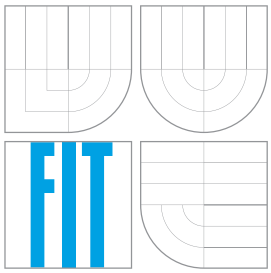
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

# MOBILNÍ PŘÍSTUP K DATŮM Z AUTOMATIZAČNÍCH PRVKŮ (VÝVOJ FRONT-END ČÁSTI APLIKACE)

MOBILE ACCESSING OF AUTOMATION DATA (FRONT-END DEVELOPMENT)

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN KUBIŠ
AUTHOR

VEDOUCÍ PRÁCE — Ing. ŠÁRKA KVĚTOŇOVÁ, Ph.D.
SUPERVISOR

BRNO 2016

*Zadání bakalářské práce/18798/2015/xkubis13*

**Vysoké učení technické v Brně - Fakulta informačních technologií**

Ústav informačních systémů Akademický rok 2015/2016

# Zadání bakalářské práce

Řešitel: **Kubiš Jan**
Obor: Informační technologie
Téma: **Mobilní přístup k datům z automatizačních prvků (vývoj Front-End části aplikace)**
**Mobile Accessing of Automation Data (Front-End Development)**
Kategorie: Softwarové inženýrství

Pokyny:

1. Seznamte se s technologiemi .NET MVC (Model View Controller) a SVG (Scalable Vector Graphics).
2. Prostudujte komponenty uživatelského rozhraní Telerik Kendo, navrhněte uživatelské rozhraní webové aplikace za použití těchto komponent a upravte (přepište) aplikaci podle daných kritérií.
3. Prostudujte SVG pro webové a mobilní aplikace a vytvořte uživatelsky přívětivé a atraktivní živé demo určené k prezentaci dat z kontrolního systému OPC.
4. Prostudujte a přizpůsobte SVG Editor tak, aby byl schopný vytvořit základní elementy kontrolního systému OPC a propojte je s OPC signály (přizpůsobte aktuální funkčnost s ohledem na specifické požadavky ABB).
5. Implementujte změny a optimalizujte, otestujte na vhodně zvoleném vzorku dat a navrhněte možná budoucí vylepšení.

Literatura:


- Telerik Kendo User Interface [online]. Available at: http://www.telerik.com/kendo-ui
- SVG Editor [online]. Available at: http://svg-edit.googlecode.com/svn-history/r1771/trunk/editor/svg-editor.html
- Freeman, A.: Pro ASP.NET MVC 5 Platform. Apress, 2014. ISBN 978-1-430-26542-9.
- Lange, J.: OPC (englischsprachige Ausgabe): From Data Access to Unified Architecture. Vde Verlag Gmbh, 2010. ISBN 978-3-800-73242-5.
- Rinaldi, J: OPC UA: The Basics: An OPC UA Overview For Those Who May Not Have a Degree in Embedded Programming. CreateSpace Independent Publishing Platform, 2013. ISBN 978-1-482-37588-6.


Pro udělení zápočtu za první semestr je požadováno:

- Body 1 a 2.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Květoňová Šárka, Ing., Ph.D.**, UIFS FIT VUT
Konzultant: Mináč Ján, Ing., ABB
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
*vedoucí ústavu*

# Abstrakt

Obsah této technické zprávy popisuje návrh, implementaci a funkcionalitu webové aplikace, jež byla vyvinuta v rámci bakalářské práce a jejíž účelem je sběr, přenos a prezentace dat ze zařízení, která jsou tato data schopna poskytovat dle specifikací standardu OPC (OLE for Process Control – Object Linking and Embedding for Process Control). Aplikace využívá architektury klient-server. Back-end aplikace je implementován v jazyce C# a front-end, na který se tato práce zaměřuje, využívá kombinaci programovacích, skriptovacích a značkovacích jazyků C#, JavaScript, HTML a SVG. Aplikace je vyvíjena pro společnost ABB (ta je vlastníkem všech zdojových kódů) a očekává se její nasazení v komerční sféře. Podporovanou platformou je operační systém Microsoft Windows 10.



# Abstract

Content of this report describes design, implementation and functionality of a web application, which was developed within bachelor thesis and which purpose is to collect, transfer and present data from devices that are able to provide the data according to OPC standard specification (OLE for Process Control – Object Linking and Embedding for Process Control). The application makes use of client-server architecture. Application back-end is implemented in C# and front-end, which is the subject of this report, uses combination of programming, scripting and markup languages C#, JavaScript, HTML and SVG. Application is being developed for ABB company (who is owner of all source codes) and it is expected to be put into operation in commercial sector. Supported platform is operating system Microsoft Windows 10.



# Klíčová slova

webová aplikace, automatizační data, OPC, ASP.NET, MVC, SVG

# Keywords

web application, automation data, OPC, ASP.NET, MVC, SVG



# Citace

KUBIŠ, Jan. *Mobilní přístup k datům z automatizačních prvků (vývoj Front-End části aplikace)*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Květoňová Šárka.

# Mobilní přístup k datům z automatizačních prvků (vývoj Front-End části aplikace)

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením paní Šárky Květoňové. Další informace mi poskytli Michael Filsák a Mário Čuboň. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Kubiš
24. května 2016

## Poděkování

Děkuji vedoucí práce paní Šárce Květoňové za zprostředkování a odborné vedení práce. Dále bych chtěl poděkovat Michaelovi Filsákovi a Máriovi Čuboňovi za čas věnovaný konzultacím a cenné rady vedoucí k úspěšnému dokončení práce.

# Obsah

# Kapitola 1

# Úvod

Klienty společnosti ABB jsou podniky z oblasti průmyslu, jež pro výrobu či jiné produktivní činnosti využívají ruzná zařízení – specializované stroje. Ty mohou mít sadu čidel a senzorů, kterými jsou schopny zachytávat momentální stav jistých veličin (počet kusů, úroveň elektrického napětí, zatížení, teplota aj.) v závislosti na účelu a typu zařízení. Výstupy ze senzorů lze sledovat v operačním středisku podniku například signalizací kontrolek nebo výpisem na jednoduchý LCD display a nastavovat jejich hodnotu tlačítky a spínači. Vše se tak děje na určitém místě pomocí specializovaných elektronických panelů a součástek. Nevýhodou je tím pádem omezený přístup k těmto datům.

Proto vznikla žádost na vytvoření aplikace, jež by všechny uvedené funkcionality zpřístupňovala odkudkoliv a běžela v prostředí běžného operačního systému. Kromě dostupnosti je tak výhodou snadno modifikovatelné rozhraní (úpravou kódu) a různá rozšíření, například pokročilá prezentace dat.

Má práce se zabývá front-end částí a spoléhá tak na existenci pozadí aplikace, na nějž se dá navázat. Implementace back-end části byla realizována kolegou Lukášem Habartou, studentem stejné fakulty, jehož část je předmětem jiné bakalářské práce. Přestože se má práce zaměřuje na front-end, kvůli celistvosti výkladu a jeho snažšímu pochopení jsou v některých pasážích letmo popsány i části back-endu.

Text práce je rozdělen do pěti kapitol. Aby se dalo lépe pochopit, k čemu vlastně vytvořená aplikace slouží, je nezbytné v přístí kapitole nejprve alespoň zběžně popsat standard OPC. Mimo to zde budou probrány technologie, jež aplikace využívá ke svému fungování. Třetí kapitola je věnována návrhu aplikace, konrétně její architektuře, účelu jednotlivých částí a jejich vzájemnému propojení. Čtvrtá kapitola se zabývá fází realizace vytvořeného návrhu a popisuje implementační detaily aplikace. V předposlední kapitole jsou zmíněna naplánovaná rozšíření. V závěru je shrnutí dosažených cílů a přínosů této práce.

# Kapitola 2

# Studované technologie

## 2.1 Standard OPC

Standard je oficiálně spravován konsorciem OPC Foundation. Konsorcium vytváří jeho formální specifikaci, své vlastní ukázkové programy a také vytváří knihovny, které lze zahrnout při vývoji vlastní aplikace a poté s jejich pomocí se standardizovaným objektem pracovat. Knihovny jsou implementovány v jazycích C, C++, Java, Visual Basic .NET a dostupná je také knihovna psaná v jazyce C#, jež bude v našem projektu použita. Standard nachází uplatnění především v průmyslových odvětvích, kde zajišťuje iteroperabilitu mezi elektronickými zařízeními od různých výrobců. Prakticky se jedná o sadu rozhraní, metod a objektů, kterými zařízení komunikuje se svým okolím bez ohledu na způsob vlastní vnitřní implementace[8]. Hodnoty a signály všech takových zařízení lze potom číst či do nich zapisovat jednotným způsobem.

Standard lze rozdělit do zhruba deseti skupin lišících se svým účelem. Naše aplikace operuje s následujícími skupinami:

- **OPC Data Access** (dále OPC DA) pro automatizované, kontinuální získávání hodnot signálů zařízení
- **OPC Alarms & Events** (dále OPC AE) sloužící k událostmi řízenému získávání dat
- v budoucnu se počítá s přidáním skupiny technologií **OPC Unified Architecture** (OPC UA), která zahrnuje jak OPC DA tak OPC AE a mnoho dalších funkcionalit ze skupin zbývajících

Celá technologie OPC využívá architektury klient-server a lze tak rozdělit na dvě části.

- **OPC Server** je implementován s ohledem na použité zařízení (komunikace s ním musí probíhat specifickým komunikačním protokolem – ovladač zařízení), proto je často dodáván výrobcem zařízení. Získává ze zařízení data, která zpřístupňuje ve standardizovaném formátu a stejně tak umožňuje standardizovaný způsob zápisu dat do zařízení. Na jeden server může být připojeno více klientů.
- **OPC Klient** je schopen se připojovat na server (může být připojen i na více serverů) a vykonávat nad ním zmíněné operace. Většinou bývá rozšířen o další funkcionality podle požadavků na jeho použití. Tato část bude implementována a rozšířena i v naší aplikaci, pomocí klienta budeme komunikovat s existujícími OPC servery.

Standard definuje rozhraní a způsob komunikace klienta a serveru. Jejich implementace je zcela na výrobci/vývojáři. Ten tím pádem určuje rychlost, spolehlivost a interoperabilitu konečného produktu.

## 2.2 HyperText Transfer Protocol

Technologie HTTP spolu se značkovacím jazykem HTML jsou základními stavebními prvky webové apikace. Protokol slouží (nejen) pro transport dokumentů psaných právě v jazyce HTML mezi zařízeními na síti. K tomu využívá TCP spojení, komunikuje na portu 80[4] (může i na jiném, nicméně pokud není v URL adrese uveden, implicitní hodnota je 80). Transport probíhá na architektuře klient-server. Klient posílá dotaz ve formě URL, čímž požaduje konkrétní dokument od serveru (původní myšlenka, dnes existuje více způsobů a forem). Server odpovídá zprávou v textové podobě obsahující výsledek hledání požadovaného dokumentu – stavový kód, po něm následuje hlavička dokumentu s dodatečnými informacemi, prázdný řádek a nakonec obsah požadovaného dokumentu. Při dotazu lze dodatečná data umístit buď přímo do URL (jistá omezení) nebo je umístit do těla dotazu. Některé dotazy jsou považovány za bezpečné (nemění data serveru), jiné zase za nebezpečné.

Proto vzniklo několik různých dotazovacích metod, jejichž specifikace určuje, k čemu by měly být využívány. Nejpoužívanějšími metodami jsou `GET`, `POST`, `PUT` a `DELETE`.

## 2.3 ASP.NET

ASP.NET je přímý nástupce webové technologie ASP (Active Server Pages). Obě technologie jsou vyvinuty společností Microsoft a slouží k vytváření dynamických webových stránek na straně serveru předtím, než jsou zaslány ke klientovi, umožňujíc tak vývoj komplexních webových aplikací a služeb. V čistém ASP se k tomuto účelu nejčastěji používají skriptovací jazyky VBScript a JScript. Ty lineárně projdou HTML kód klientem požadovaného dokumentu. Úseky se statickým obsahem jsou předány dál beze změny, úseky určené ke zpracování (vyznačené určitými symboly, mezi které se vepisuje požadovaná funkcionalita) jsou zpracovány odpovídajícím interpretem, jehož výstup může být vložen do výsledného HTML kódu nebo je dále zpracován. ASP má několik vestavěných tříd (`Request`, `Response`, `Server` aj.) s atributy a metodami, nelze však vytvářet žádné další. Existují jisté postupy, které to zdánlivě umožňují, ty jsou ale velmi neintuitivní a nepodobají objektově orientovanému návrhu, na který jsme zvyklí. Jedná se spíše o dočasné řešení.

Naproti tomu, ASP.NET pokrývá všechny nedostatky vyplývající z předchozího odstavce. Jelikož se jedná o součást platformy .NET, je možné k dynamickému vytváření stránky používat kód v jakémkoliv programovacím (nikoliv skriptovacím) jazyce převeditelném do CIL (Common Intermediate Language). Nejčastěji se můžeme setkat s použitím jazyků C# a Visual Basic.NET. Převod do CIL provádí kompilátor jazyka. Díky tomu je možné odhalit chyby v kódu už při kompilaci. Kompilace sice nějakou dobu trvá, s použitím výsledného kódu je ale stroj schopen danou stránku vytvářet rychleji než při použití skriptování. Další výhodou kompilace do CIL je možnost používat komponenty a knihovny napsané v různých jazycích.

## 2.4 Web Forms

Původně bylo ASP.NET synonymem s dnešním ASP.NET Web Forms, jehož cílem je vytvořit při vývoji webové aplikace abstrakci tak, aby se co nejvíce podobal vývoji desktopové aplikace (například skrytím detailů jazyka HTML a protokolu HTTP). Do HTML dokumentu lze psát speciální elementy (Rich server controls) podobné těm z kódu pro desktop (`Label`, `TextBox`), přiřazovat jim atributy a zachytávat na nich události. Místo přímého vykreslení prvku je generován odpovídající HTML a JavaScript kód lišící se v závislosti na použitém prohlížeči klienta. Některé z těchto prvků dokonce umožňují data binding (prvek `GridView`). Technologie se také snaží jistým způsobem odstranit protokolu HTTP „bezestavovost". Přidáním skrytého elementu je tak možné mezi po sobě jdoucími požadavky zjišťovat předchozí stav různých prvků (takovým elementům říkáme souhrně ViewState), což je jinak velmi pracné a vyžaduje spoustu řádků kódu navíc.

Framework má samozřejmě také nevýhody – na něm stavěné aplikace jsou velmi náročné na údržbu a robustní s ohledem na využití síťového přenosu. Obslužné funkce akcí HTML prvků nejsou testovatelné kvůli svým parametrům. To znemožňuje pro nás preferovaný test-driven development (testy řízený vývoj). I přes tyto nevýhody je Web Forms v současnosti stále široce využíván a u Microsoftu má podporu v podobě nových záplat a vylepšení.

## 2.5 Model-View-Controller

Model-View-Controller (dále MVC) je vzor popisující architekturu aplikace nebo její části. Není tedy specifický pro žádný programovací jazyk, jde spíše o způsob řešení problému v daném kontextu. Podstatou tohoto vzoru je rozdělení částí kódu do soběstačných celků podle jejich účelu a tím od sebe oddělit doménový (datový) model, uživatelské rozhraní a řídící logiku aplikace (části).

Těmito celky jsou komponenty tří typů: model, view (pohled) a controller (řadič).

### 2.5.1 Model

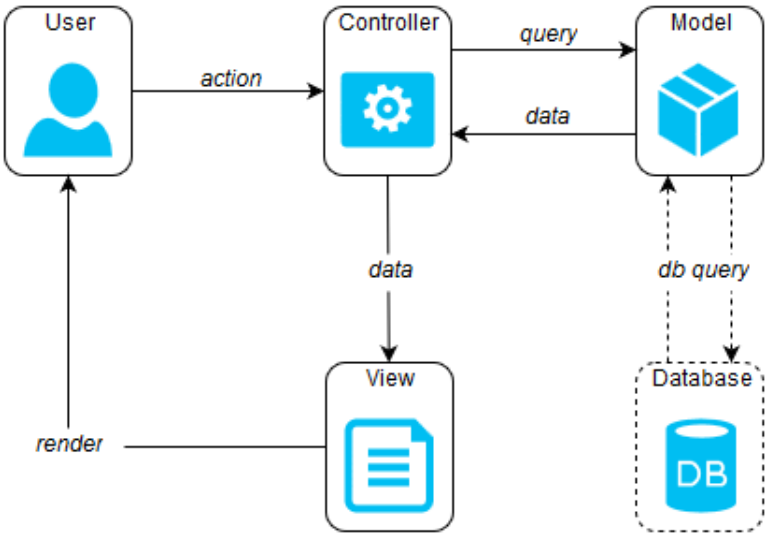
Model primárně představuje data, se kterými aplikace pracuje. Často je modelem myšlen doménový model. Ten zahrnuje také všechny asociované operace, transformace a pravidla pro manipulaci s těmito daty, tedy logiku. Můžeme si jej představit jako třídu, má atributy a metody. Model je plně izolovaný – nezajímá jej, odkud vstupní data přišla, ani kam budou výstupní data předána. Pokud aplikace využívá databázi, pracuje s ní právě v modelu.

### 2.5.2 Pohled

Kód definující uživatelské rozhraní. V rámci MVC většinou vykresluje určitá data modelu. Tak jako model, ani pohled se nestará, odkud data k zobrazení přišla. Jeho úkolem je pouze jejich vykreslení na úrčená místa podle šablony.

### 2.5.3 Řadič

Řadič zajišťuje řízení zmíněných komponent. Přijímá příchozí požadavky, provádí oprace nad modelem a vybírá pohled, který se má zobrazit. Pokud se v pohledu vykreslují dynamická data z modelu, měla by mu být předána právě řadičem. Je to tedy jakýsi prostředník mezi uživatelem, modelem a pohledem.




Obrázek 2.1: Architektonický vzor MVC

Takto rozdělený kód je nejen přehlednější, ale i vysoce flexibilní a tím pádem jednodušší na refaktorizaci, údržbu či modifikaci. MVC usnadňuje i vývoj – jak pro jednotlivce, tak pro tým, jelikož lze každou část navrhovat či implementovat nezávisle na jiné a vývoj tedy může probíhat paralelně.

## 2.6 ASP.NET MVC

Postupem času vzniknul jako alternativa k Web Forms (nikoliv jako jeho náhrada) framework ASP.NET MVC. Vývojáři bez zkušeností s vývojem webových aplikací bude práce s ASP.NET MVC nejspíš dělat mnohem větší problém než s Web Forms. Vůbec už nejde o paradigma událostmi řízeného programování. Vše je zde řízeno požadavky HTTP, což vyžaduje hlubší znalost tohoto protokolu. Není zde žádná snaha o odstranění bezestavovosti HTTP (neexistence ViewState), stejnětak není abstraktizován HTML kód. Ten si píše vývojář sám – serverové prvky (Rich server controls) nejsou k dispozici. To sice poskytuje větší kontrolu nad generovaným HTML kódem, nicméně opět vyžaduje jeho hlubší znalost.

Zkušenému vývojáři však zmíněné nevýhody práci nijak neztíží, naopak je mu nabídnuto více volnosti a především umožněno využít obrovských výhod architektonického vzoru MVC. Těmi jsou kromě těch zmíněných v části 2.5 znovupoužitelnost kódu, vysoká testovatelnost umožňující test-driven development nebo výběr nástroje určeného pro vykreslování pohledu – tzv. view engine. Ten určuje, jakým stylem a jakými direktivami budou označena místa v HTML dokumentu s dynamickým obsahem. Lze použít klasický view engine z Web Forms nebo spoustu jiných (je totiž možné vytvářet své vlastní), nejznámější z nich je asi Razor (běžně dokumenty s příponou .cshtml) vytvořený přímo pro ASP.NET MVC.

Vekou roli v aplikaci stavěné na ASP.NET MVC hraje routing (směrování). Ten zajišťuje, aby byl podle uživatelovy akce spouštěn adekvátní řadič. Na schématu 2.1 by byl umístěn ze směru od části User těsně před částí Controller (šipka *action*). Ve webové aplikaci tohoto typu uživatel odstartuje akci odesláním požadavku HTTP. Ten má různý tvar, užívá různé metody (GET, POST aj.). Routing jej převede na zvolený tvar (nastavení je zcela na vývojáři), který aplikace interně používá a podle kterého je schopna spustit správý řadič. U Web Forms se naproti tomu musel použít přímý odkaz na fyzický soubor na ser-

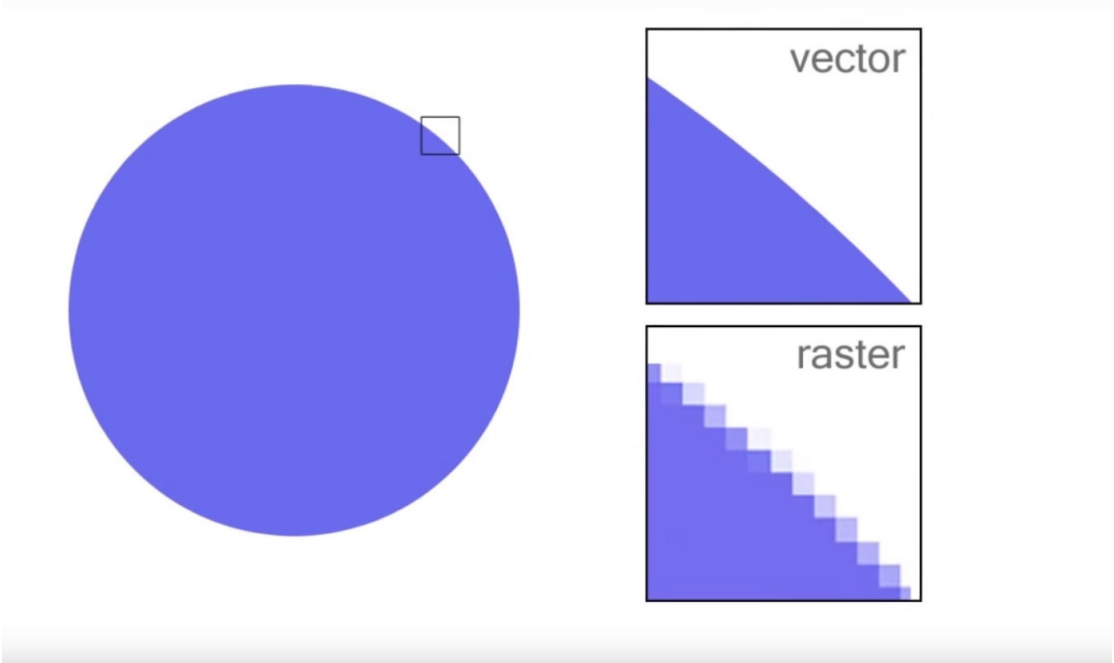
veru. URL je potom nejen čitelnější pro uživatele, ale může také optimalizovat nalezitelnost webu mezi vyhledávači.

## 2.7 Vektorová grafika

V oblasti výpočetní techniky existují dva základní způsoby ukládání a zpracování grafických prvků.

První způsobem je rastrová grafika. Grafický prvek je reprezentován obdélníkovou maticí bodů, které nazýváme pixely. Každý z pixelů má určité vlastnosti – polohu a barvu. Takto uložený prvek má jednu velkou nevýhodu – při zvětšování ztrácí na kvalitě, při extrémním zvětšení lze vidět samotnou mřížku (hranice jednotlivých pixelů). Tato reprezentace je jedinou možností automatizovaného zachycení reálného světa (běžné digitální fotoaparáty, skenery...).

Tuto nevýhodu řeší druhý způsob reprezentace zvaný vektorová grafika. Grafický prvek je popsán matematickým modelem – množinou bodů, přímek, křivek a jiných geometrickýh tvarů[3]. Intuitivně lze říci, že jde vlastně o návod, jak daný prvek nakreslit. Jednotlivé tvary jsou spíše ucelené objekty nežli shluk pixelů. Obrázek definovaný vektorově lze nejen přibližovat do nekonečna (stroj vždy aktuální vzhled přepočítá), ale přináší i jiné výhody. S každým objektem je možné pracovat odděleně, provádět na něm translace, rotace, změnu tvaru, barev, či změnu měřítka. Říkáme, že takový objekt je content-aware – je si vědom sám sebe a svého obsahu.




Obrázek 2.2: Porovnání ztráty kvality při přiblížení vektorového a rastrového obrázku

Paměťová náročnost takového obrázku je většinou menší než při rastrové reprezentaci. Představme si uložení triviálního obrázku s vybarveným kruhem:

- rastrová grafika – je nutné pro každý pixel uložit jeho souřadnice a barvu. Počet pixelů závisí na šířce a výšce obrázku.
- vektorová grafika – k plné rekonstrukci stačí 3 informace: souřadnice středu kruhu, jeho poloměr a barva výplně.

Vektorová grafika nachází uplatnění především v Computer-Aided Drafting (CAD) aplikacích, při tvorbě grafiky u které je potřeba vysokých rozlišení, v aplikacích využívající jazyk PostScript nebo v oblasti animací.

## 2.8 SVG

SVG je zkratkou pro Scallable Vector Graphics. Jde o značkovací jazyk využívaný v oblasti webů, určený k popisu dvoudimensionální vektorové grafiky. Stejně jako HTML, CSS, XML a spousta dalších, i tento standard je spravován konsorciem W3C. Dlouhodobým cílem je uplatnění tohoto formátu jako základním otevřeným formátem pro práci s vektorovou grafikou v oblasti internetu. Struktura jazyku je převzata z jazyka XML, elementy mají pouze specifické názvy a atributy. Syntaxe umožňuje různé způsoby zápisu, atributy mohou být odděleny čárkami nebo mezerami, některé mohou být dokonce uvedeny v rámci kaskádových stylů.

Kromě výhod plynoucích z využití vektorové grafiky jsou výhodami snadná přenositelnost díky nezávislosti na konkrétní platformě, čitelnost kódu pro člověka a jeho editace v jakémkoliv textovém editoru, strojová čitelnost textu z obrázku například pro účely vyhledávání nebo možnost využití nástrojů, jež byly vyvinuty pro práci s formátem XML.

## 2.9 JavaScript

Je multiplatformí, objektově orientovaný skriptovací jazyk. Syntaxí je velmi podobný jazykům C++ nebo Java, nicméně na rozdíl od uvedených jazyků je JavaScript beztypový. Kromě syntaxe s nimi nemá nic moc společného. Jazyk je základem pro dynamické webové stránky. Stejně jako je HTML dokument popisující stránku dynamicky upravován na straně serveru (popsáno v části 2.3), může být upravován i u klienta. Právě k dynamickým změnám na straně klienta jazyk slouží. Interpretaci JavaScriptu by měl zajišťovat webový prohlížeč klienta.

Na rozdíl od zpracování stránek na serveru, kde se využívá předpřipravených šablon a logicky umístěných direktiv pro vkládání obsahu, JavaScript se v celém dokumentu odkazuje na prvky pomocí DOM[9] (Document Object Model) ve spojení s jejich identifikátory a ostatními atributy.

V poslední době, s rozmachem aplikací s webovým rozhraním, se jazyku JavaScript dostává obrovské podpory v podobě různých knihoven a frameworků. Ten nejznámější je asi framework jQuery. Umožňuje například použití jednoduché syntaxe pro výběr DOM elementu a pro práci s ním, manipulaci s jeho CSS styly, nastavování obsluhy při události na elementu, AJAXová volání a spoustu dalších.

## 2.10 Bootstrap

Vzhledem k požadavkům na responsibilitu uživatelského rozhraní aplikace a jejích prvků bylo potřeba sáhnout po sadě kaskádových stylů, které responsibilitu zajišťují. Jako ideální se jeví framework Boostrap (vyvinut vývojáři Twitteru). Vývojáři ulehčí psaní rutinních záležitostí – zajišťuje přívětivou stylizaci, ať už po vizuální stránce nebo ve způsobu umístění a obtékání prvků uživatelského rozhraní, automatické výpočty relativní šířky/výšky prvku, rozložení HTML prvků vzhledem k těm ostatním a vůbec celému dokumentu pomocí mřižkového rozložožení – grid layoutu. Mimo stylizaci také pomocí vlastní knihovny implementované v JavaScript zajišťuje běžným prvkům rozšířenou funkcionalitu, která je na první pohled chtěná a intuitivní, ale prvky ji v HTML základu nepodoporují. Pro příklad elementu `<input>` sloužícímu ke znakovému zadávání uživatelského vstupu při vykreslení přidá prvek, který při kliknutí veškerý dosavadní vstup smaže. Přestože je Boostrap, alespoň z mého pohledu, moderní a uživatelsky přívětivý, v aplikaci jej nelze použít samotný.

Jako nedostačujiící se ukázalo řešení prvků pro zobrazení tabulárních dat, které jsou v aplikaci stěžejní a hojně využívané a kterým Bootstrap nezajišťuje responsibilitu podle mých představ. Základní stylizace – barvy a odsazení – potom musela být upravena vlastním stylem tak, aby odpovídala přesně stanoveným požadavkům z ABB Guideline.


Obrázek 2.3: Stylizace prvku `<input>` knihovnou Bootstrap

## 2.11 SignalR

Je žádoucí u klienta zobrazovat co nejaktuálnější data z automatizace. Jelikož na server může být v jednu chvíli připojeno velké množství klientů, pro které je nutné data zajistit, hrozí nebezpečí zahlcení serveru požadavky od klienta. Po diskusi s mentory se zdá jako nejvhodnější, vzhledem k implementačnímu jazyku serveru a jazyku klienta, využití technologie SignalR vyvinuté Microsoftem přímo pro ASP.NET. Základem a velkou výhodou této technologie je možnost získávání dat od serveru, aniž by klient musel posílat jakýkoliv požadavek. Tím pádem stačí, aby na server přišel jediný požadavek – v našem případě nejspíš zpráva z backendu, že byla aktualizována data – na jehož základě poté rozešlě upozornění nebo rovnou data (záleží na případu) vybraným klientům[1]. Ti, kteří mají na toto upozornění naimplementovanou reakci, data zpracují.

Knihovna jako taková musí být vložena a referencována jak na serveru, tak u klienta. V serverové části je kód psán v jazyce `C#`. Aby byla funkce na serveru schopna přijímat a odesílat data s využitím této knihovny, je nutné, aby rodičovský objekt, ve kterém se funkce nachází, dědil z knihovního objektu `Hub`. Lze si také zvolit, do které části serveru se mají takto vytvořené huby namapovat a pod jakou cestou budou zpřístupněny klientům. Tím se tedy dají serverem nabízené služby dobře rozdělit vzhledem k funkcionalitě, kterou nabízejí, či typům klientů, které budou obsluhovat.

V klientské části běží SignalR pod JavaScriptem. S HTML dokumentem, ve kterém budou využity jeho funkce, se tedy musí u klienta stáhnout a zainicalizovat skript poskytující podpůrné funkce (realizováno pomocí bundles – veškerý kód je obsažen v jednom skriptovacím souboru, pro všechny dokumenty stejný). Mimo zahrnutí skriptu je nutné v každém

HTML dokumentu zvlášť specifikovat cestu, pomocí níž se lokalizuje skupina hubů, se kterými bude chtít klient komunikovat (řídí se podle místa namapování na serveru).

S voláním serverové části z klienta byl z počátku problém, který se dlouho nedařilo vyřešit. Po několika pokusech a hledání možné příčiny vyšlo najevo, že řešení je jednoduché, ne však uplně intuitivní. SignalR totiž při provádění mapování převádí huby, které jsou implementované v C#, do odpovídajícího kódu v JavaScriptu, který potom na serveru běží. JavaScript používá rozdílnou konvenci pojmenování proměnných a funkcí, které na rozdíl od C# začínají malým písmenem. Z klientské části je tedy nutné je volat s ohledem na tuto konvenci.[5]

```
public class DaServerServiceHub : Hub{
  private readonly IDaServerService _daService = new DaServerService();data) {

    public IList<DaServer> GetAll(bool admin){
        return _daService.GetAll(new ApiInput<bool>(admin)).Result;
    }
}
```

Obrázek 2.4: Registrace objektu do seznamu hubů, jehož funkce mohou být poté volány z klientské části.

```
refillTable: function (dataTable) {
    $.connection.daServerServiceHub.server.getAll(false).done(function (data) {
        dataTable.clear();
        dataTable.rows.add(data);
        dataTable.draw();
    });
}
```

Obrázek 2.5: Volání funkce serveru musí dodržovat konvence jazyka JavaScript

# Kapitola 3

# Návrh aplikace

## 3.1 Existující software

Hledání aplikací se stejnou nebo alespoň podobnou funkcionalitou dopadlo neúspěchem. Zaměstnanci ABB nám bylo potvrzeno, že takový nástroj zatím neexistuje. O jeho vývoj se údajně pokouší finská divize zmíněné společnosti, zatím bezvýsledně. Ten začal již pred dvěma lety a momentální status je neznámý, dostupný není ani žádný prototyp. Jeden takový program sice vytvořen byl, jednalo se však pouze o pilotní projekt, takže se nikdy neplánovalo jeho nasazení. Nese název 800xA Mobile Client a byl vytvořen Ostravskou pobočkou společnosti ABB. Autorem je jeden z našich mentorů Michael Filsák. Podle jeho slov se postupovalo co nejrychleji kupředu a účelem vývoje bylo vytvořit vhodnou architekturu aplikace, vyzkoušet a vybrat nejvhodnější technologie, aby následně mohl být vytvořen kvalitnější návrh pro zcela novou aplikaci. A právě tato nová aplikace, respektive její část, je předmětem mé bakalářské práce.

### 3.1.1 AngularJS

Hlavním rozdílem ve frontendové části je úplné odstranění technologie AngularJS. AngularJS je specifický svou syntaxí. Ta vyžaduje, aby byl příslušný HTML dokument, který chystáme naplnit daty, rozšířen o jisté atributy, čímž vlastně vytvoříme jakýsi data-binding danému HTML elementu, kterému je takový atribut přidělen. Všechny tyto atributy začínají prefixem `ng-`, za nímž následuje název požadované funkcionality frameworku (aby byla zachována validita HTML dokumentu, lze použít také prefix `data-ng-`).

Například atribut `ng-app="MODULE"` (většinou přidělen pro celý HTML dokument přidáním direktivy k elementu `<html>`) určuje parametrem `MODULE` název hlavního modulu aplikace, který je implementován v JavaScriptu a který celou AngularJS aplikaci spustí. Jak už bylo zmíněno, v aplikaci tohoto typu jde primárně o zobrazení tabulárních dat, což se řeší direktivou `ng-repeat` u elementu který se má opakovat, v tabulce element `<tr>`.

Kromě atributů s prefixem `ng-` jsou také AngularJS zpracovávány výrazy uzavřené dvěma složenými závorkami `{{EXPRESSION}}`. `EXPRESSION` má JavaScript syntaxi a můžeme se v něm odkazovat na prvky pole, atributy objektu, objekty samotné, proměnné nebo třeba volat metody. Vše dobře demonstruje ukázka kódu 3.1.

Tento způsob zápisu je velmi podobný front-end kódu aplikace psané ve WPF (Windows Presentation Foundation), kde se mezi elementy dokumentu (pro WPF je to formát XAML) také vepisují direktivy pro data-binding. Technologie byla nahrazena kvůli rozhodnutí, že by celá aplikace neměla být vyloženě SPA (Single Page Application) a také hlavně kvůli

nekompatibilitě druhé verze AngularJS s první, což je u vývojářů velmi nepopulární až nepřípustné.

```
<tbody>
    <tr ng-repeat="fw in frameworks | filter:searchQuery ">
        <td>{{fw.id}}</td>
        <td><a ng-href="{{fw.link}}">{{fw.name}}</a></td>
        <td>{{fw.description}}</td>
        <td>{{fw.language}}</td>
    </tr>
</tbody>
```

Obrázek 3.1: Dynamické generování tabulárních dat pomocí AngularJS

Pilotní projekt měl mimo rozhraní vykreslované prohlížečem také svoji vlastní desktopovou aplikaci psanou ve Windows Presentation Form. S tím se v rámci našeho projektu do budoucna počítá také, nicméně zatím není jasné, jestli bude použita stejná technologie. Prioritou každopádně zůstává přístup přes internetový prohlížeč. Co se týče back-end části, byl využit stejný typ databáze (MSSQL), pro ulehčení přístupu a práce s ní se však používala technologie PetaPoco, která je nyní nahrazena jinou. Pro komunikaci s OPC serverem se používaly knihovny od společnosti Softing namísto současných knihoven od OPC Foundation.

## 3.2 Architektura aplikace

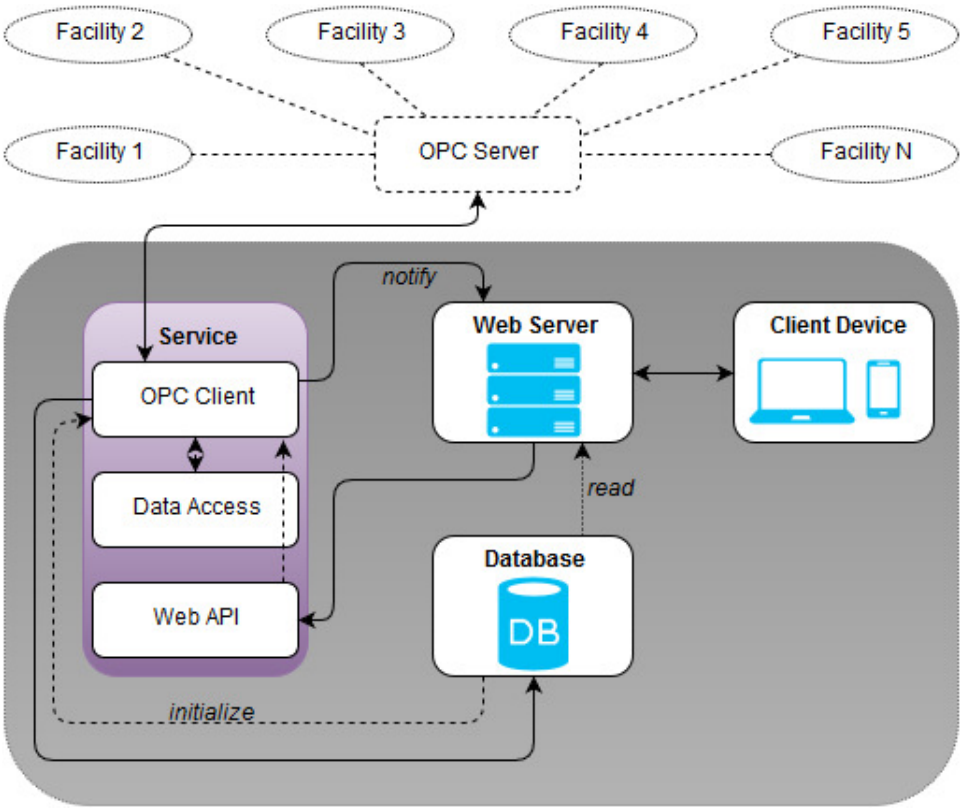
Všechny námi implementované části by se daly rozdělit na 3 nebo 4 fyzické oblasti (stroje):

- první oblastí jsou **klientská zařízení** (desktop, mobilní zařízení), kde bude zpřístupňováno uživatelské rozhrání aplikace (stránka ve webovém prohlížeči) a odkud bude celá ovládána
- další je stroj na kterém beží **webový server** obsluhující všechna připojená klientská zařízení na základě jejich požadavků (protokol HTTP respektive HTTPS)
- **databázový server**
- **OPC klient**

## 3.3 Webový server a klient

Webový server poběží na operačním systému Windows a bude hostován softwarovým serverem IIS (Internet Information Services). Ten je v základě všech novějších verzích systému Windows, defaultně se však nespouští, při vývoji a nasazení je tedy potřeba jej nastavit a jeho spouštění automatizovat. IIS je rozšiřitelný o jisté moduly. Kromě vyřizování požadavků různých protokolů (námi využívané HTTP a HTTPS, podporuje i FTP, FTPS, SMTP) tak například může velmi jednoduše zajistit proces autentifikace (4.3.1) uživatelů přistupujících na jeho služby. Podporuje všechny služby které se chystáme využít – ASP.NET, WebAPI a SignalR (protokol WebSocket).

Jako základ webové aplikace bude využita technologie ASP.NET MVC, jelikož splňuje požadavky pro přehlednost, snadnou udržovatelnost a modifikovatelnost, případně snadnou




Obrázek 3.2: Architektura aplikace

rozšiřitelnost zdrojových kódů. Mimo to je takový kód dobře testovatelný, což je pro náš vývoj nezbytné. Testy by měly být psány pro každou větší část aplikace. Při jistých modifikacích aplikace vývojář může velmi snadno opomenout fakt, že takovou modifikací si sice pomůže ke svému dílčímu cíli, nicméně v již implementovaných částech taková změna zapříčiní ztrátu správné funkcionality. Psaní testů je nejsnadnějším způsobem, jak takovému scénáři zabránit. Po dosažení požadované funkcionality v jedné části může být ověřeno zachování ostatních funkcionalit aplikace spuštěním sady testů.

Důvod fyzického oddělení webového serveru a stroje na kterém poběží OPC klient je zřejmé. Webový server je pouze jakýmsi prostředníkem, přes kterého lze přistupvat na data OPC kienta či do nich zapisovat. Jelikož se však jedná o dva různé koncepty, je velmi výhodné je implementovat tak, aby od sebe byly plně oddělitelné. Umístěním webového serveru na stejný stroj by mohla být negativně ovlivněna výkonnost OPC klienta, což je nežádoucí. V potaz je brána i možnost útoku na server, při stávající architektuře by se dotkla jen webového serveru.

## 3.4 OPC klient a databáze

OPC klientem se zde myslí část obrázku 3.2 pojmenovaná *Service*. Jde celkem o soubor tří komponent. První komponentou je samotný OPC klient obsahující všechny obslužné metody a třídy potřebné pro komunikaci s OPC serverem. Aby se dal OPC klient ovládat z webového severu přes síť, je nutné využít další komponenty – WebAPI. WebAPI zpřístupní vybrané metody, či třídy a jejich metody na síti. Metody potom mohou být volány pomocí

klasických URL odkazů i s výběrem typu požadavku (`GET`, `POST`...). Třetí komponentou je knihovna DataAccess. Ta poskytuje všechny obslužné operace, které lze provádět nad databází (CRUD - Create / Read / Update / Delete), mimo jiné i její vytváření, nastavování a mazání.

Celá část je pojmenována *Service* právě proto, že by měla být implementována jako Windows service. Taková služba by se dala přirovnat k procesu typu daemon na unixových systémech. Běží jako proces na pozadí, který je spouštěn při startu systému. Startovat by měl ještě předtím, než se do systému kdokoliv přihlásí. To se vyplatí například ve chvíli, kdy se stroj s OPC klientem restartuje či vypne. Stačí jej opět spustit a ovládání OPC serveru, zápis dat do databáze či logování činnosti pokračuje – všechna potřebná nastavení jsou uložená v databázi, podle nichž se OPC klient bude schopen reinicializovat.

Zatím není zcela jasné, zda databáze poběží na stejném stroji jako část *Service*. Nejspíše bude záležet na politice firmy, jíž bude produkt dodáván. V obou případech ale komunikace s databází probíhá stejně. Všechny frameworky vyžadují tzv. *connection string* určující adresu, kam se příslušnými metodami zasílají SQL dotazy. Komunikace probíhá na určitém portu závislém na typu databázového serveru. Aplikace bude stavěna pro práci s databází MSSQL od společnosti Microsoft.

## 3.5 Interní komunikační protokol

Komunikace mezi webovým serverem a OPC klientem probíhá pomocí volání API metod na straně OPC klienta. Implementačně to znamená, že tyto metody nejsou volány přímo, tedy běžným voláním funkce. Volání si lze představit jako volání univerzální funkce, jež akceptuje název požadované metody jako parametr, tedy jako datový typ `string`. Takovou univerzální funkcí je v našem případě metoda objektu umožňujícího odesílat HTTP požadavky a parametrem je URL adresa WebAPI metody. Z toho vyplývají jistá úskalí: IDE ani překladač nemají možnost odhalit chybu před či při kompilaci. Ta se projeví až za běhu.

Aby nedocházelo k překlepům při psaní názvu funkce nebo k úplnému opomenutí její implementace do návrhu přibyla část, která oba konce komunikace sjednocuje. Toho lze docílit rozšířením aplikace o vlastní knihovnu obsahující soubor tříd typu `interface`, tedy soubor rozhraní. Část webového serveru volající metody na back-endu a část back-endu, kde budou implementována těla metod, z tohoto rozhraní budou dědit. To obě části přinutí implementovat metody se stejným názvem a stejnými parametry, jinak se kód nepřeloží. V případě volající části (webový server) musí být programově zajištěn převod názvu metody do datového typu `string`. To v moderním programovacím jazyce není problém – při překladu kódu do CIL jsou ukládána metadata překládaného kódu[2] (aktuální program, aktuální objekt, jeho atributy, aktuální metoda) a za použití knihovny `System.Reflection` je lze snadno získat.

Kromě názvu metod se musí shodovat i datový typ parametrů. V této protokolové části jsou tedy obsaženy i všechny datové modely, se kterými aplikace pracuje jak na straně webového serveru, tak na straně OPC klienta. Parametry jsou při volání WebAPI metody předávány serializované – buďto do řetězce typu JSON (požadavek `POST`) nebo jsou obsaženy přímo v URL adrese volané metody (požadavek `GET`).

## 3.6 Minimalizace zátěže

Předpokládá se, že OPC klient bude sám o sobě velmi vytížen. Bylo nám řečeno, že v praxi je bežné, když musí zpracovávat tisíce tzv. tagů (tag je právě jeden signál ze zařízení určitého datového typu, ze kterého se dají číst hodnoty nebo je zapisovat). Jedním z požadavků na aplikaci je tak minimalizace zátěže OPC klienta, aby mohl především komunikovat s OPC serverem a získávat z něj data. která následně uloží do databáze. Až na jisté vyjímky je představa taková, že připojení uživatelé budou všechna data číst z databáze, nikoliv z OPC klienta. S OPC klientem se bude komunikovat pouze v případě, že se uživatel chystá něco modifikovat. Jak synchronizovat zápis a čtení tak, aby to byla pro aplikaci co nejmenší zátěž a uživatel měl vždy co nejaktuálnější data, je předmětem jednání. Možná řešení jsou uvedena v sekci 5.1.

# Kapitola 4

# Implementace a testování

Pro vývoj aplikace jsem použil integrované vývojové prostředí Visual Studio. Pro ulehčení práce v týmu a současné zajištění správy verzí posloužil vestavěný verzovací systém Team Foundation Server. Frameworky a styl psaní by měly zajišťovat kompatibilitu mezi všemi moderními prohlížeči, doporučený je však Internet Explorer verze 10 a vyšší, pro který je uživatelské rozhraní vyvíjeno přednostně. Dodržování pravidel pro responzivní design umožňuje aplikaci spustit i na zařízeních s menším rozlišením. Smysl má samozřejmě pouze ve vybraných částech, převážně v uživatelské sekci.

## 4.1 Struktura zdrojových souborů

Celá aplikace je rozdělena do šesti samostatných projektů, každý zajišťující jinou funkcionalitu.

1. **DataAccess** – knihovna umožňující práci s databází. Obsahuje její nastavení a operace, které nad ní lze provádět
2. **OpcClient** – spojení s OPC serverem
3. **Service** – hlavní programová smyčka spouštějící OpcClient a WebApi
4. **Shared** – specifikace rozhraní služeb které musí být implementovány jak na straně WebClient, tak na straně OpcClient pro jednotnou komunikaci. Obsahuje také rozhraní objektů, které jsou při takové komunikaci používány jako parametry. Ty jsou zároveň modely webového serveru.
5. **WebApi** – zpřístupnění služeb OpcClient přes požadavky HTTP
6. **WebClient** – implementace webového serveru a klienta – kontrolery, pohledy, služby, klientský JavaScriptový kód

## 4.2 Komunikace se serverem

Pro lepší funkcionalitu a dynamičnost uživatelského rozhraní bylo využito volání funkcí na serveru pomocí kódu napsaného v JavaScriptu. Ten také zajišťuje dynamické překreslování některých částí uživatelského rozhraní, nejčastěji v administrátorské sekci. Tyto části zobrazují aktuální data z backendu aplikace bez potřeby celou stránku znovu načítat a vykreslovat. Jejich aktualizace se děje asynchronně a umožňuje tak uživateli plynulou práci.

K asynchronnímu volání služeb na serveru se zpočátku využívalo technologie AJAX, ten však postupně byl téměř ve všech částech nahrazen voláním pomocí SignalR.

Příjemcem těchto volání je určitá funkce na straně serveru, která získá z backendu aplikace nebo z databáze požadovaná data a zasílá je zpět. Pro serializaci požadovaných/získávaných dat využívá ASP.NET implicitně řetězců ve formátu JSON. Tento způsob serializace je výhodný v mnoha směrech. Z hlediska implementace především v tom, že má širokou podporu jak u klienta (JavaScript), tak také na serveru, kde bylo využito knihovny pro .NET od Newtonsoftu. Veškerou konverzi C# nebo JavaScript objektů do řetězce formátu JSON tak lze provádět pohodlně pomocí knihovních funkcí. Z hlediska objemu přenesených dat je až o 30% úspornější[10] než konkuretní XML formát, kde onu část zabírají redundantní data - značky a jejich atributy (ty na druhou stranu vytváří lepší popis obsahu). Formát JSON se jako nevhodný jeví v případě serializace binárních dat, která zatím není nikde využívána.

Některé ze serverových funkcí byly původně implementovány tak, aby místo navrácení objektu serializovaného do JSON řetězce nejprve interně vytvořily celý HTML dokument, naplnily jej požadovanými daty a vysázený dokument odeslaly zpět ke klientovi (defakto šlo o volání kontroleru). Obslužná funkce v JavaScriptu poté jenom na zvolené místo vloží vytvořený HTML kód. To je výhodné hlavně pro vývojáře, jelikož může jeden dokument obsahující větší množství HTML kódu rozdělit do více dokumentů a vykreslovat je v jednom nadřazeném pohromadě. Z obsáhlých stránek tedy lze udělat jednotlivé logické celky a tím zvýšit přehlednost při jejich vývoji, především v rámci jednoho celku. Nevýhodou je ale zajišťování provázanosti daných celků, protože jsou informace rozděleny do více míst. To se často projevovalo při editaci existujícího kódu, kdy je potřeba kontrolovat obsah více souborů. Se zavedením stylizace Azure (4.3.2) v administrátorské části se obsah všech původně takto oddělených dokumentů spojil a vykreslování částečných pohledů se přestalo používat.

## 4.3 Struktura webu

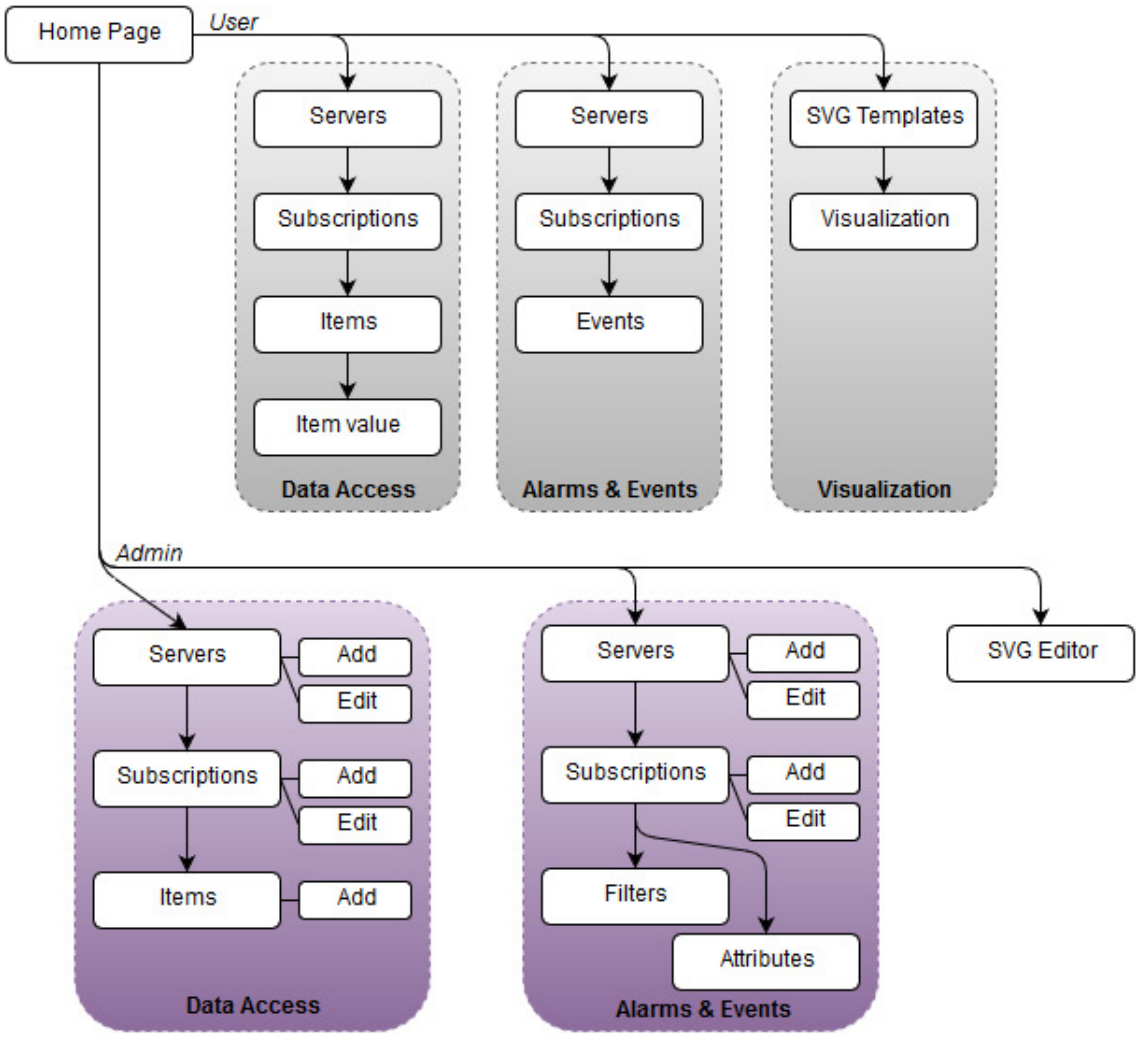
### 4.3.1 Autentifikace a autorizace

Jsou to dva podobně znějící ale významově odlišné pojmy. Web aplikace je zpřístupněn pouze přihlášenému (autentifikovanému) uživateli. Při neoprávněném přístupu je uživatel vždy přesměrován na přihlašovací stránku. Toto omezení je realizováno jednoduchým přidáním atributu nad hlavičku metody kontroleru či nad celý kontroler, čímž omezení aplikujeme na všechny jeho metody. Do atributu se píše přímo role oprávněná takovou metodu volat. Stejnětak je přesměrován přihlášený uživatel volající metodu, ke které nemá dostatečná oprávnění (autorizace).

### 4.3.2 Sekce

Celkem byly implementovány 2 role – obyčejný uživatel a administrátor. Administrátor má práva jako běžný uživatel plus administrátorská.

Úvodní pohled nabízí 3 sekce přístupné běžnému uživateli a jednu sekci přístupnou pouze s právy administrátora. Uživatelskými sekcemi jsou *Data Access*, *Alarms & Events* a *Visualisation*. První dvě jmenované jsou strukturované velmi podobně – jde o soubor pohledů obsahující tabulky. Pomocí tabulek se lze mezi pohledy přepínat a procházet tak hierarchicky strukturu OPC serverů. V případě *Data Access* je to pohled s výpisem všech




Obrázek 4.1: Mapa webu

dostupných DA serverů. Při výběru jednoho z nich je uživatel přesměrován na výpis všech jeho skupin (subscriptions), při výběru skupiny je zobrazen výpis všech položek (tagů) takové skupiny. Poklikáním na položku se otevře pohled, kde lze do položky zapisovat nová hodnota (pokud není jen ke čtení). V případě *Alarms & Events* také prochází seznamem AE serverů a seznamem skupin po výběru jednoho konkrétního. Volbou skupiny je uživatel přesměrován na výpis všech událostí (events). V sekci *Visualisation* je k dispozici seznam všech SVG vizualizací umístěných na serveru. Výberem jedné z nich se uživatel dostane na pohled, kde vizualizace probíhá.

Existuje sice pouze jeden pohled spadající pod administrátorskou sekci, ta by se ale dala rozdělit do podobných celků jaké má uživatelská. Rozdíl je v tom, že při volbě takového celku se nemění celý pohled, ale proběhne jen dynamické překreslení jeho částí. Při tvorbě této sekce jsem se inspiroval rozložením a funkcionalitou rozhraní, jakého využívá web Azure společnosti Microsoft. To se odrazilo nárustem obsahu jednak HTML souboru pohledu, který musí obsahovat všechny zobrazitelné podčásti, ale také souboru obsahující JavaScriptový kód k zajištění dynamické funkcionality pohledu. V administrátorské sekci má uživatel možnost modifikovat strukturu OPC serverů – přidávat, odebírat a editovat

DA nebo AE servery, jejich skupiny, přidávat nebo mazat položky těchto skupin.

### 4.3.3 Navigace

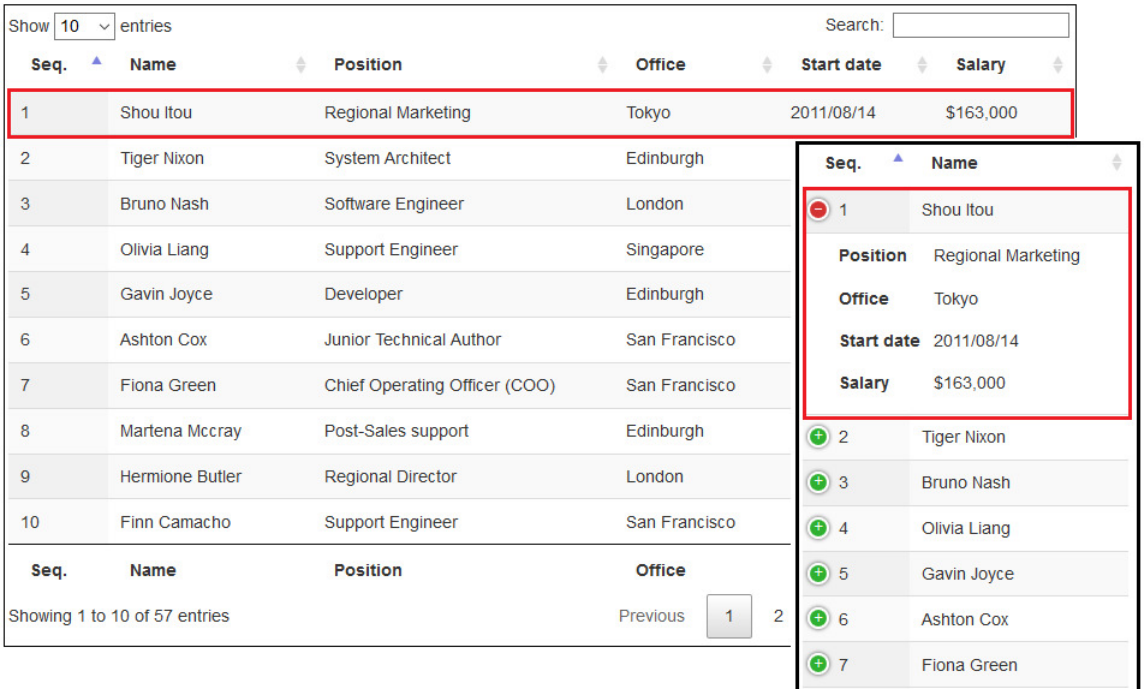
Pro lepší orientaci a vědomí uživatele o aktuální pozici v rámci aplikace je součástí každého pohledu k dispozici navigace, tzv. breadcrumb. Jde o horizontálně rozloženou komponentu, kde jsou hierarchicky vedle sebe umístěny nadřazené úrovně pohledů. Uživateli tím zpětně říká, jak se na aktuální pohled postupně dostal a umožňuje mu kliknutím přejít na kterýkoli z nadřazených pohledů. Tento navigační prvek je v oblasti uživatelských rozhraní hojně využívaný. Běžný uživatel se s ním mohl setkat v prostředí operačních systémů, kde je takto zobrazována pozice v souborovém systému při procházení adresářové struktury. Nejjednodušší cestou bylo využít předpřipravené komponenty v rámci knihovny Bootstrap. Ta k tomuto účelu využívá seznam – element `<ul>`, kde jednotlivé pohledy jsou položkami `<li>` tohoto seznamu.

## 4.4 Tabulární data

Samotný Bootstrap na ošetření responzivity tabulek nestačí. Přímo v dokumentaci je zmíněna třída `responsive`, ta však na příliš malé obrazovce pouze umožní horizontální posouvání obsahu tabulky, takže řádky nejsou vidět celé.

Tento problém řeší framework Telerik Kendo zmíněný v zadání práce, konkrétně Kendo UI Grid. Ještě než se začalo s tabulkami pracovat, vedení kvůli nevyhovujícím licencím použití Telerik Kendo zavrhlo a musela se tak najít alternativa.

Responzivita u tabulek lze řešit jedině její transformací. Pro ilustraci si představme případ, kdy má tabulka 10 sloupců. Každý sloupec musí mít adekvátní šířku aby pojmul svůj obsah. Pokud by měl takový počet sloupců být vedle sebe, šířka obrazovky zařízení by byla samožejmě nedostačující. Probírány byly dvě možnosti transformací. První je vynechání některých sloupců při zúžení tabulky, druhou možností je transformace sloupců na řádky a vykreslování jednotlivých řádků tabulky jako samostatných objektů. K tomuto účelu byl nakonec využit JavaScriptový plug-in DataTables s licencí MIT. Responsivitu řeší spojením zmíněných transformací. Při zúžení tabulky zůstanou zobrazeny jen vybrané sloupce, ostatní jsou skryty. U zobrazených sloupců je poté přidán prvek umožňující takový řádek rozkliknout a nahlédnout na informace ze sloupců skrytých. Responzivity lze dosáhnout přidáním třídy `responsive`, která se ale kryje se stejnojmenou třídou z Bootstrap. DataTables naštěstí poskytuje všechny své třídy duplikované s prefixem, proto šlo využít třídy `dt-responsive`.




Obrázek 4.2: Skrytí sloupců a jejich transformace do řádků při zúžení tabulky DataTables

## 4.5 Integrace SVG editoru

Jedním z úkolů bylo vybrat nějaký z dostupných online SVG editorů a integrovat ho do naší aplikace. Při výběru se provnávaly nejrůznější aspekty těchto editorů. Těmi byly například kompatibilita mezi prohlížeči, intuitivita prostředí a ovládání, spektrum možných operací a jiné. Důležitým aspektem byl kromě viditelných vlastností i zdrojový kód editoru. Preferován byl kód, který je co nejjednodušeji rozšiřitelný a modifikovatelný. Podmínkou zužující výběr bylo licencování MIT.

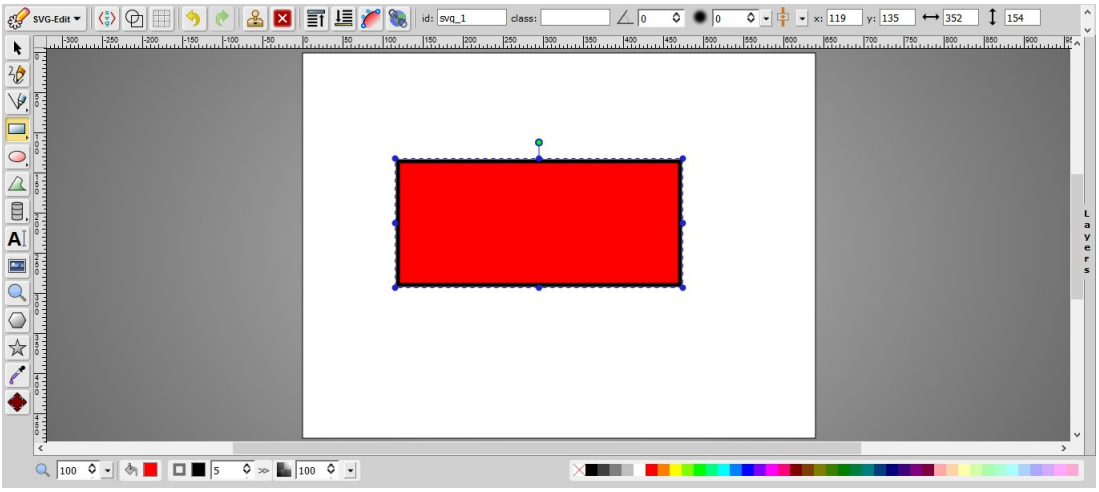
### 4.5.1 SVG-Edit

Editor vytvořený vývojáři Google. Byl nám doporučen i našimi mentory. Naneštěstí hned při prvním ohledání jsem zjstil, že v prohlížeči Firefox kreslí objekty úplně jinde, než je aktuální poloha kurzoru. Jako nefunkční se ukázalo také přibližování a posouvání na kolečku myši. Prostředí se proti konkurenčnímu Method Draw zdá být neintuitivní a nemoderní.

### 4.5.2 Method Draw

Testován v Chrome, Firefox i Internet Explorer. Proti SVG-Edit je mnohem modernější a intuitivnější. Prvky pro editaci atributů vybraného objektů jsou lépe popsány i ovládány. Kromě zobrazení mřížky lze taky zapnout přichytávání k mřížce, tato možnost u SVG-Edit chybí. Výhodou je také možnost použití klávesových zkratek, minimálně pro kopírování a vkládání objektů. V editoru chybí možnost propojení objektů čarou a taky ze není zaveden systém vrstvení, tyto dvě funckionality však nejspíš nebudou pro naše účely potřeba.

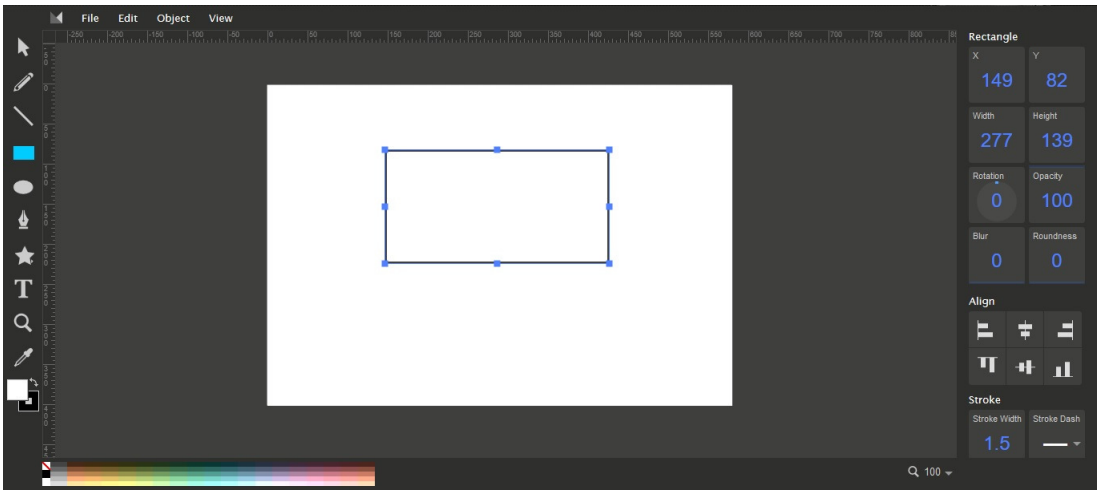
Tento editor byl zvolen pro začlenění do naší aplikace. Při importaci a procházení zdrojových kódů jsem zjistil, že velká část kódu je převzata z editoru SVG-Edit. Zdá se, že




Obrázek 4.3: Náhled webové aplikace SVG-Edit

kreslící jádro je stejné, jen byl změněn vzhled prostředí.

Do editoru byla prozatím přidána možnost vybraný objekt propojit s odpovídajícím tagem některého z automatizačních prvků. Pro jeho výběr byl do rozhraní editoru přidán speciální panel, kde jsou dostupné tagy vylistovány. Propojení je realizováno přidáním identifikátoru zvoleného tagu do určitého atributu objektu. Přímá hodnota takového tagu bude čtena až při samotné vizualizaci, kdy se na základě její hodnoty obrázek nějakým způsobem mění. Pokud je však vkládán text, hodnota tagu je přímo vypsána. O přiřazení tagu objektu se lze přesvědčit zobrazením zdrojového kódu aktuálního SVG nákresu.




Obrázek 4.4: Náhled webové aplikace Method Draw

## 4.6 Vizualizace dat

Základem vizualizací v naší aplikaci je využití jazyka SVG k ilustraci komponent v kombinaci s jazykem JavaScript pro jejich animaci a ovládání. Jednotlivé soubory s SVG šablonami jsou zatím uloženy na serveru, nikoliv v databázi. Jde o soubory s příponou `.svg`

v předem určeném adresáři. Obsah adresáře je načítán pomocí SignalR volání, všechny nalezené SVG soubory jsou vylistovány do tabulky. Po volbě konkrérního souboru není do klientského zařízení posílán soubor jako takový – posílán je řetězec naplněný textovým obsahem souboru na straně serveru, abychom se vyhnuli přenášení binárního objektu. Tato akce je taktéž realizována pomocí SignalR. Řetězec potom stačí bez jakýchkoliv úprav pomocí JavaScriptového kódu vložit do předpřipraveného elementu, vykreslení vizualizace provádí prohlížeč.

### 4.6.1 D3.js

Data-Driven Documents je JavaScriptová knihovna sloužící k vytvoření komplexních, dynamických a interaktivních vizualizací ve webovém prohlížeči a propojení webových dokumentů s datovými zdroji. Přestože je vizualizace vlastně kódem ve formátu XML, knihovna jQuery pro manipulaci s takovým kódem není uplně dostačující, proto je v naší aplikaci využito D3. Obě knihovny ulehčují práci při výběru prvků a modifikaci DOM, v mnoha případech mají velmi podobnou syntaxi. Umožňují přidávání a odebírání elementů, editaci atributů elementů, stylizaci elementů. D3 dosahuje svých cílů pomocí čtyř kroků[7]:

1. *načtení* dat do paměti prohlížeče
2. *propojení* dat s elementy dokumentu, případně vytvoření nových elementů
3. *transformace* těchto elementů, nastavování vizuálních vlastností elementu v závislosti na datech se kterými je propojený
4. *přechod* elementů mezi jednotlivými stavy jako odpověď na uživatelovy akce

Pro veškerou manipulaci s SVG objekty je v aplikaci použito knihovny D3. Ve vytvořené ukázkové vizualizaci je to tak především efekt rotace a postupného vyplňování objektu (toho je dosaženo dynamickou změnou gradientu výplně).

## 4.7 Testování

K projektu WebClient byla vytvořena sada testů. Ta momentálně pomáhá udržet aplikaci funkční při dílčích změnách a refactoringu. Testována je zatím hlavně interakce uživatelského prostředí na uživatelovy akce. Testy jsou integrační, což znamená, že je ověřována propojenost jednotlivých komponent uvnitř aplikace. K tomuto účelu nám posloužil testovací framework Selenium.

Lze si zvolit prohlížeč, ve kterém chceme webovou aplikaci testovat. Jelikož vývoj provádím v prohlížeči Internet Explorer a vyvíjenou část rovnou testuji, testy jsou zatím psány pro Mozillu Firefox, abych se ujistil, že funguje správně i tam. V budoucnu bude prováděno stejné testování ve všech prohlížečích, které by měla aplikace podporovat.

Jak už jsem zmínil, testována je hlavně interakce uživatelského prostředí. To se děje pomocí simulace uživatelovy akce, ovládáním klávesnice a myši. V kódu se lze odkazovat na elementy stránky pomocí jejich atributu `id` a tak nad nimi vykonat kliknutí nebo třeba vkládání textu. Taková akce vede ke změně na stránce nebo změně celé stránky. Předpoklad se ověřuje jako ve většině testovacích frameworků funkcí `assert`.

Před a po každém dílčím testu jsou volány funkce `SetUp` respektive `TearDown`.

# Kapitola 5

# Budoucí vývoj

Při poslední schůzce byl vytvořen plán pro další iteraci. Je potřeba celou aplikaci refaktorizovat a stabilizovat. Ze všeho nejdřív se celá pokryje testy, aby při refaktorizaci jedné části nedošlo ke ztrátě funkcionality v části jiné.

## 5.1 Synchronizace při čtení dat

Na posledním meetingu byla diskutována hlavně problematika synchronizace OPC klienta a webového serveru a její možná řešení. Při momentální implementaci by OPC klient ukládal do databáze aktualizované hodnoty po určitém intervalu. Tento interval je dostupný i ve webové aplikaci, a tak může klient číst z databáze ve stejném intervalu. Tyto dva intervaly však nejsou synchronizovány – vzniká tak riziko, že uživatel bude číst vždy těsně před aktualizací hodnot v databázi. Ideální by samozřejmě bylo, kdyby si klient nová data načítal ihned po aktualizaci dat v databázi.

Jako řešení se zdá být implementace opačné komunikace – volání metod webového serveru z OPC klienta. OPC klient ví, kdy sám do databáze zapisuje a může tak poslat na server notifikaci, že byla data aktualizována. Musí se však dbát na co nejnižší zátěž OPC klienta (3.6), takže například komunikace musí být jednostranná – OPC klient pošle notifikaci, ale už nesmí čekat na odpověď, která by mohla trvat delší dobu nebo by nemusela přijít vůbec. Komunikace se tak stává z části nespolehlivou a je potřeba ji nějak pojistit.

## 5.2 Logování událostí

V celé aplikaci přibudou na vybraná místa příkazy umožňující zaznamenávat stav aplikace, hodnotu proměnných a dodatečné informace týkající se běhu aplikace v danou chvíli ve zvoleném formátu. Až bude aplikace využívána zákazníkem a dojde k chybě, všechny relevantní informace budou obsaženy v logovacím souboru. To umožní rekonstrukci takového stavu a zjednoduší pátrání po původu chyby. Některé nástroje umožňují dodatečné operace – například ukládání těchto informací do databáze, jejich odesílání emailem, umístění na specifickou adresu pro vzdálený přístup[6] apod.

Pro záznam událostí na backendu je zvažována techologie Log4Net, pro události spojené s webovou aplikací se bude využívat ELMAH.

## 5.3 Windows autentifikace

Momentálně je v aplikaci využívána tzv. formulářová autentifikace. Uživatel vyplní svoje uživatelské jméno a heslo, načež jsou vložená data zkontrolována s těmi v databázi. Do budoucna se počítá s rozšířením o další možnost – Windows Authentication. Vše je víceméně automatizováno, kontrolováno je povolení přístupu pro doménu, do níž spadá uživatelský účet, pod kterým je uživatel momentálně přihlášen do operačního systému.

## 5.4 SVG šablony

V rámci vizualizace je potřeba vytvořit celá knihovna objektů popsaných kódem SVG reprezentujících různá zařízení z automatizace. Tyto objekty by mimo tvar měly mít také předem určené možné změny, které je na objekt možné aplikovat – typ objektu. Na základě typu se potom bude spouštět odpovídající animace. V rámci editoru typ také bude určovat, které části jsou modifikovatelné uživatelem (směr animace, rozsah animace, barvy, textové popisky apod.) a jejich možné provázání s daty z automatizace. To nám umožní alespoň částečnou univerzálnost vytvářených vizualizací.

Mimo zmíněná vylepšení se počítá se zavedením TypeScrit jako náhrady za JavaScript. Přináší spoustu vylepšení, z nichž asi největší vyplývá z názvu – typovost.

Dalším plánovaným vylepšením, které se dotkne back-endu i front-endu, je procházení tagů dostupných na OPC serveru pomocí stromové struktury. Server takovou strukturu obsahuje implicitně, tagy jsou tak rozděleny do určitých logických celků. Pro jednoduchost a zběžné ověření funkčnosti jsou momentálně všechny tagy zobrazovány na stejné úrovni. Taková změna bude vyžadovat vytvoření datového typu (modelu), do kterého lze stromovou strukturu uložit a která se bude přenášet mezi OPC klientem a webovým serverem. Na straně webového serveru potom JavaScriptový kód umožňující pohyb v takové struktuře.

# Kapitola 6

# Závěr

V rámci bakalářské práce se povedlo implementovat aplikaci umožňující práci s daty z automatizace. K zajištění komunikace s automatizačními prvky bylo využito doporučených knihoven od OPC Foundation. Uživatelské rozhraní umožňující konkrétní operace s automatizačními prvky je vykreslováno prohlížečem a je dostupné odkudkoliv, jedná se totiž o mobilní webovou aplikaci. K reprezentaci dat byla mimo běžné textové výpisy zprovozněna také vizualizace využivající technologii SVG.

Na základě kvalitního návrhu je aplikace rozdělena na nezávislé logické celky. Jednotlivé celky jsou díky své specifické implementaci vysoce testovatelné, snadno modifikovatelné a rozšiřitelné, což byl jeden z hlavních cílů vzhledem k plánovanému vývoji aplikace v budoucnu.

Kromě této technické zprávy bylo k aplikaci a jejím částem napsáno několik dokumentací v anglickém jazyce sloužící pro firemní účely.

Největším přínosem pro mě bylo pochopení principů a fungování webových aplikací a jejich implementace v praxi. Momentálně zažívají webové aplikace rozkvět a proto bych se chtěl této oblasti věnovat i v budoucnu. Veškerá práce byla evidována a vývoj mé části doposud trval celkem 185 hodin.

# Literatura


[1] Aguilar, J. M.: *SignalR Programming in Microsoft ASP.NET*. Microsoft Press, 2014, ISBN 978-0-7356-8388-4.

[2] Albahari, J.; Albahari, B.: *C# 4.0 in a Nutshell, 4th Edition*. O'Reilly Media, 2010, ISBN 978-0-596-80095-6.

[3] Eisenberg, J. D.: *SVG Essentials*. O'Reilly Media, 2002, ISBN 978-0-596-00223-7.

[4] Gourley, D.; Totty, B.; Sayer, M.; aj.: *HTTP: The Definitive Guide*. O'Reilly Media, 2002, ISBN 978-1-56592-509-0.

[5] Jennings, T.: *SignalR*. 2014, [Online; navštíveno 14.5.2015].
URL http://tylerscode.com/2014/11/signalr/

[6] Mitchell, S.: *Logging Error Details with ELMAH*. Červen 2009, [Online; navštíveno 17.5.2016].
URL
http://www.asp.net/web-forms/overview/older-versions-getting-started/deploying-web-site-projects/logging-error-details-with-elmah-cs

[7] Murray, S.: *Interactive Data Visualization for the Web*. O'Reilly Media, 2013, ISBN 978-1-4493-3973-9.

[8] Opc Task Force: *OPC Overview*. 1998, [Online; navštíveno 14.5.2016].
URL
https://www.fer.unizg.hr/_download/repository/OPC_Overview_1.00.pdf

[9] Rauschmayer, A.: *Speaking JavaScript*. O'Reilly Media, 2014, ISBN 978-1-4493-6503-5.

[10] Vávrů, J.: *jQuery Mobile*. Computer Press, 2013, ISBN 978-80-251-3811-3.

# Přílohy

# Seznam příloh

# Příloha A

# Obsah CD

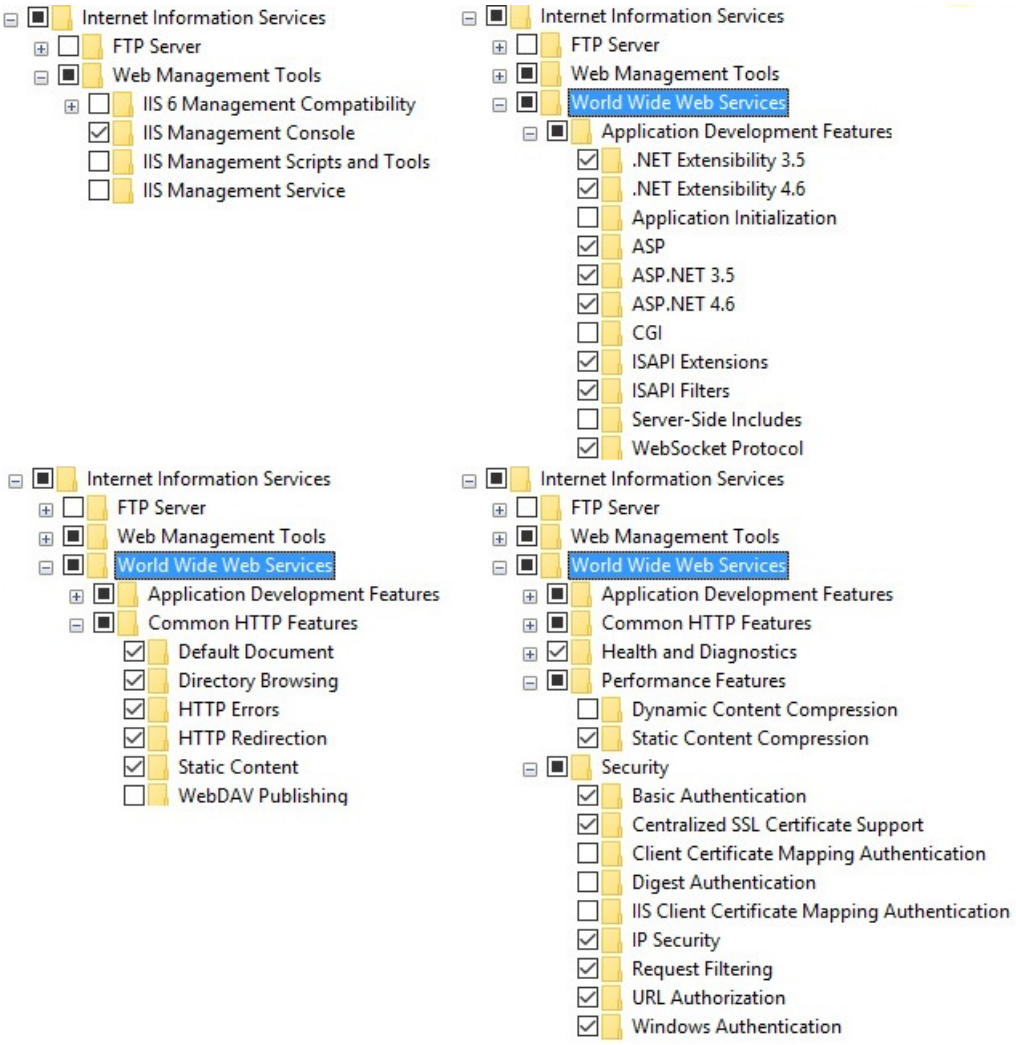
Přiložené CD obsahuje:

- Technická zpráva ve formátu PDF
- Zdrojové soubory technické zprávy
- Zdrojové soubory aplikace

# Příloha B

# Manual

1. Instalace OS Microsoft Windows 10
2. Instalace IDE Microsoft Visual Studio 2015 Update 2
3. Instalace simulátoru OPC servevru
4. Instalace databázového serveru MSSQL 2015; pojmenování instance `SQLEXPRESS`
5. Zprovoznění komponent v *Programs* & *features* podle obrázku B.1
6. Otevření hlavního `.sln` souboru ve Visual Studio
7. Spuštění pomocí Visual Studio




Obrázek B.1: *Programs & features*