VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019 Lukáš Babinec

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Lukáš Babinec**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Siemens s.r.o., IT
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Lukáš**

Konzultant bakalářské práce: Ing. Tomáš Kresta

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 24. dubna 2019 .....

Prohlášení zástupce spolupracující právnické osoby

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

Siemens, s.r.o.
o.z. Elektromotory Frenštát
Markova 952
744 01 Frenštát p.R.

Ve Frenštátě p. R. dne: 25.4.2015

Ing. Tomáš Kresta

Mé poděkování patří Ing. Petru Lukášovi za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování bakalářské práce věnoval. Chtěl bych také poděkovat Ing. Tomášovi Krestovi, za umožnění praxe a Tomášovi Šafránkovi za podporu při jejím plnění.

Abstrakt

Tato práce popisuje průběh konání odborné praxe ve firmě Siemens, s. r. o. Zaměřuje se především na oblast vývoje webových aplikací, analýzu dat ve firemním informačním systému AM/IT a využití rozšířeného podnikového informačního systému SAP. Soustředí se především na ukázky základních principů vývoje webových aplikací v ASP.NET i za pomocí Kendo UI. Většina problémů, které byly řešeny, prošly celým procesem vývoje od jednání se zákazníkem, analýzou, vývojem až po samotné nasazení do produkce. V závěru práce jsou shrnuty znalosti a principy, kterých bylo při absolvování praxe nabyto.

Klíčová slova: bakalářská práce, Kendo UI, SAP, ASP.NET, MVC, agilní programování

Abstract

This thesis describes the practice at the Siemens, s. r. o. It focuses mostly on the field of web development, data analysis on the internal information system called AM/IT and wide used corporate information system SAP. The thesis mostly displays parts of the main principles of web development in ASP.NET framework with the use of Kendo UI. Most of the problems, that were solved, went through the whole process from the talk with the client, analysis, development and to the deployment to the production. In the end of the thesis, there are summarized principles and knowledge, which were gained in the process of the practice.

Key Words: bachelor thesis, Kendo UI, SAP, ASP.NET, MVC, agile programming

Obsah

Seznam použitých zkratek a symbolů

ASP	–	Active server pages
SAP	–	Systems - Applications - Products in data processing
HTML	–	Hyper Text Markup Language
AM/IT	–	Aplication management/information technology
API	–	Application Programming Interface
CLR	–	Common Language Runtime
HTTP	–	Hypertext Transfer Protocol
MVC	–	Model–View–Controller
XML	–	Extensible Markup Language
ERP	–	Enterprise resource planning
ABAP	–	Advanced Business Application Programming
RFC	–	Remote function call
UI	–	User Interface
DOM	–	Document object model
KPI	–	Key performance indicator

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Tato bakalářská práce popisuje průběh absolvování individuální odborné praxe ve firmě Siemens s. r. o, ve které je zaváděn průběžný proces digitalizace. Schopnost mít celý výrobní proces zaznamenán v digitální podobě a ne pouze na papíře nám umožňuje kontrolu nad jeho samotným průběhem. Mít digitální povědomí o tom, kde je která součástka naskladněna, který zaměstnanec úkon s ní provedl a na kterém stroji, je v dnešní době k nezaplacení.

Všechny tyto digitální informace nám dále umožňují analýzu procesů, za účelem zlepšení efektivity výroby produktu. Umožňují nám přehledný náhled k historickým datům. Také usnadňují naše každodenní činnosti ve výrobním procesu.

Společnost s pobočkou ve Frenštátě pod Radhoštěm se zabývá výrobou asynchronních motorů. Byl jsem členem týmu, jehož úkolem bylo pro tuto pobočku vyvíjet a spravovat interní aplikace.

V této odborné práci se budu především věnovat tomu, jak je tento princip digitalizace ve firmě Siemens s. r. o. zaváděn a jak jsem se mohl na tomto zavádění podílet. Velká část této digitální stránky výroby přísluší podnikovému informačnímu systému SAP, jehož stručný popis je rovněž součástí práce.

Jedno témat na kterém jsem pracoval, byl informační systém AM/IT. Systém, který slouží ke koordinaci a přehledu práce jednotlivých vývojářů. Dále nabízí také rozšířené analytické nástroje pro přehledy, tvorby reportů a jiné možnosti. Mým úkolem v tomto systému bylo provést několik analýz dat, které byly následně převedeny do prezentovatelné vizuální podoby. U velké většiny webové prezentace bylo využito 'frontendového' rozhraní Kendo UI. Toto rozhraní jsem využil několikrát během své praxe.

Komponentu s názvem Kendo editor jsem implementoval i ve firemní aplikaci Success Story. Během mé praxe bylo zadavateli projektů vytvořeno několik požadavků na rozšíření této aplikace na kterých jsem se podílel. Součástí mého úkolu bylo i nahrazení uživatelského rozhraní novou verzí a bylo i upraveno množství a typ vstupních dat. V práci jsem se rozhodl popsat proces implementace on-line textového editoru.

2 O společnosti Siemens, s. r. o.

2.1 Profil společnosti

Skupina Siemens Česká Republika je jednou z největších elektrotechnických firem v Česku, na českém trhu působí již přes 125 let. Zaměstnává přes 13 000 zaměstnanců. Skupina Siemens Česká Republika je součástí globálního koncernu Siemens AG. Jeho mezinárodní vedení sídlí v Berlíně.

Historie Siemens AG sahá až do roku 1847, kdy byl 1. října založen Wernerem von Siemensem. Werner von Siemens, byl vynálezcem telegrafu, který používal střelku kompasu k ukazování písmen na desce, místo tehdejšího využití Morseovy abecedy.Společnost se zaměřuje na oblast elektrifikace, automatizace a digitalizace. Mé konkrétní působení bylo v Siemens, s. r. o., odštěpný závod Elektromotory Frenštát. Tento odštěpný závod patří mezi jednoho ze světových dodavatelů nízkonapěťových asynchronních elektromotorů. Takovýto motor můžeme nalézt v různých čerpadlech, kompresorech a klimatizačních jednotkách.

2.2 Pracovní zařazení

Do firmy jsem nastoupil jako .NET vývojář interních WEB aplikací, ve firmě je upřednostňován jiný interní název pro tuto pozici a to "Programátor-Analytik". Součástí praxe bylo také absolvování schůzek s interními zákazníky, kteří sdělovali své požadavky na nové aplikace nebo požadovali další rozšíření již stávajících aplikací o novou funkcionalitu. Probíhaly diskuze o časových náročnostech jednotlivých požadavků, diskuze nad firemními daty pro následující analýzy a diskuze o technických možnostech jejich provedení. Zadáním byl také návrh uživatelského rozhraní jednotlivých aplikací, který představuje způsob práce uživatelů s takovou aplikací. Jedním z dalších kroků vývoje byl samozřejmě vývoj aplikace samotné. Při vývoji bylo potřeba zvážit několik aspektů, např. typ databázového systému, časové využití aplikace, šifrované připojení do systému SAP, dynamičnost samotného připojení i dynamičnost a modularita samotné aplikace, kvůli možným budoucím rozšířením. Následně po testování vždy došlo k jednotlivému představení finální verze vyřešeného úkolu zadavateli projektu a v posledním kroku nasazení do produkčního prostředí.

Mé fungování v této pozici bylo naprostým příkladem agilního vývoje softwaru. Agilní vývoj softwaru umožňuje rychle vytvářet požadovaný software a pružně reagovat na změny v jeho požadavcích. Takovýto agilní přístup je však často označován v rozporu již s osvědčenými metodikami vývoje softwaru, především kvůli tomu, že dává preferenci v rychlosti a dynamičnosti vytvořeného softwaru nad jinými nezbytnými částmi vývoje. Mezi další klíčové prvky tohoto způsobu patří soustředění na jednotlivce, ale zároveň se nesmí zapomenout také na komunikace mezi těmito jednotlivci. Dále se zde nedbá na rozsáhlé a vyčerpávající dokumentace, které často bývají časově náročné. Preferována je také přímá komunikace se zákazníkem. [1, p.111-123]

3 SAP, Kendo UI a další použité technologie

3.1 Vývoj webových aplikací v ASP.NET

Jak jsem již zmiňoval, způsob vývoje softwaru je ve firmě soustředěn na jedince. Je tedy na každém programátorovi jaký jazyk a principy si při vývoji zvolí. Avšak při nástupu jsem byl seznámen s tím, že většina webových aplikací je vyvíjena v rámci ASP.NET. To především z důvodu předplacených licencí Microsoft služeb a také v případné snadné komunikaci mezi jinými prvky platformy Microsoft. Tím je myšleno API, databázové servery, webové servery, cílová zařízení klientů a jiné.

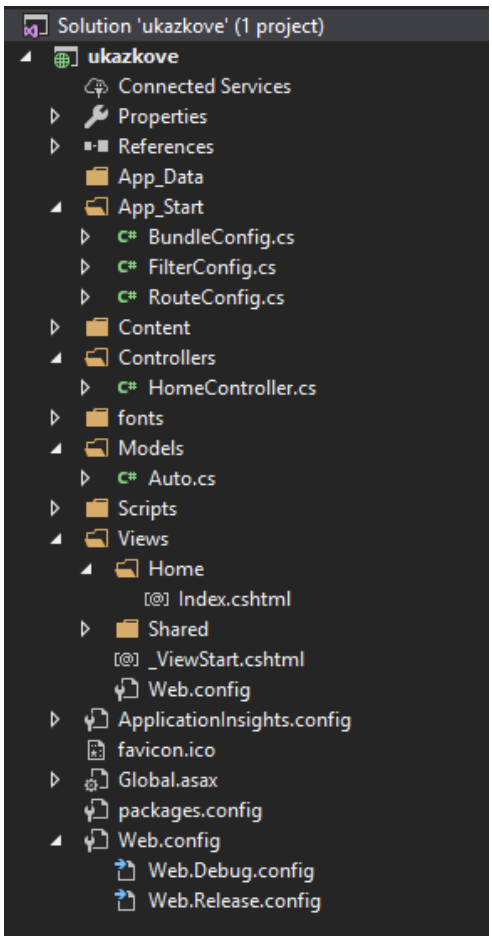
ASP.NET je aplikační rámec vyvinut společností Microsoft k vyvíjení internetových aplikací, webových stránek a také webových služeb. Je následovníkem předchozí technologie nazvané ASP (Active server pages). Tato nová technologie je postavena na CLR (Common Language Runtime), což umožňuje psát ASP.NET aplikace hned v několika jazycích, které jsou podporovány .NET platformou. Aplikace takto vytvořené jsou tzv. "server–side". Tím rozumíme, že kód na serveru zpracuje HTTP požadavky od klienta, vytvoří stránku dle specifických skriptů podle požadovaného dotazu a navrátí stránku klientovi, který ji zobrazí. Opakem je pak "client–side", který představuje JavaScript kód, který spouští aplikace na klientské straně. V realitě jsou pak oba tyto přístupy kombinovány. V mém případě jsem pro práci s rámcem .NET využíval programovací jazyk C#.

3.1.1 MVC – Model-View-Controller

Při vytváření ASP.NET aplikace jsem vždy implementoval návrhový vzor model-view-controller, kterého lze při vytváření nového projektu využít. Tento vzor rozděluje aplikace do následujících logických celků:

- Model – představuje doménovou reprezentaci informací, se kterými systém dále pracuje.
- Controller – je tzv. řadič, který reaguje na požadavky, je spojem mezi prezentační a doménovou vrstvou tzn. mezi View a Model.
- View – představuje prezentační vrstvu, slouží k prezentování informací od řadiče specifickým způsobem.

3.1.2 Adresářová struktura ASP.NET MVC projektu



Obrázek 1: Adresářová struktura ASP.NET MVC projektu

Struktura projektu obsahuje několik souborů a složek (viz Obrázek 1), mezi nimi jsou také složky s názvy námi zmíněného MVC. Podíváme se z jakých hlavních částí se takováto ukázková ASP.NET MVC aplikace skládá a jaký je životní cyklus dotazu v této aplikaci. Celá tato ukázka bude prezentována na verzi .NET Framework 4.6.1.

3.1.3 Dotaz od klienta a směrování

Dotaz je zpráva od klienta cílená na server. Mějme již běžící ASP.NET aplikaci nasazenou na nějaké doméně. Stačí, když skrze internetový prohlížeč pošleme HTTP GET dotaz na danou doménu[1].

Aplikace na serveru obdrží náš dotaz a musí se rozhodnout, jak s daným dotazem naložit a na co se vlastně klient ptá. K rozpoznání, do které funkce kterého řadiče má vlastně dotaz

[1]HTTP GET dotaz je vytvořen internetovým prohlížečem po dotazu na naši doménu např. www.example.com

přijít pomáhá při rozhodování tzv. směrování. Směrování je proces, který umožňuje zpracovávat dotazy, včetně zaslaných parametrů a předávat je příslušným kontrolérům a funkcím, jenž obsahuje. Směrování lze nastavit v souboru RouteConfig.cs.

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

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

Výpis 1: RouteConfig.cs

Výpis 1 obsahuje specifikaci směrování. Můžeme vidět, že lze některé dotazy za pomocí správného nastavení již v tomto prvotním procesu ignorovat a dále je neposílat dál.

3.1.4 Kontrolér

Dalším krokem, ve kterém nás směrování dovedlo v naší aplikaci je správný kontrolér a funkce, která je reakcí na samotný dotaz.

Námi vlastní vytvořený řadič *HomeController* dědí z nadřazené třídy *Controller*, která implementuje mnoho metod a rozhraní. Tyto metody zajišťují společné chování pro všechny kontroléry a umožňují automatický běh nezbytných částí aplikace, které by bylo jinak pracné samostatně implementovat. Ve většině případů jednodušších aplikací není třeba se tímto chováním zaobírat. Využití má například u implementování lokalizace, autorizace kontrolérů aj.

```
[HttpGet]
public ActionResult Index()
{
    return View(new Auto() { id = 1,typ = "Škoda",Barva = "červená" });
}
```

Výpis 2: Metoda kontroléru

HomeController obsahuje tolik funkcí, kolik je na něj možno požadovat dotazů. Různé kontroléry samozřejmě mohou obsahovat různé funkce. Některé mohou obdržet data, která mohou být doménovou logikou uložena do databáze. Jedna ukázková funkce je Index (viz Výpis 2). Tato funkce je po dotazu GET na doménu např. www.example.com v našem kontroléru *HomeCon-*

troller zavolána z důvodu výchozího nastavení v našem směrování (viz Výpis 2). Tato funkce bude sloužit uživateli k prezentování vytvořeného objektu z doménové vrstvy *Auto*.

Můžeme hned pozorovat, že naše funkce je dekorovaná tzv. atributem "HttpGet". Tento tag pomáhá kompilátoru při rozhodovaní, kterou funkci má zavolat. Můžeme totiž například definovat dvě stejnojmenné funkce se stejnou definicí, ale podle typu volaného dotazu se kompilátor rozhodne, kterou zavolá. Toto rozhodování může být určeno jak tímto tagem, tak dále i názvem funkce. Např. GetData nám říká, že se bude jednat o HTTP dotaz typu GET.

Funkce *Index* neobdrží žádný parametr a vrací objekt typu *ViewResult*, který dědí z *ViewResultBase* a ten poté z *ActionResult*, jenž je definovaný jako návratový typ naší funkce *Index*. Platforma .NET poté vybere ze složky Views/Home název souboru podle názvu funkce, z které byla funkce *View* volána a začne generovat stránku, která je zaslána klientskému prohlížeči. V našem případě Index.cshtml. Tato vybraná stránka bude zkombinovaná se souborem _ViewStart.cshtml, který obsahuje sdílené rozvržení všech stránek.

3.1.5 Prezentační vrstva - View

Platforma .NET vytvoří podle předaných parametrů (většinou ve formě objektu modelu) pomocí *Razor Engine* výslednou stránku. Tento engine, který napomáhá snadno a dynamicky vytvářet HTML stránky. Kombinuje v našem případě C# kód s HTML syntaxí. Toto například umožňuje generovat bloky stránky podle podmínek v předaných modelech nebo je generovat v cyklu a tím výrazně minimalizovat napsaný kód (viz Výpis 3).

```
@model ukazkove.Models.Auto

<div>
    <br />
    @if (Model != null)
    {
        <span>@Model.id</span><br />
        <span>@Model.typ</span><br />
        <span>@Model.Barva</span>
    }
</div>
```

Výpis 3: Zobrazení modelu ve View

3.1.6 Soubor Web.config

Dalším souborem projektu, se kterým budeme často pracovat je soubor *Web.config*. Je definován ve značkovacím jazyce XML. Tento soubor slouží ke konfiguraci našeho projektu. Jsou v něm implicitně zapsány informace pro kompilátor i běh aplikace. V dalším pro nás velmi využívaném případě se zde pracuje s uložením připojovacího řetězce pro informace o připojení do databáze

nebo také uložení informací pro připojení do systému viz kapitola Podnikový informační systém SAP R/3.

3.2 Podnikový informační systém SAP R/3

SAP je Německá softwarová společnost, která se zaměřuje na vývoj podnikových aplikací, zkratkou EAS (Enterprise Application Software). Podle společnosti Forbes byla společnost SAP označena v roce 2016 třetí největší softwarovou společností na světě [5].

Podnikový informační systém neboli také plánovaní podnikových zdrojů zkratkou ERP (Enterprise Resource Planning) je označení pro proces, kdy se nějaká organizace za pomocí integrovaných aplikací snaží sbírat, ukládat, řídit a zobrazovat data z několika obchodních procesů. Mezi tyto procesy paří například plánování, nákup, prodej, finance, lidské zdroje, marketing a jiné. Každé oddělení, které spravuje takovýto proces, potřebuje svou vlastní aplikaci, která bude v tomto ohledu využívána. Jednotlivé moduly mají mezi sebou schopnost komunikovat.

Jedním za takovýchto systémů je systém s názvem SAP R/3. Někdy dále uváděn pouze jako SAP. Systém se skládá z několika ERP modulů (viz Tabulka 1). Některé z těchto modulu také určuje rozdělení jednotlivých oddělení ve společnosti Siemens.

Tabulka 1: Názvy modulů SAP R/3

Zkratka	Celé názvy modulů
FI	Finanční účetnictví
Co	Kontroling
AM	Evidence majetku
PS	Plánování dlouhodobých projektů
WF	Řízení oběhu dokumentů
IS	Specifická řešení různých odvětví
HR	Řízení lidských zdrojů
PM	Údržba
MM	Skladové hospodářství a logistika
QM	Management kvality
PP	Plánování výroby
SD	Podpora prodeje

3.2.1 Technologie systému

Systém přináší několik funkcionalit. V názvu SAP R/3 písmeno R představuje 'real-time data proccessing', což ve volném překladu znamená zpracovávání dat v reálném čase. A číslice 3 představuje třívrstvou architekturu *model-view-controller*, která již byla v této práci zmíněna.

Moduly, které jsou programátory jednotlivých oddělení vytvářeny jsou psány ve vlastním proprietární jazykem ABAP (Advanced Business Application Programming). R/3 poskytuje programátorům kompletní vývojové prostředí, které jim umožňuje do modulů přidávat vlastní funkcionalitu, která je pro daný běh procesu ve společnosti potřeba. ABAP komunikuje s databází pomocí SQL dotazů.

V celém tomto systému se dbá na velkou bezpečnost. Komunikace i data jsou šifrovány. Jednotlivá připojení mohou být zprostředkována odděleně od vnější sítě. Tímto je docíleno, že firemní data mohou být jen stěží odcizena.

3.2.2 SAP S/4 HANA

Tento systém je nástupce původní verze S/3. Poskytuje podobné plánování podnikových zdrojů, avšak oproti předchozí verzi přichází s rozšířenými možnostmi využití nových technologií, cloudových služeb a strojového učení. Mezi další funkcionality tohoto nového systému patří prediktivní analýza, analýza textu, aplikační server a mnoho dalšího.

Celý tento nový systém je postaven na proprietárním databázovém systému s názvem SAP HANA. SAP HANA je paměťově orientovaný databázový systém, což znamená, že velká většina dat je držena v hlavní paměti systému. To má za důsledek nárůst na rychlosti přístupu k datům. Od technologie non-volatilních neboli stálých pamětí odpadá problém se ztrátou dat odpojením od sítě. Držet veškerá data v hlavní paměti by bylo velice nákladné, a tak se zde užívá technika zvaná *in-memory storing*. Oproti technice cachování nejsou v paměti udržovaná naposledy čtená data, ale jsou z disku v paměti udržovaná data, která jsou čtená obecně nejfrekventovaněji.

Dále je SAP HANA sloupcově orientovaná relační databáze. Tím rozumíme, že databázový systém ukládá data po sloupcích ale ne po řádcích.(viz Tabulka 2 a 3). Tento způsob ukládání však není pro naši práci s databázovým systémem nijak oproti řádkovému rozdílný. Lze tedy užívat SQL pro čtení a zpracování dat. Tento sloupcový přístup je upřednostňován z důvodu, že ve většině případech se nedotazujeme na celý řádek z tabulky, ale pouze na jednotlivé sloupce této tabulky. Tím pádem nemusíme číst zbytečně velké množství dat, ale pouze ta data, která potřebujeme. Z tohoto důvodu je doba vykonávání dotazu v těchto případech dotazů zlepšena.

Tabulka 2: Řádkově orientovaný systém

RowId	EmpId	Příjmení	Křestní	Plat
001	10	Smith	Joe	40 000
002	12	Jones	Mary	50 000
003	11	Johnson	Cathy	44 000
004	13	Jones	Bob	55 000

Tabulka 3: Sloupcově orientovaný systém

EmpId	Příjmení	Křestní	Plat
10:001	Smith:001	Joe:001	40000:001
12:002	Jones:002	Mary:002	50000:002
11:003	Johnson:003	Cathy:003	44000:003
22:004	Jones:004;	Bob:004	55000:004

3.2.3 Využití SAPu z mé strany

Má pozice ve firmě se sice netýkala programování SAP modulů v jazyce ABAP, avšak znalost tohoto systému pro mne byla klíčová z důvodu, že jsem musel pravidelně komunikovat s programátory jednotlivých modulů. I přesto, že SAP je propracovaným systémem se spoustou funkcionalit, vytvořit v jazyce ABAP příjemné uživatelské rozhraní není vždy snadné. V tomto případě přicházejí na řadu ASP.NET vývojáři. Vytvářet webové rozhraní, které by zobrazovalo data ze SAP systému nebo do něj data posílalo bylo jedním z mých dalších úkolů.

3.2.4 Programové rozhraní pro komunikaci se systémem SAP

Standardním rozhraním pro vnější komunikací se systémem SAP je Remote Function Call, volným překladem volání vzdálené funkce a dále jen zkratkou RFC. RFC volá takzvaný funkční modul, který je poté spouštěn na vzdáleném SAP serveru [3]. Takto zavolaný funkční modul, kterému lze předat i parametry, spouští interní funkcionalitu na SAP serveru i s možností navrácených dat. Volání na tento server může být synchronní, asynchronní nebo do fronty.

Společnost SAP poskytuje knihovny pro hned několik jazyků, které se poté implementují do cílových projektů ve kterých je RFC používáno. Tato knihovna vyžaduje množství konfigurací a práce s ní při obyčejném volání není jednoduchá. Proto v našem případě je k tomuto dopsána vlastní knihovna - nová vrstva, která tuto práci zjednodušuje a unifikuje.

Tato konfigurace obsahuje hned několik údajů mezi nimi jsou například jméno cílového serveru, jazyk, mód připojení, informace o konfiguraci SNC[2], dále jméno, heslo pro připojení a mnoho dalšího.

3.3 Kendo UI

Tvořit snadno a rychle dynamické webové aplikace za pomocí HTML, CSS a JavaScriptu by bylo časově náročné. V tomto případě přichází na řadu rámce. Rámec je jakousi software strukturou, která slouží jako podpora při programování a vývoji software. Obsahuje podpůrné programy,

[2]Secure Network Communications – volným překladem bezpečné síťové komunikace, nám zajišťuje bezpečné připojení na vzdálený SAP server

knihovny i návrhové vzory. Využití takovéhoto rámce usnadňuje práci tím, že představuje již jistá řešení typických problémů, kterými se již dále nemusíme zaobírat.

Jedním z takovýchto příkladů je *frontendový rámec* zvaný Kendo UI od společnosti Telerik. Jak již už název napovídá tento rámec představuje prostředky pro stavbu uživatelského prostředí (UI – zkratka ze slova User Interface). Obsahuje několik variant nejrůznějších způsobů stylování. Tyto návrhy se dají aplikovat na velké spektrum UI komponent jako jsou, interaktivní tabulky s funkcemi jako je např. třídění záznamů a další. Dále nejrůznější grafy, seznamy, barevné palety, kalendáře a mnoho dalšího. Kendo UI je možné zakomponovat hned do několika webových rámců jako je Angular, React, Vue a v našem případě jím bylo jQuery. Při práci s ním je používán jazyk JavaScript.

3.3.1 DataSource

Jak jsem již zmiňoval, většina z těchto UI prvků rámce je vysoce interaktivní. Jako příklad uvažujeme HTML tabulku. Abychom s ní mohli interagovat musíme ji nejprve naplnit daty. Kendo UI generuje tabulku klasicky za pomocí HTML *td* a *tr* elementů z dat, které obdrží z tvz. *DataSource*. *DataSource* je prvek ukrytý na pozadí aplikace a stará se o automatizaci procesů, které zahrnují požadování dat a jejich odesílání za pomocí HTTP dotazů. Tento prvek je tedy připojen do konfigurace UI prvku, pro náš příklad tabulky. A nad touto tabulkou obstarává veškeré CRUD[3] operace. Tento prvek obsahuje velké množství nastavení, jako příklad filtrování daných dat, seskupování do skupin, určitý styl stránkování a mnoho jiného. Všechny tyto operace mohou probíhat synchronně nebo asynchronně. *DataSource* může být sdílený mezi několika prvky, tímto můžeme ušetřit režii a již používat stažená data. [4]

[3]Create Read Update Delete (Vytvářet, číst, upravovat a mazat) jsou, čtyři základní operace, které se mohou dít nad nějakou množinou dat.

3.3.2 Příklad použití Kendo UI

Mějme bežící webovou aplikaci a k ní připojené veškeré potřebné Kendo UI knihovny a také jQuery knihovnu (viz Výpis 4).

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="styles/kendo.common.min.css" />
    <link rel="stylesheet" href="styles/kendo.default.min.css" />
    <link rel="stylesheet" href="styles/kendo.default.mobile.min.css" />

    <script src="js/jquery.min.js"></script>
    <script src="js/kendo.all.min.js"></script>
</head>
</html>
.
.
```

Výpis 4: Příklad Kendo UI 1.

Uvnitř elementu *body* mějme tabulku s již předdefinovanou hlavičkou (viz Výpis 5).

```
.
.
<table id="movies" class="metrotable">
    <thead>
        <tr>
            <th>Rank</th>
            <th>Rating</th>
            <th>Title</th>
            <th>Year</th>
        </tr>
    </thead>
    <tbody>
    </tbody>
</table>
.
.
```

Výpis 5: Příklad Kendo UI 2.

Uvnitř elementu *script*, který označuje blok JavaScript kódu můžeme začít definovat vzor obsahu tabulky za pomocí tvz. *template.* (viz Výpis 6). *Template* neboli vzor nám umožňuje snadno vytvářet kousky HTML dokumentu, které mohou být snadno míseny s JavaScript daty. *Razor engine* poté zpracuje tento kousek vzoru a vykreslí jej. Kousky JavaScript kódu jsou umisťovány do těchto značek '#=' a ukončující '#' Tento proces je velice podobný již zmíněnému View u aplikací ASP.NET.

```
.
.
<script id="template" type="text/x-kendo-template">
    <tr>
        <td>#= rank #</td>
        <td>#= rating #</td>
        <td>#= title #</td>
        <td>#= year #</td>
    </tr>
</script>
.
.
```

Výpis 6: Příklad Kendo UI 3.

Začneme s definicí *DataSource* (viz Výpis 7). Definice a volání funkce pro čtení může začít až poté, kdy bude DOM[4] tabulky celý připraven k použití. Pokud by se tak nestalo, objekt pro tabulku by ještě nebyl vláknem uvolněn a Kendo UI by se s ním již snažilo pracovat, to by vyústilo v chybu.

```
.
.
<script>
    $(document).ready(function() {
        // template z definice
        var template = kendo.template($("#template").html());

        var dataSource = new kendo.data.DataSource({
            transport: {
                    read: {
                        url: "https://demos.telerik.com/kendo-ui/service/Example"
                        dataType: "json",
                        type: "GET"
                    }
                },
            change: function() { // registruj se na event CHANGE
                $("#movies tbody").html(kendo.render(template, this.view())); // napln tabulku
                     daty
            }
        });

        // volani funkce cteni
        dataSource.read();
    });
</script>
.
.
```

[4]Document object model, neboli objektový model dokumentu

Výpis 7: Příklad Kendo UI 4.

Na předchozí ukázce (Výpis 7) můžeme pozorovat, že došlo k vytvoření a naplnění proměnné *tepmlate*, která byla později předána jako parametr funkci *render* společně s požadovanými daty, které byly obdrženy z funkce *view* objektu *DataSource*. Data jsou po zavolání metody *read* požadována z výše uvedené URL adresy s HTTP požadavkem typu GET a navrácena jako datový typ JSON[5]. Po všech těchto krocích může výsledek s přidáním CSS vypadat například takto.

rank	rating	title	year
1	9.2	the shawshank redemption	1994
2	9.2	the godfather	1972
3	9	the godfather: part ii	1974
4	8.9	il buono, il brutto, il cattivo.	1966
5	8.9	pulp fiction	1994
6	8.9	12 angry men	1957
7	8.9	schindler's list	1993
8	8.8	one flew over the cuckoo's nest	1975
9	8.8	inception	2010
10	8.8	the dark knight	2008

Obrázek 2: Výsledek aplikace s použitím DataSource u Kendo UI.

[5]Java Script Object Notation

4 Zadání úkolů

4.1 KPI útvaru PROD 6

Jedním ze zadaných úkolů, na kterém mohu názorně naznačit výše zmíněné postupy a technologie je zobrazovaní ukazatele KPI. Tento ukazatel je pro útvar údržby, zkráceně označován jako PROD 6.

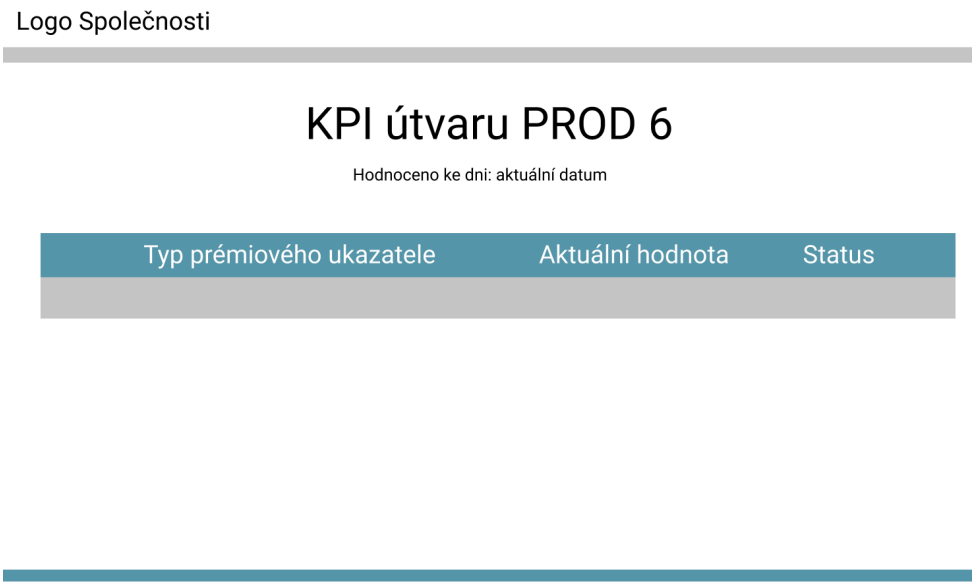
Zkratka KPI (z anglického *Key performance indicator*) přeloženo, jako klíčový ukazatel výkonnosti. Tento indikátor se běžně používá k měření úspěšnosti aktivity dané organizace, v našem případě útvaru. Úkolem bylo získat tuto hodnotu indikátoru pro dva ukazatele. To jsou:

1. Dostupnost strojů a zařízení
2. Vícenáklady před dodáním zaviněné poruchou stroje

K těmto indikátorům je nastavena jejich určitá cílová hranice, můžeme nazývat anglicky *threshold*. Pokud je daný indikátor splněn, je dle jeho příslušné splněné hodnoty přidružena danému oddělení prémie. Pohybujeme-li se v určité mezi je prémie nižší, pokud jsme pod hodnotou, prémie se ještě sníží.

Jakmile byla tato hodnota z dat vypočtena, daný *threshold* nastaven a k němu přiřazeno prémiové ohodnocení, bylo potřeba tyto motivační údaje útvaru přehledně zobrazit. Tento úkol byl rozdělen mezi dva vývojáře. Mě, který jsem měl na starost získání a zobrazení dat. A jako další, kolegyni, která za stranu modulu SAP měla za úkol stanovit z dat příslušného útvaru hodnoty indikátorů a dle zadání vedoucího nastavit příslušný *treshold*. Jak jsem již zmiňoval výše spolupráce ASP.NET vývojářů a SAP programátorů je klíčem řešení většiny zadaných projektů.

Při realizaci ASP.NET aplikace jsem musel zvážit několik aspektů. Na údržbě byla vyhrazena informační tabule s uhlopříčkou 32” a HD rozlišením. Bylo nutné myslet na to, aby písmo bylo snadno čitelné i z dálky. Pro snazší rozpoznání změny bylo nutné využít barevného podkreslení. Dále, tyto zobrazované informace samozřejmě musí být aktuální, tudíž je potřeba požadovat nové informace od serveru v určitých intervalech. V úvahu muselo být také bráno, že počet ukazatelů a jejich hodnoty prémií se mohou v poměrně krátkých časových úsecích měnit, tudíž bylo potřeba myslet na již zmíněnou dynamičnost kódu, aby při nadcházejících změnách bylo potřeba co nejméně úprav. Jelikož se jedná o poměrně jednoduchou aplikaci, dají se na jejím řešení ukázat výše zmíněné postupy a problematiky. Jak využití SAP, tak i tvorba aplikací v ASP.NET. Zadání podobného charakteru bylo v průběhu mé praxe několik, komplexnost a speciální požadavky se však lišily. Odhadovaná časová náročnost tohoto úkolu už s případnou znalostí všech potřebných knihoven je dva dny.



Obrázek 3: Skica zobrazení údajů na informační tabuli.

4.2 Interní aplikace AM/IT

Při mém nástupu mně byla na starost předána interní aplikace s názvem AM/IT. Zkratka představuje zkratky názvů oddělení pro které je aplikace určena. Tato aplikace slouží ke správě informací o probíhajících projektech na oddělení. Dalším hlavním účelem je samotné zaznamenávání informací o odvedené práci na projektech. Tyto informace jsou dále využívány dalšími členy oddělení, kteří například díky těmto informacím vytvářejí faktury k jednotlivým projektům. Všechny získané informace mohou být také využity k různým analýzám, přičemž několik z nich bylo mým úkolem. Aplikace již byla v produkčním stavu, avšak po dobu mé praxe vyvstávaly požadavky na její úpravu. Úpravy se týkaly změn uživatelského rozhraní, změn způsobů zadávání nebo typu vstupních dat, dále vytváření přehledů nad daty nebo implementování záznamů změn určitých hodnot v systému.

Klíčem k práci s tímto systémem bylo se s ním nejprve seznámit a pochopit strukturu informací, kterou obsahuje. Pochopit také všechny use case[6], které jsou se systémem možné. Jak jsem se již zmiňoval o agilním programování a malé preferenci k dokumentaci, tento projekt nebyl zdokumentován tudíž k účelu pochopení aplikace bylo potřeba pár sezení, kde bylo možné pochopit a analyzovat všechny interní pojmy, zkratky a procesy.

4.2.1 Vykázáno versus Objednávka

Osoby s rolí vyššího přístupu než má vývojář mají přístup do sekce zvané Přehled. Jak název napovídá sekce obsahuje přehledy nad daty ze systému. Můžeme zde nalézt přehledy jak tabul-

[6]Use case – česky případ užití je listem akcí nebo událostí definující cílový způsob práce s nějakým systémem

kové, tak i grafové. Informace v přehlednější formě o uživatelích, zákaznících, tématech a mnoha dalších.

Mým úkolem bylo rozšířit tento přehled o další analýzu s názvem *Vykázáno vs Objednávka*. Klíčem této analýzy bylo přehledně zobrazit informace o zákaznících, kolik tito zákazníci mají objednaných kapacit[7] a kolik reálně na tyto zákazníky bylo odvedeno hodin a to vše uváženo v daném časovém období. Protože samotný graf by byl sice zajímavý, ale málo říkající požadavek proto byl rozšířen. Po kliku na sloupec se zákazníkem se zobrazí tabulka všech vývojářů pracujících pro tohoto zákazníka s odvedenými hodinami a objednanou kapacitou. Ideální stav by samozřejmě měl být v daném období naprosté vyvážení mezi hodnotami práce a objednáním. Realita je však jiná a pro rychlé a přehledné zobrazení bylo požadováno zvýraznění nesplněných hodnot v tabulce.

4.2.2 Pracovní vytíženost vývojářů

Další úkol byl z části podobný předcházejícímu Vykázáno versus Objednávka. Jedná se znovu o analýzu nad systémem AM/IT. Vedoucí týmů chtěli získat větší přehlednost nad svými vývojáři a dozvědět se, kolik na sobě mají navázané práce. Zobrazení má být přehledně ukázáno v grafu a pro větší detail se po kliku na vývojáře zobrazí tabulka s přehledem jeho dosud nesplněných úkolů. Úkoly budou obsahovat přímý odkaz na témata uložena v systému AM/IT.

Cílem je na vodorovné ose zobrazit všechny vývojáře a k nim ve sloupcovém grafu ukázat kolik hodin jim zbývá v sumě na všech tématech vykázat. Vstupní parametr datum je zde jako počáteční systémové a cílové volitelné. Již zmiňovanou "zarážkou" bylo požadováno zobrazit fond hodin. To je počet hodin, které je za dané období možné maximálně vykázat. Tato analýza nám umožní pohled na jednotlivé vývojáře a jejich rozložení práce, zdali jsou někteří přetěžovaní a nebo naopak je jejich pracovní vytížení nižší.

Nutno podotknout, že se v systému vyskytují nekorektní data a výjimky. Lidé mají už dlouhodobě zapsány projekty, které jsou zastaveny nebo nejsou zcela uzavřeny. To vše je příčinou velmi vysokých hodnot u některých vývojářů.

4.3 Success story

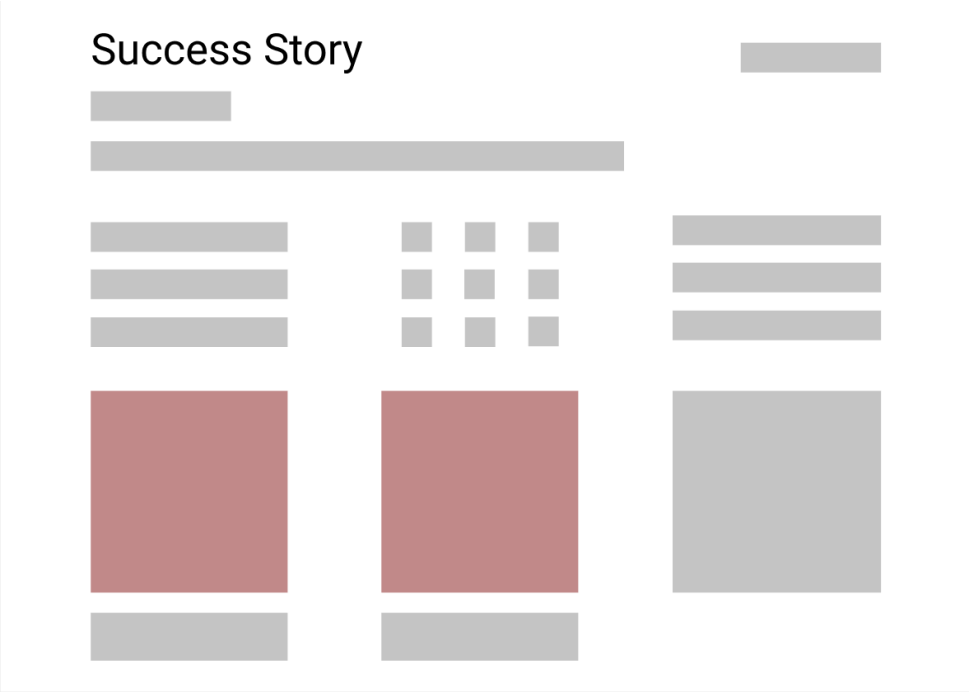
Mezi další aplikace, které mi byly předány na starost patří Success Story. U ní se vyskytlo hned několik požadavků, které níže popíšu. Nejprve však pár slov o samotné aplikaci.

Aplikace je databází pro standardizované vizualizace, tzv. Success Story. Success Story představuje dokument o úspěšně vypracovaném projektu. Cílem této aplikace je přehledné zobrazení všech takovýchto vypracovaných projektů, možnost filtrování, přikládání příloh a obrázků, export do PDF dokumentu. Aplikace je vytvořena jako více jazyčná, také i samotné dokumenty umožňuje ukládat v několika jazycích.

[7] Kapacitou je míněno schopnost vývojáře odvádět práci, uváděno v hodinách.

4.3.1 Umožnit uživatelům vkládat a zobrazovat formátovaný obsah

Zadání úprav na této aplikaci bylo obsáhlejší. Mělo dojít ke změnám vzhledu formuláře, rozšíření počtu vstupních hodnot, vytvoření obrázkové galerie u dokumentu a několika dalším. Nejzajímavějším úkolem mi přišlo rozšíření o textový editor, který budu dále popisovat. Formulář zadávání nového dokumentu obsahuje dva velké sloupce, které slouží k zadávání delšího textu popisu a hodnocení vypracování projektu. Na Obrázku 4 můžeme vidět skicu formuláře a zvýrazněné sloupce, o kterých je řeč. Skica představuje i návrh výstupního formátu dokumentu. Je tedy pochopitelné, že textový obsah by z estetického hlediska měl být formovatelný.



Obrázek 4: Skica dokumentu Success Story

Původní verze byla taková, že jako vstupní prvek sloužil HTML element s názvem *TextArea*, jehož obsah se poté ukládal do databáze. *TextArea* však umožňuje vepisovat pouze čistý text bez jakéhokoliv formátování. Úkolem bylo navrhnout řešení pro možnost formátování textu u těchto oblastí.

5 Řešení úkolů

5.1 KPI útvaru PROD 6

Aplikace tedy bude představovat pouze jednu stránku. Zobrazující informace o jednotlivých ukazatelích KPI. Tento ukazatel obsahuje informace o druhu daného ukazatele, aktuální hodnotě ukazatele a výsledných prémiích, které se z ukazatele odvíjejí. Dále je předána do prezentační vrstvy informace o kterou úroveň indikace prémií se jedná. Jak jsem již bylo zmíněno, hodnota *threshold* se může v týdnech měsících měnit a držet tuto informaci v proměnné by bylo neefektivní. Proto, z důvodu snadné údržby jsem se rozhodl umístit tyto hodnoty do již zmiňovaného souboru *Web.config* a případně pouze na serveru změnit hodnoty v tomto souboru. Model KPI můžeme vidět na Výpisu 8.

```
public class KPIdata
    {
        public int Ukazatel;
        public string Hodnota;
        public float Premie;
        public string StatusPremie {
            get {
                float up = float.Parse(ConfigurationManager.AppSettings["upThreshold"]);
                float down = float.Parse(ConfigurationManager.AppSettings["downThreshold"]);

                if (this.Premie >= up)
                {
                    return "ok";
                }
                else if(this.Premie <= down)
                {
                    return "bad";
                }
                else
                {
                    return "notok";
                }
            }
        }
    }
```

Výpis 8: Model KPI

5.1.1 Získání dat

Teď, když už máme definovánu třídu modelu, který budeme chtít předávat a zobrazovat v prezentační vrstvě, je načase data získat. Pro předání prezentační vrstvě jsem se rozhodl více ukazatelů zaobalit do kolekce datového typu *List*. Z důvodu, že počty ukazatelů se mohou měnit. Z kontroléru po požadavku *GET* na hlavní *Index* stránku naší aplikace, budu vždy volat na naši pomocnou třídu *KPI* a její metodu *GetKPIData()*. Tímto docílíme, že se vždy po každém obnovení stránky zeptáme serveru na nová data. A tato metoda nám vrátí již zmíněný *List* našich *KPIdata* objektů.

```
private static string plant = "FRE1";

public static List<KPIdata> GetKPIData()
{
    var result = CallSap(plant);
    DataTable dtb;
    List<KPIdata> data = null;
    if (result.Success)
    {
        dtb = result.Response.GetDataTable("VYSTUP");
        data = new List<KPIdata>();
        foreach (DataRow item in dtb.Rows)
        {
            data.Add(new KPIdata()
            {
                Ukazatel = int.Parse(item["UKAZATEL"].ToString()),
                Hodnota = item["HODNOTA"].ToString().Replace('.', ','),
                Premie = float.Parse(item["PREMIE"].ToString()),
            });
        }
    }

    return data;
}
```

Výpis 9: Získání dat

Na Výpisu 9 vidíme vidíme metodu *CallSap*, které předáváme parametr s označením závodu plní objekt *result*. Objekt je generického typu a obsahuje proměnnou *Success* typu *bool*, která indikuje úspěch volání, *Message* typu *string*, která obsahuje zprávu volajícímu a objekt *Response* typu generického, který obsahuje výsledek volání. Tento unifikovaný návratový typ volání do SAPu nebo do databáze je součástí již zmiňované interní knihovny a snažíme se jej společně integrovat do všech projektů. Objekt *result*, tedy obsahuje výsledná data volání a ty jsou typu *DataTable*. Projdeme následně všechny řádky požadované tabulky a naplníme list obdrženými KPI ukazateli.

5.1.2 Metoda CallSap

Této metodě se předává parametr s označením závodu. Ve Výpisu 10 je vytvořena instance objektu ze třídy *Connection* z již zmíněné interní knihovny pro připojení. Objektu je nastaven cílový systém. A vytvořen objekt typu *Hashtable* a v něm nastaveny parametry, které budou předný u volání funkčního modulu. Následující kód byl v pozdější době zabalen do separované třídy, která představuje nadřazenou vrstvu nad interní Siemens knihovnou, která představuje způsob pro připojení do SAPu. Důvod je jednoduchý – unifikace.

```
private static Siemens.Result<Sap.Connector.Result> CallSap(string _plant)
{
    Connection SapConnection = new Connection();
    SapConnection.SetSystem(_sapSystem);

    Hashtable FunctionModuleParameters = new Hashtable();
    FunctionModuleParameters.Add("WERKS", _plant);

    var _result = new Siemens.Result<Sap.Connector.Result>();
    try
    {
        _result.Response = SapConnection.GetData(functionModuleName,
            FunctionModuleParameters);
        if (_result.Response.Succeed)
        {
            _result.Success = _result.Response.Succeed;
            _result.Message = _result.Response.Error;
        }
        else
        {
        Siemens.Log.Append("Models.KPI.CallSap Result – Chyba čtení SAP");
        }
    }
    catch (Exception e)
    {
        _result.Success = _result.Response.Succeed;
        _result.Message = _result.Response.Error;
        Siemens.Log.Append("Models.KPI.CallSap – Chyba čtení SAP",e.ToString());
    }

    return _result;
}
```

Výpis 10: CallSap Metoda

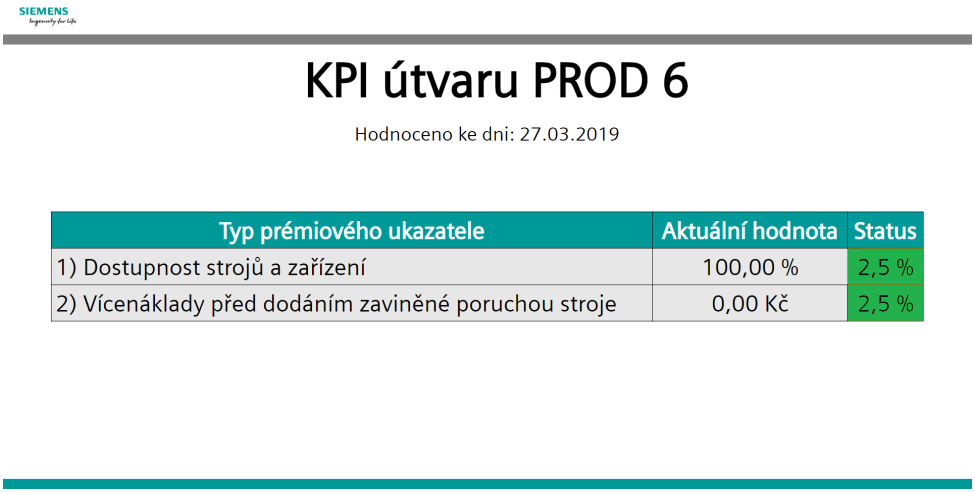
5.1.3 Další kroky

Jakmile byl nachystán model i s metodu pro získání dat, bylo nutné pouze v kontroléru danou metodu zavolat a model předat do *View*. Ve *View* si připravíme vzhled naší stránky a za pomocí již zmiňovaného *Razor Engine* zobrazíme data obsažená v modelu. Za pomocí přidání tzv. meta informací do hlavičky zajistíme automatické obnovování stránky (viz Výpis 11). Při tomto obnovení se vždy zažádá o nová data v kontroléru. Tím jsme docílili toho, že naše informace na zobrazovací tabuli budou vždy aktuální.

```
<head>
    <meta http-equiv="refresh" content="3600">
</head>
```

Výpis 11: Meta informace v hlavičce

Finální vzhled naší zobrazovací tabule můžeme vidět na Obrázku 5.



Obrázek 5: Výsledný vzhled KPI

5.2 Interní aplikace AM/IT

Aplikace psána způsobem již zmíněným několikrát v této práci a to je skrze ASP.NET rozhraní a také za využití zmíněného rozhraní Kendo UI. Na pozadí se skrývá databáze od Microsoft SQL Serveru obsahující 26 tabulek. Porozumění této databázi bylo klíčové při tvorbě požadovaných dotazů. Nutno však říci, že aplikace AM/IT již měla svého předchůdce psaného v jazyce Visual Basic a již předchozí návrh databáze, ze kterého se vycházelo i při vytváření nové verze aplikace. Původní databáze nebyla navržena vhodným způsobem. Při seznamování s ní jsem objevil několik nesrovnalostí jako jsou například nesprávné datové typy u atributů jako datum a čas, což při tvoření dotazů výrazně snižovalo přehlednost kódu pokud je třeba dlouze údaje převádět, dále chybějící integritní omezení a jiné nedostatky. Aplikace není naštěstí příliš vytěžována a použití

je nejvyšší zejména ke konci měsíce, kdy je potřeba vykázat hodiny za měsíc. V opačném případě by dotazy s velkým počtem přístupů mohly způsobit jisté problémy s výkonností systému.

5.2.1 Vykázáno versus Objednávka

Zobrazovat data budeme ve sloupcovém grafu. Zadáním bylo zobrazit všechny zákazníky, tyto zákazníky budeme chtít vyobrazovat na horizontální ose. Na svislé ose uvedeme úroveň odpracovaných hodin. Jako třetí ukazatel využijeme zarážku, která nám ve sloupci ukáže úroveň přiřazené kapacity zákazníkovi. Do dotazu budou vstupovat vstupní parametry a to datumový rozsah v měsících. V Tabulce 4 vidíme několik hlavních tabulek, ze kterých jsme selektovali požadované data.

Tabulka 4: Výpis tabulek pro dotaz k Vykázáno vs Objednávka

Název tabulky	Důležité atributy	Popis
person	id, fullname, active	uživatelé systému
person_role	role_id	informace o správné roli uživatele
n_customer	id, name	informace o zákazníkovi, jeho jméno
topic	id, psp_cz, psp_de	projekty a přiřazený zákazník
topic_schedule	topic_id, id, developer_id	plán vývojáře pracujícího na tématu
n_psp_customer	psp_id	fakturační element vycházející ze SAP
work	schedule_id, hours	informace o vykázané práci
fond	mesic, hodin	informace fondu hodin v měsíci
orders	customer_id, mesic	informace o objednávce na daný měsíc
person_order	person_, order_id	vazba objednávky na osobu

Vazbou mezi těmito tabulkami, přidáním podmínek k jednotlivých atributům a aplikování správných agregačních funkcí jsme docílili výsledku. Ke každému uživateli je za pomocí spojení typu *cross join* spojen zákazník a k němu uvedeny hodiny a jeho objednávka. Příklad výsledku řádku můžeme pozorovat v Tabulce 5.

Tabulka 5: Řádek Výsledku dotazu Vykázáno vs Objednávka

DevId	DevName	CusId	CusName	Kapacita	WorkHours
4	Lukáš Babinec	1	FST	62,25	60

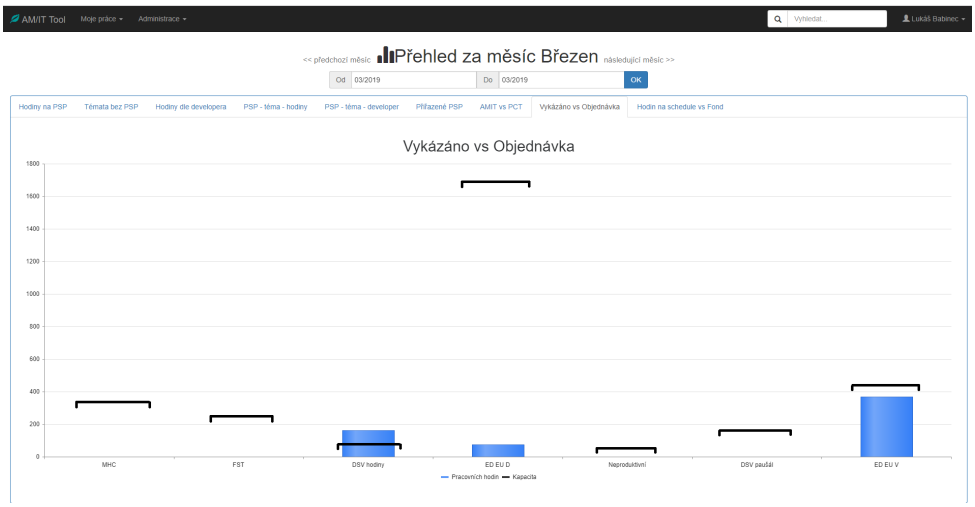
Načtením prvku grafu vzniká žádost v již zmiňovaném *DataSource*, který pošle požadavek na data do kontroléru, který spustí proceduru s dotazem na databázi a výsledná data posílá zpět. Graf s následnou konfigurací svislých a vodorovných os, aplikováním agregace (konkrétně sumy) nad hodinami a seskupením informací podle zákazníků se dostaneme k výsledku. Po kliknutí na sloupec je odchycena událost, po které chceme zobrazit vyskakovací okno s vyfiltrovanými

daty podle zvoleného zákazníka z prvku *DataSource*, který nám drží již stažená data. Po vykreslení DOM modelu tabulky projdeme všechny řádky a porovnáme buňky s údaji o kapacitě a odvedených hodinách, na buňky aplikujeme následující kód.

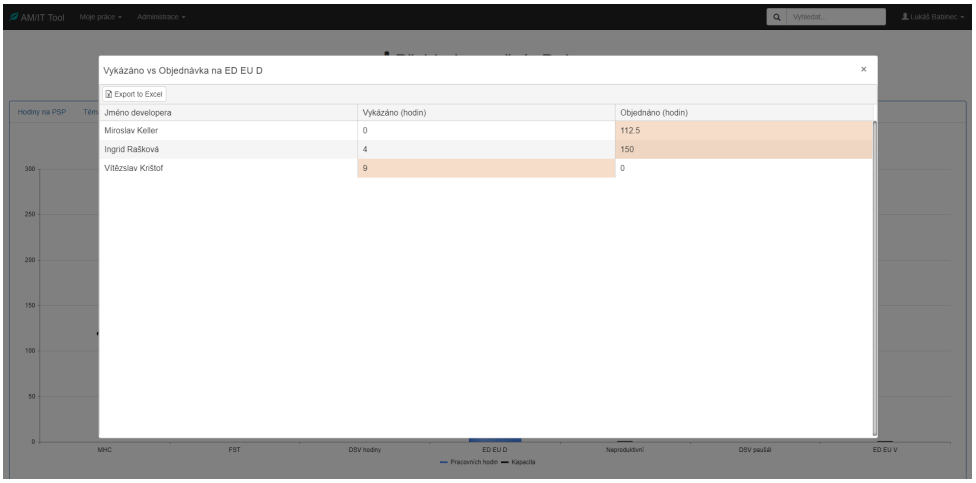
```
.
.
<script>
   if (Kapacita > WorkHours)
    {
        cellKapacita.addClass("errorClass");
        cellKapacita.addClass("errorClass:hover");
    }
    else if (Kapacita < WorkHours)
    {
        cellWorkHours.addClass("errorClass");
        cellWorkHours.addClass("errorClass:hover");
    }
</script>
.
.
```

Výpis 12: Kód customizace tabulky

Výsledek našeho grafu (viz Obrázek 6). Po kliknutí na sloupec zákazníka se zobrazí již zmiňovaná tabulka s detailem pracovníků (viz Obrázek 7).



Obrázek 6: Graf Vykázáno versus Objednávka



Obrázek 7: Tabulka Vykázáno versus Objednávka

5.2.2 Pracovní vytíženost vývojářů

Řešení tohoto problému je velice obdobné předchozímu. V nové záložce je vytvořen za pomocí rámce Kendo UI prvek graf s připojenou komponentou *DataSource*, která má nastavenou adresu s akcí kontroléru, který vrací data z procedury, kterou vytvoříme nad databází.

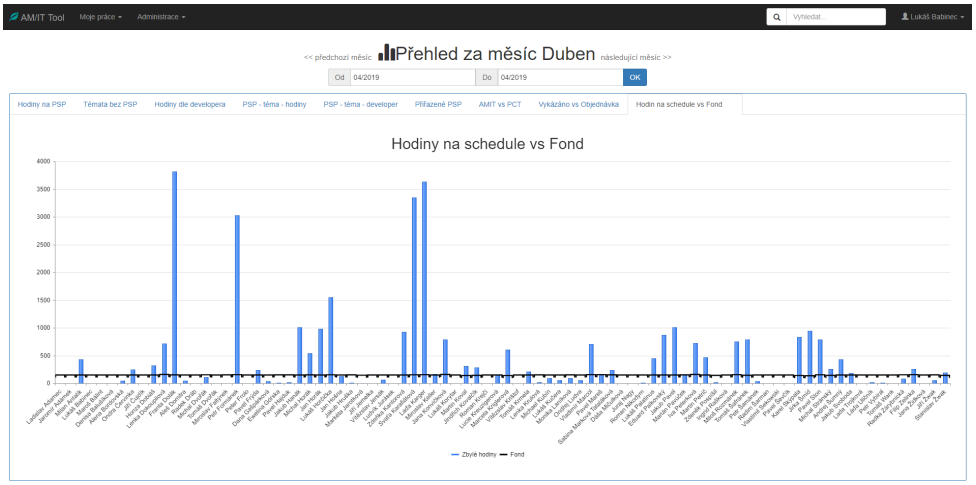
Jak jsem již zmiňoval nevhodně navrženou databázi, mnou vytvořený dotaz má ve finální podobě okolo 130 řádků, nebudeme se mu věnovat tedy do detailů. Hlavní důvod objemu kódu je zpracovaní již zmíněných nevhodně zvolených datových typů a také velký počet podmínek, které řeší různá datumová rozpětí. Pro některé případy zpracování nevhodně zvolených datových typů byla napsána funkce, která tento problém do jisté míry eliminovala.

Cílem bylo získat z dotazu co nejdrobnější data o vytíženosti vývojářů, která poté můžeme filtrovat nebo agregovat na klientské straně. K dotazu přistupujeme tak, že u každého vývojáře vypíšeme k jakému tématu je zapsaný z tabulky *topic_schedule* (viz Tabulka 4). K této informaci napočítáme počet hodin, které mu ještě zbývají na tématu odvést. Tím rozumíme rozdíl hodin na tématu a počet již odpracovaných hodin. Dále uvažujeme ideální případ, že z možných kapacit v časovém období bude na všech tématech pracovat stejně. Tudíž možný fond v zadaném časovém období rozdělíme rovným dílem mezi všechna témata na kterých vývojář pracuje. Další úvahu máme u výsledného součtu hodin, která vývojářům zbývá, zde musíme zobrazovat jen část hodin, která dílem odpovídá časovému období zadaného z maximálního možného, který je uveden na projektu. To z důvodu, že v grafu nechceme ukazovat v krátkém zadaném období velký počet hodin, na jejíž splnění je období delší. Ukázka výsledku řádku z dotazu (viz Tabulka 6).

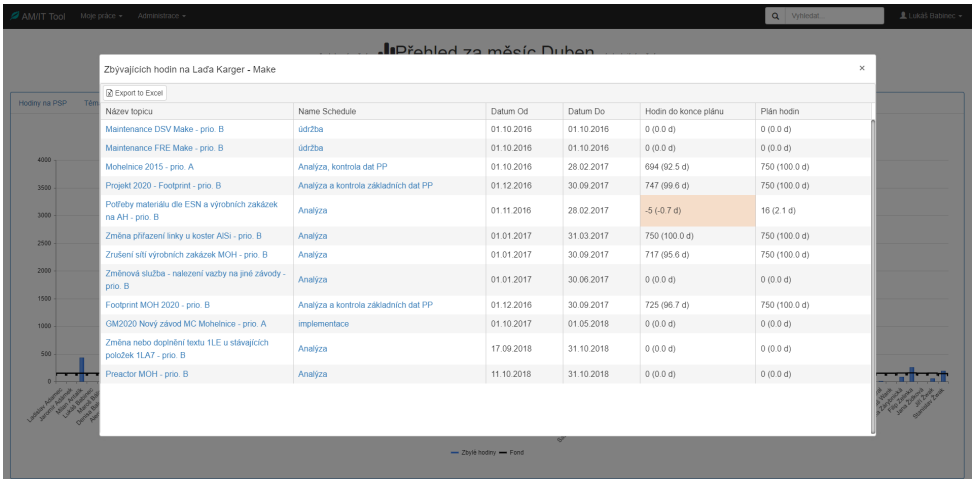
Tabulka 6: Řádek Výsledku dotazu

Název výsledného atributu	Hodnota	Popis
DevId	80	id vývojáře
DevName	Lukáš Babinec	celé jméno vývojáře
DevTeam	Deliver	název týmu vývojáře
ToId	245	id tématu
ToName	Plánování v nástrojárně - prio. B	název tématu
TsID	4037	id zapsání osoby k tématu
TsName	Implementace	úloha osoby k tématu
TsFrom	2017-01-01	datum jeho zapsání od
TsTo	2019-09-30	datum jeho zapsání do
TsPlan	450	hodin zapsaných na plán
zbyle_hodiny	421	zbývající hodiny na plánu
Fond	50	část fondu, na plán

Aplikujeme agregační funkci suma nad hodinami a seskupíme podle všech zbylých atributů. Výsledek a detailní tabulku vidíme na Obrázku 8 a 9.



Obrázek 8: Graf Vytíženosti vývojářů



Obrázek 9: Tabulka detailu Vytíženosti konkrétního vývojáře

5.3 Success Story

5.3.1 Umožnit uživatelům vkládat a zobrazovat formátovaný obsah

Úkol zněl implementovat pro dvě vstupní textové oblasti nějakou formu textového editoru, která nám umožní text formátovat a takto formátovaný text i zobrazovat.

Řeč je tedy o WYSIWYG, což představuje akronym z anglické věty "What you see is what you get“, česky „co vidíš, to dostaneš“. Tím míníme na první zdání textový editor, který však pracuje na pozadí s HTML a CSS prvky. Na tomto principu jsou postaveny celé produkty, které tímto způsobem nabízejí, tvorbu vlastních webových stránek.

V tomto případě Kendo UI nabízí pro tento případ řešení. V jejich nabídce najdeme komponentu s příhodným názvem *Editor*. Editor umožňuje veškeré funkcionality, které potřebujeme. Změna velikosti písma, volbu tučného písma, vložení číslovaného seznamu, změnu barvy písma a mnoho jiného, avšak pro náš případ využijeme pouze zmíněných hlavních funkcí.

Použití je následovné. Mějme v našem HTML dokumentu formulář, uvnitř něj element *TextArea*, kterému přiřadíme *id* pro jeho snadnou selekci v dokumentu. Pro náš příklad budeme hovořit textovém zadávání pro popis.

```
<textarea id="popis" name="Description"></textarea>
```

Výpis 13: Element TextArea

Jakmile je náš DOM s elementem vykreslen můžeme nad ním vytvořit Kendo *Editor* způsobem, jakým jsme si již ukázali v sekci Kendo UI. Při vytváření můžeme v jeho konfiguraci nastavit, které ovládací prvky by měl *Editor* obsahovat.

```
$("#vyhodnoceni").kendoEditor({
    tools: [
        "bold",
        "italic",
        "underline",
        "justifyLeft",
        "justifyCenter",
        "foreColor",
        "insertOrderedList",
        "insertUnorderedList",
        "fontSize"
    ],
    encoded: false,
});
// obsah prvku Editor
$("#vyhodnoceni").val();
```

Výpis 14: Vytvoření Editoru

Dalším klíčovým atributem, který zde nastavíme je *encoded*, tento atribut máme nastaven na *false* z důvodu, že chceme ukládat přímo do naší databáze nezpracovaný HTML kód. Zde však můžeme narazit na problém, že pokud bychom chtěli obsah editoru, odeslat do modelu nebylo by to možné. .NET modely totiž implicitně nedovolují, aby se proměnné typu *string* plnily HTML kódem z bezpečnostních důvodů. Toto však můžeme změnit přidáním "atributu" s názvem *AllowHtml* (viz Výpis 15).

```
[AllowHtml]
public string Description { get; set; }
```

Výpis 15: Meta tag promněné modelu

Již máme vytvořený editor, víme jak přistupovat k jeho obsahu a máme zajištěno, že když ho odešleme v modelu do kontroléru, kontrolér data v pořádku obdrží. Takto obdržená data již snadno uložíme do databáze, teď nám zbývá formátované údaje zobrazit.

Představme si, že již námi zmiňovaný *DataSource* zažádal o data a my je chceme v našem formuláři zobrazit. Model uložený na klientské stranně obsahuje proměnnou *Description*, obsahující námi zformátovaný text v HTML podobě.

Mějme v našem formuláři *span* element, který můžeme snadno vybrat a do jeho atributu s názvem *innerHtml* přidejme námi zmíněný HTML kód. To bude mít za výsledek zobrazení formátovaného textu uvnitř *span* elementu.

```
$('#Description')[0].innerHTML = model.Description;
```

Výpis 16: Naplnění elementu formátovaným obsahem

6 Závěr

6.1 Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe

V průběhu praxe jsem stavěl své fungování na znalostech nabytých během mého studia, neboť tyto znalosti, jak praktické, tak i teoretické, mi napomáhaly při řešení zadaných úloh. Teoretický základ mi napomáhal rychle se zorientovat u úloh, a díky tomu pro mě většina pojmů se kterými jsem se setkal nebyly novými. Tím mám především namysli obecné znalosti komunikací, technických pojmů i databázových systémů. Velkou výhodou při vykonávání mé praxe byla dle mého názoru dobrá znalost .NET platformy. Díky tomu jsem rychle dokázal pochopit principiální funkčnost cizího kódu. Použití různých třídních metod nebo reflexe pro mě nebylo cizí. Avšak nešlo pouze o chápání již napsaných kódů, ale mohl jsem zde aplikovat a procvičit si nabyté znalosti efektivního a dobře čitelného objektově orientovaného kódu. Znalosti mi také napomohly při odhaleni a vyladění kódu, který zrovna nebyl příkladem toho jak by se pro .NET platformu psát mělo.

Velkým přínosem při plnění mé praxe byla znalost práce s databázemi. Ať už šlo o tvoření databází nových nebo jen jejich správu. Možnost tréninku, který škola poskytla při psaní nejrůznějších SQL dotazů byl velký. Díky tomuto jsem mohl tvořit dotazy a investovat svůj čas do logiky v datech, které jsem chtěl zpracovávat. Velmi dobrou zkušeností byla také práce s prostředím Management Studio, které pro mě tím pádem nebylo nové a mohl jsem se v něm snadno orientovat. To vše za podkladu dobře obdržených teoretických znalostí databázových systémů.

6.2 Znalosti či dovednosti scházející studentovi v průběhu odborné praxe

Znalosti, které bych označil jako nedostačující hned z počátku praxe byly rozhodně ty, které se týkaly vývoje webových aplikací. Musel jsem formou samostudia absolvovat několik kurzů na vývoj aplikací v ASP.NET a obstarávání požadavků na klientské straně za pomocí JavaScriptu. Jakmile jsem zjistil jak, implementovat základní aspekty webových aplikací jako jsou například *sessions*, mohl jsem se začít více do hloubky zaobírat tím, jaké mají tyto aspekty výhody a nevýhody, jaké jiné možnosti řešení klasických problémů se v dnešní době implementují. Mohl jsem se začít dále více zabývat správnou a efektivní strukturou jednotlivých souborů takové aplikace, nebo také jaké jsou ty nejzákladnější otázky tykající se zabezpečení.

Nové pro mě také bylo vůbec o povědomí o existujícím podnikovém informačním systému SAP. Získané znalosti pouvažuji za velice přínosné, jelikož je tento systém v mnoha firmách implementován a proto informace o něm a schopnost jeho využití považuji za užitečné při mém uplatňování na trhu práce.

6.3 Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Za svou praxi jsem se seznámil s organizační strukturou společnosti Siemens s. r. o. Byl jsem zde zaměstnán jako vývojář interních webových aplikací. Jako velký přínos považuji seznámení se s reálným procesem vývoje softwaru od obdržení zadání až po samotné nasazení a řízení dalšího provozu. Jako další užitečnou zkušenost považuji být součástí týmu, kde je využíváno principů agilního programování. Obohacující byl také náhled do samotné společnosti a poznání jejího fungování.

Za dobu mého působení ve firmě mi bylo předáno na starost několik již rozpracovaných aplikací, ke kterým dále vyvstávaly požadavky. Seznámil jsem se s interní aplikací AM/IT, u které jsem strávil delší čas nad pochopením řešení její databáze a následně jsem nad její strukturou vytvářel přehledy, které byly poté implementovány do webového uživatelského rozhraní. U této aplikace jsem se také poprvé setkal s Kendo UI, u kterého jsem pocítil nezbytnost využití jemu podobného rozhraní při vytváření webových aplikací. Implementace takového rozhraní nám ušetří spoustu práce při řešení nejrůznějších obecných problémů.

Mezi mé další úkoly, při kterých jsem si mohl vyzkoušet celý proces vytváření vlastních projektů i jejich samotné nasazení, patřilo vytváření informačních tabulí. Tyto tabule měly různou komplexnost, některé využívaly lidskou interakci, některé měly za úkol měnit v časovém horizontu zobrazované informace. Při těchto úkolech probíhaly diskuze s řešiteli funkčních modulů SAP, které jsem dále volal a získával z nich data.

Jako další zadání jsem obdržel na starost aplikaci Success Story, ke které vyvstalo několik požadavků na úpravy. Šlo o změny uživatelského rozhraní, druh i množství zadávaných dat, vizuální změny a také rozšíření o textový editor, jehož implementaci jsem se v práci věnoval.

Hlavním přínosem absolvované praxe bylo získání zkušeností s tím, jak postupovat při řešení zadaných úloh, při kterých je potřeba myslet na to, že se jedná o reálný produkt, který bude využíván některými interními zaměstnanci denně. Musel jsem být pečlivý při eliminaci chyb, jelikož musel být zajištěn u těchto aplikací takřka neomezený provoz. Cítil jsem jistou zodpovědnost při práci s daty, která jsem zpracovával, tudíž jsem musel mít na mysli i jejich korektní zpracování.

Literatura

[1] KADLEC, Václav. Agilní programování : Metodiky efektivního vývoje softwaru. Brno : Computer Press, 2004. 280 s. ISBN 80-251-0342-0.

[2] Daniel Abadi; Samuel Madden (31 July 2008). "Debunking Another Myth: Column-Stores vs. Vertical Partitioning". The Database Column. Archived from the original on December 4, 2008.

[3] RFC documentation by SAP

[4] Kendo-UI documentation

[5] “The World’s Biggest Public Companies, Software & Programming.” Forbes. Retrieved 21 October 2016.