Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra managementu

Vývoj aplikace pro podporu projektového řízení

Bakalářská práce

Autor: Dominik Žilka
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Jan Petružálek

Odborný konzultant: Michal Brtníček, Meebio s.r.o.

Hradec Králové srpen 2018

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 17.8.2018 Dominik Žilka

Poděkování:
Rád bych poděkoval vedoucímu bakalářské práce, panu Ing. Janu Petružálkovi, za odborné vedení práce, trpělivost a cenné rady. Poděkování též patří mé rodině za soustavnou podporu během celé doby mého studia.

Anotace

Bakalářská práce se zabývá problematikou vývoje webových aplikací s důrazem na jazyk PHP a framework Nette. Dělí se na dvě základní části, přičemž v první, teoretické, části jsou čtenáři představeny základní principy webových aplikací, technologie pro vývoj front-endu (HTML, CSS a Javascript) a zejména pak jazyk PHP a Nette framework. Použití jednotlivých jazyků a technologií je demonstrováno na názorných příkladech. Druhá část práce je praktická a technologie představené v první části jsou v praxi použity při vývoji aplikace pro podporu projektového řízení pro reálného zadavatele. Vývoj aplikace začíná popisem problémů aktuálního řešení a požadavky zadavatele na řešení nové, následuje analýza a samotná implementace. Na závěr je nové řešení zhodnoceno a jsou zde i popsány plány na rozšiřování aplikace do budoucna.

Annotation

Application development for Project Management Support

The bachelor thesis deals with web applications development and focuses mainly on PHP technology and Nette framework. It is divided into two parts. The first theoretical part presents the basic principles of web applications functioning, front-end technologies (HTML, CSS, Javascript) and most importantly PHP language and Nette framework. The use of each technology is described using examples. The second – practical - part of thesis focuses on the development of an application for a real company, which starts with a description of the current employed solution and the company's requirements for the new application. It is then followed by an analysis and the actual implementation. The thesis concludes with an evaluation of the new application and a description of plans for future enhancements.

Obsah

Seznam obrázků

Seznam ukázek zdrojových kódů

Seznam grafů

Seznam tabulek

1 Úvod

V posledních letech jsme, díky rychlému rozšiřování vysokorychlostního internetu a pokroku webových technologií (zejména AJAX, HTML 5, atp.), svědky velkého rozmachu tzv. webových aplikací. Tyto aplikace, na rozdíl od desktopových, nemusí jejich uživatelé instalovat (nainstalovány jsou na aplikačním serveru), nejsou svázány s uživatelovým hardwarem a k běhu jim stačí pouze prostředí webového prohlížeče (který zde slouží jako klient). Mezi další nesporné výhody těchto aplikací můžeme jmenovat např. jednoduchý způsob aktualizace (aplikaci stačí aktualizovat na jednom místě, tj. na serveru) nebo nezávislost na konkrétní platformě (není nutné vyvíjet separátní aplikace pro různé operační systémy, které klienti používají). Webové aplikace v drtivé většině používají architekturu klient-server čili existuje rozhraní (klient), skrze které uživatel posílá požadavky serveru a serverová část, která na dotazy odpovídá.

Webové aplikace dnes dokážou zastat drtivou většinu úloh, které byly až donedávna výhradní doménou aplikací desktopových, v důsledku čehož jsou často nuceni výrobci původně desktopových aplikací nabízet i on-line alternativy svých produktů. Příkladem může být i třeba společnost Microsoft se svojí sadou Office. Kromě kancelářských balíků jsou webové aplikace často využívány jako e-mailoví klienti (Roundcube, Horde, …), aplikace pro komunikaci (ať už jde o sociální sítě, diskuzní fóra atp.), nástroje pro správu serverů a databází (ISPConfig, phpMyAdmin a další) nebo dokonce i pokročilé grafické editory (např. Photopea).

Dalším segmentem, kde převládají webové aplikace jsou nástroje pro projektového řízení, tedy aplikace, které mají za cíl pomoci s plánováním, rozdělováním a organizací práce na konkrétních projektech.

2 Cíl práce

Cílem práce je uvést čtenáře do problematiky vývoje webových aplikací, nastínit základní mechanismy jejich fungování, představit dostupné technologie a ty pak aplikovat na konkrétní zadání od reálné společnosti.

Práce bude rozdělena do dvou částí, první, teoretická, rozebere a představí existujícím technologiím pro vývoj webových aplikací (jak pro vývoj klientské, tak serverové části). Důraz bude kladen hlavně na jazyk PHP a technologii Nette.

Druhá část bude praktická, bude se věnovat vývoji aplikace pro podporu projektového řízení na platformě Nette. Začne popisem potřeb zadavatele, společnosti Meebio, s.r.o., včetně zhodnocení současného stavu. Následně je zde popsána analýza zadaného projektu a posléze i vlastní implementace navrženého řešení. Na závěr budou cíle práce vyhodnoceny, včetně budoucího rozšíření aplikace.

3 Webové aplikace

Webová aplikace je software obvykle založený na architektuře klient-server, který používá jako klienta webový prohlížeč (který v tomto případě představuje tzv. tenkého klienta). Jak už název napovídá, tato architektura obsahuje dvě vzájemně oddělené vrstvy:

- **Klient** – Část aplikace (obvykle s grafickým rozhraním), se kterou pracuje uživatel a skrze kterou posílá požadavky (např. zobraz nadcházející události v kalendáři) na server. Zároveň přijímá odpovědi na uživatelovi požadavky a prezentuje je uživateli.
- **Server** – Část aplikace, která zpracovává uživatelské požadavky a vrací na ně odpovědi. Pojmem server se zde rozumí jak server webový, tak server databázový.

(1)

První webové aplikace se začaly objevovat s rozmachem internetu zejména v průběhu druhé poloviny 90. let 20. století. V dnešní době webové aplikace představují neustále rostoucí segment trhu, a to zejména pro jejich nesporné výhody:

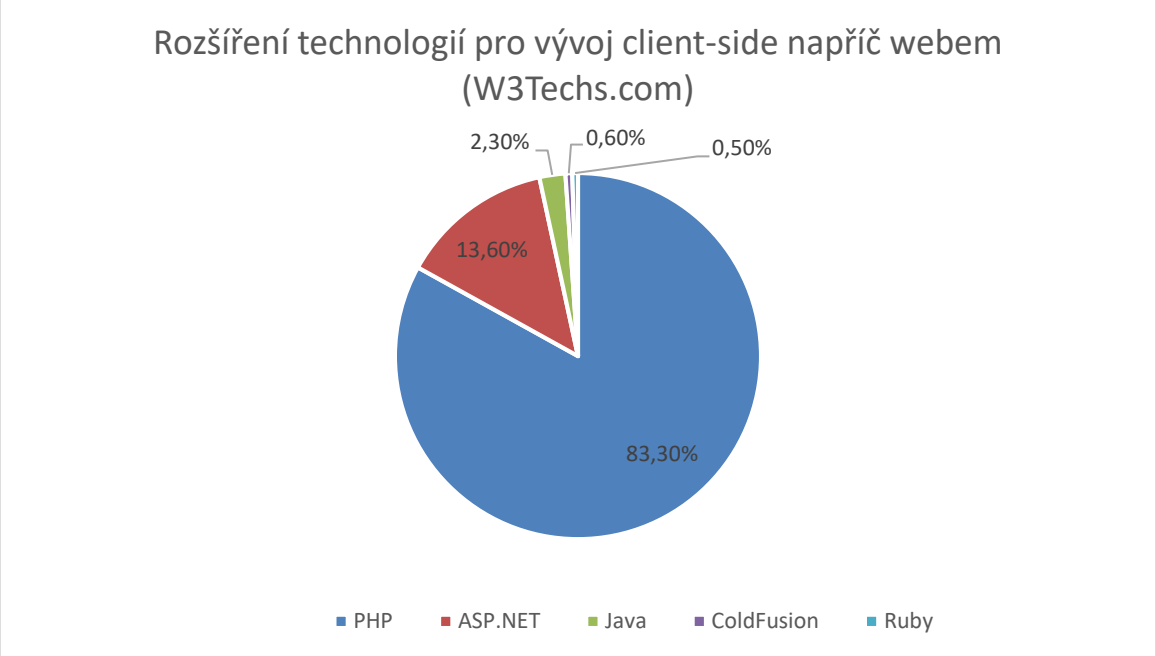
- **Odpadá nutnost instalace klienta a jsou multiplatformní** – jako klient slouží webový prohlížeč, který dnes představuje nedílnou součást všech moderních operačních systémů
- **Snadná aktualizace klienta** – klient se znovu načítá při každém obnovení stránky, odpadá tudíž nutnost řešit proces aktualizace na straně uživatele při vydání nové verze softwaru
- Dostupnost všude, kde je připojení k internetu a internetový prohlížeč

3.1 Technologie pro vývoj webových aplikací

Webové aplikace obvykle používají, jak už vyplývá z výše popsané architektury, kombinaci server-side skriptů (k operacím s daty) a client-side skriptů (k prezentaci dat uživatelům). Nejčastěji používané technologie, jak pro front-end, tak back-end, si nyní dovolím ve stručnosti představit.

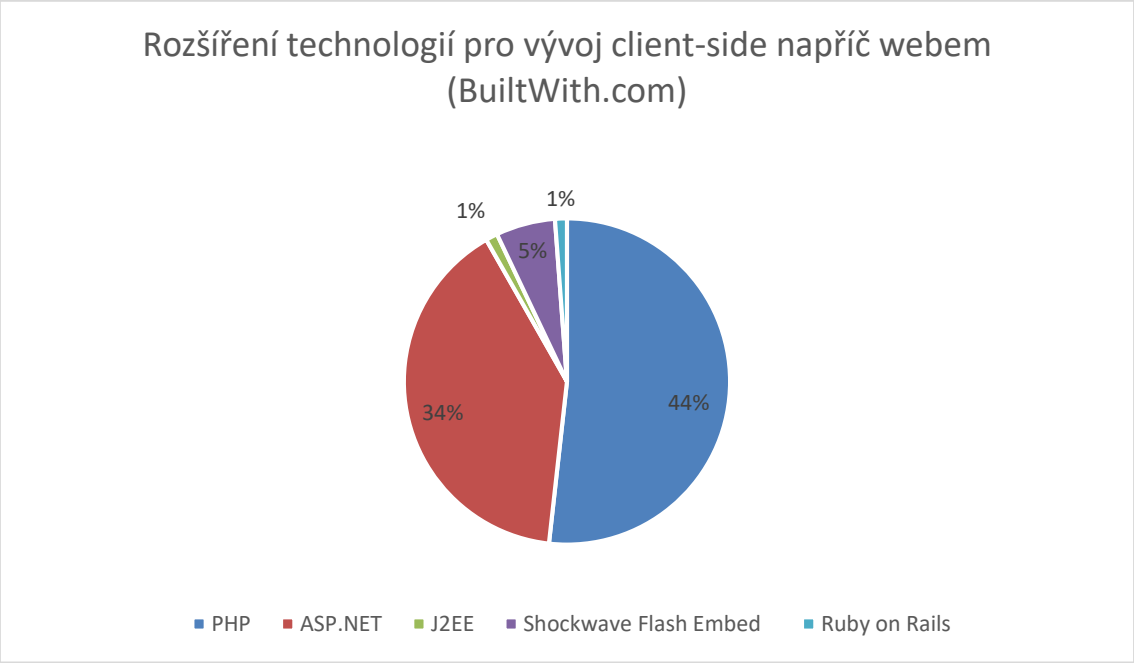
3.1.1 Server-side technologie

Jak již bylo zmíněno, serverová část aplikace (tzv. back-end) zpracovává požadavky od klientů, skladuje a zpracovává data (často s pomocí databázového serveru). Paleta použitelných technologií pro vývoj této části aplikace je více než pestrá, tudíž zmíním pouze těch několik nejpoužívanějších. Stanovit nejpoužívanější technologie bohužel nepředstavuje zrovna snadný úkol. V grafu níže jsou zobrazeny výsledky výzkumu serveru W3Techs.com, který na denní bázi vytváří statistiky používaných technologií napříč celým webem, a který v době psaní práce (2018) indexuje více než 10 milionů webů.



Graf 1 Rozšíření webových technologií podle W3Techs.com, zdroj: https://w3techs.com/technologies/overview/programming_language/all

Poněkud odlišené výsledky (vzhledem k pravděpodobně odlišné metodice výzkumu) ovšem vyplývají z průzkumu serveru BuiltWith.com. Pro úplnost je zde zmiňuji taktéž.



Graf 2 Rozšíření webových technologií podle BuiltWith.com, zdroj: https://trends.builtwith.com/framework

Ať už vycházíme z grafu 1 či grafu 2, vychází nám, že nejčastěji užívané technologie na straně serveru jsou:

- PHP
- .NET (ASP.NET)
- Java/J2EE
- Flash
- Ruby on Rails

Tyto technologie se nyní pokusím přiblížit trochu více. Dovolím si vynechat Flash, neboť se od roku 2017 jedná o výrobcem (společností Adobe) zavrhnutou technologii (zejména kvůli přetrvávajícím problémům s bezpečností a celkovou zastaralost), jejíž podpora úplně skončí v roce 2020 a ve většině dnešních prohlížečů již skončila. (2)

PHP bude později věnována samostatná kapitola (vzhledem k tomu, že v něm budu psát praktickou část práce), proto jej v tuto chvíli vynechám.

3.1.1.1 .NET

Technologie .NET je platforma pro tvorbu aplikací vyvíjená společností Microsoft od konce devadesátých let, přičemž první stabilní verze byla vydána 13. února 2002. Framework .NET je primárně určen pro operační systém Windows (nicméně v dnešní době existují i možností, jak jej využít i na jiných operačních systémech). Platformu .NET lze využít k tvorbě různých typů aplikací, od aplikací běžících pouze v prostředí příkazového řádků (tzv. konzolové aplikace) přes aplikace s vlastním GUI (Graphic User Interface), až po aplikace webové (s využitím části frameworku zvané ASP.NET).

Technologie .NET se skládá ze dvou základních částí:

- **Common Language Runtine (CLR)** – běhové prostředí, které má na starosti překlad (kompilaci) zdrojového kódu programovacího jazyka do „Common Intermediate Language“ (CIL) a následně do kódu strojového. Zajišťuje správu paměti, a to jak přidělování nové paměti, tak její uvolňování (po např. už dále nepoužívaných objektech) – tzv. „garbage collection“. Dále také poskytuje rozhraní pro objektově-orientované programování (dědičnosti, podpora interface, přetěžování atp.), mechanismy pro odchytávání výjimek, nástroje pro tvorbu aplikací používajících více vláken atd.
- **.NET Framework Class Library** – robustní knihovna tříd, kterou lze používat ve vyvíjených aplikací. Knihovna je dělena do jednotlivých namespaců (např. System) a poskytuje programátorům mnoho nástrojů pro často využívané operace jako je např. práce s textovými soubory, XML, databází atp.

Framework .NET nepředepisuje využití jednoho konkrétního programovacího jazyka, vzhledem k tomu že všechen kód je zde překládán do CIL. Díky tomu lze v .NETu pracovat hned v několika jazycích, přičemž mezi ty nejobvyklejší patří:

- **C# –** objektově-orientovaný jazyk (zejména, nicméně podporuje více paradigmat) vytvořený Microsoftem kolem roku 2000, jeden z nejpoužívanějších programovacích jazyků současnosti. Syntaxi přebírá z jazyka C a je velmi podobný programovacímu jazyku Java
- **F# -** multi-paradigmatický jazyk (nicméně preferované je funkcionální programování), vycházející ze syntaxe jazyků ML a OCaml.
- **Visual Basic .NET –** nástupce jazyka Visual Basic, obsahuje plnou podporu objektů a od jazyka C# se liší zejména syntaxí, jejich využití v rámci .NET je podobné

(3)

ASP.NET

Vzhledem k tomu, že se ve své práci věnuji vývoji webových aplikací, je třeba se alespoň krátce zmínit o možnostech vývoje webových aplikací na platformě .NET. Webové aplikace zde obstarává technologie ASP.NET určená právě pro tvorbu dynamických webových aplikací. Je nástupcem starší technologie ASP (Active Server Pages).

Stejně jako v případě samotného frameworku .NET (který umožňuje vícero přístupů k vývoji aplikací), tak i technologie ASP.NET nabízí více přístupů, jak postavit webovou aplikaci. Mezi nejvýznamnější modely technologie ASP.NET patří:

- **ASP.NET Web Forms** – vůbec první dostupný model v ASP.NET (první verze vyšla již v roce 2002), základním elementem ve Web Forms jsou komponenty (např. tlačítka), na které se vážou události. Tento model již není podporován v poslední verzi ASP (ASP.NET Core).
- **ASP.NET MVC –** implementace architektury MVC (Model-View-Controller, podrobněji popsán níže) v prostředí ASP.NET
- **ASP.NET Core –** nejnovější inkarnace technologie ASP.NET (navazující na APS.NET MVC)

3.1.1.2 Java

Přímým konkurentem technologií postavených na frameworku .NET jsou technologie postavené na programovacím jazyku Java. Java je multi-paradigmatický programovací jazyk vyvinutý společností Sun Microsystems (dnes součást společnosti Oracle). Stejně jako v případě technologií z rodiny .NET i zde existuje více technologií jejíž prostřednictvím lze vyvíjet webové aplikace.

JSP (JEE)

Jako odpověď na vzrůstající oblibu PHP a JSP byla v roce 1999 uvolněna první verze technologie JSP (Java Server Pages). JSP patří do balíku technologií označované jako Java Enterprise Edition (JEE, dříve J2EE). Samotná distribuce JEE zahrnuje kromě JSP, také aplikační server (který se stará např. o správu databázových spojení), technologii JSF (Java Server Faces) a Servlety. Dohromady tento balík tvoří silný nástroj pro psaní robustních webových aplikací.

Servlet představuje třídu v Javě, která se stará o vyřizování http požadavků, JSP pak představuje XML dokument se speciálními značkami (využívající jmenný prostor jsp:), který je na serveru překládán do podoby servletů. Ačkoliv jdou tyto dvě technologie používat odděleně, je dobrým zvykem je kombinovat. Servlety se starají o aplikační logiku, JSP se pak stará o logiku prezentační (suplují šablonový systém). (4)

Alternativou k JEE může být například framework Spring.

3.1.1.3 Ruby on Rails

Další technologií, u které bych se rád krátce zastavil je framework Ruby on Rails. Jak již jeho název napovídá, je naprogramován v programovacím jazyce Ruby. Jazyk Ruby je interpretovaný (tzn. nekompiluje se), skriptovací jazyk, podobně jako například PHP, Perl nebo Python, představený v roce 1995 japonským programátorem Jukihirem Macumotem. Ruby je plně objektové (všechno v Ruby je objektem), a vyznačuje se poměrně jednoduchou syntaxí. (5)

Framework samotný byl poprvé uveden v roce 2004 dánským programátorem Davidem Heinemeierem Hanssonem, který jej vyvinul pro svůj projekt Basecamp (aplikace pro řízení projektů a týmovou komunikaci).

Samotná filozofie RoR je postavena tak, aby bylo třeba psát, co nejméně kódu, a to díky zavedeným konvencím, které uživatel-programátor dodržuje. Tomuto přístupu se říká „Convention over configuration“, česky „Konvence před konfigurací“. V praxi to znamená, že uživatel konfiguruje pouze ty části aplikace, které se liší od běžně užívaných zvyklostí (konvencí), což má za následek urychlení vývoje. Ruby on Rails implementuje, stejně jako mnoho jiných frameworků, návrhový vzor MVC (model-view-controller).

Dalším principem, kterého se framework Ruby on Rails drží je důraz na využití REST API, což je způsob, jak posílat požadavky od klienta na server pomocí jednoduchých HTTP volání. (6)

3.1.2 Client-side technologie

V kostce jsme si představili některé z používaných technologií na straně serveru, nyní je čas zaměřit se na technologie, které jsou používány na straně klienta. Představíme si tři základní technologie, které jsou stěžejní pro službu WWW, a to HTML, CSS a Javascript.

3.1.2.1 HTML

HTML nebo Hyper-Text Markup Language je univerzální značkovací jazyk pro tvorbu webových stránek, které jsou navzájem propojeny hypertextovými odkazy. Za jeho vznikem stojí Tim Berners-Lee, který jeho první verzi vydal v roce 1993 (v té době jako pracovník švýcarského CERNu). (7)

V následujících odstavcích si popíšeme syntaxi jazyka a strukturu HTML dokumentu. Dovolím si upozornit, že vycházím z aktuální verze HTML (tj. HTML 5), odlišnostem oproti jiným verzím HTML se pak budu věnovat v samostatné podkapitole.

Syntaxe jazyka HTML

HTML dokumenty jsou složeny z tagů (značek) a jejich vlastností (atributů). Tagy obalují jednotlivé části textu a tím jim dávají význam a zapisují se mezi znaky < a > („špičaté“ závorky). Jazyk HTML rozlišuje tagy dvojího typu, a to párové a nepárové.

Párové značky představují obvyklejší typ. Tag se zde skládá ze dvou částí, a to ze značky otevírací a uzavírací. Uzavírací značka se liší od značky otevírající tím, že hned po znaku závorky obsahuje znak lomítka. Příkladem takové značky může být tag pro tučný text.

```
<strong>Lorem ipsum</strong>
```

Zdrojový kód 1 Ukázka párového HTML tagu

Nepárové tagy, jak již název napovídá, obsahují pouze jednu část. Příkladem budiž tag pro vložení obrázku.

```
<img src="obrazek.jpg" alt="Obrázek č. 1">
```

Zdrojový kód 2 Ukázka nepárového HTML tagu

V příkladu výše je zároveň názorně demonstrováno, jak se v jazyce HTML zapisují atributy tagů, tedy za název vlastnosti, do uvozovek (případně lze i do apostrofů). Tagy se zapisují malými písmeny. Platí, že nejdříve otevřený tag se uzavírá až jako poslední.

Struktura HTML dokumentu

Validní HTML dokument by měl obsahovat čtyři povinné části:

- Deklaraci typu dokumentu (tzv. DOCTYPE) - v HTML 5 se již neuvádí verze HTML, tzn. jako typ dokumentu se uvede pouze hodnota html.
- Kořenový tag <html> (a párový </html>) obalující celý dokument
- Hlavičku dokumentu obalenou tagy <head> a </head>
- Tělo dokumentu zabalené do tagů <body> a </body>

Celá struktura pak bude zjednodušeně vypadat takto:

```
<!DOCTYPE html>
<html>
      <head>
            <!--- Hlavička dokumentu --->
      </head>
      <body>
            <!--- Tělo dokumentu --->
      </body>
</html>
```

Zdrojový kód 3 Zjednodušená struktura HTML dokumentu

Jak z výše uvedeného schématu vyplývá, nejdůležitějšími částmi HTML dokumentu jsou hlavička a tělo, proto si je pojďme v krátkosti představit i s tagy, které se v nich vyskytují. V přehledech tagů jsou uváděny pouze vybrané tagy, kompletní seznam použitelných tagů lze dohledat v oficiální specifikaci HTML. (8) (8)

Hlavička HTML dokumentu

Hlavička představuje prostor pro metadata webové stránky (tzn. data o datech, která se přímo neobjevují ve vlastní stránce), odkazy na externí stylopisy a skripty. V neposlední řadě také obsahuje titulek stránky. Nejobvyklejšími tagy, které hlavička dokumentu obsahuje jsou:

- **<title>** - Nadpis (název) celého HTML dokumentu. Jeho obsah je obvykle vidět v nadpisu okna (tabu) v prohlížeči. Jde o nejdůležitější tag pro vyhledávače. V celém (validním) dokumentu se vyskytuje právě jednou. Párový tag.
- **<link>** - Tag sloužící k připojení externích stylopisů, favicon a feedů (např. RSS). Neparový tag, obvykle obsahující atributy href (cesta k externímu souboru), rel (typ připojeného souboru, např. stylesheet pro styl), případně media (určující typ zobrazení, ke kterému se daný stylopis váže)
- **<script>** - Párový tag, odkazující buď na externí skript (cestu ke skriptu pak obsahuje atribut src) nebo obsahující daný skript přímo mezi svými značkami. Často bývá doplněn atributem type, který indikuje typ skriptu (např. „text/javascript“). Tag script může být umístěn i v těle dokumentu.
- **<meta>** - Nepárový tag obsahující meta data o dokumentu. Obsahuje buď atributy name (pro identifikaci typu informace jako je např. popisek – „description“ nebo informace o autorovi dokumentu – „author“) a content (vlastní informace), nebo atributy http-equiv (pro manipulaci s hlavičkami HTTP požadavku) a content (obsah hlavičky), případně atribut charset, pro uvedení typu kódování stránky.

Následuje přehled nejpoužívanějších meta tagů:

Meta tag	Význam
`description`	Popisek stránky, tag často používaný vyhledávači pro indexaci.
`author`	Obsahuje informace o autorovi dokumentu.
`keywords`	Tag sloužící k uvedení klíčových slov dokumentu. Bohužel moderní webové vyhledávače jej v drtivé většině případů ignorují. Důkazem budiž vyjádření společnosti Google (9) nebo společnosti Seznam.cz (10)
`charset`	Označuje kódováni, ve kterém je dokument napsán. Obvykle se uvádí na prvním místě hlavičky.
`viewport`	Říká prohlížeči, jak velkou šířku má použít pro vykreslování stránky. Tento tagu se využívá zejména pro aktivaci responsivního layoutu dokumentu.

Tabulka 1 Přehled meta tagů

Na závěr uvádím příklad, jak může vypadat kompletní hlavička HTML dokumentu:

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="UTF-8">
		<meta name="description" content="Ukázka hlavičky HTML">
		<meta name="author" content="Dominik Žilka">
<meta name="viewport" content="width=device-width, initial-scale=1.0">

		<title>Ukázková hlavička HTML dokumentu</title>

		<link rel="stylesheet" href="style.css" media="screen">

		<script type="text/javascript" src="script.js">
	</head>
	<body>
		<!--- Tělo dokumentu --->
	</body>
</html>
```

Zdrojový kód 4 Kompletní hlavička HTML dokumentu

Tělo HTML dokumentu

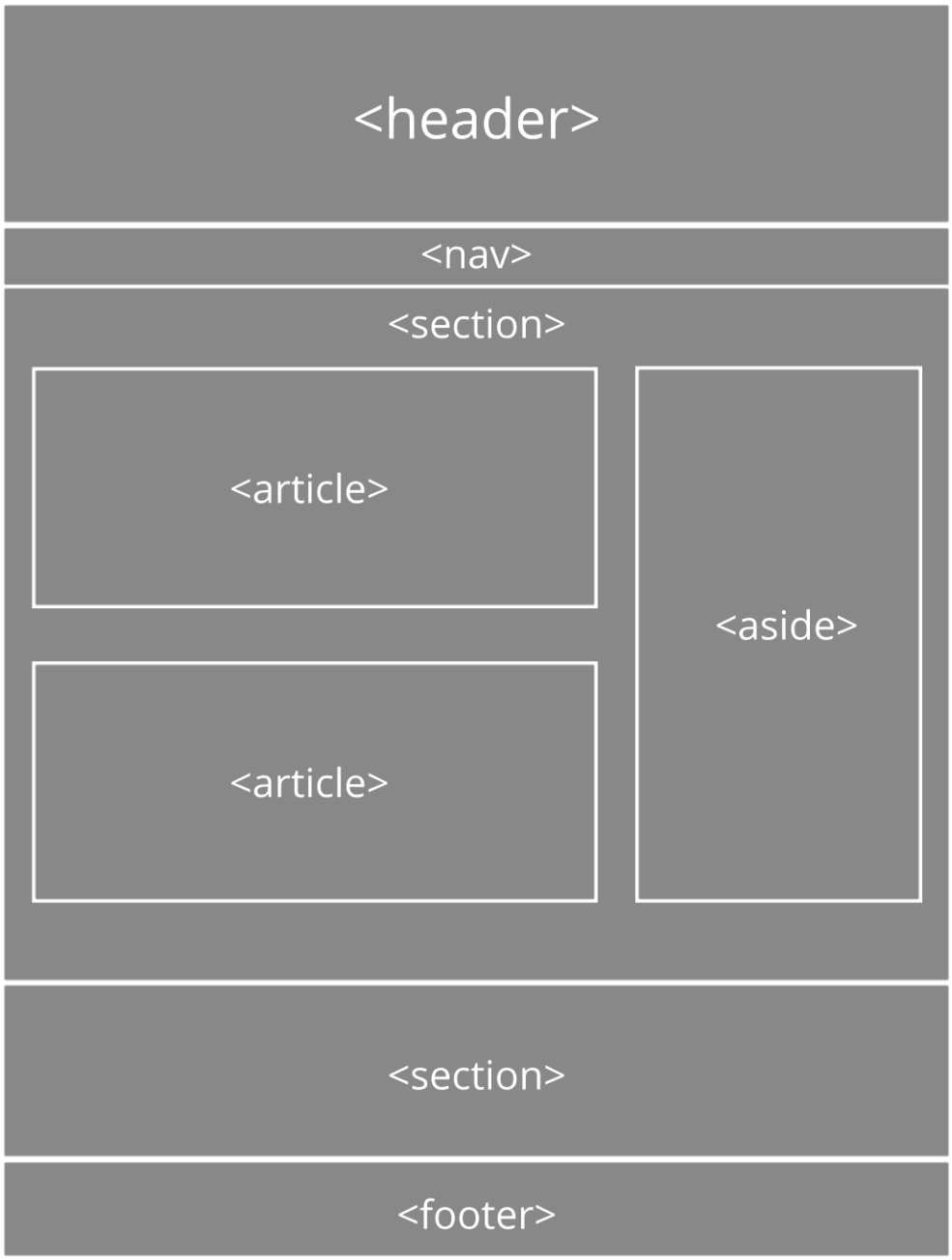
Tělo HTML dokumentu (čili všechen obsah umístěný mezi značky <body> a </body>), představuje tu část HTML dokumentu, kterou uživateli webový prohlížeč vykreslí. Protože tagů, které se mohou v těle HTML dokumentu nacházet, je velké množství, rozdělím je pro větší přehlednost do skupin.

Tagy pro členění dokumentu

Do této skupiny patří tagy, které dělí dokument do logických celků a zároveň zlepšují strojovou čitelnost výsledného HTML dokumentu. Ve starších verzích HTML existoval pro tuto funkci pouze tag , v HTML 5 přibyla široká paleta nových tagů, která možnosti správného sémantického strukturování HTML dokumentu značně rozšiřuje. Kromě již zmíněného tagu , HTML 5 nově rozlišuje i tagy:

- **<article>** - představuje článek
- **<aside>** - sidebar, obsah v postranním sloupci
- **<footer>** - patička dokumentu, případně sekce
- **<header>** - hlavička dokumentu, případně sekce
- **<main>** - hlavní obsah dokumentu
- **<nav>** - místo pro navigaci dokumentu
- **<section>** - sekce webu, která v sobě může obsahovat další element

Mezi výše zmíněnými tagy existuje určitá hierarchie, kterou si můžeme jednoduše demonstrovat na nákresu, který bude představovat webovou stránku s hlavičkou, navigací, sidebarem, patičkou a výpisem článků.



Obrázek 1 Sémantické tagy v HTML 5, zdroj: vlastní

Tagy pro formátování textu

V minulé kapitole jsem rozebíral, jak je vhodné HTML dokument strukturovat, nyní se můžeme podívat na to, jaké možnosti nám HTML nám nabízí z pohledu formátování textu.

Tag	Význam
`<p>Odstavec.</p>`	Párový tag pro odstavec. Blokový element.
`<strong>Tučné písmo.</strong>`	Párový a řádkový tag pro tučné písmo.
`<em>Kurzíva</em>`	Párový a řádkový tag pro kurzívu.
`<a href="#" title="link">odkaz</a>`	Párový a řádkový tag pro vložení hypertextového odkazu.
` `	Nepárový, řádkový element, označující konec řádku.
`<hr>`	Horizontální čára.

Tabulka 2 Tagy pro formátování textu

Nadpisy

Kromě hlavního názvu dokumentu uvedeném v tagu <title> může HTML dokument obsahovat i libovolný počet dalších nadpisů uzavřených do tagů <h1>, <h2>, <h3>, <h4>, <h5> a <h6>. Číslo za písmenem h v tagu představuje prioritu nadpisu, kdy element <h1> představuje nadpis nejvyšší úrovně, <h6> pak nejnižší úrovně.

Dokument by měl obsahovat právě jeden nadpis nejvyšší úrovně (<h1>). Dále platí, že by neměl být v kódu použit nadpis vyšší úrovně před nadpisem nižší úrovně, tak jak je to demonstrováno v následující ukázce:

```
<h1>Nadpis první úrovně</h1>
<h3>Nadpis třetí úrovně</h3>
<h2>Nadpis druhé úrovně</h2>
```

Zdrojový kód 5 Špatná struktura nadpisů v HTML dokumentu

Správná struktura nadpisů bude vypadat takto:

```
<h1>Nadpis první úrovně</h1>
<h2>Nadpis druhé úrovně</h2>
<h3>Nadpis třetí úrovně</h3>
```

Zdrojový kód 6 Správná struktura HTML nadpisů

Seznamy

Dalším hojně využívaným elementem v HTML dokumentech jsou seznamy. HTML nám v tomto směru přináší více možností:

- **<ul>** - nečíslovaný seznam
- **<ol>** - číslovaný seznam
- **<dl>** - definiční seznam

První dva typy seznamů mají podobnou syntaxi, která se liší pouze tagem seznamu:

```
<ul>
      <li>První nečíslovaného položka seznamu</li>
</ul>
<ol>
      <li>První položka číslovaného seznamu</li>
</ol>
```

Zdrojový kód 7 Ukázka číslovaného a nečíslovaného seznamu

Jak je vidět z příkladu výše, do tagu <ul> (př.) se vnořují jednotlivé položky seznamu (odrážky) zabalené do tagu <li>. Jiný tag se do <ul> nebo <ol> nezanořuje. Trochu odlišná situace je u posledního typu seznamu, tedy seznamu definičního. Tag <dt> představuje termín, <dd> pak jeho definici.

```
<dl>
      <dt>Pojem</dt>
      <dd>Vysvětlení pojmu</dd>
</dl>
```

Zdrojový kód 8 Definiční seznam v HTML

Formuláře

Formuláře představují důležitý prvek webu, skrze který může uživatel s webem (nebo webovou aplikací) interagovat. Podpora formulářů je zejména v poslední verzi HTML (ve verzi 5) rozsáhlá a určitě bychom je v našem přehledu neměli opomenout.

Formulář se skládá z obalujícího tagu <form> a formulářových prvků (které mohou být ještě rozděleny do skupin tagem <fieldset>). Obalující tag <form> má obvykle minimálně tyto dva atributy:

- **action** – url adresa, která zpracuje data z formuláře
- **method** – určuje, jestli se formulář bude zpracovávat HTTP metodou POST nebo GET (nejzásadnější rozdíl spočívá v tom, že GET požadavky jsou vidět v url, POST nikoliv)

Prvky formuláře obvykle představuje nepárový tag <input>. Typ prvku formuláře se určuje pomocí atributu type. Přehled často užívaných prvků formulářů ukazuje následující tabulka:

type	význam
text	Textové pole na jeden řádek.
password	Pole určené pro zadávání hesla.
file	Pole určené pro nahrání souboru.
hidden	Skryté pole (vůbec se nevykresluje)
radio	Tlačítko radio
checkbox	Zaškrtávací tlačítko
submit	Tlačítko pro odeslání formuláře
button	Tlačítko, které neodesílá formulář (dá se napojit např. na JS událost)
reset	Tlačítko, které resetuje všechny hodnoty ve formuláři
date (HTML 5)	Pole určené pro zadávání datumu (bez času)
datetime-local (HTML 5)	Pole určené pro zadávání datumu a času
number (HTML 5)	Pole určené pro zadávání čísel
email (HTML 5)	Pole určené pro zadávání e-mailu
tel (HTML 5)	Pole určené pro zadávání telefonního čísla

Tabulka 3 Typy tagu input

Kromě formulářových prvků v tagu <input>, existují ještě další prvky, které stojí za zmínku:

- **select** – výběr z předefinovaných možností, zobrazený pomocí výběru s roletkou
- **textarea** – pole pro zadání textu (víceřádkové)

Všechny výše zmíněné formulářové prvky můžeme doplnit např. o atributy placeholder (ukázkový text, který po kliknutí na příslušné pole a zadání prvního znaku mizí), value (předdefinovaná hodnota). Každý formulářový prvek musí také obsahovat atribut name, který definuje název prvku (a který je v rámci formuláře unikátní). Prvky formuláře je také vhodné doplnit popisky, ke kterým slouží tag <label>. Na závěr části o formulářích si pojďme ukázat kompletní ukázku formuláře v HTML.

```
<form action="index.php" method="post">
<fieldset>
            <label for="jmeno">Jméno</label>
            <input type="text" name="jmeno" placeholder="Jan Novák">

            <label for="em">E-mail</label>
            <input type="email" name="em" placeholder="jan@novak.cz">

            <label for="zkusenost">Zkušenost</label>
            <select name="zkusenost">
                  <option value="nizka">Začátečník</option>
                  <option value="stredni">Pokročilý</option>
                  <option value="vysoka">Expert</option>
            </select>

            <label for="upload">Životopis</label>
            <input type="upload" name="cv">

            <input type="submit" name="submit" value="Odeslat">
      </fieldset>
</form>
```

Zdrojový kód 9 Ukázka kompletního formuláře v HTML

Tabulky

Posledním typem značek, které si představíme, jsou tagy pro tvorbu tabulek. Historicky se tabulky v HTML používali i k jiným účelům než k reprezentaci dvourozměrných dat (zejména k tvorbě tzv. tabulkového layoutu), s příchodem a rozšířením CSS se nicméně od těchto technik již upustilo.

Samotná tabulka v HTML je obalena párovým tagem <table>, tvoří se po řádcích obalených tagem <tr>. V řádcích se pak nachází jednotlivé buňky obalené v tazích <td>, případně <th> pokud se jedná o záhlaví. Tabulka může být rozdělena na jednotlivé části pomocí párových tagů <thead>, <tbody> a <tfoot>.

```
<table>
	<tr>
		<th>Nadpis prvního sloupce</th>
		<th>Nadpis druhého sloupce</th>
	</tr>
	<tr>
		<td>1. sloupec, 1. řádek</td>
		<td>2. sloupec, 1. řádek</td>
	</tr>
	<tr>
		<td>1.  sloupec, 2. řádek</td>
		<td>2. sloupec, 2. řádek</td>
	</tr>
</table>
```

Zdrojový kód 10 Tabulka v HTML

Vývoj jazyka HTML

Historie jazyka HTML je úzce spjata s historií technologie WWW (World Wide Web), kterou vytvořil Tim Berners-Lee na konci 80. let (představena byla v roce 1989) ve švýcarském CERNu (Conseil européen pour la recherche nucléaire, česky Evropská organizace pro jaderný výzkum, vědecká organizace). Spolu se samotnou technologií WWW, stvořil značkovací jazyk HTML a také první webový prohlížeč (v roce 1991), který uměl jazyk HTML interpretovat. První verze, označovaná jako „HTML Tags" obsahovala 18 tagů, z nichž některé se užívají dodnes. Jako příklad si můžeme uvést např. tag <a> pro odkaz, tagy nadpisů <h1> - <h6>, nebo tagy <ul> a <li> (ačkoliv se původně tyto tagy psaly velkým písmeny).

Po odchodu z CERNu založil Tim Berners-Lee v roce 1994 konscoricum W3C, které se od té doby (v posledních letech spolu s organizací WhatWG) stará o vývoj jazyka HTML a jeho nových verzí. Hlavních verzí HTML bylo do dnešní dne publikováno několik:

- **Větev HTML 1** – první (a již zmíněná) verze HTML, kterou stvořil Berners-Lee ještě v CERNu. Nepodporovala grafiku, finální verze nesla číslo 1.2
- **Větev HTML 2** – vydaná pod hlavičkou organizace IETF. Postupně (v rámci různých minor verzí) nabídla již podporu grafických prvků a interaktivních formulářů (včetně uploadu)
- **HTML 3.2** – vydána již pod hlavičkou konsorcia W3C. Záměrně se zmiňuji o HTML 3.2, protože verze 3 zůstala pouze ve fázi technického návrhu, vzhledem k potížím s implementací, které měli vývojáři tehdejších prohlížečů. Verze HTML 3.2 přinesla nově například podporu tabulek.

- **HTML 4** – vydána na konci prosince 1997, opět pod hlavičkou konsorcia W3C. Zajímavé je, že vyšla rovnou ve třech verzích:
    - Strict – verze, která zakazuje použití znaků, které byly označeny jako deprecated (čili zastaralé), jako příklad můžeme jmenovat tagy nebo <font>
    - Transitional – verze, která naopak užití deprecated tagů ještě povoluje
    - Frameset – verze, která rozděluje stránku do jednotlivých rámců (framů), které jsou na sobě vzájemně nezávislé
- HTML 4.01 – na dlouhou dobu poslední verze HTML, která vyšla v roce 1997

Po HTML 4.01 přišla éra XHTML, nového jazyka postaveného na základech jazyků XML a HTML. XHTML (název samotný vznikl složením zkratek XML a HTML) bylo jako oficiální norma představeno konsorciem W3C v roce 2002. Ačkoliv syntaxe jazyka XHTML je velmi podobná HTML, liší se v některých detailech. Uveďme si jich pár. Vzhledem k tomu, že XHTML je založeno na XML, všechny tagy musí být ukončeny. V případě tagů nepárových je tedy nezbytné provést tzv. „samoukončení“. To se provede znakem lomítka („/“) před ukončující špičatou závorkou.

```
<br> <!--- zápis v HTML --->
<br /> <!--- zápis v XHTML>
```

Zdrojový kód 11 Nepárové tagy v XHTML

XHTML také striktně předepisuje, že všechny tagy je třeba psát malými písmeny a hodnoty všech atributů se zapisují do uvozovek (případně apostrofů).

Obdobně jako v HTML 4, i v případě XHTML 1 existovali tři formy (Strict, Transitional a Frameset). Poslední oficiálně vydanou verzí XHTML byla verze XTHML 1.1 (která se oproti XHTML 1.0 Strict lišila pouze v detailech), existoval i pracovní návrh XHTML 2, ale jeho vývoj byl zastaven.

Po 15 letech od poslední verze HTML 4, byla představena nová verze HTML 5, tentokrát ovšem ne od konsorcia W3C, nicméně od organizace WhatWG, která sdružuje tvůrce webových prohlížečů. Ačkoliv název této nové (a zatím poslední verze) názvem odkazuje na HTML, je kompatibilní i s XHTML a mnoho věcí od něj přebírá (např. nutnost psát tagy malými písmeny a obsahy atributů do uvozovek). Kromě toho přináší nové možnosti, co se týče sémantiky obsahu (viz výše), podporu SVG, nové možnosti vkládání audia a videa přímo do dokumentů (skrze tagy <video> a <audio>) a např. také nové typy formulářových polí. (7)

3.1.2.2 CSS

Další technologie, o které se zmíním jsou kaskádové styly, označované zkratkou CSS (z anglického názvu Cascading Style Sheets). Jak již název technologie napovídá, CSS určuje výslednou podobu webové stránky. Základní myšlenka pro vznik kaskádových stylů bylo oddělení obsahu od jeho formy (a formátování). Jazyk HTML určuje strukturu obsahu a vlastní obsah, CSS pak určuje vzhled, jakým bude HTML stránka vykreslena prohlížečem.

CSS také vzniklo (a nadále vzniká) pod hlavičkou konsorcia W3C, přičemž první verze spatřila světlo světa v roce 1996. Ta obsahovala nástroje pro formátování textu, pozadí dokumentu, zarovnávání, barev, základní pozicování atp. Druhá verze, poprvé představena v roce 1998, rozšířila možnosti pozicování jednotlivých elementů (přibyla absolutní a relativní pozice), přidala možnosti překrývání jednotlivých elementů pomocí z-indexu. CSS3, doposud poslední major verze, pak přinesla možnosti CSS animací, oblých rohů, nové možností zarovnávání elementů a mnoho dalších změn.

Syntaxe a princip jazyka CSS

CSS určuje soubor vlastností, které budou platit pro různé části HTML dokumentu. Část HTML dokumentu, na kterou budeme daný styl aplikovat (selektor) se dá určit v jazyce CSS vícero způsoby:

1. Navázáním na všechny elementy stejného typu - např. na všechny nadpisy velikosti 3 (tzn. pro všechny HTML tagy <h3>)
2. Stylování na základě atributů
    a. Na základě atributu class – každý HTML element může mít neomezeně tříd uvedených v atributu class (oddělených od sebe mezerou)
    b. Na základě atributu id – každý HTML element může mít jedno id (uvedené v atributu id), přičemž id je v rámci HTML dokumentu unikátní
    c. Stylování na základě jiných atributů než třída nebo id
3. Stylování na základě pseudoelementů – pseudoelementy jsou značky pro stylování pouze části elementu (první písmeno, stav odkazu po najetí myši, lichý řádek tabulky apod.)

Samotná CSS deklarace má tři části, a to selektor, vlastnost a hodnotu. Selektor může obsahovat více vlastností. Vlastnosti se píšou obvykle pod sebe a ukončují se středníkem. Vlastnosti selektoru jsou pak obaleny složenými závorkami. Pojďme si jednotlivé typy selektorů ukázat na příkladech:

```
/* Stylování na element – platí pro všechny nadpisy h3 */
h3 {
font-size: 16px;
color: #bbb;
}
/* Stylování na třídu – platí pro všechny element, které mají v atributu
class hodnotu “bily-text” */
.bily-text {
font-size: 16px;
color: #fff;
}
/* Stylování na id – platí pro element, který má v atributu id hodnotu
predmluva*/
#predmluva {
font-style: italic;
}
/* Stylování pomocí pseudoelementu, stylování odkazu po najetí myši */
a:hover {
color: #fff;
}
```

Zdrojový kód 12 Ukázka stylování elementů v CSS

Selektory se pomocí čárek mohou také seskupovat:

```
/* Pokud chceme nastylovat nadpisy první – třetí úrovně, můžeme využít
tohoto zápisu */
h1, h2, h3 {
text-transform: uppercase;
}
```

Zdrojový kód 13 Seskupování selektorů v CSS

Zápis selektoru může být i zanořený (v rámci struktury stylovaného HTML dokumentu). Např. pokud chceme nastylovat všechny odkazy v hlavičce webu (uzavřené do tagu <header>), můžeme to provést takto:

```
header a {
      color: #fff;
      text-decoration: underline;
}
```

Zdrojový kód 14 Zanořený selektor v CSS

Zároveň, pokud bychom chtěli nastylovat všechny odkazy, které jsou v hlavičce webu a zároveň nejsou začleněny do žádného dalšího elementu (čili se jedná o přímé potomky), můžeme použít tento CSS selektor:

```
header > a {
      color: #fff;
      text-decoration: underline;
}
```

Zdrojový kód 15 Selektor pro přímého potomka

Jak se dá odvodit z výše uvedených příkladů, jeden element se dá popsat skrze mnoho různých selektorů. Zároveň téměř každý element má v každém prohlížeči již předdefinovaný styl. Aby prohlížeč věděl, kterému stylu má dávat přednost, existuje v CSS takzvaný „kaskádový“ princip. Platí, že vždy má přednost nejspecifičtější selektor (například header > a, je více specifický než header a, a ten má zase vyšší prioritu než a). Zároveň uživatelem definovaný styl má přednost před výchozím stylem prohlížeče. Toto chování se dá změnit užitím značky !important, která může být uvedena u libovolné CSS vlastnosti. (11)

Popisovat všechny vlastnosti CSS (je jich více než 350) by vzhledem k existenci příslušného manuálu vydaného konsorciem W3C bylo zbytečným nošením dříví do lesa, proto nezbývá než čtenáře právě do něj odkázat. (12)

Responsivní design a CSS

Mluvíme-li o kaskádových stylech a o moderním stylování webových stránek, měli bychom se alespoň okrajově zmínit o responsivním designu. Responsivní design představuje techniku, kdy se webová stránka přizpůsobuje rozlišení zařízení, na kterém je aktuálně prohlížena. Výhoda tohoto přístupu je nasnadě, místo vytváření několika verzí jedné webové stránky se vytvoří pouze jedna.

CSS 3 nám v tomto směru nabízí velmi účinný nástroj zvaný media queries. Media query se skládá z typu média (screen, apod.) a jedné či více podmínek vztahující se k médiu (často rozlišení v px). (13)

```
@media only screen and (min-width: 768px) {
      .container {
            width: 700px;
      }
}
```

Zdrojový kód 16 Ukázka media query

CSS preprocesory

Ačkoliv možnosti CSS jsou rozsáhlé, některé věci, na které je uživatel zvyklý z programovacích jazyků zde chybí (např. proměnné, cykly atp.). Tyto neduhy se dají vyřešit pomocí tzv. CSS preprocesorů. CSS preprocesory doplňují syntaxi CSS právě o výše zmíněné funkce. Zdrojové soubory CSS preprocesorů pochopitelně nejsou kompatibilní s definicí CSS, tudíž je prohlížeče nemohou zpracovat. Proto se jejich zdrojové kódy před nasazením nejprve kompilují do CSS. (14) Známými příklady CSS preprocesorů jsou SASS, LESS a Stylus.

CSS frameworky

CSS framework představuje předpřipravenou knihovnu stylů (a také HTML dokumentů a javascriptů), které usnadňují řešení typických problémů, se kterými se potýkají kodéři při realizaci téměř každé webové stránky. Mezi typické věci, které obvykle frameworky obsahují patří:

- **Grid systém** – systém rozdělení layoutu na sloupce (často 12), do kterých se pak umisťují jednotlivé prvky webových stránek (skrze určení na kolik sloupců, se má daný prvek roztáhnout, přičemž toto určení se může lišit pro různá rozlišení)
- **Předpřipravené komponenty** – typicky ve frameworcích najdeme již hotové mobilní hlavní menu (řešeno pomocí ikonky tří čárek, tzv. hamburger menu), předpřipravená modální okna, taby, případně slideshow a další
- **Předpřipravená typografie**
- **Resetování stylů** – napraví drobné nuance mezi základním vykreslováním některých elementů v různých prohlížečích

CSS frameworků existuje dnes větší množství, mezi nejznámější patří Foundation, Wellcome a zejména Bootstrap. (15).

3.1.2.3 Javascript

Poslední technologií, kterou si v rámci front-endových technologií společně představíme je Javascript. Javascript je skriptovací jazyk využívající objektové (i funkcionální) paradigma. Vznikl v roce 1995 ve společnosti Netscape. Javascript je dnes implementovaný a dostupný ve všech moderních prohlížečích (i když může být explicitně zakázán) a po HTML a CSS představuje třetí základní kámen technologie WWW.

Ačkoliv se Javascript řadí mezi objektové jazyky, jeho objektový model je odlišný od většiny jiných objektových jazyků (v podstatě všechno v Javascriptu je objekt, včetně funkcí a asociativních polí).

Javascript je jako jazyk stále vývoji, existuje mnoho jeho implementací a verzí, v posledních letech se nejvíc ujala verze ECMAScript.

Javascript bývá často rozšiřován různými frameworky (např. jQuery, Prototype, MooTools a jiné). Pokud nejsou použity, mluvíme o tzv. Vanila Javascriptu. Javascript představuje základ technologie AJAX (vývoj webových aplikací založený na asynchroním načítání dat bez nutnosti překreslení celé stránky).

Syntaxe Javascriptu

Pojďme se podívat na základní syntaxi Javascriptu. Jazyk Javascript je case-sensitive (čili v něm záleží na velikosti písmen), jednotlivé příkazy se oddělují středníkem. Obecně se dá říct, že syntaxe Javascriptu volně vychází ze syntaxe jazyka Java, se kterým se Javascript často zaměňuje. Nicméně zde podobnost mezi Javascriptem a Javou končí, jedná se o dva odlišné jazyky a technologie.

Začněme s obligátním příkladem „Hello, World!“. Co se výpisu textu týče, máme více možností. Buď použijeme konzoli prohlížeče (ke které se dostaneme skrze vestavěný objekt console, samotný výpis pak provedeme metodou log) anebo můžeme text vypsat přímo do těla webové stránky skrze vestavěný objekt document a jeho metodu write. Další variantou by bylo vypsat text do vyskakovacího okna pomocí funkce alert. Více napoví příklad:

```
// Příklady Hello, World!
document.write('Hello, world!'); // výpis do stránky
console.log('Hello, world!'); // výpis do konzole
alert('Helllo, world!'); // výpis do vyskakovacího okna
```

Zdrojový kód 17 Hello, world v Javascriptu

Deklarace proměnných se provádí pomocí klíčového slova var, konstanty (čili proměnné jejichž hodnotu nelze již v průběhu běhu programu přepsat) definujeme klíčovým slovem const. ECMAScript zavádí ještě klíčové slovo let, které má stejnou funkci jako var, nicméně platí jen v daném bloku kódu. Při deklaraci proměnné není třeba uvádět její datový typ. Pojďme si ukázat deklaraci proměnných na příkladu:

```
// deklarace proměnné
var x = 6;
let y = 9;
// deklarace konstanty
const pi = 3.14;
```

Zdrojový kód 18 Deklarace proměnných v Javascriptu

Z předchozí ukázky je vidět, jak se v Javascriptu zapisují komentáře (pomocí //). Zmínil jsem se, že u deklarace proměnné není třeba uvádět datový typ, pro úplnost si ale pojďme datové typy, které existují v JS vyjmenovat a představit:

- **boolean** – logická proměnná, může nabývat hodnot true nebo false
- **number** – reprezentace pro číselnou hodnotu, Javascript nerozlišuje, na rozdíl od jiných jazyků, celočíselné proměnné (integer) od těch s plouvoucí desetinnou čárkou (float, případně double)
- **string** – reprezentace řetězce
- **undefined** – všechny proměnné, které nebyly v rámci běhu programu inicializovány mají právě tento datový typ. Stejně tak, pokud se program odkáže na objekt, který neexistuje, bude mu navrácena právě tato hodnota

- **null** – hodnota, kterou má ve výchozím stavu přiřazenou každá proměnná, která již byla definována, ale nebyla ještě definována její hodnota
- **array** – pole (indexováno od 0)
- **object** – představuje objekt

Pro úplnost doplním, že pokud potřebujeme zjistit datový typ proměnné, pomůže nám s tím operátor typeof. Ve výše uvedeném přehledu jsme zmínili datový typ pole a object, příklad demonstruje jejich definice:

```
// definice pole
var ovoce = ['jablko','pomeranc','jahoda'];
// vypíše do konzole 2. hodnotu pole ovoce, čili pomeranc
console.log(ovoce[1]);

// definice objektu
var osoba = {jmeno: 'Jan', prijmeni: 'Novák', bydliste: 'Hradec
Králové'};

// vypíše do konzole hodnotu atributu prijmeni, čili 'Novák'
console.log(osoba.prijmeni);
```

Zdrojový kód 19 Pole a objekty v Javascriptu

Můžeme pokračovat v představování dalších základních konstruktů tohoto programovacího jazyka. Začněme, u podmínek:

```
// podmínka, která určí jestli je číslo, uložené v proměnné cislo
větší nebo rovna 0 a výsledek vypíše do konzole
if (cislo >= 0){
      console.log('Číslo je větší nebo rovno 0');
} else {
      console.log('Číslo je menší než 0').
}
```

Zdrojový kód 20 Podmínky v Javascriptu

Dalším konstruktem, který si představíme jsou cykly. Co se cyklů týče, jazyk Javascript nám nabízí více typů cyklů:

- **for** – cyklus s daným počtem opakování, skládá se z:
  - inicializátoru (nastaveném na úvodní hodnotu)
  - podmínky (k ukončení cyklu)
  - inkrementu
  - těla cyklu
- **for / in** – cyklus, který projde a zpracuje všechny atributy objektu (obdoba konstrukce foreach), skládá se z:
  - deklarace proměnných
  - těla cyklu
- **while** – cyklus s podmínkou na začátku, skládá se z:
  - podmínky
  - těla cyklu
- **do while** – cyklus s podmínkou na konci, skládá se z:
  - těla cyklu
  - podmínky

V cyklech můžeme použít příkazy break a continue. Příkaz break ukončí celý cyklus a continue ukončí aktuální iteraci a přejde na další.

```
// ukázka for cyklu
for (let x = 0; x < 5; x++){
      console.log(x);
}
// ukázka cyklu while
while (x < 65){
      x *= 2;
}
```

Zdrojový kód 21 Cykly v Javascriptu

Další důležitým prvkem jazyka jsou funkce. Ty se definují pomocí klíčového slova function. Do závorky za slovo function se pak uvádí jednotlivé parametry (bez datových typů). Pokud má funkce vracet hodnotu na závěr svého běhu, provede se to klíčovým slovem return. (16)

```
// funkce, která vynásobí dvě čísla a vrátí výsledek
function vynasob (cislo1, cislo2){
return cislo1 * cislo2;
}
```

Zdrojový kód 22 Funkce v Javascriptu

Události a přístup k DOM

Důležitou věcí, bez které se při vývoji webových aplikací s Javascriptem neobejdeme, jsou události a jejich handlery. Handlery představují způsob, jak v Javascriptu pracovat s HTML elementy na základě definovaných událostí. Pro manipulaci s HTML elementy se dnes používá rozhraní DOM (Document Object Model, opět standard pod správou konsorcia W3C), s nímž se setkáme ve všech prohlížečích.

Než se pustíme do samotných událostí, pojďme si nejprve ukázat, jak se v Javascriptu pracuje s jednotlivými HTML elementy. Příslušnou instanci elementu můžeme dostat skrze několik metod objektu document:

- document.getElementById("id") – vrátí instanci elementu, jehož id odpovídá parametru metody (pokud existuje)
- document.getElementsByClassName("class") - vrátí kolekci elementů, jejíž třída (atribut class) odpovídá parametru metody (pokud existují)
- document.getElementsByTagName("tag") - vrátí kolekci elementů, jejíž tag odpovídá parametru metody (pokud existují)

```
// změna HTML obsahu elementu zprava
var message = document.getElementById("zprava");
message.innerHTML = "Objednávka byla vložena!"
```

Zdrojový kód 23 Změna obsahu HTML elementu pomocí JS

Událostí existuje v Javascriptu velké množství, uveďme alespoň těch několik nejdůležitějších:

- onload – provádí se po tom, co prohlížeč dokončí renderování stránky
- onclick – provede se po kliknutí na daný element
- onchange – provede se po změně daného elementu
- onkeydown – vykoná se po stisknutí určené klávesy
- onmouseover – vykoná se po najetí myši na určitý element
- onsubmit – provede se po odeslání formuláře

Je čas opět na příklad, pojďme si ukázat, jak s pomocí události onclick dosáhneme toho, že po kliknutí na odkaz se vyvolá dialogové okno (pomocí již představené funkce alert). Nejprve si ukažme samotný skript, který se bude volat po kliknutí.

```
<script type="text/javascript">
function showAlert(){
            alert('Hello, world!');
}
</script>
```

Zdrojový kód 25 Událost po kliknutí na odkaz

Funkce je připravena, nyní je potřeba ji navázat na HTML element. Což můžeme provést pomocí atributu onclick.

```
<a href="#" title="Hello World!" onclick="showAlert()"
id="alert">Zobrazení okna</a>
```

Framework jQuery

V úvodu této kapitoly jsem se zmínil o tom, že při psaní v Javascriptu se často používá javascriptových frameworků, které zjednodušují samotnou práci (například při manipulaci s DOM, událostmi, animacemi a práci s AJAXovými požadavky) a rozšiřují jeho možnosti pomocí předpřipravených komponent. Představíme si, alespoň v krátkosti framework jQuery.

jQuery je rozšířený javascriptový framework, kontinuálně vyvíjený již od roku 2006, podporovaný dnes všemi masově používanými prohlížeči.

Základním stavebním kamenem tohoto frameworku je jQuery object, který se v normálním módu zapisuje pomocí znaku dolaru ($). Záměrně říkám v normálním módu, neboť kromě normálního módu obsahuje jQuery ještě tzv. no-conflict mode, kde se místo znaku dolaru používá řetězec jQuery. No-conflict mode existuje protože i jiné javascriptové knihovny používají značku dolaru, a tudíž by zde mohlo docházet ke konfliktům, které by měly za následek znefunkčení kódu.

Pojďme si nyní ukázat, jak se v jQuery pracuje s DOM elementy. Pokud chceme pracovat s libovolným elementem v HTML dokumentu, použijeme objekt $ a jako parametr mu předáme CSS selektor. Chceme-li v jQuery pracovat s celým dokumentem, použijeme řetězec document (předáme referenci na vestavěný objekt document). Obdobně bychom postupovali, pokud bychom místo dokumentu chtěli pracovat např. s oknem prohlížeče (zde by se použil objekt window). V příkladu níže je upravená podoba příkladu, ve kterém jsme měnili obsah elementu s id zprava. Chceme, aby se kód provedl hned poté, co prohlížeč dokončí vykreslování HTML dokumentu, proto náš kód obalíme do události ready, kterou zavoláme na document.

```
$(document).ready( function(){
      $("#zprava").html("Objednávka byla vložena!");
});
```

Zdrojový kód 26 Změna obsahu elementu v jQuery

Pro úplnost si můžeme ukázat, jak bychom v jQuery zapsali i druhý příklad (event listener po kliknutí na odkaz). (17)

```
$(document).ready( function(){
      $("a#alert").click( function(){
            alert("Hello, world!");
      });
});
```

Zdrojový kód 27 Definice událost v jQuery

3.2 Jazyk PHP

PHP je multi-paradigmatický (dá se v něm využít jak procedurální, tak objektové programování) programovací jazyk určený zejména pro vývoj webových aplikací (jejich backendu). Původně se jednalo o čistě skriptovací jazyk, jehož skripty se vkládaly přímo do HTML dokumentu, nicméně od svého vzniku v roce 1994 prošlo PHP překotným vývojem, který trvá dodnes. PHP je interpretovaný jazyk (tudíž k běhu skriptu není potřeba předchozí kompilace). Vychází z jazyka C, což se odráží i na jeho syntaxi.

Jak by asi pozornému čtenáři došlo, PHP je zkratka, jejíž význam není úplně jednoznačný, respektive, v průběhu let se měnil. Když v roce 1994 původní autor jazyka PHP, dánsko-kanadský programátor Rasmus Lerdorf řešil problém, jak zaznamenávat počty přístupů na jeho osobní stránky, napsal si v jazyce C knihovnu, která tento problém řešila. Tuto knihovnu (kterou postupně rozšiřoval) posléze nazval „Personal Home Page Tools“, jejíž název se později zkrátil na název PHP Tools. S příchodem PHP veze 3 (které můžeme považovat za první verzi PHP jako jazyka, tak jak ho známe dnes) vyvstala potřeba zkratku změnit, tak aby více reflektovala skutečnou podstatu jazyka. Od té doby je tedy zkratka PHP interpretovaná jako PHP: Hypertext Preprocessor (česky PHP: Hypertextový Preprocesor). Tento typ zkratek se nazývá rekurzivní (dalším příkladem by byla zkratka GNU, která je zkratkou spojení GNU's not UNIX, česky GNU není UNIX).

Jak již bylo zmíněno PHP je jazyk interpretovaný (opakem interpretovaného jazyka je jazyk kompilovaný), takže jeho zdrojový ke svému běhu nepotřebuje být nejprve zpracován kompilátorem (překladačem). Ke spuštění PHP skriptu je potřeba pouze tzv. PHP interpreter, který běží buď jako modul webového serveru (často serveru Apache) nebo je připojen jako CGI (případně FastCGI) skript. Pro zajímavost ještě doplním, že od verze PHP 5.1 je možné s pomocí rozšířením PHP-GTK psát v jazyce PHP i desktopové aplikace, zde už ovšem je kompilace nutná.

Jazyk PHP obsahuje rozsáhlé knihovny pro práci s řetězci, podporuje použití širokého množství různých typů databází, umožňuje práci s cURL a sockety, s velkou řadou internetových protokolů, XML soubory, obrázky, atp.

Je nezávislé na platformě, PHP interpreter je možné jej provozovat na všech hlavních systémech (Linux, Windows, macOS). (18)

3.2.1 Stručná historie jazyka PHP

Než se naplno pustíme do syntaxe a popisu jednotlivých možností jazyka PHP, pojďme se na chvíli zastavit u jeho historie. V minulém odstavci jsem zmínil, že vývoj PHP byl v mnoha ohledech překotný, abych toto svoje tvrzení dokumentoval, pokusím se vývoj PHP v kostce shrnout.

Jak již bylo zmíněno, základy PHP položil kanadsko-dánský programátor Rasmus Lerdorf v roce 1994. Tehdy se jednalo skupinu skriptů napsaných v jazyce C, zpracovávaných skrze CGI. Myšlenku na tvorbu na PHP mu vnukla potřeba měřit počet návštěv na svém osobním webu. Lerdorf postupně knihovnu vylepšoval a přidával do ní další nové funkce (skrze které se již například dalo komunikovat s databází, atp.). V této době se také vzala zkratka PHP (a to od slovního spojení Personal Home Page Tools, jak svou knihovnu nazval). Následně ovšem (když svou knihovnu na podzim roku 1995 přepsal) od této zkratky na krátkou dobu upustil a nazýval tuto svoji knihovnu Form Interpreter (zkratka FI). Proto se o těchto raných verzích PHP mluví jako o technologii PHP/FI.

Po zmíněném přepsání celé knihovny na podzim roku 1995 se jazyk PHP začal již vzdáleně podobat svému aktuálnímu stavu, respektive, některé funkcionality přetrvaly v jazyku až do dnešních dnů (srpen 2018). Syntaxe jazyka byla do velké míry podobná jazyku Perl.

Po dalším přepsání celého kódu knihovny zveřejnil v dubnu roku 1996 Rasmus Lerdorf další verzi svého projektu. Do názvu knihovny se vrátila zkratka PHP a samotný projekt se již více začal podobat programovacímu jazyku než knihovně. V této verzi (která byla označena číslovkou 2), byla přidána podpora mnoha databázových systémů, cookies, uživatelsky definovaných funkcí, apod.

PHP verze 3 představuje první verzi PHP, tak jak jej známe dnes, čili šlo už spíše o programovací jazyk místo knihovny. Za jeho vývojem stály tehdejší studenti univerzity v Tel Avivu, Andi Gutmans a Zeev Suraski. Jádro bylo opět kompletně přepsáno s důrazem na větší modularitu a možností stavět na PHP složitější webové aplikace. Mezi nejdůležitější novinky této verze patří podpora objektového

programování a více konsistentní syntaxe. Zároveň, jak jsem se zmínil v minulé kapitole, s verzí 3 přišla revize významu zkratky PHP (z původní Personal Home Page na PHP: Hypertext Preprocessor).

Po dvou letech (v roce 2000) od vydání PHP verze 3 přišla verze 4, která se liší od svého předchůdce zejména lepším výkonem, který byl dán novým enginem interpreteru jazyka, nazvaném Zend Engine (spojením křestních jmen tvůrců Zeev a Andi).

Verze 5 vyšla po dlouhém vývoji v roce 2004 a kromě nového enginu Zend 2.0 přinesla kompletně přepracovaný objektový model. Vývojová větev 5 je dodnes rozšířená a podporovaná.

Další verze PHP, která měla spatřit světlo světa a na které se usilovně pracovalo byla verze 6, jejíž hlavní novinkou měla být nativní podpora kódování Unicode. Bohužel PHP 6 se nikdy vydání nedočkalo a jeho vývoj byl ukončen jako slepá ulička ve prospěch aktuální větve PHP 7, jejíž první verze spatřila světlo světa v 3. prosince 2015. PHP 7 přináší několik zásadních novinek jako například možnost typování návratových hodnot metod, typování argumentů metod (i pro skalární datové typy), nové možnosti v objektovém modelu (např. viditelnost konstant), nové operátory atp. Zatím poslední verze s označením 7.2 vyšla 30. listopadu 2017.

3.2.2 Základní syntaxe

Představili jsme si základní fakta o jazyku PHP, stručně shrnuli jeho historii, nyní se již bez obav můžeme pustit do představení jeho syntaxe.

Stejně jako u Javascriptu začněme u příkladu „Hello, world!“. PHP kód se ukládá do souborů s koncovkou .php a samotný kód se obaluje mezi značky <?php a ?> (koncová značka se může případně i vynechat). V minulosti se PHP skripty často kombinovaly s HTML dokumenty (vznikaly tzv. dynamické HTML dokumenty, mezi programátory často označované jako tzv. spaghetti code (19)), ale vzhledem k nepřehlednosti takového kódu se od tohoto přístupu upustilo.

```
<?php
      echo "Hello, world!";
?>
```

Zdrojový kód 28 Hello, world v PHP

Jak je z výše uvedeného příkladu patrné, jednotlivé příkazy se v PHP oddělují středníkem a pro výpis na obrazovku, využíváme příkaz echo. Kromě příkazu echo, existuje v PHP i příkaz print. Mezi těmito příkazy byl historicky rozdíl, nicméně v dnešní době se chovají oba příkazy prakticky stejně. Z příkladu výše je dále také patrné že pro zadávání řetězců se v PHP používají apostrofy. Kromě apostrofů lze použít také uvozovky, přičemž rozdíl je v tom, že pokud do řetězce v uvozovkách vložíme proměnnou, vypíše se její obsah. Doplním ještě, že řetězce se v PHP spojují pomocí tečky. Syntaxe jazyka je case-sensitive.

Nyní se pojďme podívat, jak se v PHP definují proměnné, neboť je zde jedna specifičnost. Všechny začínají znakem $, a až po něm následuje řetězec názvu (nesmí ovšem začínat číslem). U proměnných se neuvádí jejich datový typ (ten se určí podle obsahu):

```
<?php
// nastavíme hodnotu proměnné
$variable = 125;
?>
```

Zdrojový kód 29 Definice proměnné v PHP

V ukázce výše si lze povšimnout ještě jedné věci, o které jsme se doposud nezmínili, a to jsou komentáře. Co se komentářů týče, nabízí nám PHP celkem tři způsoby, jak komentovat kód:

- Pomocí // - jednořádkový komentář, platí od // až po konec řádku
- Pomocí # - jednořádkový komentář, platí od # až po konec řádku
- Pomocí /* - víceřádkový komentář, který skončí až znakem */

Kromě konstant můžeme definovat i konstanty. To provedeme pomocí vestavěné funkce define. Tato funkce má dva parametry, název konstanty a její hodnotu.

```
<?php
// definice konstanty
define("SMYSL_ZIVOTA",  42);
// a její výpis
echo SMYSL_ZIVOTA;
?>
```

Zdrojový kód 30 Definice konstant v PHP

Konstanty definované funkcí define jsou konstantami globálními, tzn. dá se je využít v celé aplikaci, jak se definují konstanty tříd si ukážeme později.

3.2.3 Datové typy

Než budeme pokračovat v přehledu syntaxe jazyka PHP, dovolím si menší odbočku a představím datové typy, se kterými jazyk pracuje. Při definici proměnné se sice typ neuvádí, ale to neznamená, že by PHP typy nemělo. Jazyk obsahuje 4 skalární typy (obsahují jednu hodnotu, bez další vnitřní struktury):

- **boolean** – logická proměnná, nabývá hodnot true nebo false
- **integer** – celočíselná proměnná, umožňuje i zápis pomocí jiné než desítkové soustavy (pomocí prefixu 0b, můžeme zapisovat binární čísla, pomocí 0x zase hexadecimální)
- **float** – číselná proměnná s plouvoucí desetinnou čárkou
- **string** – proměnná určená pro uložení řetězců

Kromě 4 skalárních datových typů, zde existují i 4 složené datové typy:

- **array** – reprezentuje pole
- **object –** reprezentuje objekt
- **callable –** představuje anonymní funkci uloženou do proměnné
- **iterable –** představuje datovou strukuturu, kterou lze projít cyklem foreach

A konečně, kromě výše uvedených datových typů, existují v jazyce PHP i dva speciální typy, které uvedu rovněž:

- **resource** – speciální typ proměnné, který představuje referenci na externí zdroj (např. spojení s databází apod.)
- **null** – reprezentuje definovanou proměnnou, která nemá stanovenou hodnotu

Složené datové typy, si zaslouží trochu podrobnější rozebrání, proto si je pojďme představit na příkladech. Dovolím si pro tuto chvíli vynechat datový typ object, ke kterému se dostaneme později spolu s objektovým programováním v PHP. Začněme tedy u datového typu pole, které se v PHP dá definovat dvěma způsoby:

- pomocí klíčového slova array (starší způsob)
- pomocí hranatých závorek (novější typ)

```
<?php
// definice pomocí klíčového slova array
$ovoce = array("jablko", "hruska", "jahoda");

// definice pomocí závorek
$ovoce2 = ['jablko','hruska','jahoda'];

// k jednotlivým položkám pole pak přistupujeme pomocí indexů
v hranatých závorkách za názvem pole
echo $ovoce[0]; // vypíše hodnotu prvního prvku pole, tj. 'jablko'

?>
```

Zdrojový kód 31 Pole v PHP

Pole jsou automaticky indexovány integery od 0, nicméně pokud nám tento způsob nevyhovuje, můžeme si indexy definovat sami (a stvořit tak asociativní pole). Indexy můžou být definovány jak integerem, tak stringem, nicméně index musí být, pochopitelně, v rámci daného pole, unikátní.

```
<?php
$ovoce = [
      "a" => "jablko",
      "b" => "hruska",
      "c" => "Jahoda",
      ];

      // a výpis
      echo $ovoce["a"]; // vypíše "jablko"
?>
```

Zdrojový kód 32 Asociativní pole v PHP

Co se týče typu callable, tak tento typ umožňuje uložit do sebe anonymní funkci (tzv. closure).

```
<?php
/* do proměnné $druhaMocnina uložíme funkci, která umocní svůj
argument na druhou */
$druhaMocnina = function ($cislo){
      return $cislo*$cislo;
};

// volání
echo $double(2);
?>
```

Zdrojový kód 33 Closure v PHP

3.2.4 Výrazy a operátory

V kapitole o datových typech jsme si ukázali některé operátory, které nám jazyk PHP nabízí, pro shrnutí nabízím následující přehledy:

3.2.4.1 Aritmetické operátory

$a + $b	Součet proměnných $a a $b.
$a - $b	Rozdíl proměnných $a a $b.
$a * $b	Součin proměnných $a * $b.
$a / $b	Podíl proměnných $a a $b.
$a % $b	Zbytek po celočíselném dělení $a proměnnou $b.
$a ** $b	Umocnění $a na $b.
$a++	Inkrementování proměnné $a o 1
$b--	Dekrementování proměnné $a o 1

Tabulka 4 Aritmetické operátory

3.2.4.2 Logické, přiřazovací a porovnávací operátory

$a && $b, případně $a and $b	AND (logické a zároveň)
$a || $b, případně $a or $b	OR (logické nebo)
$a xor $b	XOR
!$a	Negace proměnné $a
$a == $b	$a je shodná s $b, ale typy nemusí být shodné
$a === $b	$a je shodná s $b, včetně datových typů
$a != $b	$a není stejná jako $b, bez porovnání typu
$a !== $b	$a není stejná jako $b nebo se neshodují jejich typy
$a > $b ($a < $b)	$a je větší (menší) než hodnota $b.
$a >= $b ($a <= $b)	$a je větší nebo rovna (menší nebo rovna) $b
$a <==> $b	Tzv. spaceship operátor, pokud je $a větší než $b vrátí hodnotu 1, pokud je větší $b vrátí -1, pokud jsou hodnoty stejné, vrátí 0.

Tabulka 5 Logické, přiřazovací a porovnávací operátory

3.2.4.3 Operátory při práci s řetězci a poli a další operátory

(string) $a . (string) $b	Spojí řetězec $a s řetězcem $b
(array) $a + (array) $b	Připojí na konec pole $a pole $b
$a instanceof Trida	Zjistí, zdali je objekt $a instancí třídy Trida (nebo jejího potomka)
($a >= $b) ? $c = $a : $c = $b	Ternární operátor, v případě pravdivosti výrazu v závorce provede část za otazníkem, v případě nepravdivosti část za dvojtečkou.

Tabulka 6 Operátory pro práci s řetězci a poli a jiné operátory

3.2.5 Podmínky

Nyní si již můžeme povědět něco o základních konstrukcích, které můžeme v jazyce PHP využít. Začneme u podmínek. Podmínka, stejně jak je zvykem v ostatních příbuzných jazycích (z rodiny C), začíná klíčovým slovem if. Za ním uzavřený do jednoduchých závorek následuje výraz, jehož pravdivost podmínka vyhodnocuje. Následuje jeden nebo více příkazů, které se provedou, pokud je výraz podmínky vyhodnocen jako pravda. Pokud je výraz jenom jeden, dá se podmínka napsat bez pomoci složených závorek.

```
<?php
      // zápis bez využití složených závorek
      if($a > $b) echo 'a je větší';

      // zápis s využitím složených závorek
      if($a > $b){
            echo 'a je větší';
      }
?>
```

Zdrojový kód 34 Podmínky v PHP

Podmínku lze doplnit i konstrukcí else, případně elseif, které se budou vykonávat, pokud bude výsledek prvního výrazu nepravda.

```
<?php
if($a > $b){
      echo 'a je větší';
} else if($a < $b) {
      echo 'b je větší';
} else {
      echo 'hodnota proměnných je stejná';
}
?>
```

Zdrojový kód 35 Podmínky v PHP II.

Složité podmínkové stromy je vhodné nahradit konstrukcí switch. Naopak pro jednodušší podmínky lze využít ternární operátor.

3.2.6 Cykly

PHP nám nabízí celkem 4 typy cyklů:

- Cyklus s podmínkou na začátku (while)
- Cyklus s podmínkou na konci (do-while)
- Cyklus s daným počtem (for cyklus)
- Cyklus foreach (projde celé pole, respektive objekt implementující interface Traversable)

Chování a syntaxe cyklů je obdobná jako v ostatních programovacích jazycích, proto si myslím, že postačí ukázka:

```
<?php
// cyklus while s podmínkou na začátku
$cislo = 2;
while($cislo < 10){
      $cislo += 5;
}
// cyklus for, výpis čísel od 1 do 10 s pomocí inkrementátoru
v proměnné $i
for($i = 1; $i <= 10; $i++){
      echo $i;
}
// cyklus foreach procházející pole čísel
$cisla = array(2, 4, 6, 7);
foreach ($cisla as $cislo) {
    echo $cislo;
}
?>
```

Zdrojový kód 36 Cykly v PHP

Stejně jako např. v javascriptu, i v PHP existují klíčová slova continue (ukončí aktuální iteraci cyklu a přejde na další) a break (ukončí vykonávání cyklu úplně).

3.2.7 Funkce

I v PHP najdeme funkce, ty se zde definují klíčovým slovem function, v závorce mohou mít libovolné množství parametrů. Pokud má funkce vracet nějakou návratovou hodnotu, užije se k tomu klíčové slovo return, které příslušnou hodnotu vrátí a ukončí vykonávání funkce.

```
<?php
// funkce, která umocní číslo na druhou
function druhaMocnina($cislo){
      return $cislo*$cislo;
}
?>
```

Zdrojový kód 37 Deklarace funkce v PHP

V kapitole o vývoji jsem se zmínil o tom, že s příchodem PHP 7 (respektive 7.1) můžeme definovat typy parametrů a návratový typ funkce.

```
<?php
// na začátku skriptu definujeme, že budeme typovat
declare(strict_types=1);

function druhaMocnina(int $cislo) : int {
      return $cislo*$cislo;
}

echo druhaMocnina(2); //vrátí 4
?>
```

Zdrojový kód 38 Deklarace funkce v PHP s užitím skalárních typů

3.2.8 Vestavěné funkce

PHP obsahuje poměrně rozsáhlou knihovnu vestavěných funkcí, které se dají použít kdekoliv ve skriptu. Tyto funkce jsou rozděleny do jednotlivých modulů, tudíž všechny nemusí být na každém serveru dostupné. Probírat všechny (nebo jenom jejich podstatnou část) by bylo zbytečným přepisováním oficiálního manuálu, proto čtenáře odkážu právě na něj. (18)

Kromě vestavěných funkcí obsahuje PHP i předdefinované proměnné, které umožňují práci např. s požadavky přišedšími skrze HTTP metody POST ($_POST), a GET($_GET), session ($_SESSION) atp.

3.2.9 Objektový model jazyka PHP

Poslední částí vztahujícím se k jazyku PHP, u které bych se rád zastavil je objektový model a objektové programování. Principy OOP v PHP jsou podobné jako v příbuzných jazycích (např. v Javě), nicméně existují zde určitá omezení a odlišnosti, které se nyní pokusím shrnout. Začněme u definice třídy.

Třída se v PHP definuje klíčovým slovem class. Na rozdíl třeba od Javy nebo C# se ovšem v PHP neurčuje viditelnost tříd. Třídy se sdružují do tzv. namespaců, což lze chápat jako obdobu jako balíčků v Javě, tedy způsob, jak předejít kolizím jmen tříd (každá třída by měla mít v aplikaci unikátní název). Platí pravidlo, že v jednom souboru s příponou .php by měla být definována právě jedna třída.

Stejně jako v jiných jazycích, třídy obsahují atributy, metody (zahrnující konstruktor a destruktor) a konstanty. Konstruktor může mít třída (na rozdíl od jiných programovacích jazyků) pouze jeden. Definuje se pomocí metody zvané __construct, která může mít libovolný počet parametrů, přičemž parametrům se dá nastavit výchozí hodnota null (a tím z nich učinit de facto parametry nepovinné), což může představovat určitou alternativu k nemožnosti přidat třídě více konstruktorů.

Atributy třídy se zapisují na začátek deklarace třídy. U atributů lze určit jejich viditelnost, máme na výběr celkem tři možnosti:

- **public** – atribut je viditelný odkudkoliv (čili jak ze třídy, tak z jejích potomků, tak mimo třídu), výchozí možnost
- **protected** – atribut je viditelný pouze ze třídy samé a z jejich potomků
- **private** – atribut je viditelný pouze ze třídy samé

Dále se určuje název atributu (unikátní v rámci třídy) a může se mu rovnou určit výchozí hodnota. Zároveň může mít deklarovaný datový typ. Pro definování metod platí stejná pravidla jako pro definování funkcí, které jsme si již představili. Zároveň se i u metod pochopitelně určuje viditelnost, stejně jako v případě atributů. Metoda může být také statická (pomocí klíčového slova static). Třídy od sebe mohou pochopitelně i dědit, a to pomocí klíčového slova extends.

```
<?php
declare(strict_types=1);
namespace Model;
class NewsArticle extends Article {
	private string $title;
	private string $content;

	public function __construct(string $title, string $content){
		$this->title = $title;
		$this->content = $content;
	}

	public function getTitle() : string {
		return $this->title;
	}
}
?>
```

Zdrojový kód 39 Třída v PHP

Z výše uvedeného příkladu je patrné, že referenci na sebe samu třída uchovává v pseudo-atributu $this (zároveň se tak nesmí žádný atribut jmenovat, jde o vyhrazené klíčové slovo). Pokud přistupujeme k atributům nebo metodám libovolného objektu, používáme k tomu operátor ->.

PHP má také vestavěnou podporu abstraktních tříd (skrze klíčové slovo abstract, přidané před klíčové slovo class) a podporu finálních tříd (pomocí klíčového slova final). Abstraktní třída, stejně jako v jiných jazycích, nemůže mít instance a může v sobě obsahovat abstraktní metody. Finální třída, naproti tomu nemůže mít žádné potomky (žádná třída z ní nesmí dědit).

Kromě tříd obsahuje PHP take podporu pro interface. Interface se definuje pomocí klíčového slova interface. Interface pochopitelně obsahuje pouze hlavičky metod. Pokud třída implementuje určitý interface, použije se zde klíčové slovo implements.

Další, u které je nutné se zastavit je trait. Trait je způsob, kterým PHP řeší omezené možnosti dědění (v PHP lze dědit pouze od jedné třídy). Trait vlastně představuje v podstatě knihovnu funkcí, kterou lze vložit do libovolné funkce pomocí klíčového slova use.

Klíčové slovo use se zároveň používá pro vytváření aliasů k jiným třídám, které třída může využívat. (20)

3.3 Frameworky v jazyce PHP

Ačkoliv samotná knihovna funkcí (která zahrnuje i předpřipravené třídy) je poměrně obsáhlá, neobsahuje žádnou předpřipravenou kostru aplikace, na které by se dalo stavět. Jistě, existuje možnost si takovou kostru připravit (a tím si defacto svůj vlastní framework vytvořit) anebo využít služeb nějakého již existujícího frameworku. Problémů, se kterými nám framework v jazyce PHP pomůže je celá řada, pojďme si jich alespoň pár vyjmenovat:

- **Autoloading tříd** – pokud využijeme některý z moderních frameworků, nemusíme se starat o načítání tříd a dbání na to, aby byly tyto třídy načítány ve správném pořadí
- **Návrhový vzor** – každý z moderních frameworků má již v sobě připravenou implementaci některého z masově rozšířených návrhových vzorů (typicky jde o MVC nebo MVP)
- **Databázová vrstva** – PHP sice již v základu obsahuje obálku PDO, nicméně její možnosti jsou do jisté míry nedostatečné (oproti ostatním platformám pro vývoj webových aplikací), frameworky tento nedostatek kompenzují
- **Šablonovací systém** – umožňuje uživateli nepsat tzv. spaghetti code a

Frameworků pro PHP existuje velké množství. Aplikaci, kterou budu vytvářet v rámci praktické části budu stavět na frameworku Nette, nicméně v jazyce PHP existuje daleko více frameworků. Proto si krátce dovolím alespoň pár z nich zmínit.

3.3.1 Zend framework

Zend framework se řadí mezi dlouhodobé stálice na poli PHP frameworků. Jeho vývoj probíhá od roku 2005. Doposud poslední verze frameworku, Zend Framework 3 obsahuje více než 60 balíčků, které se dají použít samostatně nebo dohromady. Architektura Zendu implementuje návrhový vzor MVC, klade důraz na automatizovaného testování, rychlost a bezpečnost. S více než 289 miliony instalací jde pravděpodobně nejpoužívanější PHP framework, který používají i velké společnosti. (21)

3.3.2 Symphony

Prakticky ve stejné době jako se začal rodit Zend, začal i vývoj jeho pravděpodobně největšího konkurenta, frameworku Symphony. Symphony, stejně jako Zend, obsahuje širokou paletu hotových knihoven a svým rozsahem se tak hodí i pro větší pro větší projekty. Symphony se výrazně od Zendu odlišuje přístupem k databázi, protože již v základním buildu využívá ORM, konkrétně knihovnu Doctrine. (22) Zároveň ve výkonnostních testech překonává Zend. (23)

3.3.3 Laravel

Laravel se řadí mezi mladší frameworky (vyšel až v roce 2012) a staví na jiných principech než předchozí dva zástupci. Jeho základní myšlenkou je jednoduchost a snaha o minimalizaci zdrojového kódu. Jako bonus nabízí vlastní ORM zvané Eloquent a šikovný systém pro správu rout. Používá MVC architekturu. (24)

3.3.4 Code Igniter

Poslední framework, u kterého se v tomto krátkém výčtu zastavíme je Code Igniter. Tento framework staví na podobných principech jako výše zmíněný Laravel, tzn. menší rozsah než Symphony a Zend, upřednostňující jednoduchá, ale efektivní řešení. Stejně jako Laravel, Zend a Symfony implementuje návrhový vzor MVC, nabízí nástroje na tvorbu formulářů, vlastní databázovou vrstvu a dbá také na bezpečnost. (25)

3.4 Nette framework

Nette je původem český, velice progresivní framework. Díky své zemi původu je populární zejména v Česku a na Slovensku, nicméně postupně začíná expandovat i do zahraničí. V anketě uživatelů serveru SitePoint.com v roce 2015 se dokonce umístil na 3. místě (ačkoliv si je třeba uvědomit, že velké množství hlasů pocházelo právě z Česka). (26)

Stejně jako většina moderních frameworků, i Nette se dnes skládá ze samostatných komponent, které se dají libovolně kombinovat (i s technologiemi z jiných frameworků). Nette podporuje moderní programovací techniky (DRY – don't repeat yourself, KISS – keep it short and simple), nabízí vlastní šablonovací systém Latte, debugger Tracy a efektivní databázovou vrstvu Nette\Database. Obsahuje nativní ochranu proti Cross-site scriptingu (XSS) pomocí striktního ošetřování všech řetězců, jednoduše použitelnou ochranu proti CSFR (Cross-Site Request), session hijackingu a dalším obvyklým typům útoků. V neposlední řadě nabízí robustní implementaci Dependency Injection, silný nástroj na vytváření formulářů, rozšíření možností tříd v PHP a dobrý výkon. Pro česky mluvící programátory představuje také nespornou výhodu rozsáhlá česká dokumentace a početná komunita, která se okolo frameworku pohybuje.

Protože v Nette frameworku bude postavena praktická část této práce je na čase se na jednotlivé součásti Nette podívat podrobněji.

3.4.1 Krátký pohled do historie Nette

Dovolím si opět začít malým shrnutím dosavadního vývoje Nette. Historie frameworku se začala psát už v roce 2004, kdy vznikla první, neveřejná, verze Nette frameworku. Poté trvalo dlouhých 5 let než hlavní (a tehdy jediný) vývojář Nette, český programátor David Grudl, poskytl svoji práci komunitě.

Po několik let bylo Nette do jisté míry „one-man-show" Davida Grudla, který zajišťoval veškerý vývoj. V poslední době se nicméně již komunita okolo frameworku změnila a vývoj se rozdělil mezi více lidí, které zastřešuje organizace Nette Foundation.

3.4.2 Architektura Nette

Nette implementuje návrhový vzor MVP (Model-View-Presenter), který je variací staršího návrhového vzoru MVC. Zásadní rozdíl mezi MVC a MVP tkví v tom, kdo zachytává uživatelovi vstupy. V případě MVC se o zachytávání vstupů stará výhradně Controller. V MVP zachytí uživatelův vstup součásti View (komponenty) a následně je předají do Presenteru k vyhodnocení a ke zpracování. Dalším rozdílem je to, že v klasické architektuře MVC má Model přímou vazbu na View, v MVP tato vazba není, tudíž všechna data z modelu proudí do view přes presenter. (27)

Model MVP v sobě zahrnuje tři oddělené vrstvy:

- **Model** – základní datová a funkční struktura aplikace. Obsahuje aplikační logiku, udržuje si svůj vnitřní stav a s ostatními vrstvami komunikuje pomocí svého rozhraní. Neví o existenci view, ani presenteru.
- **View** – zobrazuje uživateli data z modelu, která mu předá presenter. Zachytává uživatelovi vstupy a předává je ke zpracování presenteru.
- **Presenter** – zpracovává uživatelovi vstupy a následně volá příslušnou aplikační logiku (tj. model), jeho odpověď předá k vykreslení do view.

3.4.2.1 Presentery a jejich životní cyklus

Pojďme se detailněji zastavit u presenterů. Pokaždé když Nette zaznamená http požadavek, tak jej zavolá router. Router je objekt, který umí předat konkrétní http požadavek příslušnému presenteru ke zpracování (respektive jeho továrně, která zajistí, že se vytvoří instance daného presenteru). Presenter následně převezme přeložený požadavek a vytvoří odpověď (tzn. dotáže se modelu). Odpovědí nemusí být pouze HTML stránka, ale také např. XML soubor nebo i soubor.
Samotným presenterem se rozumí potomek třídy Nette\Application\UI\Presenter.

Presenter má definované metody, které představují akce, které presenter obsluhuje. Vytvořme si např. akci vypis, která vypíše všechny produkty e-shopu, a předejme z presenteru data do šablony.

```
use Nette\Application\UI\Presenter;
class TestPresenter extends Presenter {
      public function renderVypis() {
            $this->template->data =
                        $this->productManager->getProducts();
      }
}
```

Zdrojový kód 40 Nette: Předání dat do šablony

Jak je vidět, vytvořili jsme veřejnou metodu (akci), kterou jsme pojmenovali renderVypis. Ono slůvko render pro nás bude ještě důležité, neboť se jedná o jeden z možných typů akcí, další si představíme záhy. V těle naší nové metody si pak můžeme povšimnout způsobu předání proměnné do šablony, což provedeme tak, že si zavoláme šablonu pomocí atributu presenteru template, vytvoříme novou proměnnou (kterou pak v šabloně budeme volat jako $data) a přiřadíme do ní obsah metody getProducts() z modelu.

3.4.3 Životní cyklus presenteru

Kromě metod render<Název> umí presenter pracovat i s dalšími typy metod, které se volají v určitém pořadí. Pojďme si je představit. Jako první se samozřejmě zavolá konstruktor presenteru. Po konstruktoru presenter vykoná metodu startup(). Po metodě startup se začnou volat metody s názvem, který začíná action, což je obdoba nám již známého render s tím rozdílem, že se volá ještě předtím než se začne cokoliv renderovat (může se jednat například o přihlášení uživatele do systému a jeho následné přesměrování jinam).

Následně se volají takzvané signály čili subrequesty. Pojmenování těchto akcí začíná slovem handle. Tyto metody jsou určeny primárně pro komponenty a zpracování AJAX požadavků. Po signálech presenter zavolá metodu beforeRender(), která se hodí například pro nastavení šablony, která se zavolá v metodě render. Po metodě render se zavolá už pouze metoda shutdown(), která se vyvolá při ukončení životního cyklu presenteru. (28)

3.4.4 Šablonovací systém

Jak už bylo několikrát zmíněno Nette obsahuje svůj vlastní šablonovací systém Latte, který je podle mého názoru jednou z největších předností Nette. Využití šablonovacího systému je nanejvýš vhodné, chceme-li se vyvarovat tzv. spaghetti kódu (tj. kombinace PHP a HTML kódu), ačkoliv nás k tomu Nette vysloveně nenutí.

Očekává se, že každá akce presenteru bude mít svoji vlastní šablonu, v separátním souboru s příponou .latte. Latte představuje samostatnou komponentu a dá se použít i mimo framework.

3.4.4.1 PHP vs. Latte

Pojďme si uvést praktický příklad. Opět předpokládejme, že vyvíjíme e-shop a chceme vypsat do klasické bootstrapové struktury seznam produktů. Takhle nějak bychom to mohli provést ve spaghetti codu (kombinace HTML a PHP):

```
<?php if ($items){ ?>
    <?php $counter = 1; ?>
    <div class="col-lg-3" id="item-<?php echo $counter++; ?>">
    <?php foreach ($items as $item){ ?>
        <span class="title"><?php echo
htmlSpecialChars(mb_convert_case($item->title,MB_CASE_TITLE)); ?>
        </span>
        <span class="price"><?php echo $item->price; ?></span>
    <?php } ?>
    </div>
<?php } ?>
```

Zdrojový kód 41 Ukázka výpisu bez využití Latte

A nyní to samé s použitím Latte syntaxe:

```
  {if ($items)}
      {foreach $items as $item}
      <div class="col-lg-3" id="item-{$iterator->counter}">
            <span class="title">{$item->title|capitalize}</span>
            <span class="price">{$item->price}</span>
      </div>
      {/foreach}
  {/if}
```

Zdrojový kód 42 Latte syntaxe

Jak je vidět syntaxe Latte šablon je poměrně jednoduchá a intuitivní, její základní složkou jsou makra, která se uzavírají do složených závorek. Dále můžeme v šabloně přímo pracovat s předanými proměnnými z presenteru a to pomocí tzv. filtrů. Filtr připojíme k proměnným pomocí znaku |.

3.4.4.2 Makra

Základním stavebním kamenem Latte jsou makra. Makra nahrazují základní programátorské konstrukce (jako jsou podmínky, cykly, atp.). Jak jsme si již naznačili výše, makra se uzavírají do složených závorek, nicméně je třeba poznamenat, že existuje ještě jeden způsob, kterému se říká n:notace. Ta funguje tak, že se makro uvede jako speciální atribut HTML elementu, na kterém se dané makro vykonává. Upravme si nyní náš příklad z předchozí kapitoly do tvaru s n:notacemi: Nyní je na čase představit si některá konkrétní makra.

```
{if ($items)}
      <div class="col-lg-3" id="item-{$iterator->counter}"
n:foreach="$items as $item">
            <span class="title">{$item->title|capitalize}</span>
            <span class="price">{$item->price}</span>
</div>
{/if}
```

Zdrojový kód 43 Makra v Latte

- **cykly** – {foreach},{for},{while}, makra pro cykly jsou obohacena tím, že Nette automaticky vytváří iterátor, který počítá průchody cyklem. Dále je možné použít makra {breakIf} a {continueIf} pro podmínečné ukončení cyklu, respektive pro pokračování na další iteraci
- **podmínky** – {if}, {elseif}, {else}, zde si dovolím krátkou poznámku o použití makra {else} v rámci podmínky {if}, kdy je vždy třeba podmínku nakonec uzavřít, jak ukazuje další příklad:

```
{if $hodnota == 2}
Hodnota je dva.
{else}
Hodnota není dva.
{/if}
```

Zdrojový kód 44 Podmínky v Latte

Pomocí maker jsme schopni definovat i novou proměnnou, a to pomocí makra {var}, komponenty, což mohou být například formuláře, vykreslujeme pomocí makra {control}. Maker existuje samozřejmě ještě mnohem více, zde jsem si dovolil vypsat několik stěžejních.

Dovolte mi ještě nyní malou odbočku ke stavbě šablon. Je zcela běžné, že některé elementy webových stránek jsou pro všechny podstránky totožné (menu, hlavička, patička, apod.). To Latte řeší tak, že je definována jedna hlavní šablona zvaná @layout.latte, která do sebe pomocí speciálního makra {include} či {block} vkládá další šablony. V ostatních šablonách pak máme opět pomocí maker {block} definováno, co kam patří. Pojďme si to opět demonstrovat na našem známém příkladu e-shopu. Předpokládejme, že kromě šablony na výpis produktů a jejich ceny, máme ještě soubor @layout.latte, který vypadá takto:

```
<!DOCTYPE html>
<html>
<head>
      <meta charset="utf-8">
      <title>{block title}{/block} | Eshop</title>
</head>
<body>
      <header>
            <h1>E-shop</h1>
      </header>
      <main>
            {block content}{/block}
      </main>
</body>
</html>
```

Zdrojový kód 45 Layout v Latte

A naši předchozí šablonu, pak upravíme následovně:

```
{block title}Výpis produktů{/block}
{block content}
      {if ($items)}
      <div class="col-lg-3" id="item-{$iterator->counter}"
n:foreach="$items as $item">
            <span class="title">{$item->title|capitalize}</span>
            <span class="price">{$item->price}</span>
      </div>
      {/if}
{/block}
```

Zdrojový kód 46 Šablona v Latte s bloky

3.4.4.3 Filtry

Poslední věcí, u které se zastavíme při našem povídání o šablonách v Nette jsou filtry, na které jsme už narazili. Takže jenom pro připomenutí, filtry jsou funkce, která nám přímo v šabloně umožní upravit nebo přeformátovat data do výsledné podoby, kterou chceme uživateli ukázat. Zapisují se za data, která mají upravovat, oddělují se od nich pomocí |. Pojďme si to opět ukázat v praxi.

Pořád se zaobíráme e-shopem. Nyní jsme se rozhodli, že kromě názvu a ceny, chceme zobrazovat zákazníkovi i datum přidání výrobku do nabídky. Předpokládejme, že data bereme z databáze, kde je máme uloženy ve formátu rrrr-dd-mm, a my je chceme zobrazit v jiném formátu. Jak na to? Použijeme filtr date:

```
{block content}
      {if ($items)}
      <div class="col-lg-3" id="item-{$iterator->counter}"
n:foreach="$items as $item">
            <span class="title">{$item->title|capitalize}</span>
            <span class="price">{$item->price}</span>
            <span class="date">{$item->date|date:'j. n. Y'}
      </div>
      {/if}
{/block}
```

Zdrojový kód 47 Filtry v Latte

Filtr date očekává parametr, kterým mu předáme, jaký formát času si přejeme zobrazit. Užijeme syntaxe PHP funkce time(), nicméně pokud bychom chtěli využít notace s % (tj. použít parametry funkce date), tak je to také možné.

Pozornému čtenáři jistě neuniklo, že kromě filtru date je v ukázce použit ještě jeden filtr, a to filtr capitalize, který způsobí, že název produktu bude napsán v kapitálkách. Stejně jako u maker i zde si dovolím bodově zmínit několik dalších:

- **Truncate** – ožízne řetězec na požadovanou délku, pokud se řetězec zkrátí, automaticky přidá trojtečku (což se dá změnit zadáním druhého parametru)
- **Webalize** – upraví řetězec do podoby použitelné v URL
- **Lower**, **Uper** – udělá všechna písmena malá, respektive velká

Latte samozřejmě umožňuje i vytváření vlastních filtrů.

3.4.5 Formuláře

Dalším silným nástrojem Nette jsou formuláře, pro které má Nette opět speciální balíček. Nette\Forms (jak se balíček jmenuje) nám umožní vytvořit jak vlastní formulář složený z jednotlivých formulářových prvků, tak jim přiřadit validační pravidla, jednoduše je zabezpečit a zpracovat.

3.4.5.1 Vytvoření formuláře v presenteru

Prvním krokem k vytvoření formuláře je vytvoření jeho továrničky v presenteru (případně v jiné komponentě). Následně mu přiřadíme jednotlivá pole, validační pravidla a nezapomene přidat odkaz na funkci, která zpracuje jeho data.

Pojďme si tvorbu formuláře opět představit na příkladu. Předchozí příklady se vztahovaly k e-shopu, tak u toho zůstaňme i tentokrát. Představme si, že chceme v administraci e-shopu mít možnost přidávat nové produkty. Řekněme, že formulář bude mít 2 pole (název a cena). Formulář pojmenujeme productForm.

```
class HomepagePresenter extends Nette\Application\UI\Presenter {

// vytvoříme továrničku, která bude vracet formulářovou komponentu
      protected function createComponentProductForm(){

// nejprve si vytvoříme instanci třídy Form
            $form = new Nette\Application\UI\Form;

// nyní formuláři přidáme dvě textová pole pomocí metody addText
            $form->addText('title','Název produktu');
            $form->addText('price','Cena produktu');

// nesmíme zapomenout na submit tlačítko
            $form->addSubmit('submit','Přidat');

      // nyní už je třeba jen přidat odkaz na metodu, která formulář
      zpracuje a předat ji náš formulář, to provedeme přes atribut
      onSuccess, na závěr vrátíme formulář

            $form->onSuccess[] = array($this,'productFormSucceeded');
            return $form;
      }
  }
```

Zdrojový kód 48 Továrnička na výrobu formuláře

Tím máme formulář vytvořený, na co jsme ovšem zapomněli, tak to jsou validační pravidla. Těžko budeme chtít ukládat produkty bez ceny a bez názvu. To můžeme zařídit několika způsoby:

1. Můžeme zavolat na našem formulářovém prvků metodu setRequired
2. Nebo si můžeme vlastní pravidlo pomocí metody addRule

Samozřejmě nemusíme přidávat pokaždé jen textová políčka nebo submity, Nette disponuje podporou pro všechny formulářové prvky, které známe z HTML (textarea, select, radio, soubor, atp.), příslušné metody jsou k nalezení v dokumentaci, případně jsou k náhledu i v praktické části práce.

3.4.5.2 Zpracování formuláře

Formulář máme vytvořen, ochráněn proti prázdným hodnotám, nyní je třeba jeho data zpracovat. V minulém kroku jsme určili, že o zpracování dat z formuláře se postará metoda productFormSucceeded, nyní je třeba ji implementovat.

```
public function productFormSucceeded(Nette\Application\UI\Form $form){
      // uložíme si data do databáze
      $this->database->table('product')->insert(array(
            'title' => $form->values->title,
            'price' => $form->values->price,
      ));

      // vypíšeme uživateli hlášku a přesměrujeme uživatele
      $this->flashMessage('Produkt byl uložen do databáze');
      $this->redirect('Product:default');
}
```

Zdrojový kód 49 Zpracování hodnot z formuláře v Nette

Když se podíváme na tuto ukázku kódu, tak kromě vkládání do databáze, které si později spolu s databázovou vrstvou rozebereme trochu více, je třeba si představit ještě dvě další důležité funkce Nette, a to flash messages a přesměrování. Začneme tím prvním. Flash messages (česky můžeme přeložit jako zprávičky) představují systémový kontejner, do kterého můžeme vkládat zprávy (skrze metodu na presenteru), a které si za pomoci speciálního makra můžeme vypsat do šablony. Zpráva má omezenou životnost (3 sekundy).

Co se týče přesměrovávání, tak každý presenter v Nette má metodu redirect, která zařídí přesměrování na libovolnou akci libovolného presenteru v aplikaci. Do parametru metody se zadá název presenteru a dvojtečkou se oddělí příslušná akce. Pokud není název presenteru dán, má se za to, že se přesměrovává na aktuální presenter.

3.4.5.3 Vykreslení formuláře do šablony

Máme vytvořený formulář, umíme ho zpracovat, teď už nám jen stačí ho někam vykreslit, tzn. do šablony. Na to nám Latte nabízí hned několik možností. Nejjednodušší je použít základní makro pro výpis komponenty, a to control. V našem případě bude vypadat takto:

```
{control productForm}
```

Zdrojový kód 50 Výpis komponenty v šabloně

Pokud zavoláme makro control a jako parametr mu předáme formulář, Nette použije základní renderer formulářů, který nám vrátí formulář v přibližně následující struktuře:

```
<table>
<tr class="required">
    <th><label class="required" for="frm-name">Název
produktu:</label></th>

    <td><input type="text" class="text" name="title" id="frm-title"
value="" /></td>
</tr>

<tr class="required">
    <th><label class="required" for="frm-price">Cena:</label></th>

    <td><input type="text" class="text" name="price" id="frm-price"
value="" /></td>
</tr>
```

Zdrojový kód 51 Výstup výchozí rendereru formuláře

Tato struktura ovšem nemusí vždy vyhovovat, proto nám Nette umožňuje vypsat si formulář manuálně, po jednotlivých elementech, skrze příslušná Latte makra. Druhou možností je napsat vlastní formulářový renderer. Takto bychom manuálně formulář vypsali do struktury, kterou používá framework Bootstrap:

```
<form n:name="productForm" class="form-horizontal style-form">
	<div class="form-group">
		<label for="title" class="col-lg-2">{label title /}</label>
		<div class="col-lg-10">
			<input n:name="title" class="form-control">
		</div>
	</div>
	<div class="form-group">
		<label for="price" class="col-lg-2">{label price /}</label>
		<div class="col-lg-10“>
			<input n:name="price" class="form-control">
		</div>
	</div>
	 <div class="form-group">
		<div class="col-lg-offset-2 col-lg-10">
			<button class="btn" type="submit">Odeslat</button>
		</div>
	</div>
</form>
```

Zdrojový kód 52 Ruční výpis jednotlivých položek ve formuláři

3.4.5.4 Zabezpečení

Ochránit formulář psaný v Nette proti CSRF (čili Cross-Site Request Forgery), což je jeden z nejčastějších typů útoků, který uživatele donutí navštívit stránku, která pak skrytě udělá útok v aplikaci, ve které je uživatel přihlášen, není nikterak složité. Pro ochránění formuláře stačí přidat k definici formuláře následující:

```
$form->addProtection();
```

Zdrojový kód 53 Ochrana formuláře v Nette

3.4.6 Práce s databází

Nette obsahuje i vlastní vrstvu pro práci s databází, pojďme si ji představit. Respektive dokonce vrtsvy dvě: Database Core a Database Explorer. První zmíněná zajišťuje vlastní spojení a umožňuje pokládání dotazů v jazyce SQL. Database Explorer naproti tomu umožňuje práci s databází bez nutnosti pokládání dotazů.

Připojení k databázi lze provést buď ručně, jak je ukázáno v příkladu níže nebo se nastavení převezme z konfigurace (o té se zmíníme později):

```
use Nette\Database\Connection;
$connection = new Connection($dsn, $user, $password);
```

Zdrojový kód 54 Připojení k databázi v Nette

3.4.6.1 Základní práce s databází v Database Explorer

Připojení k databázi existuje, nyní si ukázat práci s Database Explorer. Database Explorer umí pokládat efektivní dotazy bez nutnosti psaní SQL dotazů. Základní metodou, kterou budeme potřebovat je metoda table. Té jako parametr předáme název tabulky se kterou se chystáme pracovat. Následně můžeme volat příslušné metody (select, update, delete), které představují obdobu operací v SQL. Ukažme si to na příkladu.

```
$this->database->table('product')->where('id',7)->delete();
```

Zdrojový kód 55 Ukázka práce s Nette\Database

Výše uvedený příklad vymaže z tabulky produktů, produkt s id 7. Jak je z příkladu patrné, nejprve se určí, se kterou tabulkou se bude pracovat, následně pomocí metody where (která reprezentuje SQL klauzuli WHERE) vybereme záznam s id 7. Zavoláním metody delete provedeme jeho smazání. Obdobně se bude postupovat, pokud budeme chtít provést operaci INSERT nebo UPDATE:

```
$this->database->table('product')->update(array(
      'title' => $form->values->title,
      'price' => $form->values->price,
))->where('id',7);

$this->database->table('product')->insert(array(
      'title' => $form->values->title,
      'price' => $form->values->price,
));
```

Zdrojový kód 56 Update a insert v Nette\Database

Database Explorer automaticky složí dotaz ve správném pořadí, proto je jedno, zdali metodu where zavoláme hned po metodě table() nebo až nakonec. Chceme-li vybírat pomocí primárního klíče, můžeme použít metodu wherePrimary. Metoda where přijímá buď dva parametry (název sloupce a hodnota) nebo pole (pokud je podmínek vícero). Ale to není všechno, můžeme použít i operátory. Více napoví následující tabulka:

Nette\Database	SQL
`$table->where("field", $value)`	`field = $value`
`$table->where("field", NULL)`	`field IS NULL`
`$table->where("field > ?", $val)`	`field > $val`
`$table->where("field", array(1, 2))`	`field IN (1, 2)`
`$table->where("field", $conn->table($tableName))`	`field IN (SELECT $primary FROM $tableName)`
`$table->where("field", $conn->table($tableName)->select('col'))`	`field IN (SELECT col FROM $tableName)`

Tabulka 7 Klauzule where v Nette\Database,
zdroj: https://doc.nette.org/cs/2.4/database-explorer

Pokud nepoužijeme metodu delete, update nebo insert, vrátí nám Nette sadu výsledků zabalenou v objektu třídy Selection. Třída Selection, implementuje interface Traversable (dá se projít pomocí cyklu foreach). Je složen z objektů typu ActiveRow, které reprezentují řádek v databázi. Kromě metody where existuje v Database Exploreru samozřejmě existují i metody limit(), order(), having(), atp. Tyto metody představují protějšek ke stejnojmenným klauzulím v jazyce SQL. Mimo

to, můžeme použít i metody count() ,min(), max(). Database Explorer umí vybírat a filtrovat i podle hodnot z jiných tabulek, postačí, aby databáze měla správně nastavené cizí klíče.

3.4.6.2 ActiveRow

Poslední naší zastávkou v povídání o Nette\Database bude zmínka o ActiveRow. Jak již bylo zmíněno, každý řádek, který nám Nette\Selection vrátí je instancí třídy ActiveRow, což je třída, která poskytuje standardní rozhraní pro přístup k vybraným sloupcům. Data můžeme číst dvěma způsoby:

```
echo $product->id;
echo $product['title'];
```

Zdrojový kód 57 Práce s ActiveRow

3.4.7 Dependency Injection

Dependency Injection je technika, která usnadňuje získávání závislostí (objektů), které třídy potřebují ke svému fungování, tak aby se v třídách nemusely vytvářet instance, což se může hodit například v momentě, kdy potřebujeme v rámci celé aplikace předávat databázové spojení (objekt, který jej zajišťuje), tak aby si jej každý objekt nemusel vytvářet sám. Dependency Injection odnímá zodpovědnost za vytváření instancí závislostí třídě a předává ji tomu, kdo danou třídu vytváří, což se škáluje, až se ke slovu dostane DI kontejner, který má přehled o dostupných službách z konfiguračního souboru aplikace a potřebnou referenci do třídy vloží.

Závislosti se v Nette dají do objektů předávat různými způsoby, nejobvyklejším přístupem je předávání pomocí konstruktoru. Ukažme si to na příkladu. V konfiguraci aplikace máme nastavené spojení s databází (reprezentované objektem třídy Context). Toto spojení nyní potřebujeme dostat do presenteru:

```
class TestPresenter extends Presenter {
      protected $database;
      public function __construct(Context $database){
            $this->database = $database;
      }
}
```

Zdrojový kód 58 Dependency Injection v praxi

Jak je vidět z příkladu, přidali jsem do konstruktoru našeho presenteru parametr, který říká, že presenter vyžaduje k vytvoření instance spojení s databází (reprezentované právě objektem Context). Když Nette vytváří instanci presenteru, DI mu ji předá. (29)

3.4.8 Konfigurace Nette

V minulé kapitole o DI jsme se zmínili o konfiguraci frameworku, proto se u ní také v krátkosti zastavím. Samotná konfigurace Nette je obvykle umístěna ve složce config v adresáři app (což se dá ovšem v zaváděcím souboru bootstrap.php změnit). Konfigurace je psaná v notaci NEON. V konfiguračním souboru můžeme registrovat jednotlivé služby (tak aby se o nich dozvěděl DI kontejner), určit výchozí presentery aplikace a v neposlední řadě také nastavit spojení s databází.

3.4.9 AJAX a Nette

Nette si umí velmi dobře poradit i s AJAX přístupem, tedy se způsobem manipulace s daty a překreslováním šablon bez nutnosti znovu načítat webovou stránku. Nette si umí svoje šablony rozdělit na takzvané snippety (pomocí stejnojmenných maker), které se pak invalidují a jejich obsah může být na pozadí překreslován. Jediné, co k tomu potřebujeme je obsluha akcí na front-endu. V případě praktické části práce je demonstrováno užití technologie Nittro.

3.4.10 Práce s obrázky

Práce v čistém PHP za použití GD knihovny není úplně jednoduchá. Nette nabízí skrze svoji třídu Nette\Utils\Image sofistikovanější rozhraní. Uveďme si alespoň jednoduchý příklad, kdy budeme chtít nahrát do aplikace fotografii a tu při ukládání ještě zmenšit na šířku 1024:

```
// vytvoříme instance třídy Image z souboru který v sobě obsahuje
resource
$image = \Nette\Image::fromfile($img);
// obrázek zmenšíme
$image->resize(1024, NULL);
// a výsledek uložíme
$image->save($this->dir.$id.".jpg",100);
```

Zdrojový kód 59 Práce s obrázky v Nette

3.4.11 Práce s e-maily

Stejně jako práce s obrázky, tak svá úskalí má i práce se emaily, zvláště pak s českou diakritikou. I zde může být Nette užitečné a nabízí nám, dvě třídy, které nám pomohou. Jednou z nich je třída Nette\Mail\Message, která obsahuje vlastní zprávu a třída SendmailMailer, která zajistí odeslání zprávy. Odesílání zpráv se dá taky jednoduše zkombinovat s Latte šablonami. Jak by tedy vypadalo odeslání emailu zákazníkovi z e-shopu?

```
use Nette\Mail\Message;
$mail = new Message;
$mail->setFrom('Franta <franta@example.com>')
    ->addTo('shop@eshop.cz')
    ->addTo('jirka@gmail.com')
    ->setSubject('Potvrzení objednávky produktu')
    ->setBody("Dobrý den,\nvaše objednávka byla přijata.");
```

Zdrojový kód 60 Vytváření e-mailu v Nette

A následné odeslání:

```
use Nette\Mail\SendmailMailer;

$mailer = new SendmailMailer;
$mailer->send($mail);
```

Zdrojový kód 61 Odesílání e-mailu v Nette

3.5 Databáze MySQL

Než přejdu k praktické části práce, dovolím si představit databázový systém, který bude výsledná aplikace používat. MySQL je jedním z nejpoužívanějších databázových systémů. Jedná se o technologii původně vyvíjenou švédskou společností MySQL AB. Ta byla v roce 2008 odkoupena společností Sun Microsystems, který zase v roce 2010 koupila společnost Oracle. MySQL je open-source technologie.

Jedná se o relační databázový systém (data jsou rozdělená do tabulek), se kterým se komunikuje, jak již jeho název napovídá pomocí jazyka SQL. Hlavní předností MySQL je její rychlost. MySQL není závislá na platformě, nabízí podporu tabulek, pohledů, procedur a triggerů.

4 Praktická část práce

V praktické části se budu věnovat vývoji aplikaci pro podporu projektového řízení pro společnost Meebio s.r.o., agenturu zabývající se online marketingem a vývojem webových řešení, zejména v oblasti e-commerce. Aplikace se bude věnovat výkazům práce jednotlivých pracovníků firmy, včetně statistik, které budou sloužit zejména k přípravě podkladů fakturace pro jednotlivé klienty společnosti. Aplikace bude rozlišovat různé uživatelské role.

Z technologického pohledu je potřeba, aby byla aplikace kompatibilní s ostatními technologiemi firmy (zejména kvůli dalšímu rozvoji aplikace a serverovému zázemí), proto bude pro vývoj použit jazyk PHP a framework Nette. Jako úložiště dat poslouží MySQL databáze.

4.1 Popis problému a jeho aktuálního řešení

Pro firmu v současné době pracuje přes 25 lidí, z nichž určitá část externě. Proto je pro firmu nezbytné, aby existovaly přesné výkazy práce, které se budou tvořit postupně, tak jak daní lidé pracují (při začátku práce na daném úkolu odstartují měřič času a vypnou ho v momentě, kdy je práce na úkolu hotova nebo přerušena) a ne pouze jednou (nebo několikrát) měsíčně. Firma zpravidla pracuje na více projektech najednou. Z výkazů o odvedené práci mohou pak projektoví manažeři vidět, v jakém stavu jsou jednotlivé úkoly na různých projektech a jak se daří naplňovat dopředu stanovené (a klienty schválené) odhady. Uživatelé, kteří systém používají se dělí do skupin podle jejich profese (grafici, back-end developeři, front-end developeři, marketingoví specialisté, atp.)

Tato problematika je momentálně řešena kombinací několika systémů:

- **Asana** – nástroj pro projektové řízení, kde jsou veškeré informace o jednotlivých projektech a kde se řeší případné problémy a komunikace teamu
- **Toggl** – time-tracking nástroj, kam uživatelé přidávají jednotlivé záznamy o času stráveném na různých úkolech, tyto záznamy jsou přiřazovány k jednotlivým projektům

- **Apple Numbers / Microsoft Excel** – předpřipravené tabulky, do kterých se doplňují hodnoty ze systému Toggl, a které následně srovnávají reálně odpracované hodiny s odhadovanými a ze kterých se následně tvoří podklady pro fakturaci

4.1.1 Problémy aktuálního řešení

Aktuální stav řešení této problematiky je neuspokojivý, a to z několika důvodů. Tím nejpalčivějším je pravděpodobně nutnost kopírování dat mezi jednotlivými systémy, a to zejména mezi systémem Toggl a tabulkami vytvořenými v Numbers, případně v Excelu. Zde vyvstává ještě další problém v tom, že projektoví manažeři nepoužívají stejný balík kancelářských aplikací, a proto zde často dochází k vzájemné nekompatibilitě jednotlivých tabulek. Celý tento proces je také velmi náchylný na chybu, neboť možnosti exportu ze systému Toggl jsou omezené.

Dalším závažný problém představují některé dílčí nedokonalosti aplikace Toggl (s ohledem na nastavené procesy ve společnosti), jedná se zejména o:

- **Možnost vkládání manuálního záznamu do výkazů** – problém nastává v momentě, kdy někteří spolupracovníci vkládají před uzávěrkou měsíce do výkazů práci za celý předchozí měsíc a nezapisují výkazy postupně, tak jak je stanoveno firemními pravidly, systém Toggl bohužel neumožňuje tyto ruční záznamy buď zakázat, nebo alespoň je označit, tudíž možnosti kontroly jsou v tomto případě velmi omezené
- **Nevyhovující struktura entit** – každý uživatel je v Togglu zařazen do pracovní skupiny podle svého profese (do tzv. workspace), jednotlivé projekty se pak řadí pod tyto workspace. Problém nastává v momentě, kdy na jednom projektu pracuje více lidí různých profesí (typicky např. grafik, front-end developer a back-end developer), protože se projekt se musí zakládat pro každý workspace zvlášť a při následném exportu se tento fakt nesmí opomenout
- **Dílčí úkoly v rámci projektu –** Toggl také neumožňuje přednastavit do projektu úkoly, ke kterým by uživatelé pouze přiřazovaly svoje časové

položky. Každý uživatel je tak nucen popsat svůj úkol znovu, a díky tomu opět dochází k nepřesnostem

4.2 Požadavky na nové řešení

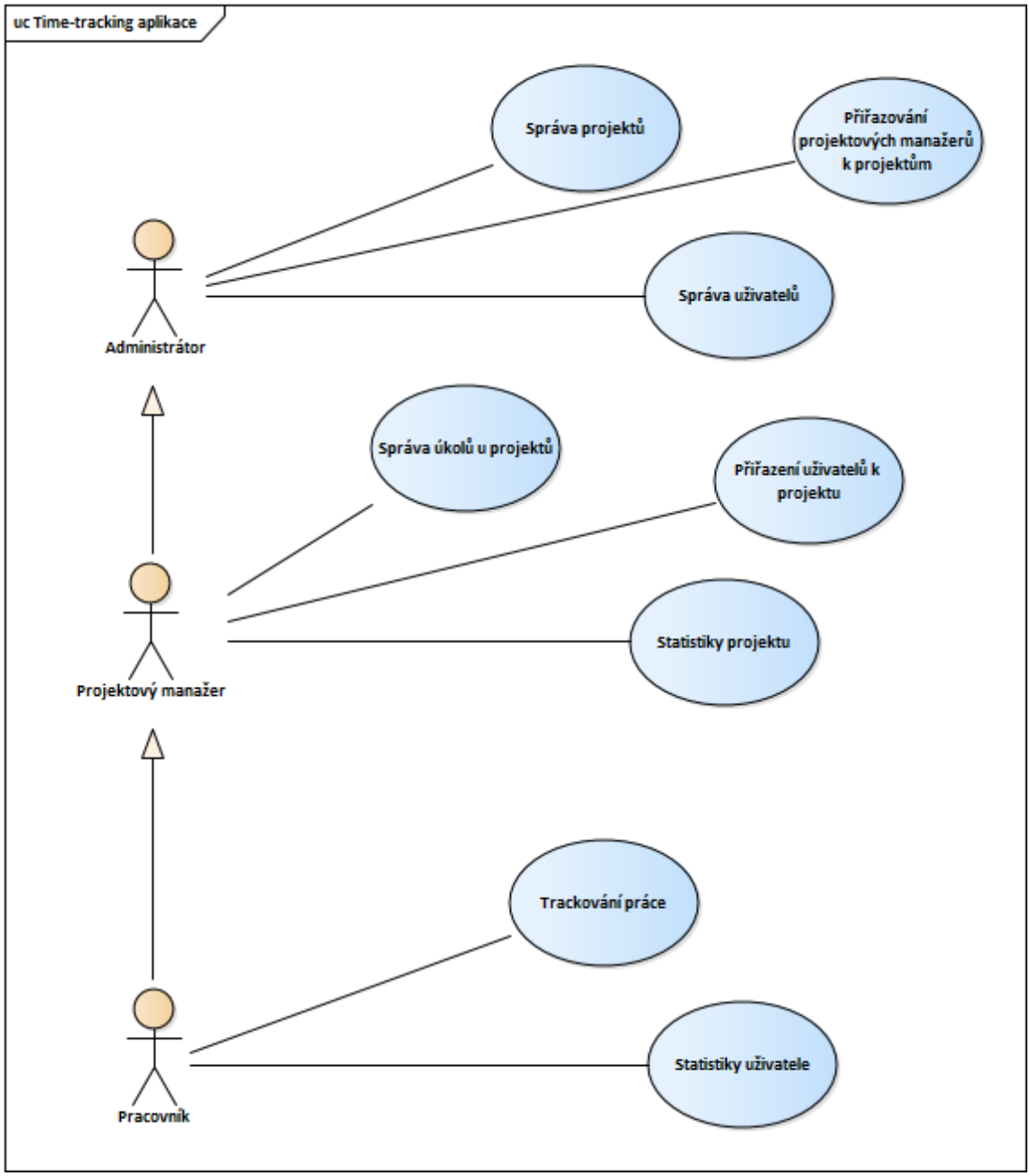
Nová aplikace by měla kromě spojení úkolů, které momentálně řeší dvě aplikace (Toggl a tabulkový procesor, Asana prozatím zůstane, tak jak je), vyřešit výše popsané problémy a zefektivnit celý proces a snížit množství chyb, které vznikají mechanickým opakováním různých úkonů. Aplikace bude vyvíjena pod pracovním názvem MeebioTimer.

Technologicky bude aplikace postavna na poslední dostupné verzi frameworku Nette, MySQL databázi. Poběží pod nejnovějším PHP 7.2 na jednom ze serverů společnosti (provozovaných na operačním systému Debian se serverem Apache). Vzhled aplikace bude vycházet ze open-source šablony AdminLTE, postavené na frameworku Bootstrap. Kromě řešení výše zmíněných problémů by měla také přinést nové funkce, které aktuální řešení bohužel neumožňuje. Jednotlivé požadavky na funkčnost aplikace budou dále rozebrány v analýze projektu.

Aplikace musí být vyvinuta tak, aby ji bylo do budoucna rozšiřovat o nové funkcionality a upravovat a vylepšovat na základě potřeb společnosti.

4.3 Use case model a popis uživatelských rolí

Základní požadavky zadavatele byly představeny, je čas si popsat jednotlivé uživatelské role a jejich případy užití. Nejprve si ukažme Use Case Model využívající notaci UML vytvořený pomocí nástroje Enterprise Architect.



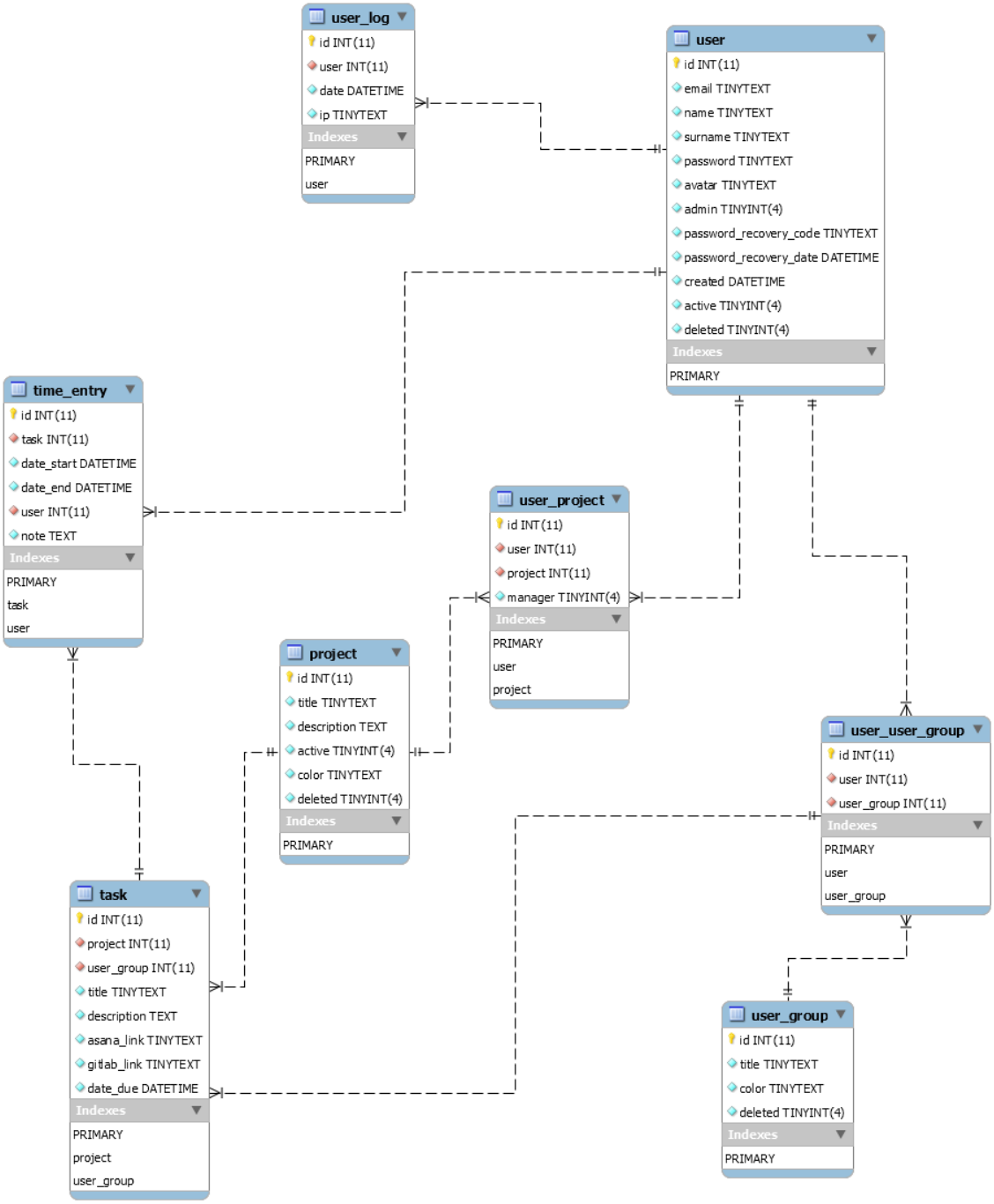
Obrázek 2 Use-case diagram aplikacce

Jak z diagramu vyplývá, systém počítá se třemi uživatelskými rolemi:

- **Pracovník** – reprezentuje zaměstnance firmy, který je přiřazen k projektům a do určité skupiny (nebo skupin) podle své profese. Může vykazovat práci k úkolům (u projektů, ke kterým je přiřazen). Také vidí statistiky své práce na všech projektech, na kterých pracoval.
- **Projektový manažer** – reprezentuje pracovníka firmy, který má na starosti řízení jednoho či více projektů (na dalších může pracovat jako „řadový“ pracovník, má u nich stejná práva jako Pracovník), může ke svým projektům přiřazovat pracovníky a tvořit úkoly včetně odhadů. Zároveň vidí statistiky všech pracovníků na projektech sobě přiřazených.
- **Administrátor –** spravuje celou aplikaci, může vytvářet nové skupiny uživatelů, přiřazovat do nich uživatele, vytvářet nové uživatele, editovat všechny stávající (včetně jejich deaktivace a smazání). Vytváří nové projekty a zároveň všechny projekty může spravovat jako jejich projektový manažer. Pochopitelně může také zaznamenávat čas ke všem úkolům ve všech projektech.

4.4 Popis struktury a databáze

Jednotlivé entity a vazby demonstruje následující návrh databáze, vytvořený v programu MySQL Workbench:



Obrázek 3 Diagram databáze

Dovolím si výše zmíněný diagram, alespoň v krátkosti, popsat, po jednotlivých tabulkách:

- **User** – představuje tabulku všech uživatelů v databázi, tabulka má vazbu na tabulku user_log, do které se zapisují jednotlivé přihlášení uživatele
- **User_group** – definuje uživatelské skupiny, protože jeden uživatel může být členem více skupin, existuje mezi tabulkami user a user_group vazebná tabulka, která realizuje tuto vazbu M:N
- **Project** – tabulka všech projektů, ke kterým lze vykazovat práci. Vzhledem k tomu, že uživatel může být přiřazen pro práci na víc projektech, existuje i zde vazebná tabulka mezi user_project
- **User_project** – vazebná tabulka mezi tabulkou user a project, kromě cizích klíčů na tyto dvě zmíněné tabulky obsahuje i sloupec, který rozhoduje, zdali je uživatel na konkrétním projektu v roli manažera nebo pouze pracovníka (uživatel může pracovat na některých projektech jako pracovník, jiné může obhospodařovat jako projektový manažer)
- **Task –** tabulka úkolů, které se přiřazují k jednotlivým úkolům a k jednotlivým uživatelským skupinám
- **Time_entry –** tabulka záznamů odpracovaného času, má vazbu na uživatele, který záznam provedl

4.5 Použité technologie a požadavky

Jak již bylo zmíněno, aplikace je psaná v PHP s využitím vlastností funkcionalit posledních verzí PHP. Pro její provozování je tedy nezbytný server s nainstalovaným interpreterem PHP (ideálně verze 7.2, nicméně verze 7.1 postačí) a MySQL databází. Aplikace je postavena na aktuálně poslední (srpen 2018) verzi Nette frameworku, tudíž je nezbytné, aby server splňoval i jeho požadavky, které jsou dohledatelné na oficiálních stránkách frameworku. Zároveň pro otestování je k dispozici i oficiální tester požadavků.

Knihovny (vendory) použité k vývoji aplikace (jejího backendu) jsou spravovány přes Composer, což je nástroj pro správu balíčků a jejich závislostí v jazyce PHP. Ovládá se pomocí jednoduchých příkazů psaných do terminálu. Composer generuje soubor (ve formátu JSON), který představuje seznam všech užitých balíčků a jejich závislostí. Pokud je v pořádku tento soubor (composer.json) není třeba při kopírování aplikace (ať už na produkční server nebo i do verzovacího systému) přenášet i veškeré knihovny, stačí na serveru spustit příslušný příkaz a vše zařídí Composer. Stejně tak v případě aktualizace knihoven.

Vzhled aplikace je založen na šabloně AdminLTE, která je postavena na CSS frameworku Bootstrap. Protože samotná šablona obsahuje velké množství souborů (CSS i Javascriptů) je vhodné je nenačítat jednotlivě, ale i tento proces automatizovat. Tento problém řeší v aplikaci nasazení nástroje Gulp, který má za úkol zminifikovat a spojit všechny Javascripty a CSS soubory v přesně daném pořadí. Gulp je nástroj založený na technologii Node.js (a jejíž instalace je nezbytná pro jeho funkčnost) sloužící k automatizaci sestavení aplikace (tzv. buildu). Instrukce Gulp se, stejně jako u Composeru, uchovávají ve formátu JSON (konkrétně v souboru gulpfile.js).

Protože framework Nette nemá ve své základní kostře žádnou součást, která by se postrala o zpracování AJAXových požadavků na straně klienta, je třeba ho vhodnou knihovnou doplnit. V tomto případě jsem se rozhodl doplnit Nette Javascriptovým frameworkem Nittro. Nittro je poměrně nová technologie (její první verze byla publikována teprve před dvěma lety) zajímavá tím, že je schopna (při

správném nakonfigurování) „zAJAXovat“ všechny požadavky a tím zásadně zlepšit UX aplikace.

Testovací verze aplikace je nasazená na adrese https://bp.dominikzilka.cz, stejně tak její zdrojové kódy. Vzhledem k tomu, že na aplikaci se bude pracovat i po odevzdání práce, doporučuji si stáhnout aktuální zdrojové kódy a aplikaci otestovat právě na této adrese.

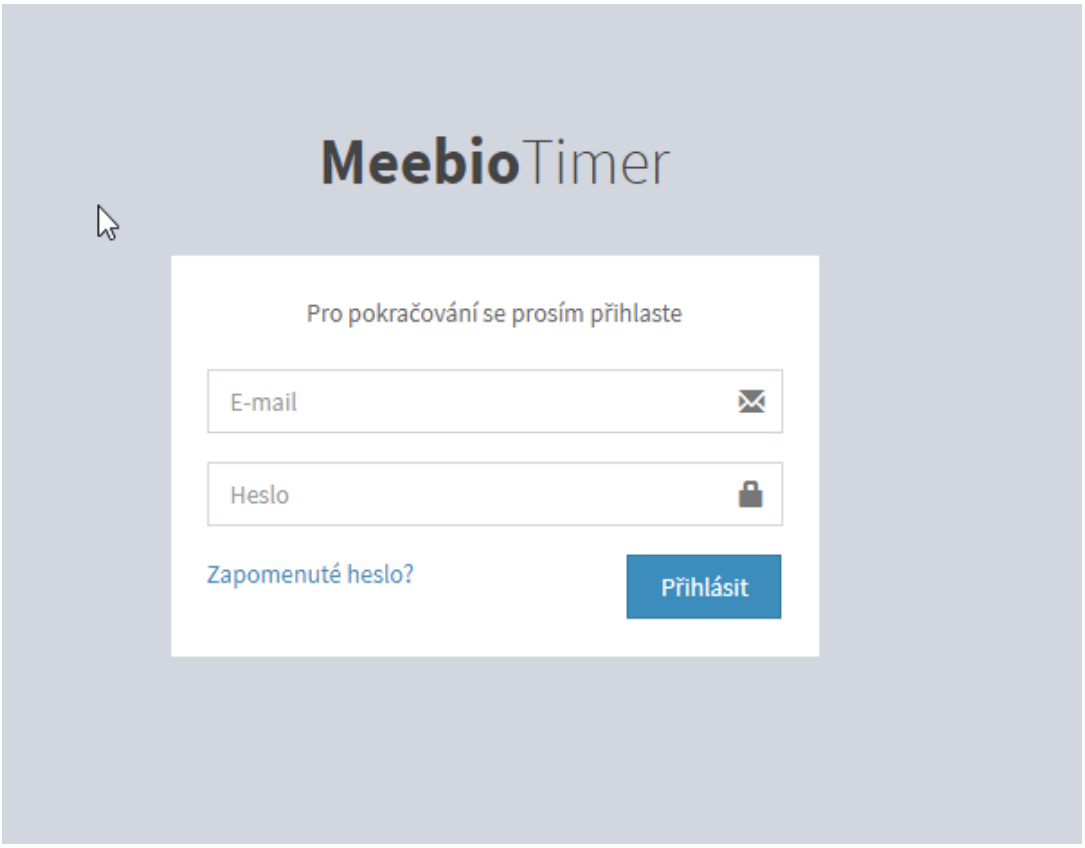
4.6 Popis struktury souborů

Souborová struktura aplikace vychází ze základní souborové struktury frameworku Nette. V kořenovém adresáři projektu se nachází tyto složky:

- **app** – představuje adresář, kde se nachází soubory aplikace
  - config – umístění konfigurace projektu
  - forms – formulářové komponenty
  - model – modely
  - components - komponenty
  - presenters – presentery
    - templates – šablony presenterů, včetně layoutu
  - router – umístění routeru
- **log** – slouží pro zápis odchycených výjimek, nutná práva pro zápis
- **node_modules** – zdrojové moduly Gulpu
- **resource** – assety, které Gulp minifikuje
- **temp** – cache paměť frameworku, nutná práva pro zápis
- **vendor** – všechny balíčky spravované composerem
- **www** – adresář obsahující index.php, jediný "viditelný" z prohlížeče
  - userdata – složka s daty, která mohou uživatelé nahrávat (avatary)
  - public – složka, kam Gulp ukládá zminifikované a sloučené assety

4.7 Použití aplikace

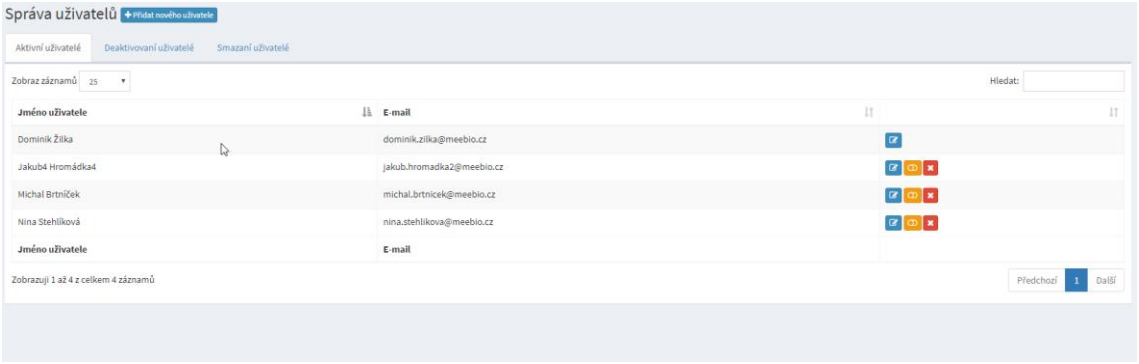
Při otevření aplikace se uživateli zobrazí přihlašovací obrazovka, kam uživatel zadá svoje přihašovací údaje. Pokud zapomene heslo, může si ho nechat obnovit (systém mu zašle na e-mail speciální url, kde si bude moci své heslo změnit)



Obrázek 4 Přihlašovací obrazovka

Po přihlášení se uživateli zobrazí dashboard, kde uvidí svoje poslední položky z časovače a bude moci zahájit nový odpočet. V sekci Moje reporty si uživatel může podrobně filtrovat svůj odpracovaný čas v minulosti. Pokud je uživatel zároveň projektovým manažerem některého z projektů, vidí sekci s projekty, kde může ke každému projektu, jehož je správcem spravovat úkoly a prohlížet si záznamy o projektu.

Pokud je uživatel v roli administrátora vidí zároveň i možnosti nastavení systému, tzn. správu uživatelů, kde může přidávat / mazat jednotlivé uživatele, měnit jejich oprávnění, přiřazené týmy, atp., správu pracovních skupin a projektů. Každý uživatel si může upravit svůj profil.



Obrázek 5 Správa uživatelů systému

4.8 Budoucnost aplikace a její další vývoj

Funkční základ aplikace zohledňující požadavky zadavatele je hotový, nicméně samotná práce na aplikaci tím nekončí, spíše naopak. V krátkodobém horizontu bude aplikace podrobena internímu testování více uživatelů, ze kterého vzejdou nové poznatky, které budou postupně zapracovávány. Rezervu dále spatřuji v návrhu uživatelského rozhraní, kde je prostor pro zjednodušení jednotlivých obrazovek, tak aby se s nimi jednodušeji a efektivněji pracovalo.

Z dlouhodobějších plánů bych zmínil možnost provázání aplikace s dalšími systémy používaných napříč společností (např. se systémem Asana, tak aby se úkoly nemusely zakládat dvojmo nebo s fakturačním systémem Fakturoid, aby systém byl schopen sám předpřipravovat faktury), zasílání notifikací jak administrátorům tak novým uživatelům. Také bych rád do aplikace připravil šablony projektů, tak aby se jednotlivé úkoly pro projekty nemusely pokaždé zakládat znovu.

5 Závěr

Práce měla stanoveny dva základní cíle, a to představit čtenáři základní principy tvorby webových aplikací s důrazem na jazyk PHP a framework Nette (a neopomenout ani technologie pro front-end) a následně demonstrovat možnosti využití těchto technologií na skutečném projektu. Při představování jednotlivých technologií jsem se snažil o maximální názornost skrze příklady a ukázky zdrojových kódů. Věřím, že teoretická část přináší alespoň základní vhled do problematiky vývoje v PHP a Nette frameworku a v kombinaci s představením front-endových technologií (HTML, CSS a Javascript) bude čtenář schopen bez problému schopen prohlubovat své znalosti studiem dalších zdrojů (zde mi nezbývá než doporučit oficiální dokumentaci frameworku Nette a jeho diskuzní fórum s velmi aktivní komunitou, co se týká PHP, tak zde mohu doporučit, dle mého názoru skvělou, knihu Jakub Vrány 1001 tipů a triků pro PHP).

Ve druhé části jsem se zabýval vývojem aplikace v Nette, která má za cíl usnadnit vnitřní fungování společnosti Meebio. Hlavním problémem při řešení návrhu aplikace byl bez pochyby správný návrh struktury aplikace, tak aby byla odstranila problémy aktuálního stavu ve společnosti a zároveň nepřinesla další, a aby nové řešení více reflektovalo způsob fungování firmy než řešení stávající. Výslednou aplikaci považuji za základ nového řešení, jehož vývoj odevzdáním bakalářské práce rozhodně nekončí. Podařilo se vyřešit základní problémy a požadavky (roztříštěnost dat napříč systémy, zjednodušení práce pro projektové manažery při vyhodnocování stavu jednotlivých úkolů a projektů), nyní aplikaci čeká testování. Největší rezervy spatřuji v UI, na které se do budoucna vývoj určitě zaměří. Dále mám v plánu připravit systém notifikací a zejména připravit automatizované exporty do fakturačního systému a prozkoumat možnosti propojení s aplikací Asana, která představuje hlavní nástroj pro projektový management ve společnosti. V průběhu práce jsem si mohl vyzkoušet nové technologie a postupy, které jsem doposud nepoužil (širší využití nástroje Gulp, otestovat si typování v PHP, atp.).

6 Seznam zdrojů

1. **Ndegwa, Amos.** What is a Web Application? *MaxCDN.com.* [Online] 31. Březen 2016. https://www.maxcdn.com/one/visual-glossary/web-application/.

2. **Adobe Corporate Communications.** Flash & The Future of Interactive Content. *Adobe Blog.* [Online] Červenec 7, 2017. https://theblog.adobe.com/adobe-flash-update/.

3. **Microsoft.** Overview of the .NET Framework. *Microsoft Docs.* [Online] 30. 03 2017. https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview.

4. **Kučera, František.** Java na webovém serveru: první web. *Zdrojak.cz.* [Online] Devel.cz Lab s.r.o., 15. Leden 2010. https://www.zdrojak.cz/clanky/java-na-webovem-serveru-prvni-web/.

5. **Ruby Comunity.** Official Ruby FAQ. *Ruby programming language.* [Online] [Citace: 16. 08 2018.] http://www.ruby-lang.org/en/documentation/faq/1/.

6. **Rails Guides tým.** Getting Started with Rails. *Ruby on Rails Guides.* [Online] https://guides.rubyonrails.org/getting_started.html.

7. **W3C.** The history of the Web. *W3C Wiki.* [Online] https://www.w3.org/wiki/The_history_of_the_Web.

8. —. HTML 5.2. *W3.org.* [Online] https://www.w3.org/TR/html52/.

9. **Cutts, Matt.** Google does not use the keywords meta tag in web ranking. *Google Webmaster Central Blog.* [Online] 21. Září 2009. https://webmasters.googleblog.com/2009/09/google-does-not-use-keywords-meta-tag.html.

10. **Seznam.cz.** Optimalizace. *Seznam Nápověda.* [Online] https://napoveda.seznam.cz/cz/fulltext-hledani-v-internetu/optimalizace-faq/#keywords.

11. **W3Schools.com.** CSS Introduction. *W3Schools.com.* [Online] Refsnes Data. [Citace: 05. 07 2018.] https://www.w3schools.com/css/css_intro.asp.

12. **W3C.** CSS Snapshot 2017. *W3C.org.* [Online] 31. Leden 2017. https://www.w3.org/TR/css-2017/.

13. **Media Queries.** *W3C.org.* [Online] W3C®, 19. Červen 2012. https://www.w3.org/TR/css3-mediaqueries/.

14. Sass Basics. ***Sass.*** [Online] [Citace: 21. 07 2018.] https://sass-lang.com/guide.

15. Introductuion. ***Bootstrap.*** [Online] [Citace: 23. 07 2018.] https://getbootstrap.com/docs/4.1/getting-started/introduction/.

16. Falanagan, David. ***JavaScript: The Definitive Guide, Fifth Edition.*** **Sebastopol : O'Reily, 2006.**

17. The jQuery Foundation. How jQuery Works. ***jQuery Learning Center.*** **[Online]** The jQuery Foundation, 14. Březen 2016. https://learn.jquery.com/about-jquery/how-jquery-works/**.**

18. the PHP Documentation Group. PHP: Manual. *PHP.* [Online] The PHP Group. [Citace: 2. Srpen 2018.] http://php.net/manual/en/.

19. Vrána, Jakub. ***1001 tipů a triků pro PHP.*** Brno : Computer Press, a.s., 2010.

20. The PHP Group. Classes and Objects. ***Manual.*** [Online] The PHP Group. [Citace: 6. Srpen 2018.]

21. About. ***Zend framework.*** **[Online] Zend.** [Citace: 8. Srpen 2018.] https://framework.zend.com/about.

22. Symfony SAS. Symfony Documentation. [Online] [Citace: 08. Srpen 2018.] https://symfony.com/doc/current/index.html#gsc.tab=0.

23. InfoDroid. PHP Benchmarks. [Online] [Citace: 11. 08 2018.] http://www.phpbenchmarks.com/en/comparator/frameworks.html.

24. Laravel team. Laravel - The PHP Framework For Web Artisans. [Online] [Citace: 10. Srpen 2018.] https://laravel.com/docs/5.6.

25. British Columbia Institute of Technology. CodeIgniter Features. ***CodeIgniter 3.1.9 documentation.*** [Online] 12. Červen 2018. https://www.codeigniter.com/user_guide/overview/features.html.

26. Skvorc, Bruno. The Best PHP Framework for 2015: SitePoint Survey Results. ***SitePoint.*** [Online] 30. Březen 2015. https://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/.

27. Everything You Wanted To Know About MVC and MVP But Were Afraid To Ask. ***You've Been Haacked.*** [Online] 16. Červen 2008.

https://haacked.com/archive/2008/06/16/everything-you-wanted-to-**know-about-mvc-and-mvp-but.aspx/.**

28. Nette Foundation. MVC aplikace & presentery. *Nette Framework.* [Online] 12. 08 2018. https://doc.nette.org/cs/2.4/presenters.

29. —. DI: Získávání závislostí. *Nette framework.* [Online] [Citace: 12. Srpen 2018.] https://doc.nette.org/cs/2.4/di-usage.

30. —. Dokumentace. ***Nette framework.*** **[Online]** Nette Foundation. [Citace: 12. Srpen 2018.] https://doc.nette.org/cs/2.4/.

Přílohy

1. CD se zdrojovými kódy

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2017/2018

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Žilka Dominik	Suchá 96, Nechanice - Suchá	I14174

TÉMA ČESKY:

Vývoj aplikace pro podporu projektového řízení

TÉMA ANGLICKY:

Application development for project management support

VEDOUCÍ PRÁCE:

Ing. Jan Petružálek - CIT

ZÁSADY PRO VYPRACOVÁNÍ:

Cílem práce je vytvořit aplikaci, která usnadní projektové řízení ve vývojové společnosti malé až střední velikosti. Bude sloužit zejména k zaznamenávání odpracovaného času na různých projektech, vyhodnocování skutečně odpracovaného času proti času odhadovanému. Aplikace bude webová, vyvinutá podle best-practises v oblasti vývoje webových aplikací. Součástí práce bude i rešerše, která bude pokrývat oblast webových technologií pro vývoj aplikací, analýza sofwaru a vlastní implementace.

Cíle:
1. Vypracovat literární rešerši o vývoji webových aplikací, seznámit se s dostupnými technologiemi v jazyce PHP, vybrat vhodnou technologii a tu podrobněji představit.
2. Seznámit se s dostupnými nástroji pro projektové řízení a stručně je popsat
3. Analyzovat potřeby společnosti a navrhnout vlastní řešení a mechanismy softwaru
4. Implementace navrženého řešení
5. Testování implementovaného řešení, jeho optimalizace a návrhy na vylepšení do budoucna

Stručná osnova:
1. Úvod
2. Literární rešerše
   a. Webové aplikace a dostupné technologie pro jejich vývoj
   b. Jazyk PHP a MySQL databáze
   c. Framework Nette a jeho principy
   d. Dostupné nástroje pro projektové řízení
3. Cíle práce
4. Analýza potřeb společnosti a návrh řešení
   a. Popis aktuální situace ve společnosti
   b. Požadavky společnosti na nové řešení
   c. Analýza a návrh vlastní aplikace
5. Implementace řešení
   a. Popis řešení jednotlivých úkonů
6. Testování, optimalizace a dokumentace
   a. Práce se softwarem
   b. Představení uživatelských rolí
   c. Testování
   d. Návrhy na vylepšení
7. Shrnutí výsledků
8. Závěry a doporučení
9. Seznam použité literatury

SEZNAM DOPORUČENÉ LITERATURY:

1. CUI, Wei, et al. The research of PHP development framework based on MVC pattern. In: Computer Sciences and Convergence Information Technology, 2009. ICCIT'09. Fourth International Conference on. IEEE, 2009. p. 947-949.
2. KERZNER, Harold; KERZNER, Harold R. Project management: a systems approach to planning, scheduling, and controlling. John Wiley & Sons, 2017.
3. VRÁNA, Jakub. 1001 tipů a triků pro PHP. Computer Press, Albatros Media as, 2017.
4. ROMER, Michael. PHP Persistence: Concepts, Techniques and Practical Solutions with Doctrine. Apress, 2016.
5. Grudl, David. "Nette Framework: MVC & MVP." Zdroják. cz (2009).

Podpis studenta: Dominik Šilla **Datum:** 15.2.2018

Podpis vedoucího práce: **Datum:** 15.2.2018