Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra Informačních Technologií

Diplomová práce

Návrh a vývoj systému pro evidenci pracovní doby

Bc. Jiří Novák

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jiří Novák

Informatika

Název práce

Návrh a vývoj systému pro evidenci pracovní doby

Název anglicky

System design and development for working time records

Cíle práce

Cílem diplomové práce je navrhnout a implementovat webový systém, který zaměstnavatelům usnadní evidenci pracovní doby tak, aby byly naplněny všechny požadavky kladené zákoníkem práce při minimálním časovém vytížení administrátora systému a evidovaných zaměstnanců.

Metodika

Metodika řešené problematiky diplomové práce bude založena na studiu a analýze odborných informačních zdrojů. Praktická část práce bude zaměřena na návrh a implementaci systému evidence pracovní doby s využitím technologií PHP a MySQL. Pro technický návrh systému bude využito poznatků z oboru softwarového inženýrství. Vlastní programování systému bude podléhat paradigmatům OOP a Code First, což umožní snadnou udržovatelnost a rozšiřitelnost systému.

Doporučený rozsah práce

60-80 stran

Klíčová slova

Evidence pracovní doby, PHP, MySQL, OOP, Code First, systém, aplikace, zákoník práce

---

Doporučené zdroje informací

DOMES, M. *Tvorba internetových stránek pomocí HTML, CSS a JavaScriptu.* Kralice: Computer Media, 2005. ISBN 80-86686-39-6.

GILMORE, W J. – POKORNÝ, J. *Velká kniha PHP 5 a MySQL : kompendium znalostí pro začátečníky i profesionály.* Brno: Zoner Press, 2011. ISBN 978-80-7413-163-9.

KOFLER, M. *Mistrovství v MySQL 5 : [kompletní průvodce webového vývojáře].*.

KOSEK, J. *PHP a XML.* Praha: Grada, 2009. ISBN 978-80-247-1116-4.

ULLMAN, L. *PHP a MySQL: názorný průvodce tvorbou dynamických www stránek.* Brno: CP, 2004. ISBN 80-251-0063-4.

Zákoník práce: zákon č. 262/2006 Sb. ze dne 21.dubna 2006. Praha: Ústav práva a právní vědy, 2014. Právo a management. ISBN 978-80-87974-02-5.

Předběžný termín obhajoby

2018/19 LS – PEF

Vedoucí práce

Ing. Petr Benda, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 4. 9. 2018

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 10. 2018

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 10. 03. 2019

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Návrh a vývoj systému pro evidenci pracovní doby" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne: _____

Poděkování

Rád bych touto cestou poděkoval Ing. Petru Bendovi Ph.D. za vedení této práce a připomínky vznesené v průběhu jejího zhotovování. Dále bych chtěl poděkovat Ing. Pavlu Pánkovi a Mgr. Martině Zoubkové za odborné konzultace.

Návrh a vývoj systému pro evidenci pracovní doby

Abstrakt

Předkládaná diplomová práce se snaží adresovat problematiku povinnosti vedení evidence pracovní doby se zvláštním důrazem na pracovní evidenci u pracovníků zaměstnaných v akademickém prostředí.

V počáteční fázi se práce soustředí na kompilaci informací z relevantních informačních zdrojů, analýzu relevantní legislativy a konzultace s odborníky.

Za pomoci syntézy získaných poznatků je vypracováno zadání pro vývoj informačního systému, který je navržen tak, aby významně snížil čas a náklady spojené s vedením evidence pracovní doby v akademickém prostředí. Následně jsou vypracovány architektonické návrhy pro vývoj zadaného systému formou webové aplikace.

Na závěr je vyvinut prototyp systému pro vedení pracovní evidence, odpovídající vypracovanému zadání a architektonickým návrhům, který je, po přizpůsobení pro potřeby konkrétních institucí, možné nasadit jako plnohodnotný systém pracovní evidence.

Klíčová slova: Evidence pracovní doby, PHP, MySQL, OOP, Code First, systém, webová aplikace, zákoník práce, ORM

Design and development of system for managing worktime records

Abstract

This diploma thesis is supposed to address the problematic of obligations related to worktime records management with special focus on the legislation in the academic environment.

In its initial phase, the thesis focuses on study of trustworthy information sources as well as on analysis of the relevant legislative. As a follow-up a description and architecture of a software system designed to address the problematics is designed in a form of web application.

As a final part of the thesis, a protype of system is developed in a form of web application respecting the mentioned description and architectonic designs, which can be used as full worktime management system once it is slightly modified to accommodate the specific needs of individual institutions.

Keywords: Work time records, PHP, MySQL, OOP, code first, system, web app, labour law, ORM

1 Obsah

Seznam obrázků

Seznam tabulek

Úvod

Tato diplomová práce je věnována problematice vedení evidence pracovní doby, které je povinností každého zaměstnavatele a vyplývá ze legislativy zákoníku práce. Problematika je řešena v kontextu pracovní evidence v akademickém prostředí, jeho potřeby nejsou v současném znění zákona brány v potaz a pozměňovacím návrhům, které byly vytvořeny, aby tuto situaci řešily.

Důležitou součástí práce je analýza legislativy a kompilace povinností z ní vyplývající, aby mohl být navržen systém pro evidenci pracovní doby, který umožní zajistit dodržování této legislativy v akademickém prostředí. Mimo současné legislativy jsou analyzovány i pozměňovací návrhy zákonů, které jsou v tuto chvíli projednávány, a které mají velký potenciál ovlivnit způsob, jakým je evidence pracovní doby v akademickém prostředí vedena.

Další důležitou součástí práce je návrh elektronického systému, který usnadňuje vedení evidence pracovní doby a vyvinutí jeho funkčního prototypu. Prototyp systému je implementován formou webové aplikace za využití moderních technologií a je vyvinut tak, aby byl obecný a nezahrnuje tedy potřeby konkrétních institucí či pracovišť.

V závěru práce je provedena evaluace vyvinutého systému a vyhodnocení jeho přínosu pro vedoucí pracovníky v akademickém prostředí při vedení pracovní evidence.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem diplomové práce je navrhnout a implementovat webový systém, který zaměstnavatelům usnadní evidenci pracovní doby tak, aby byly naplněny všechny požadavky kladené zákoníkem práce při minimálním časovém vytížení administrátora systému a evidovaných zaměstnanců.

2.2 Metodika

Metodika řešené problematiky diplomové práce bude založena na studiu a analýze odborných informačních zdrojů. Praktická část práce bude zaměřena na návrh a implementaci systému evidence pracovní doby s využitím technologií PHP a MySQL. Pro technický návrh systému bude využito poznatků z oboru softwarového inženýrství. Vlastní programování systému bude podléhat paradigmatům OOP a Code First, což umožní snadnou udržovatelnost a rozšiřitelnost systému.

3 Teoretická východiska

V první sekci diplomové práce jsou zkompilována teoretická východiska a relevantní informace získané při literární rešerši, které následně poslouží jako východiska pro praktickou část této diplomové práce, která bude věnována navržení systému a vytvoření funkčního prototypu.

Kompilace teoretických východisek se bude skládat z analýzy zákonů, ze kterých vyplývá povinnost pro vedení pracovní evidence, rešerše moderních metodik pro navrhování informačních systémů a ve finále rozbor technologií, které budou využity pro vývoj prototypu systému.

3.1 Právní aspekty evidence pracovní doby

Tato kapitola je věnována rešerši a analýze zákonů, ze kterých vyplývá povinnost vedení pracovní evidence. Povinnost vedení pracovní evidence je primární iniciativou pro vypracování této diplomové práce, a proto je nutné prostudovat zákony vztahující se k tomuto tématu, aby bylo možné sestavit seznam požadavků, které bude muset systém splňovat, aby bylo při jeho využívání možno vyhovět legislativě.

Význam vedení evidence pracovní doby

Evidence pracovní doby je míněna jako způsob doložení správného rozvržení práce. Dále pracovní evidence slouží pro kontrolu dodržování nařízení týkajících se délky pracovní doby, povinných přestávek a předepsaného odpočinku. Evidence pracovní doby má primárně sloužit zaměstnavateli, aby zajistil že zákonná ustanovení jsou řádně dodržována. Může však být vyžádána k nahlédnutí příslušnými kontrolními orgány nebo samotnými zaměstnanci a v takovém případě musí být zaměstnavatel schopen evidenci k nahlédnutí poskytnout. Pokud není zaměstnavatel schopen doložit transparentní evidenci pracovní doby, hrozí mu pokuta 400 tisíc českých korun. (5)

Forma vedení evidence pracovní doby

Forma vedení pracovní evidence není zákonem nijak specifikována, a záleží tedy pouze na samotném zaměstnavateli, jakým způsobem se rozhodne své povinnosti plnit. Jediné specifikum je, že zaměstnavatel musí zvolit formu takovou, aby byla evidence přehledná, srozumitelná a prokazatelná. (6)

Je však důležité zdůraznit, že evidence pracovní doby není evidence docházky – ta může sloužit pouze jako podklad pro prokazatelnost evidence pracovní doby. Docházka eviduje pouze časy příchodů a odchodů na pracoviště, což zdaleka nepokrývá požadavky kladené na evidenci pracovní doby, která musí obsahovat informace o veškerých pracovních výkonech, jejich typech (práce v noci, práce přesčas), držených pracovních pohotovostech, poskytování přestávek a odpočinků mezi směnami. Za vedení evidence pracovní doby a správnost obsažených údajů je přímo zodpovědný vedoucí pracovník, pod kterého dotyční zaměstnanci spadají. (7) (8)

3.1.1 Analýza zákoníku práce (Zákon č. 262/2006 Sb. (9))

Povinnost zaměstnavatele vést pracovní evidenci vyplývá z §96 zákoníku práce, podle něhož je zaměstnavatel povinen evidovat pracovní dobu zaměstnance. Z pracovní evidence musejí být patrné začátky a konce následujících úkonů:

- Pracovní doby
- Nařízená práce přesčas
- Domluvená práce přesčas
- Noční práce
- Doby držení pracovních pohotovostí
- Doby pracovních výkonů během pracovních pohotovostí
- Přestávky
- Nepřetržitý odpočinek mezi směnami

Pracovní doba

§78a zákoníku práce definuje pracovní dobu následovně:

„Pro účely úpravy pracovní doby a doby odpočinku je pracovní dobou doba, v níž je zaměstnanec povinen vykonávat pro zaměstnavatele práci, a doba, v níž je zaměstnanec na pracovišti připraven k výkonu práce podle pokynů zaměstnavatele.“

Z hlediska evidence pracovní doby je důležité, aby byly dodrženy následující pravidla:

- Délka pracovní doby nesmí přesáhnout 12 hodin denně pro zaměstnance starší 18 let. (§83 zákoníku práce);
- Délka pracovní doby nesmí přesáhnout 8 hodin denně pro zaměstnance mladší 18 let a součet odpracovaných hodin ve všech pracovněprávních vztazích nesmí u mladistvých přesáhnout 40 hodin týdně. (§79a zákoníku práce);
- V případě rovnoměrné pracovní doby, součty délek pracovní doby nesmí překročit 40 hodin týdně. Tento limit je nižší pro dvousměnný provoz (38,75h), třísměnný provoz (37,5h) a v hornictví (37,5h). Tyto limity mohou být sníženy na základě kolektivních smluv – zkrácené úvazky. (§79 zákoníku práce);

- V případě nerovnoměrné pracovní doby se využívá konta pracovní doby, ze kterého se hodiny mohou nerovnoměrně čerpat ve vyrovnávacím období 26 po sobě jdoucích týdnů, které lze rozšířit na až 52 týdnů po sobě jdoucích skrze kolektivní smlouvu. (§86 zákoníku práce).

Nařízená práce přesčas

Práci přesčas může podle §93 zákoníku práce nařídit zaměstnavatel výjimečně a z vážných provozních důvodů a může zasahovat i do nařízené týdenní minimální doby nepřetržité odpočinku mezi směnami, která je detailněji popsána v §92 zákoníku práce. Práce přesčas může být nařízena i během svátků a dnů pracovního klidu za situací popsaných v §91 zákoníku práce, a může tak být učiněno nejvýše dvakrát v období po sobě jdoucích 4 týdnů.

Časový objem nařízené práce přesčas nesmí nikdy být více než 8 hodin týdně a zároveň nesmí být překročen roční limit přesčasových hodin, který je 150 hodin. Zde je patrný význam pracovní evidence, jakožto nástroje ke kontrole dodržování těchto limitů.

Domluvená práce přesčas

Limit stanovený pro nařízenou práci přesčas lze podle §93 zákoníku práce překročit po domluvě se zaměstnancem, a to za předpokladu že přesčasy nepřekročí v průměru 8 hodin týdně, přičemž období, ze kterého se průměr počítá činí maximálně 26 po sobě jdoucích týdnů. Toto období může být rozšířeno na až 52 týdnů po sobě jdoucích skrze kolektivní smlouvu.

Z hlediska evidence pracovní doby, je důležité rozlišovat nařízenou práci přesčas od domluvené práce přesčas a zajistit dodržování zmíněných limitů.

Práce v noci

Noční doba je definovaná v §78j zákoníku práce, jakožto časové období mezi desátou hodinou večerní a šestou hodinou ranní a práce vykonaná v noční době je nazývána prací v noci. Zaměstnanec, který odpracuje alespoň 3 hodiny ze své pracovní doby ve 24 po sobě jdoucích hodinách v noční době je podle §78k zákoníku práce označován jako zaměstnanec pracující v noci.

Z hlediska evidence pracovní doby je důležité zajisti, aby bylo zřejmé kolik hodin každý zaměstnanec odpracuje v noční době, a pokud je zaměstnanec klasifikován jako

zaměstnanec pracující v noci, tak je potřeba dodržet omezení vyplývající z §94 zákoníku práce o noční práci. Z těchto omezení vyplývá, že celková délka směny zaměstnance pracujícího v noci nesmí přesáhnout 8 hodin v období 24 hodin po sobě jdoucích. Pokud toto není možné z provozních důvodů zajistit, tak musí být zajištěno, aby průměrná délka směny zaměstnance pracujícího v noci nepřekročila v průměru 8 hodin v období 26 po sobě jdoucích týdnů, přičemž při počítání průměru se vychází z pětidenního pracovního týdne.

Doby držení pracovních pohotovostí

Podle §78h zákoníku práce je pracovní pohotovost doba, kdy musí být zaměstnanec připraven k výkonu práce nad rámec rozvržené činnosti a je držena na dohodnutém místě, které je odlišné od pracoviště. Pracovní pohotovost může být podle §95 zákoníku práce stanovena pouze po dohodě se zaměstnancem a součet délek pracovních pohotovostí nesmí přesáhnout 400 hodin v jednom kalendářním roce. (6)

Z hlediska evidence pracovní doby je potřeba evidovat začátek a konec každé pracovní pohotovosti, aby bylo možné doložit dodržování limitu 400 hodin na kalendářní rok.

Doby výkonů práce během pracovní pohotovosti

Výkony práce probíhající během pracovní pohotovosti se přičítají k běžné pracovní době, pokud nebyla pro dané období již vyčerpána. Pokud byla pracovní doba pro dané období vyčerpána, tak se takový výkon práce přičítá k práci přesčas, jak vyplývá z §95 zákoníku práce.

Z hlediska evidence pracovní doby je potřeba evidovat tyto výkony zvlášť od přesčasů a běžně rozvržené pracovní doby, ale pro účely kontrol limitů pracovní doby a přesčasů je nutné s těmito výkony práce počítat podle pravidel stanovených výše.

Přestávky

Povinnost zaměstnavatele poskytnout pracovníkům oddychové přestávky je popsána v §88, který ustanovuje, že přestávka musí proběhnout nejdéle po 6 hodinách nepřetržité práce (včetně) a mladistvých zaměstnanců je tato doba stanovena na 4,5 hodiny (včetně). Tato přestávka se nezapočítává do běžné pracovní doby a může rozdělena na více menších přestávek, přičemž alespoň jeden úsek musí trvat nejméně 15 minut. Přestávky také nesmějí být poskytnuty na začátku nebo konci pracovní doby. V případě, že z provozních důvodů nemohou být práce přerušeny, je potřeba zaměstnanci nějak jinak zajistit přiměřenou dobu na oddych a jídlo, přičemž tato doba se započítává do pracovní doby a nemusí výt evidována. Nicméně v případě mladistvých musí být vždy poskytnuta plnohodnotná přestávka podle výše zmíněných pravidel.

Dále podle §89 zákoníku práce, pokud má zaměstnanec právo na bezpečnostní přestávku podle právních předpisů, tak se tato přestávka započítává do pracovní doby a pokud přestávka na jídlo a oddech připadne na dobu bezpečnostní přestávky, tak se i tato doba započítává do pracovní doby.

Z hlediska evidence pracovní doby je nutné evidovat, že byly poskytnuty bezpečnostní a oddychové přestávky, není však nutno evidovat začátky a konce těchto přestávek.

Nepřetržitý odpočinek mezi směnami

Podle §90 zákoníku práce je zaměstnavatel povinen rozvrhnout pracovní dobu tak, aby měl zaměstnanec mezi koncem jedné směny a začátkem další směny alespoň 11 hodin nepřetržitého odpočinku nebo 12 hodin, pokud se jedná zaměstnance mladšího 18 let. Za zvláštních podmínek specifikovaných v §90 zákoníku práce může být povinný odpočinek zaměstnanci staršímu 18 let zkrácen na minimum 8 hodin za podmínky, že mu bude následující odpočinek náležitě prodloužen.

Speciálním případem podle §90a zákoníku práce jsou sezónní práce v zemědělství, kde může být zaměstnanci staršímu 18 let zkrácen povinný odpočinek na 8 hodin s tím, že mu budou krácené hodiny odpočinku nahrazeny během následujících tří týdnů od zkrácení.

Dále podle §92 zákoníku práce zaměstnanci náleží minimální nepřetržitý odpočinek v týdnu, který musí trvat alespoň 35 hodin u zaměstnance staršího 18 let a alespoň 48 hodin u zaměstnance mladšího 18 let. Ve speciálních případech popsaných v §90 zákoníku práce

je možné zkrátit minimální nepřetržitý týdenní odpočinek u zaměstnanců starších 18 let na 24 hodin, s tím že za období dvou týdnů bude činit součet nepřetržitých týdenních odpočinků alespoň 70 hodin.

§90 zákoníku práce také ustanovuje, že v zemědělství může být dohodnuto se zaměstnanci, že nepřetržitý týdenní odpočinek bude poskytován formou rozdělení 105 hodin během tří týdnů nebo 210 hodin během 6 týdnů. Způsob distribuce těchto hodin není specifikována.

Během nepřetržitého odpočinku mezi směnami a nepřetržitého týdenního odpočinku může být držena pracovní pohotovost a může být vykonávána práce přesčas.

Z hlediska evidence pracovní doby je potřeba, aby bylo prokazatelné, že zaměstnancům je poskytnut nepřetržitý odpočinek za výše zmíněných podmínek.

Závěr

Na základě analýzy zákoníku práce byl sestaven seznam kritérií, která musí systém pro pracovní evidenci splňovat, aby byli splněny zákonné povinnosti vyplývající z §96 zákoníku práce. Z pracovní evidence musí být zřejmé nebo dohledatelné:

- Začátky a konce pracovních dob, nakázaných přesčasů, domluvených přesčasů, pracovních pohotovostí a výkonů práce během pracovních pohotovostí.
- Týdenní součty pracovních dob a nakázaných přesčasů.
- Součty dob domluvených přesčasů za zvolené období s délkou 26 – 52 týdnů.
- Roční součty dob pracovních pohotovostí.
- Poskytnutí oddychových přestávek a bezpečnostních přestávek.

3.1.2 Pracovní doba akademických pracovníků

Vzhledem k tomu, že se tato práce zabývá evidenci pracovní doby v akademickém prostředí, je potřeba uvést, že zákonné povinnosti popsané v předchozí kapitole vůbec neberou v potaz zaměstnancem řízenou pracovní dobu typickou pro akademické prostředí. V minulosti již proběhly pokusy o úpravu zákona pro zohlednění výše popsaných specifik, nicméně pozměňovací návrhy zákonů nikdy neprošly schvalovacím procesem českého zákonodárného aparátu. Situací ohledně pracovní doby v akademickém prostředí se opakovaně zabývá poslanec prof. Ing. Karel Rais CSc., MBA, dr. h. c., podle jehož slov není možné, aby akademickým pracovníkům byla definována pevná pracovní doba, neboť ta je těchto zaměstnanců tvořena nejen rozvrženou pedagogickou činností, ale i samostatnými tvůrčími a výzkumnými činnostmi, které si zaměstnanec v praxi rozvrhuje sám. Vysoké školy samy označují povinnosti vyplývající ze zákoníku práce za něco, co je vhodné pouze pro fabriku a zmiňují, že výzkumnou činnost není možné provozovat podle „píchaček“ a pevných pracovních dob, kterou nejsou slučitelné s akademickým prostředím a jeho potřebami. (10)

Neúspěšný pozměňovací návrh zákona č. 262/2006 Sb. tisk 903/0-2

Pan poslanec již v roce 2016 sám vznesl pozměňovací návrh zákoníku práce, který měl situaci vyřešit tím, že bude vyjme akademické pracovníky z povinnosti rozvrhování pracovní doby, nicméně tento návrh nebyl uveden v platnost, protože během jeho projednávání nastal konec volebního období.(11)

3.1.3 Pozměňovací návrh zákona č. 111/1998 Sb. č. 109/0-8

Dne 21. února 2018 byl poslanecké sněmovně přeložen nový pozměňovací návrh pro zákon o vysokých školách skupinou poslanců skládající se z Jana Hamáčka, Jana Birkeho, Kateřiny Valachové a Petra Dolínka. (12)

Tento návrh přidává do zákona o vysokých školách nový §70a, s názvem *„Pracovní doba akademických pracovníků“*, který oficiálně rozděluje pracovní činnosti akademických pracovníků na:

- Přímou pedagogickou činnost;
- Práce související s přímou pedagogickou činností;
- Vědeckou, výzkumnou, vývojovou a inovační, uměleckou nebo další tvůrčí činnost.

Nový paragraf dále ustanovuje následující:

- *„Akademický pracovník je povinen být na pracovišti zaměstnavatele nebo na jiném dohodnutém místě v době stanovené rozvrhem jeho přímé pedagogické činnosti a v případech, které stanoví v souladu se zákoníkem práce zaměstnavatel.“*
- *„Jde-li o výkon jiné práce než podle odstavce 2, vykonává akademický pracovník sjednanou práci v pracovní době, kterou si sám rozvrhuje, a na místě, které si sám určí. Náklady, které akademickému pracovníkovi vzniknou výlučně v souvislosti s výkonem práce na jiném místě než na pracovišti zaměstnavatele podle věty první, se nepovažují za náklady vzniklé v souvislosti s výkonem závislé práce, a není-li dohodnuto jinak, hradí je akademický pracovník.“*
- *„Zaměstnavatel **eviduje jen tu část pracovní doby, kterou sám rozvrhuje.**“*

Dopad pozměňovacího návrhu č. 109/0-8

Z hlediska této práce a evidence pracovní doby je nejdůležitější poslední ustanovení, které velmi znatelně zjednodušuje vedení pracovní evidence pro akademické pracovníky tím, že zaměstnavatel bude povinen vést evidenci pracovní doby pouze pro činnosti, které sám rozvrhuje – tedy výuku, dozory na zkouškách účasti na konferencích atp., zatímco samostatné tvůrčí a výzkumné činnosti zaměstnance jsou od evidence pracovní doby oproštěny. (13)

Schvalovací proces pozměňovacího návrhu č. 109/0-8

Průběh schvalování tohoto pozměňovacího návrhu byl během vypracovávání této diplomové práce ostře sledová, neboť jeho ne/schválení má silný vliv na konečný výstupy.

- **21. března 2018**: Poslanecká sněmovna obdržela pozměňovací návrh č 109/0.
- **31. října 2018**: Návrh byl schválen poslaneckou sněmovou.
- **20. prosince 2018**: Návrh byl zamítnut senátem a vrácen do poslanecké sněmovny.
- **22. ledna 2019**: Sněmovna setrvala na schválení návrhu.
- **30. ledna 2019**: Prezident republiky podepsal zákon v jeho novém znění.

Pozměňovací návrh úspěšně prošel celým českým zákonodárným aparátem (12) a vstoupí v platnost dne **1. července 2019**. (13)

Aby tato diplomová práce setrvala relevantní i po jejím odevzdání v březnu 2019, bude vypracována s předpokladem, zákon je již platný v jeho novém znění.

3.2 Teorie vývoje webových aplikací - Frontend

Tato kapitola je věnována uživatelsky viditelné části webu, jinak nazývané jako „Frontend“. Frontend je část webové aplikace, která se stáhne ze serveru do paměti prohlížeče, kde je spuštěna zobrazena a interpretována. Jedná se tedy především o uživatelské rozhraní a klientské skripty, jejichž účelem může být např. zjednodušení ovládání aplikace pro uživatele nebo klientská validace vstupů. Netřeba dodávat, že když je tento kód spouštěn v prohlížeči, tak je potřeba testovat vyvíjený frontend na všech relevantních platformách – v tomto případě ve všech prohlížečích. (14)

Vývoj frontendu je specifický tím, že vývojář pracuje ve velkém množství s deklarativními programovacími jazyky, s jejichž pomocí jsou vytvářeny strukturované dokumenty, které jsou nakonec za pomoci renderovacího jádra prohlížeče převedeny do grafického výstupu na obrazovce. Kromě pouhé implementace designu s využitím deklarativních jazyků je možné implementovat i výše zmíněné klientské skripty za pomoci imperativních programovacích jazyků, které jsou též interpretovány prohlížečem. (15)

Zmíněné deklarativní jazyky jsou v případě vývoje webových aplikací zastoupeny značkovacím jazykem HyperText Markup Language (HTML), jehož primárním účelem je strukturování a zobrazování dat a popisným jazykem Cascading Style Sheets (CSS), který je primárně určen jako nástroj pro implementaci grafického designu a formátování. Jako imperativní skriptovací jazyk u webových aplikací většinou figuruje JavaScript, známý též jako ECMA Script. JavaScript má na rozdíl od HTML a CSS jakousi konkurenci v podobě VBScriptu (pouze Internet Explorer), Silverlightu nebo Java Appletů, nicméně v praxi se používá pouze JavaScript a odvozené frameworky. (16)

3.2.1 HyperText Markup Language (HTML)

HTML je značkovací deklarativní programovací jazyk, určený k popisu formy zobrazení textového obsahu, obrázků a dalších druhů multimédií na webové stránce. K zápisu HTML se používají strukturální značky (tagy), které umožňují prohlížeči rozeznat jaká forma obsahu je zobrazována. (17)

Od počátku webu vešlo v používání mnoho verzí HTML, každá přinášející nové funkcionality.

Tabulka 1 - Vývoj HTML (18)

Verze HTML	Rok vydání
HTML	1991
HTML 2.0	1995
HTML 3.2	1997
HTML 4.01	1999
XHTML	2000
HTML5	2014

Vývoj ve způsobech využívání HTML

Během jeho existence bylo HTML průběžně obohacováno o možnosti formátování obsahu, které jím bylo zobrazováno, což se nejvíce projevovalo webů z éry HTML4, kde téměř každý tag obsahoval informace o formátování vloženého obsahu. Tyto praktiky byly postupně potlačeny s rozvojem kaskádových stylů, které jsou schopné za pomoci tříd, pseudotříd, identifikátorů a pseudoidentifikátorů přiřadit vlastnosti a formátování mnohem efektivněji. V éře HTML5 je již formátování skrze HTML atributy považováno za dávno překonanou praktiku, které by se vývojáři neměli dopouštět, i přesto, že prohlížeče tyto metody stále podporují. (17)

Validita HTML dokumentů

Pro vytváření kvalitních HTML dokumentů, nebo aplikací generujících kvalitní HTML dokumenty je potřeba, aby výsledné HTML bylo validní podle standardů konsorcia W3C. Mezi požadavky pro validitu dokumentů se řadí například nutnost správného uzavírání značek, respektování pravidel pro vnořování značek a pravidel pro formátování

atributů značek. Zajištění validity dokumentu je důležité pro zvýšení kompatibility mezi prohlížeči a také z hlediska optimalizace pro vyhledávače, neboť nevalidní HTML stránky jsou buď ignorovány ve vyhledávání anebo jsou řazeny za validními, i když jsou pro dané vyhledávání relevantnější. (19,17)
Pro ověření validity HTML dokumentu je možné využít některý z mnoha online nástrojů, jako například https://html5.validator.nu/, který je určen konkrétně pro dokumenty ve verzi HTML5.

3.2.2 HTML preprocesory

Pro zlepšení kvality a udržovatelnosti kódu se často využívá takzvaných preprocesorů, které umožňují vývojářům vytvářet zdrojový kód pro HTML dokumenty s využitím jiné než HTML syntaxe. Ideou pro vytváření a využívání preprocesorů jsou následující čtyři programátorská přesvědčení: (25,26)

- Kód by měl být „hezký“
- Kód by se neměl opakovat
- Kód by měl být dobře zanořovaný
- Struktura kódu by měla být jasná na první pohled

Zdrojový kód psaný používaný preprocesory je v naplňování těchto předpisů znatelně lepší než holé HTML, ale není interpretovatelný v prohlížečích, a proto je při sestavení webové aplikace převeden do běžného HTML, srozumitelného pro prohlížeče. Mezi nejvýznamnější zástupce z řad HTML preprocesorů patří: (25)

- Slim
- Haml (HTML abstraction markup language)
- Pug (také známý jako “Jade”)
- Emblem.js

Nejpopulárnějším HTML preprocesorem je v současnosti Pug/Jade, což lze přisoudit jeho integrací do MVC frameworku Express pro prostředí Node.js.(24)

3.2.3 Cascading Style Sheets (CSS)

Kaskádové style jsou používány ve vývoji webových aplikací jakožto způsob pro definici formátování obsahu obsažených v elementech HTML. Stejně jako HTML, bylo i CSS navrženo konsorciem W3C, které dohlíží nad vývojem webových technologií. Hlavní ideou kaskádových stylů je oddělit strukturu a obsah ve značkovacích jazycích od formátování a grafické realizace dokumentů. (20)

Tabulka 2 Vývoj CSS (21)

Verze CSS	**Rok vydání**
CSS 1.0	1996
CSS 2.0	1998
CSS 2.1	2011
CSS 3.0	Stále ve vývoji

Syntaxe kaskádových stylů

Zápis kaskádových stylů se vždy skládá ze dvou částí – množiny selektorů a množiny deklarací. Selektory zpravidla reprezentují názvy HTML prvků, třídy anebo identifikátory specifikované v dokumentu. Mimo těchto tradičních selektorů kaskádové styly podporují i sadu pseudoselektorů, jako například:‘*:first-child*‘, ‘*:nth-child‘,’&:hover’* a mnoho dalších, což usnadňuje pokročilejší formátování, které by jinak vyžadovalo využití skriptů. Po vymezení selektorů vždy následuje pár složených závorek obsahující deklarace popisující požadované formátování, které se vždy skládají z názvu vlastnosti, kterou je žádoucí upravit, dvojtečky a požadované hodnoty.

Výhody kaskádových stylů

Mezi hlavní výhody formátování za pomoci kaskádových stylů se řadí především mnohem širší možnosti formátování oproti využití HTML atributů, a to zejména v oblasti deklarování okrajů (margin/padding) a rozložení prvků na webu, k čemuž se v čistém HTML používaly soustavy vnořených tabulek, které vytvářeli velmi nepřehledný kód a zároveň to bylo velmi pracné. Další velkou výhodou kaskádových stylů je možnost jejich „cachování“ – tedy jejich ukládání v lokálním prostředí prohlížeče pro zrychlení dalšího načtení webu a možnost nastavení různých stylů pro různá zařízení, čemuž vděčíme za vznik responzivních webů a webových aplikací. Mimo to lze deklarace kaskádových stylů také ovlivňovat za pomoci klientských skriptů (většinou JavaScriptu), což dává vývojářům znatelně rozšířené možnosti při vytváření dynamických webu, které mění svůj vzhled v reakci na uživatelský vstup. (20)

Nevýhody kaskádových stylů

Hlavní nevýhodou kaskádových stylů jsou rozdílné úrovně podpory v majoritních prohlížečích, což vyžaduje neustálé testování a přizpůsobování pro „problematické“ prohlížeče. K tomu se občas využívá podmíněných komentářů, které jsou v ideologickém rozporu s deklarativním charakterem HTML i kaskádových stylů.(20)

Dalším úskalím využívání kaskádových stylů je jejich velmi obtížná udržovatelnost, což se projevuje zejména v případech, kdy více kodérů edituje CSS v jednom projektu. Může docházet ke kolizím názvů selektorů, zbytečnému opakování kódu a vynucování si vlastností za pomoci *„ !importnant “* deklarace, což všechno dohromady silně komplikuje údržbu kódu a znesnadňuje změny v implementaci designu.(23)

Ukázka podmíněného formátování: (20)

```
<!--[if IE]>
 <style type="text/css">
 #upozorneni { color: blue;}
 </style>
 <![endif]-->
```

3.2.4 CSS Preprocesory

CSS preprocesory rozšiřují základní CSS o další funkcionality, jako jsou například deklarace proměnných, cykly, skládání, dědění, zanořování a cykly, které umožňují vývojářům efektivněji vytvářet snadno čitelné deklarace stylů. (28)

Proměnné znatelně zlepšují udržovatelnost kódu, protože je možné si s jejich pomocí nadefinovat opakovaně využívané hodnoty, jako třeba barvy, zaoblování hran, odsazení odstavců a další, což vývojář ocení v okamžiku, kdy se designér rozhodne změnit některý z těchto globálních a atributů. S holými kaskádovými styly by vývojář musel upravovat dotyčnou hodnotu v každém místě jejího výskytu, zatímco s preprocesorem pouze upraví hodnotu konfigurační proměnné.(27)

Nejvýznamnějšími CSS preprocesory v současné době jsou SASS/SCSS a LESS.

LESS

Preprocesor LESS je vyvíjený v JavaScriptu, což umožňuje kompilaci přímo v prohlížeči a implementaci dynamických funkcionalit, které by nebyly se samotným CSS nebo jiným preprocesorem možné. Kromě těchto dynamických funkcí je LESS jako preprocesor typický hlavně tím, že mění syntaxi stylů. (26) Ukázka syntaxe LESS:

Zdrojový LESS

```
.header {
 .menu { border-radius: 5px; .no-borderradius &
      { background-image: url('images/button-background.png'); }
  }
}
```

Výstupní CSS

```
.header .menu {
 border-radius: 5px; } .no-borderradius
.header .menu { background-image: url('images/button-background.png');
}
```

SASS/SCSS

Tento preprocesor je dostupný ve dvou syntaxích – SASS a LESS. Tradiční SASS syntaxe je stejně jako LESS velmi odlišná od standardních stylů, ale je zde i možnost využívat alternativní syntaxi SCSS (= Sassy CSS), která je téměř totožná s tradičním CSS, ale stále umožňuje vnořování selektorů. (26) Ukázka syntaxe SCSS:

Zdrojové SCSS:

```
$font-stack:    Helvetica, sans-serif;

$primary-color: #333;

body {

  font: 100% $font-stack;

  color: $primary-color;

}
```

Výstupní CSS:

```
body {

  font: 100% Helvetica, sans-serif;

  color: #333;

}
```

3.2.5 CSS Frameworky

Frameworky kaskádových stylů jsou soubory předpřipravených tříd, které umožňují vývojářům ušetřit spoustu času, protože s jejich využitím nemusejí v každém projektu znovu vynalézat kolo. Hlavním přínosem společným pro všechny CSS frameworky je implementace takzvaných „gridů“, jichž mají některé frameworky více i druhů. Dalším velkým přínosem frameworků jsou předpřipravené responsivní třídy a unifikace přístupů pro vývoj frontendu. (30)

Bootstrap

Zdaleka nejpoužívanějším CSS frameworkem je Bootstrap, který vděčí za svou popularitu především velkému množství tutoriálů a dodatečných pluginů. Stejně jako všechny ostatní frameworky nabízí solidní „grid“, responsivní třídy a mnoho předpřipravených webových prvků. (31)

Poznámka autora: Bootstrap je dostupný buď jako předkompilované CSS anebo v podobě zdrojových kódů v LESS, kde je možné si framework detailněji nakonfigurovat. Bootstrap tedy bude přirozenější volbou pro vývojáře, kteří využívají preprocesor LESS, pokud využívají preprocesor.

Foundation

Zurb Foundation je druhým nejpopulárnějším CSS frameworkem a na rozdíl od Bootstrapu se soustředí více na to, aby dal vývojářům nástroje pro snadné vytvoření jejich vlastních webových prvků raději, než aby dodával velkou sadu předpřipravených prvků. Foundation je také považován ze lepší volbu v případě, že je kladen maximální důraz na responzivitu webové aplikace. (31)

Poznámka autora: Foundation framework je stejně jako Bootstrap dostupný buď v předkompilované verzi v podobě čistého CSS, anebo mohou vývojáři využít zdrojové kódy dostupné v SCSS, kde si mohou framework do detailu nakonfigurovat. Z tohoto faktu obdobně vyplývá, že Foundation framework je přirozenější volbou pro vývojáře, kteří využívají preprocesor SASS s dialektem SCSS, pokud využívají preprocesor.

3.2.6 JavaScript (JS)

Autorem původní implementace JavaScriptu je Brendan Eich, který v roce 1995, kdy jazyk vznikl pracoval pro společnost Netscape, známou především pro historický webový prohlížeč Netscape Navigator. V červenci 1997 byl JavaScript standardizován společností ECMA (European Computer Manufacturers Association) a tato standardizovaná verze, která je dnes nejpopulárnějším programovacím jazykem nese oficiální název ECMAScript. Navzdory svému neoficiálnímu, ale častěji používanému názvu „JavaScript“, nemá tento programovací jazyk s Javou společného téměř nic a jedná se pouze o marketingový tah. S jazykem Java má pouze podobnou syntaxi, kterou je možné pozorovat i u ostatních populárních programovacích jazyků jako C, C++ a C#. (32)

Ukázka implementace klientských skriptů formou JavaScriptu: (33)

Následující blok kódu zobrazuje využití JavaScriptu uvnitř HTML dokumentu, kde je za pomoci klientského skriptu vložen dodatečný obsah.

```
<html>

   <body>

      <script language = "javascript" type = "text/javascript">

         <!--

            document.write("Hello World!")

         //-->

      </script>

   </body>

</html>
```

Idea a využití JavaScriptu

Paradigmaticky se jedná o objektově orientovaný událostmi řízený programovací jazyk, který je využíván v drtivé většině webových aplikací pro vytváření klientských skriptů. Mimo klientských scriptů je možné JavaScript najít i jako serverový skriptovací

jazyk v podobě technologie Node.js, který v současné době nabírá na popularitě a nabízí i mnoho frameworků, z nichž některé implementují i moderní MVC architekturu. (32)

Bezpečnostní rizika JavaScriptu

Jako všechny technologie, i JavaScript má svá úskalí. V případě JavaScriptu je tímto úskalím jeho zneužitelnost k páchání nekalých činností. Při vytváření jakýchkoliv klientských skriptů je potřeba pamatovat na to, že skript je plně čitelný uživatelem, u kterého je spouštěn a může být i za běhu modifikován přímo v prohlížeči, kde je interpretován. Nelze tedy spoléhat na Javascript, jako na jediný nástroj pro kontrolování uživatelských aktivit a validaci vstupů, protože charakter jeho spouštění umožňuje snadnou modifikaci ze strany uživatele a obcházení stanovených omezení.(34)

Další velkou slabinou Javascriptu je jeho zneužitelnost formou Cross-Site Scriptingu (XSS), kde dochází ke snaze „podstrčit“ skript prohlížeči jiného uživatele ke spuštění, což může způsobit nechtěné chování, kdy například uživatel nechtěně odesílá své stisky kláves na server cross-site scriptera nebo provádí jiné nevědomé akce skrze svůj prohlížeč, který naivně spouští cizí skript.(34)

Ukázka XSS útoku

Následující skript odešle cookie uživatele na webovou službu útočníka, což může být využito ke zneužití účtu uživatele. (35)

```
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
```

Poznámka autora: *Nejefektivnější obranou proti tomuto počínání je tzv „escapování“ JavaScriptu na relevantních uživatelských vstupech. Jedná se o proceduru, která očistí uživatelské vstupy o veškerý JavaScriptový kód, anebo nějak znemožní jeho fungování.*

3.2.7 JS frameworky, knihovny a preprocesory

Jakožto jazyk pro klientské skriptování plnil JavaScript vždy svoji funkci, ale to neznamená, že by nebylo co vylepšovat. V případě JavaScriptu je tohoto docíleno skrze velké množství rozšiřujících frameworků, které šetří čas vývojářům a unifikují svět vývoje klientských skriptů. Využití JavaScriptových frameworků umožňuje vývoj větších klientsky orientovaných webových aplikací s interaktivními prvky bez vzniku větších problémů s údržbou a strukturování zdrojových kódů. Většina JavaScriptových frameworků funguje na principu architektury MVC, který je známý svojí škálovatelností a snadnou udržovatelností. (36)

Kromě rozsáhlých frameworků, které často úplně mění způsob používání JavaScriptu je tento jazyk známý i pro svojí velmi bohatou zásobu knihoven. Knihovny jsou předpřipravené bloky kódu, které zjednodušují realizaci některých úkonů, anebo předpřipravují často implementované funkce, co dále přispívá k unifikaci světa JavaScriptu. Nejvýznamnější JavaScriptovou knihovnou pro vývojáře webových aplikací je jQuery, které znatelně zjednodušuje přístup a manipulaci HTML prvků, a také usnadňuje tvoření AJAX požadavků, které jsou součástí každé moderní webové aplikace. (37)

Stejně u jako HTML a CSS jsou i pro JavaScript k dispozici preprocesory, které mění syntaxi jazyka nebo přidávají nové funkce, či přísnější nároky na vývojáře. Nejvýznamnějšími JS preprocesory jsou v současnosti TypeScript a CoffeeScript. (40)

Knihovna jQuery

Jedná se o JavaScriptovou knihovnou s podporou ve všech významných prohlížečích, která je určená především ke zjednodušení provádění manipulací s HTML dokumentem a programování chování prvků v HTML dokumentu. Stejně jako CSS se snaží oddělit formátování dokumentu od struktury zobrazování obsahu, smyslem jQuery je oddělit veškeré chování a vyřizování událostí od obsahu popsaného značkovacím jazykem HTML. (38)

Framework Angular/AngularJS

Angular je open source framework JavaScriptu podporovaný ze strany Google, který se pyšní největší komunitou vývojářů v oblasti vývoje webových aplikací. Angular se snaží především o realizaci dynamického HTML dokumentu, čehož dosahuje rozšířením

HTML o řadu dodatečných atributů a elementů. Významným atributem Angularu je takzvané dvoucestné „bindování“, které vytváří vazby mezi uživatelskými vstupy v dokumentu a datovým modelem aplikace. (36)

Framework React.js

Tento framework je relativní novinkou ve světě JavaScriptových frameworků a je známý především proto, že je využíván Facebookem a Instagramem, což mu dává silnou pozitivní propagaci. První vydání proběhlo v roce 2013, ale jeho komunita velmi rychle narostla a v současnosti je to nejrychleji rostoucím frameworkem v komunitě JavaScriptu. React se zaměřuje na rychlé vykreslování složitých uživatelských rozhraní, k čemuž využívá virtuální objekt modelu dokumentu (DOM), který je přístupný ve stejné podobě na straně klienta i serveru. Velkou výhodou frameworku je jeho znovu-využitelnost, protože všechno v Reactu je implementováno ve formě samostatných komponent. (36)

Preprocesor TypeScript

TypeScript je JS preprocesor vyvíjený společností Microsoft jako a je publikován jako open source pod licencí Apache. Účelem TypeScriptu je především rozšíření možností škálovatelnosti oproti holému JavaScriptu a „zpřísnění“ jazyka skrze implementaci povinného typování, které vede ke kvalitněji psanému kódu, ale má zvyšuje nároky na kvalitu vývojářů. Podobně jako u ostatních preprocesorů, kód napsaný v TypeScriptu je nutné zkompilovat do standardního JavaScriptu při sestavování aplikace, aby mohl být interpretován v prohlížeči. (40)

Preprocesor CoffeeScript

CoffeeScript je na rozdíl od TypeScriptu vyvíjený komunitou vývojářů, a soustředí se především na vylepšení syntaxe JavaScriptu, raději než na jeho transformaci v jiný jazyk s jinými pravidly. Bohužel nemá zdaleka takovou podporu v prohlížečích jako TypeScript, a proto není zdaleka tak populární. (40)

3.3 Teorie vývoje webových aplikací – Backend

V předchozí kapitole byl popsán význam klientské části webové aplikace a typický „stack“ technologií, ze kterých se frontend obvykle skládá. Tato kapitola bude naopak věnována té části webové aplikace, se kterou uživatel nikdy nepřijde do přímého kontaktu, pokud funguje správně.

Backend webové aplikace se skládá ze souboru technologií, obvykle zahrnujících SQL nebo NoSQL databázové úložiště, HTTP serveru a nějakého skriptovacího programovacího jazyka. Backend je zodpovědný za přijímání HTTP požadavků, jejich validaci a zpracovávání náležitých odpovědí, většinou v podobě HTML dokumentu, který bude zobrazen v prohlížeči uživatele. (41)¨

Zpracování požadavku od klienta nevyhnutelně zahrnuje rozpoznání obsahu, který je klientem vyžadován, k čemuž se používá takzvaný „routing“. Routing funguje tak, že URL adresa zaslaná klientem je rozdělena na předem definovaný set prvků, které mohou zahrnovat moduly, parametry a odkazy na funkce v backendových sckriptech, které mají být spuštěny. V případě webových aplikací je url téměř vždy „routována“ ke spuštění nějakého serverového skriptu, který zpravidla spouští další a další serverové skripty, dokud v finále jeden ze skriptů neukončí serverovou úlohu a pošle odpověď klientovi. (42)

Bezpečnostní požadavky na implementace backendu

Vývojáři backendů by měli kromě svých primárních dovedností alespoň základní přehled v oboru informační bezpečnosti, aby dokázali u výstupů své práce eliminovat maximum bezpečnostních zranitelností, které by jinak mohli způsobit neočekávané problémy a potenciálně nevyčíslitelné škody. (42)

3.3.1 Relační (SQL) databáze

Relační databáze jsou nejpoužívanějším způsobem, jakým webové aplikace ukládají persistentní data. Principem relačních databází je vytvoření přísného schématu databázových tabulek, které modelují data skrze matematickou teorii relací. Takto definovaný model umožňuje provádět dotazy psané dotazovacím jazyce SQL, který se může mírně lišit v závislosti na využité implementaci relační databáze. Podle relační teorie jsou data uložena formou záznamů (řádek) v tabulkách, které mají určitý počet atributů (sloupců) a jsou mezi sebou provázány za pomoci cizích klíčů. Hodnoty atributů jsou navíc ještě podřízeny takzvanými omezeními („constraints“), které pomáhají udržení datové integrity a kompatibilitu s relační teorií. (43) Mezi nejpopulárnější implementace SQL databází patří: (44)

- Oracle
- MySQL
- Microsoft SQL
- PostgreSQL
- MariaDB

Oracle

Oracle je současně nejvíce využívanou implementací SQL databáze, za co vděčí především tomu, že byla jednou z prvních a je k dispozici mnoho edicí, díky čemuž je pro klienty snadné škálovat dle jejich potřeb. Oracle je navržen na práci s obrovským množstvím dat a má také vysoké požadavky na hardware, které mohou podražit jeho adopci a samotná licence pro provozování Oracle databáze je také poměrně drahá, takže je vhodný především pro organizace s vysokým rozpočtem, které potřebují pracovat s velkým množstvím dat a mají využití pro pokročilé funkcionality Oracle databáze. (44)

MySQL

Tato implementace relačních databází je nejpopulárnější, jakožto úložiště dat webových aplikací. Na rozdíl od Oracle je možné provozovat MySQL databázi zcela zdarma, i když nejsou ve „free“ verzi k dispozici veškeré funkce. Původní verze byla vyvinuta společností Sun Microsystems a byla publikována jako open source, nicméně ta

byla později odkoupena společností Oracle, která sama vyvíjí vlastní databázový systém, což způsobilo paniku mezi uživateli MySQL, kteří se začali přiklánět k alternativám z obavy, že Oracle nebude chtít vyvíjet dva navzájem si konkurující databázové systémy. MySQL je ideální pro organizace a vývojáře, kteří potřebují plnohodnotný SQL systém, ale nemají rozpočet pro hardwarově náročný Oracle a jeho licenci. (44,45)

Microsoft SQL

Implementace SQL od společnosti Microsoft byla původně určena pouze pro běh na operačním systému Windows, ale od verze z roku 2016 byla přidána podpora i pro operační systémy Linux. Na rozdíl od Oracle není Microsoft SQL tak efektivní při využití hardwaru a jeho licence jsou podobně drahé, nicméně tato implementace je známá tím, že velmi dobře spolupracuje s dalšími technologiemi společnosti Microsoft. Tento databázový systém je tedy nejvhodnější pro velké společnosti s velkým rozpočtem, které využívají mnoho dalších Microsoft technologií. (44)

PostgreSQL

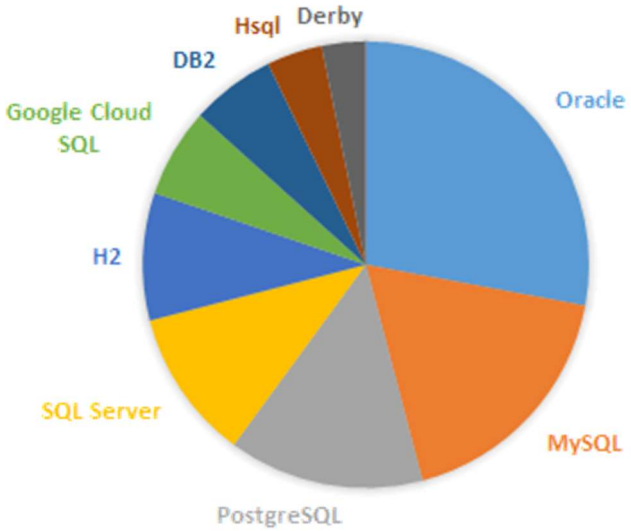
Obdobně jako MySQL je i tato implementace relačních databází velmi populární mezi vývojáři webových aplikací, čemuž vděčí především za to, že podporuje formát JSON a je zcela zdarma. Dalšími výhodami této implementace je schopnost držení velkých objemů dat a podporu využívání databázového systému mnoha uživateli současně. Na druhou stranu PostgreSQL trpí nekvalitní dokumentací, obtížnou konfigurací a špatnou výkonností při spouštění velkého množství dotazů současně. PostgreSQL je tedy ideální pro organizace, které mají nižší rozpočet, potřebují skladovat velké množství dat, ale nepotřebují, aby databáze byla rychlá. (44)

MariaDB

MariaDB nabízí několik prodejních edicí, z nichž jedna je dokonce zdarma a nabízí celý soubor zásuvných modulů, které mohou rozšiřovat funkcionality databázového systému. MariaDB je známá pro svoji stabilitu, rychlost a možnost zobrazení „progress baru“, při zpracovávání dotazů. Dále je velmi významný fakt, že MariaDB je téměř kompletně kompatibilní s MySQL. Tato implementace databázového systému je vhodná především pro organizace a vývojáře, kteří hledají alternativu pro MySQL. (44)

Popularita databázových systémů

Zde je k nahlédnutí graf znázorňující popularitu významných SQL systémů:



Obrázek 1 - Graf popularity SQL systémů (46)

3.3.2 NoSQL Databáze

NoSQL je koncept databázového systému, kde se pro ukládání a organizaci dat využívají jiné prostředky než tradiční relační tabulky. Důvodem pro uvedení takovéhoto přístupu je požadavek na lepší horizontální škálovatelnost, kterou přísná schémata SQL databází značně komplikují. NoSQL databázové systémy žádné schéma nemají, a proto tomuto omezení nepodléhají, neboť tyto systémy jsou většinou implementovány formou "key => value", spíše než definováním striktních relací a jsou silně optimalizovány pro práci s velký množstvím záznamů. Největší využití NoSQL databázových systému lze pozorovat v oblasti „Big Data" a takzvaných „Real-Time" webů. Úložiště často silně preferují dostupnost a toleranci oproti datové konsistenci. Hlavními bariérami, které brání rozšíření NoSQL databázových systémů v praktickém využití jsou nepřítomnost plnohodnotné implementace transakčního modelu ACID, nedostatečná standardizace systémů a fakt, že mnoho organizací investovalo znatelné prostředky do svých systémů fungujících na bázi SQL databází respektujících relační teorii. (47)

MongoDB

Nejvýznamnější implementací NoSQL databázového systému je MongoDB, které podporuje využívání strukturovaných i nestrukturovaných dat. Tento databázový systém je velmi ohebný a přizpůsobivý, což je možné, protože se připojuje do aplikací se za pomoci databázových ovladačů, které jsou vždy optimalizovány pro danou platformu. Na rozdíl od ostatních NoSQL databázových systémů je MongoDB schopné integrovat i schématické datové struktury a zpracovávat v omezené míře SQL dotazy, což z něj učinilo přívětivou volbu pro organizace a vývojáře, kteří chtějí vyzkoušet NoSQL databázi, ale zároveň nechtějí zcela zahodit veškeré nabrané zkušenosti a významné výhody SQL databázových systémů. Dalšími přednostmi MongoDB jsou podpora zásuvných modulů a validace vstupů na databázové úrovni. Další velkou výhodou je pokročilá integrace tohoto databázové systému do vývojářského prostředí Node.js a jeho populárního frameworku Express. (44)

3.3.3 Programovací jazyk PHP

PHP je serverový skriptovací jazyk, určený pro vývoj webových aplikací. První verze jazyka PHP byla představena v roce 1994 pod jeho plným názvem „Personal HomePage“, nicméně později bylo PHP převzato do údržby organizací Zend Foundation a interpretace zkratky se změnila na „Hypertext Preprocessor“. Soubory s PHP kódem nesou koncovku „.php“ a dříve byla součástí koncovky i verze (.php2, .php3 atd.), což umožňovalo hostingovým službám rozpoznat, jakou verzi PHP interpretu mají použít pro správný běh souboru. V současnosti se už rozšířené koncovky nepoužívají. Kód PHP může být kromě samostatných souborů umístěn i přímo do HTML, kde by ho měl správně nakonfigurovaný HTTP server rozpoznat a předat interpretu ke zpracování. Původně byl jazyk paradigmaticky navržen jen pro tvoření procedurálních skriptů, ale od verze 3 bylo přetvoř PHP přetvořeno na jazyk multi-paradigmatický s přidáním podpory objektově orientovaného programování a později i funkcionálního programování. (48, 49)

V současnosti je PHP nejrozšířenějším serverovým skriptovacím jazykem pro webové aplikace s podílem 82% v květnu 2017. Vděčí za to zejména široké podpoře ze strany webových hostingů, jeho využité v nejrozšířenějším bezplatném CMS, známém pod názvem „Wordpress“ a rozšířenosti backendové platformy LAMP na serverech. (50)

Hello world v PHP

Následující blok kódu zobrazuje standardní demonstrační script zvaný „Hello World”. Na začátku kódu je možné pozorovat řetězec „*<?php”* , který napovídá HTTP serveru, kde začíná PHP kód a umožňuje předání skriptu PHP interpretu. Na konci kódu je zase možné pozorovat řetězec „*?>*“, který znázorňuje, kde php kód končí. Tento ukončující řetězec není nutný, pokud soubor neobsahuje nic jiného než PHP kód. Zobrazený kód může být přesně v této podobě vložen do jakékoliv HTML stránky, a pokud bude mít http server k dispozici interpret, tak bude zpracován. (48)

```php
<?php
  $promenna = "Ahoj, světe!";
  echo $promenna;
?>
```

Výhody PHP

Nejvýraznější předností jazyka PHP je jeho rozmanitost a snadná přístupnost pro začínajícího vývojáře, což je ještě více podpořeno velmi kvalitní dokumentací na webovém portálu php.net, který je spravován organizací Zend Foundation, zodpovědnou za standardizaci PHP a vývoj jeho interpretů. Dalšími výhodami je jeho velká komunita vývojářů PHP, a tedy i velké množství neoficiálních rad tipů, které nejsou v dokumentaci k nalezení a dále také velké množství knihoven, se kterými se dá postavit celá menší aplikace během několika hodin. A nakonec je potřeba zmínit, že PHP je podporované na drtivé většině laciných hostingů, takže každý může svoji PHP aplikaci uveřejnit, aniž by si musel sám konfigurovat produkční prostředí na vlastním serveru. (52)

Nevýhody PHP

Největší zápory PHP vyplývají z jeho největších předností. Možnost snadného vstupu do světa vývojářů PHP způsobuje, že v tomto jazyce programuje mnoho amatérů, kteří pouze skládají úseky ostatních kódu posbíraných z komunitních portálů a dodávají tyto slepené paskvily jako hotové webové aplikace. (52)

Composer

Pro zjednodušení správy a instalace knihoven se v PHP projektech používá multiplatformní balíčkový systém Composer. Za pomoci Composeru je možné v každém projektu definovat na jakých knihovnách a jejich verzích je projekt závislý, načež je pak možné tyto knihovny při sestavování produkčního či vývojového prostředí nainstalovat nebo aktualizovat jediným příkazem. Další významnou funkcí Composeru je jeho schopnost připravit prázdnou šablonu většiny významných PHP frameworků, což značně zjednodušuje vývojářům započetí prací s novým frameworkem, na který ještě nemají vlastní repositář s předpřipravenou šablonou. (53)

3.3.4 PHP Frameworky

Nevýhody PHP zmíněné v předchozí kapitole se snaží řešit řada frameworků, které unifikují strukturu PHP aplikací a implementují moderní architektury webových aplikací jmenovitě například architekturu Model-View-Controller (MVC), která bude podrobněji popsána v následující kapitole. Mezi největší přednosti PHP frameworků patří to, že znatelně zrychlují vývoj webových aplikací a usměrňují vývojáře ke tvoření kódu, který lze opakovaně použít, přináší lepší škálovatelnost. (54)

Nette

Nette framework je nejpopulárnějším frameworkem v České republice a jeho zvládnutí je tedy křest každého českého vývojáře webových aplikací. Jeho vývoj řídí český programátor David Grudl a zaměřuje se především na řešení nízko-úrovňového zabezpečení webových aplikací a pohodlnou implementaci MVC architektury. Framework je založený na tvorbě komponent, které jsou dle potřeby opakovaně využívány, čímž se snižuje míra zbytečně opakovaného kódu a také je k dispozici dokumentace kompletně v češtině, což dává další motivaci českým vývojářům učit se pracovat právě v tomto frameworku. (55)

Laravel

Laravel je relativně nový přídavek do světa PHP frameworků, protože byl poprvé představen až v roce 2011. Tento framework je nejoblíbenější mezi vývojáři globálně a má k dispozici rozsáhlý ekosystém pro jeho hostování. Specifikou Laravelu je silně odlehčený šablonovací systém s názvem „Blade“, důraz na přehlednou syntaxi kódu a usnadnění repetitivních úloh, jako je autentizace, držení uživatelských relací a řazení požadavků do front. Mimo to Laravel nabízí také předpřipravené lokální vývojové prostředí zvané „Homestead“, které usnadňuje vývojářům započetí práce s frameworkem. (54)

Symfony

Symfony je základem pro robustní CMS zvané „Drupal“ a je i závislostí pro výše zmíněný framework Laravel. Tento framework se soustředí hlavně na snadné tvoření formulářů a vývoj formou znovupoužitelných komponent, nicméně na rozdíl od Nette a Laravelu je poněkud obtížnější pro začínající vývojáře. (54)

3.3.5 Architektura Model View Controller (MVC)

Principem architektury MVC je oddělení logiky od prezentace obsahu, což umožňuje vývoj kvalitnějších a škálovatelnějších webových aplikací. Tato architektura je velmi populární u backendových prostředí, zejména u ASP.NET, kde je jeho MVC šablona nejpoužívanější šablonou pro vývoj webové aplikace, ale i u PHP frameworků Laravel, Symfony a Nette, které adoptovaly MVC architekturu již od začátku své existence MVC silně podporuje komponentovou tvorbu software a směruje vývojáře pryč od vytváření takzvaného „špagetového“ kódu, který je špatně udržovatelný. (56, 57)

Model

Model je nejdůležitější komponentou architektury, která se stará o veškerou logiku aplikace a vykonávání dotazů do databáze. Stará se o dynamické chování databáze, správu uživatelů a modelování datových struktur. (57)

View (pohled/zobrazení)

Tato část MVC architektury má na starosti zobrazení výstupu uživateli. Jedná se tedy většinou o HTML šablonu, která podporuje dodatečné funkce, zahrnující iterace větvení a zobrazování proměnných předaných z controlleru. Je zde snaha zachovávat maximální jednoduchost a vyhýbat se jakékoliv logice, neboť ta patří do modelu a controlleru. Zpravidla se pro vytváření View používají šablonovací jazyky, jako například Latte, Yaml nebo Jade. (56)

Controller (Kontroler)

Kontroler podle parametrů v URL pozná jaký obsah má zobrazit a zároveň má k dispozici kontext o tom, který uživatel se snaží obsah zobrazit, a může tedy rozhodnout, zda daný uživatel má mít k požadovanému obsahu přístup. Kontroler také podle URL rozhodne jaký pohled má uživateli zobrazit a jakými daty ho naplnit. (56)

3.4 Objektově orientované programování (OOP)

Objektově orientované programování je programovací paradigma (stejně jako procedurální programování), jehož cílem je vytváření menších a opakovatelně použitelných jednotek kódu – tříd objektů. (1)
Základní stavební jednotkou v objektově orientovaném programování je jsou tzv. „objekty“, které pro vnější prostředí figurují jako uzavřené jednotky, se kterými lze manipulovat pouze za pomoci jejich interních procedur (metod). Tyto procedury jsou ovládány za pomoci veřejného rozhraní, díky kterému lze objekty abstrahovat a je možné objekt užívat i bez znalosti jeho vnitřní struktury.

3.4.1 Specifika objektově orientovaného programování

Objekt je obecně tvořen následujícími součástmi:

- **Vnitřní stav objektu**, který je popsán za pomoci souboru proměnných (tzv. atributů);
- **Vnitřní chování objektu**, které je zastoupeno souborem metod;
- **Protokol práv**, který rozlišuje veřejné součásti objektu od uzavřených.

Třída objektu definuje všechny tyto vnitřní součásti a podle ní je možné vytvářet nové objekty – instance tříd. Každá třída objektů může obsahovat reference na samu sebe nebo na další třídy. (2)

Dědičnost tříd
Jedná se o jednu ze základních vlastností objektově orientovaného programování, která umožňuje tvoření nových datových struktur na základě již existujících. Odvozená třída zdědí veškeré atributy a metody od své rodičovské třídy, což umožňuje vyhnout se tvoření duplikátního kódu a šetří čas programátora. (3) (4)

Abstraktní třídy

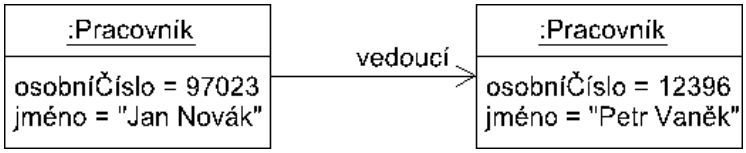
Tento typ třídy je existuje pouze jako předloha k případnému dědění a neexistuje v systému v jeho základní nadefinované podobě. (4)

Polymorfismus

Jedná se o vlastnost objektově orientované programování, která umožňuje využívat jednotné rozhraní pro práci s různými třídami objektů. (4)

3.4.2 Vztahy mezi objekty

Aby bylo možné z objektů vytvořit funkční aplikaci, je důležité, aby mezi sebou definované objekty nějak komunikovali. Toho lze docílit tím, že se vytvoří vazby mezi objekty, které umožňují sdílení metod a posílání zpráv mezi provázanými objekty. V programovacích jazycích se toto chování vytváří za pomoci objektových referencí a ukazatelů. Relace mezi objekty mohou být jednosměrné i obousměrné, což se v objektovém schématu znázorňuje buď šipkou (pro jednosměrné) nebo jednoduchou spojující čarou (pro obousměrné). (3)

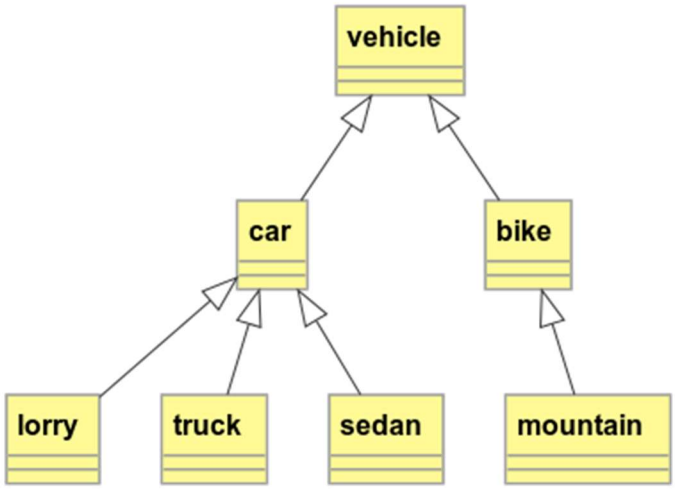


Obrázek 2 - Relace mezi objekty (zdroj: http://www.cs.vsb.cz/)

Vztahy mezi objekty lze rozdělit na následující typy:

- Generalizace
- Klasifikace
- Kompozice
- Agregace

Generalizace

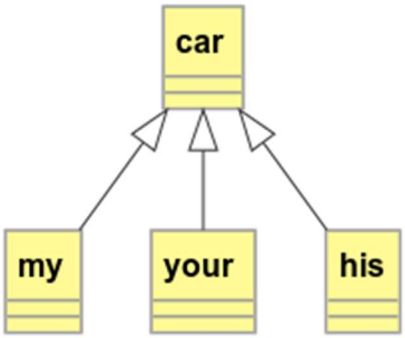


Obrázek 3 – Znázornění generalizace (Zdroj: http://voho.eu/wiki/oop/)

Generalizace neboli zobecnění slouží k potlačení rozdílů existujících mezi jednotlivými třídami a zdůraznění společných vlastností. Opakem generalizace je specifikace. Ideologie generalizace je nejčastěji používána v objektově orientovaném programování jakožto způsob implementace polymorfismu a případné recyklace kódu. Generalizace je často popisována jako „dědičnost“, protože umožňuje vytváření hierarchických vztahů mezi jednotlivými třídami. Z pohledu paradigmatu objektově orientovaného programování je možné, aby každá třída měla neomezený počet předků, ale v praxi je u většiny programovacích jazyků možné, aby třída měla pouze jednoho předka.

Generalizace umožňuje programátorovi využít existující kód bez zbytečné práce navrch nebo rozšířit možnosti předka pro specifický účel. Také je možné změnit přepsat chování metody předka („method override“). (2)

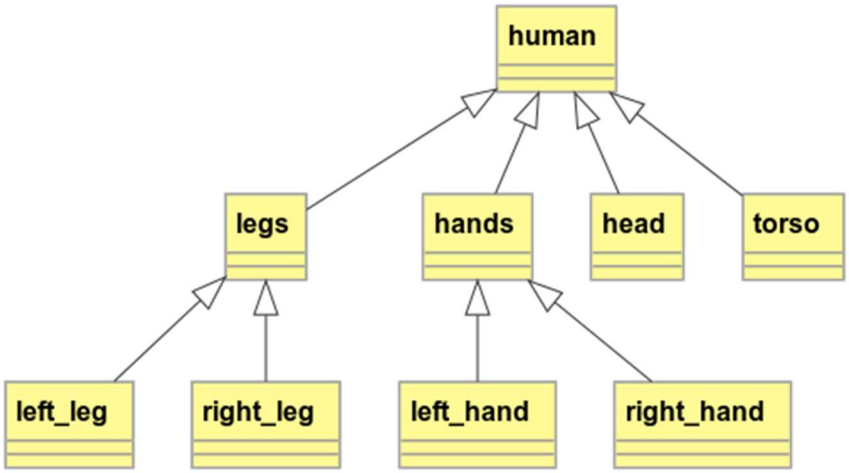
Klasifikace



Obrázek 4 - Znázornění klasifikace (Zdroj: http://voho.eu/wiki/oop/)

Klasifikace je vztah mezi objekty, který potlačuje různorodosti instancí a zdůrazňuje vlastnosti tříd jakožto homogenních útvarů. (2)

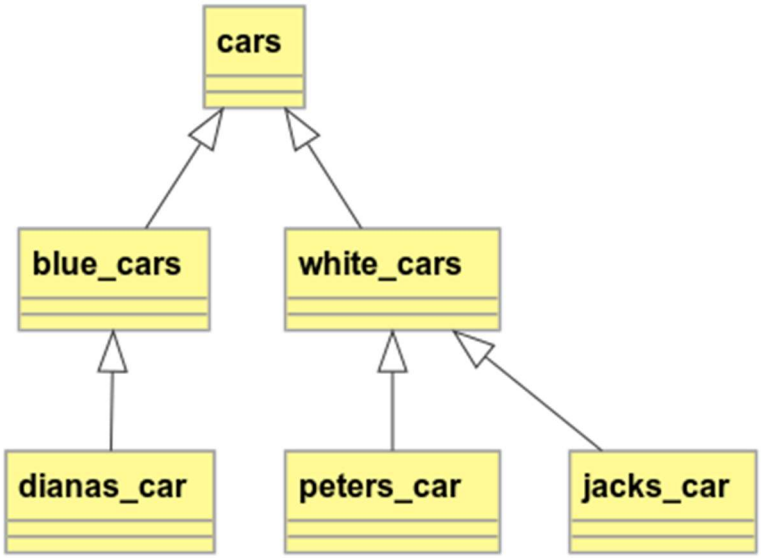
Kompozice



Obrázek 5 - Znázornění kompozice (Zdroj: http://voho.eu/wiki/oop/)

V kompozici jsou potlačovány rysy komponent a zvýrazňovány rysy celku. Jedná se o vztah, ve kterém jsou jednotlivé komponované objekty svázané s právě jedním nadřazeným objektem (např. vztah položky-faktura). Komponované objekty nemohou existovat jakožto samostatné entity. V případně že nadřazený objekt zanikne, zaniknout i komponované objekty. (3) (2)

Agregace

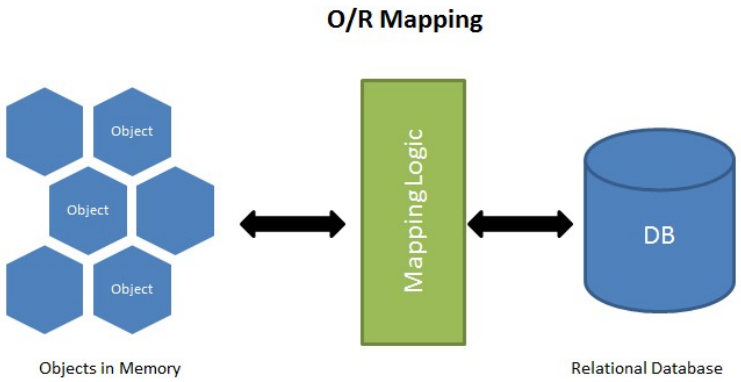


Obrázek 6 - Znázornění agregace (Zdroj: http://voho.eu/wiki/oop/)

V agregaci objektů jsou potlačovány rozdíly mezi jednotlivými objekty a je kladen důraz na agregaci jakožto celek. Opět zde funguje hierarchie nadřazeného objektu a skupiny podřazených objektů, nicméně na rozdíl od kompozice v agregaci podřazené objekty mohou existovat, i když nejsou součástí žádného nadřazeného objektu. Tento vztah lze přirovnat například ke vztahu mezi monitory a počítačem – počítač může mít vícero monitorů, ale monitory mohou fungovat i s jakýmkoliv jiným počítačem, než ke kterému jsou právě připojeny. (2) (3)

3.4.3 Objektově relační mapování (ORM)

Objektově relační mapování je programátorská technika umožňující konverzi mezi relačními a objektovými daty. Většinou je tato idea implementována skrze bezplatné knihovny, které umožňují vývojáři nadefinovat si celou datovou strukturu aplikace, která je následně za pomoci pravidel využitého ORM převedena do SQL struktury a dotazů. Objektově relační mapování funguje jako most mezi objektově orientovaném programováním a relačními tabulkami SQL databázových systémů. Zatímco SQL databázové systémy spoléhají na přísné strukturální schéma, objektově relační mapování umožňuje do tohoto typu datových úložišť dodat některé z výhod NoSQL databází. Při tomto však objektově relační mapování obětuje mnoho efektivity a je tedy považováno za velmi pomalé řešení. Objektové relační mapování se také pyšní tím, že jeho uživatelé musí znát pouze svůj programovací jazyk a nepotřebují vědět jak funguje samotné SQL, které je vrstvami ORM abstrahované, nicméně většinou to není pravda, protože pro složitější aplikace bude muset vývojář vždy vytvářet nějaké vlastní dotazy v proprietální SQL derivaci jeho implementace ORM. (58)



Obrázek 7 - Schéma vztahu mezi OOP a ORM (60)

Doctrine

Doctrine je objektově relační mapovací nástroj pro PHP, který je integrován nebo může výt integrován do většiny PHP frameworků. Za pomoci Doctrine si vývojář může nadefinovat objektové entity, reprezentující veškeré datové struktury v jeho aplikaci, což mu umožní vyvíjet aplikaci v objektově orientovaném PHP, aniž by musel ve většině situací znát a využívat SQL, které se nachází pod objektově relačním mapováním. (59)

4 Vlastní práce

Tato část práce je věnována rozboru vlastní práce autora, která zahrnuje sestavení zadání pro vývoj prototypu systému pracovní evidence, který usnadní zaměstnavateli usnadní plnění povinností vyplývajících ze zákoníku práce. Při sestavování zadání je bráno v potaz, že se jedná o systém pracovní evidence pro akademické prostředí, pro které je legislativa upravena pozměňovacím návrhem zákona 109/8, jenž vstoupí v platnost v červenci roku 2019.

Sestavování zadání je konzultováno s Ing. Pavlem Pánkem, jehož zkušenosti s vedením evidence pracovní doby na Katedře Řízení PEF ČZU vnášejí do práce praktické praktický nadhled.

Po finalizaci zadání přichází návrh vhodného softwarového řešení plnícího zadání v podobě webové aplikace. Pro znázornění návrhu struktury systému je využito popisných diagramů, a pro návrh uživatelského rozhraní je použita metoda papírového prototypování.

Poslední a nejdůležitější fáze praktické části práce je realizace prototypu systému a jeho nasazení na veřejné dostupné testovací prostředí.

Autentizace do systému

Přístup do systému bude realizován za pomoci protokolu LDAP, aby mohli uživatelé využívat již existující přístupové údaje, na které jsou zvyklí z dalších systému provozovaných v jejich instituci. Klíčem pro práci se zaměstnanci v systému pracovní evidence bude identifikátor z jejich instituce, což je v případě PEF ČZU identifikátor UIČ.

4.1 Zadání pro vývoj systému EPD

Aby byl systém použitelný, jakožto softwarový nástroj pro vedení povinné evidence pracovní doby, tak musí realizovat následující úkony:

- Vkládání a editaci záznamů o začátcích a koncích pracovní doby, povinných oddychových přestávek a dalších povinně evidovaných informací.
- Poskytnutí přístupu zaměstnanci k náhledu do jeho pracovní evidence, pokud o to požádá.
- Poskytnutí přístupu k náhledu do celé pracovní evidence příslušnému kontrolnímu úřadu, pokud je to vyžadováno.
- Upozornění pro vedoucího pracovníka, pokud se údaje v evidenci dostanou do rozporu s pravidly stanovenými zákoníkem práce.

Systém bude podporovat správu více individuálních pracovišť zároveň.

Vypuštění elementů netypických pro akademické prostředí

Vzhledem k tomu, že se jedná se jedná o prototyp systému pracovní evidence pro akademické prostředí, budou z něj vypuštěny následující pracovní úkony:

- Pracovní pohotovosti;
- Práce v noci;
- Bezpečnostní přestávky;
- Práce přesčas.

Tyto úkony vedoucí pracovník zaměstnancům v akademickém prostředí typicky nerozvrhuje. Pokud k těmto úkonům dochází, tak se tak stává pouze při pracovních úkonech rozvržených samotným zaměstnancem, které podle novely zákona dříve zmíněné novely zákona není potřeba evidovat. Systém tedy bude evidovat pouze následující úkony:

- Pracovní doba rozvržená zaměstnavatelem;
- Povinná oddychová přestávka.

4.1.1 Uživatelské role systému

Systém bude obsahovat následující uživatelské role:

- Administrátor systému pracovní evidence;
- Vedoucí pracoviště;
- Asistent vedoucího pracoviště;
- Zaměstnanec;
- Pracovník pověřený kontrolou pracovní evidence.

Administrátor systému pracovní evidence má následující oprávnění:

- Vytvářet v systému nová pracoviště;
- Přiřazovat vedoucí pracovišť k již existujícím pracovištím;
- Provádět exporty databáze pracovní evidence;
- Může zároveň zastávat další uživatelskou roli s výjimkou role kontrolního pracovníka.

Vedoucí pracoviště může provádět následující úkony:

- Spravovat agendu zaměstnanců svého pracoviště;
- Spravovat pracovní evidenci zaměstnanců pracoviště;
- Vytváření přístupu pro pracovníka pověřeného kontrolou pracovní evidence;
- Jmenování asistentů vedoucího pracoviště z řad svých podřízených.

Asistent vedoucího pracoviště může vykonávat následující zásahy:

- Spravovat agendu zaměstnanců svého pracoviště;
- Spravovat pracovní evidenci zaměstnanců svého pracoviště.

Zaměstnanec smí:

- Nahlížet do své pracovní evidence.

Pracovník pověřený kontrolou pracovní evidence smí:

- Nahlížet do pracovní evidence všech zaměstnanců pracoviště.

4.1.2 Komponenty systému

Systém se bude skládat z následujících komponent:

- Konzole administrátora systému;
- Agenda zaměstnanců pracoviště a archiv bývalých zaměstnanců;
- Agenda asistentů vedoucího pracoviště;
- Náhled na záznamy pracovní evidence (měsíční a týdenní);
- Nástroj pro ruční vkládání a odebírání záznamů;
- Nástroj pro importování rozvrhu zaměstnance;
- Nástroj pro vytvoření přístupu kontrolnímu pracovníkovi.

Konzole administrátora systému

V této komponentě se bude nacházet seznam vedených pracovišť a formulář pro přidání nového pracoviště. V seznamu pracovišť budou vypsána jména přiřazených vedoucích pracovišť a bude přítomno tlačítko pro jejich odebrání. Pokud je vedoucí pracoviště odebrán, tak jsou odebráni i jeho asistenti. V případě, že pracoviště nebude mít vedoucího pracoviště, se místo tlačítka odebrání zobrazí tlačítko pro přidání, které vyvolá formulář pro přidání vedoucí pracovníka.

Netřeba dodávat, že tato komponenta bude k dispozici pouze *administrátoru systému pracovní evidence*.

Agenda zaměstnanců pracoviště

Zde se bude nacházet seznam zaměstnanců pracoviště, formulář pro přidání nového zaměstnance do agendy a odkaz na archiv, ve kterém jsou k nalezení bývalí zaměstnanci, kteří již nejsou vedení v hlavní agendě. U každého ze zaměstnanců bude přítomná možnost odebrání (přesunutí do archivu), odkaz na zobrazení jeho pracovní evidence a centrum upozornění. Centrum upozornění bude prázdné, pokud evidence zaměstnance bude v souladu se zákoníkem práce. V opačném případě se začervení a nabídne zobrazení systémem detekovaných rozporů se zákoníkem práce.

Tato komponenta bude v plné funkčnosti k přístupná *vedoucí pracovišť* a *asistenty vedoucích pracovišť*. V omezené podobě, oproštěné o všechny funkce s výjimkou odkazů na

evidence jednotlivých zaměstnanců bude tato komponenta zpřístupněna i *pracovníku pověřenému kontrolou pracovní evidence*, kterému bude sloužit jako rozcestník.

Agenda asistentů vedoucího pracoviště

Tato komponenta se bude skládat ze seznamu asistentů vedoucího pracoviště a formuláře pro přidání nového asistenta z řad osob vedených v agendě zaměstnanců pracoviště. Seznam asistentů bude obsahovat tlačítko pro odebrání asistenta u každé položky.

Tato komponenta bude k dispozici pouze *vedoucím pracovišť*.

Náhled na záznamy pracovní evidence

Náhled na záznamy pracovní evidence bude zpracován hierarchicky podle časové perspektivy. Náhledy záznamů pracovní evidence budou k dispozici v měsíční a týdenní formě. Při zobrazení záznamů zaměstnance se nejdříve zobrazí měsíční náhled aktuálního měsíce a roku, ve kterém bude možné přepínat mezi měsíci a roky. V náhledu budou zobrazeny souhrnné součty a upozornění za rok, měsíc, jednotlivé týdny a dny. Každý týden bude obsahovat odkaz pro zobrazení týdenního náhledu. V týdenním náhledu budou k dispozici souhrnné součty a upozornění za rok, týden a jednotlivé dny. Každý den bude znázorněn časovými osami pro pracovní výkony a přestávky a u každého dnu budou vypsána veškerá upozornění, která se ho týkají v plném znění.

Tato komponenta bude přístupná všem uživatelům systému pracovní evidence s tím rozdílem, že každý do ní bude jinak přistupovat. *Zaměstnancům* bude sloužit jako forma náhledu na jeho záznamy, *kontrolním pracovníci* přes ní budou kontrolovat dodržování zákoníku práce a pro *vedoucí pracovišť* a *asistenty vedoucích pracovišť* bude náhled sloužit jako forma zobrazení přehledu pracovní evidence jimi spravovaných zaměstnanců. Nástroje pro vkládání a odebírání záznamů budou k dispozici pouze pro *vedoucí pracovišť* a *asistenty vedoucích pracovišť*.

Nástroj pro ruční vkládání a odebírání záznamů

Odebírání záznamů bude implementováno formou nabídky při najetí myši na záznam v časové ose denního náhledu. Pro přidávání záznamu bude sloužit formulář v denním náhledu, který bude obsahovat pole pro začátek a konec pracovní činnosti. Při vkládání

systém ověří, zda nedochází k překrytí pracovní doby, a pokud tomu nastane, tak bude uživatel upozorněn a vložení bude zamítnuto.

Dále bude k dispozici nástroj pro vložení povinné přestávky, které budou moci být vloženy v případě potřeby vkládány na separátní časovou osu v případě, že doba výkonu práce přesáhne šest hodin. Pokud bude vložena pracovní činnost delší než šest hodin a nebude pro tento časový úsek vložena povinná přestávka, tak budou *asistenti vedoucího pracovišť* a *vedoucí pracoviště* upozorněni v centru upozornění v agendě zaměstnanců a v denním náhledu.

Tyto nástroje budou k dispozici pouze *vedoucím pracovišť* a *asistentům vedoucích pracovišť*.

Nástroj pro import rozvrhu zaměstnance

Pro zmenšení nutného množství interakce se systém bude k dispozici nástroj pro import rozvrhu zaměstnance, který u akademického pracovníka bude tvořit většinu jeho rozvržené pracovní činnosti. Tato komponenta bude muset být snadno upravitelná, kvůli rozdílným formám a formátům vstupních dat jednotlivých pracovišť či institucí. V prototypu bude implementována tak, aby přijímala CSV exporty osobních rozvrhů z UIS ČZU.

Tato komponenta bude přístupná v agendě zaměstnanců pro *vedoucí pracovišť* a *asistenty vedoucích pracovišť*.

Nástroj pro vytvoření přístupu kontrolnímu pracovníkovi

Tento nástroj sloužit k vytvoření dočasného přístupu externí osobě pověřené kontrolou pracovní evidence a dodržování zákoníku práce. Bude tak možno učinit skrze formulář přístupný v agendě zaměstnanců, který bude obsahovat vstup pro email, začátek platnosti přístupu a konec přístupu, aby bylo možné přístup omezit pro dobu požadovanou příslušným kontrolním úřadem. Kontrolní pracovník se bude přihlašovat zadaným emailem a heslem, které mu na tento email bude zasláno.

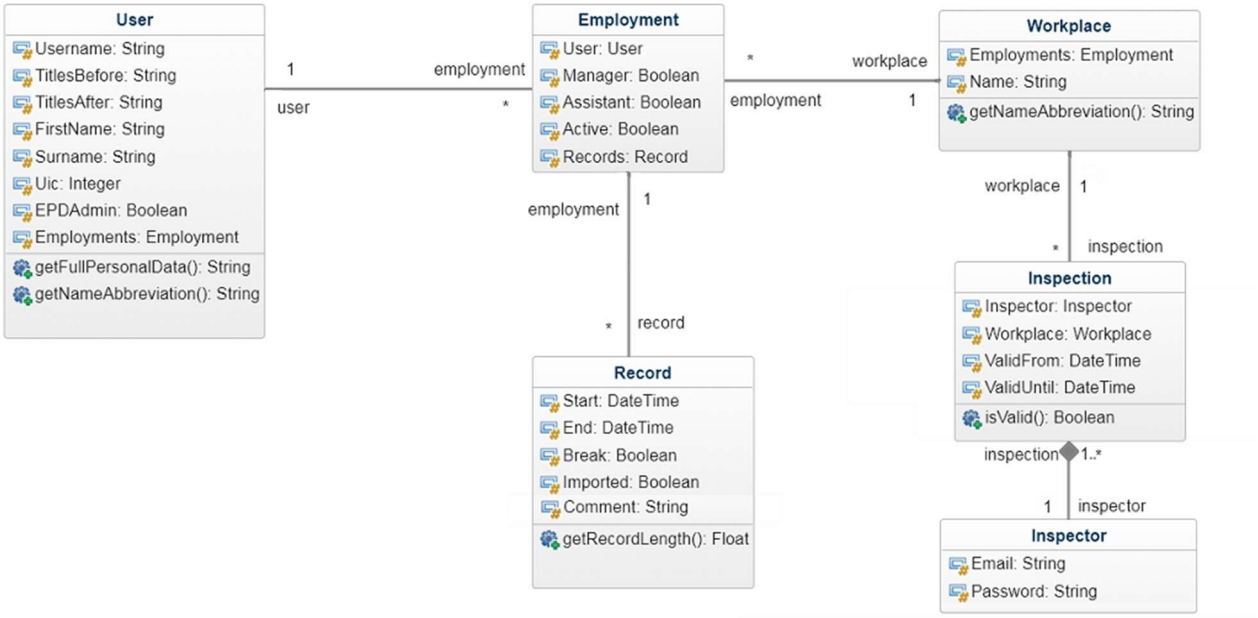
Formulář bude přístupný pouze *vedoucím pracovišť*.

4.2 Návrh systému

Pro návrh systému, který je schopen řešit zadání popsané v přechozí kapitole bylo využito prototypování s použitím metody „drátěných prototypů“ (tzv. Wireframů). Pro popis systému bylo využito UML, podle kterého byl vytvořen diagram tří a USE-CASE diagram. Důvodem pro vytvoření právě těchto diagramů je to, že tyto diagramy nejlépe popisují vizualizují strukturu systému. Diagram aktivit byl vypuštěn, protože v systému nejsou procesy dostatečně složité na to, aby jejich vizualizace měla smysl.

4.2.1 Diagram tříd

Diagram tříd byl vytvořen za pomoci webového zkušební verze nástroje GenMyModel dostupného pod adresou www.genmymodel.com/. Diagram tříd neznázorňuje datové struktury využívané pouze pro interní procesy systému, jako jsou záznamy aktivit uživatelů a záznamy rozesílaných emailů.



Obrázek 8 - Diagram tříd (vlastní zpracování)

4.2.2 Datový slovník

Tabulka 3 - Třída "User"

Atribut	Typ	Popis
Username	Varchar	UIS login
TitlesBefore	Varchar	Tituly před jménem
TitlesAfter	Varchar	Tituly za jménem
FirstName	Varchar	Křestní jméno
Surname	Varchar	Příjmení
Uic	Integer	UIČ uživatele
EPDAdmin	Boolean	Uživatel je administrátor

Tabulka 4 - Třída "Employment"

Atribut	Typ	Popis
Manager	Boolean	Je vedoucím pracoviště
Assistant	Boolean	Je asistentem vedoucího
Active	Boolean	Zaměstnanec není archivní

Tabulka 5 - Třída "Record"

Atribut	Typ	Popis
Start	Datetime	Začátek aktivity
End	Datetime	Konec aktivity
Break	Boolean	Záznam je přestávka
Imported	Boolean	Záznam byl importován
Comment	Varchar	Komentář k záznamu

Tabulka 6 - Třída "Workplace"

Atribut	Typ	Popis
Name	Varchar	Název pracoviště

Tabulka 7 - Třída "Inspection"

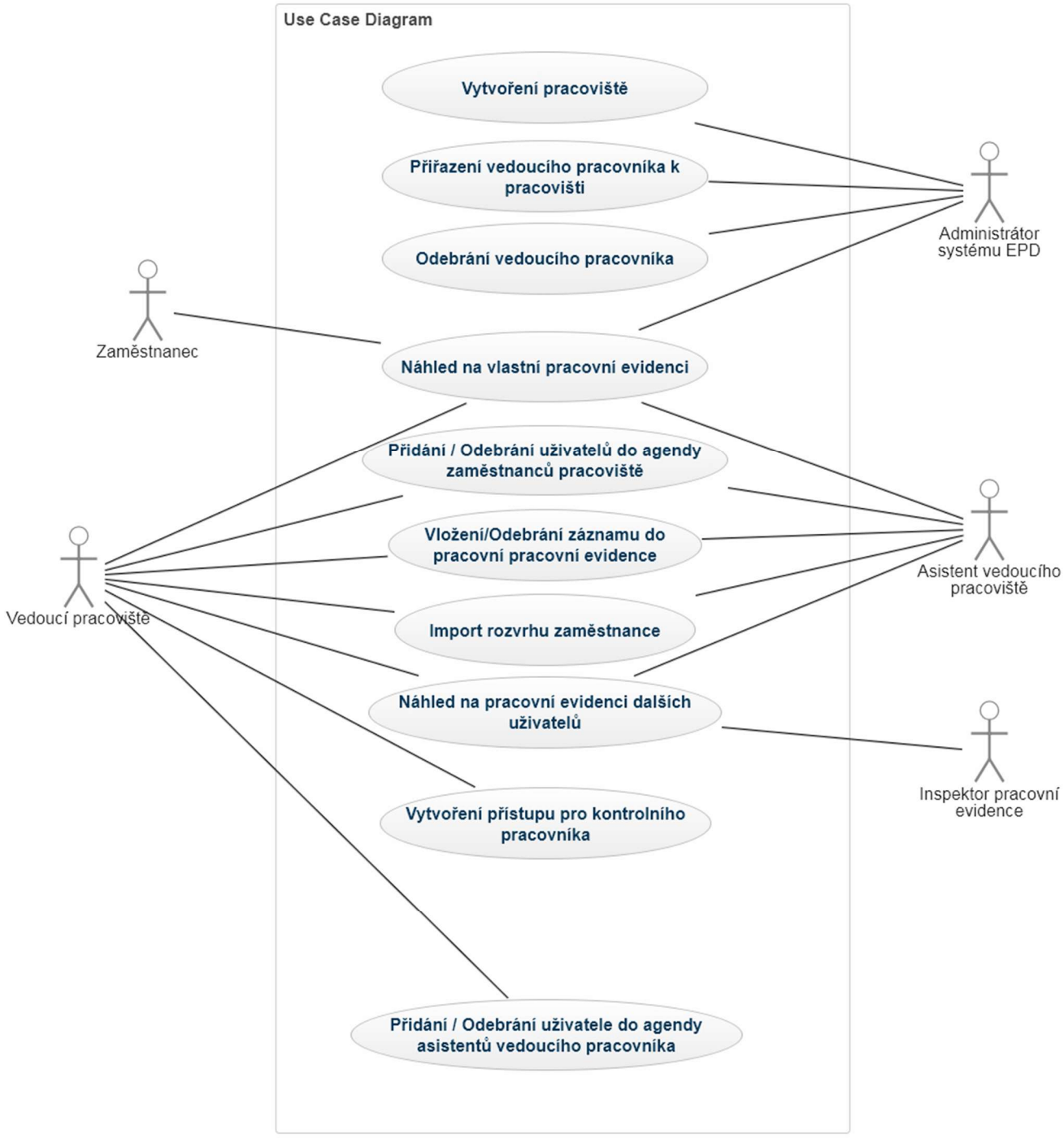
Atribut	Typ	Popis
ValidFrom	Datetime	Začátek platnosti přístupu
ValidUntil	Datetime	Konec platnosti přístupu

Tabulka 8 - Třída "Inspektor"

Atribut	Typ	Popis
Email	Varchar	Email inspektora, login
Password	Varchar	Heslo inspektora

4.2.3 Use-Case Diagram

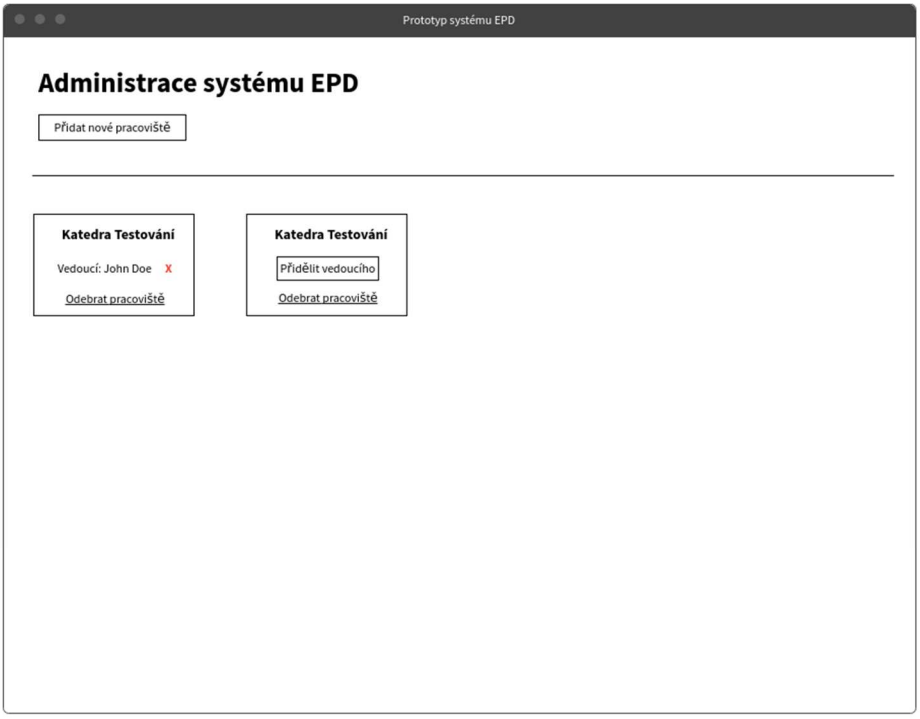
Diagram tříd byl vytvořen za pomoci webového zkušební verze nástroje GenMyModel dostupného pod adresou www.genmymodel.com/.



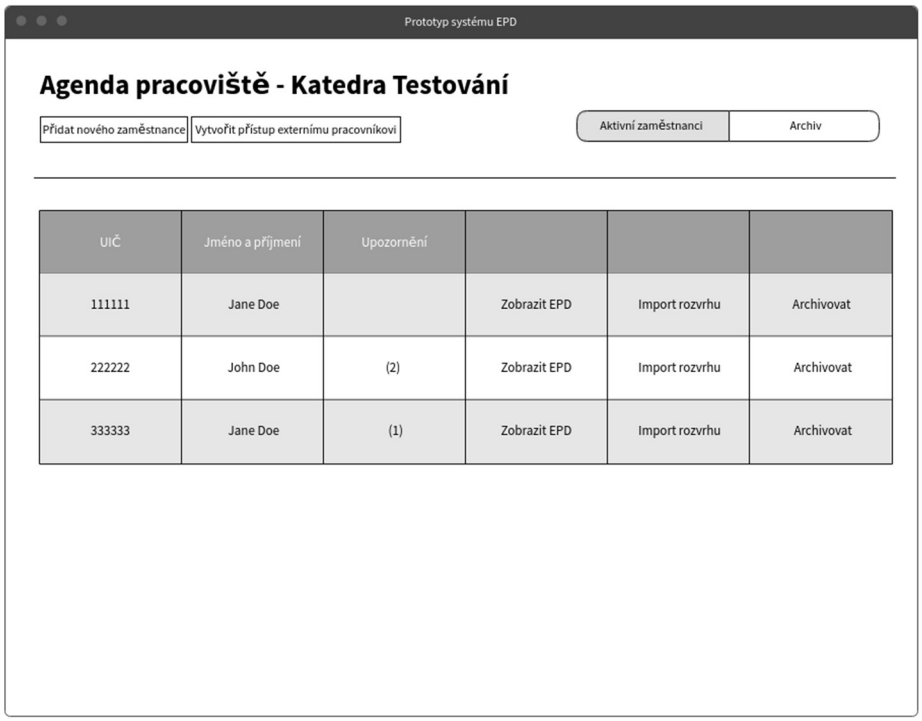
Obrázek 9 - Use-Case Diagram (vlastní zpracování)

4.2.4 Drátěné prototypy (Wireframy)

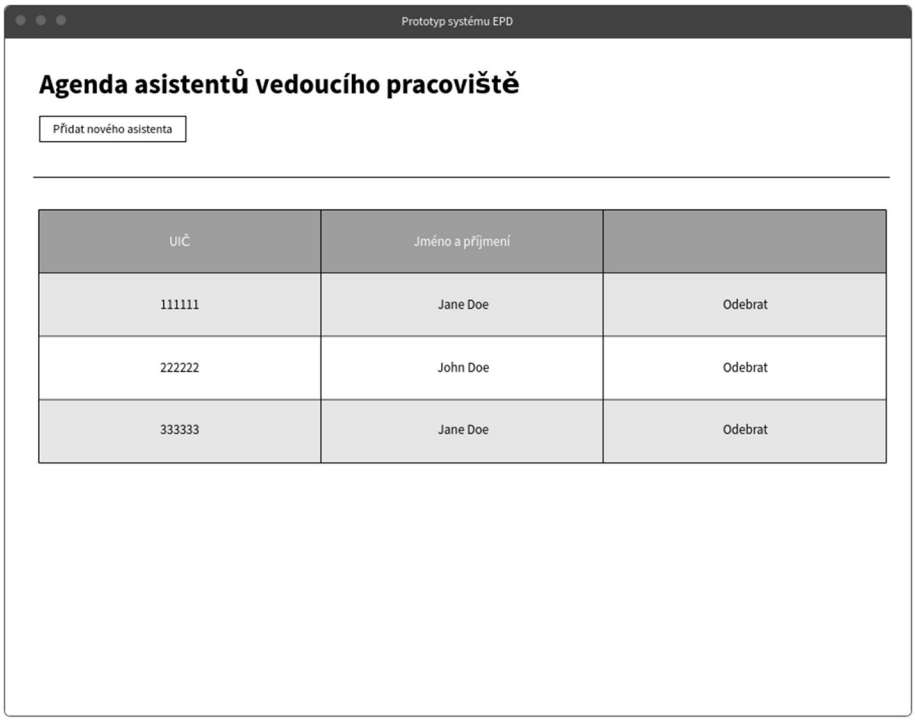
Drátěné prototypy systému pro evidenci pracovní doby byly vytvořeny za pomoci zkušební verze webového nástroje MockFlow dostupného na adrese https://www.mockflow.com/. Drátěné prototypy vizualizují komponenty systému popsané v zadání pro tvorbu systému. Kapitola 5.1



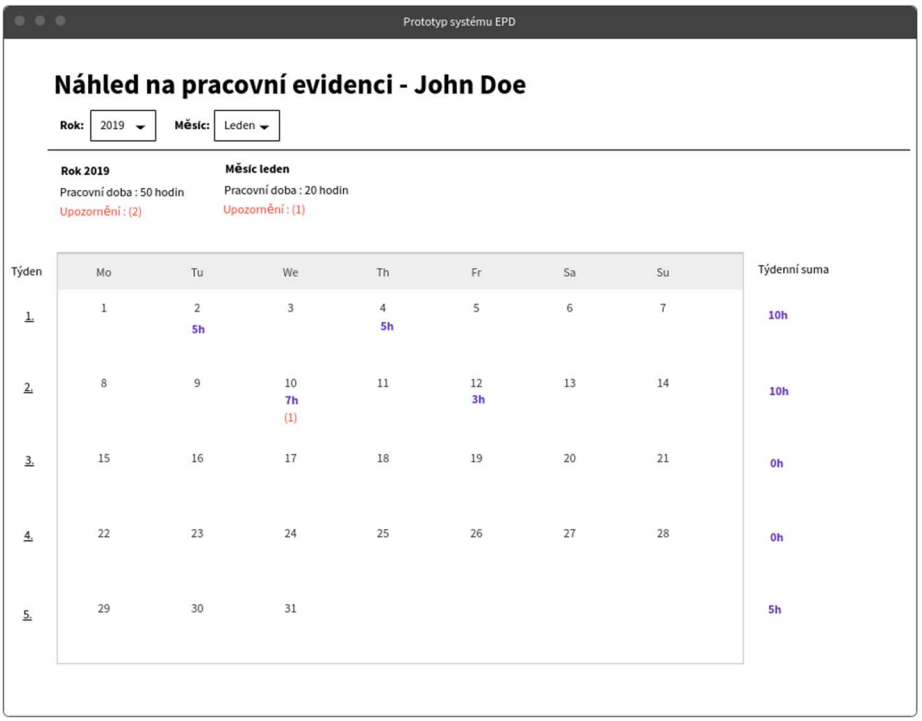
Obrázek 10 - Konzole administrátora systému (vlastní zpracování)



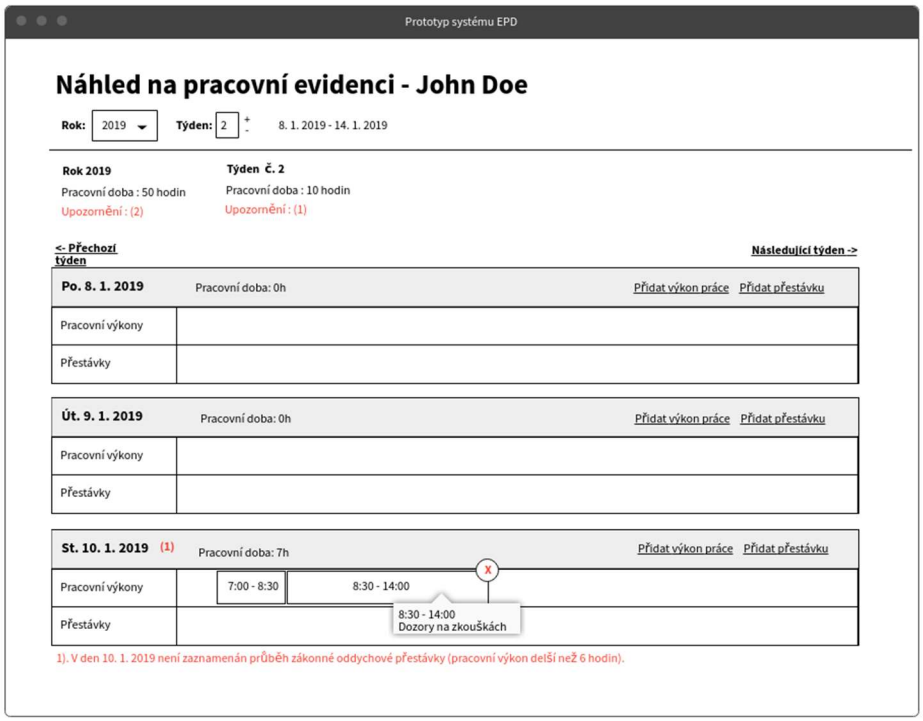
Obrázek 11 - Agenda zaměstnanců pracoviště a archiv bývalých zaměstnanců (vlastní zpracování)



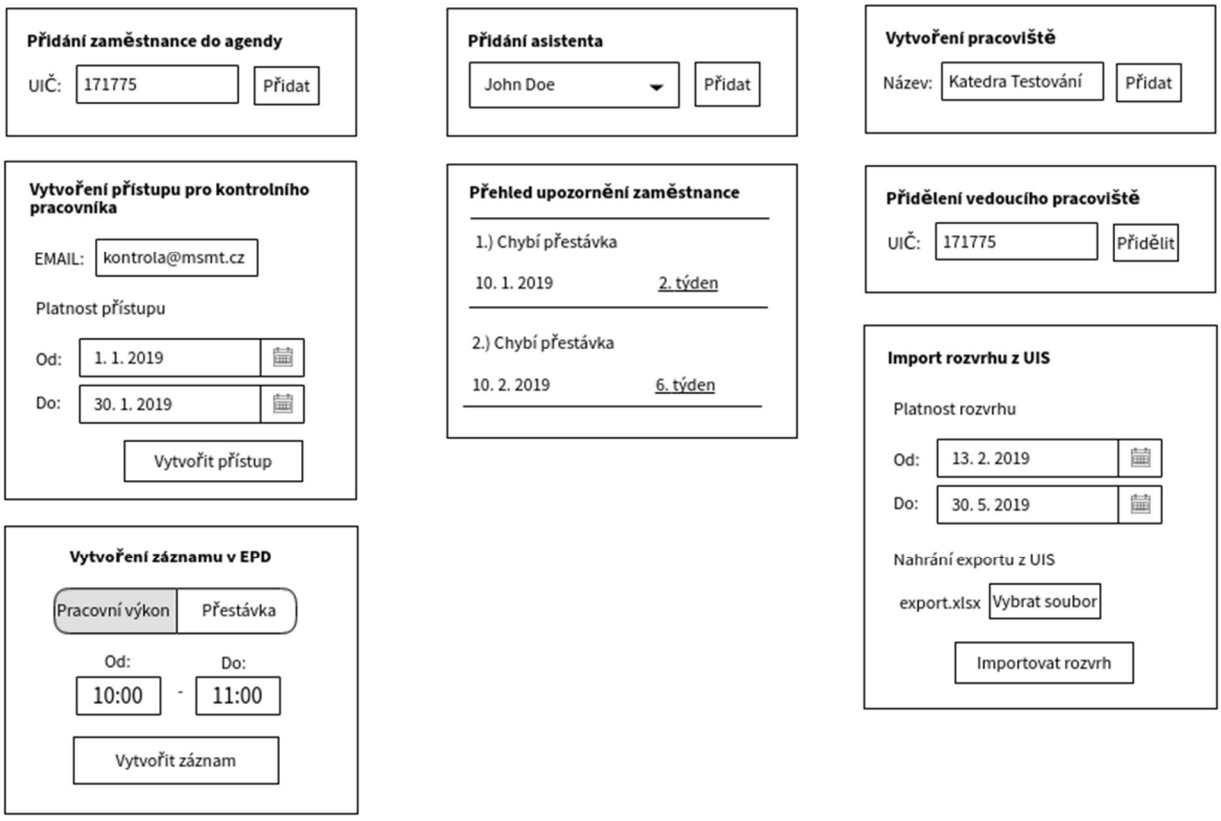
Obrázek 12 - Agenda asistentů vedoucího pracoviště (vlastní zpracování)



Obrázek 13 - Náhled na záznamy pracovní evidence - měsíční náhled (vlastní zpracování)



Obrázek 14 - Náhled na záznamy pracovní evidence - týdenní náhled (vlastní zpracování)



Obrázek 15 - Formuláře a sub-komponenty (vlastní zpracování)

4.3 Realizace funkčního prototypu systému

Na základě zadání z kapitoly 4.1, architektonických návrhů a drátěných prototypů z kapitoly 4.2 je vytvořen funkční prototyp systému pro evidenci pracovní doby. Aplikace je nazývána prototypem, protože nezahrnuje specifické povinnosti a nadstavby, které vyplývají z kolektivních a smluv a potřeb konkrétních pracovišť. Samozřejmě by nebyl problém implementovat tato specifika v kontextu České Zemědělské Univerzity, nicméně to by z aplikace učinilo řešení určené pro jednu konkrétní instituci, zatímco tato práce se soustředí na nalezení obecného řešení problematiky, a nikoliv na dodání hotového řešení pro Českou Zemědělskou Univerzitu. Vytvořený prototyp lze tedy považovat za 90 procent kompletního systému pro evidenci pracovní doby, který může být snadno dokončen pro naplnění potřeb specifické instituce.

Prototyp systému v byl vyvíjen vodopádovým vývojovým procesem, který je zakončen v průběhu fáze verifikace. Vodopádový model byl zvolen, protože jeho charakteristiky odpovídají potřebám procesu tvoření diplomové práce navzdory tomu, že při vývoji prototypu bylo aplikováno paradigma „Code First“, které je typické pro agilní vývoj. Pro navazující vývoj za účelem nasazení aplikace pro konkrétní instituci jsou tedy doporučovány agilní metody vývoje.

4.3.1 Použité technologie

Při vývoji prototypu systému pro evidenci pracovní doby byly využity následující technologie a nástroje:

- PHP/Nette/Composer;
- MySQL/ Doctrine ORM;
- SCSS;
- Zurb Foundation;
- Gulp/Node.js./NPM;
- Google Charts;
- JavaScript/jQuery/AJAX.

PHP\Nette\Composer

Pro vývoj backendového prostředí systému byl využit programovací jazyk PHP a framework Nette. Architektonicky je backendové prostředí postavené na principech návrhového vzoru Model-View-Controller (MVC), kterému se v případě nasazení Nette frameworku říká také Model-View-Presenter (MVP) Pro import rozvrhů exportovaných z UIS bylo ještě potřeba doinstalovat balíček PhpOffice, k čemuž byl použit balíčkovací systém Composer.

MySQL/Doctrine ORM

Jako datové úložiště pro prototyp systému bylo použito MySQL, ale vzhledem k využití abstrakční vrstvy Doctrine je možné použít jakýkoliv druh databázového úložiště, který je podporován ze strany Doctrine Database Abstraction Layer. Samotný systém vůbec nepracuje s databázovým úložištěm, ale pouze s objektovým modelem zmapovaným za pomoci Doctrine ORM. Entity mapované v Doctrine ORM zahrnují kromě entit popsaných v diagramu tříd ještě další „servisní“ entity, jako jsou záznamy odesílaných emailů a uživatelských aktivit v systému. Při práci s ORM bylo postupováno podle paradigmatu nazývaného „Code First“.

SCSS (SASS)

Pro tvoření definic kaskádových stylů je ve zdrojových souborech systému využit preprocesor SASS s využitím syntaxe SCSS. Pro kompilaci SCSS do CSS je vytvořen automatizovaný proces pro Gulp, aby bylo možné styly při vývoji kompilovat v reálném čase.

Zurb Foundation

Na implementaci základní responzivnosti a zjednodušení vývoje front-endu bylo využito frameworku Zurb Foundation. Zurb Foundation bylo zvoleno především proto, že jeho zdrojové soubory jsou psané v syntaxi SCSS a bylo tedy možné tento framework snadno zakomponovat do projektu vyvíjeného systému bez instalace dalších prerekvizit.

Z Frameworku je v systému také na mnoha místech využita funkce „Reveal“, která implementuje vyskakovací okna s formuláři.

Gulp/Node.js/NPM

Pro automatizaci repetitivních procesů vývoje front-endu je využito nástroje Gulp, s jehož konfiguračním souboru (gulpfile.js/gulpfile.babel.js) jsou nadefinované funkce pro automatické zkompilování Sassu při změně zdrojových souborů, funkce pro skládání JavaScriptových knihoven a pro automatizovanou komprimaci obrázků. Všechny nástroje, které jsou v Gulp procesech využívány jsou instalovány za pomoci balíčkového systému NPM a určené pro spouštění v prostředí Node.js. Závislosti jsou definované v souboru „package.json“ umístěném v kořenovém adresáři projektu.

Google Charts

Znázornění časové osy pracovního dne a vyznačení záznamů pracovní evidence je realizováno s využitím modulu „Timeline“ z API Google Charts.

JavaScript/jQuery/AJAX

V prototypu systému je využito čistého JavaScriptu pro práci s API Google Charts a jQuery pro zasílání AJAX dotazů tam, kde je jejich využití vhodné a přípravu formulářů. Všechny ostatní JavaScriptové funkcionality jsou zajištěny za pomoci frameworku Zurb Foundation.

4.3.2 Struktura projektu systému

Pro zlepšení udržovatelnosti kódu byly důležité části projektu rozděleny do modulů, které obstarávají jednotlivé funkce a výstupy systému. Všechny presentéry uvnitř modulů dědí některý z abstraktních presentéry z jmenného prostoru App\Presenters a všechny šablony uvnitř modulů jsou obalovány některým z layoutů umístěných ve jmenném prostoru App\Templates.

Seznam modulů:

- Administration (App\AdministrationModule);
- Agenda (App\AgendaModule);
- Dashboard (App\DashboardModule);
- Files (App\FilesModule);
- Inspection (App\InspectionModule);
- Records (App\RecordsModule);
- User (App\UserModule).

Každý modul je nadále pro přehlednost pořádek rozdělen na následující adresáře:

- Controls;
- Entities;
- Presenters.

Adresář „Controls“

Tento adresář obsahuje veškeré formuláře vztahující se k danému modulu. Formuláře jsou implementovány jako potomci třídy Nette\Application\UI\Control a skládají se ze šablony formuláře, třídy s definovanými vlastnostmi formuláře a „továrny“, která se stará o injekci závislostí a propojení komponenty formuláře s presentéry.

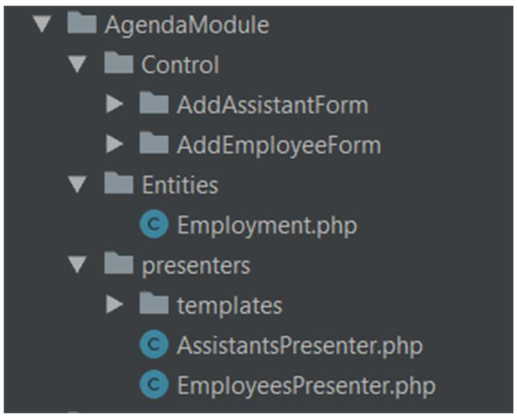
Adresář “Entities”

Účelem tohoto adresáře je uchovávání entit Doctrine ORM, které jsou logicky sdružené s daným modulem. Veškeré třídy v tomto adresáři by se měly nacházet ve jmenném prostoru „App\Entity“.

Adresář „presenters“

Zde se nacházejí všechny presentéry patřící do daného modulu. Veškeré presentéry, které pro spuštění vyžadují přihlášení uživatele jsou potomky abstraktní třídy „App\Presenters\SignedPresenter“. Presentéry, které pro spuštění přihlášení uživatele nevyžadují jsou potomky abstraktní třídy „App\Presenters\BasePresenter“. Veškeré třídy v tomto adresáři by se měly nacházet ve jmenném prostoru daného modulu.

Obrázek 16 - Struktura individuálních modulů

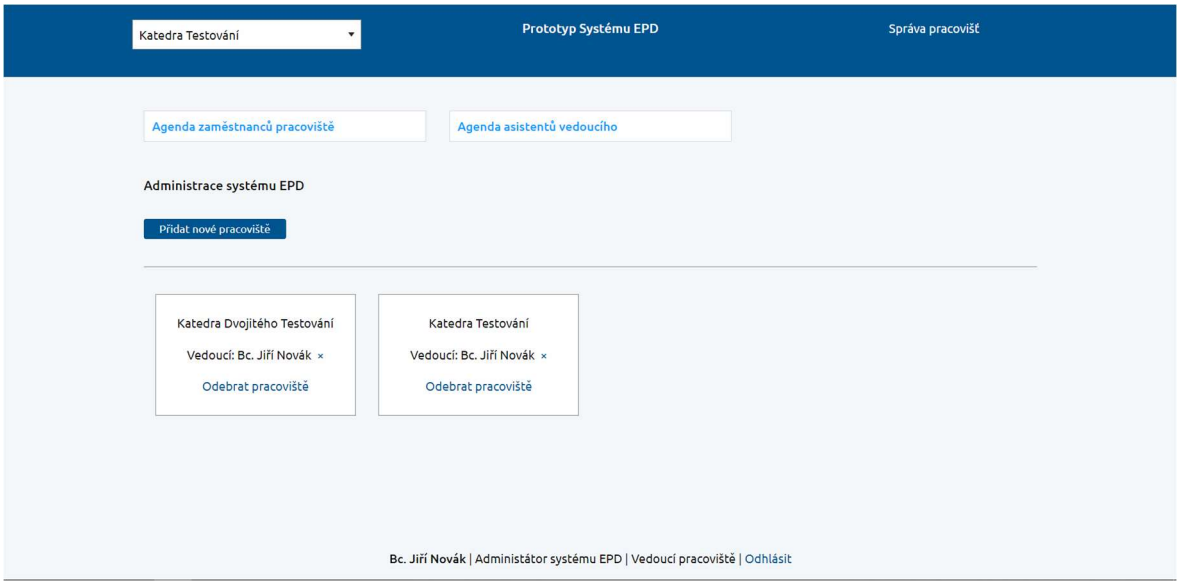


Zdrojové soubory front-endu

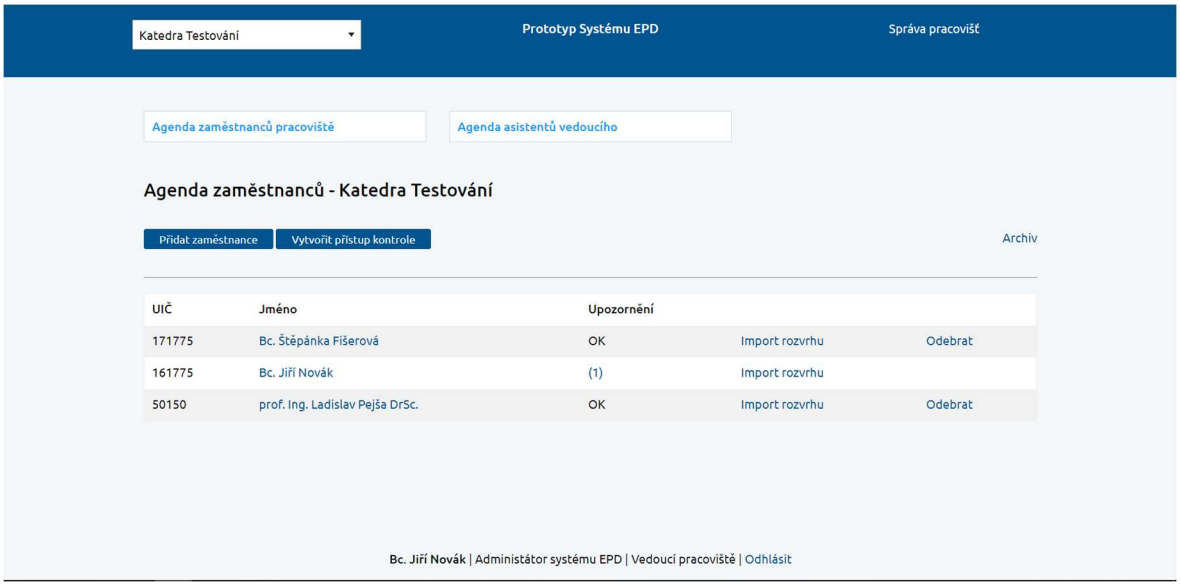
Zdrojové soubory stylů, javascriptu a obrázků se nacházejí v adresáři App\webassets.

4.3.3 Výsledná podoba prototypu systému

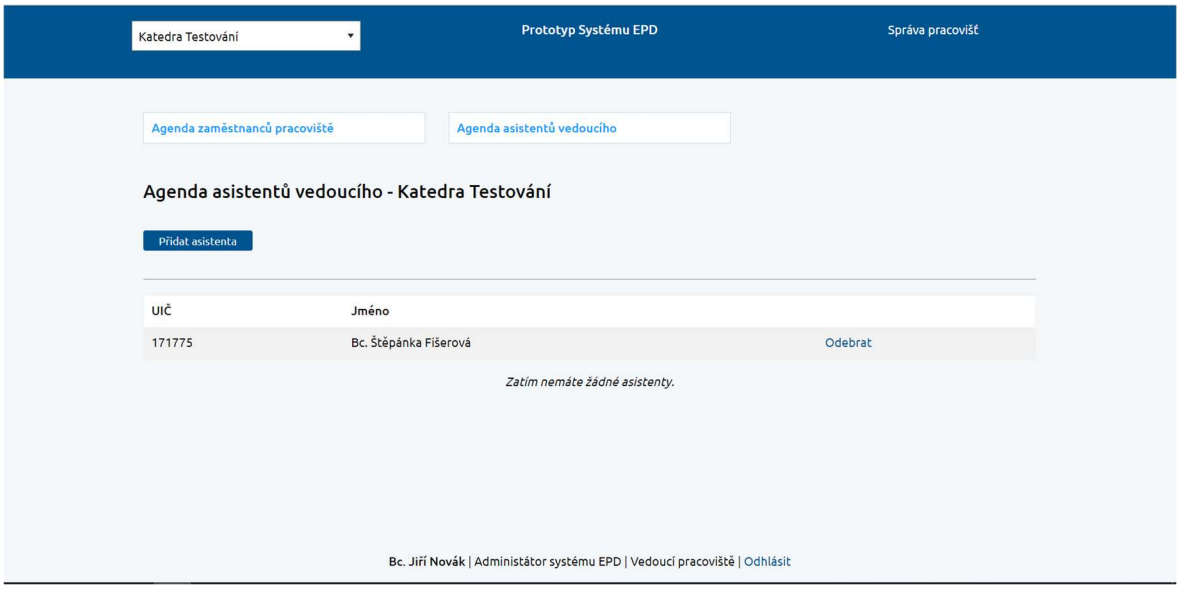
V této kapitole jsou k dispozici k nahlédnutí snímky obrazovky zobrazující finální podobu vyvinutého prototypu systému pro vedení evidence pracovní doby.



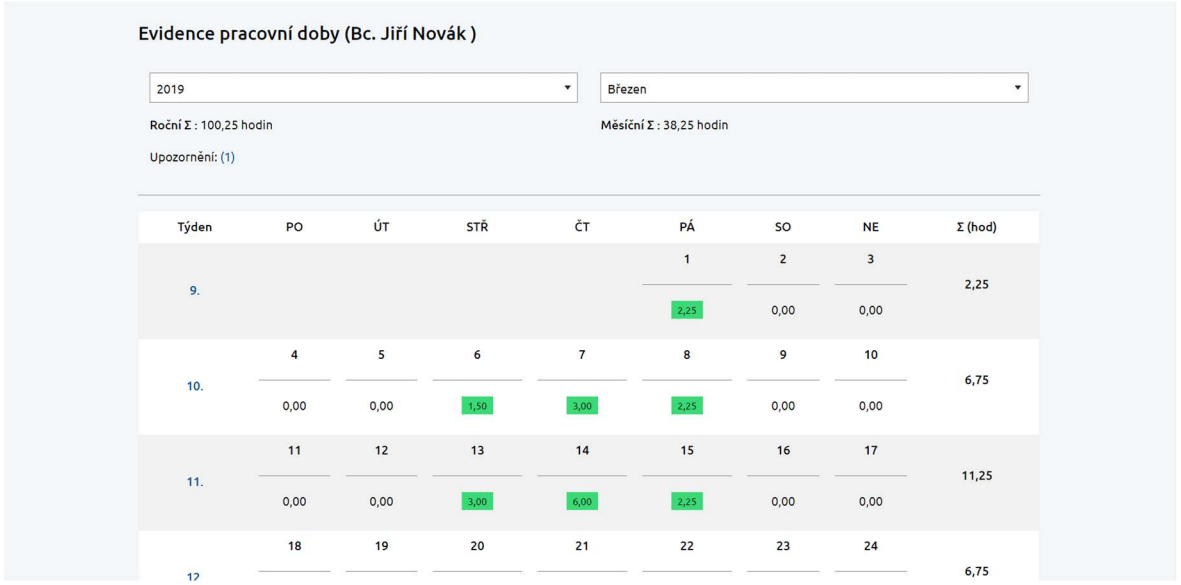
Obrázek 17 - Screenshot, Administrace systému EPD (vlastní zpracování)



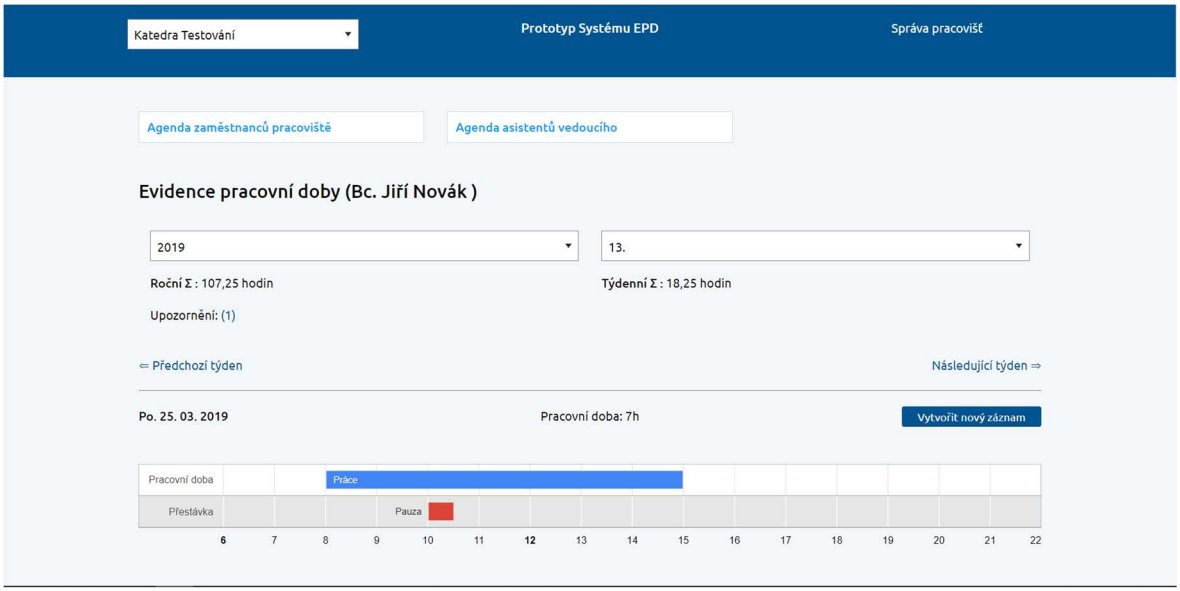
Obrázek 18 - Screenshot, Agenda zaměstnanců (vlastní zpracování)



Obrázek 19 - Screenshot, Agenda asistentů vedoucího pracovníka (vlastní zpracování)



Obrázek 20 - Screenshot, Náhled na pracovní evidenci - měsíční zobrazení (vlastní zpracování)



Obrázek 21 – Screenshot, Náhled na pracovní evidenci - týdenní zobrazení (vlastní zpracování)

4.3.4 Nasazení systému

Pro úspěšné nasazení systému do nového prostředí je možné buď využít autorem připravené provozní prostředí, anebo vytvořit si vlastní provozní prostředí, které splňuje provozní požadavky systému.

Autorem připravené provozní prostředí

Autorem připravené provozní prostředí je dostupné buď jako virtuální stroj pro nástroj VirtualBox, anebo jako obraz disku ve formátu RAW pro nasazení na fyzický stroj. Jedná se o linuxové prostředí založené na distribuci CentOS 7, na kterém jsou nainstalovány všechny potřebné prerekvizity a nasazen systém ve verzi aktuální k datu dokončení této diplomové práce.

Virtuální stroj by měl splňovat následující specifikace:

- 1 CPU;
- >512 MB operační paměti;
- 10 GB úložného prostoru.

Virtuální stroj a obraz disku ve formátu RAW jsou k dispozici na následujícím adrese: https://bit.ly/2Xx70Hw

Vytvoření vlastního prostředí

Pro vytvoření vlastního provozního prostředí je možné využít jakýkoliv operační systém, na který lze nainstalovat následující prerekvizity:

- PHP ^7.1 a Composer
- Node.js ^6.11.2 a NPM
- Libovolné databázové úložiště podporované ze strany Doctrine DBAL
- Nette ~2.4
- PhpOffice
- Libovolný HTTP server

5 Výsledky práce

Tato kapitola je věnována souhrnu výsledků dosažených v jednotlivých částech práce a následnému představení konkrétních dílčích výsledků.

Výsledky teoretické části práce

V teoretické části této diplomové práce byla provedena analýza legislativy spojené s pracovní evidencí, čehož bylo následně využito v praktické části této práce v rámci sestavování zadání pro vývoj prototypu systému evidence pracovní doby.

Následně byly prozkoumány moderní metody vývoje webových aplikací, aby bylo možné zvolit vhodný postup pro vývoj již zmíněného prototypu, za pomoci zvážení kladů a záporů popsaných technologií a jejich možných přínosů pro co nejefektivnější splnění formulovaného zadání.

Výsledky praktické části práce

V praktické části této diplomové práce bylo využito poznatků získaných v teoretické části pro vytvoření a formulaci zadání informačního systému, který je schopen adresovat problematiku evidence pracovní doby v kontextu akademického prostředí.

Systém byl navržen tak, aby byl připraven na změny v legislativě vycházející z pozměňovacího návrhu zákoníku práce č. 109/8, které vstoupí v platnost dne 1. července 2019 a je silně orientovaný na minimalizaci času, a tedy i nákladů spojených s administrativní prací vydané při vedení evidence pracovní doby. Při návrhu systému byla brána v potaz i jeho integrace s dalšími informačními systémy, které mohou cílové instituce provozovat, a to zejména v oblasti využití již existujících dat a autentifikace uživatelů.

Funkční prototyp systému odpovídající vypracovanému zadání a architektonickým návrhům byl vyvinut jako webová aplikace za pomoci moderních technologií a pro účely obhajoby diplomové práce byl dočasně uveřejněn pod adresou epd.novak-jiri.cz.

Systém je nazýván prototypem, protože pro jeho plné nasazení je potřeba, aby byl rozšířen tak, aby respektoval konkrétní potřeby cílové instituce, které mohou vyplývat z jejích kolektivních smluv nebo specifických potřeb spojených s jejími interními procesy.

5.1 Konkrétní výsledky práce

V průběhu zhotovování diplomové práce bylo dosaženo následujících konkrétních výsledků:

- Vypracování analýzy zákoníku práce z hlediska evidence pracovní doby;
- Vypracování analýzy zákoníku práce z hlediska evidence pracovní doby v akademickém prostředí;
- Vypracování zadání vývoje systému řešícího pracovní evidenci v akademickém prostředí;
- Vypracování architektonických návrhů vývoje systému plnícího identifikované zadání;
- Vyvinutí prototypu systému evidence pracovní doby v akademickém prostředí.

Analýza zákoníku práce z hlediska evidence pracovní doby

Tento výstup předkládané diplomové práce je objektivně nejvýznamnější, protože může být využit každým podnikatelem, který zaměstnává další osoby. Na základě retrospektivně vedených rozhovorů s podnikateli bylo zjištěno, že povědomí o legislativě spojené s pracovní evidencí je značně omezené a často je evidence pracovní doby zaměňována s evidencí docházky.

Analýzy legislativy v kontextu evidence pracovní doby, které jsou veřejně dostupné v době zhotovení předkládané práce, buď pokrývají problematiku pouze z perspektivy jednoho konkrétního zaměstnání, nebo jsou zastaralé, případně slouží pouze k propagaci služeb či produktů, a tedy vyzdvihují a potlačují fakta dle potřeby marketingu.

Analýza zákoníku práce z hlediska evidence pracovní doby v akademickém prostředí

Dalším neméně významným výstupem práce je obohacení analýzy zákoníku práce o specifika, která souvisejí s pracovními procesy probíhajícími v akademickém prostředí a zahrnutí nejnovějších legislativních úprav, která pracovní evidenci v tomto prostředí modifikují. Tato analýza může posloužit zejména vedoucím akademických pracovišť pro pochopení nové legislativy.

Zadání pro vývoj systému řešícího pracovní evidenci v akademickém prostředí

Formulované zadání může být využito ve vývoji systému pracovní evidence pro jakoukoliv akademickou instituci, protože při jeho vypracování byl kladen důraz na to, aby

zadání bylo obecné a nezahrnovalo specifické potřeby instituce, kde je předkládaná diplomová práce zhotovována. Zadání také není závislé na konkrétní formě implementace, takže výsledný systém, který odpovídá formulovanému zadání, může být dodán i jako mobilní nebo desktopová aplikace.

Architektonické návrhy systému plnícího vypracované zadání

Význam tohoto výstupu práce spočívá zejména v tom, že představované návrhy jsou znovu-použitelné, pro opětovný vývoj, pokud by někdo chtěl systém vyvinout s využitím jiných technologií, než které jsou využité v prototypu. Architektonické návrhy sice předpokládají implementaci systému formou webové aplikace a využití objektově orientovaného programování, ale dávají implementujícím vývojářům svobodu ve výběru konkrétních nástrojů použitých k vytvoření navrženého systému.

Prototyp systému

Prototyp systému lze považovat za jeden z klíčových výsledků práce, protože pro některé instituce může již v jeho současné podobě sloužit jako kompletní systém pro vedení pracovní evidence a pro ostatní instituce by mělo být velmi jednoduché dokončit či upravit prototyp tak, aby systém respektoval jejich specifické požadavky.

6 Závěr

Problematika vedení pracovní evidence v akademickém prostředí byla adresována vypracováním analýzy relevantní legislativy, která pomůže vedoucím pracovníkům ve všech odvětvích v pochopení jejich povinností.

Dle vypracované analýzy zákoníku práce bylo zjištěno, že problémy spojené s vedením pracovní evidence v akademickém prostředí jsou zapříčiněné především jeho administrativní náročností a nekompatibilitou zákonných norem s pracovními procesy probíhajícími v akademické sféře.

Vzhledem k problémům vyplývajícím z nekompatibilní legislativy byla následně provedena i analýza relevantních pozměňovacích návrhů zákonů, které byly projednávané v období vypracovávání předkládané práce. Z této analýzy vyplynulo, že problematikou vedení pracovní evidence v akademickém prostředí se zabývá pozměňovací návrh zákoníku práce č.109/8, který by v případě jeho schválení vyřešil v podstatě veškeré problémy spojené s nekompatibilitou legislativy. Pozměňovací návrh zákona byl zpočátku odmítnut senátem a vrácen do poslanecké sněmovny, poslanecká sněmovna však přijetí návrhu potvrdila více než dvoutřetinovou většinou a návrh byl podepsán prezidentem dne 30. ledna 2019. Tím bylo přijetí definitivně potvrzeno a pozměňovací návrh vstoupí v platnost dne 1. července 2019, čímž lze nekompatibilitu legislativy považovat za vyřešenou.

Problémy spojené s administrativní náročností vedení evidence pracovní doby v akademickém prostředí byly adresovány vypracováním zadání systému, který je navržen tak, aby vedoucím pracovníkům v akademickém prostředí maximálně usnadnil vedení pracovní evidence za pomoci využití importu existujících dat a napojení na autentifikační procedury již implementované v jejich institucích. Zadání je dostatečně obecné na to, aby systém mohl být implementován jakoukoliv formou a s využitím libovolných technologií, čímž je adresován důraz kladený na integraci se stávajícími systémy instituce.

Pro demonstrační účely byl vypracován návrh systému, který implementuje již zmíněné zadání formou webové aplikace s využitím objektově orientovaného programování a paradigmatu Code First. Na základě návrhu byl s využitím programovacího jazyka PHP a frameworku Nette implementován prototyp. Z výsledků implementace vyplynulo, že informační systém, který je schopen využívat existující data může snížit administrativní náročnost vedení pracovní evidence na absolutní minimum, protože drtivou většinu záznamů

pracovní evidence je možné extrahovat z osobních rozvrhů akademických pracovníků dostupných v univerzitním informačním systému. Vyvinutý prototyp naplňuje obecné zadání systému a některým institucím může sloužit i v jeho současné podobě jako plnohodnotný systém pro evidenci pracovní doby. Pro instituce, které mají kolektivní smlouvy rozšiřující zákoník práce, či případně realizují specifické pracovní procesy, je možné systém s nevelkým úsilím upravit tak, aby byl schopný řešit i jejich specifické potřeby.

Prototyp bude nadále vyvíjen i nad rámec diplomové práce, přičemž bude vyvinuta snaha o napojení na další datové zdroje, jako jsou například rozpisy konzultačních hodin a přehledy zkoušek a dozorů, což by mělo ještě více snížit administrativní náročnost a vedoucím pracovníkům akademických pracovišť ušetřit ještě více času, který může být lépe využit pro podílení se na výzkumné a vědecké činnosti.

7 Seznam použitých zdrojů

1. MENCL, Michal. Základní principy a pojmy objektově orientovaného programování. In: PÉHÁPKO.CZ: učebnice skriptovacího jazyka PHP [online]. [cit. 2019-03-08]. Dostupné z: http://pehapko.cz/oop/uvod

2. HORDĚJČUK, Vojtěch. Objektově orientované programování. In: VOHO [online]. 2019 [cit. 2019-03-08]. Dostupné z: http://voho.eu/wiki/oop/

3. BENEŠ, Miroslav. Principy objektově orientovaného programování. Vysoká Škola Báňská [online]. [cit. 2019-02-27]. Dostupné z: http://www.cs.vsb.cz/benes/vyuka/upr/texty/objekty/index.html

4. ČÁPKA, David. Dědičnost a polymorfismus. In: IT Network [online]. 2019 [cit. 2019-03-08]. Dostupné z: https://www.itnetwork.cz/csharp/oop/c-sharp-tutorial-dedicnost-a-polymorfismus

5. CHLÁDKOVÁ, Alena. Evidence pracovní doby. Mzdová Praxe [online]. 2007, 28.6.2007 [cit. 2019-02-27]. Dostupné z: http://www.mzdovapraxe.cz/archiv/dokument/doc-d2357v3114-evidence-pracovni-doby/

6. REDAKCE, epravo.cz. Maximální rozsah pracovní pohotovosti. In: Epravo.cz [online]. 2019, 26.6.2001 [cit. 2019-03-08]. Dostupné z: https://www.epravo.cz/top/clanky/maximalni-rozsah-pracovni-pohotovosti-9812.html

7. DOUŠOVÁ, Jana. Evidence pracovní doby není to samé, co docházka. In: Portál.POHODA [online]. 2012, 2.9.2016 [cit. 2019-03-08]. Dostupné z: https://portal.pohoda.cz/zakon-a-pravo/pracovni-pravo/evidence-pracovni-doby-neni-to-same-co-dochazka/

8. HRIADELOVÁ, Katarína a Aneta MATOUŠKOVÁ. Evidence docházky vs. evidence pracovní doby. V čem se liší?. Tulipize [online]. 29.1.2018 [cit. 2019-02-27]. Dostupné z: https://tulipize.cz/2018/01/29/evidence-dochazky-vs-evidence-pracovni-doby-v-cem-se-lisi/

9. ČESKÁ REPUBLIKA. Zákon č. 262/2006 Sb. In: Zákony pro lidi [online]. 2019, 07.06.2006 [cit. 2019-03-08]. Dostupné z: https://www.zakonyprolidi.cz/cs/2006-262/odkaz

10. RAIS, Karel. Současný zákoník práce nereflektuje specifika vysokoškolského prostředí: Projev na 19. schůzi Poslanecké sněmovny 3. října 2018 k zákoníku práce. In: Parlamentní listy [online]. 2019, 3.10.2018 [cit. 2019-03-08]. Dostupné z: https://www.parlamentnilisty.cz/politika/politici-volicum/Rais-ANO-Soucasny-zakonik-prace-nereflektuje-specifika-vysokoskolskeho-prostredi-553925

11. POSLANECKÁ SNĚMOVNA PARLAMENTU ČESKÉ REPUBLIKY, Sněmovní tisk 903 Novela z. - zákoník práce - EU. [online]. 9. 9. 2016. [cit. 2019-02-27]. Dostupné z: http://www.psp.cz/sqw/historie.sqw?o=7&t=903

12. POSLANECKÁ SNĚMOVNA PARLAMENTU ČESKÉ REPUBLIKY, Sněmovní tisk 109 Novela z. - zákoník práce. [online]. 22. 2. 2018 [cit. 2019-02-27]. Dostupné z: http://www.psp.cz/sqw/historie.sqw?o=8&T=109

13. POSLANECKÁ SNĚMOVNA PARLAMENTU ČESKÉ REPUBLIKY- Sněmovní tisk 109 Novela z. - zákoník práce. [online]. 22. 2. 2018 [cit. 2019-02-27]. Dostupné z: http://www.psp.cz/sqw/text/tiskt.sqw?o=8&ct=109&ct1=8

14. REDAKCE. Frontend. In: Adaptic: Internetová řešení podle vašich potřeb [online]. 2019 [cit. 2019-03-08]. Dostupné z: http://www.adaptic.cz/znalosti/slovnicek/frontend/

15. DOSTALOVÁ, Zuzana. Frontend vs. Backend. In: Czechitas [online]. 2016, 1.7.2014 [cit. 2019-03-08]. Dostupné z: https://www.czechitas.cz/cs/blog/zaciname-s-it/frontend-vs-backend

16. SHIOTSU, Yoshitaka. Web Development 101: Top Web Development Languages to Learn in 2018. In: Upwork [online]. 2019, 28.11.2017 [cit. 2019-03-08]. Dostupné z: https://www.upwork.com/blog/2017/11/top-web-development-languages-2018/

17. ROUSE, Margaret. HTML: Hypertext Markup Language. In: TheServerside [online]. 2019, Červen 2018 [cit. 2019-03-08]. Dostupné z: https://www.theserverside.com/definition/HTML-Hypertext-Markup-Language

18. REDAKCE. HTML Introduction. In: W3schools.com [online]. 2019 [cit. 2019-03-08]. Dostupné z: https://www.w3schools.com/html/html_intro.asp

19. MUNROE, Lee. Valid (X)HTML - Is It Important?. In: @leemunroe [online]. 2019, Prosinec 2008 [cit. 2019-03-08]. Dostupné z: https://www.leemunroe.com/how-important-is-valid-html-web-standards/

20. KOMUNITA. Kaskádové styly. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2019, 12.1.2019 [cit. 2019-03-08]. Dostupné z: https://cs.wikipedia.org/wiki/Kask%C3%A1dov%C3%A9_styly

21. KOMUNITA. Cascading Style Sheets. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2019, 6.3.2019 [cit. 2019-03-08]. Dostupné z: https://en.wikipedia.org/wiki/Cascading_Style_Sheets#CSS_1

22. REDAKCE. CSS Syntax and Selectors. W3schools.com [online]. 2019 [cit. 2019-03-08]. Dostupné z: https://www.w3schools.com/css/css_syntax.asp

23. DRASNER, Sarah. On Style Maintenance. CSS-TRICKS [online]. 21.10.2016 [cit. 2019-02-27]. Dostupné z: https://css-tricks.com/on-style-maintenance/

24. REDAKCE. What are the best HTML preprocessors?. Slant [online]. 2019, 20.2.2019 [cit. 2019-03-08]. Dostupné z: https://www.slant.co/topics/5427/~html-preprocessors

25. BONFIM, Kande. Improve your development workflow using HTML preprocessors. Startae [online]. 31.5.2016 [cit. 2019-03-08]. Dostupné z: https://startae.com/blog/improve-your-development-workflow-using-html-preprocessors/

26. MATT. A Beginner's Guide to CSS/HTML Preprocessors. Fluent.software [online]. 2017 [cit. 2019-03-08]. Dostupné z: http://fluent.software/a-beginners-guide-to-csshtml-preprocessors/

27. MICHÁLEK, Martin. Průvodce CSS preprocesory: co a jak?. Https://www.vzhurudolu.cz [online]. 10.3.2014 [cit. 2019-03-08]. Dostupné z: https://www.vzhurudolu.cz/blog/12-css-preprocesory-1

28. MONUS, Anna. 10 Reasons to Use a CSS Preprocessor in 2018. RAYGUN [online]. 2018, 28.5.2018 [cit. 2019-03-08]. Dostupné z: https://raygun.com/blog/10-reasons-css-preprocessor/

29. REDAKCE. Overview. LESS CSS [online]. [cit. 2019-03-08]. Dostupné z: http://lesscss.org/features/

30. KOMUNITA. CSS framework. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2019, 3.2.2019 [cit. 2019-03-08]. Dostupné z: https://en.wikipedia.org/wiki/CSS_framework

31. SHALEYNIKOV, Anton. Top 5 Most Popular CSS Frameworks that You Should Pay Attention to in 2017. HACKERNOON [online]. 24.10.2017 [cit. 2019-02-27]. Dostupné z: https://hackernoon.com/top-5-most-popular-css-frameworks-that-you-should-pay-attention-to-in-2017-344a8b67fba1

32. KOMUNITA. JavaScript. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2019, 12.1.2019 [cit. 2019-03-08]. Dostupné z: https://cs.wikipedia.org/wiki/JavaScript

33. REDAKCE. JavaScript - Syntax. Tutorialspoint [online]. [cit. 2019-03-08]. Dostupné z: https://www.tutorialspoint.com/javascript/javascript_syntax.htm

34. REDAKCE. JAVASCRIPT SECURITY. VERACODE [online]. 2019 [cit. 2019-03-08]. Dostupné z: https://www.veracode.com/security/javascript-security

35. KALLIN, Jakob a Irene Lobo VALBUENA. Part One: Overview: What is XSS?. Excess XSS: A comprehensive tutorial on cross-site scripting [online]. 2013 [cit. 2019-03-08]. Dostupné z: https://excess-xss.com/

36. VÁŠKO, Ondřej. Top 10 frameworků pro moderní frontend. *WDT* [online]. 1996-2019, 10.2.2017 [cit. 2019-03-08]. Dostupné z: https://www.wdt.cz/inovace/top-10-frameworku-pro-moderni-frontend-a589ddee72867455fd9c17a2d

37. KOMUNITA. JavaScript library. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2019, 15.1.2019 [cit. 2019-03-08]. Dostupné z: https://en.wikipedia.org/wiki/JavaScript_library

38. KOMUNITA. JQuery. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2019, 12.1.2019 [cit. 2019-03-08]. Dostupné z: https://cs.wikipedia.org/wiki/JQuery

39. VÁŠKO, Ondřej. Top 10 frameworků pro moderní frontend. *WDT* [online]. 1996-2019, 10.2.2017 [cit. 2019-03-08]. Dostupné z: https://www.wdt.cz/inovace/top-10-frameworku-pro-moderni-frontend-a589ddee72867455fd9c17a2d

40. REDAKCE. TypeScript vs CoffeeScript. EDUCBA [online]. 2019 [cit. 2019-03-08]. Dostupné z: https://www.educba.com/typescript-vs-coffeescript/

41. REDAKCE. Back-End Web Architecture. Code Academy [online]. 2019 [cit. 2019-03-08]. Dostupné z: https://www.codecademy.com/articles/back-end-architecture

42. WODEHOUSE, Carey. A Beginner's Guide to Back-End Development. Upwork [online]. [cit. 2019-02-27]. Dostupné z: https://www.upwork.com/hiring/development/a-beginners-guide-to-back-end-development/

43. ROSENBERG, Burton. Relational Databases. College of Arts and Sciences: Computer Science [online]. 2007 [cit. 2019-03-08]. Dostupné z: http://www.cs.miami.edu/home/burt/learning/Csc598.073/notes/reldb.html

44. ARSENAULT, Cody. The Pros and Cons of 8 Popular Databases. KeyCDN [online]. 2019, 20.4.2017 [cit. 2019-03-08]. Dostupné z: https://www.keycdn.com/blog/popular-databases

45. KOMUNITA. MySQL. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2019, 11.9.2018 [cit. 2019-03-08]. Dostupné z: https://cs.wikipedia.org/wiki/MySQL

46. FONTAINE, Axel. Relational Database Popularity Ranking. *Axel Fontaine* [online]. 23.3.2013 [cit. 2019-02-27]. Dostupné z: https://axelfontaine.com/blog/database-popularity.html 7

47. KOMUNITA. NoSQL. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2019, 4.7.2016 [cit. 2019-03-08]. Dostupné z: https://cs.wikipedia.org/wiki/NoSQL

48. KOMUNITA. PHP. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2019, 7.3.2018 [cit. 2019-03-08]. Dostupné z: https://en.wikipedia.org/wiki/PHP

49. REDAKCE. PHP /základy/. Tvorba Webu [online]. 2003, 20.4.2017 [cit. 2019-03-08]. Dostupné z: https://www.tvorba-webu.cz/php/

50. KOMUNITA. PHP. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2019, 28.2.2019 [cit. 2019-03-08]. Dostupné z: https://cs.wikipedia.org/wiki/PHP

51. O`DELL, Jolie. 8 Experts Break Down the Pros and Cons of Coding With PHP. Mashable [online]. 2019, 19.11.2010 [cit. 2019-03-08]. Dostupné z: https://mashable.com/2010/11/19/pros-cons-php/?europe=true#Ub5uQeAOYgqX

52. CUNNINGHAM, Ward. Php Pros And Cons. Wiki C2 [online]. 18.6.2014 [cit. 2019-03-08]. Dostupné z: http://wiki.c2.com/?PhpProsAndCons

53. PETROFČÍK, Matúš. Composer. IT Network [online]. 2019 [cit. 2019-03-08]. Dostupné z: https://www.itnetwork.cz/php/ostatni/composer

54. MONUS, Anna. 10 nejlepších PHP frameworků pro vývojáře. Interval [online]. 2019, 29.10.2015 [cit. 2019-03-08]. Dostupné z: https://www.interval.cz/clanky/10-nejlepsich-php-frameworku-pro-vyvojare/

55. KOMUNITA. Nette Framework. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2019, 22.2.2018 [cit. 2019-03-08]. Dostupné z: https://cs.wikipedia.org/wiki/Nette_Framework

56. ČÁPKA, David. Popis MVC architektury. IT Network [online]. 2019, 29.10.2015 [cit. 2019-03-08]. Dostupné z: https://www.itnetwork.cz/php/mvc/objektovy-mvc-redakcni-system-v-php-popis-architektury

57. KOMUNITA. Model–view–controller. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2019, 8.3.2018 [cit. 2019-03-08]. Dostupné z: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

58. KOMUNITA. Object-relational mapping. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2019, 17.2.2019 [cit. 2019-03-08]. Dostupné z: https://en.wikipedia.org/wiki/Object-relational_mapping

59. REDAKCE. Getting Started with Doctrine. Doctrine Project [online]. [cit. 2019-03-08]. Dostupné z: https://www.doctrine-project.org/projects/doctrine-orm/en/2.6/tutorials/getting-started.html

60. SASIDHARAN, Mithun. Should I Or Should I Not Use ORM ?. Medium [online]. 10.8.2016 [cit. 2019-03-08]. Dostupné z: https://medium.com/@mithunsasidharan/should-i-or-should-i-not-use-orm-4c3742a639ce