FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	MANTA - aplikace pro management test analýzy
Student:	Lenka Pohanová
Vedoucí:	Ing. Bc. Miroslav Benda
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Cílem práce je navrhnout, vyvinout a následně implementovat webovou aplikaci pro management přípravy test analýzy v rámci IT oddělení.
Aplikace nahradí dosavadní řešení, kde se data udržují v tabulkovém souboru umístěném na sdíleném disku. Toto řešení je nevyhovující, jelikož při souběžné práci více uživatelů se přepisují data. Také je poměrně značné riziko nechtěného odstranění dat. Tyto problémy má nová aplikace minimalizovat a tím zefektivní práci.
Požadavky na aplikaci:
1) Zobrazení všech projektů a jejich atributů jako tabulku
2) Editace a archivace projektů
3) Export projektů do tabulkového souboru
4) Rozeznávání rolí uživatelů a jejich oprávnění
5) Použití ASP.NET
6) Nesmí docházet ke ztrátě dat v důsledku práce více uživatelů
Další úkoly:
1) Popište podobná existující řešení, návrh a implementaci aplikace
2) Zdokumentujte uživatelské požadavky a průběh testování
3) Zdůvodněte volbu použitých technologií

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 3. ledna 2019

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

MANTA - aplikace pro management test analýzy

Lenka Pohanová

Katedra softwarového inženýrství
Vedoucí práce: Ing. Bc. Miroslav Benda

16. května 2019

Poděkování

Chtěla bych poděkovat svému vedoucímu práce, a tím byl pan Ing. Bc. Miroslav Benda. A dále všem zaměstnancům CETINu, kteří mi pomáhali s testováním a s napojením aplikace na firemní systémy.

Prohlášení

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

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 16. května 2019 ……

Odkaz na tuto práci

Pohanová, Lenka. *MANTA - aplikace pro management test analýzy.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

Práce tvoří jádro budoucí aplikace pro firmu CETIN. Nahradí dosavadní Excel tabulku webovou aplikací. Hlavní prvek aplikace je tedy tabulka, která zachovává potřebnou funkcionalitu Excel tabulky, a to možnost podbarvit buňky tabulky, měnit, mazat nebo archivovat záznamy. Aplikace zvýší ochranu dat, především před nechtěnou editací. Aplikace využívá Windows autentizaci pro autorizaci jednotlivých rolí. Jednotlivé role mají různý přístup ke sloupcům tabulky i k funkcionalitě. Tyto přístupy lze nastavit přímo v aplikaci. Data jsou uložena v databázi. Použita byla architektura MVVM, která vychází z architektury MVC. Aplikace je napsána v C#, pomocí Visual Studia.

Klíčová slova C#, webová aplikace, MVVM, autorizace, autentizace, export do .xlsx, HTML, zobrazení dat z databáze

Abstract

The thesis forms the core of future application for CETIN. Replaces the existing Excel spreadsheet with a web application. The main element of the application is a table that maintains the necessary functionality of the Excel table, namely the ability to color the table cells, change, delete or archive records. The application will increase data protection, especially against unwanted editing. The application uses Windows authentication to authorize individual roles. Individual roles have different access to table columns and functionality. These approaches can be set directly in the application. The data is stored in the database. MVVM was used, based on MVC architecture. The application is written in C, using Visual Studio.

Keywords C#, web application, MVVM, authorization, authentication, export to .xlsx, HTML, view data from database

Obsah

Seznam obrázků

Seznam výpisů kódu

Úvod

Dnešní svět je hlavně o informacích. Ale v dnešní uspěchané době nestačí informaci pouze získat, musí se získat rychle a pohodlně. Pro firmy jsou informace existenčně důležité, jejich zničení může znamenat nemalé finanční ztráty. Ale i znehodnocení méně důležitých dat je nepříjemné, k jejich opětovnému zjišťování potřebujeme čas navíc, a nemáme vždy jistotu, že informace budou totožné. A právě uchování informací řeším v této práci.

Práce je určena pro testovací oddělení firmy CETIN (Česká telekomunikační infrastruktura a. s.), kde momentálně pracuji. Nahrazuje dosavadní Excel tabulku, kde se uchovávají informace o aktuálním stavu jednotlivých změn, které má oddělení otestovat. S touto tabulkou se velmi intenzivně pracuje, a velice snadno může dojít k rozházení dat nebo jejich smazání. Každá změna je na jeden řádek a pomocí sloupců se filtrují hledané záznamy. Navíc se velmi často informace aktualizují z jiných systémů, bohužel ručně. Toto řešení je tedy použitelné, ale značně nepohodlné. A proto jsem se rozhodla ho nahradit.

Cílem je tedy vlastní systém na ukládání a úpravu dat. Kvůli snadné dostupnosti vytvořím Webovou aplikaci, která bude přístupná pouze zaměstnancům CETINu. Pro bezpečnější práci s daty použiji databázi. Tím zamezím nechtěnému proházení dat. Další neméně důležitou výhodou oproti současnosti je možnost práce více uživatelů současně.

V teoretické části se soustředím na klíčové rozhodnutí ohledně technologií a postupů. Nejvíce se budu věnovat volbě jazyků a architektury. Vzhledem k citlivosti ukládaných informací se zmíním i o zabezpečení dat. Poté popíšu proces vzniku aplikace. Od návrhu přes implementaci až k testování.

Kapitola 1

Cíl práce

Cílem praktické části je nahradit základní použití Excel tabulky webovou aplikací, která vyhoví potřebám testovacímu oddělení firmy CETIN. Aplikace bude napsána pomocí ASP.NET.

Aplikace umožní:

- zobrazovat tabulku s aktuálními změnami pro testování,
- editaci a archivaci těchto změn,
- archivaci do tabulky podobné hlavní tabulce,
- export změn pro testování do tabulkového souboru,
- upravit zobrazení dat podle oprávnění,
- povolit činnosti v závislosti na oprávnění,
- bezpečnou editaci hlavní tabulky.

Uživatelské požadavky popíši v části Analýza a návrh. Dále se budu věnovat také podobným existujícím řešením. A v teoretické části zdůvodním volbu použitých technologií a popíši návrh aplikace. Samozřejmě se budu věnovat implementaci aplikace a průběhu testování.

Kapitola 2

Teorie

2.1 Technologie

Výběr technologií se mi značně zjednodušil požadavkem, že se bude jednat o webovou aplikaci. Uvažovala jsem hlavně nad Enterprise Java a C#. Nakonec jsem si zvolila C#. Mé důvody si můžete přečíst níže viz kapitola 4.2: Výběr technologií a architektury.

Zde bych ráda zdůraznila hlavní rozdíl mezi webovou a desktopovou aplikací. Webová aplikace je bezstavová, to znamená, že v okamžiku, kdy vykreslíme webovou obrazovku uživateli, server již nemá o této stránce informace, a nijak se o ní nestará. V okamžiku, kdy uživatel vytvoří nějakou akci, například kliknutí na tlačítko, teprve začíná server znovu reagovat. Vytváří novou stránku bez znalosti, co je momentálně uživateli vykresleno. Toto má za důsledek, že nelze uchovávat změny v proměnných, vždy se vytvoří nová instance. Na tuto vlastnost je nutné myslet během celého návrhu i vývoje.

2.1.1 Enterprise Java

Enterprise Java je obecně často používaný jazyk pro tvorbu rozsáhlých podnikových aplikací. Patří pod Oracle, více zde [1]. Avšak do vývoje mohou značně zasahovat jiné firmy, tím mám na mysli hlavně návrhy rozšíření. Tím se stala Java Enterprise velmi nepřehlednou a syntakticky nekonzistentní, zvláště pro začínající vývojáře. Pokud toto ovšem není pro někoho problém, lze v Enterprise Java vytvořit programátorsky přehledné webové aplikace. Pro mou malou aplikaci je to však naprosto zbytečné. Ve vývoji by mne právě velikost této technologie a nutnost přesného dodržení šablon značně zdržovala.

2.1.2 C#

C# je technologie, kterou vytvořila společnost Microsoft, inspiraci hledal v technologiích C++ a Java. Z obou jazyků si vzal to nejlepší. Přehlednost kódu z C++ a vyšší pohled na programování z Javy. Více se dočtete zde [2]. Microsoft udržuje syntaxi jednotnou a snadno pochopitelnou.

2.1.3 .NET Framework

.NET Framework je systém skládající se z několika částí, které se starají o bezpečný vývoj a běh programů. Patří sem například běhové prostředí CLR, základní třídy systému .NET a samozřejmě i ASP.NET. Nyní se na tyto části podíváme podrobněji.

2.1.3.1 Běhové prostředí CLR

CLR je zkratkou Common Language Runtime. Tato část .NET Frameworku se stará o spuštění programů a přiřazuje jim zdroje, například paměť. Program se však nespouští z kódu napsaného programátorem, ale z metadat vytvořených pomocí MSIL.

MSIL neboli Microsoft Intermediate Language je mezijazyk, který lze vygenerovat hned z několika jazyků. Tyto jazyky upravuje do jednotného formátu.

Microsoft podporuje přímo tyto jazyky:

- C#
- C++
- Visual Basic .NET
- JScript.NET

Některé další použitelné jazyky bez podpory Microsoftu:

- Borland
- Delphi 7

Díky MSIL je tedy možné napsat .NET aplikaci v libovolném z těchto jazyků, dokonce je možné mít každou část aplikace v jiném jazyku, což se může ve velkých vývojových týmech hodit, zvláště když každý člen má jiné zkušenosti. CLR překládá do spustitelného kódu MSIL až když je konkrétní podprogram potřeba. V této době již totiž zná všechny potřebné informace pro překlad do strojového kódu konkrétního počítače.

2.1.3.2 Základní třídy systému .NET

Právě možnost používat více jazyků vedla k nutnosti vytvořit alespoň základní rozhraní, které musí všechny jazyky splňovat. Toto rozhraní nabízí různé funkce, datové typy a objekty. Pro bližší představu sem patří například datový typ String.

2.1.3.3 ASP.NET

ASP.NET je technologie, která má základy postavené na .NET Frameworku. Jedná se o soubor funkcí a objektů, díky nimž lze vytvářet webové stránky na serveru a uživateli zaslat již hotové HTML stránky. Jedná se tedy o model klient/server, který je rozšířen o události.

Model klient/server v ASP.NET se nejlépe vysvětluje na procesu komunikace webové stránky s uživatelem. Celý proces začíná zadáním URL adresy do prohlížeče. Adresa se předá serveru, který požadovanou stránku vyhledá a přeloží ji pomoci CLR. Poté spustí vytvořený kód, z kterého na serveru vytvoří HTML kód. Teprve tento již hotový HTML kód je poslán zpět do prohlížeče, který zobrazí stránku uživateli.

Události vznikají na základě nějaké akce uživatele, například klikne na tlačítko. Prohlížeč o této události informuje server pomocí JavaScriptu, který vkládá již během sestavování do HTML kódu. Server na tuto událost reaguje podle potřeby například změněnou obsahu stránky, kterou pak odešle zpátky klientovi, který ji standardně zobrazí.

Právě možnost řídit aplikaci pomocí událostí dovoluje vytvářet značně interaktivní stránky. Mezi další výhodu oproti jiným technologiím je výše popsané běhové prostředí CLR, díky němuž není technologie ASP.NET závislá na konkrétním programovacím jazyce a díky kompilaci je i rychlejší než jiné jazyky, které se nekompilují ale interpretují. A v neposlední řadě je možné poměrně snadno vytvářet v ASP.NET propracované webové stránky. Pokud vás více zajímá čistá ASP.NET aplikace, tak se více dozvíte zde [3].

2.2 Prezenční vrstva

Jako technologii jsem si zvolila ASP.NET, tudíž jsem měla dvě možnosti pro prezenční vrstvu, a to čisté ASP.NET, nebo Razor. Nakonec jsem si zvolila Razor, hlavně pro jeho jednoduchost. Více se dozvíte v kapitole 2.2.3: Razor. Razor je postaven na základech HTML, které rozšiřuje. Také jsem použila CSS, proto začnu postupně od HTML přes CSS a nakonec se dostanu až k cílové Razor syntaxi.

2.2.1 HTML

HTML je zkratkou Hypertext Markup Language. Jedná se o značkovací jazyk, který se používá pro tvorbu webových stránek. Je to základ, který by měli ovládat všichni, kteří se chtějí věnovat webovým stránkám. Více o HTML se dozvíte na těchto stránkách [4].

Zajímavé je, že HTML nevzniklo jako jazyk pro formátování. Původním účelem bylo zaznamenat strukturu dokumentů a jejich vazeb. Tento původní standard začali rozšiřovat výrobci prohlížečů, kteří si přidali vlastní tagy například pro fonty písma. Reagovali tím na požadavky autorů webových stránek i jejich uživatelů. Podrobněji tuto problematiku naleznete zde [5]. Tento roztroušený vývoj je znát dodnes, kdy se může stát, že se stejná stránka zobrazí na různých prohlížečích různě, nebo je potřeba pro stejnou funkcionalitu využít jiný tag. Proto je vhodné otestovat svůj kód na více prohlížečích.

2.2.2 CSS

CSS neboli kaskádové styly umožňují formátovat dokumenty pomocí hromadných pravidel. Můžeme tak například na jednom místě určit jak budou vypadat tabulky, nadpisy a další prvky HTML kódu. Díky tomuto přístupu se můžeme v HTML kódu soustředit pouze na strukturu dokumentu a grafickou podobu řešit později. Tento postup i značně zjednodušuje případnou změnu vzhledu webové stránky.

Přestože HTML tagy pro in-line styly jsou stále používané a mají svůj smysl, jejich vývoj byl již zastaven. Nelze tedy očekávat nová vylepšení nebo inovace. Stále však budou podporovány prohlížeči, takže je lze bez obav dále používat. Pro menší webové stránky, kde každá stránka vypadá jinak je to dokonce i vhodné. Více o kaskádových stylech se dozvíte v této knížce [5].

CSS styly nacházejí své uplatnění hlavně na rozsáhlých webových stránkách, kde značně usnadní dodržení jednotného stylu. Běžně se také používají při dodání menší funkcionality, nebo formuláře jinou firmou. Firma dodá jednomu nebo více zákazníkům základní strukturu produktu a zákazník si ji právě pomocí CSS stylů přizpůsobí vzhledu svých stránek. Pokud je použití CSS stylů navrženo kvalitně, uživatel webových stránek nemá téměř šanci poznat, že tato část stránky je od jiného vývojáře.

2.2.3 Razor

Syntaxe Razor je určena pro ASP.NET aplikace. Jeho použití se snadno rozeznatelné díky použití `@výraz`, případně `@{blok výrazů}`. Kromě toho jsou soubory obsahující syntaxi Razor označené příponou .cshtml nebo případně .vbhtml, díky nimž server rozpozná, že se jedná o Razor rozšíření.

Syntaxe Razor byla vytvořena jako zjednodušení ASP.NET pro začínající programátory webových aplikací. Hlavně pokud znají některý z jazyků C, C++, C#, Visual Basic nebo JavaScript, kterým se tato syntaxe značně podobá. Více se o základní syntaxi Razor dozvíte zde [6].

Přestože se jedná o zjednodušení až na úroveň PHP, nutí programátora dodržovat základní bezpečnostní pravidla i tvořit přehlednější kód, Navíc usnadňuje přechod k samotnému ASP.NET, který umožňuje vytvářet mnohem sofistikovanější webové stránky. Podrobnější názor zkušenějšího vývojáře naleznete zde [7]. S tímto názorem musím po zkušenosti se syntaxí Razor více méně souhlasit. Opravdu se podle mého názoru povedlo Microsoftu vytvořit jednoduchou syntaxi, se kterou lze jednoduše vytvořit hezčí a funkčnější stránky než v samotném HTML. I když samozřejmě by mi čisté ASP.NET umožnilo mnohem víc, pro tuto bakalářskou práci mi Razor bohatě stačil.

2.3 Architektura

Při tvorbě webových stránek, které uchovávají nějaká data v databázi, se téměř vždy používá některá vícevrstvá architektura. Je to dáno i tím, že je potřeba ve webové aplikaci spojit několik rozdílných činností. V běžných webových stránkách to bývá komunikace s databází, zobrazení dat uživateli a kód, který to vše řídí. I proto je asi architektura MVC nejpoužívanější a má i nejvíce možných rozšíření. Zde se budu věnovat těm, které jsem využila ve své práci.

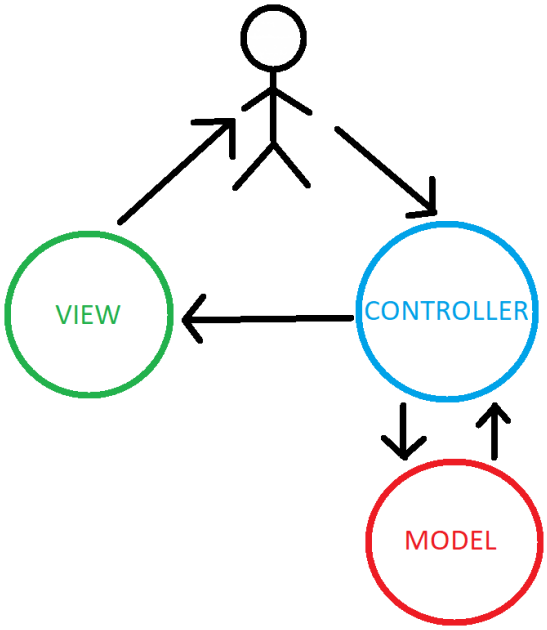
2.3.1 MVC

MVC je zkratka slov model, view a controller. Toto rozdělení představuje základní rozdělení funkcionality, a v mnoha případech je postačující. Pro základní představu využiji obrázek 2.1: MVC, který představuje základní komunikaci architektury MVC.

Model předá data controlleru, ten řekne view, aby data zobrazil. View vytvoří výslednou stránku z předaných dat uživateli. Poté controller čeká na nějakou akci uživatele, například kliknutí na tlačítko, a podle potřeby pošle data do modelu, nebo vyzvedne nová data. Nakonec nechá vytvořit nové view a čeká na další příkaz.

2.3.1.1 Model

Model se hlavně zabývá vkládáním dat do aplikace a jejich čtením. Většinou mapuje data z databáze do objektové formy, která se snadno používá v dalších částech aplikace. Je možné zde dodat i některé validace, abychom neuložili neplatná data. Model se tedy striktně stará jen o práci s daty, neví odkud data přišla, ani jak se zobrazí uživateli.



Obrázek 2.1: MVC

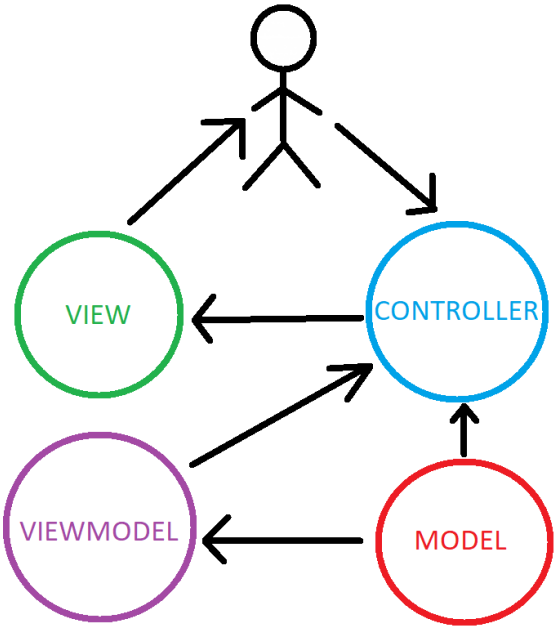
2.3.1.2 View

View se stará o zobrazení dat uživateli. Neví nic o tom, odkud data přišla, pouze je zobrazí. View je nejčastěji napsáno v HTML, případně doplněno jiným jazykem umožňujícím vkládat proměnné nebo cykly. Pokud view obsahuje taková rozšíření, pak se tyto proměnné a cykly přeloží ještě na straně serveru a na klientský počítač se posílá čisté HTML.

2.3.1.3 Controller

Controller řídí celou aplikaci. Přijme pokyny od uživatele a rozděluje práci mezi model a view. V controlleru se mohou vyskytovat složitější funkce i výpočty. Každé tlačítko, každá validní URL adresa nejprve zavolá příslušnou metodu v controlleru.

Hlavní nevýhodou architektury MVC je, že platí pravidlo které říká, že jedno view se stará jeden controller, který ovládá jeden model. Jinými slovy na jedné stránce nelze zobrazit data z více tabulek. Proto vznikla různá rozšíření této architektury. Více o architektuře MVC se dozvíte zde [8].



Obrázek 2.2: MVVM

2.3.2 MVVM

MVVM rozšiřuje strukturu MVC o jednu vrstvu, té říkáme viewmodel. Díky této vrstvě můžeme na jedné stránce zobrazit data z více tabulek. Viewmodel vrstva si načte všechny modely, které potřebuje, a jako jeden objekt je pošle do controlleru, ten s viewmodelem pracuje jako s jedním objektem. A ten již umí poslat do view. Obsáhlejší teoretický výklad naleznete zde [9]. Pokud vás zajímá, jak MVVM architekturu vytvořit v MVC šabloně, přejděte do kapitoly 5.2: Architektura MVVM v šabloně pro MVC.

2.3.3 ViewBag

ViewBag je odvozen z třídy `DynamicViewData`, tvoří jakýsi obal ViewData. Díky této vlastnosti nemusíme použít přetypování. Jeho životnost je dána životností zobrazení, do kterého je poslána. ViewBag je také pohodlnější na použití. Viz ukázka kódu 2.1: ViewBag. První řádek patří do controlleru, nastavíme jím hodnoty do ViewBag. Poté následuje ukázka, jak číst data ve View. Důležité je zdůraznit, že data lze posílat jen ve směru z controlleru do view. Více se o zobrazení můžete dočíst zde [10].

```
 ViewBag.Settings = db.Settings_Name_Columns.First();

@{
    ViewBag.Title = "Setting_Edit_Autorization";
    Layout = "~/Views/Shared/_Layout.cshtml";
    var Column_Name = ViewBag.Settings;
}
<td>
        @Column_Name.Name_Column_1
</td>
```

Výpis kódu 2.1: ViewBag

2.3.4 ViewData

ViewData se od ViewBag liší jen nepatrně. Nejedná se o totiž dynamický typ. ViewData se odvozuje od objektu `ViewDataDictionary`. To znamená, že se jedná o slovník, kde klíč je jakýkoli náš string a hodnota je objekt. Jelikož view již neví o jaký objekt se jedná, musíme mu to říct. Viz ukázka kódu 2.2: ViewData, kde první tři řádky patří do controlleru a zbytek do view, jako v předchozí ukázce. Na tomto příkladu je i názorně vidět, že ViewData a ViewBag lze kombinovat. Více se o zobrazení můžete dočíst zde [10].

```
 Settings_Name_Columns Sett = new Settings_Name_Columns();
            Sett = db.Settings_Name_Columns.First();
            ViewData["Name_Column"] = Sett;

@{
    ViewBag.Title = "Edit";
    Manta_dev_Data.Settings_Name_Columns SettingsList =
   (Manta_dev_Data.Settings_Name_Columns)ViewData["Name_Column"];
}

<th>
        @SettingsList.Name_Column_1:
</th>
```

Výpis kódu 2.2: ViewData

2.4 Autentizace a autorizace

Pojmy autentizace a autorizace se často zaměňují, je však mezi nimi značný rozdíl. Při autentizaci se ověřuje uživatel, je mu přiřazena nějaká identita. Slouží pouze k identifikaci. Oproti tomu autorizace se využívá k stanovení práv a omezení pro konkrétního uživatele, k jeho identifikaci se využívá autentizace. Více se dozvíte v kapitole 5.7: Oprávnění a role.

2.5 XLSX formát

Soubor XLSX je formát sešitu aplikace Excel. Tento formát se poprvé objevil v Excelu 2007 a vychází z formátu XLS, na rozdíl od formátu XLS nepodporuje formát XLSX makra. Formát je založen na jazyce XML a využívá kompresi ZIP. Díky tomu je paměťově úspornější. Více se dozvíte zde [11]. XLSX byl také prohlášen ISO formátem, lze ho najít pod identifikátorem ISO 29500: 2008-2016. Obsáhlý popis tohoto formátu včetně historie, porovnání s jinými formáty, licence, dokumentace a dalších informací naleznete zde [12].

Kapitola 3

Podobná řešení

V této kapitole popíši podobná existující řešení, která jsem našla a zhodnotím jejich vlastnosti. Porovnám jejich výhody a nevýhody oproti mému řešení. Také uvedu důvody, proč tato jiná řešení nebyla vhodná pro oddělení testingu firmy CETIN. Ale nejprve popíši současný stav, kdy se data ukládají v programu Excel, a proč je nevyhovující.

3.1 Excel

Excel patří do kancelářského balíčku firmy Microsoft. Tento program je hojně používán a obsahuje mnoho funkcí pro práci s daty. Je uživatelsky přívětivý a jeho základní ovládání vcelku intuitivní. Více naleznete zde [13].

Hlavní nevýhodou, kvůli které toto řešení není ideální, je, že jako základní samostatnou informaci považuje obsah jedné buňky. Tato vlastnost má jednu nevýhodu a to tu, že při filtrování a řazení se občas prohází záznamy v rámci řádku, aniž by na tuto zkutečnost upozornil. Toto chování je důvod, proč je tento systém pro oddělení testingu nevyhovující.

3.2 Calc

Calc patří do kancelářského balíčku LibreOffice, je velmi podobný Excelu. Jeho použití by tedy nemělo vliv na vyřešení problému s Excelem, funguje totiž na podobných principech. Hlavní rozdíl je, že LibreOffice je svobodný software. Více naleznete zde [14].

3.3 Jira

Jira je produkt společnosti ATLASSIAN. ATLASSIAN se zaměřuje na podporu týmové práce. A to od správy firmy až po spolupráci na dokumentech. Aplikace Jira se soustředí na sledování požadavků a projektů. Lze v ní vytvořit časový plán i přiřadit jednotlivé úkoly.

Je ovšem navržen pro agilní vývoj, jeho použití pro vývoj v rámci release by bylo nepohodlné, navíc se jedná o placenou aplikaci. Více naleznete zde [15].

3.4 Redmine

Redmine je open source řešení pro správu projektů. Podporuje hlavně agilní vývoj. Pro účely jednoho oddělení je příliš komplikovaný a práce v něm by byla pomalá. Více se dozvíte zde [16]. Funkčně se spíše blíží k řízení celých projektů, a ne k jedné části vývoje.

3.5 Easy Project

Easy Project je další z placených nástrojů. Je přehledný, ale opět má až zbytečně mnoho funkcí pro jedno oddělení. Je také určen hlavně na agilní vývoj, i když podporuje i Scrum. Pokud vás tento nástroj zajímá, dočtete se o něm více na jejich stránkách [17].

3.6 People manager

People manager jako jediná aplikace z mého výběru podporuje plánování po releasech. Bohužel se opět jedná o aplikaci, která hlídá celý životní cyklus produktu. Přestože nevypadá tak sofistikovaně jako předchozí aplikace, je také placená. Více opět naleznete zde [18].

V souhrnu by se tedy dalo říct, že existující řešení jsou buď stejná jako původní řešení, nebo placená. Navíc všechny zmíněné aplikace slouží hlavně pro řízení celých projektů a ne jedné fáze vývoje. V budoucnu se také plánuje napojení vyvíjené aplikace na firemní systémy. Aplikace je tvořena s respektem k způsobu práce oddělení.

Kapitola 4

Analýza a návrh

V této kapitole se soustředím na uživatelské požadavky a na změny původního návrhu během vývoje. Pokračovat budu výběrem technologií, architektury a grafické podoby. Nakonec se budu věnovat důvodům, proč jsem řešila přidání nových sloupců pomocí rezervních sloupců.

4.1 První návrh

Při zhodnocování základních požadavků jsem vycházela z používané funkcionality Excel tabulky, kterou má aplikace nahradí a rozšíří potřebným směrem. A i když rozsah této bakalářské práce plně nepokrývá celou budoucí funkcionalitu, je již v použitelném stavu. Po několika konzultacích jsme stanovili tyto požadavky.

Funkční požadavky:

- ukládání a editace záznamů o jednotlivých WP,
- archivace starých záznamů,
- export do formátu XLSX,
- rozeznání uživatelů a jejich rolí,
- zobrazovat některé informace jen některým uživatelům podle jejich rolí,
- možnost podbarvit některé buňky,
- univerzálnost obsahu jednotlivých buněk (textový řetězec),
- pro roli admin, možnost měnit názvy sloupců z webového rozhraní,
- pro roli admin, možnost měnit viditelnost sloupců pro konkrétní role z webového rozhraní.

Nefunkční požadavky:

- webová aplikace,
- ukládání dat v databázi,
- jazyk C#.

Pro lepší domluvu jsem vytvořila první verzi základní obrazovky pomocí HTML viz obrázek 4.1: První návrh obrazovky, prozatím bez jakýchkoli dat. Tento základní obrázek velmi pomohl při domluvě konečné podoby aplikace.

4.2 Výběr technologií a architektury

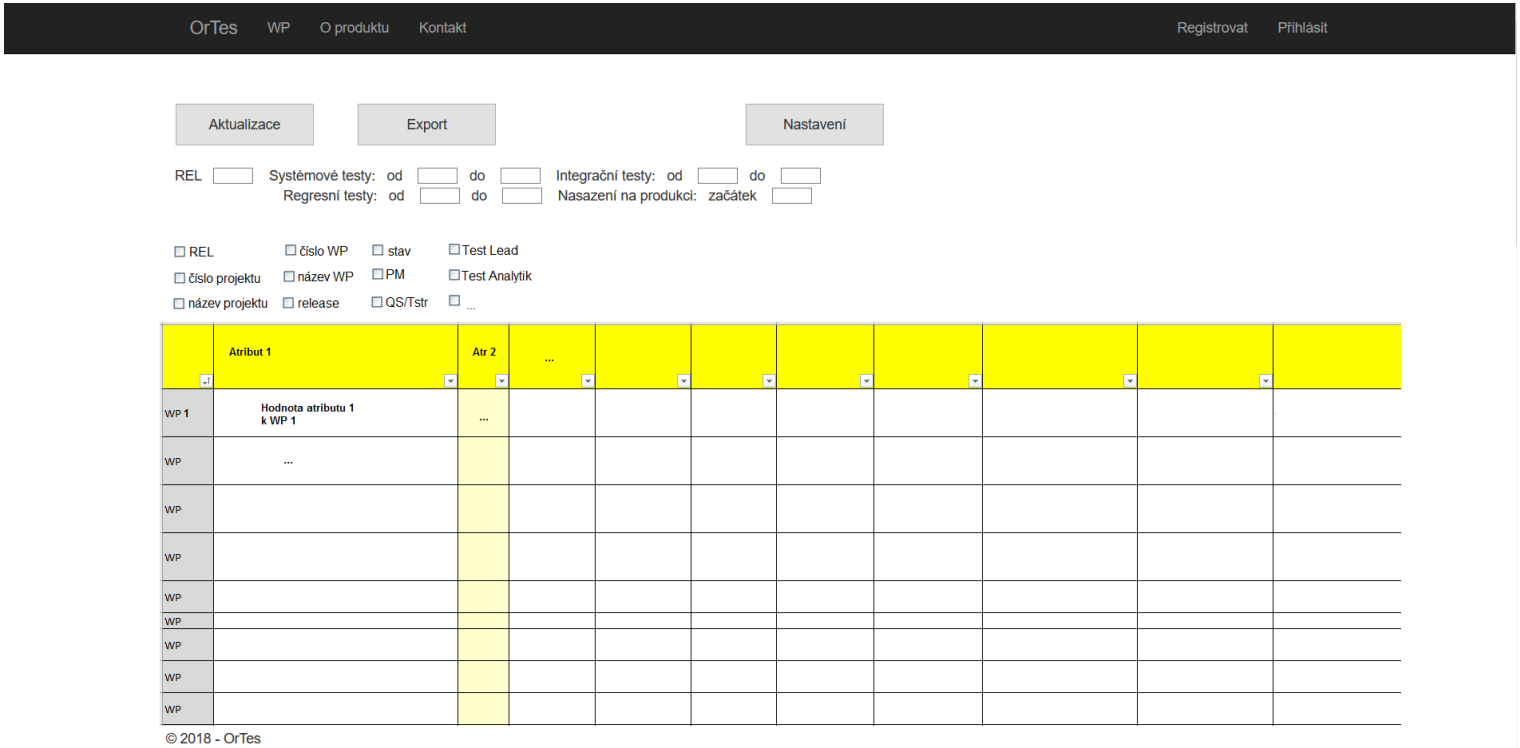
Když jsem již měla základní představu o funkcionalitě, začala jsem vybírat technologie a architekturu. Z praktických důvodů jsem se držela u známých architektur a rozšířených technologií. K nim lze totiž nalézt dostatek informací i při samostudiu.

4.2.1 Technologie

Již od počátku bylo vcelku jasné, že se bude jednat o webovou aplikaci. Mezi hlavní výhody patří snadná dostupnost a absence instalace pro uživatele. Navíc je aplikace umístěna na doméně .cetin, díky čemuž jsou data v ní vcelku dobře chráněna. Vybírala jsem tedy z technologií, které umožňují tvorbu webových stránek. Mezi nejpoužívanější patří Enterprise Java, C#, a PHP.

Nakonec jsem si zvolila C#. Z C# jsem znala jen základy, ale byly dostatečné k samostudiu. Na C# se mi nejvíc líbilo asi to, že je přehledný a celistvý. Na rozdíl od Enterprise Javy, která je sice rozšířena, ale stává se z ní monstrum, které kombinuje mnoho stylů zápisu a poměrně dost věcí se děje na základě kódu, který je kdesi vygenerován, a zasahovat do tohoto kódu je pro začátečníka velmi nebezpečné. S PHP nemám žádné zkušenosti a nechtěla jsem riskovat nějaké překvapení, které by mi znemožnilo něco vyvinout.

Neméně důležitým důvodem, proč jsem zvolila C#, je to, že jej vytvořil Microsoft. Pro interní správu oprávnění se totiž používá Windows autorizace, viz kapitola 5.7.2: Autorizace. Databáze, ve které budou data, je Microsoft SQL server. A aplikace bude umožňovat i export do XLSX formátu, který je taktéž pod Microsoftem, viz kapitola 5.9: Export do XLSX formátu. Je tedy pravděpodobné, že jednotlivé části budou spolu dobře komunikovat a jejich propojení bude podporované.



Obrázek 4.1: První návrh obrazovky

4.2.2 Architektura

Výběr architektury se zdál z počátku snadný, jednoduchá webová aplikace, jejíž hlavním úkolem je zobrazovat data z databáze. Dalším požadavkem byla snadná rozšiřitelnost a přehlednost. Architektura MVC se jevila jako ideální. S touto architekturou jsem také začala vyvíjet.

V průběhu práce jsem ovšem narazila na problém. Klasické MVC umí na jedné stránce zobrazit jen jednu tabulku, a já potřebovala na jedné stránce data z více tabulek. Po prostudování různých možností viz kapitola 2.3: Architektura jsem zvolila MVVM a ViewBag. MVVM jsem použila ve většině případů, hlavně pro informační obrazovky. ViewBag jsem použila pro zobrazení názvů sloupců v Editačních obrazovkách. Zde jsem totiž použila MVC architekturu, a názvy sloupců zde měly pouze popisný charakter a nepotřebovala jsem s nimi zde nějak výrazně manipulovat.

Při programování souběžnosti, viz kapitola 5.10: Kontrola souběžnosti, jsem byla nucena vyměnit ViewBag za ViewData. I když se zdají tyto struktury zaměnitelné, v tomto případě to neplatilo. ViewBag je totiž dynamický typ a při opětovném načtení stránky je již prázdný. Viz kapitola 2.3.3: ViewBag a 2.3.4: ViewData.

4.3 Změny návrhu v průběhu vývoje

Přestože jsem znala základní funkcionalitu, vyskytly se drobnosti, které jsem byla nucena měnit při samotném vývoji. Některé vznikly nedorozuměním, jiné si vyžádalo bezpečnostní oddělení firmy. Značnou výhodu jsem měla v tom, že jsem většinou pracovala v kanceláři, takže jsem se kdykoliv mohla zeptat vedoucího na doplňující otázky.

4.3.1 Role a oprávnění

Ze začátku jsem počítala s tím, že přihlášení uživatelů a správu rolí si budu řešit sama v aplikaci. Bohužel se toto řešení příliš nezamlouvalo bezpečnostnímu oddělení firmy. Při první informaci, že tuto funkci budu muset nechat na interní aplikaci, jsem trochu znejistila, vůbec jsem nevěděla, jak tento systém funguje. Tuto informaci jsem se navíc dozvěděla až ve chvíli, když jsem začala s firmou domlouvat server a databázi, tudíž jsem již nějakou představu o struktuře aplikace měla. Naštěstí stačilo do aplikace přidat Windows autentizaci a potřebné jmenné prostory. V konečném důsledku to znamenalo zjednodušení.

4.3.2 Data z více tabulek na jedné obrazovce

V úplných počátcích jsem si aplikaci představovala mnohem více rozkouskovanou, v podstatě co tabulka nebo funkce to jedna obrazovka, hlavně co se části nastavení týče. Vedoucí měl ovšem představu spíše o podobě, že související informace budou co nejvíc u sebe a že akce s tabulkou pro aktivní WP budou pro jistotu alespoň na dvě kliknutí. Toho jsem dosáhla tím, že editace, archivace a smazání záznamu z tabulky je až na obrazovce editace jednoho WP. Toto mě také donutilo využít architekturu MVVM, viz kapitola 5.2: Architektura MVVM v šabloně pro MVC.

4.3.3 Kromě hlaviček ukotvit i levou stranu tabulky

Požadavek na ukotvení hlavičky tabulky vznikl poměrně brzy, a i když není explicitně součástí zadání, přišlo mi to jako užitečný prvek. Poté, co jsem předvedla první funkční tabulku s ukotvenou hlavičkou, bylo jasné, že tlačítko pro editaci na začátku tabulky, které se schová při posunutí tabulky vpravo, není zrovna praktické. Obzvlášť když tabulka naplněná daty může mít klidně i dvojnásobnou šířku, než zabírá viditelná část.

Rezervní řešení bylo, že tlačítko pro editaci bude i na konci tabulky, naštěstí se mi povedlo najít funkční řešení, které navíc nebude kolidovat s jinými funkcemi tabulky. Podrobněji toto řešení rozepíši v kapitole 5.4.2: Ukotvení hlavičky a levých sloupců. To ovšem nebyl ještě konec, po dalším předvedení vznikla myšlenka, že by nebylo špatné, aby kromě samotné editace zůstaly na místě i další dva sloupce, které obsahují identifikátor a název WP. Tím by bylo dosaženo mnohem větší přehlednosti. Naštěstí se mi tento požadavek povedlo uskutečnit bez opětovné změny technologie.

4.4 Grafická podoba

Veškerá grafika je přizpůsobena funkčnosti aplikace a neruší ani při delší práci. Využila jsem hlavní navigační lištu pro základní pohyb v aplikaci. Tabulky s aktivními a archivovanými WP jsem odlišila pomocí barevného pozadí v hlavičkách názvů sloupců. Navíc stránka s aktivními WP obsahuje kalendář. Všechny ostatní stránky jsou dostatečně odlišitelné sami o sobě hlavně díky zcela odlišné funkci. Podoba obrazovky s aktivními WP byla dána hlavně funkčností tabulky s WP. Ukázku konečné podoby hlavní obrazovky naleznete zde 4.2: Hlavní tabulka s WP.

Velkou pozornost jsem věnovala vzhledu obrazovky pro editaci aktivních WP. Jedná se totiž o rozdílný postup při změně záznamu v tabulce oproti Excelu. Zásadní byla snadná orientace a uzpůsobení velikosti polí předpokládanému obsahu. Jelikož je možné, že v budoucnu se budou rozměry polí měnit, a navíc nezávisle na sobě, určila jsem jejich velikost přímo v HTML kódu. Pro menší riziko chyb jsem navíc barevně odlišila tlačítka pro uložení, archivaci a smazání dat tohoto WP. Náhled obrazovky naleznete zde 4.3: Obrazovka pro editaci jednoho WP.

Stejně tak jsem pro lepší orientaci dala přehled oprávnění pro různé role vedle sebe. Jsou tak nejlépe vidět rozdíly. Oprávnění fungují na principu, že uživatel má dané oprávnění, pokud alespoň jedna jeho role má toto oprávnění. Obrazovku se můžete prohlédnout zde 4.4: Obrazovka Nastavení.

Jelikož je aplikace primárně určena pro oddělení testingu, kde jsou všichni zvyklí na základní grafické rozhraní aplikací plné tabulek, tak nikomu nevadí strohost grafiky. Naopak by přílišná zdobnost byla nežádoucí. Proto mé rozhraní neobsahuje žádné zbytečné grafické prvky.

4.5 Přidání nových sloupců

Z důvodu snadnějšího pozdějšího rozšíření počtu sloupců bez zásahu programátora jsem vytvořila více sloupců, než je potřeba. Dokud nebudou potřeba, budou schované. Jejich viditelnost je jednoduše nastavitelná z uživatelského rozhraní, stejně jako viditelnost sloupců pro různé role uživatele. Stejně tak jsou z uživatelského rozhraní nastavitelné i názvy sloupců.

Tyto požadavky byly vzneseny hlavě kvůli životnosti aplikace. V oddělení testingu se C# nepoužívá, a pokud jej někdo zná, tak velmi povrchně. Tudíž najít někoho, kdo by přidal sloupec, až nebudu ve firmě přítomna, bude obtížné. Tedy po potřebě přidat další sloupec by byla aplikace nepoužitelná. Díky předchozímu používání Excel tabulky, je počet sloupců vcelku stabilní. Po konzultaci s vedoucím práce jsme počítali tak s 5 rezervními sloupci. Přesto jsem tento odhad ještě zvětšila, a to na 9 rezervních sloupců.

Toto nejjednodušší možné řešení problému jsem zvolila hlavně kvůli výše zmíněnému problému s nedostatkem náhradních programátorů a značnému zjednodušení kódu oproti variantě dynamického přidání sloupce, které by si vyžádalo složitější modelovou vrstvu a následnou práci s více tabulkami najednou.

Manta WP Archiv Nastavení Login

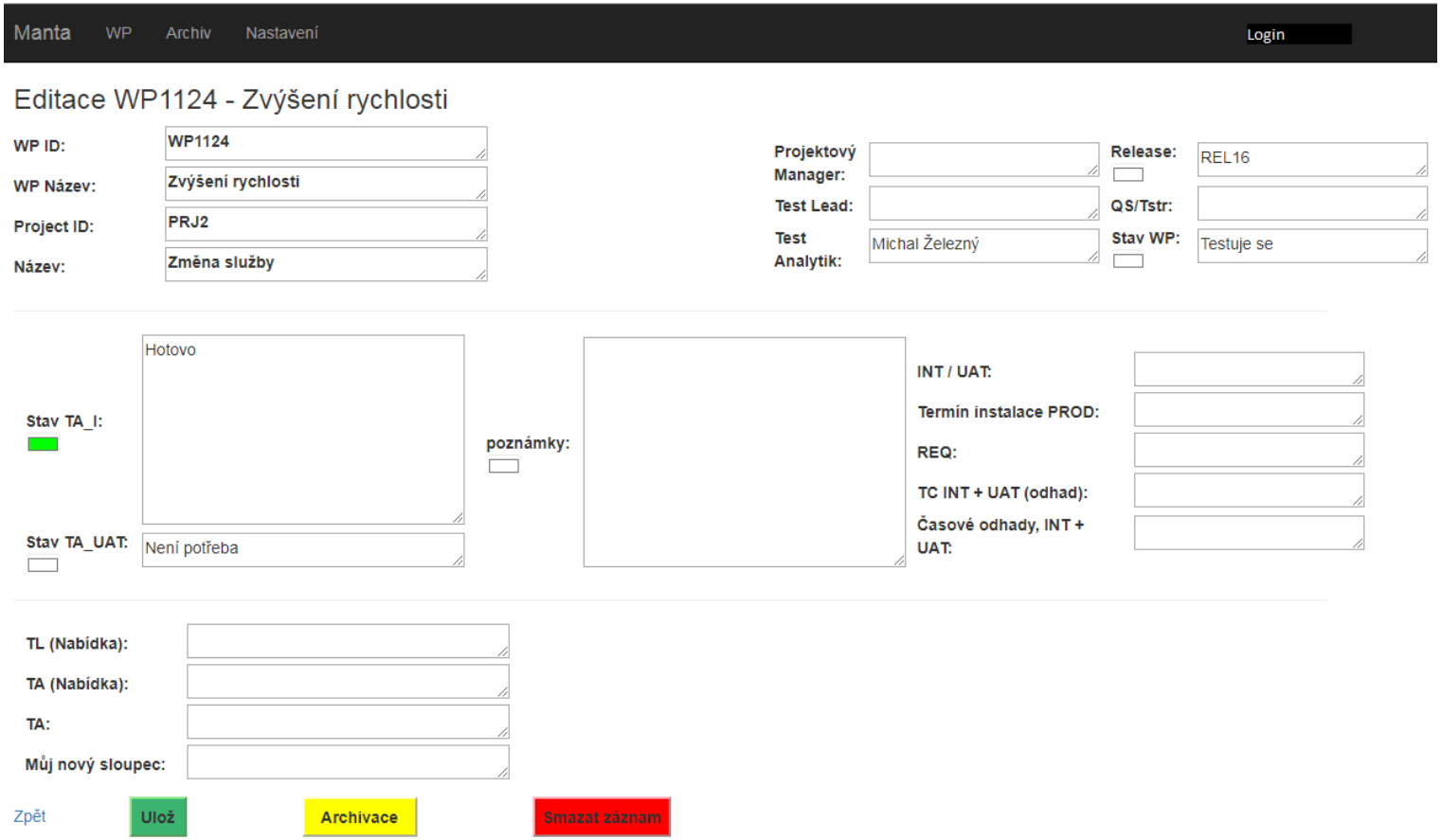
REL 16

Systémové testy: 4. 12. 2018 – 7. 12. 2018
Integrační testy: 10. 12. 2018 – 30. 1. 2019
Regresní testy: 1. 2. 2019 – 8. 2. 2019
Nasazení na produkci: 12. 2. 2019

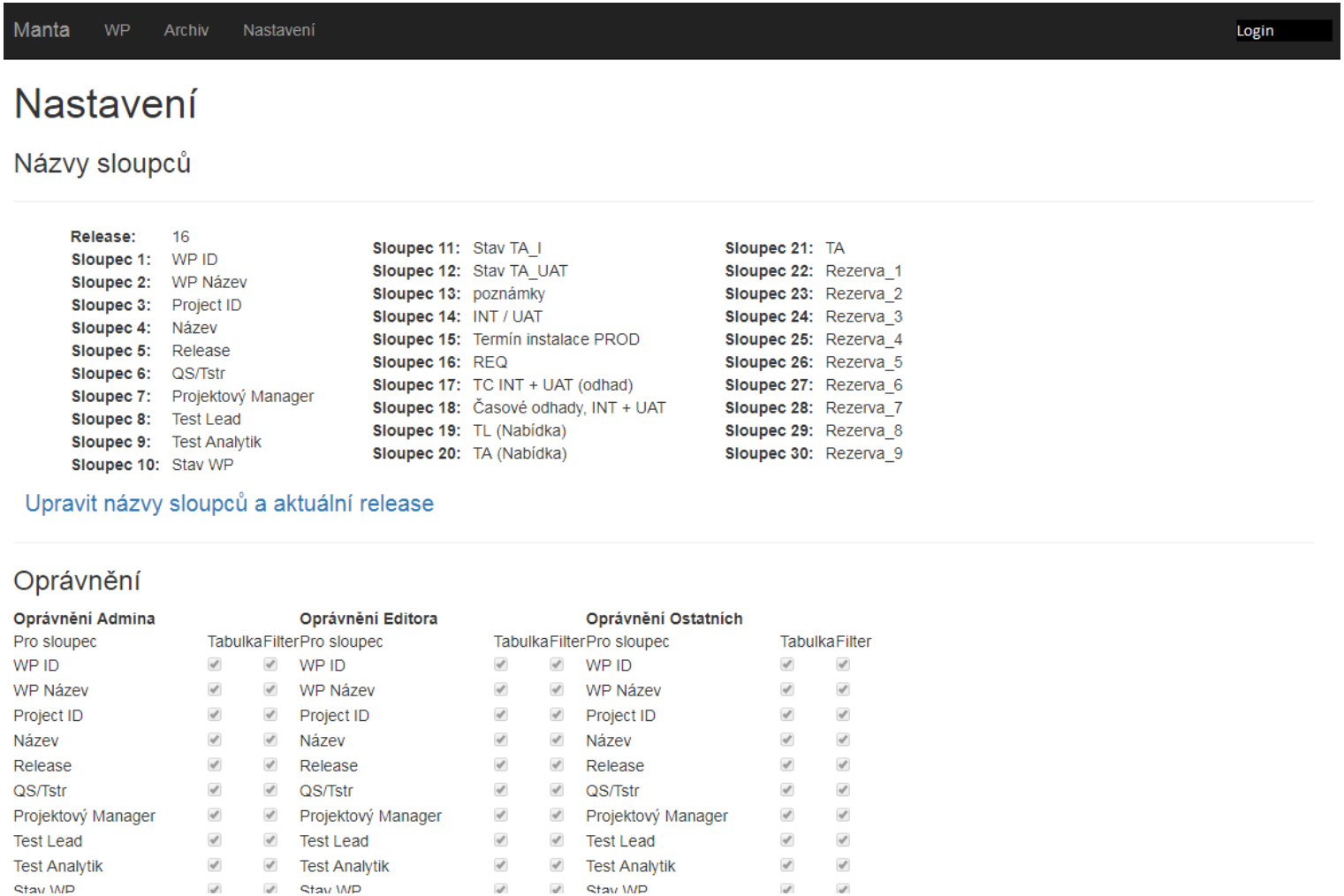
Export

Akce	WP ID	WP Název	est ead	Test Analytik	Stav WP	Stav TA_I	Stav TA_UAT	poznámky
		tlačítka	ářová	Pokorný	testů			
Edit	W11200	Přidání atributu do tabulky	ef bý	Michal Železný		Na revizi	Na revizi	
Edit	W11300	Vytvoření nové stránky pro plugin		Jan Pokorný		Hotovo	Domlouvá se	
Edit	W11400	Změna názvu tlačítka		Michal Železný		Na revizi		
Edit	W11500	Zvýšení rychlosti	ka ářová		příprava testů	Pracuje se		
Edit	W11600	Vytvoření šablony pro email zákazníkům		Jan Pokorný		Hotovo	Domlouvá se	
Edit	W11700	Archivace dokumentů		Michal Železný	Domluva zadání	Pracuje se		
Edit	W11800	Sjednocení výstupů	ka ářová	Michal Železný	Storno	Není potřeba		
Edit	W11900	Změna sazby	ef bý		Testuje se	Na revizi	Na revizi	
Edit	WP1120	Přidání atributu do tabulky		Jan Pokorný		Doplnění zadání	Hotovo	

Obrázek 4.2: Hlavní tabulka s WP



Obrázek 4.3: Obrazovka pro editaci jednoho WP



Obrázek 4.4: Obrazovka Nastavení

Kapitola 5

Realizace

Fáze realizace se mi značně mísila s fází návrhu. Jelikož se pro mě jednalo o první větší aplikaci, navíc v technologiích, které jsem do této práce příliš neznala, potřebovala jsem si hodně funkcionalit zkusit, jestli vůbec budou možné. Základní představu jsem brala hlavně z uživatelské znalosti jiných aplikací. A samozřejmě jsem měla stále na mysli požadavek, aby aplikaci v případě nouze dokázal upravit i programátor s minimem zkušeností.

5.1 Vývojové prostředí

Výběr vývojového prostředí bylo asi nejjednodušší rozhodnutí v celé bakalářské práci. C# vytvořil Microsoft a má pro něj i vlastní vývojové prostředí a to Visual Studio. Verzi jsem vybrala 2017, byla to nejnovější stabilní verze v době, kdy jsem začala s vývojem.

5.2 Architektura MVVM v šabloně pro MVC

Visual Studio nemá možnost vygenerovat MVVM architekturu. Umí však vytvořit MVC šablonu. Jak upravit MVC šablonu, aby z ní vzniklo MVVM jsem našla zde [19]. Hlavní myšlenka je, vytvořit jedno řešení se třemi projekty. Hlavní projekt je vygenerovaný MVC projekt, z něho použijeme view a controller. Další projekt nám nahradí view-model a poslední datovou vrstvu. Nesmíme zapomenout je správně propojit, tedy do view-model vrstvy přidat odkaz do datové vrstvy a do hlavního projektu odkazy na view-model i datovou vrstvu. Aby toto všechno ve Visual Studiu fungovalo je potřeba přidat Entity Framework.

5.3 Databáze

Na provozu bude databáze SQL serveru od Microsoftu umístěna mimo server, na kterém poběží aplikace. Pro vývoj mám zprovozněnou lokální databázi taktéž od Microsoftu. V databázi mám uložené jednoduché tabulky bez relací.

5.4 Vlastnosti hlavní tabulky

Hlavní tabulka toho musí umět najednou celkem mnoho. Má některé sloupce a hlavičku ukotvené viz kapitola 5.4.2: Ukotvení hlavičky a levých sloupců. V budoucnu bude umožňovat filtraci nad každým sloupcem pro více hodnot, a v neposlední řadě se bude umět i seřadit viz kapitola 5.4.1: Řazení podle sloupce. Pro každou s těchto funkcionalit existuje několik řešení, ovšem problém nastává právě v jejich kombinaci. Proto jsem pro každou funkci použila jiný způsob. Pro řazení jsem využila JavaScript, pro ukotvení hlavičky a levých sloupců jsem využila CSS a pro filtraci použiji v budoucnu select2. Díky tomuto technologickému rozdělení mám jistotu, že se nebudou navzájem rušit.

5.4.1 Řazení podle sloupce

Řazení tabulky podle libovolného sloupce je jedna s funkcionalit, která mi trvala trochu delší dobu než jsem čekala. A i když není součástí cílů této práce, je to v konečné aplikaci velmi důležitá funkce, tudíž jsem chtěla mít jistotu, že lze naprogramovat v mé aplikaci. Pro tuto funkcionalitu jsem se nakonec rozhodla použít existující řešení [20].

5.4.2 Ukotvení hlavičky a levých sloupců

Ukotvení hlaviček jsem řešila z počátku pomocí JavaScriptu, toto řešení mi však nešlo zprovoznit současně se skriptem pro řazení. Hledala jsem tedy jinou cestu a narazila jsem na tuto stránku [21]. A přestože jsem se snažila toto řešení přizpůsobit mým potřebám, nepodařilo se mi to. Hlavní problém byl to, že při posunutí vnitřních dat ukotvené sloupečky přetékaly. Takže jsem měla například tabulku a nad ní vysunutý sloupeček s odkazy na editaci. Nicméně mě toto řešení navedlo na správnou cestu, a to využití CSS stylů.

Z CSS stylů jsem využila vlastností position a z-index. `Position: sticky` upravuje své chování podle toho, zda je zrovna viditelný jeho rodič. Pokud není rodič vidět, chová se jako `Position: relative`, a pokud je rodič zobrazen, chová se jako `Position: fixed`. Spolu s `Position: sticky` musí být použit alespoň jeden z příkazů top, nebo left, které určují posunutí prvku o zadaný počet pixelů. Jejich použití mi nedělalo problém, jelikož šířka prvních dvou sloupců je vždy stejná. Více se o vlastnosti position dozvíte zde [22].

Dále bylo potřeba určit, které buňky se budou jak překrývat. K tomu slouží vlastnost z-index. Čím větší hodnota, tím ve vyšší vrstvě bude prvek zobrazen. Pokud je hodnota nastavena na 0, chová se standardně tedy, že se prvky zobrazují v pořadí, v jakém jsou napsané v kódu. Více se o z-index dozvíte zde [23].

Nakonec přidávám ukázku mé implementace této funkcionality 5.1: Ukotvení hlavičky a levých sloupců. Odstranila jsem veškeré grafické formátování. WP_th je označení pro hlavičky sloupců a WP_body_th označuje levé sloupce. Za povšimnutí stojí příkaz `:nth-child(n)`, který umožní nastavit vlastnost pro n-tého potomka daného prvku.

```
.WP_div { overflow: scroll; position: relative;}
.WP_table { position: relative; }
.WP_th { position: sticky; top: 0;}
    .WP_th:first-child { left: 1px; z-index: 1; }
    .WP_th:nth-child(2) { left: 31px; z-index: 2; }
    .WP_th:nth-child(3) { left: 73px; z-index: 2; }
.WP_body_th { position: sticky; left: 0; }
    .WP_body_th:nth-child(2) { position: sticky;
                        left: 31px; z-index: 1; }
    .WP_body_th:nth-child(3) { position: sticky;
                        left: 73px; z-index: 1; }
```

Výpis kódu 5.1: Ukotvení hlavičky a levých sloupců

Jedinou nevýhodou bylo, že se při pohybu tabulky odsunulo i ohraničení ukotvených sloupců. Měli tloušťku sice jenom jeden pixel, přesto to bylo celkem vidět zvláště, když prosvítalo barevné pozadí buňky. Po pokusech nastavit ohraničení z-index, jsem nakonec tento problém obešla pomocí obrysu, takzvaného outline, který se používá obdobně jako border.

5.5 Barvy buněk

I když obarvení buněk není součástí zadání, v původní Excel tabulce se pro některé sloupce používá, hlavně pro zpřehlednění celkového stavu. Poté, co jsem si zvykla na způsob práce s daty v ASP.NET, jsem nakonec přišla na velmi snadné řešení tohoto problému. Barvu jsem si uložila do databáze jako řetězec v podobě hexadecimálního kódu, a poté jsem tuto hodnotu přiřadila vlastnosti background-color, jak můžete vidět na ukázce 5.2: Určení barvy buňky.

```
<td class="WP_td" style="background-color:
@Convert.ToString(item.Color_1)">
      <span style="white-space:pre-line">
            @Convert.ToString(item.Column_5)
      </span>
</td>
```

Výpis kódu 5.2: Určení barvy buňky

5.6 Editace a Archivace

Jedná se u události vyvolané kliknutím na tlačítko. To znamená, že musím ve view vytvořit tlačítka, která mě přesměrují na správný controller. Ten pak příslušně upraví modely. Tlačítko pro editaci je tak časté, že ho máme předpřipraveno pod typem submit. Pro tlačítko Archivace si správné určení controlleru musíme zajistit sami pomocí onclick. Příklad naleznete zde 5.3: Tlačítka ve view.

```
<p> <table> <tr>
      <td style="width:150px">
         <button type="submit" name="save" value="Save">
         <B>Uloz</B></button>
      </td>
      <td style="width:200px">
         <button onclick="location.href='@Url.Action
               ("Add","Archiv",new { id=Model.Id })';
               return false;><B>Archivace</B></button>
                  </td>
</tr> </table> </p>
```

Výpis kódu 5.3: Tlačítka ve view

Když máme hotové view, je načase vytvořit controllery. Conroller pro editaci WP zcela standardní, příjme upravená data a pokud jsou validní uloží je databáze. Příklad si můžete prohlédnout zde 5.4: Controller pro edit. Jak vytvořit tlačítko v MVC aplikaci se podrobněji dozvíte zde [24].

```
[HttpPost]
public ActionResult WP_Edit(Aktive_WP awp){
        if (ModelState.IsValid)
                {db.Entry(awp).State = EntityState.Modified;
                    db.SaveChanges();
                    return RedirectToAction("WP");}
            } return View(awp); }
```

Výpis kódu 5.4: Controller pro edit

Controller pro Archivaci je trochu nestandardní. První odlišnost je, že se nachází v controlleru pro archiv, a ne pro WP, jelikož mění tabulku archivu. Také musíme vytáhnout jednotlivá WP z tabulky aktivních WP a umístit je do nového objektu archivu. Kód naleznete zde 5.5: Controller pro Archiv.

```
    [HttpGet]
        public ActionResult Add(int Id)
        {    var archivedWP = db.Aktive_WP.Find(Id);
             var archiv = new Archiv();
             archiv.Id_Aktive_WP = archivedWP.Id;
             archiv.Color_1 = archivedWP.Color_1;
             ...
             archiv.Column_30 = archivedWP.Column_30;
             db.Archiv.Add(archiv);
             db.SaveChanges();
             return RedirectToAction("WP", "WP");   }
```

Výpis kódu 5.5: Controller pro Archiv

5.7 Oprávnění a role

Díky požadavkům firmy, viz kapitola 4.3.1: Role a oprávnění, stačilo systém napojit na standardní ověřování Windows, který lze nechat vygenerovat spolu se základní strukturou MVC aplikace a funguje i po úpravách na MVVM. Celé oddělení testingu využívá na firemních počítačích operační systém Windows, tudíž není potřeba řešit případy uživatelů bez tohoto systému. Navíc v případě nouze půjde využít export do Excel tabulky. Do budoucna totiž plánuji kromě jiných rozšíření i možnost importu z Excel tabulky.

5.7.1 Windows autentizace

Visual Studio umí vygenerovat vše potřebné pro Windows autentizaci již při zakládání projektu. Windows autentizace je v aplikaci povolena pomocí kódu v souboru Web.config viz kód 5.6: Povolení Windows autentizace. Více se dozvíte zde [25]. Pokud chci zobrazit nebo zjistit identitu uživatele, stačí se ve View dotázat pomocí `User.Identity.Name`. Tento příkaz vrátí uživatele jako string.

```
<configuration>
    <system.web>
        <authentication mode="Windows" />
    </system.web>
</configuration>
```

Výpis kódu 5.6: Povolení Windows autentizace

5.7.2 Autorizace

Když mám funkční autentizaci, můžu začít s autorizací. Přiřazování a odebírání oprávnění je řízeno firemním systémem, tudíž se touto funkcionalitou nebudu zabývat.

Z bezpečnostních důvodů chci zamezit přístupu neautentizovaným uživatelům. Tohoto nejsnáze dosáhnu opět změnou konfigurace. V souboru Web.config přidáme kód 5.7: Zákaz přístupu do aplikace anonymním uživatelům. Zde si můžu v případě potřeby nastavit i jiná pravidla. Deny definuje uživatele, kteří nemají do mé aplikace přístup. Pokud bych chtěla určit uživatele, kteří budou mít přístup, použila bych slovo allow. Případně lze oba způsoby nakombinovat. Pokud bych měla v úmyslu umožnit přístupy pro všechny uživatele s určitou rolí, změním users na roles. Nyní se soustředím na vkládaný string. Mohu buď zapsat všechny možné hodnoty oddělené čárkou, nebo použít zástupné symboly. Pro anonymní uživatele to je "?", a pro všechny uživatele "*". Více se dočtete zde [26].

```
<configuration>
     <system.web>
        <authorization>
            <deny users="?" />
        <\authorization>
    </system.web>
</configuration>
```

Výpis kódu 5.7: Zákaz přístupu do aplikace anonymním uživatelům

K úpravě přístupových práv k jednotlivým akcím, mohu využít omezení u conterolleru viz kód 5.8: Určení přístupových práv na úrovni controlleru. V první fázi budou oprávnění řešena pouze na view vrstvě. Konečná plně zabezpečená verze bude implementována až se v provozu ujasní, která role potřebuje jaká práva.

```
[Authorize(Roles = "Admin")]
    public ActionResult Edit()
    {
    }
```

Výpis kódu 5.8: Určení přístupových práv na úrovni controlleru

Pokud chci upravit zobrazované informace na stránce, mohu použít příkaz if s dotazem `User.IsInRole("Admin")`. Tímto způsobem řídím zobrazované sloupce v tabulce. Abych mohla tento příkaz použít, musím do projektu přidat odkaz na jmenný prostor `System.DirectoryServices.AccountManagement`, a obor názvů `System.Security.Principal` do controlleru.

5.8 Nastavení oprávnění z rozhraní

Požadavek na rozšiřitelnost z uživatelského rozhraní byl nejnáročnější z pohledu počtu řádků kódu. Informaci o nastavení viditelnosti jednotlivých sloupců v závislost na roli jsem si uložila do databáze. Tuto informaci jsem propagovala až do view, kde jsem pomocí podmínek pro každý sloupec určila, zda se má zobrazit, či ne viz kód 5.9: Nastavení přístupových práv pro jeden sloupec.

```
@if (((Model.DataCollection_Settings_Group
      .Where(r => r.User_Group == "All")
        .FirstOrDefault().Column_5_Show) == true) ||
      (User.IsInRole(Admin)&&
         (Model.DataCollection_Settings_Group
           .Where(r => r.User_Group == "Admin")
             .FirstOrDefault().Column_5_Show) == true) ||
      (User.IsInRole(Editor) &&
         (Model.DataCollection_Settings_Group
           .Where(r => r.User_Group == "Editor")
             .FirstOrDefault().Column_5_Show) == true)
   ){
 <th class="WP_th" id="6_header">
       @Model.DataCollection_Settings_Name_Column
          .First().Name_Column_5</th>
} </tr> ... </thead> ... <tbody>
@foreach (var item in Model.DataCollection_Aktive_WP)
```

```
{ ... <tr>
 @if (((Model.DataCollection_Settings_Group
     .Where(r => r.User_Group == "All")
       .FirstOrDefault().Column_3_Show) == true) ||
     (User.IsInRole(Admin) &&
        (Model.DataCollection_Settings_Group
          .Where(r => r.User_Group == "Admin")
            .FirstOrDefault().Column_3_Show) == true) ||
     (User.IsInRole(Editor) &&
        (Model.DataCollection_Settings_Group
          .Where(r => r.User_Group == "Editor")
               .FirstOrDefault().Column_3_Show) == true) )
 {    <td class="WP_td">
          <span style="white-space: pre-line">
                @Convert.ToString(item.Column_3)
          </span>
      </td>       } ... }
```

Výpis kódu 5.9: Nastavení přístupových práv pro jeden sloupec

5.9 Export do XLSX formátu

Na internetu existuje mnoho řešení pro export do XLSX formátu, pro mé účely jsou však příliš obsáhlé. Já totiž potřebuji pouze základní export tabulky bez grafů, obrázků a dalších možností, které Excel nabízí.

Navíc jsem musela zajistit, že na serveru nebude samotný Excel, jelikož by bylo nutné platit a spravovat licence. Tyto požadavky splňuje balíček OpenXml přímo od Microsoftu. Potřebné teoretické informace jsem nalezla hlavně zde [27] a zde je praktické využití včetně videa [28].

U samotného dokumentu je potřeba si ohlídat správně vytvořenou základní strukturu. Tu si vytvoříme hned na začátku. Za povšimnutí stojí hlavně SpreadsheetDocument, který vytvoří samotný dokument, pak Workbook, do kterého se vloží jednotlivé listy. Následuje samotné vložení dat, data se vkládají po řádcích, do kterých jsou vloženy jednotlivé buňky. V mém případě jsem nejdříve vložila hlavičky, a poté pomocí cyklu i všechna data. Když jsou data bezpečně uložena, stačí doplnit pár posledních údajů, jako jméno výsledného souboru. Na konci je zajímavá ještě hodnota `Response.ContentType`, která označuje XLSX dokument. Jako poslední se odešle výsledný dokument uživateli, kterému se standardně uloží. Zkrácenou ukázku nejdete zde 5.10: Vytvoření XLSX souboru. Tento kód je umístěn v controlleru a je volán přes tlačítko Export.

```
[HttpGet]
        public ActionResult Export()
        {
        MemoryStream ms = new MemoryStream();
        SpreadsheetDocument xl = SpreadsheetDocument
        .Create(ms, SpreadsheetDocumentType.Workbook);
        WorkbookPart wbp = xl.AddWorkbookPart();
        WorksheetPart wsp = wbp.AddNewPart<WorksheetPart>();
        Workbook wb = new Workbook();
        FileVersion fv = new FileVersion();
        fv.ApplicationName = "Microsoft Office Excel";
        Worksheet ws = new Worksheet();

        UInt32Value i = 1;
        SheetData sd = new SheetData();
        Row r_head = new Row { RowIndex = i };
        i++;

        Cell cellH1 = new Cell();
        cellH1.DataType = CellValues.String;
        cellH1.CellValue = new CellValue
        (db.Settings_Name_Columns.Where(w => w.User ==
         "All").FirstOrDefault().Name_Column_1);
            r_head.Append(cellH1);
         ....
         sd.Append(r_head);
         foreach (var item in db.Aktive_WP)
         {
         Row r = new Row { RowIndex = i };
         i++;

         Cell cell1 = new Cell();
         cell1.DataType = CellValues.String;
         cell1.CellValue = new CellValue(item.Column_1);
         r.Append(cell1);
        ...
    sd.Append(r);
    }
    ws.Append(sd);
    wsp.Worksheet = ws;
    wsp.Worksheet.Save();
    Sheets sheets = new Sheets();
```

```
Sheet sheet = new Sheet();
sheet.Name = "TA aktivní";
sheet.SheetId = 1;
sheet.Id = wbp.GetIdOfPart(wsp);
sheets.Append(sheet);
wb.Append(fv);
wb.Append(sheets);

xl.WorkbookPart.Workbook = wb;
xl.WorkbookPart.Workbook.Save();
xl.Close();
string fileName = "MANTA-" + DateTime.Now
.ToString("yyyy_MM_dd HH.mm.ss") + ".xlsx";
Response.Clear();
byte[] dt = ms.ToArray();

Response.ContentType = "application/vnd
.openxmlformats-officedocument.spreadsheetml.sheet";
Response.AddHeader("Content-Disposition",
 string.Format("attachment; filename={0}", fileName));
Response.BinaryWrite(dt);
Response.End();
return RedirectToAction("WP");
}}}
```

Výpis kódu 5.10: Vytvoření XLSX souboru

5.10 Kontrola souběžnosti

Pro svou aplikaci jsem si zvolila optimistickou souběžnost, jelikož je méně náročná na výkon a uživatele omezuje až když je to potřeba. Celkem povedený popis naleznete zde [29]. Pro implementaci souběžnosti je potřeba přidat do modelu sloupeček typu timestamp, tuto hodnotu si aplikace spravuje sama, a to tak, že pokaždé, když se pracuje s daným sloupcem, zvětší svou hodnotu o jedničku.

Díky této vlastnosti metoda `SaveChange()` dokáže vyhodit výjimku v případě, že byla data konkrétního řádku během editace změněna. Tuto výjimku DbUpdateConcurrencyException lze pak následně odchytit. Je vhodné zde zkontrolovat, zda ke kolizi na úrovní jednotlivých buněk opravdu došlo. A případně uživateli vypsat, co se změnilo, a to jen tam kde je to potřebné. Příklad naleznete zde 5.11: Kontrola souběžnosti v controlleru. Nesmíme zapomenout ve view doplnit, že hodnota timestepu se nemá měnit. Toho dosáhneme přidáním `@Html.HiddenFor(model => model.RowVersion)`.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult WP_Edit([Bind(Include = "Id,
RowVersion, Color_1, ..., Column_30 ")]Aktive_WP awp)
{
  try
  {
    if (ModelState.IsValid)
    {
      db.Entry(awp).State = EntityState.Modified;
      db.SaveChanges();
      return RedirectToAction("WP");
    }
  }
  catch (DbUpdateConcurrencyException ex)
  {
    var entry = ex.Entries.Single();
    Aktive_WP clientValues = (Aktive_WP)entry.Entity;
    Aktive_WP databaseValues = (Aktive_WP)entry
    .GetDatabaseValues().ToObject();

    if (databaseValues.Column_1 != clientValues.Column_1)
        ModelState.AddModelError("Column_1",
         "Soucasná hodnota: "
            + databaseValues.Column_1);
    ...
    if (databaseValues.Column_30 != clientValues.Column_30)
        ModelState.AddModelError("Column_30",
         "Soucasná hodnota: "
            + databaseValues.Column_30);

    ModelState.AddModelError(string.Empty, "Tento záznam ...");
    awp.RowVersion = databaseValues.RowVersion;
  }
  return View(awp);
}
```

Výpis kódu 5.11: Kontrola souběžnosti v controlleru

Kapitola 6

Testování

Testování je nedílnou součástí vývoje softwaru jakéhokoliv druhu. Ověřuje nám totiž funkčnost našeho produktu. Existuje mnoho stylů testování a každý má svůj význam, navíc se v praxi často mísí, případně jsou přizpůsobeny konkrétním požadavkům výsledného produktu nebo testovacímu týmu.

Přestože začínajícím programátorům přijde nějaký složitější systém testování zbytečný, brzy zjistí, že opak je pravdou. Testy nám totiž šetří čas. Čím dříve se totiž chyba objeví, tím lehčeji ji lze odstranit. Ať už z důvodu menšího úseku, ve kterém se musí najít, nebo díky tomu, že si programátor ještě podrobně pamatuje, co a jak psal.

Testy jsou také používané jako ověření splnění zadání, popřípadě smlouvy. Jsou totiž snadno měřitelné a ověřitelné. Pokud jsou definovány již na začátku vývoje, mohou sloužit i k domluvě postupů práce v aplikaci i podobě jednotlivých komponent. Nesmí to však přerůst do situace, kdy stojí celý vývoj jen na těchto testech a neuvažuje se nad dalšími eventualitami.

Abych se mohla věnovat průběhu mého testování, je potřeba si nejprve popsat různé pohledy na testování. Různé pohledy vlastně dávají odpověď na otázky ohledně průběhu testování. Tyto otázky je vhodné zodpovědět před plánováním samotného testování. Neméně důležitá je i otázka, za jakým účelem testuji aplikaci. Tato otázka nám sice nepomůže určit, jaké testy použít, určí nám však rozsah testů.

Základní otázky před zahájením testování

- Kdo bude testy provádět?
- Jak moc bude znát tester podrobnosti implementace?
- Co se bude testovat?

6.1 Způsob provádění testů

Kdo bude provádět testy je jedna ze základních otázek. Testy se mohou provádět ručně nebo je může provádět nějaký program. Volba závisí na velikosti aplikace, časových možností a znalosti výsledné aplikace. Často se tyto postupy kombinují. Automatické testy se pak většinou využívají pro základní funkčnost a již schválené funkční celky. Tím se jednoduše zkontroluje, zda se něco nerozbilo. Manuální testy pak slouží pro kontrolu vyvíjených částí.

6.1.1 Automatizované testy

Automatizace testování je čím dál tím víc žádané. Šetří totiž čas a prostředky celého týmu. Hodí se hlavně pro stabilní aplikace, nebo její části. Jejich vytvoření totiž zabere nějaký čas, který se vrátí jen v případě, že lze automatizované testy používat delší dobu. Často se také tyto testy spouští přes noc. Mezi největší nevýhody patří to, že se s nimi musí počítat již během samotného vývoje. A nutnost test přepsat v okamžiku, kdy se aplikace na dotčených úsecích změní. Mezi nejčastější jazyky pro vytvoření automatizovaného testu patří Java a Python. Více se o automatizovaných testech dozvíte zde [30].

6.1.2 Manuální testy

Manuální testy se využívají u menších aplikací s krátkou životností, či častými změnami. Užitečné jsou také v počátcích vývoje, kdy je většinou provádí přímo tester. Hlavní výhodou je, že testy lze provést okamžitě. Nevýhodou je časová náročnost a možnost chyby. Mezi nejčastější chyby patří špatné provedení testu nebo chybné zkontrolování výsledků.

6.2 Míra znalostí testera o aplikaci

Mezi důležitá rozhodnutí k testování patří i to, kolik informací testerovi zpřístupníme. V praxi tato rozhodnutí často závisí i na politice dané firmy, jak moc si své kódy chrání. Obecně platí, že čím blíž je tester k zákazníkovi, tím míně ví o kódu samotném. A i když je na první pohled žádané, aby tester věděl o kódu co nejvíce, je užitečný i pohled testera, který vidí funkčnost jen z venku.

6.2.1 White box

White box je testování, kdy tester zná kód programu, případně má vedle sebe vývojáře s programem. Toto testování se využívá hlavně při kontrole zabezpečení dat. Také, pokud kód kontroluje někdo jiný než vývojář, slouží ke kontrole správnosti kódu. Toto testování dokáže najít mnoho problematických úseků, které jiné přístupy většinou nenaleznou. Je vhodné hlavně u aplikací,

kde je zabezpečení opravdu důležité, chrání totiž nejen před vnějšími útoky, ale i před více či méně úmyslnou chybou vývojáře. Nevýhodou je značná nákladnost tohoto testování. Je k němu totiž potřeba mnoho času někoho, kdo rozumí systémům a zároveň velmi dobře ovládá daný jazyk. Další informace naleznete zde [31].

6.2.2 Black box

Black box testování je pravým opakem white box testování. Tester neví nic o vnitřní implementaci programu. Hlavní výhodou tohoto testování je, že tester nepotřebuje žádné odborné znalosti. Musí jen vědět, jaký vstup mu má dát jaký výstup. Tato výhoda má však i svou stinnou stránku, tester nemá jak zjistit kvalitu kódu. Nejčastěji se používá pro akceptační testy. Více informací případně naleznete zde [32].

6.2.3 Grey box

Grey box kombinuje oba předchozí postupy. Tester zná algoritmy, případně architekturu aplikace, avšak nemá přístup k samotným kódům. Tester tedy potřebuje jen základní znalosti o programování a algoritmech. Tam, kde není kladen velký důraz na bezpečnost, je tento průběh testů vhodným kompromisem mezi nákladností a otestovaností. Používá se hlavně pro interní testy. Více se dozvíte zde [33].

6.3 Zaměření testů

A nakonec si musíme určit, jaký účel budou testy mít. Jinak se přistupuje k průběžným testům a jinak k závěrečným. Každá firma má svůj vlastní systém testování, některé testy spojují do jedné množiny, jiné někdy vynechají. Tyto skupiny nejsou v praxi vůbec striktně definované, jedná se spíše o zvykové pojmenování. Já se zde budu věnovat těm nejzákladnějším.

6.3.1 Základní testy

Do této kategorie patří testy, které dělá programátor během samotného vývoje. Odhalené chyby jsou nejsnáze opravitelné a nejméně nákladné. Je zde však potřeba určitá disciplína programátora. Mezi tyto testy řadíme například unit testy. Jelikož tyto testy dělá přímo programátor, patří tyto testy do kategorie whitebox testování.

6.3.2 Smoke testy

Tyto testy mají za úkol zkontrolovat, zda je aplikace alespoň v spustitelném stavu. Zkontroluje se, zda existují všechny kódy a lze je spustit. Poté se aplikace spustí a provedou se nejzákladnější funkce. Tento test se obvykle provádí před předáním programu testerům. U dlouhodobých projektů ve firmě, kde spravují mnoho aplikací, se většinou tyto testy automatizují a spouští pravidelně. Více se můžete dozvědět zde [34].

6.3.3 Integrační testy

Integrační testy většinou patří mezi první testy, které provádí tester. Slouží hlavně ke kontrole komunikace jednotlivých komponent. Kontroluje se zde i schopnost aplikace běžet na cílovém prostředí, tím se myslí hlavně cílový hardware, operační systém a komunikace s okolními systémy. Více se o integračních testech dovíte zde [35].

6.3.4 Systémové testy

Systémové testy patří k jedněm z nejdůležitějším testů pro dodavatele aplikace. Testuje se v nich samotná funkčnost produktu. Ve většině případů se také jedná o poslední fázi testování před předáním aplikace zákazníkovi. Po těchto testech by si měl být dodavatel jistý, že předává fungující a kvalitní aplikaci. A i když žádné testy nedokážou zaručit úplnou bezchybnost aplikace, měly by být dostatečně rozsáhlé, aby těch chyb bylo co nejméně. Určitě se vyplatí v této fázi projít i akceptační testy. Je potřeba také myslet na to, že případnou opravou jedné chyby může vzniknout chyba jiná. Proto by mělo platit, že poslední spuštění všech systémových testů proběhlo bez potřeby opravy aplikace. Další informace naleznete zde [35].

6.3.5 Akceptační testy

Akceptační testy probíhají u zákazníka. Zákazník si s jejich pomocí ověří funkčnost aplikace. Často jsou nutnou podmínkou pro převzetí aplikace. Bývají definované již ve smlouvě. V akceptačních testech by se nemělo přijít na závažnější chyby, a když už se objeví, měly by být co nejrychleji opraveny. Opakovaný neúspěch akceptačních testů může vést až k odmítnutí převzetí aplikace zákazníkem. Na akceptační testy by měla být aplikace připravena hlavně díky systémovým testům. Více se testech dozvíte zde. [35].

6.4 Průběh testování

Jelikož je má aplikace menšího rozsahu, bez složitějších postupů, rozhodla jsem se průběžné testování obstarat sama. Každou novou funkcionalitu jsem si ihned zkusila a jednou za týden jsem prošla celou aplikaci. Dále jsem zhruba jednou měsíčně, nebo podle potřeby, konzultovala stav se zadavatelem, který si také aplikaci odzkoušel. Samozřejmě nemohlo chybět závěrečné otestování. Pro větší objektivitu jsem nakonec poprosila o zhodnocení kolegu, který se vývoje neúčastnil.

6.4.1 Mé testy

Testy, které jsem si prováděla sama, bych zahrnula mezi manuální, white box, základní a systémové testy. Základní testy mi sloužili hlavně ke kontrole funkcionality, kterou jsem právě řešila. Systémovými testy jsem si průběžně kontrolovala celou aplikaci. I když bývá zvykem psát pro systémové testy scénáře, zde nebyly potřeba. Aplikace totiž neobsahuje složité postupy, aby bylo potřeba dokumentovat, jak co proběhlo. Každou činnost lze udělat pouze jedním způsobem. Navíc vzhledem k rozsahu aplikace nebyl problém projít veškerou funkcionalitu.

6.4.2 Uživatelské testy

Do této kapitoly bych zařadila průběžné testy zadavatele, poruším tím sice tvrzení, že akceptační testy se provádí až na konci, samotným účelem sem však spadají. Testy proběhly hlavně po dokončení větších úseků. Samozřejmě proběhli i závěrečné uživatelské testy, které dopadly dobře.

6.4.3 Testy neznalé osoby

Jako velkou nevýhodu jsem považovala, že testování se účasnily pouze osoby, které znaly celý průběh vývoje. Pokud totiž znáte původní podobu, víte jak se aplikace rozšiřovala, je velice těžké vidět některá nevhodná rozhodnutí. Proto jsem poprosila kolegu, který do této doby věděl jen to, že nějaká taková aplikace vzniká, aby aplikaci zhodnotil a otestoval.

Kolegovi jsem dala seznam základních operací, které měl provést a ohodnotit. Poté jsem pozorovala, jak si s úkoly poradil a zaznamenala jsem si případné komentáře. Také jsem ho poprosila o zhodnocení rozdílů mezi aplikací a dosavadní Excel tabulkou.

Tester měl za úkol:

- změnit aktivní WP,
- archivovat aktivní WP,
- smazat aktivní WP,
- exportovat aktivní WP,
- upravit zobrazení sloupců podle oprávnění,
- změnit release kalendáře,
- provést souběžnou editaci jednoho záznamu.

Všechny testy dopadly podle očekávání dobře. Kolegovi se nelíbilo, že archiv neobsahuje datum vložení. Tuto poznámku chápu, a já osobně bych tuto informaci také uvítala, avšak zadavatel byl jiného názoru. O tento časový údaj neměl od počátku návrhu archivu zájem. Druhý podle něj nevhodný postup je, že po kliknutí na tlačítko smazat se nezobrazí dotaz, zda si uživatel opravdu přeje záznam smazat. Tato hláška má určitě své opodstatnění k zamezení nechtěného smazání. Zajišťuje, že uživatel musí kliknout alespoň na dvě místa, aby záznam smazal. V aplikaci je tato dvojitá akce ošetřena díky nutnosti otevřít příslušné WP v editačním okně. Opět to tak zadavatel chtěl, hlavně kvůli úspoře času. Pro co nejmenší pravděpodobnost kliknutí na špatné tlačítko, jsem tato tlačítka odlišila barevně. Tím se dostávám k pozitivním reakcím. Velice se mu líbilo ukotvení levých sloupců a hlaviček. Prý značně zvyšují přehlednost celé tabulky. Líbilo se mu také funkční použití barev. V konečném zhodnocení přínosu aplikace zmínil hlavně možnost historizace jednotlivých WP pomocí archivu. Hlavní výhody, které měla aplikace přinést, tedy možnou souběžnou editace a zmenšení pravděpodobnosti zničení dat, podle mě neviděl hlavně z toho důvodu, že do jeho náplně práce nepatří úpravy stávající Excel tabulky.

Závěr

Cílem práce bylo vytvořit webovou aplikaci, která nahradí v potřebných funkcionalitách Excel tabulku tak, aby vyhovovala testovacímu oddělení firmy CETIN. Hlavním přínosem bude ochrana dat, ať z pohledu nechtěného smazání, či proházení atributů jednotlivých záznamů. K těmto nepříjemnostem v Excel tabulce běžně docházelo.

Aplikace umožňuje zobrazení tabulky s aktuálními záznamy, jejich editaci a mazání. Dále je přístupný archiv již přímo nepotřebných záznamů. Tyto tabulky jsou barevně rozlišeny. Navíc nad hlavní tabulkou je kalendář, který zobrazuje aktuální časový plán. Dále je možné exportovat tabulku do XLSX souboru. Oprávnění na přístup k jednotlivým sloupcům a filtrům je řízen z aplikace. Přístup k jednotlivým funkcím v závislosti na roli je řízen přímo z kódu. Zamezení ztrátě dat při souběžné editaci je zajištěno pomocí varovných upozornění při ukládání tzv. pesimistická souběžnost.

ASP.NET jsem nakonec použila ve zjednodušené verzi Razor. Aplikace bude v budoucnu napojena na firemní databáze, ze kterých bude čerpat data. Import jednotlivých nových záznamů bude řízen ručně, kalendář se bude aktualizovat automaticky. Dále bude k tabulkám přidán filtr pro každý sloupec, do kterého bude možné zadat více hodnot. Filtry budou skrývatelné na základě rolí podobně jako sloupce tabulky. Kvůli lepší uživatelské přívětivosti bude do editace záznamů přidán našeptávač možných hodnot. Tabulka archivu nebude zobrazovat všechny záznamy najednou, ale bude je stránkovat.

Jádro aplikace, které bylo cílem této práce, je tedy hotové a připravené na napojení k firemním systémům. Testování proběhlo bez problémů. V době odevzdání této práce nebyly nahlášeny žádné další požadavky.

Literatura

[1] Java EE. Oracle [online]. Praha: Oracle, 2019 [cit. 2019-05-07]. Dostupné z: https://www.oracle.com/cz/java/technologies/java-ee.html

[2] Úvod do jazyka C# a rozhraní .NET Framework. Microsoft [online]. Washington, USA: Microsoft, 2015 [cit. 2019-05-07]. Dostupné z: https://-docs.microsoft.com/cs-cz/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework

[3] PÍSEK, Slavoj. ASP.NET - začínáme programovat: podrobný průvodce začínajícího uživatele. Praha: Grada, 2003. ISBN 80-247-0526-5.

[4] HTML příručka. Jak psát web [online]. Praha: Dušan Janovský, 2017 [cit. 2019-05-01]. Dostupné z: https://www.jakpsatweb.cz/html

[5] STANÍČEK, Petr. CSS Kaskádové styly: Kompletní průvodce. Druhé vydání. Brno: Computer Press, 2003. ISBN 80-7226-872-4.

[6] Úvod k programování v rozhraní ASP.NET Web používající syntaxi Razor (C#). Microsoft [online]. Washington, USA: Microsoft, 2014 [cit. 2019-05-11]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/web-pages/overview/getting-started/introducing-razor-syntax-c

[7] Podřízne Razor PHP?. Zdroják [online]. Praha: Štěpán Bechynský, 2010 [cit. 2019-05-11]. Dostupné z: https://www.zdrojak.cz/clanky/podrizne-razor-php

[8] MVC architektura. Itnetwork [online]. Praha: David Čápka, 2016 [cit. 2019-05-08]. Dostupné z: https://www.itnetwork.cz/navrh/mvc-architektura-navrhovy-vzor

[9] Vzor Model-View-ViewModel. Microsoft [online]. Washington, USA: microsoft, 2017 [cit. 2019-05-08]. Dostupné z: https://docs.micro-

soft.com/cs-cz/xamarin/xamarin-forms/enterprise-application-patterns/mvvm

[10] Views in ASP.NET Core MVC. Microsoft [online]. Washington, USA: Microsoft, 2019 [cit. 2019-05-10]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/mvc/views/overview?view=aspnetcore-2.2

[11] DODGE, Mark a Craig STINSON. Mistrovství v Microsoft Excel 2010. Brno: Computer Press, 2011. Mistrovství. ISBN 978-80-251-3354-5.

[12] XLSX, (Office Open XML, Spreadsheet ML) ISO 29500:2008-2016, also ECMA-376, Editions 1-5. Digital Formats [online]. Digital Formats, 2017 [cit. 2019-05-11]. Dostupné z: https://www.loc.gov/preservation/digital/formats/fdd/fdd000398.shtml

[13] Funkce aplikace Excel (podle kategorie). Microsoft [online]. Washington, USA: Microsoft, 2019 [cit. 2019-04-09]. Dostupné z: https://support.office.com/cs-cz/article/funkce-aplikace-excel-podle-kategorie-5f91f4e9-7b42-46d2-9bd1-63f26a86c0eb

[14] Calc. Https://cs.libreoffice.org/ [online]. Berlín, Německo: The Document Foundation, 2019 [cit. 2019-04-09]. Dostupné z: https://cs.libreoffice.org/discover/calc

[15] Jira Software. Atlassian [online]. Sydney, Australia: Atlassian, 2019 [cit. 2019-05-07]. Dostupné z: https://cs.atlassian.com/software/jira

[16] Redmine. Redmine [online]. Jean-Philippe Lang, 2014 [cit. 2019-05-07]. Dostupné z: https://www.redmine.org/

[17] Easy Project 2019 Essentials. Easy Project [online]. Praha: Easy Project, 2019 [cit. 2019-05-10]. Dostupné z: https://www.easyproject.cz/software-pro-rizeni-projektu/essentials

[18] Peoplemanager. Peoplemanager [online]. Praha: peoplemanager, 2015 [cit. 2019-05-10]. Dostupné z: http://www.peoplemanager.cz/#atry

[19] Using MVVM in MVC Applications – Part 1 [online]. USA: Paul D. Sheriff, 2017 [cit. 2019-04-12]. Dostupné z: https://www.codeproject.com/Articles/1174826/Using-MVVM-in-MVC-Applications-Part-4

[20] SortTable. Jfisddle [online]. Paul S., 2013 [cit. 2019-05-12]. Dostupné z: http://jsfiddle.net/zscQy/

[21] Freezepanes. Disco-nova [online]. Markku Uttula, 2014 [cit. 2019-05-12]. Dostupné z: https://www.disconova.com/open_source/files/freezepanes.htm

[22] Position. Jak psát web [online]. Praha: Dušan Janovský, 2017 [cit. 2019-05-12]. Dostupné z: https://www.jakpsatweb.cz/css/position.html

[23] Jak psát web. Z-index [online]. Praha: Dušan Janovský, 2017 [cit. 2019-05-12]. Dostupné z: https://www.jakpsatweb.cz/css/z-index.html

[24] Create Edit View in ASP.NET MVC. TutorialsTeacher [online]. TutorialsTeacher, 2019 [cit. 2019-05-12]. Dostupné z: https://www.tutorialsteacher.com/mvc/create-edit-view-in-asp.net-mvc

[25] Ověřování uživatelů pomocí ověřování systému Windows (C#) [online]. Washington, USA: Microsoft, 2009 [cit. 2019-04-12]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/mvc/overview/older-versions-1/security/authenticating-users-with-windows-authentication-cs

[26] Jak implementovat ověřování systému Windows a autorizace v technologii ASP.NET [online]. Washington, USA: Microsoft, 2019 [cit. 2019-04-12]. Dostupné z: https://support.microsoft.com/cs-cz/help/323176/how-to-implement-windows-authentication-and-authorization-in-asp-net

[27] Create a spreadsheet document by providing a file name (Open XML SDK). Microsoft [online]. Washington, USA: Microsoft, 2017 [cit. 2019-05-12]. Dostupné z: https://docs.microsoft.com/en-us/office/open-xml/how-to-create-a-spreadsheet-document-by-providing-a-file-name

[28] OpenXml - ASP.Net - How to create simple Excel file and add some text to cells - Step by step [online]. blogspot, 2014 [cit. 2019-05-12]. Dostupné z: http://howtodomssqlcsharpexcelaccess.blogspot.com/2014/06/openxml-aspnet-how-to-create-simple.html

[29] Ošetření souběžnosti se sadou Entity Framework v aplikaci ASP.NET MVC (7 10). Microsoft [online]. Washington, USA: Microsoft, 2013 [cit. 2019-05-12]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/mvc/overview/older-versions/getting-started-with-ef-5-using-mvc-4/handling-concurrency-with-the-entity-framework-in-an-asp-net-mvc-application

[30] Tápe váš tým jak začít s automatizací testování?. Kitner [online]. Modřice, Česká republika: Ing. Radek Kitner, 2018 [cit. 2019-05-13]. Dostupné z: https://kitner.cz/testovani_softwaru/tape-vas-tym-jak-zacit-s-automatizaci-testovani

[31] Grey box test. Clever and smart [online]. Česká republika: Miroslav Čermák, 2008 [cit. 2019-05-13]. Dostupné z: https://www.cleverand-smart.cz/white-box-test

[32] Grey box test. Clever and smart [online]. Česká republika: Miroslav Čermák, 2008 [cit. 2019-05-13]. Dostupné z: https://www.cleverand-smart.cz/black-box-test

[33] Grey box test. Clever and smart [online]. Česká republika: Miroslav Čermák, 2008 [cit. 2019-05-13]. Dostupné z: https://www.cleverand-smart.cz/grey-box-test

[34] Smoke testy. Testování softwaru [online]. Tomáš Hlava, 2011 [cit. 2019-05-14]. Dostupné z: http://testovanisoftwaru.cz/tag/smoke-testy/

[35] Fáze a úrovně provádění testů. Testování softwaru [online]. Tomáš Hlava, 2011 [cit. 2019-05-14]. Dostupné z: http://testovanisoftwaru.cz/category/druhy-typy-a-kategorie-testu

Příloha **A**

Seznam použitých zkratek

ASP.NET Technologie vycházející .NET Frameworku. Jedná se o soubor funkcí a objektů, které vytvářejí webové stránky na serveru a uživateli zaslat již hotové HTML stránky

CETIN Česká telekomunikační infrastruktura s. r. o.

CLR je zkratkou Common Language Runtime. Je součástí .NET Frameworku. Stará o spuštění programů a přiřazuje jim zdroje

CSS Kaskádové styly, umožňují formátovat dokumenty pomocí hromadných pravidel

HTML Hypertext Markup Language. Jedná se o značkovací jazyk, který se používá pro tvorbu webových stránek

ISO Mezinárodní formát. Stanovuje standard pro danou věc

MSIL Microsoft Intermediate Language je mezijazyk pro .NET Framework

MVC Model-view-controller, architektura pro webové aplikace

MVVM Mode-view-viewModel, architektura pro webové aplikace

PHP Skriptovací jazyk pro webové stránky

PRJ Projekt, soubor WP s jedním účelem, například nová poskytovaná služba

REL Release, časový úsek, během něhož se pracuje na určité skupině WP

URL Uniform Resource Locator, doménová adresa serveru s potřebnými parametry

WP Work package, soubor změn potřebných pro jeden funkční celek, který lze nasadit do provozu, například nový formulář na webových stránkách

XLS Starší formát pro Excel tabulky

XLSX Novější formát pro Excel tabulky

ZIP Komprimovaná složka

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD a spuštění aplikace
- `Zadání`
- `Manta` ..... zdrojové kódy implementace
  - `Manta` ..... hlavní project
  - `Manta_dev_Data` ..... datová vrstva
  - `Manta_dev_ViewModel` ..... ViewModel vrstva
  - `Manta.sln`
- `Manta_bez_rolí` ..... Manta spustitelná na počítači mimo síť CETIN
  - `Manta` ..... hlavní project
  - `Manta_dev_Data` ..... datová vrstva
  - `Manta_dev_ViewModel` ..... ViewModel vrstva
  - `Manta.sln`
- `text` ..... text práce
  - `BP_Pohanová_Lenka_2019.pdf` ..... text práce ve formátu PDF
  - `BP_Pohanová_Lenka_2019.zip` ..... kódy textu BP