybové hlásenie zo servera.

Na serveri sa o spracovanie požiadavky o vytvorenie pracovnoprávneho vzťahu, ale aj o spracovanie iných požiadaviek spojených s týmito vzťahmi stará ovládač `Employee.php`.

V tejto aplikácii majú všetky ostatné formuláre zhodnú štruktúru implementácie.

6.5.4 Evidencia odpracovaného výkonu

Evidovať odpracovaný výkon zamestnanca môže každý zamestnanec v rámci svojho výkonu prostredníctvom mobilnej aplikácie a používateľ s rolou riaditeľ v rámci výkonu všetkých zamestnancov.

Keďže v obidvoch prípadoch je implementácia obdobná, v tejto podkapitole je pre zmenu popísaná implementácia evidencie výkonu v mobilnej aplikácii.

Formulár pre evidenciu výkonu v Android mobilnej klientskej aplikácii je tvorený súborom `activity_wage_hours_add.xml`, ktorý implementuje používateľské rozhranie formulára a súborom `WageHoursAddFragment.java`, ktorý predstavuje logiku tohto formulára.

V triede `WageHoursAddFragment` sa nachádzajú metódy, ktoré nastavujú formulár pri jeho zobrazení. Ide o metódy `setCalendar()`, `setEnability()` a `setButton()`. Prvá spomenutá metóda implementuje interaktívny kalendár pre voľbu dňa odpracovaného výkonu. Metóda `setEnability()` sa zasa stará o to, aby formulár zobrazil len potrebné vstupné prvky formulára pre zadanie výkonu, keďže nie každý zamestnanec je platený tou istou formou mzdy. Posledná spomenutá metóda obsahuje implementáciu toho, čo sa má stať po kliknutí na tlačidlo potvrdzujúce formulár.

Táto metóda je implementovaná podobne ako metóda `create()` spomenutá v podkapitole 6.5.3. Aj tu sú volané metódy `checkFormular()`, `checkFormularTypes()` a `setModelsFromInputs()`, ktoré majú tú istú úlohu. Nakoniec je pomocou riadku kódu `new`

`MyUploadTask().execute();` zaslaná HTTP požiadavka na server. Tento kód vykoná implementáciu zahrnutú vo vnorenej triede `MyUploadTask`, ktorá dedí z triedy `AsyncTask`. Tato rodičovská trieda zabezpečí to, aby sa implementácia triedy potomka vykonala na druhom vlákne a nie na hlavnom vlákne aplikácie.

Na strane servera sa o spracovanie požiadaviek spojených s odpracovaným výkonom zamestnancov stará ovládač implementovaný v zdrojovom súbore `Hours.php`.

Ukážka formulára mobilnej aplikácie pre evidovanie odpracovaného výkonu bola uvedená na obrázku 5.5.

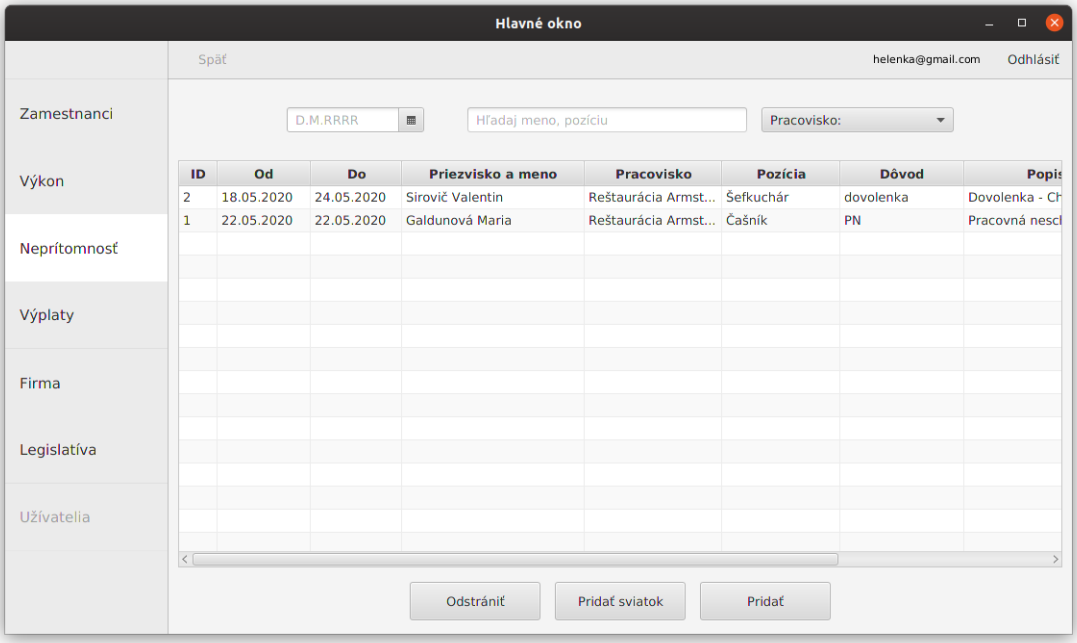
6.5.5 Evidencia neprítomnosti zamestnanca

Evidovať neprítomnosť zamestnanca je možné len prostredníctvom desktopovej aplikácie. Práva k tomuto úkonu majú len používatelia s rolami zamestnávateľ a účtovník.

Ak používateľ klikne v navigačnom paneli na tlačidlo `neprítomnosti`, zobrazí sa mu prehľad neprítomnosti za posledné tri dni v tabuľke. Ak zadá konkrétny dátum, zobrazia sa mu všetky neprítomnosti za vybraný deň. O tento prehľad sa stará ovládač a pohľad s názvom `PageAbsence`.

Pod týmto prehľadom je možné s neprítomnosťami aj pracovať. Pomocou tlačidiel je možné odstrániť existujúcu neprítomnosť, vytvoriť novú neprítomnosť alebo vytvoriť neprítomnosť v prípade, ak zamestnanec nepracuje v daný deň kvôli sviatku. Tieto úkony sú spustené po kliknutí na prislúchajúce tlačidlá, implementované pomocou metód `onRemoveClick()`, `onAddClick()` a `onFeastClick()`. V prípade odstránenia neprítomnosti sa skontroluje či používateľ vybral z tabuľky položku, ktorú chce odstrániť, a ak áno odošle sa požiadavka na server prostredníctvom metódy `sendDelete()`. V ostatných dvoch prípadoch sa zobrazia dialógové okná s formulármi, ktoré sú implementované prostredníctvom zdrojových súborov s názvami `AddAbsende` a `AddFeast`.

Na strane rozhrania servera sa o spracovanie požiadaviek klienta postarajú metódy `crt_bsnc()` a `crt_fst()` ovládača `Absence.php`.



Obr. 6.4: Náhľad používateľského rozhrania prehľadu absencií (zdroj: autor)

6.5.6 Výpočet výplaty zamestnanca

Výpočet výplat môže v desktopovej aplikácii vykonávať používateľ s rolou riaditeľ alebo účtovník. Implementácia výpočtu mzdy je však veľmi obsiahla a preto sú v tejto podkapitole uvedené len základné komponenty implementácie.

Implementácia používateľského rozhrania pre tento účel, ktorým je dialógové okno, je obsiahnutá v adresári s cestou `application/gui/payment/create`. Tieto zdrojové súbory využívajú logiku pre jednotlive prepočty pri tejto činnosti. Táto logika je implementovaná v zdrojových súboroch, ktoré sa nachádzajú na ceste `application/core`.

Hlavnou triedou adresára `core` je `PaymentManager`, ktorý obsahuje objekty tried `GrossWageManager`, slúžiaca pre výpočet hrubej mzdy, a `NetWageManager`, slúžiaca pre výpočet čistej mzdy výplaty.

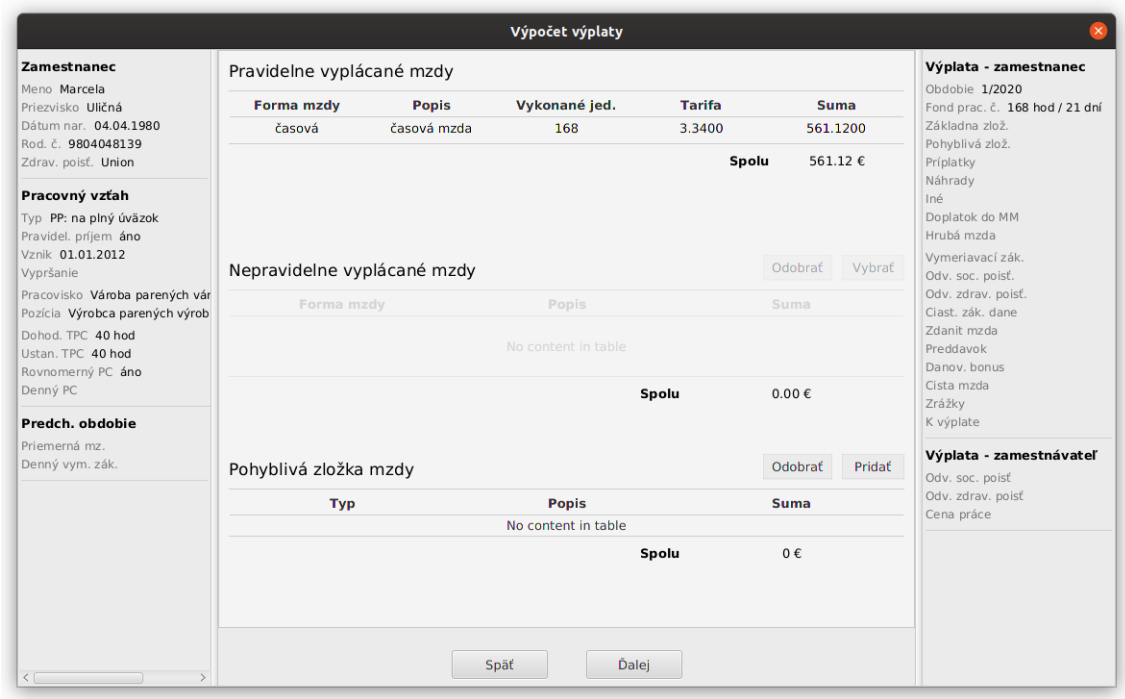
Objekt triedy `GrossWageManager` obsahuje objekty tried:

- `WorkedPerformanceManager` – slúži pre výpočet a správu odpracovaného výkonu zamestnanca,
- `BasicComponentManager` – slúži pre výpočet a správu základných zložiek mzdy,
- `DynamicComponentManager` – slúži pre výpočet a správu pohyblivých zložiek mzdy,
- `SurchargesManager` – slúži pre výpočet a správu zložiek príplatkov mzdy,
- `WageCompensationManager` – slúži pre výpočet a správu náhrad miezd,
- `OthersComponentsManager` – slúži pre správu iných zložiek mzdy,
- `MinimalWageDeficitManager` – slúži pre výpočet a správu doplatku do minimálnej mzdy.

Objekt triedy `NetWageManager` zasa obsahuje objekty tried:

- `LeviesSocialInsuranceManager` – slúži pre výpočet a správu odvodov do sociálnej poisťovne,
- `LeviesHealthInsuranceManager` – slúži pre výpočet a správu odvodov do zdravotnej poisťovne,
- `DeductionsManager` – slúži pre správu ostatných zrážok zo mzdy.

Ostatnými členskými premennými sú premenné zobrazujúce peňažné hodnoty, ktoré sú dôležite pre výpočet čistej mzdy. Pre ukladanie a prácu všetkých peňažných hodnôt pri výpočte mzdy zamestnancov bol využitý dátový typ `BigDecimal`.



Obr. 6.5: Náhľad používateľského rozhrania výpočtu mzdy zamestnanca (zdroj: autor)

Kapitola 7

Testovanie

Finálnou časťou pri vývoji informačného systému je jeho testovanie, na základe ktorého je vývojár schopný odhaliť chyby systému, ktoré by mal následne odladiť. Táto časť testovania informačného systému by mala byť tou poslednou no zároveň najdôležitejšou pred samotným zavedením informačného systému do praxe.

Pri vývoji systému, ktorým sa zaoberá táto práca sa samotné testovanie informačného systému uskutočnilo v dvoch fázach. Tou prvou bolo priebežné testovanie vývojárom, a to počas samotného vývoja, resp. implementácii tohto informačného systému. Druhá fáza testovania systému už prebiehala po samotnej implementácii a prebehla v spolupráci s potenciálnymi používateľmi.

7.1 Testovanie vývojárom

Pri implementácií každej novej funkcionality systému dochádzalo k jej následnému otestovaniu vývojárom. Prostredníctvom toho boli v systéme odhalené rôzne chyby ako sú chyby v komunikácii so serverom, algoritmické chyby či chyby súvisiace s operáciami vykonávane nad databázovým systémom.

Pre testovanie serverovej časti, teda serverového rozhrania a databázového systému, bol použitý softvér s názvom Postman, ktorý je spomenutý v podkapitole 6.1.3. Funkčnosť jednotlivých verzií Testovanie funkčnosti klientských aplikácií zasa prebiehalo manuálne ich spúšťaním prostredníctvom používaných vývojových prostredí.

Toto testovanie serverovej či klientskej časti prebiehalo lokálne. To bolo možné učiniť vďaka softvéru XAMPP, ktorý umožňuje spustenie MySQL serveru a Apache serveru na lokálnych počítačových zariadeniach.

7.2 Testovanie potenciálnymi používateľmi

Informačný systém bol otestovaný aj potenciálnymi používateľmi, a to prostredníctvom súkromného zamestnávateľa, ktorý vlastní viacero prevádzok, resp. pracovísk a zamestnáva desiatky zamestnancov.

Zamestnávateľ bol v prvom rade zoznámený s klientskými aplikáciami informačného systému s doprovodom vývojára. Vývojár sledoval prácu zamestnávateľa a odpovedal na otázky v súvislosti s ovládaním jednotlivých aplikácií. V prípade, že zamestnávateľ mal

neustále nejasnosti ohľadom nejakej činnosti používateľského rozhrania, bolo nutné túto činnosť zjednodušiť.

Kapitola 8

Záver

Cieľom tejto práce bolo vytvoriť informačný systém, ktorý by ľubovoľnému zamestnávateľovi uľahčil výpočet miezd jeho zamestnancov. Zároveň bolo potrebné zabezpečiť, aby mali zamestnanci možnosť evidovať svoj odpracovaný výkon prostredníctvom mobilnej aplikácie. Okrem spomenutých funkcionalít bol dôraz kladený aj na jednoduché vytváranie zamestnancov, vytváranie ich pracovnoprávnych vzťahov, evidenciu neprítomností zamestnancov, spravovanie firemných a legislatívnych údajov a generovanie výplatných pások zamestnancov do súborov PDF. Všetky tieto funkcionality boli úspešne implementované, a teda informačný systém je plne funkčný.

V technickej správe boli najprv popísané princípy tvorby informačných systémov. Rovnako boli v tejto časti aj uvedené technológie, ktoré sa pri tejto tvorbe v praxi využívajú. Ďalšou teoretickou časťou tejto správy bolo popísanie najdôležitejších skutočností legislatívy Slovenskej republiky ohľadom výpočtu miezd, od ktorých tento výpočet závisí.

Nasledujúcimi kapitolami už boli praktické etapy vývoja informačného systému. Tou prvou bola analýza a špecifikácia požiadaviek. V nej boli popísané základné požiadavky na informačný systém, ktoré vzišli z jeho analýzy. Výsledkom tejto analýzy bol diagram prípadov použitia, v ktorom boli uvedené základné funkčné požiadavky systému, a to z pohľadu aktérov – jednotlivých rolí používateľov, ktoré sa v systéme vyskytujú.

Ďalšou praktickou kapitolou bol návrh informačného systému. Jeho najdôležitejšími časťami boli výber architektúry a ER diagram, ktorý zobrazuje ukladanie dát v databáze spolu s vysvetlením významu jeho jednotlivých entitných množín a vzťahov medzi nimi. Následne bol stručne popísaný aj návrh používateľského rozhrania jednotlivých aplikácií informačného systému.

Ďalej bol čitateľ oboznámený aj so samotnou implementáciou. V jej prvej časti boli uvedené použité technológie, v tej ďalšej zasa základná implementácia v rámci vrstiev použitej architektúry. Nakoniec bola popísaná implementácia najzákladnejších častí systému. Po implementácii nasledovalo samotne testovanie systému. Tá bola rozdelená do dvoch fáz a to na priebežné testovanie programátorom a testovanie potenciálnymi používateľmi.

Keďže vývoj informačného systému a jeho jednotlivých aplikácií dopadol úspešne, je možné označiť ciele tejto práce za splnené. Zdrojové súbory implementácie tohto systému je možné nájsť na pamäťovom médiu, ktorého obsah je uvedený v prílohe A.

8.1 Možné pokračovanie projektu

Jedným z možných ale aj potrebných pokračovaní vývoja systému by bol vývoj mobilnej aplikácie zamestnanca aj pre operačný systém iOS. Keďže mobilné operačné systémy Android a iOS spolu momentálne jednoznačne kraľujú trhu mobilných zariadení, bol by zabezpečený prístup k informačnému systému drvivej väčšine zamestnancov ktorejkoľvek firmy. V tomto ohľade je zaujímavou alternatívou aj webová aplikácia pre zamestnanca, ktorá by sa dokázala používateľským rozhraním prispôsobiť mobilným zariadeniam.

Ďalšou možnosťou budúceho vývoja systému je aj pridanie funkcionality pre ohlásenie neprítomnosti zamestnanca zamestnávateľovi. Tým by zamestnanec nemusel priamo kontaktovať zamestnávateľa pre toto ohlásenie. Následne by zamestnávateľ v informačnom systéme túto absenciu potvrdil, čím by sa automaticky do systému zaznamenala.

Literatúra

[1] *API Testing with Postman* [online]. Sparkbox [cit. 2020-05-16]. Dostupné z: https://seesparkbox.com/foundry/api_testing_with_postman.

[2] *Architektura klient-server (Client–server model) - ManagementMania.com* [online]. ManagementMania [cit. 2020-05-11]. Dostupné z: https://managementmania.com/cs/architektura-klient-server.

[3] *ASP.NET - Introduction - Tutorialspoint* [online]. [cit. 2020-05-11]. Dostupné z: https://www.tutorialspoint.com/java/index.htm.

[4] *C++ Overview - Tutorialspoint* [online]. [cit. 2020-05-11]. Dostupné z: https://www.tutorialspoint.com/cplusplus/cpp_overview.htm.

[5] *CSS Tutorial - Tutorialspoint* [online]. [cit. 2020-05-11]. Dostupné z: https://www.tutorialspoint.com/css/index.htm.

[6] *HTML Tutorial - Tutorialspoint* [online]. [cit. 2020-05-11]. Dostupné z: https://www.tutorialspoint.com/html/index.htm.

[7] *Java Tutorial - Tutorialspoint* [online]. [cit. 2020-05-11]. Dostupné z: https://www.tutorialspoint.com/asp.net/asp.net_introduction.htm.

[8] *Model-View-Controller — CodeIgniter 3.1.11 documentation* [online]. [cit. 2020-05-19]. Dostupné z: https://codeigniter.com/userguide3/overview/mvc.html.

[9] *Most Popular Databases In The World* [online]. [cit. 2020-05-11]. Dostupné z: https://www.c-sharpcorner.com/article/what-is-the-most-popular-database-in-the-world/.

[10] *Mzdové podmienky - MPSVR SR* [online]. MPSVR SR [cit. 2020-05-12]. Dostupné z: https://www.employment.gov.sk/sk/praca-zamestnanost/vztah-zamestnanca-zamestnavatela/odmenovanie/mzdove-podmienky.html.

[11] *PHP Tutorial - Tutorialspoint* [online]. [cit. 2020-05-11]. Dostupné z: https://www.tutorialspoint.com/php/index.htm.

[12] *Pracovnoprávne vzťahy - Národný inšpektorát práce* [online]. NIP - Národný inšpektorát práce [cit. 2020-05-12]. Dostupné z: https://www.ip.gov.sk/pracovnopravne-vztahy/.

[13] *Python - Overview - Tutorialspoint* [online]. [cit. 2020-05-11]. Dostupné z: https://www.tutorialspoint.com/python/python_overview.htm.

[14] *Run apps on the Android Emulator | Android Developers* [online]. [cit. 2020-05-16]. Dostupné z: https://developer.android.com/studio/run/emulator.

[15] *Třívrstvá architektura (Three-tier architecture) - ManagementMania.com* [online]. ManagementMania [cit. 2020-05-11]. Dostupné z: https://managementmania.com/cs/trivrstva-architektura-three-tier-architecture.

[16] *Welcome! - The Apache HTTP Server Project* [online]. Documentation Group [cit. 2020-05-16]. Dostupné z: https://httpd.apache.org/.

[17] *Zákon č. 311/2001 Z. z.: Zákonník práce Slovenskej republiky.* 2001. Dostupný na: https://www.epi.sk/zz/2001-311/znenie-20200404/casove-verzie.

[18] *Mobile Operating System Market Share Worldwide* [online]. statcounter, 2020 [cit. 2020-05-16]. Dostupné z: https://gs.statcounter.com/os-market-share/mobile/worldwide.

[19] Bartko, T. *Dohoda o vykonaní práce a dohoda o pracovnej činnosti v roku 2020 – podmienky a vzory | Podnikajte.sk* [online]. Podnikajte.sk, 2018 [cit. 2020-05-12]. Dostupné z: https://www.podnikajte.sk/pracovne-pravo-bozp/dohoda-o-vykonani-prace-pracovnej-cinnosti-2020-vzory.

[20] Bruckner, T. *Tvorba informačních systémů: principy, metodiky, architektury.* Praha: Grada, 2012. ISBN 978-80-247-4153-6.

[21] Burget, R. *Informační systémy - Pojem informačního systému, data, informace* [Fakultná prednáška]. FIT VUT Brno.

[22] Danajovičová, K. *Mzdové centrum - Zrážky zo mzdy – komplexne* [online]. mzdovecentrum.sk, 2019 [cit. 2020-05-13]. Dostupné z: https://www.mzdovecentrum.sk/aktuality/zrazky-zo-mzdy-komplexne-tt-mc-8-2019.htm.

[23] Hill, R., Hirsch, L., Lake, P. a Moshiri, S. *Guide to Cloud Computing.* Springer, 2013. ISBN 978-1-4471-4603-2.

[24] Kocanová, S. *Evidencia pracovného času (dochádzky) zamestnancov | Podnikajte.sk* [online]. Podnikajte.sk, 2019 [cit. 2020-05-13]. Dostupné z: https://www.podnikajte.sk/pracovne-pravo-bozp/evidencia-dochadzky-zamestnancov.

[25] Kubová, S. *Druhy pracovných pomerov a dokumenty súvisiace so vznikom pracovného pomeru | Podnikajte.sk* [online]. Podnikajte.sk, 2017 [cit. 2020-05-12]. Dostupné z: https://www.podnikajte.sk/zamestnanci-a-hr/druhy-pracovnych-pomerov.

[26] Kubová, S. *Zamestnanie dohodára v porovnaní so zamestnancom 2019 a 2020 | Podnikajte.sk* [online]. Podnikajte.sk, 2019 [cit. 2020-05-12]. Dostupné z: https://www.podnikajte.sk/zakonne-povinnosti-podnikatela/dohodar-porovnanie-zamestnanec-2019-2020.

[27] Lacko, L. *Databáze: datové sklady, OLAP a dolování dat s příklady v Microsoft SQL Serveru a Oracle.* Brno: Computer Press, 2003. ISBN 80-7226-969-0.

[28] Lacko Luboslav. *Mistrovství - Android*. Brno: Computer Press, 2017. ISBN 978-80-251-4875-4.

[29] Podnikam.sk. *Zložky mzdy | Podnikam.sk* [online]. Podnikam.sk, 2016 [cit. 2020-05-13]. Dostupné z: https://podnikam.sk/zlozky-mzdy/.

[30] Podnikam.sk. *Formy mzdy | Podnikam.sk* [online]. Podnikam.sk, 2018 [cit. 2020-05-12]. Dostupné z: https://podnikam.sk/formy-mzdy/.

[31] Pohorelá, P. *Stupne náročnosti práce (pracovných miest) | Podnikajte.sk* [online]. Podnikajte.sk, 2018 [cit. 2020-05-12]. Dostupné z: https://www.podnikajte.sk/zakonne-povinnosti-podnikatela/stupne-prace-narocnosti-pracovnych-miest-prace.

[32] Poornima, G. a Girish, R. *Enhancing PHP coding with CodeIgniter*. Educreation Publishing, 2019. ISBN 978-93-5373-055-0.

[33] Šmíd, V. *Management informačního systému: Pojem informačního systému* [online]. [cit. 2020-05-10]. Dostupné z: https://www.fi.muni.cz/~smid/mis-infsys.htm.

Príloha A

Obsah CD

- **/xondri05/instalacia/** – inštalačné pokyny informačného systému
- **/xondri05/spustitelne_subory/desktop_jfx/** – súbory pre spustenie desktopovej aplikácie
- **/xondri05/spustitelne_subory/mobile_android/** – súbory pre spustenie mobilne aplikácie v OS Android
- **/xondri05/TZ/pdf/** – PDF verzia technickej správy
- **/xondri05/TZ/zdrojove_subory/** – zdrojové súbory technickej správy
- **/xondri05/zdrojove_subory/db/** – implementácia databázy IS
- **/xondri05/zdrojove_subory/desktop_jfx/** – implementácia klientskej desktopovej aplikácie
- **/xondri05/zdrojove_subory/mobile_android/** – implementácia klientskej mobilnej aplikácie pre OS Android
- **/xondri05/zdrojove_subory/server_rest_api/** – implementácia REST serverového rozhrania