Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2015/2016

Zadání bakalářské práce

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

Pokyny:

1. Prozkoumat technologie .NET MVC a OPC Foundation standard (OLE for process control).
2. Upravit architekturu stávající ABB mobilní aplikace propojující data z kontrolního systému OPC (doporučit možná vylepšení).
3. Odebrat stávajícího OPC Softing klienta a nahradit ho ABB preferovaným OPC Foundation (klient) - včetně implementace propojení s novým klientem. Všechny kroky musí být schváleny zadavatelem (ABB).
4. Zrevidovat aktuální databázové schéma a doporučit vhodné ORM pro toto řešení (PetaPoco, AsyncPoco atp.).
5. Kompletně nahradit stávající Angular JS za .NET MVC technologii.
6. Implementovat nezbytné změny a optimalizace, provést testy a navrhnout možná budoucí vylepšení.

Literatura:

1. Galloway, J., Wilson, B., Allen, K. S., Matson, D.: Professional ASP.NET MVC 5. Indiana: John Wiley Sons, Inc., 2014. ISBN 978-1-118-79475-3.
2. Freeman, A.: Pro ASP.NET MVC 5 Platform. Apress, 2014. ISBN 978-1-430-26542-9.
3. Lange, J.: OPC (englischsprachige Ausgabe): From Data Access to Unified Architecture. Vde Verlag Gmbh, 2010. ISBN 978-3-800-73242-5.
4. 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.
5. OPC Foundation [online]. Available at: https://opcfoundation.org

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).

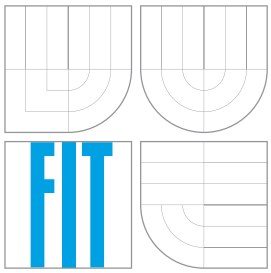
Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo 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

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

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 BACK-END ČÁSTI APLIKACE)

MOBILE ACCESSING OF AUTOMATION DATA (BACK-END DEVELOPMENT)

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — LUKÁŠ HABARTA
AUTHOR

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

BRNO 2016

Abstrakt

Tato bakalářská práce se zabývá průzkumem a popisem technologií použitých při vývoji mobilní aplikace. Dále návrhem a implementací mobilního přístupu k datům z automatizačních prvků. Aplikace pracuje se servery založenými na standardu OPC Foundation Classic.

Abstract

This bachelor thesis is concerning about research and description of technologies which were used for mobile application development. Design and implementation of mobile accessing of automation data is also discussed. Application works with servers based on OPC Foundation Classic standard.

Klíčová slova

Automatizace, získávání dat, zpracování dat, OPC, MVC, C#, Data Access, Alarm & Events

Keywords

Automatization, data accessing, data processing, OPC, MVC, C#, Data Access, Alarm & Events

Citace

HABARTA, Lukáš. *Mobilní přístup k datům z automatizačních prvků (vývoj Back-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 Back-End části aplikace)

Prohlášení

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

.....
Lukáš Habarta
17. května 2016

Poděkování

Rád bych poděkoval paní Šárce Květoňové za ochotu a pomoc při psaní této práce. Dále bych rád poděkoval společnosti ABB Česká republika, že mi umožnila tuto práci psát a její zástupci mi poskytli odborný dohled. Ze společnosti ABB patří mé poděkování hlavně těmto osobám: Michael Filsák, Mário Čuboň, Ondřej Spilka, Jan Mináč a Jindřich Šonka.

Obsah

Kapitola 1

Úvod

Tato práce vznikla podle zadání společnosti ABB Česká republika za účelem vytvořit mobilní aplikaci, která by umožňovala komunikaci se serverem založeným na standardu OPC Foundation včetně čtení a zápisu dat, upozorněním na vzniklé události ve výrobě. Kvůli velkému rozsahu byla aplikace rozdělena do dvou částí. Část týkající se back–endu popisuje tato práce, část týkající se front–endu bude popisovat práce kolegy Jana Kubiše. Z důvodu rozdělení práce a prolínání jednotlivých částí je velmi těžké se věnovat jen popisu back–endu, některé informace dávají smysl až při konfrontaci s front–endem. Pro pochopení celého konceptu je vhodné si přečíst i práci od kolegy Kubiše.

Před samostatným vývojem aplikace bylo nutné analyzovat stávající aplikaci a zhodnotit její nedostatky včetně postupu, jak ji vylepšit. Při vývoji byl kladen důraz na čistý a čitelný kód, který bude možné lehce upravovat a rozšiřovat podle potřeby.

Při vývoji bylo nutné vzít v potaz speciální požadavky. Mezi tyto požadavky patří například bezpečnost, práce v reálném čase a dostatečné testování, aby se zabránilo pádu aplikace a tím i případným škodám, které by tímto mohly vzniknout. Tato teoretická část práce obsahuje základní popis použitých technologií, popis problémů, které nastaly až během vývoje, včetně jejich řešení. Princip vytváření OPC klienta je zde popsán jako seznam funkcí, které byly použity, jejich parametry včetně popisu významu a důležitosti vybraných parametrů. Tato práce se bude také zabývat návrhem a tvorbou databáze, která je pro tuto aplikaci nutná, dále testováním aplikace a hodnocením dosažených výsledků včetně návrhů na možná vylepšení.

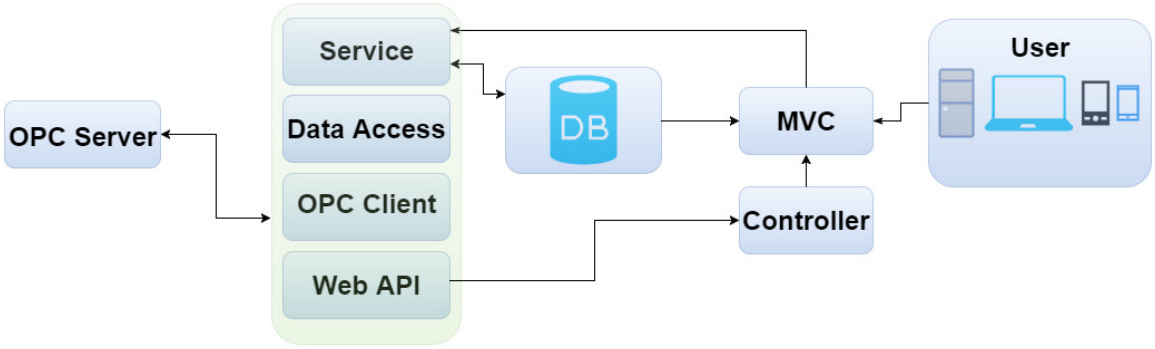
Kapitola 2

Návrh nasazení systému v praxi

Jelikož jsem neměl možnost vyzkoušet systém v praxi, jde pouze o odhadované nasazení. Při kreslení tohoto obrázku jsem bral v potaz bezpečnost systému proti zásahu zvenčí. Systém je možné rozdělit na tři pomyslné části:

- Uživatel komunikující s MVC
- Uzavřené části systému, ke kterým se uživatel bez použití MVC nedostane. Jde o Databázi, Service, DataAccess, OpcClient
- OPC server

Uživatel komunikující s MVC nemůže nijak komunikovat s jinou částí systému. Pokud by se podařilo potencionálnímu útočníkovi dostat nekorektní data až k service, tak se data nedostanou do samotného OPC serveru, protože OPC server je samostatná 3.část, s kterou může komunikovat pouze OPC Client.



Obrázek 2.1: Předpokládaný způsob nasazení v praxi

Management mi po konzultaci schválil úpravu v podobě přidání jedné části systému, a to konkrétně cache paměti mezi front–endem a back–endem. Tato cache paměť bude sloužit k uchovávání dat s méně častou změnou a jako uložiště hodnot pro všechny připojené uživatele. K tomuto kroku vedlo plánované nasazení SignalR na straně back–endu, který bude informovat WebClienta o nových datech v OPC serveru. Uživatel tato data bude stahovat z databáze, ale je neefektivní, aby stahoval i data, která neprošla změnou. Protože bude připojeno předem neznámé množství uživatelů, mohlo by dojít k zahlcení databáze.

Jako příklad uvedu následující výpočet: je připojeno 10 uživatelů, každý uživatel sleduje 1 000 hodnot, zajímá se o 500 hodnot, které se mění každou sekundu a 500 hodnot, které se mění každou minutu. Back–end informuje SignalR hub o změně dat a SignalR začne všem uživatelům stahovat data. Aby byla zachována nezávislost uživatele na jiném uživateli, bude se stahovat 10 000 hodnot každou sekundu, to znamená 10 000 dotazů na databázi. Polovina těchto hodnot se přitom nezměnila a velká část hodnot je společná pro více uživatelů. Tato metoda je neefektivní a jako řešení se nabízí plánované použití cache paměti. Všechna data se uloží do cache paměti a WebClient je bude stahovat až odtud. V nejhorším případě dojde k 5 000 dotazům na databázi každou sekundu a jednou za minutu k 10 000 dotazům.

Ve většině případů ale dojde k daleko menšímu množství dotazů, protože se budou stahovat jen data, která se v cache paměti ještě nenachází.

Tato část se v nákresu ještě nenachází, protože není zahrnuta ani v kódu a tento nákres ukazuje pouze jednotlivé části již hotového kódu.

Požadavky společnosti ABB – neformální specifikace

Požadavky na funkcionalitu jsou součástí zadání, ale byly zde ještě dodatečné požadavky, které byly předány neformálně:

- Aplikace bude umožňovat připojení na libovolný server splňující požadavky
- Není možné se připojit 2x na server s adresou již připojeného serveru
- Subscription musí mít uživatelem definované jméno, tzn. nebude náhodně generované serverem
- Každý item na serveru může být v subscription pouze jednou

Kapitola 3

Mobilní aplikace

Problém, který se pokusí naše aplikace vyřešit je situace, kdy ve velkém podniku je potřeba kontrolovat značné množství strojů, zařízení a dalších přístrojů produkujících konkrétní informace, které je potřeba roztřídit a využít. V dnešní době je pro každou sekci zřízeno kontrolní středisko, které se sestává z velkého množství monitorů a každý monitor zobrazuje relevantní informace pro určitého kontrolního technika. S rostoucím množstvím strojů a snahou o větší bezpečnost a efektivitu práce roste i množství potřebných kontrolních techniků a monitorů. Jedná se o nemalé finanční náklady pro společnosti a tedy i potencionální zákazníky, jelikož náklady společnosti na bezpečnost se musí promítnout i do ceny konečného výrobku.

Největší problém nastává, pokud se kontrolní technik vzdálí od své stanice, a to třeba i z důvodu opravy nebo hlášení nějakého problému. Jeho stanoviště se totiž stává nekontrolovaným a musí sehnat nějakou osobu, která je proškolená a bude stav monitorovat místo něj. Další problém může nastat při chybě hardwaru počítače, který daný technik používá. Pokud se počítač pokazí a přestane fungovat, tak technik ztrácí přehled nad tím, co se děje v místech, která má kontolovat. Tento problém řeší naše aplikace.

Jelikož aplikace podobného druhu ještě neexistuje, jsou očekávané výsledky pouze odhad a nejsou nijak podloženy. Aplikace má za úkol usnadnit kontrolním technikům práci a snížit případně i náklady firem. Po řádném nastavení umožňuje aplikace každému technikovi kontrolovat jeho stanici i mimo monitory za pomoci libovolného mobilního zařízení s přístupem k lokální síti. Pokud se technik vzdálí mimo lokální síť je pro práci vyžadováno připojení k internetu. Aplikace umožňuje zobrazovat pro technika užitečné informace v reálném čase, včetně možnosti zápisu hodnot do nastavení jednotlivých strojů. Pro zobrazení chodu zařízení je použita vizualizace za pomoci pohybu SVG elementů napojených na konkrétní signály a informace z OPC serveru.

3.1 Stávající mobilní aplikace

Stávající mobilní aplikace od společnosti ABB, jejíž tvůrcem je Michael Filsák, jenž byl zároveň našim mentorem v průběhu vývoje, měla několik nedostatků, které bylo třeba vyřešit. Aplikace umožňovala základní vizualizaci dat za pomocí SVG elementů, získávání dat z OPC serveru, jejich aktualizaci pomocí pravidelného dotazu na backendu. Mezi nedostatky byla chybějící databáze, použití AngularJS, aktualizace hodnot pomocí dotazu (zbytečná zátěž serveru), použití knihoven od společnosti Softing.

Vzhledem k rozsahu jednotlivých problémů bylo nutné celou aplikaci přepsat. A to bylo

cílem této bakalářské práce. Bylo potřeba přidat databázi jako uložiště hodnot a nastavení, nahradit AngularJS za MVC.NET, dotazování se na hodnoty nahradit opačným principem - server informuje klienta o nových hodnotách a ten si je stáhne z databáze,čehož lze docílit pomocí technologie SignalR. Dalším úkolem bylo použít knihovny od OPC foundation namísto knihoven od společnosti Softing. Aplikace měla umožnit práci s OPC Data Access a OPC Alarm & Events, čehož bylo docíleno.

3.2 Vývoj mobilní aplikace

Mobilní aplikací je zde myšlena aplikace, která umožňuje mobilní přístup, nikoliv jako aplikace, která je určena pro mobilní zařízení. Při vývoji aplikace, která umožňuje použití i na mobilních zařízeních, je potřeba se věnovat několika základním bodům:

- K jakému účelu bude aplikace využita
- Cílová skupina uživatelů
- Pro jaký operační systém bude vyvíjena
- Design aplikace
- Náročnost na koncové zařízení

K jakému účelu bude aplikace využita - to bylo u naší aplikace předem známo a s tímto se aplikace vyvíjela. Tuto otázku za nás vyřešila společnost ABB. Bude vyvíjena pro kontrolu přístrojů z oblasti automatizace výroby.

Cílová skupina uživatelů - kontrolní technici, případně administrátoři využívající kontrolní systémy, které spolupracují se standardem OPC Foundation.

Pro jaký operační systém bude vyvíjena - jedná se o webovou aplikaci, proto není na operační systém kladen důraz. Aplikace je funkční jak na Android, iOS, WindowsPhone, tak i na operačních systémech pro běžné počítače, které mají webový prohlížeč.

Design aplikace - tato otázka se týká spíše front–endu, proto ji popíši jen stručně. Bylo třeba dosáhnout responzivního designu pro zobrazení na běžných velikostech a rozlišeních obrazovek jak mobilních zařízení, tak běžných počítačů. Byla použita technologie od společnosti Twitter - Bootstrap, DataTables a jako vzhled u administrátora je použit Azure. Využití Azure u administrátora má své opodstatnění, kdy administrátor může mít otevřeno větší množství panelů a posouvání směrem dolů je nepřirozené a špatně zachovává hierarchii.

Náročnost na mobilní zařízení - je otázkou použitých technologií na front–endu, jelikož celý back–end bude fungovat odděleně od front–endu. Zjednodušeně lze říct, že by s aplikací mělo fungovat každé mobilní zařízení s aktualizovaným prohlížečem na poslední verzi, u běžných počítačů jsou podporovány prohlížeče Google Chrome, Mozilla Firefox, Safari, Microsoft Edge, Internet Explorer od verze 10.

Kapitola 4

Technologie použité při vývoji back–endu aplikace

Pro vývoj byly technologie dané požadavky společnosti ABB, mezi jejich požadavky bylo nahrazení AngularJS za MVC ASP.NET, použití knihoven od OPC foundation.

4.1 MVC ASP.NET

MVC pro ASP.NET je framework pro tvorbu webu.

4.1.1 Architektonický vzor MVC

Jak již bylo zmíněno v nadpise, MVC je architektonický vzor, bez kterého si nelze složitější web snad ani představit. Jde o zkratku z Model-View-Controller, jehož účelem je rozdělit logiku a samotný výstup, jinak by se kód u složitějšího webu stal hůře čitelným.

Model
Typický model obsahuje logiku včetně dotazů do databáze, výpočtů hodnot a dalších operací, které jsou nevhodné pro View. Uživatel jako takový nemá možnost přímé komunikace s modelem. Model přijme dotaz a vrátí data, která byla požadována. Základní model nepotřebuje vědět, který uživatel data vyžádal, ani ke kterému uživateli data půjdou, výjimkou jsou data patřící konkrétnímu uživateli, ale tato informace bývá zpravidla posílána v dotazu.

View
Zajišťuje samotné zobrazení dat. Uživatel pracuje s konkrétním view, pomocí kterého posílá dotazy modelu a vykresluje data, která model posílá zpět. View bývá zpravidla nějaký druh HTML a případného JavaScriptu. View je součástí front–endu, v této aplikaci se pro view používá:

- ASP.NET Razor C#, upravené HTML pro ASP.NET MVC
- JavaScript
- jQuery
- D3.js pro vizualizaci dat - pohyb SVG elementů
- SignalR pro získávání dat z back–endu aplikace

Controller
Jedná se o prostředníka, který zajišťuje komunikaci mezi view a modelem. Od uživatele dostává přes view dotaz pomocí parametrů v URL a přeposílá je modelu. Model po zpracování vrátí data kontroléru a ten je zobrazí uživateli pomocí view. [2] [6]

4.2 OPC

OPC je zkratkou pro OLE for Process Control. Jedná se o standard, který specifikuje rozhraní mezi klientem a serverem, stejně tak jako mezi serverem a serverem. Standard dále specifikuje přístup k datům v reálném čase, monitorování Alarms & Events, přístup k historickým datům a další aplikace. Když byl standard uveden v platnost v roce 1996, bylo jeho cílem abstrahovat specifické protokoly PLC do standardizovaného rozhraní umožňující HMI/SCADA systémům komunikaci s middle–man, který by převedl generické požadavky na čtení/zápis do specifických požadavků každého zařízení a naopak. HMI je zkratkou Human–Machine Interface, HMI je vstupně–výstupní zařízení, pomocí kterého operátor ovládá nějaký proces, a zároveň zobrazuje operátorovi výstupní data procesu. SCADA je zkratkou Supervisory Control and Data Acquisition a jedná se o centralizovaný systém, který monitoruje a kontroluje celou spádovou oblast. [1] Součástky a monitorovací zařízení splňující standard OPC foundation mají široké použití. Jejich použití je možné všude, kde je potřeba kontrolovat nastalé události a nebo jsou zde data, která jsou obnovována často, a tudíž je objem dat velký. Jiné systémy totiž nedokáží zpracovat tak velké množství dat v krátkém časovém úseku.

4.2.1 OPC classic

Jedná se o standard založený na Microsoft Windows a jejich technologiích COM/DCOM [4]. Slouží k výměně dat mezi softwarovými komponenty. Specifikace poskytuje rozdílné definice pro přístup k datům, alarmům a historickým datům. Jedná se o původní standard, který byl používán až do nástupu Unified Architecture.

OPC Data Access
Jde o specifikaci, která definuje výměnu dat včetně hodnot, času a kvality informace.

OPC Alarm & Events
Jde o specifikaci, která definuje výměnu alarmu a informace o typu události, stejně tak jako stav, při kterém došlo k vyvolání alarmu.

OPC Historical Data Access
Jedná se o specifikaci definující metody pro dotazy a analýzu, které mohou být aplikovány na historická data označená časovým razítkem.

4.2.2 OPC Unified Architecture

S uvedením service-oriented architektury pro výrobní systémy přišly nové výzvy v bezpečnosti a modelování dat. OPC Foundation vyvinula specifikaci OPC Unified Architecture, která přináší výhody jako škálovatelnost a rozšiřitelnost. OPC Unified Architecture bylo

uvedeno na trh v roce 2008 a jedná se open-platform architekturu. OPC Unified Architecture je service-oriented architektura nezávislá na platformě. Integruje všechny funkce jednotlivých OPC Classic specifikací v jeden rozšiřitelný framework.

Výše uvedený text je překladem z originálního textu společnosti OPC Foundation. [7]

4.2.3 OPC Softing

Jedná se o nádstavbu nad knihovnami od společnosti OPC foundation tak, aby umožňovaly zjednodušenou práci při psaní klientské části pracující s OPC serverem. Nevýhodou těchto knihoven je ztráta určité volnosti při práci, což bylo pro nás nežádoucí.

4.2.4 Terminologie

Zde uvedu seznam pojmů, které budu ve spojitosti s OPC používat. Bez jejich znalosti není možné textu v této práci plně porozumět.

- Data Access - princip OPC, který umožňuje číst a zapisovat základní data na OPC server
- Alarm & Events - princip OPC, který slouží k monitorování stavů a událostí na OPC serveru
- OPC server - server, který obsahuje relevantní data odpovídající standardu OPC
- Subscription - seskupení itemů do jedné skupiny - motor
- Item - někdy taky označovaný jako tag je signál vycházející ze zařízení nesoucí konkrétní hodnotu - rotace = 5000rpm
- Area - oblast platnosti konkrétního eventu - hala číslo 1
- Event - událost, která nastala na OPC - nedostatek celulózy
- Alarm - varování na určitou akutní událost - hoří

Kapitola 5

Implementace back–endu aplikace

První část implementace back–endu probíhala odděleně od celého projektu. Funkcionalitu jsem psal v konzolové aplikaci, abych měl možnost nechat si vypsat výstupní data do konzole a aby se do finálního projektu nepsal neúplný kód plný poznámek ohledně poznatků k funkcionalitě. Tato část neobsahuje ani tak samotný popis implementace, jako informace o funkcích a principu fungování OPC. Rozebrání a pochopení funkcionality OPC totiž považuji za důležitější než popis samotné implementace, která by byla nicneříkající.

5.1 Architektura aplikace

DataAccess

- Framework: .NET 4.6.1, NPoco
- Jazyk: C#
- Typ výsledné komponenty: DLL
- Popis: Poskytuje komunikaci s databází včetně operací CRUD - Create, Read, Update, Delete

OpcClient

- Framework: .NET 4.6.1
- Jazyk: C#
- Typ výsledné komponenty: DLL
- Popis: Poskytuje operace nad OPC serverem, komunikaci s databází a uložiště dat pro aktuální sezení

WebApi

- Framework: .NET 4.6.1, Web API
- Jazyk: C#

- Typ výsledné komponenty: Webová aplikace
- Popis: Umožňuje přístup k datům pro jakékoliv zařízení

WebClient

- Framework/jazyk: HTML5, Bootstrap, MVC, jQuery, CSS, JavaScript, DataTable, SignalR
- Popis: Zobrazuje data získaná z OPC serveru

Service

- Framework: .NET 4.6.1
- Jazyk: C#
- Popis: Self-hosted service, udržuje OPC Clienta naživu

5.2 OPC Data Access

Jedná se o implementaci části OpcClient.

Téměř všechny funkce z knihoven od společnosti OPC Foundation umožňují zjistit, zda se požadovaná operace úspěšně podařila nebo ne. Pokud se požadovaná operace nedokončí úspěšně, tak lze podle nastaveného parametru (u void funkcí) nebo podle návratové hodnoty zjistit příčinu chyby. Mezi návratové hodnoty, na které jsem během implementace back–endu a to jak OPC Data Access, tak i Alarm & Events části aplikace narazil, patří:

- S_OK - požadovaná operace byla úspěšně dokončena bez jakékoliv chyby
- E_FAIL - požadovaná operace nebyla úspěšně dokončena - může nastat v případě odpojení serveru
- E_INVALIDARG - neznámý argument - nastane, pokud OPC serveru předáme argument, který pro něj není validní
- OPC_E_UNKNOWNITEMID - pokud budeme pracovat s Data Access Itemem, který se již na serveru nenachází
- OPC_E_INVALIDITEMID - pokud budeme pracovat s Data Access Itemem, který se na serveru nikdy nenacházel
- OPC_E_RANGE - může nastat při pokusu o zápis hodnoty, která je mimo povolený rozsah. Například zápis hodnoty do Itemu s typem Int8, která přesahuje jeho maximální velikost
- OPC_E_BADTYPE - pokus o zápis hodnoty, která je jiného typu než jaký očekává Item
- OPC_E_BADRIGHTS - snaha o zapsání do Itemu, ke kterému nemáme dostatečná oprávnění. Například zápis hodnoty do Itemu, který je pouze ke čtení (na serveru označován jako ReadOnly)

Jsou zde i další návratové hodnoty, ale ty jsem během implementace nezaznamenal, takže nevím, za jaké situace je možné takovou návratovou hodnotu dostat. Dokumentace k OPC Data Access [5] a OPC Alarm & Events [3] popisuje pouze obecně význam těchto návratových hodnot, ale již nepopisuje konkrétní situaci, kdy k takové návratové hodnotě může dojít. Výše zmíněné návratové hodnoty jsem dostával při implementaci OPC Data Access části, při OPC Alarm & Events části jsem narazil pouze na S_OK a E_FAIL. Dokumentace k OPC Data Access a OPC Alarm & Events je po přihlášení dostupná ke stáhnutí na URL uvedené v citaci.

Prvotní komplikace

Na počátku implementace back–endu naší aplikace jsem narazil na problém s dostupností OPC serveru. Aby aplikace mohla být nasazena v praxi, musí být také tak vyvíjena. Jelikož dostupnost běžně používaného OPC serveru pro vývoj nebyla reálná, museli jsme se spokojit s emulátorem OPC serveru. Emulovaný OPC server se v počítači váže na DCOM. Prvotním plánem bylo použití a přeložení zdrojových kódů od OPC Foundation s cílem vytvořit lokální emulátor OPC serveru. I přes značnou snahu se mi toho nepodařilo docílit, konzultoval jsem tedy problém s Michaelem Filsákem ze společnosti ABB. Ten mi doporučil použití emulátoru OPC serveru od společnosti Softing.

Osvojení si principů práce s OPC

Emulovaný server od společnosti Softing se podařilo úspěšně zprovoznit, takže následovala fáze zjištění funkčnosti serveru a jak se s ním dá komunikovat pomocí kódu napsaného v C#. Pro pochopení principu fungování a psaní kódu, který má spolupracovat s OPC serverem, bylo nutné prostudovat zdrojové kódy ukázkového OPC klienta od OPC Foundation. Jelikož klient obsahoval funkce, které se na emulovaném serveru nedaly vyzkoušet, bylo zkoumání o to těžší, protože nebylo předem známo, zda je chyba na straně ukázkového klienta nebo emulovaného serveru. Jako zdroj informací se nedaly použít ani fóra na internetu (StackOverflow) ani jiná dostupná literatura, protože když už obsahovala něco ohledně OPC, byl to spíše popis jeho funkcionality a v tom lepším případě i úseky kódu, ale žádný mně nebyl prospěšný. Požádal jsem tedy ABB o dokumentaci k Data Access OPC. Tato dokumentace není na oficiálních stránkách lehce dostupná. Po jejím prostudování musím konstatovat, že je dobře napsaná, ale příliš obecně. Nastala tedy fáze vývoje pokus-omyl, kdy jsem musel zkoušet různé varianty parametrů v metodách, abych docílil kýženého výsledku. To bylo vzhledem k nedostatečné znalosti principu OPC a minimálnímu množství literatury značně komplikované. Tato část vývoje byla nejdelší ze všech.

OPC Data Access server

Server je základem celého OpcClienta, proto bylo nejdůležitější částí jeho zprovoznění a připojení. Odehrává se na něm vše, co je pro nás důležité. Obsahuje totiž subscription včetně itemů, se kterými se v aplikaci dále pracuje. První částí vývoje bylo samotné připojení na server, který simuloval práci v režimu Data Access. Připojení zajišťuje funkce connect z knihovny OPC. Tato funkce je volána nad serverem, který byl vytvořen podle zadané IP adresy. U IP adresy se musí kontrolovat její validnost a zda se jedná o adresu k OPC Data Access serveru. Pokud některý požadavek není splněn, dojde k vyvolání výjimky. Adresa je ve formátu opcAA://BB/CC/XXXX, kde AA je identifikátor serveru, např. opcda nebo

opcae, BB je adresa v síti - při vývoji se používala adresa localhost, CC je libovolný název serveru, XXXX je COM/DCOM handler. BB a CC je možné vynechat, v takovém případě je server na localhost a bez pojmenování. Nejdůležitější parametr adresy je DCOM handler, pomocí kterého se připojíme k serveru, ostatní parametry jsou pouze pomocné. Příkladem validní adresy pro Data Access server je:
`opcda://localhost/DemoDa/{2E565242-B238-11D3-842D-0008C779D775}`
jde o adresu Demo Serveru od společnosti Softing. Po připojení na server je možné z něj získat základní údaje jako aktuální čas, čas spuštění serveru, čas poslední aktualizace, stav serveru (běží, neběží, pozastaven, ...). Jednotlivé servery, které jsou připojeny, se uloží do listu Data Access serveru (nutné pro práci na front–endu).

OPC Data Access Subscription

Další částí vývoje bylo vytvoření subscription na serveru. Jelikož se subscription vytváří na serveru, tak je nutné, aby server byl připojen. K tomuto účelu slouží parametr Connected, který indikuje, zda je server připojen či odpojen. Na serveru je náročné vyhledávat informace o subscription, proto byl zaveden List subscriptions, do kterého se ukládá každá vytvořená subscription, včetně informací jako je Name a nadřazený server. List subscription je poté využíván aplikací na front–endu. Při vytváření subscription je potřeba znát význam jednotlivých parametrů. Základním parametrem je Name. Jedná se o naše pojmenování subscription, tento parametr je volitelný. Pokud Name nezadáme, tak OPC server vygeneruje náhodné jméno, které potom přiřadí. Tato vlastnost je v naší aplikaci zakázána a Name je povinný parametr při vytváření subscription. Nejdůležitější parametr je Update Rate, který určuje, jak často se budou obnovovat data v subscription. Parametr se zadává v ms. Demo servery, které se nám podařilo získat. Mají jednu zajímavost: pokud vytvořím dvě subscription s rozdílným update rate (pro ukázku subscription1 100ms a subscription2 10 000ms) a budou obsahovat stejné dynamicky generované itemy, tak se hodnota generování řídí podle update rate rychlejší subscription. Např. výchozí hodnota je 0, subscription1 bude ukazovat v čase 100ms hodnotu 1, 200ms hodnotu 2, ... subscription2 nám ukáže v čase 10s hodnotu 100, ostatní časy bude hodnota neustále 0. Tento princip nemám možnost ověřit v praxi, takže se musíme řídit tím, že to OPC server tak zpracovává. Z logického hlediska je to nepochopitelné, každý item na serveru by měl mít vlastní čas, po jehož uplynutí bude generovat hodnotu, a neřídit se podle subscription. Tímto může totiž dojít k rychlému zahlcení serveru a následnému pádu. Počet subscription není z naší strany nijak limitován.

OPC Data Access Item

Poslední částí vývoje u OPC Data Access je práce s itemy. Jedná se o nejnižší článek v hierarchii objektů v OPC Data Access. Item jako takový náleží serveru, ale protože na serveru mohou být i tisíce itemů a pracovního technika zajímají jen itemy poskytující pro něj relevantní informace, přidávají se itemy do subscription. Knihovny od OPC Foundation sice obsahují prohledávání serveru, ale pouze v jedné úrovni. Pro naši aplikaci jsme implementovali vlastní funkci, která udělá ze všech itemů na serveru stromovu strukturu, a druhou funkci, která obsahuje názvy všech itemů včetně jejich cesty ve stromě. Item jako takový obsahuje následující položky:

- dataType - určuje jakého datového typu je hodnota

- value - konkrétní hodnota itemu
- quality - nese informaci o kvalitě itemu
- timestamp - časové razítko
- accessRights - určuje operace, které lze s itemem provádět: čtení/zápis/čtení + zápis
- scanRate - čas v ms, kdy server zjišťuje hodnotu

Pro náš účel jsou nejdůležitější položky dataType a value. DataType je specifická položka v tom, že určuje datový typ value, například boolean, integer, string, ... Zajímavostí je, že obsahuje tzv. OpcType, který je ekvivalentem významu libovolný typ, do položky s dataType = OpcType je tedy možné podle dokumentace zapsat boolean(True, False), string(Demo Value), integer(100), dateTime(13.5.2016 18:12:55), ... Bohužel toto se nepovedlo potvrdit u itemu, který měl uvedený typ OpcType a value = 1 nebo libovolné číslo, při pokusu o zápis stringu se hodnota do value nezapsala.

5.3 Databáze

Jde o implementaci části DataAccess.

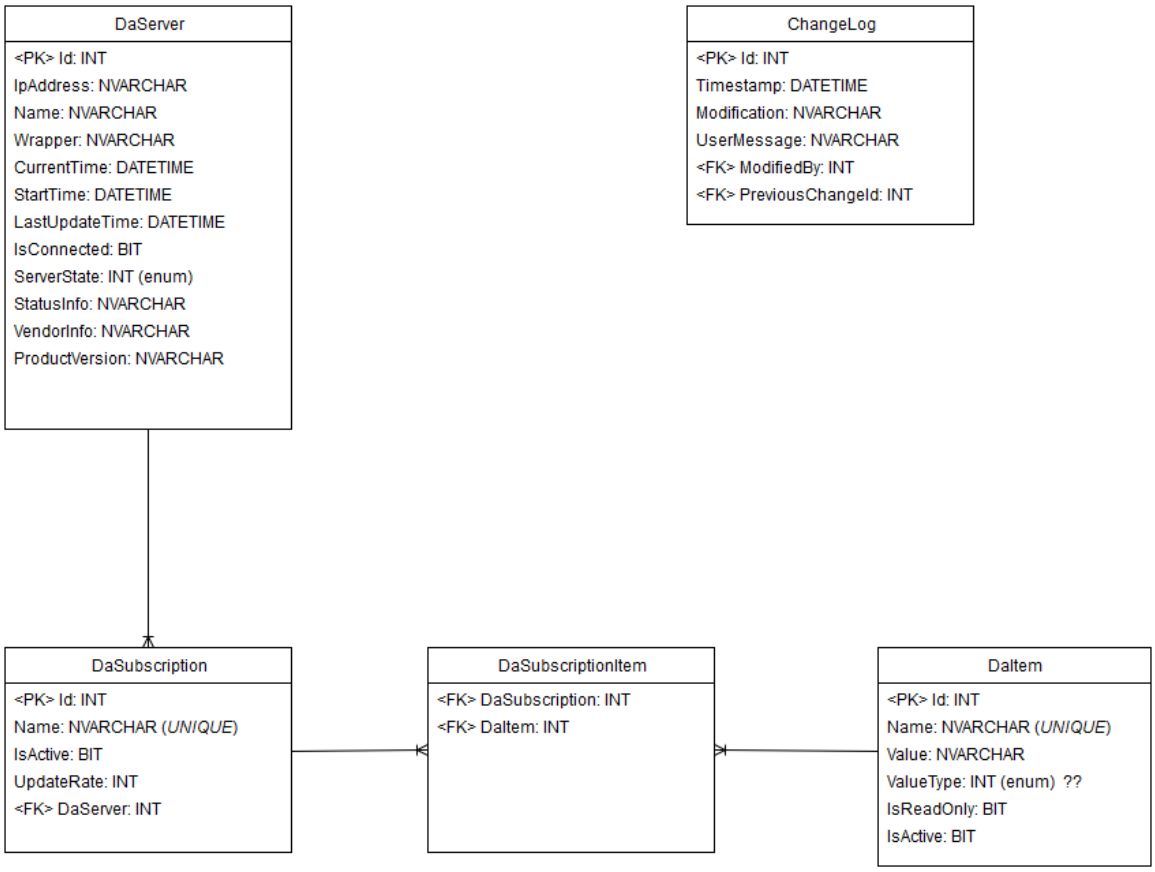
Vzhledem k velkému množství dat, která se posílají z OPC serveru, bylo nutné do návrhu začlenit databázi. Samotný OPC server totiž není navržen k tak velké zátěži, jakou bude představovat naše aplikace. OPC server je konstruován k posílání dat na jedno místo, kde se data skladují a v případě potřeby se posílají operátorům na displej. Vzhledem k podstatě naší aplikace, kdy je současně připojeno předem neznámé množství zařízení (z bezpečnostních důvodu není možné maximální počet limitovat), bylo potřeba vyřešit problém se zátěží na straně OPC serveru. Jediná přípustná varianta je databáze. Po provedení návrhu databáze jsme zjistili, že je vhodné využít některý z Frameworků pro práci s databází. První ideou byl EntityFramework, bohužel pro takové množství dat, jaké představuje OPC server, je příliš pomalý, tudíž nepoužitelný. Volba padla na framework Telerik. Aby nedocházelo k vícenásobné instanciaci, je u databáze použit návrhový vzor Singleton.

Návrh

Samotný návrh databáze procházel postupným vývojem, Počáteční návrh byl totiž nedostatečný a v průběhu implementace se rozšiřoval podle zjištěných skutečností až do stávající podoby, která je zobrazena na obrázcích 5.1, 5.2, 5.3.

5.3.1 Telerik

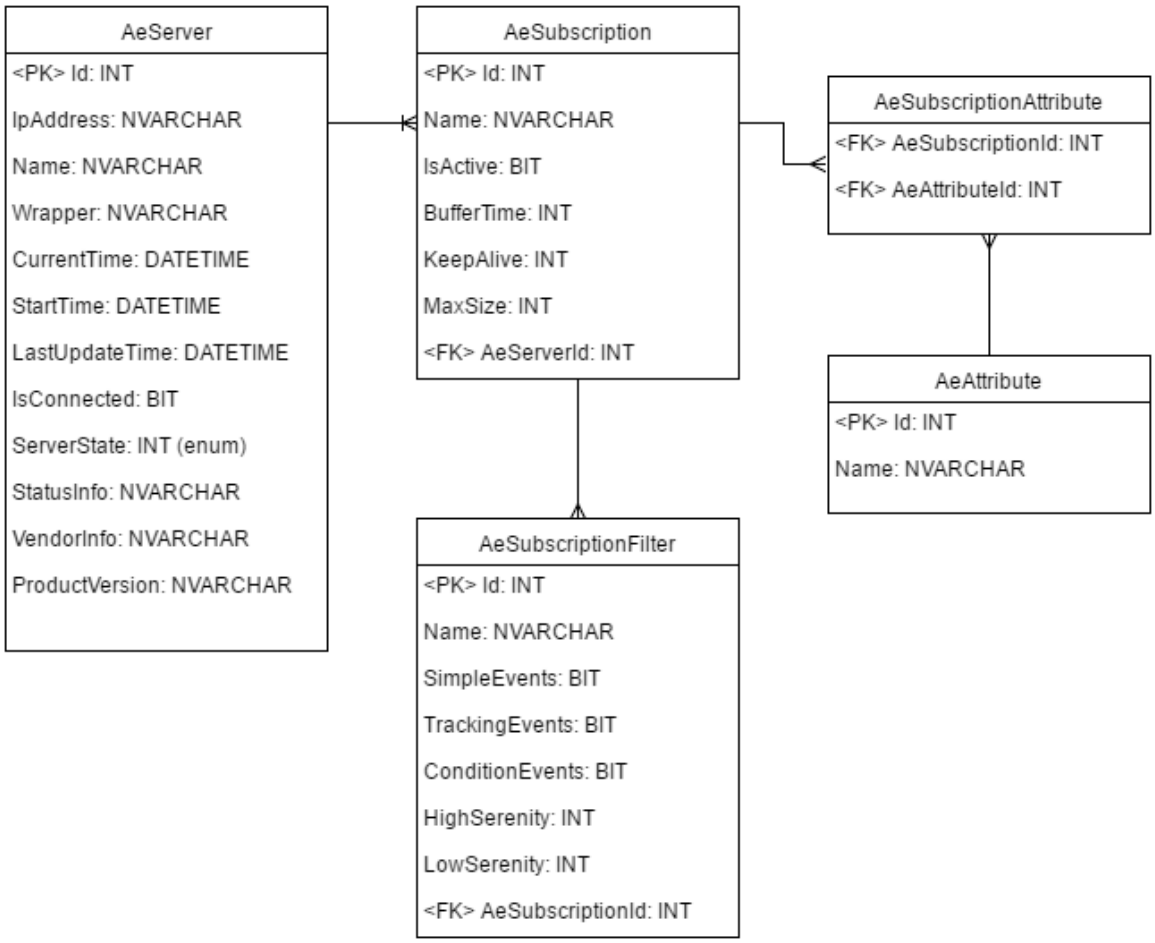
Telerik splňoval všechny naše požadavky a umožňoval použití metody code–first pro mapování databáze. Rozhodli jsme se tedy, že jej do aplikace nasadíme a budeme jej využívat. První problém nastal hned na počátku. Telerik požadoval, aby modely pro code–first mapování byly umístěny ve stejném projektu jako je Telerik. Jelikož naše modely byly v projektu Shared, museli jsme je duplikovat do projektu DataAccess. Aby bylo možné komunikovat mezi DataAccess a ostatními projekty, vytvořili jsme konvertory, které zajistily kompatibilitu napříč aplikací. Další problém nastal v pozdní fázi vývoje, kdy jsme se pokusili propojit DataAccess a WebClient pomocí SignalR. Telerik obsahuje cache paměť první (L1) a druhé (L2) úrovně. Jelikož je Telerik ORM, tak je ve výchozím nastavení L2 vypnutá, ale L1 nelze



Obrázek 5.1: Entity Relationship Diagram pro OPC Data Access

vypnout. Toto způsobilo velké problémy, kdy se data ukládala do L1 a i přes update v databázi se zobrazovala data z cache paměti. Toto chování je nežádoucí, protože aplikace musí zobrazovat aktuální data v reálném čase. Toto chování bylo způsobeno použitím dvou druhů komunikace, kdy jeden využíval komunikace přes HTTP protokol a druhý využíval přímé volání metody z DataAccess. Částečným řešením bylo ponechání pouze jednoho druhu komunikace, v našem případě HTTP. Poslední problém s Telerikem nastal při implementaci autentifikace a autorizace uživatelů. Ve výchozím stavu využíváme základní ASP.NET autorizaci a autentizaci, ale chceme mít databázi pod naším dohledem. Rozhodli jsme se tedy přidat jednotlivé modely, které ASP.NET využívá, a nahradit původní Entity framework za Telerik. Toto se později ukázalo jako největší problém, ke kterému jsme nenalezli řešení. Působením více faktorů se stávalo, že Entity framework předběhl Telerik a začal mapovat databázi pro uživatele místo Teleriku. Telerik toto zjistil a poslal výjimku, která způsobila pád aplikace. Po dlouhém hledání jsme nalezli příčinu. Problém způsobila kombinace IIS, MVC.NET a ASP.NET autentizace a autorizace. IIS i po ukončení aplikace měl v cache paměti uložený StartUp z našeho WebClienta a ignoroval přepsání Entity na Telerik. Tento problém měl jen dvě řešení:

- přepsat logiku ASP.NET
- použít jiný framework



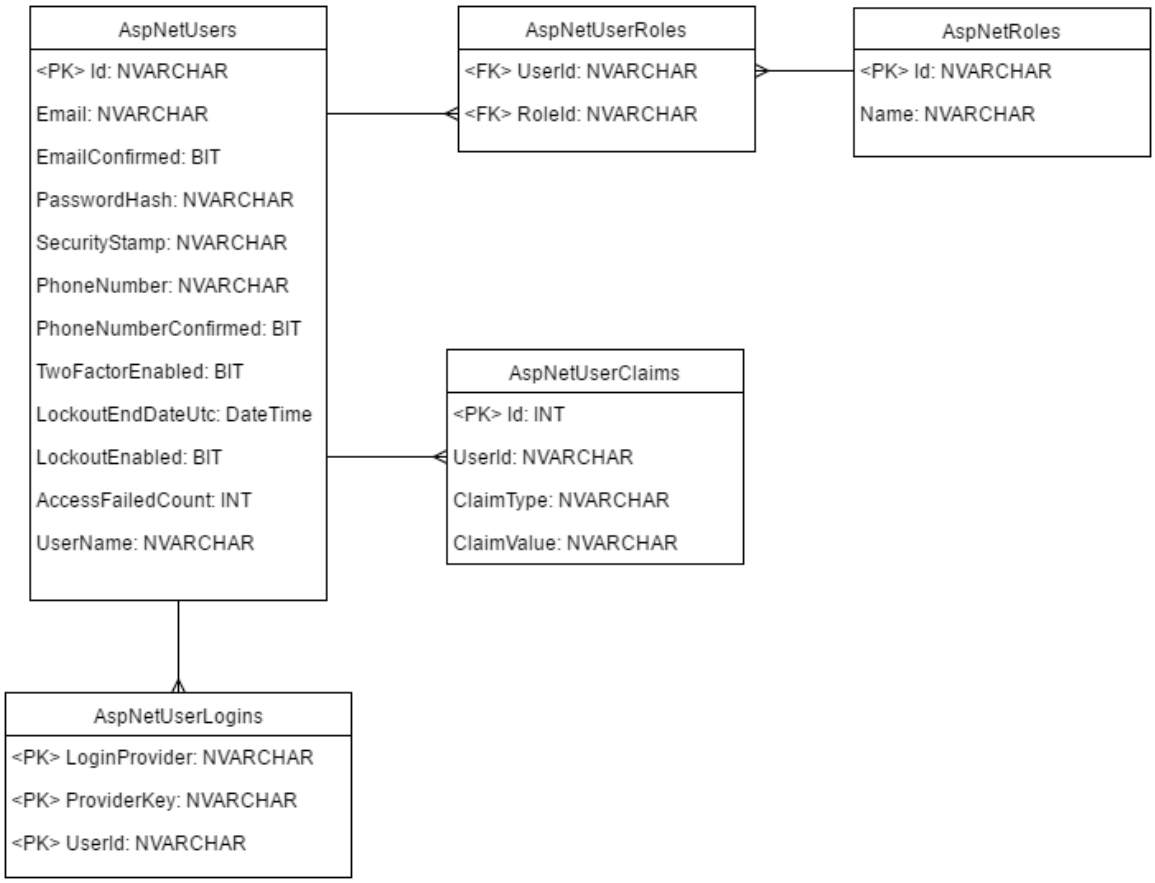
Obrázek 5.2: Entity Relationship Diagram pro OPC Alarm & Events

Zvolili jsme druhou variantu.

5.3.2 NPoco

Framework NPoco je odnoží známějšího PetaPoco. Jelikož NPoco není ORM, ale micro ORM, tak nám dává větší zodpovědnost za funkce. Toto se ukázalo jako výhoda. Vyřešil se problém s cache pamětí, protože Micro ORM nemá vrstvu, která by držela data v cache. Vyřešil se i problém s ASP.NET. Cenou za tato řešení bylo, že jsme se vzdali code-first mapování a musíme databázi inicializovat ručně pomocí SQL příkazů. SQL scripty namapují databázi podle ER diagramů 5.1, 5.2, 5.3. Vzhledem k rozdílu mezi Telerik (ORM) a NPoco (Micro ORM) není možné použít stejné databázové modely, a jelikož NPoco neumožňuje dynamické generování POCO modelů tak jako PetaPoco, je nutné modely poupravit pro použití s NPoco. Tyto modely jsou doplněny o atributy, které definuje NPoco:

- TableName - určuje jméno tabulky v databázi
- PrimaryKey - definuje, který atribut je primární klíč
- Column - tímto parametrem řekneme, pod jakým názvem má hledat sloupec v databázi



Obrázek 5.3: Entity Relationship Diagram pro ASP.NET

- Ignore - pokud se v modelu nachází atributy, které v dabázi nejsou, můžeme je díky tomuto ignorovat
- ResultColumn - tímto parametrem označujeme atributy, které nebudou zahrnuty při vkládání a aktualizaci dat v databázi
- ComputedColumn - stejný význam jako ResultColumn s tím rozdílem, že bude zahrnut v generovaném SQL
- SerializedColumn - takto označený atribut bude serializován výchozím serializérem

TableName je použit pro korekci jména tabulky, např.
`public class DaItemDatabase {[TableName(„DaItem")]}`
Bez definování tohoto parametru očekává NPoco v databázi tabulku se stejným jménem jako je název třídy, která je použita v dotazu. Příklad dotazu, který vytáhne všechna data z tabulky DaItem: var result = DbContext.Instance.Fetch<DaItemDatabase>(); Pokud bychom nechtěli použít TableName, je potřeba formulovat SQL dotaz typu SELECT * FROM TableName, kde TableName je název tabulky, aby NPoco vyhledávalo ve správné tabulce.

5.4 Alarm & Events

Jedná se o další implementaci části OpcClient. Tato část nebyla vyvíjena současně s OPC Data Access, protože společnost ABB požadovala pořadí OPC Data Access, Databáze, OPC Alarm & Events.

Očekával jsem, že implementace OPC Alarm & Events části bude hodně podobná již naimplementované OPC Data Access části. Toto se bohužel nenaplnilo a všechna úskalí, která nastala při implementaci OPC Data Access, se opakovala. Poučen předchozími komplikacemi jsem si hned na začátku požádal o dokumentaci k OPC Alarm & Events a doufal, že tato dokumentace bude funkce popisovat méně obecně než dokumentace k OPC Data Access. Bohužel tomu tak nebylo.

Jelikož naše aplikace obsahuje na back–endu implementované funkce pro Alarm & Events, ale tyto funkce nejsou napojeny na front–end, budu používat pro ilustraci obrázky z Demo Klienta od OPC Foundation, kterého jsem zkompiloval z dostupných zdrojových kódů.

OPC Alarm & Events Server

OPC Alarm & Events Server je hodně podobný OPC Data Access serveru, a to jak v principu připojení, tak v práci s ním. Naše aplikace využívá pro připojení k OPC Alarm & Events Serveru IP Adresu:
`opcae://localhost/DemoAe/{2E565243-B238-11D3-842D-0008C779D775}`
Aby byla IP adresa považována za validní a bylo možné se na ni připojit v režimu Alarm & Events, je potřeba, aby opět splňovala požadované vlastnosti. Prvním požadavkem je identifikátor opcda/opcae, tato adresa obsahuje identifikátor opcae, takže první podmínka pro připojení je splněna. Další podmínkou je DCOM/COM handler pro systém Windows, handler {2E565243-B238-11D3-842D-0008C779D775} odkazuje na demo server od společnosti Softing. Ostatní části IP adresy jsou volitelné. Pokud to tedy shrnu, jde o IP adresu odkazující na OPC Alarm & Events server umístěný na IP localhost (127.0.0.1) s názvem DemoAe, vázaným na COM/DCOM handler {2E565243-B238-11D3-842D-0008C779D775} v systému.

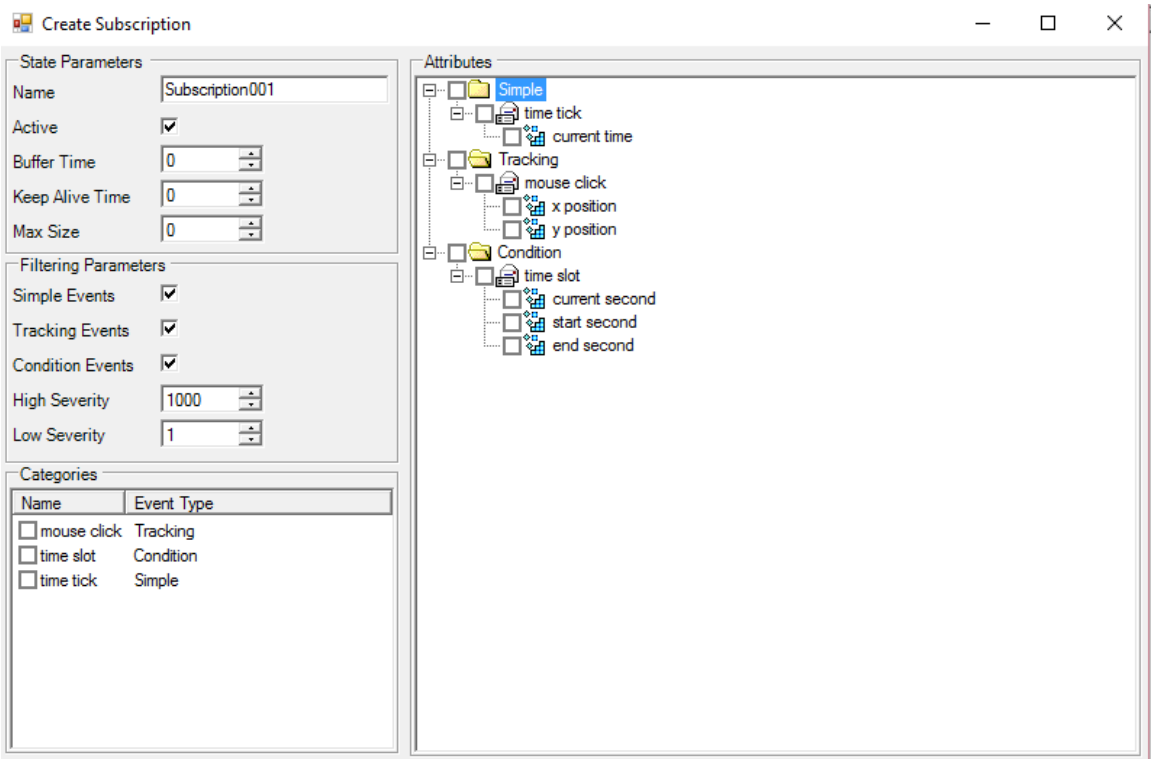
Připojení na server probíhá opět funkcí connect z knihovny od OPC Foundation. Bez úspěšného připojení na server není možné s ním dále pracovat, proto se při připojení nastaví parametr connected, podle kterého se v aplikaci dále orientujeme a zjišťujeme, zda je server připojen a můžeme s ním tedy pracovat. Funkce connect je obalena blokem Try-Catch, aby při zadání nevalidní IP adresy aplikace nespadla, ale pouze nastavila parametr connected na false. Každý server, na který se pokoušíme připojit, a to ať je výsledek úspěšný nebo neúspěšný, se uloží do Listu serverů pro výpis uložených serverů na front–endu.

OPC Alarm & Events Subscription

Obrázky jsou zde přiloženy pro lepší představu o níže zmíněném vysvětlení principu práce se subscription.

Subscription pod OPC Alarm & Events Serverem požaduje následující parametry pro vytvoření:

- Name - interní pojmenování subscription pro lepší orientaci během práce.
- Active - určuje, zda je subscription aktivní nebo neaktivní. Pokud je subscription



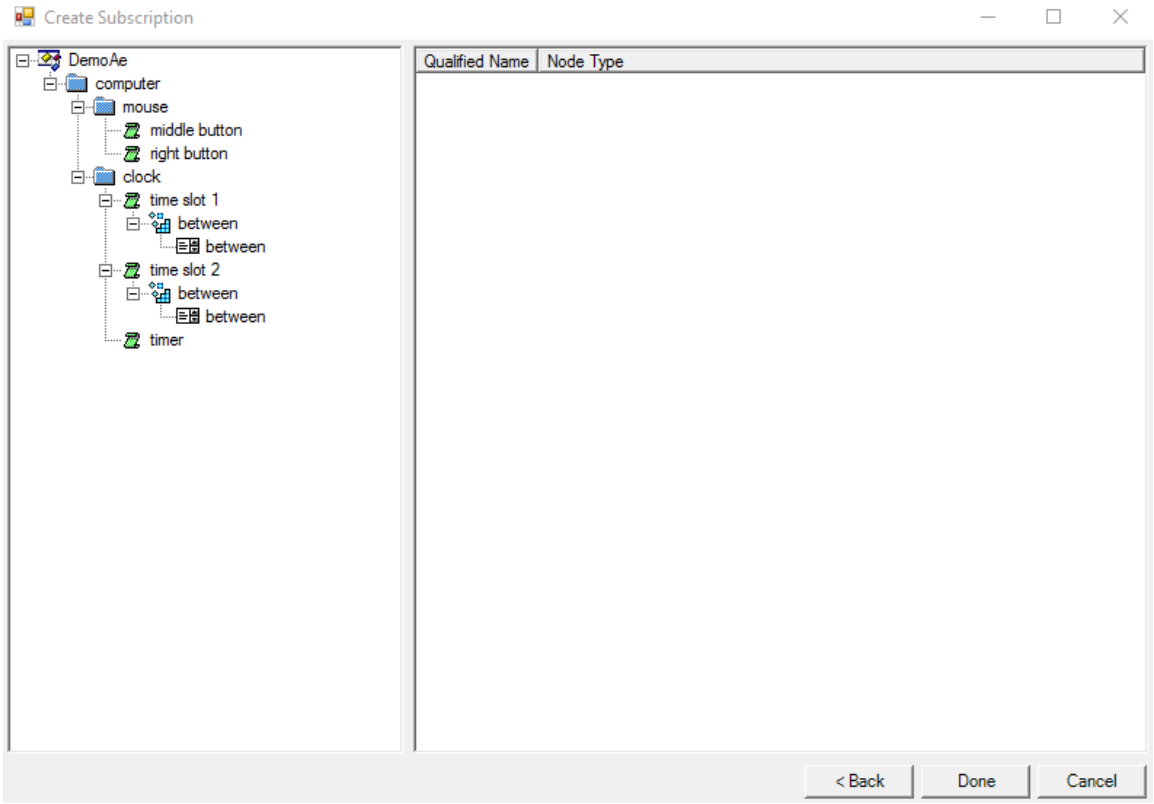
Obrázek 5.4: Grafické zobrazení přidání subscription v klientovi od OPC Foundation - nastavení

neaktivní, je vedena v seznamu dostupných subscirption, ale její alarmy a události nejsou zahrnuty.

- BufferTime - je uveden v ms a udává, jak často bude server posílat notifikace o událostech. Při nastavení na hodnotu 0 bude server posílat notifikaci, jakmile nastane událost na serveru.
- KeepAliveTime - zadává se v ms a určuje, po jaké době posílat notifikaci o událostech na serveru, pokud mezitím nenastaly události nové. Po uplynutí tohoto času se tedy pošle notifikace s posledními událostmi na serveru. Částečně slouží ke zjištění stavu serveru. Pokud budou chodit notifikace, znamená to, že server běží a je připojen, jen nenastaly žádné nové události.
- MaxSize - určuje, kolik událostí může být posíláno v jednom Callbacku. Pokud je hodnota nastavena na 0, znamená to, že není žádné limitní omezení.

Subscription je vytvářena pomocí funkce CreateSubscription, která očekává výše zmíněné parametry.

Po vytvoření subscription je potřeba nastavit všechny parametry pro získávání relevantních informací ze serveru. Protože takto vytvořená subscription nemá žádné informace o tom, co uživatele zajímá, a proto nic neposílá. Dalším krokem pro nastavení subscription je nastavení toho, co nás ze serveru zajímá. Pro lehčí pochopení budu používat informace z vložených obrázků 5.4, 5.5, 5.6.

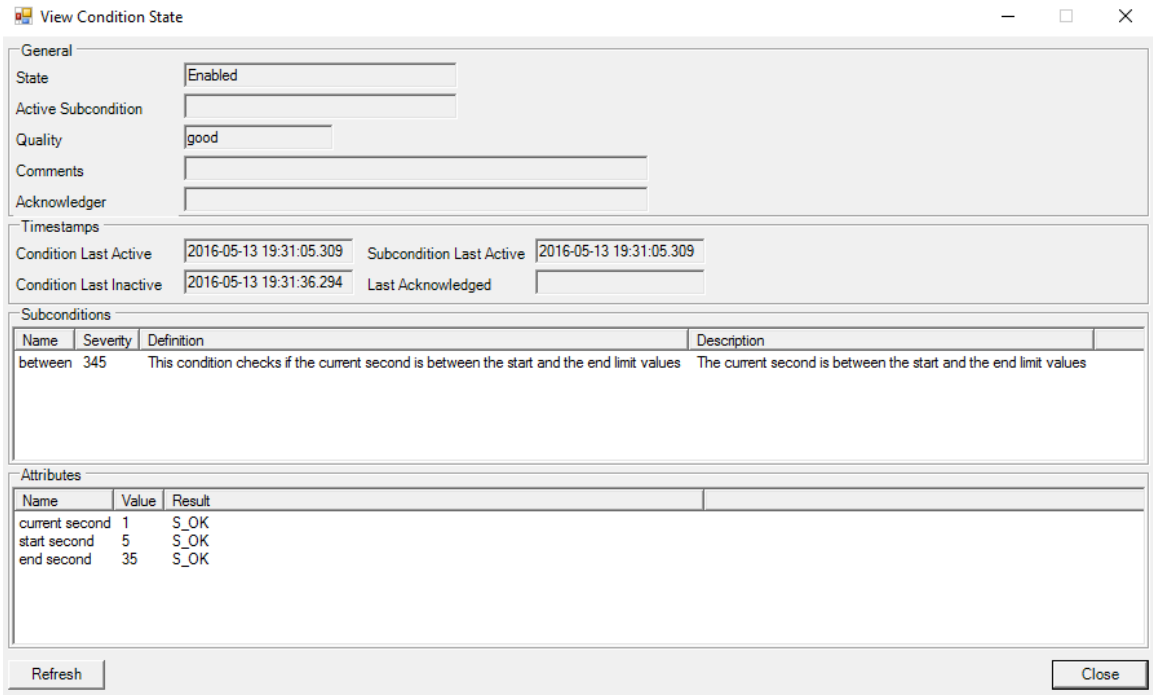


Obrázek 5.5: Grafické zobrazení přidání subscription v klientovi od OPC Foundation - výběr oblasti a zdroje

Prvním krokem po vytvoření subscription je aplikace filtrů na ni. Subscription umožňuje tři typy událostí: Simple, Tracking, Condition. Zvolíme tedy ty typy událostí, které požadujeme, aby aplikace pod danou subscirption zobrazovala. U filtrů je ještě možné nastavit Severity, jde o parametr určující důležitost události. Čím vyšší číslo, tím důležitější událost, můžeme tedy odfiltrovat události, které mají pro pozici konkrétního pracovníka příliš nízkou nebo naopak vysokou důležitost. Alespoň jeden typ události musí být vybrán, jinak aplikace zahlásí chybu. Nebylo by totiž podle čeho vypisovat události, které nastaly na serveru.

Po nastavení filtrů je potřeba nastavit atributy umístěné na serveru. Pro zobrazení atributů na front–endu naší aplikace je připravena funkce, která prohledá server a atributy uloží do stromové struktury. Atributy jsou rozděleny podle událostí a kategorií podle toho, kam spadají. Na rozdíl od filtrů je možné nechat výběr atributů prázdný, v takovém případě ale nebudou vypisovány žádné události. Výběrem atributu určujeme konkrétní věc, která nás na serveru zajímá, např. při výběru current time nás zajímá událost změny času, tzn. při každé změně času (v řádu sekund) se nám objeví notifikace o události na serveru. Jelikož server většinou neobsahuje pouze 6 atributů, ale stovky až tisíce, tak jednotlivý výběr by byl časově příliš náročný. Proto je zde ještě seznam kategorií a je možné pomocí nich vybrat všechny atributy, které pod danou kategorii spadají.

V další části je možné vybrat oblast, pod kterou nás zajímají zvolené atributy. Na demo serveru se nachází oblast Computer s podoblastmi mouse a clock. Jako zdroje události obsa-



Obrázek 5.6: Grafické zobrazení přidání subscription v klientovi od OPC Foundation - výpis condition state

huje mouse: middle button a right button. Clock má jako zdroje události: time slot 1, time slot 2 a timer. Zdroje time slot 1 a time slot 2 obsahují ještě podmínku between(condition), která určuje, za jaké podmínky bude událost vyvolána. Podmínka between obsahuje informace:

- State - obsahuje informaci o stavu podmínky, může nabývat hodnot Active, Enabled, Active AND Enabled
- Active Subscondition - informuje o podmínkách, které náleží pod aktuální podmínku
- Quality - informuje o kvalitě dané podmínky
- Comments - volitelný komentář, který je nastaven na serveru administrátorem
- Acknowledger - jde o identifikátor posledního oznamovatele této podmínky
- TimeStamp - obsahuje informace o časových razítkách poslední aktivity a neaktivity podmínky a časové razítko poslední aktivity podpodmínky. Může obsahovat i informaci o posledním oznámení od oznamovatele.
- Subconditions - obsahuje informaci o podpodmínce, jako je jméno, důležitost a textová definice
- Attributes - informuje o atributech, které se vážou k dané podmínce, včetně jména aktuální hodnoty a výsledku, který se vrátil při položení dotazu

Při výběru oblasti jsem zjistil problém, který podle mého uvážení nemá žádné opodstatnění, a jedná se proto o chybu v knihovnách od OPC Foundation. Pokud se vybere nejvyšší uzel, zde je to computer, tak nejsou registrovány události. A to i přestože zdroje události obsahují plnou cestu ve stromové struktuře. Pokud se například vybere poduzel clock, tak se již události registrují bez problému.

Jako ukázku nastavíme subcription takto:

- Parametry subscription : Active, BufferTime = 0, KeepAliveTIme = 0, MaxSize = 0
- Parametry filtru : Simple Events, High Severity = 1000, Low Severity = 1
- Kategorie : time tick, toto nám vybere atribut current time
- Jako oblast vybereme computer.clock

Výstupem takového nastavení je výpis událostí viz. obrX. Tento výpis obsahuje čas, kdy událost nastala, její důležitost, zdroj události, potřebu potvrzení, podmínku, za jaké událost nastala, a zprávu obsahující popis události.

Ostatní součásti nutné pro chod aplikace

Všechny velké součásti aplikace, které plní nejdůležitější roli, jsem již popsal, ale v aplikaci se nachází několik menších součástí. Tyto součásti sice nejsou nejdůležitější, ale jsou nezbytné pro chod aplikace.

Bylo potřeba vytvořit **HTTP komunikaci** mezi front–endem a back–endem. Tato komunikace je realizována pomocí protokolu HTTP, v současné podobě využíváme převážně metody POST. Metodu GET jsme používali v počátcích vývoje, ale později se ukázalo, že je potřeba přenášet doplňující informace z front–endu na back–end a k tomu je GET nedostačující. Komunikace je řešena jako asynchronní, tzn. nečeká se na její dokončení. Asynchronnost byla zavedena, aby nedocházelo k blokování samotného programu v případech, kdy dojde k pádu, přerušení nebo zpomalení komunikační trasy. Pro posílání objektů je využita serializace do formátu JSON.

Základní posílání objektů se ukázalo jako nedostačující, a proto byl zaveden náš vlastní datový typ **ApiResult**. ApiResult je používán k posílání odpovědi ze strany back–endu na front–end. S objekty, které jsme používali doposud, nebylo totiž možné posílat informaci, zda se požadovaná operace podařila a pokud ne, tak co bylo příčinou. Jelikož se ApiResult skládá z chybové zprávy, informaci o úspěchu a samotného výsledku operace, tak je možné tyto informace již přenášet. Pro opačný proces, tedy posílání více informací než stávající objekt, je využit **ApiInput**. Jak ApiResult, tak ApiInput využívají pro přenos generické parametry.

Pro typovou kontrolu a bezpečnost jsme využili **Interface**. Každý objekt a některé třídy v Shared mají svůj vlastní interface a je zde jeden nadřazený interface IOpcObject. IOpcObject zaštituje všechny objekty, které mají co do činění s OpcClient. Interface nám navíc poskytl možnost jak kontrolovat shodnost objektů a metod na různých místech aplikace. Stačí pouze změnit interface a všechny objekty či třídy, které z něj dědí, zaznamenají tuto změnu a přinutí nás opravit je. Pokud bychom nepoužili interface, zjistíme problém až při spuštění aplikace a provedení akce, která by objevila nekonzistentnost mezi různými částmi aplikace a zahlásila chybu, případně i spadla.

5.5 Testování

Testování je důležitou součástí každé dobře napsané aplikace. Pomocí testů je možné odhalit chyby, které se nemusí jevit na první pohled. Chyba v kódu, která není odhalena během vývoje, může znamenat pro společnost využívající naši aplikaci velké problémy. Například v případě nenahlášení požáru dojde k ohrožení bezpečnosti pracovníků. Navíc každá chyba v aplikaci, jenž se používá v praxi, nese s sebou velké finanční náklady pro společnost zajišťující její vývoj a údržbu.

Naše aplikace má pro testování front–endu napsané testy pro Selenium a NUnity. K otestování back–endu jsme je zatím nemohli použít, protože až v další fázi vývoje dochází k odstranění statických proměnných a metod. Statické proměnné a metody znemožňují použití testovacích nástrojů, neumožňují totiž mock–up. Back–end byl proto testován ručně během vývoje. Testování probíhalo jak pomocí posílání dotazů z front–endu, tak také pomocí oddělení částí do konzolového projektu a posíláním vstupů s kontrolou výstupu v konzoli. Testování probíhalo pro každou část zvlášť a poté jako celek. Testování probíhalo v tomto pořadí:

- OPC Data Access
- OPC Alarm & Events
- OPC Data Access + Databáze
- OPC Alarm & Events + Databáze
- celý back–end pomocí vstupů z front–endu

Vstupy, které se testovaly, byly rozděleny na 3 části: **validní**, **validní, ale nepoužitelné v daném kontextu** a **nevalidní**.

Při samotném testování jsem zjistil několik kritických chyb, které bylo nutné opravit před dalším vývojem. Mezi nejzávažnějšími chybami byly:

„Vyhození“ neošetřené výjimky při pokusu vytvořit Alarm & Events server s nevalidní IP adresou, tím je myšlena IP adresa, která neobsahuje identifikátor opcda/opcae. Zajímavé bylo, že OPC Data Access výjimku „nevyhodilo“. Při bližším zkoumání jsem zjistil, že knihovna od OPC Foundation klasifikuje každou IP adresu bez identifikátoru jako adresu označenou identifikátorem opcda. To byl důvod, proč se OPC Data Access pouze nepřipojil a OPC Alarm & Events „vyhodilo“ výjimku.

Na základě minulého testu jsem ověřoval pád aplikace při zaslání validní IP adresy, ale v jiném kontextu, tzn. IP adresu pro Data Access server jsem poslal Alarm & Events serveru. Výjimka nastala a chyba byla v ošetření na straně knihoven OPC, kde knihovna od OPC Foundation zjistila, že jde o opcda adresu, ale přesto povolila připojení jako na Alarm & Events server.

Obě výše zmíněné chyby jsou ošetřeny pomocí bloku Try-Catch, pro ošetření na základě kontoly IP adresy totiž nemám dostatečné informace o tom, jaké IP adresy se budou používat. Konzultanti z ABB odsouhlasili použití bloku Try-Catch, připouští totiž i možnost IP adresy bez identifikátoru opcda/opcae. Pokud tedy nebude použit identifikátor, tak ze strany našeho klienta není možné zjistit, o jaký typ serveru se jedná. Nabízí se pár možností, jak toto vyřesit, a z mého pohledu nejrozumnější možností je použití nějakého parametru v URL, který bude udávat typ serveru. Nejméně výhodnou možností je ta, že zákazník oznámí adresu, na které jede OPC Alarm & Events a na které OPC Data Access server. Pro tento případ by se musela implementovat možnost ručního zadávání typu serveru.

V následujících testech jsem objevil ještě několik chyb, ty už ovšem nebyly tak závažné jako výše zmíněné. Jednalo se spíše o chyby v logice, kdy jsem měl například přehozené bitové operace AND a OR. Tato chyba by nezpůsobila pád aplikace, ale její chování nebylo korektní.

Testování databáze bylo zvláštní kapitolou, nejednalo se totiž o samotnou databázi, ale o práci databáze v určitém kontextu. Databáze se testovala pro OPC Data Access, OPC Alarm & Events, ASP.NET. U ASP.NET jsme neměli možnost testovat všechny části. Naše aplikace totiž používá u uživatele pouze email a password, dále pak 2 uživatelské role - user a administrator. Během testování databáze se kontrolovala konzistence databáze, výstup z dotazu na databázi se porovnával s očekávaným výstupem a kontrolovaly se vazby mezi tabulkami včetně správných primárních klíčů. Tady se nezaznamenaly žádné chyby, které by bránily aplikaci v chodu. Při testování databáze jsem ale zjistil problém s cache pamětí Teleriku, kdy Telerik nevracel hodnoty aktuální, ale hodnoty z cache paměti. Tento problém vyústil až k výměně Teleriku za NPoco. Tento problém je popsán v 5.3.1.

Kapitola 6

Výsledná aplikace

Dosažené výsledky v aplikaci jsme byli s kolegou třikrát prezentovat v Ostravě v ABB. Zde poté proběhla konzultace dosažených výsledků, zda jsou dostatečné. Pak probíhalo plánování následující fáze vývoje, aby se dodržel postup, který společnost ABB preferuje. Výsledná práce odevzdaná na přiloženém disku je výsledkem třetí fáze, která bohužel nemohla být konzultována se zadavatelem. Třetí fáze totiž není ještě úplně dokončena. Součástí této fáze byl refactoring již hotového kódu včetně několika změn, jako byla již zmíněná výměna Teleriku za NPoco. Nedokončení třetí fáze vývoje nijak neohrozilo splnění bodů zadání bakalářské práce.

Výslednou aplikaci popíši s ohledem na front–end, protože aplikace bez komunikačního rozhraní není použitelná. Jako komunikační rozhraní je v aktuálním stavu možné použít front–end napsaný kolegou, případně i konzolový vstup. Konzolový vstup není součástí práce, ale vzhledem k modulárnosti, na které společnost ABB trvala, se nejedná o zásadní problém. Pro naše účely nebyla konzole jako komunikační rozhraní použitelná.

Při spuštění aplikace se inicializuje OpcClient podle stavu v databázi, umožňuje tedy uživateli i administrátorovi pokračovat v práci tam, kde skončil. Všechny informace obsažené v databázi se aplikují na OPC server, pokud by došlo k nekonzistentnosti dat v databázi, jako je ruční změna některých dat na nevalidní, aplikace „vyhodí“ výjimku. V tomto případě je jedinou možností opravit nevalidní záznam v databázi, jinak není možné aplikaci používat. Oprava záznamu je možná například obnovením zálohy databáze. Tento princip byl zaveden, protože v současném stavu nejsme schopni zjistit, zda vynecháním nevalidních dat nedojde k vytvoření takových dat, která jsou sice validní, ale nejsou v kontextu, jenž byl původně zamýšlen. Toto chování je nežádoucí z hlediska bezpečnosti.

V aktuální verzi aplikace je možné se připojit na OPC Alarm & Events a OPC Data Access server, vytvořit na validním a připojeném serveru libovolný počet subscription. U OPC Data Access je možné k subscription přiřadit jakýkoliv item, který se nachází na serveru. Itemy jsou na front–endu vypisovány jako seznam všech itemů nacházejících se na serveru. Každá akce u OPC Data Access na front–endu je aplikována na samotný OPC server, takže se do chování promítnou i parametry zadané při vytváření subscription, jako je například UpdateRate.

Pro OPC Alarm & Events jsou na back–endu nachystané všechny funkce, ale kvůli absenci komunikace pomocí SignalR mezi back–endem a front–endem není možné informovat o nastalých událostech. Z tohoto důvodu není ani zahrnuta možnost přidávání filtrů a atributů do Alarm & Events subscription. U OPC Alarm & Events je tedy možné přidat server a připojit se na něj, pokud se jedná o validní IP adresu. Na serveru lze vytvářet libovolný počet subscription. Pokud jsou v databázi filtry a atributy, tak se při počáteční inicializaci

aplikují. Tuto aplikali lze vidět v debugeru. Zobrazí se taky na front–endu jako použité atributy a filtry.

Kapitola 7

Možnosti vylepšení

Tato kapitola obsahuje možnosti vylepšení naší aplikace. Některá vylepšení jsou již ve vývoji, ale jiná ještě nebyla ani konzultována se zadavatelem této práce.

Jelikož tato práce obsahuje pouze hotové části, tak vylepšení která jsou v současné době ve vývoji, nejsou zahrnuta ve zdrojových kódech.

Mezi možná vylepšení patří:

- již výše zmíněná implementace technologie SignalR, která umožní notifikovat klienta o změně dat na serveru a dát mu impulz ke stáhnutí dat.
- přidání cache paměti pro uchování dat, aby se omezila zátěž databáze ze strany klientů.
- rozšíření celé aplikace o možnost práce s dalšími typy OPC classic, jako je Historical Data Access nebo XML Data Access.
- přidat možnost použití s OPC Unified Architecture, toto vylepšení by ovšem vydalo na celou další práci, protože OPC Unified Architecture je více rozsáhlé než celé OPC classic.
- přidat automatizované testy pro testování back–endu

Kapitola 8

Závěr

V rámci této bakalářské práce byla vytvořena back–end část aplikace pro získávání dat z automatizace, která umožňuje základní práci s OPC serverem, a to jak Data Access, tak taky Alarm & Events. Na server je možné se připojit i odpojit, vytvořit subscription, do které lze u Data Access přidávat itemy z výpisu itemů na serveru. U Alarm & Events jsou na back–endu implementovány funkce pro přidání filtrů, atributů a oblastí pro nastavení subscription. Je nastaven Event Hadler pro zachytávání událostí a jejich případné přeposlání na front–end. U Data Access i Alarm & Events serveru je možné změnit jméno a to, zda je připojen, u subscription je možné změnit všechny parametry zadané při jejím vytvoření. Všechny změny jsou aplikovány přímo na serveru a dochází tedy i ke změně výstupu. Funkce na back–endu obsahují kontrolu na validitu vstupu. Podle požadavků společnosti ABB jsou texty posílané z back–endu na front–end lokalizovány do anglického jazyka. Zdrojové kódy není možné bez souhlasu autora upravovat, používat ani šířit.

Literatura

[1] SOBOTKA, L. Vizualizace procesů. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2009. 21 s. Vedoucí bakalářské práce Ing. Petr Chlápek.

[2] Galloway, J.; Wilson, B.; Allen, K. S.; aj.: *Professional ASP.NET MVC 5*. John Wiley & Sons, Inc., 2014, ISBN 978-1-118-79475-3.

[3] WWW stránky: *Alarm & Events specification v1.10*. [Online; navštíveno 10.5.2016]. URL https://opcfoundation.org/developer-tools/specifications-classic/alarms-and-events/

[4] WWW stránky: Component Object Model (COM). [Online; navštíveno 9.5.2016]. URL https://msdn.microsoft.com/cs-cz/library/windows/desktop/ms680573(v=vs.85).aspx

[5] WWW stránky: *Data Access Specification v3.0*. [Online; navštíveno 10.5.2016]. URL https://opcfoundation.org/developer-tools/specifications-classic/data-access

[6] WWW stránky: *MVC ASP.NET*. [Online; navštíveno 11.5.2016]. URL http://www.asp.net/mvc

[7] WWW stránky: *OPC Foundation*. [Online; navštíveno 10.5.2016]. URL https://opcfoundation.org/

Přílohy

Seznam příloh

Příloha A

Manual

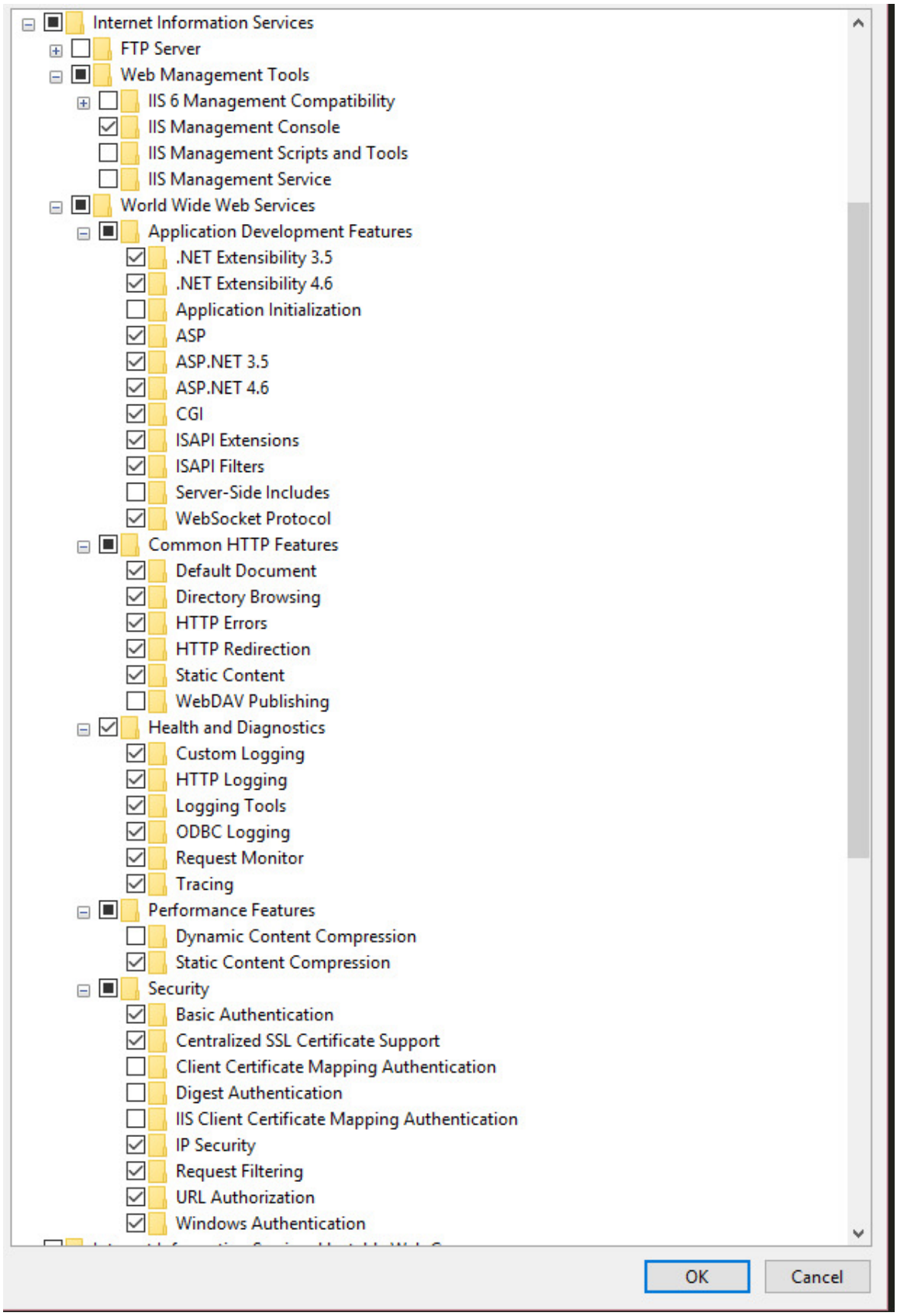
Pro spuštění aplikace ze zdrojových kódů je potřeba nainstalovat systém Microsoft Windows 10, Miscosoft Visual Studio 15, IIS 10.0 Express, MSSQL 2014, OPC server.

A.1 Instalace potřebných součástí

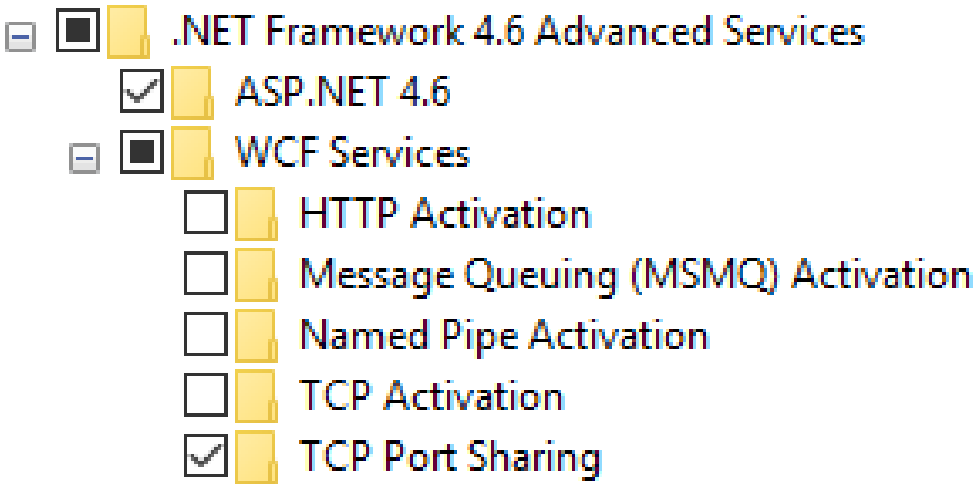
- při instalaci systému Windows 10 se řiďte pokyny uvedenými na https://www.microsoft.com/cs-cz/software-download/windows10.
- při instalaci Microsoft Visual Studio 15 se řiďte pokyny uvedenými na https://msdn.microsoft.com/cs-cz/library/e2h7fzkw.aspx
- pro instalaci a nastavení Internet Information Services(IIS)
  - otevřete programy a funkce (Programs and Features)
  - zvolte zapnout nebo vypnout funkce systému Windows (Turn Windows features on or off)
  - nastavte vše podle A.1
  - nastavte vše podle A.2
- naistalujte MSSQL 2014 with server
  - při instalaci postupujte podle pokynů na obrazovce
  - při dotazu na pojmenování instance databáze napište název: „SQLEXPRESS“
  - zvolte včetně MSSQL Manager
- nainstalujte Vámi preferovaný emulátor OPC serveru
- spusťte SQL Server 2014 Management Studio
- ze složky Abb.Cz.OpcMc.DataAccess/SQLScripts spusťte v SQL Server 2014 Management Studio script CreateDatabase, po vytvoření databáze spusťte nad databází OpcMc script CreateTables.
- otevřete soubor Abb.Cz.OpcMc.sln ve Visual Studio
- klikněte pravým tlačítkem na solution, zvolte propertie

- vyberte multiple startup project a nastavte WebClient a Service na start

Použití Aplikace
Výchozí uživatelské jméno je „user@user.a“ a heslo je „Abc123.“
Výchozí jméno administrátora je „admin@admin.a“ a heslo je „Abc123.“
Pro přidání serveru v aplikaci je potřeba zjistit adresu OPC serveru, který běží na Vašem stroji.



Obrázek A.1: Nastavení Internet Information Services



Obrázek A.2: Nastavení .NET 4.6