Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Realizace rozhraní pro komunikaci s řídící jednotkou vytápění

Bakalářská práce

Autor: Martin Křížek
Studijní obor: Aplikovaná informatika (ai3-k)

Vedoucí práce: Ing. Monika Borkovcová

Hradec Králové Březen 2016

Prohlášení:

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

V Hradci Králové dne Martin Křížek

Poděkování:

Chtěl bych poděkovat své rodině za podporu a oporu při studiu a Ing. Monice Borkovcové za věcné připomínky k formální stránce této práce.

Anotace

Cílem práce je vytvoření rozhraní s řídící jednotkou HC64 v programovacím jazyce C# s použitím .NET Framework pomocí protokolu Modbus přes HID a TCP. V první části bude představeno několik způsobů a možnosti využití kontrolních regulačních jednotek při provozu vytápění. Zároveň budou představeny technologie použité k tvorbě výsledného rozhraní. V rámci realizace bude řešeno řízení uživatelských přístupů, dále bude možnost využití jazykové lokalizace.

Klíčová slova

MVVM, WPF, C#, IRC regulace, regulace vytápění

Annotation

Title: Interface realization for communication with control unit of heating

The goal of this thesis is to create interface with control unit HC64 in C# programming language with .NET Framework using Modbus protocol over HID and TCP. In the first part will be described several ways and possibilities of using control regulation units in the heating. There will be also introduced technologies used to create required interface. The realization part focuses on user access control and localization.

Keywords

MVVM, WPF, C#, IRC regulation, heating regulation

Obsah

1 Úvod

V současnosti je kladen velký důraz na úsporu energií. Lidé se snaží ušetřit peníze na snížené spotřebě energií nejen doma, ale i v zaměstnání. V naší zeměpisné šířce, kde se střídají roční období, patří v zimních měsících velká část spotřeby energií na osvětlení a vytápění.

Konstrukce budovy má zásadní dopad na spotřebu energie, která je použita na vytápění. Použité materiály a technologie přímo ovlivňují tepelnou ztrátu objektu. Čím je tepelná ztráta vyšší, tím více z budovy uniká teplo do okolí a tím více energie je třeba spotřebovat na udržení požadované teploty. I starší budovy procházejí rekonstrukcí, při které dochází k zateplení obálky budovy. To se zpravidla skládá z výměny starých oken a dveří za nové (které lépe udržují teplo) a zároveň dochází k zateplení stěn budovy (nejčastěji polystyrenem z venkovní strany). Tyto změny snižují tepelnou ztrátu budov. Ta dříve nepatřila k hlavním sledovaným parametrům, a proto většina ze starších budov neodpovídá dnešním standardům.

V současné době se budovy podle energetické náročnosti (udávané v kWh/m^2 za rok) rozdělují do skupin A - G.

A. <51 – mimořádně úsporná budova.
B. 51 – 97 – úsporná budova.
C. 98 – 142 – vyhovující budova.
D. 143 – 191 – nevyhovující budova.
E. 192 – 240 – Nehospodárná budova.
F. 241 – 286 – velmi nehospodárná budova.
G. >286 – mimořádně nehospodárná budova.

Podle spotřeby jsou pak budovy slovně také rozdělovány na dvě kategorie:

1. Standardní dům – spadají do kategorie C.
2. Energeticky úsporný dům – spadají do kategorie A, nebo B.

Energeticky úsporný dům je dále možno rozdělit na další kategorie. Například na Nizkoenergetický dům (kategorie A), Energeticky pasivní dům (energetická

náročnost maximálně 15 kWh/m^2 za rok – podkategorie A+) a Energeticky nulový dům (maximální energetická náročnost 5 kWh/m^2 za rok – podkategorie A++). Speciální kategorií je pak ještě Energeticky nezávislý dům, který je vybaven výrobnou elektrické energie o dostatečné kapacitě, aby zajistil vlastní spotřebu.

Díky hodnotám tepelných ztrát lze lépe stanovit požadavky na výkon vytápění. V dostatečně zateplených budovách je ztráta tepla snížena na takovou úroveň, že je možné, aby nebylo vytápění zapnuto neustále ani při velkých teplotních rozdílech uvnitř a vně budovy a tím pádem dochází k větší poptávce regulace vytápění. Díky kvalitnímu zateplení se rovněž snižuje délka topné sezóny.

Regulaci lze obecně rozdělit podle několika kritérií. Mezi tato kritéria patří rozdělení podle zdroje energie (zemní plyn, solární energie, kotel na pevná paliva, tepelné čerpadlo, elektřina,…), nebo podle typu média (voda, vzduch, infračervené záření, odporové kabely, či fólie,…). Samotná regulace pak může být například ekvitermní, regulace podle referenční místnosti, nebo IRC regulace.

Regulace vytápění se stává velmi rozšířenou nejen z důvodu ušetření energie za vytápění, ale také díky komfortu, který poskytuje. Velká část regulací vytápění obsahuje nastavení různého počtu teplotních změn během dne minimálně na jeden týden. To zaručuje udržování požadovaných teplot v průběhu celého týdne i v případě, že se požadavky na jednotlivé dny liší. Typickým požadavkem je různé vytápění pro pracovní týden a víkend.

Tato práce se snaží představit několik základních typů regulací vytápění s jejich přednostmi a nedostatky. Tento výčet nicméně vzhledem k zaměření práce není vyčerpávající a má převážně orientačně informační hodnotu.

2 Cíl práce

Cílem této práce je vytvořit první verzi aplikace pro komunikaci s řídící jednotkou vytápění od firmy BMR. Vytvoření aplikace zadal přímo výrobce regulace vytápění, který specifikoval základní požadavky na některé použité technologie tak, aby odpovídaly technologiím použitým v jiných aplikacích firmy. Kompletní seznam požadavků zadavatele je v kapitole 5.

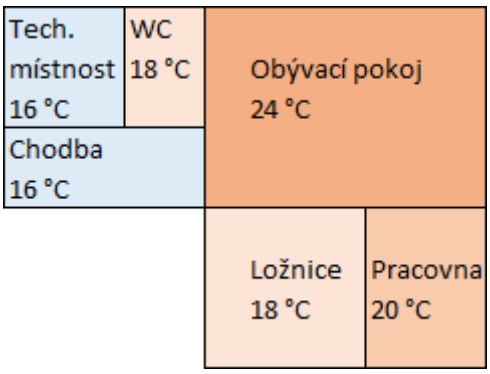
Teoretická část má za cíl seznámení s několika základními typy regulace vytápění objektů, dále zde budou představeny konkrétní technologie, které splňují požadavky zadavatele a jsou vhodné pro vývoj daného typu aplikace. Jedná se hlavně o návrhový vzoru Model-View-ViewModel (viz kapitola 7.1), který se používá jak ve WPF aplikacích, tak i v projektech založených například na JavaScriptové knihovně Knockout.

Cílem praktické části je seznámení s konkrétními postupy, které budou použity při vývoji aplikace, jako například použitý způsob lokalizace. Dále bude obsahovat zdůvodnění některých konkrétních nastavení - především pak zvolené parametry zabezpečení přihlášení, které budou podpořeny literárními zdroji a testem výkonosti.

Teoretická východiska

3 Regulace jednotlivých místností

Regulace teploty v jednotlivých místnostech nezávisle na ostatních, patří k nejkomfortnějšímu způsobu ovládání teploty v domě. Tento typ regulace je většinou zrealizován buď pomocí pokojových termostatů, nebo pomocí IRC regulace. Z obrázku Obrázek 1 je patrné, že požadavek na různé teploty v místnostech, může být založen například na účelu dané místnosti. Vzhledem k tomu, že je každá místnost řízena nezávisle na ostatních, dokáže systém snadno reagovat také na ostatní zdroje tepla, které ovlivňují pouze část objektu, nebo jen jednu konkrétní místnost. Mezi takové zdroje tepla patří například přímý sluneční svit, krb (případně komín), nebo pečící trouba. Regulace dokáže v konkrétní místnosti vytápění buď omezit, nebo ho úplně zastavit nezávisle na ostatních místnostech.



Obrázek 1 V různých místnostech lze vyžadovat různé teploty. [zdroj: vlastní]

Regulace vytápění je požadována hlavně ze dvou důvodů. Prvním jsou nižší náklady na vytápění a druhým pak komfort. Při návrhu regulace je však třeba si uvědomit, že ne všechny objekty, nebo zdroje tepla, se dají regulovat se stejnou úsporou. Pokud má objekt vysoké tepelné ztráty, nebo není rozvod tepla v objektu dostatečně dimenzovaný, je možné, že je tepelný zdroj využíván naplno v průběhu celého dne bez větších rozdílů. V takovém případě nemusí regulace vytápění přinést dostatečně velkou úsporu na to, aby se investice do ní vůbec vrátila. Různé zdroje tepla pak mohou vyžadovat jiný přístup k regulaci. Jde například o použití mechanických spínačů. V takovém případě může mít špatné použití regulace (využívání PWM) za následek snížení životnosti spínačů v důsledku častého spínání. Kvůli tomu se nejen

nevrátí investice do samotné regulace, ale ještě navíc se sníží výnosnost investice do daného tepelného zdroje. Některé topné zdroje pak pro svou deklarovanou účinnost mohou například vyžadovat co nejdelší nepřerušovaný chod, protože dosahují nejlepší účinnosti při konkrétní teplotě. Pokud se tato provozní teplota sníží (například v důsledku regulace příkonu zdroje kvůli drobnému dorovnání teploty), může se účinnost snížit i o desítky procent. To má za následek vyšší náklady na vytápění a tudíž nevratnost investice do regulace a případně i daného tepelného zdroje.

Vzhledem k různým charakteristikám a potřebám použitého zdroje tepla, je třeba mít na paměti, že se výsledné chování systému různí i po nainstalování regulace vytápění. Pokud je například použit zdroj tepla, který potřebuje pro nejlepší účinnost plný výkon, je třeba počítat s většími odchylkami od požadované teploty než u zdroje, který lze plynule řídit téměř v reálném čase bez snížení jeho účinnosti.

3.1 Pokojové termostaty

Pokojový termostat se obvykle skládá z teplotního čidla, řídící jednotky a komunikačního rozhraní pro uživatele (klávesnice a displej). Všechny tyto části bývají uloženy dohromady v jedné krabičce, která je připevněna na stěnu místnosti, kterou reguluje.

Každý pokojový termostat měří teplotu v dané místnosti a zároveň v ní podle naměřené teploty řídí vytápění. Některé pokojové termostaty mohou obsahovat například i limitní podlahové čidlo, které nedovolí, aby teplota podlahy přesáhla nastavenou maximální teplotu [19]. Vzhledem k tomu, že je každý pokojový termostat zcela autonomní, nemusí být jednotlivé pokojové termostaty nijak propojeny a jejich instalace je tak zpravidla jednodušší a nemusí vyžadovat příliš velké stavební úpravy. Pokud je ovšem ovládání zdrojů tepla centralizováno - například pokud jsou ovládací hlavice pro teplovodní systém umístěny v jedné místnosti, pak tato výhoda odpadá. Další výhodou mohou být nižší pořizovací náklady při menším počtu místností, než u IRC systému a to především díky větší jednoduchosti – u některých dražších pokojových termostatů pak tato výhoda odpadá. Další výhodou je větší volnost při instalaci. Při dostavbě další místnosti, je

možné jednoduše dokoupit další pokojový termostat, aniž by tento zásah jakkoliv ovlivnil již nainstalované pokojové termostaty. Kompletně nezávislé pokojové termostaty mají nevýhodu v tom, že vzhledem k jejich úplné nezávislosti, je třeba při hromadné změně teploty (počátečním nastavení) všechny pokojové termostaty nastavit jednotlivě. Ovládání většího počtu pokojových termostatů není pak pouze časově náročnější, ale je i méně přehledné, protože nelze z jednoho místa zkontrolovat nastavení všech místností najednou.

3.2 Pokojové termostaty se síťovou konektivitou

Jedná se o speciální typ pokojových termostatů. Jde vlastně o snahu zlepšit uživatelskou přívětivost pokojových termostatů a v některých případech přiblížit ovládání těchto pokojových termostatů IRC systému. Obvykle jde o aplikaci, kam se dá pokojové termostaty přidat a spravovat je. A to buď nezávisle (pak jde vlastně o zjednodušení ovládání pouze v tom, že jsou všechny na jednom místě), nebo lze nastavit stejně více pokojových termostatů najednou (pak se jedná o ovládání, které je blízké IRC systému). Daná aplikace může být buď na lokální síti (v osobním počítači, nebo chytrém telefonu), nebo v cloudu jako webová aplikace. Příkladem takového typu pokojového termostatu jsou pokojové termostaty firmy Nest, které přinášejí i další funkce, jako je například automatická reakce na geografickou polohu uživatele – tato poloha je získávána z polohy chytrého telefonu, který je do systému připojen [12].

První dva případy s sebou nesou omezení, které vyplývá z toho, že samy pokojové termostaty o sobě navzájem nevědí - připojení jednotlivých pokojových termostatů je definováno pouze v aplikaci na daném zařízení. Třetí případ pak trpí nepoužitelností v případě nedostupného internetového připojení.

Z předchozího odstavce je zřejmé, že se více, či méně, maže nevýhoda, která je zmíněna u pokojových termostatů v kapitole 3.1. Zároveň s ní však mizí i výhody. Vzhledem k propojení termostatů v počítačové síti, je třeba buď všem pokojovým termostatům zajistit kabelové připojení, nebo dostatečný signál bezdrátové sítě. Kvůli tomu se instalace prodražuje a komplikují se stavební úpravy, které jsou nutné ke správnému fungování pokojových termostatů. V důsledku nutnosti komunikace

v síti dále rostou požadavky na hardwarovou a softwarovou výbavu jednotlivých pokojových termostatů. To vede k jejich zdražení, což má za následek snížení maximálního počtu pokojových termostatů, které jsou cenově výhodnější, než IRC systém. Pokud není přidávání dalších pokojových termostatů do aplikace příliš složité, zůstává jako jediná výhoda větší volnost v přidávání nových pokojových termostatů. Naopak je třeba mít na paměti, že jednotlivé pokojové termostaty zvyšují zatížení lokální sítě, ke které jsou připojeny. Pokud navíc komunikují s aplikací v cloudu, zvyšuje se objem přijímaných a odesílaných dat. V neposlední řadě se také nabízí otázka zabezpečení, protože je třeba zajistit zabezpečení každého pokojového termostatu zvlášť.

3.3 Independent Room Control (IRC) regulace

Podle [2] se jedná velmi dobrý způsob regulace. Ta spočívá v tom, že v každé místnosti je umístěno teplotní čidlo, které po sběrnici posílá aktuální teplotu do řídící jednotky. Některá teplotní čidla lze, podobně jako pokojový termostat, rozšířit o další senzory (například detekce otevřeného okna). Z přijatých informací následně řídící jednotka rozhodne, jak nastavit topení v dané místnosti, aby byla co nejlépe splněna požadovaná teplota. Vzhledem k tomu, že je třeba všechna čidla propojit s řídící jednotkou pomocí nějaké sběrnice, je tento systém obvykle náročnější na instalaci kabeláže a s tím spojenou větší potřebu stavebních úprav. Případná bezdrátová komunikace vyžaduje zajištění dostatečně silného signálu u všech čidel, což může být ve výsledku náročnější, než instalace kabeláže. Výhodou tohoto systému je, že ačkoliv je topení řízeno jednotlivě po místnostech, je řízeno z jednoho místa řídící jednotkou. To znamená, že případné hromadné úpravy, či počáteční nastavování, jsou provedeny pouze jednou na řídící jednotce, která následně úpravy vyhodnotí a případné nové informace rozešle čidlům (pokud se jedná například o digitální čidlo, které zároveň zobrazuje aktuální a požadovanou teplotu v místnosti). Řídící jednotku lze, podobně jako pokojové termostaty z kapitoly 3.2, připojit do počítačové sítě (záleží na možnostech jednotky). Výhoda oproti připojení pokojových termostatů je v tom, že je snížena zátěž celé sítě, protože všechna data jsou dostupná na jednom místě. Není tak třeba vytvářet mnoho spojení a komunikovat se všemi zařízeními najednou – tato výhoda je tím znatelnější, čím více

místností objekt má. U jedné jednotky se rovněž lépe zajišťuje dodatečná bezpečnost a také se celá síť zpřehledňuje. Vzhledem k tomu, že se v současné době stále ve většině případů používá IPv4, tak je kladem i to, že jednotka obsadí pouze jednu IP adresu. Tato výhoda není pro klasickou domácnost zajímavá. Mohla by však být podstatná pro veřejné budovy (například knihovny, nebo galerie), nebo kanceláře, kde je pravděpodobné, že se bude vyskytovat velké množství zařízení, připojených do počítačové sítě. Například při dvaceti místnostech je při použití jedné řídící jednotky dosažena oproti pokojovým termostatům úspora devatenácti IP adres, které mohou využít jiná zařízení.

3.4 Řídící jednotka HC64

Tato bakalářská práce se zabývá rozhraním pro komunikaci s řídící jednotkou HC64 od firmy BMR (Obrázek 2). „Řídící jednotka HC64 je vybavena funkcemi pro regulaci el. přímotopných systémů RT64, teplovodních RNET64, ovládání předokenních rolet nebo žaluzií, přepínání topení / chlazení.“ [16]

Obrázek 2 Řídící jednotka HC64 [zdroj: vlastní]

V reálné instalaci je tato jednotka umístěna na DIN liště v regulační skříni, která obsahuje další prvky potřebné ke správné regulaci (například spínací polovodičové prvky). Tato skříň je buď zasazena pod omítku, nebo je zavěšena na zeď a to nejčastěji v místnosti s dalším technickým vybavením (například kotelna, technická místnost, nebo garáž). Do této skříně jsou přivedeny vodiče od čidel a výkonové kabely od topidel. Řídící jednotka je vybavena konektory USB typu B a RJ45. Pokud má být jednotka zapojena do počítačové sítě, je třeba do této skříně zajistit přívod i síťového kabelu. Jednotce lze nastavit pevnou IP adresu, masku sítě a její bránu. Pokud objekt disponuje připojením k internetu a veřejnou IP adresou, je možné vytvořit na routeru přesměrování příslušných portů a přistupovat tak k jednotce i vzdáleně. Jednotka má integrované responzivní webové rozhraní, pomocí kterého lze prohlížet a nastavovat uživatelská nastavení a aktuální stav systému. Přístup k tomuto rozhraní je chráněn uživatelským jménem a heslem. Rozhraní lze používat ve všech prohlížečích, které podporují JavaScript a HMTL5. Díky tomu, že je vše uloženo v jednotce, která zároveň poskytuje toto rozhraní, je možné se pomocí něho připojit k jednotce i bez přístupu k internetu (pouze v síti LAN).

„HC64 umožňuje nastavení denních nebo týdenních regulačních programů pro 32 nezávislých topných okruhů. Zobrazuje reálnou a požadovanou teplotu okruhu, status topí / netopí a další diagnostické informace pro jednotlivé okruhy nebo rozšiřující moduly.“ [16] Systém je řízen polovodiči, ne pomocí relé. Díky tomu se zvyšuje životnost a odpadají rušivé zvuky při spínání relé, či stykačů. U některých zdrojů vytápění lze díky tomu využít pulzně šířkovou modulaci (PWM) bez obav z toho, že bude snížena životnost spínacích prvků IRC systému. Ke správnému rozhodování o topení/chlazení jednotlivých okruhů mohou kromě nastavené požadované a aktuální teploty sloužit i další informace, jako je čidlo okenního kontaktu, informace o tom, zda je právě nízký, či vysoký tarif, ruční změna požadované teploty oproti definovanému režimu, u podlahového vytápění aktuální teplota podlahy (teplota podlahy nesmí přesáhnout stanovený limit, nebo nesmí klesnout pod požadovanou hodnotu), nebo možnost připojení ke snímači karty. Ten se používá například v penzionech, nebo kancelářích, či školách. Pokud je přítomna karta, znamená to, že se má regulovat k jiné požadované teplotě, než když karta

přítomna není. Díky tomu je možné na vyšší teplotu topit v místnostech pouze v době, kdy se v nich někdo zdržuje, a díky tomu lze při vytápění dosáhnout další úsporu. Rozdíl teplot je nicméně vhodně zvolit, protože při příliš velkém rozdílu teplot, by mohlo dotopení na vyšší teplotu trvat příliš dlouho. Zároveň je třeba mít na paměti, že pokud se bude jednat například o vnitřní sousední místnosti, které od sebe nejsou tam dobře tepelně izolovány, může mít příliš nízká teplota v jedné místnosti, negativní dopad na vytápění sousedních místností.

4 Další typy regulace vytápění

Mezi další důležité druhy regulace patří ekvitermní regulace a regulace podle referenční místnosti.

4.1 Ekvitermní regulace

Ekvitermní regulace slouží k regulaci zdroje tepla podle venkovní teploty. Princip spočívá v tom, že je při nižší venkovní teplotě nastavena vyšší teplota dodávaná do topného systému a tím dochází k vyrovnání aktuální tepelné ztráty, která je závislá právě na venkovní teplotě. Tato regulace se řídí pomocí ekvitermních křivek. Jedna ekvitermní křivka znázorňuje závislost venkovní teploty na teplotě dodávané do systému při požadavku na určitou nastavenou teplotu místnosti. Její tvar a průběh je závislý na tepelně-izolačních a tepelně-akumulačních vlastnostech objektu.

Snížení spotřeby energie spočívá v tom, že pro vyšší venkovní teplotu postačuje dodat do systému nižší teplotu [2]. V praxi to znamená, že pokud je ekvitermní regulace použita například při vytápění vodou (nejčastější případ), je při vyšší venkovní teplotě potřeba ohřát vodu na nižší teplotu a tím se sníží i náklady na ohřev této vody.

4.2 Regulace podle referenční místnosti

Tento typ regulace je více příbuzný typům, které jsou uvedeny v kapitole 3. Rozdíl spočívá v tom, že je řízení regulace zjednodušeno. Regulace přijímá aktuální teplotu pouze z referenční místnosti a na základě požadované teploty v této místnosti pak vytápí celý objekt. Pokud je v referenční místnosti třeba zvýšit teplotu, je topení spuštěno v celém objektu a naopak. Jedná se zpravidla o levnější typ regulace, která je ale dostatečně přesná pouze v referenční místnosti. Referenční místností se tak stává místnost, která je obývána po největší část dne, nebo místnost, kde zákazník vyžaduje co nejpřesnější aktuální teplotu k teplotě požadované. Podle [2] vyzařuje člověk do okolí přibližně stejné množství tepla, jako 100 W žárovka. Pokud je tedy v referenční místnosti více lidí, zatímco jsou ostatní místnosti prázdné, znamená to, že rozdíl teploty v místnostech bude ještě větší. Částečným řešením, které je ovšem velmi neekonomické, je mít otevřené dveře do všech místností a tím teplotu

v jednotlivých místech co nejvíce vyrovnat. Pokud je však mezi místnostmi například chodba, kterou není potřeba vytápět na tak vysokou teplotu, je tento způsob velmi nepraktický.

5 Seznam požadavků zadavatele

Rozhraní pro komunikaci s řídící jednotkou HC64 bylo zadáno firmou BMR s následujícími požadavky na použité technologie a funkčnost:

- Programovací jazyk C# s použitím .NET Frameworku.
- Komunikace s řídící jednotkou pomocí protokolu Modbus přes HID a TCP.
- Více uživatelských rolí.
- Vzhled podobný již vytvořenému webovému rozhraní.
- Možnost lokalizace.
- Podporované operační systémy Windows XP a vyšší.

6 Upřesnění požadovaných technologií

Následující body jsou upřesněním verzí požadovaných technologií tak, aby splňovali všechny požadavky zadavatele:

- C# 6.0
- .NET Framework 4.0
- WPF

6.1 C# 6.0

C# 6.0 je zatím poslední vydaná verze programovacího jazyka C#. Tato verze přináší vylepšení převážně v psaní kódu. Mezi hlavní výhody, které byly představeny, patří například klíčové slovo *nameof*, které vrací název zadané proměnné. Toho lze výhodně použít například při používání rozhraní *INotifyPropertyChanged*, jak je blíže vysvětleno v kapitole 7.2.4. Se zpracováním textu souvisí i další novinka, která umožňuje programátorovi přehledněji vytvářet textové řetězce. Pokud programátor potřebuje do textového řetězce vložit hodnotu proměnné, měl dříve dvě hlavní možnosti. Mohl text postupně skládat (viz algoritmus Ukázka kódu 1), nebo může využít metody *string.Format*, která v prvním argumentu dovoluje napsat celý text a vložit do něj zástupné znaky, které definují, na jaká místa se mají vložit další argumenty metody, jak je zobrazeno v algoritmu Ukázka kódu 2.

Ukázka kódu 1 Vložení hodnoty proměnné do textového řetězce roztržením řetězce

```
string text = "Zaměstnanec odpracoval " + currentUser.Hours + "
hodin.";
```

Ukázka kódu 2 Vložení hodnoty proměnné do textového řetězce pomocí metody *string.Format*

```
string text = string.Format("Zaměstnanec odpracoval {0} hodin.",
currentUser.Hours);
```

Druhý způsob je při menším počtu proměnných přehlednější, jelikož text není přerušen a je proto jednodušší zkontrolovat například to, zda je před (nebo za) hodnotou proměnné mezera. Při větším počtu proměnných se však tento způsob stává nepřehledným. Navíc je údržba takového kódu velmi náročná, protože číslo ve složených závorkách udává index argumentu a pokud se text změní tak, že je

vypuštěna například hodnota druhého argumentu ze sedmi, je třeba přečíslovat všechny výskyty následujících proměnných. V nejnovější verzi jazyka tak přibyla interpolace textu (String interpolation), která, jak je vidět v algoritmu Ukázka kódu 3, kombinuje oba dva zmíněné zápisy. Na první pohled je tak vidět, zda je text okolo proměnné správně formátovaný (mezery) a přímo je vidět, která proměnná se na kterém místě používá. Znak dolaru je před textem uveden proto, aby došlo v textu k rozpoznání složených závorek, jako řídícího znaku a ne jako textového znaku. U každého výskytu složených závorek, které mají být použity jako znak složených závorek, je třeba jejich výskyt zdvojit (podobně, jako při použití druhého způsobu).

Ukázka kódu 3 Vložení proměnné do textového řetězce pomocí interpolace textu.

```
string text2 = $"Zaměstnanec odpracoval {currentUser.Hours} hodin.";
```

Mezi další novinky pak patří například lepší práce s objekty, které mohou nabývat hodnoty *null*, nebo snazší používání statických tříd. V neposlední řade sem patří i počáteční inicializace vlastností objektu. Ta již nemusí být prováděna například v konstruktoru, nebo pomocí atributu, ale lze ji nastavit přímo při deklaraci, jako hodnotu atributu třídy (algoritmus Ukázka kódu 4).

Ukázka kódu 4 Počáteční inicializace vlastnosti třídy v C# 6.0.

```
public int Pocet { get; set; } = 3;
```

6.2 .NET Framework 4.0

.NET Framework je platforma usnadňující vývoj aplikací pro operační systémy Windows. Obsahuje mimo jiné rozsáhlou knihovnu tříd a poskytuje nástroje pro použití kódu, který byl napsán v jiném programovacím jazyce. Vzhledem k neustálému vývoji hardware i software společnost Microsoft vydává nové verze operačního systému. Nové verze operačního systému přinášejí do systému nové funkce a proto jsou vydávány i nové verze .NET Frameworku, které tyto funkce zpřístupňují programátorům. Bohužel není možné veškeré nové funkce podporovat ve starších operačních systémech a proto je k tvorbě aplikace možné využít nejvyšší verzi .NET Framework, který ještě podporuje nejstarší operační systém, který má být podporován aplikací. Pro Windows XP je nejvyšší možná verze .NET Framework 4.0 [23].

Novější verze .NET Frameworku například umožňuje nastavení kultury pro celou aplikační doménu [27], čímž zjednodušuje případnou lokalizaci, protože všechna nově vytvořená vlákna mají nastavenou kulturu podle toho, jak se nastaví na začátku. Mezi další vylepšení patří například to, že informace o chybě *HResult* je ve verzi 4.0 definována pouze jako *protected* [9] a tím pádem není možné tento kód při obsluze výjimky ani přečíst. Jediný způsob, jak kód chyby získat, je projít zprávu o chybě a najít v ní odpovídající část, kde je kód chyby uložen. Ve verzi 4.5 je možné tuto vlastnost přečíst přímo, jelikož je její čtení veřejně dostupné [10]. Použití staršího .NET Frameworku má vliv i na některé nové funkce v programovacím jazyce C# 6.0, které jsou s novou verzí úzce spojeny. Jedná se například o některé funkce zabývající se zpracováním textu (viz kapitola 6.1).

6.3 Windows Presentation Foundation (WPF)

U aplikace pro operační systém Windows psané v programovacím jazyce C# je na výběr mezi starší technologií Windows Forms (WinForms) a novější technologií Windows Presentation Foundation (WPF). WinForms využívá ke svému běhu grafickou knihovnu GDI+, kdežto WPF využívá Direct3D. Díky tomu lze u WPF aplikace využít hardwarové akcelerace a tím grafické efekty nejenom zrychlit, ale zobrazit je také plynuleji - o grafickou část se stará GPU, místo CPU. WPF také v základu pracuje s vektorovou grafikou. Zároveň má zabudovanou podporu pro 2D a 3D grafiku a animace, které tak lze snadno vytvořit a ovládat. S touto novou technologií byl představ nový značkovací jazyk Extensible Application Markup Language (XAML), který je určen, jako primární způsob pro definici rozložení jednotlivých prvků v okně aplikace. Zároveň lze pomocí něho definovat mimo chování i styly, které je možné jednou nadefinovat a poté je přiřazovat jednotlivým prvkům napříč celou aplikací. Styly lze měnit i za běhu aplikace, takže uživatel dokáže změnit vzhled aplikace podle svých preferencí bez toho, aby musel aplikaci restartovat. Změny se projeví okamžitě. Díky značkovacímu jazyku XAML nemusí vzhled nutně vytvářet programátor se znalostmi konkrétního programovacího jazyka (v tomto případě C#), ale jeho tvorbu lze přenechat grafikovi, který vše nadefinuje přímo. Díky návrhovému vzoru Model-View-ViewModel (MVVM) se dá také vzhled téměř kompletně oddělit od funkčního kódu (více viz kapitola 7.1).

Vzhled lze definovat zcela volně. WPF aplikace dokáže díky tomu být pro uživatele k nerozeznání od WinForms aplikace, nebo může vypadat jako počítačová hra, či interaktivní video, čehož se ve WinForms dá docílit pouze velmi složitými postupy pomocí spojování různých technologií.

Přesto, že WPF kombinuje DirectX (3D a hardwarová akcelerace) s WinForms (vývojářská produktivita), Adobe Flash (silná podpora animací) a HTML (využití značkovacího jazyka), nelze říci, že by tyto technologie nahrazovala. Například při velkém důrazu na grafiku a výkon je stále lepší využít přímo DirectX, protože WPF technologie je v první řadě myšlena jako nástupce klasických WinForms, které se snaží rozšířit hlavně v oblasti multimédií a grafiky. [3]

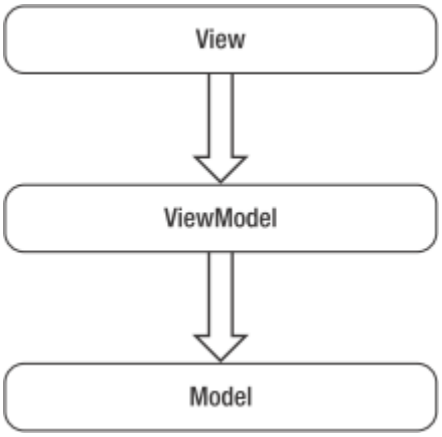
Jeden z klíčových požadavků zadavatele je vzhled podobný webovému rozhraní. Toto rozhraní je vytvořeno pomocí knihovny jQuery Mobile, která přizpůsobuje vzhled webové aplikace převážně pro chytré telefony a tablety. Vzhledem k uvedeným skutečnostem bylo se zadavatelem dohodnuto, že aplikace bude naprogramována pomocí technologie WPF. Díky tomu lze vytvořit aplikaci, která bude pro uživatele vzhledově podobná známému webovému rozhraní, ale přitom půjde o klasickou aplikaci.

7 Další potřebné technologie

V této kapitole budou představeny další technologie, které mají buď spojení na technologie představené v kapitole 6, nebo jsou potřebné ke splnění dalších požadavků zadavatele. Nejedná se tedy o technologie, která by si zadavatel přímo vyžádal.

7.1 Model-View-ViewModel (MVVM)

Jak bylo zmíněno v kapitole 6.3, je Model-View-ViewModel (MVVM) návrhový vzor, který se používá v souvislosti s technologií WPF. Tento návrhový vzor představil na svém blogu John Gossman [11] právě pro vývoj aplikací ve WPF. Jak název napovídá, skládá se ze tří základních částí: Model, View a ViewModel. Tyto části jsou propojeny co možná nejméně (provázanost je znázorněna na obrázku Obrázek 3) a díky tomu lze mimo jiné jednotlivé části ve větších týmech vytvářet současně, což snižuje dobu potřebnou na vytvoření aplikace.



Obrázek 3 Provázanost částí v návrhovém vzoru MVVM [1].

Model a ViewModel jsou klasické třídy (případně celé knihovny tříd), které mohou fungovat jako jakékoliv jiné třídy v jazyce C#. Další dva podobné návrhové vzory jsou Model-View-Presenter (MVP) a Model-View-Controller (MVC). Všechny tři návrhové vzory jsou určeny k tomu, aby oddělili část View (vzhled) od části Model (data), ale protože je každý návrhový vzor určený pro jinou technologii, nacházejí se v těchto vzorech odlišnosti a nelze je proto zaměňovat a dávat rovnítko mezi ViewModel, Presenter a Controller. MVVM se nepoužívá pouze ve WPF, ale například též v JavaScriptové knihovně Knockout, nebo ve frameworku pro webové aplikace

ZK, který je psaný v Javě. MVP se využívá například v technologii WinForms a MVC využívá například ASP.NET MVC.

7.1.1 Model

Model je ve WPF klasická C# třída, která obsahuje data, se kterými uživatel pracuje. Pokud je to vhodné, obsahuje také validaci dat, sledování změn a podobně [4]. Tato třída obvykle nemá reference na View, ani ViewModel. Díky tomu je možné ji vytvořit zcela nezávisle na ostatních částech a lze ji použít i v jiných projektech, které pracují se stejnými daty nezávisle na tom, zda tento další projekt využívá návrhový vzor MVVM, nebo zda se jedná například o konzolovou aplikaci. Také v ostatních zmiňovaných návrhových vzorech se Model stará čistě o data a neví o dalších částech programu.

7.1.2 View

View je vlastně uživatelské rozhraní, které zobrazuje data a pomocí vstupů zařízení (například klávesnice a myš) dovoluje ovlivňovat stav programu, textových dat, videa, foto alba, nebo čehokoliv dalšího, co má být uživateli zobrazeno [4]. Ve WPF se View skládá z XAML souboru a klasické třídy například v jazyce C#. Tato třída se nazývá code-behind. Okolo tohoto rozložení se stále vedou mezi programátory na diskuzních fórech (například [15][17][21]) debaty. Někteří programátoři zastávají názor, že by se View mělo skládat pouze z XAML souboru [8]. Obvyklá konstrukce však obsahuje code-behind jako součást View s tím, že je v něm pouze kód, který přímo souvisí pouze se vzhledem. Práce s některými funkcemi vzhledu je, minimálně co do rozsahu kódu, snazší přímo v daném programovacím jazyce. Pro všechny tři návrhové vzory platí, že pokud bude dané View z aplikace odstraněno, nemělo by to mít vliv na logiku aplikace, ale pouze na způsob, jakým bude uživatel komunikovat s aplikací. Tento stav znázorňuje i obrázek Obrázek 3, kde je znázorněno, že ani ViewModel, ani Model, nemají na View žádné reference.

7.1.3 ViewModel

V knize [4] je napsáno, že ViewModel je vlastně Model pro View. Tuto jednoduchou větu pak rozvádí do několika odstavců, které problematiku dále přibližují. Je třeba

si uvědomit, že ViewModel není další code-behind pro View. ViewModel je v podstatě abstrakce uživatelského rozhraní, takže by neměl mít žádnou informaci o prvcích, které jsou zobrazeny na obrazovce. Díky existenci ViewModelu je mimo jiné mnohem snazší tvořit testy pro uživatelské rozhraní, protože ViewModel je třída, může být snadno testována pomocí jednotkových testů.

Další výhodou je to, že při výměně některé kontrolky za jinou, případně výměně celého View, jsou v dalším kódu provedeny maximálně drobné úpravy, protože ViewModel nemá žádné informace o tom, jak jsou jeho vlastnosti prezentovány uživateli. Jeho hlavní práce je shromáždit data z jednoho, nebo více Modelů. Tato data pak ve vhodné formě poskytne pro View. Zatímco ViewModel se stará o to, aby byla data správně zadána (provádí validaci dat), View se stará o to, jak data získaná z ViewModelu co nejlépe zobrazit uživateli. Veřejné vlastnosti ViewModelu tak mnohem více odpovídají tomu, co se má zobrazit, než tomu, jak jsou data ve skutečnosti uložena.

7.1.4 Presenter

Presenter není součástí návrhového vzoru MVVM. Je zde uveden kvůli úplnosti a rozdílům, které ho odlišují od ViewModelu. Odlišuje se především tím, že přímo komunikuje s rozhraním, které v návrhovém vzoru MVP implementuje View. ViewModel naproti tomu s View *nekomunikuje*, ale pouze vyvolává události, ke kterým se případné View může, ale nemusí přihlásit. Pro jednotlivá View je tak v MVP třeba vytvářet rozhraní, která částečně odstiňují Presenter od View. Je to nicméně provedeno na úkor toho, že místo s View je Presenter provázán s těmito rozhraními.

7.1.5 Controller

Controller není součástí návrhového vzoru MVVM, ale pro úplnost jsou zde uvedeny některé jeho rozdíly proti ViewModelu. Zatímco Presenter a ViewModel jsou si velice podobné (každý ViewModel, nebo Presenter má pouze jedno View. Rozdíl je hlavně ve způsobu, jakým komunikují s View), Controller se liší ve více ohledech. Je to dáno převážně tím, že MVC se velmi často používá při tvorbě webové aplikace, která běží na serveru a uživatelům odesílá jako odpověď na dotazy další stránky -

View. To, které View se odešle uživateli do okna prohlížeče, řeší právě Controller, který je často spojen s více View najednou. V době, kdy uživatel pracuje s View, není toto View obvykle nijak propojeno se zbytkem aplikace, což je další důvod, proč se MVC více liší od ostatních dvou představených návrhových vzorů. Controller je přesto stále třídou, která má klasické metody. V ASP.NET MVC mohou být tyto metody volány přes URL adresu, která je speciálním routováním přeložena na volání metody daného Controlleru.

7.2 Podpora MVVM ve WPF

Jak bylo napsáno v kapitole 7.1, byl návrhový vzor MVVM představen právě pro tvorbu aplikací ve WPF. Díky tomu má MVVM v této technologii plnou podporu a za snadnou implementací stojí zejména: Binding, UpdateSourceTrigger, Converter, INotifyPropertyChanged a ICommand.

V současné době se návrhový vzor MVVM používá i u dalších nových typů aplikací. Jde o Windows Store aplikace pro operační systémy Windows a Windows Phone. Tvorba těchto aplikací je pro současné verze systémů velmi podobná tvorbě aplikací ve WPF. Rozdílná jsou zde pouze některá specifika - aplikace mají například více omezená práva v přístupu k systému a souborům. Vzhledem k tomu, že se počítá s tím, že tyto aplikace mohou být i na přenosných zařízeních (tablety mají operační systém Windows, ne Windows Phone), tak navíc tyto aplikace podporují například práci s orientací zařízení (zobrazení aplikace v režimu *portrait*, nebo *landscape*).

7.2.1 Binding

Ve WPF jsou data s View svázána pomocí takzvaného *bindingu*. Vzhledem k tomu, že ViewModel nemá informace o View, tak se toto spojení nastavuje ve View. při změně ViewModelu se pak vyvolá událost, ke které se View přihlásilo. Pokud dojde ke změně atributu z View, je upravena *přímo* vlastnost ViewModelu. Ve WPF existuje pět základních druhů bindingu [6]:

- TwoWay - pokud se aktualizuje ViewModel, je o změně informováno View. Pokud se aktualizuje View, je změna poslána do ViewModelu.

- OneWay - pokud se aktualizuje ViewModel, je o změně informováno View, ale změny provedené ve View se nepřenesou do ViewModelu.
- OneTime - jde o speciální typ OneWay bindingu, kde se hodnota z ViewModelu načte pouze při startu, nebo při změně datového kontextu.
- OneWayToSource - pokud se aktualizuje View, je změna poslána do ViewModelu, o změnách ViewModelu není View informováno.
- Default - tento typ nastaví výchozí nastavení dané kontrolky. Některé, například *TextBox*, mají jako výchozí nastavení TwoWay. Jiné potom mohou mít výchozí například OneWay.

7.2.2 UpdateSourceTrigger

Další důležitou vlastností spojenou s bindingem je nastavení, kdy se má provést aktualizace dat z View do ViewModelu. Toto nastavení se provádí pomocí vlastnosti UpdateSourceTrigger, která má následující možnosti [7]:

- PropertyChanged - změna View je do ViewModelu přenesena okamžitě.
- LostFocus - změna View je do ViewModelu přenesena až poté, co přestane být daná kontrolka aktivní.
- Default - nastaví výchozí typ dané kontrolky. Pro většinu kontrolek je to PropertyChanged, ale například pro *TextBox* je výchozí typ LostFocus, takže je celý text odeslán do ViewModelu až po dopsání a opuštění kontrolky.
- Explicit - změna View se projeví až po požadavku na aktualizaci dat z kódu. Toto nastavení lze použít například v případě, kdy se mají změny formuláře projevit až po kliknutí na potvrzovací tlačítko.

Vlastnost UpdateSourceTrigger má smysl nastavovat pouze v případě, že je vlastnost Binding nastavena tak, aby View mohlo měnit hodnotu ve ViewModelu. Například pro kontrolku *TextBlock*, která pouze zobrazuje text, který se nedá změnit, nemá smysl tuto vlastnost nastavovat, jelikož Binding bude s největší pravděpodobností typu OneWay, nebo OneTime.

7.2.3 Converter

S bindingem je také spojena další vlastnost, díky které lze definovat speciální převod hodnot. Jde o takzvaný *converter* a binding obsahuje stejnojmennou vlastnost Converter. Tento převodník je zpravidla klasická třída, která implementuje požadované rozhraní *IValueConverter*. Converter provede potřebnou konverzi zdrojových dat tak, aby se dali lépe zobrazit uživateli, a následně se dají tato data převést zpět tak, aby jim rozuměl ViewModel. Klasický converter slouží k převodu jedné hodnoty na jinou, ale pomocí implementace rozhraní *IMultiValueConverter* lze převést několik hodnot na jednu. V knize [1] je na stránce 79 mimo jiné napsáno, že veškeré výpočty mají probíhat mimo View. Jak bylo napsáno v kapitole 7.1.2, má View obsahovat pouze kód související se zobrazením informací. Converter se ve View používá z toho důvodu, aby například logickou pravdivostní hodnotu (True/False), převedl na viditelnost nějakého prvku. Je proto třeba dbát na to, aby se zde neobjevoval funkční kód, který by měl náležet spíše do ViewModelu, nebo Modelu. Converter naopak při správném použití nabízí možnost ještě více odstínit View od ViewModelu tím, že není třeba do ViewModelu zanášet vlastnosti, které jsou pouze pro View. Například zmíněná viditelnost je tak ve ViewModelu uložena ve vlastnosti typu *bool*, kterou View následně reprezentuje například jako zaškrtávací políčko, konkrétní barva nějaké kontrolky, nebo její viditelnost.

7.2.4 INotifyPropertyChanged

V jazyce C# není implicitně nic o změně vlastnosti instance třídy informováno. Pokud má být View aktuální i po aktualizaci ViewModelu, je třeba, aby se View o změně stavu dozvědělo. Rozhraní *INotifyPropertyChanged* obsahuje událost PropertyChanged, jejímž vyvoláním se View informuje o tom, že je třeba aktualizovat hodnotu dané vlastnosti. Informace o tom, která vlastnost má být aktualizována, je uložena v textovém řetězci. Nevýhodou tohoto řešení je, že není skutečný název vlastnosti s touto hodnotou nijak svázán. Pokud se změní název vlastnosti, je třeba změnit současně i textový řetězec. Stejně tak je tento systém náchylný na překlepy, protože se jedná o běžný textový řetězec, není tento text nijak porovnáván se skutečnými názvy vlastností. Tuto nevýhodu lze odstranit pomocí

klíčového slova *nameof*, které je novinkou v jazyce C# 6.0. Toto klíčové slovo lze použít přímo s nějakým typem, či proměnnou, jejíž jméno má být vráceno v textovém řetězci. Díky tomu, že textový řetězec generuje přímo kód a není zapsaný napevno, je při změně názvu přejmenováno i toto použití a název je tak vždy vrácen správně. Zároveň samozřejmě funguje i kontrola proti překlepům. Daná metoda nebyla představena pouze proto, aby se dalo lépe pracovat s tímto rozhraním. Její použití lze nalézt i například k přesnějšímu popisu při vyvolávání výjimek, práci se soubory json, nebo při záznamu chodu aplikace.

7.2.5 ICommand

Zatím zde byla popsána pouze provázanost dat mezi ViewModelem a View. Pokud uživatel zapíše text do pole pro text, je při správném použití bindingu tento text přenesen do ViewModelu. Pokud ViewModel načte z Modelu, že má být nějaká informace použita, je následně automaticky aktualizováno View například zaškrtnutím *CheckBoxu*. Následující text je věnován rozhraní, díky kterému jsou obslouženy další interakce uživatele s View. Například kliknutí na tlačítko.

Pokud k instanci dané kontrolky (například tlačítka) existuje přímý přístup, můžeme se přihlásit k celé řadě jeho událostí, které poskytuje (například stisknutí). Ve WPF toho lze dosáhnout tím, že se k této události přihlásíme v code-behind daného View. Pokud má obsluha této události vliv pouze na vzhled, pak je toto řešení správné i při použití návrhového vzoru MVVM (viz kapitola 7.1.2). Pokud však tato událost mění stav aplikace (například uložení formuláře do souboru), je třeba tuto událost obsloužit ve ViewModelu. K tomu slouží implementace rozhraní ICommand. Vlastnosti ViewModelu, které implementují rozhraní ICommand se poté pomocí bindingu propojí s jednotlivými prvky View. Tímto způsobem lze získávat informace o interakci uživatele s View ve ViewModelu bez nutné znalosti o konkrétním vzhledu. Implementace rozhraní ICommand obsahuje nejen metodu Execute, která je zavolána, když je příkaz volán, ale obsahuje též metodu CanExecute a událost CanExecuteChanged. Metoda CanExecute informuje o tom, zda se dá příkaz v současném stavu použít a událost CanExecuteChanged informuje o tom, že je třeba aktualizovat informaci, zda příkaz použít lze, nebo ne. Pokud je příkaz pomocí bindingu propojen například se stiskem tlačítka a daný příkaz nelze provést, stane

se tlačítko automaticky neaktivní (toto je výchozí nastavení chování). Jakmile je možné příkaz provést, stane se tlačítko automaticky aktivním. Programátor dokáže ovlivnit, za jakých podmínek má být možné příkaz provést. Pokud se bude jednat například o tlačítko pro odeslání formuláře, je možné povolit příkaz k odeslání až po řádném vyplnění celého formuláře. Tlačítko tak bude automaticky neaktivní, pokud bude některá část formuláře vyplněna špatně.

7.3 Zabezpečené přihlášení

Výstupem této bakalářské práce má být aplikace pro operační systém Windows, která může (ale nemusí) být připojena k síti Internet. Aplikace má podporovat několik různých uživatelských rolí podle toho, jak velké možnosti nastavení má uživatel získat. Pro přístup do aplikace bude vytvořeno několik základních účtů, jejichž informace budou uloženy lokálně. Vzhledem k tomu, že budou lokálně uloženy i veškeré soubory aplikace, nelze bezpečnost aplikace zaručit, protože se pomocí různých nástrojů zpětného inženýrství dá ze souborů aplikace vytvořit opět čitelné zdrojové kódy. Příklady takových dekompilátorů jsou například .NET Reflector[1], ILSpy[2], nebo dotPeek[3]. Z takto získaných zdrojových kódů lze následně zjistit způsob, jakým jsou přihlašovací údaje uloženy. Zároveň jde zjistit, kde jsou tyto informace uloženy.

Pokud budou přihlašovací údaje uloženy v lokální databázi, je třeba, aby měla aplikace někde uloženy přihlašovací údaje k této databázi. Opět lze pomocí dekompilátorů přímo zjistit přihlašovací údaje (pokud jsou uloženy v kódu programu), nebo zjistit umístění, kde se tyto informace nacházejí. S přihlašovacími údaji pak lze nejen data z databáze přečíst, ale i upravit - znamená to, že není třeba provádět jakékoliv útoky na uhádnutí hesla. Stačí pouze přepsat uložené heslo v

[1] http://www.red-gate.com/products/dotnet-development/reflector/

[2] http://ilspy.net/

[3] https://www.jetbrains.com/decompiler/

databázi. Budou-li data uložena v souboru, pak opět není problém tento soubor zobrazit a editovat.

Mezi základy ukládání hesel patří, že se heslo nikdy neukládá ve své původní podobě, ale je uložen pouze *hash* a případná kryptografická sůl[4]. Vzhledem k tomu, že hashovací funkce je pouze jednosměrná, nelze z uložených dat heslo nikdy zrekonstruovat. Pokud si však nechá útočník daným algoritmem vygenerovat hash ke svému heslu a má-li přístup k úložišti přihlašovacích údajů, je velmi jednoduché hash nějakého uživatele přepsat hashem, ke kterému zná útočník heslo.

Vzhledem k uvedeným skutečnostem lze tedy pouze zvýšit požadavky na útočníkovi znalosti a schopnosti. Úspěšné prolomení přístupu k aplikaci nelze vyloučit.

Po diskuzi se zadavatelem byl domluven stupeň zabezpečení, který je dostatečně silný a zároveň příliš neprodražuje vývoj aplikace.

7.3.1 Data Protection Application Programming Interface (DPAPI)

Rozhraní Data Protection Application Programming Interface (DPAPI) splňuje požadavek na minimální verzi operačního systému, kterou má aplikace podporovat [25]. .NET Framework obsahuje statickou třídu *ProtectedData*, pomocí které lze jednoduše DPAPI použít pomocí dvou statických metod použít [18]. Výhodou tohoto rozhraní je mimo jiné i to, že je programátor oproštěn od uchovávání kryptovacího klíče a nelze tak tuto informaci získat při dekompilaci zdrojových kódů aplikace. Nevýhodou tohoto rozhraní je pak to, že ke kryptování dat používá údaje aktuálně přihlášeného uživatele. To je problém u počítačů s více uživatelskými účty, které by měli mít přístup k aplikaci. Vzhledem k požadavku na vytvoření výchozích účtů je to pak problém obecný, jelikož by po zakryptování výchozích účtů na počítači vývojáře aplikace, nebylo možné tato data na jiném počítači správně interpretovat.

[4] Kryptografická sůl se přidává k hashovanému výrazu, který se tak stává obtížněji zjistitelný slovníkovým útokem.

7.3.2 Password-Based Key Derivation Function 2 (PBKDF2)

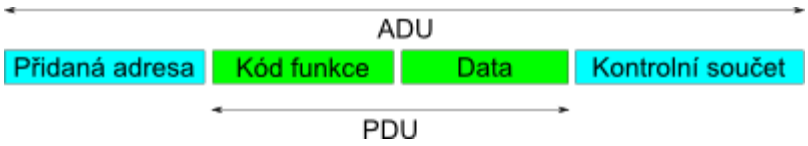
Jedná se o rozšířenou funkci, která se velmi často používá k převodu hesla na hash. Tuto funkci využívá i DPAPI, zmíněné v kapitole 7.3.1. Představila ji firma RSA Laboratories ve svém standardu PKCS #5 (Public-Key Cryptography Standards). Jedná se výpočetně náročné operace založené na iteracích pseudonáhodných funkcí. Díky velké náročnosti na výpočetní výkon, je využití této funkce výhodné k částečné ochraně před tak zvanými brute-force útoky, kterým díky tomu trvá delší dobu, než naleznou správné heslo. Pro uživatele, který funkci použije za normálních okolností po každé pouze jednou (při přihlášení), není přitom běh funkce natolik dlouhý, aby ho nějak rušil při práci.

Funkce jako argumenty přijímá původní heslo, kryptografickou sůl, počet iterací a požadovanou délku nového hesla (délku výsledného hashe). Počet iterací v podstatě udává, jak bude časově náročné získat správné heslo pomocí brute-force útoku. Čím větší počet iterací, tím více času bude třeba pro každý běh. Jak bylo zveřejněno v [5], lze při základním nastavení PBKDF2 vynechat 50% časově náročných operací a některé části je možné vypočítat předem. Autoři nicméně uvádějí, že je možné tento nedostatek částečně odstranit tím, že je zvolen dostatečně velký počet iterací. Podle [24] je ideální počet iterací je takový, že běh funkce trvá nejvyšší možný čas, který je uživatel ochotný akceptovat. Vzhledem k tomu, že není deklarována sestava hardware, na které má tato aplikace běžet, nelze určit obecně nějaký počet iterací, který bude představovat daný časový běh. V této publikaci je dále doporučeno minimálně 1000 iterací s tím, že pro extrémně kritické klíče, výkonné systémy, nebo systémy, kde není kritická odezva na uživatele, je možné za adekvátní hodnotu považovat 10 milionů iterací.

V dokumentu [24] je rovněž zmíněna důležitost dostatečně velké kryptografické soli. Její velikost definuje, kolik různých hashů lze získat při jednom heslu pouze tím, že se budou měnit hodnoty v bitech kryptografické soli. Jde o $2^{pocetBitu}$, kde *pocetBitu* je rovno počtu bitů použité soli. Doporučená velikost je minimálně 128 bitů (16 Bytů).

7.4 Modbus

Modbus je komunikační protokol aplikační vrstvy (podle OSI modelu), který poskytuje komunikaci typu klient-server [14]. Modbus byl vytvořen firmou Modicon v roce 1979 [26]. Podle této firmy také získal protokol svůj název (MODicon BUS). Tento protokol umožňuje komunikaci přes různé sběrnice a sítě. Například přes RS232, RS485, nebo TCP/IP. Protokol Modbus definuje jednoduchou Protocol Data Unit (PDU). Jde o konstrukci vycházející z protokolu, která je nezávislá na konkrétní komunikační vrstvě - patří sem kód funkce a data. V závislosti na komunikační vrstvě je možné PDU rozšířit ještě o adresu, případně kontrolní součet a vytvoří Application Data Unit (ADU). [13]



Obrázek 4 Obsah PDU a ADU v protokolu Modbus. Převzato a upraveno z [13].

Modbus definuje několik základních kódů funkcí. Patří sem například čtení, či zápis registrů. Je možné vytvořit i vlastní uživatelské funkce. Jejich kódy pak mohou nabývat hodnot 65 - 72, nebo 100 - 110. Ostatní kódy funkcí jsou rezervovány pro veřejné funkce a není je proto možné použít.

7.5 Lokalizace

Lokalizace aplikace se používá pro změnu jazyka, kterým aplikace komunikuje s uživatelem. Pokud jsou jazykově závislé objekty odděleny od kódu aplikace, je možné použít jednu aplikaci globálně a pouze vhodně přepínat zdroje s těmito objekty. Pokud by tyto objekty musely být pevně vepsány ve zdrojových souborech s kódem aplikace, musela by pro každý jazyk existovat jiná aplikace. Každá změna v programu by tak musela být provedena ve všech verzích aplikace. Navíc by bylo značně obtížné aplikaci překládat do nových jazyků. Jazyková nastavení se nemusejí týkat pouze jazyka, ale mohou zahrnovat také různá formátování.

7.5.1 Lokalizace v .NET Frameworku

.NET Framework podporuje Unicode na všech vrstvách a lze proto s Unicode znaky pracovat bez konverze jak v uživatelském rozhraní, tak například v přístupu k databázi. Základní typ souboru pro zdroje textů jsou Resx soubory. Jde o XML soubory se specifickou strukturou. Pro rozlišení konkrétní kultury se používá pěti-znakové označení. První dva znaky určují světový jazyk, následuje oddělovací pomlčka a poslední dva znaky určují konkrétní stát (například pro angličtinu Velké Británie jde o označení *en-GB*). Rozdělení nejen podle jazyka, ale i podle státu je nutné ze dvou důvodů. První je, že různé státy, mající stejný světový jazyk, používají některá slova v pozměněné formě, případně používají úplně jiná slova pro stejnou věc. Protože se ale jedná stále o stejný jazyk, daly by se tyto rozdíly v počítačovém programu pominout. Důležitější je druhý důvod. Pomocí tohoto pětipísmenného označení se nerozlišuje pouze jazyk, ale také kulturní nastavení - mezi ně patří například měna, psaní desetinných a tisícových oddělovačů, psaní data a času a podobně. Za předpokladu, že je důležité přeložit pouze texty do daného jazyka s tím, že není důležité určovat konkrétní stát, nebo tento konkrétní stát není znám, je možné použít pouze dvouznakové označení určující jazyk (například pro angličtinu *en*).

.NET Framework disponuje dvojím nastavením kultury daného vlákna programu. Jedno nastavení určuje jazyk, jehož zdrojové texty se mají vyhledat a použít pro překlad textů a podobně. Toto nastavení lze provést pomocí vlastnosti *UICulture*. Vlastnost *Culture* naproti tomu nastavuje správné formátování čísel, dat a dalších kulturně specifických formátů. Označení přitom nelze obecně zjednodušit tak, že by vlastnost UICulture vždy přijímala pouze první dva znaky a vlastnost Culture pouze poslední dva znaky, jelikož existují státy, které mají více úředních jazyků, a každý jazyk může mít vlastní formátování. Jako příklad lze použít Kanadu a formátování času. Kanada má jako úřední jazyk angličtinu (*en-CA*) a francouzštinu (*fr-CA*). Zatímco angličtina má 12-hodinový formát času rozlišující dopoledne a odpoledne pomocí znaků AM/PM, francouzština využívá 24-hodinový formát času stejně, jako například čeština. Je tedy rozdíl, zda se do vlastnosti Culture nahraje *en-CA*, nebo *fr-CA*. Díky tomu, že .NET Framework obsahuje dvě různá nastavení kultury, nemusí

nutně nastavení jazyka korespondovat s nastavením formátování. .NET Framework 4.5 přináší dvě nové vlastnosti nastavení kultury *DefaultThreadCurrentCulture* a *DefaultThreadCurrentUICulture*. Těmito vlastnostmi se provede výchozí nastavení kultury pro nově vytvořená vlákna, což značně zjednodušuje práci s programy, které mají podporovat různé jazyky. Vzhledem k tomu, že lze k tvorbě této aplikace použít maximálně .NET Framework verze 4.0 (viz kapitola 6.2), je třeba nastavit kulturu pro každé nové vlákno zvlášť.

7.5.2 Lokalizace ve WPF

WPF podporuje hlavně dva druhy lokalizace. Jedna je založená na XAML a druhá na Resx souborech. U obou typů platí, že mají obvykle jednu výchozí lokalizaci a na ostatní lze přepnout. Díky této výchozí lokalizaci je, jak je popsáno v [22], možné využívat funkce *ResourceFallback*, která se snaží nalézt nenalezený objekt v dalších lokalizacích.

- Resx soubory se nepoužívají k lokalizaci pouze v technologii WPF, ale jde o klasický způsob lokalizace aplikací psaných pod .NET Frameworkem. S tím je spojena velká podpora tohoto typu souborů jak samotným .NET Frameworkem, tak Visual Studiem. Jde o XML soubory, které jsou zkompilovány do binárních souborů a mohou být volány z kódu aplikace. Pokud je použita konkrétní kultura i s určením státu a daný objekt v ní není nalezen, je pomocí funkce *ResourceFallback* vyhledán objekt v obecné lokalizaci daného jazyka. Pokud není nalezen ani zde, je objekt vyhledán ve výchozí lokalizaci. Díky tomuto systému je možné lokalizovat celou aplikaci do konkrétního jazyka nezávisle na státu a v lokalizacích pro konkrétní národy pak definovat pouze výrazy, které se rozcházejí s výchozími výrazy daného jazyka. Nevýhodou využití Resx souborů je to, že soubory s lokalizací jsou ve formátu XML, což nemusí vyhovovat překladatelům, kteří nemusí mít potřebné znalosti k práci s tímto druhem souboru. Tento problém se tak musí řešit buď exportem dat do textového souboru, nebo speciální aplikací, která zvládá soubory přečíst a vhodně editovat.

- XAML lokalizace je založena na myšlence, že programátor vytvoří vzhled pomocí XAML souboru se všemi texty jednoho jazyka bez nutnosti používat speciální syntaxi. Společnost Microsoft vydala nástroj pro příkazový řádek zvaný *LocBaml*, který poté vyexportuje veškerý lokalizovatelný statický obsah do textového CSV souboru. Tento soubor je možné lokalizovat pro každou požadovanou kulturu a následně mohou být všechny tyto zdroje opět pomocí zmíněného nástroje složeny a zkompilovány do kulturně závislých satelitních sestavení (BAML). LocBaml je klíčovým programem pro tento způsob lokalizace, přesto byl vydán pouze jako ukázková aplikace při představení WPF technologie a je bez jakékoliv podpory. Celý tento proces lze použít pouze pro statický obsah v XAML souborech a nelze nijak ovlivnit způsob, jakým jsou jednotlivé zdroje nahrávány. Zároveň je tento způsob vhodný spíše pro aplikace, které se často nemění. Při změně aplikace je totiž opět nutné texty vyexportovat, přeložit a znovu je importovat. Při exportu textů je navíc vytvořen zcela nový soubor s texty, takže texty, které již byly přeloženy, je třeba do tohoto souboru překopírovat. ResourceFallback lze u XAML lokalizace využít pouze omezeně, jelikož všechny BAML verze obsahují všechny objekty i v případě, že je třeba pro jiný národ využívající stejný jazyk změnit jen jedno slovo. Výhodou tohoto přístupu je efektivita načítání. Protože každá kultura obsahuje svou vlastní BAML verzi daného souboru, je tento soubor načten najednou, místo toho, aby se načítal každý lokalizovaný objekt zvlášť, jako je tomu u Resx souborů.

Lokalizace pomocí Resx souborů je pro programátora lépe řiditelná, než lokalizace provedená pomocí LocBaml nástroje. Na druhou stranu je její implementace z počátku náročnější a programátor musí používat klíčová slova z Resx souboru, která budou za běhu aplikace nahrazena skutečnou hodnotou. Při použití LocBaml nástroje programátor vytvoří klasický XAML soubor s lokalizovaným obsahem a lokalizace do ostatním jazyků se provádí až po vytvoření programu. Lokalizace využívající XAML soubory je tak vlastně mezistupeň, mezi jednou aplikací přepínající lokalizované objekty a mnoha aplikacemi, kde aplikace obsahují stejný kód, ale každá je napevno v kódu lokalizovaná do jiného jazyka. To mimo jiné

znamená rychlejší načítání, ale zároveň při každé změně originálního XAML souboru vytvoření nového BAML souboru, který je třeba znovu celý přeložit, případně ručně upravit exportovaný CSV soubor z LocBaml tak, aby odpovídal změnám provedeným v originálním souboru. Kombinace obou popsaných metod je možná, ale nástrojem LocBaml jsou při lokalizaci vyžadovány další kroky a záleží proto na konkrétní situaci, zda zvýšená náročnost na tvorbu lokalizace odpovídá výhodám, které z tohoto spojení mohou plynout.

Vzhledem k nedostatečné dokumentaci lokalizace WPF aplikací, existuje několik různých způsobů (převážně využívajících Resx soubory) jak implementovat lokalizaci do WPF aplikace. Kromě dvou výše uvedených způsobů existují i jiné, které nevyužívají ani jeden typ souborů. V [20] je například představen způsob, který využívá obyčejné XML soubory, které ale na rozdíl od Resx souborů není třeba kompilovat. Při správném použití tak lze do programu dodávat lokalizace bez nutnosti nového překladu programu. Vzhledem k tomu, že tyto XML soubory nejsou nijak svázány s kódem aplikace a jednotlivé texty jsou odkazovány pomocí prostých textových referencí, je tento způsob velmi náchylný k překlepům.

8 Praktická část

V této kapitole bude rozepsáno řešení některých požadavků zadavatele, u nichž nebylo specifikováno použití konkrétní technologie. Veškerá řešení byla se zadavatelem konzultována.

8.1 Lokalizace

Kapitola 7.5 představila několik způsobů lokalizace programu. Se zadavatelem byla nakonec zvolena varianta použití Resx souborů. Daný způsob mimo jiné umožňuje využít, díky kvalitní podpoře Resx souborů ve vývojovém prostředí Vusial Studio 2015, při psaní kódu kontrolu překlepů v odkazování na požadovaný text.

Využití Resx souborů s sebou přináší nevýhodu nesnadného překladu do dalších jazyků různými překladateli, protože se jedná o XML soubor. Jeho přímá editace je tak nepřehledná. Kvůli možnosti poskytnout překladatelům co nejuniverzálnější způsob editace souborů, jsou zdrojové texty uloženy v obyčejném textovém souboru ve formátu *reference=text*, kde *reference* představuje unikátní textový řetězec, pomocí kterého je na daný text odkazováno v kódu aplikace a *text* je vlastní přeložený text. Do tohoto souboru lze vkládat i komentáře, které jsou při převodu na Resx soubor přeskočeny. Komentář je uvozen středníkem. Příklad textového souboru je zobrazen ve výpisu Ukázka kódu 5.

Ukázka kódu 5 Příklad zdrojového textové souboru s překladem.

```
;Toto je komentář, který bude při převodu souboru přeskočen
LogoTitle=Vývoj, výroba el. systémů pro měření regulací
Login=Uživatelské jméno:
LoginPassword=Heslo:
```

Takto vytvořený soubor lze pomocí nástroje ResGen převést na Resx soubor. Nástroj ResGen je součástí .NET Frameworku. Nejedná se tedy o nástroj třetí strany, ale o aplikaci, která byla vytvořena přímo pro tyto účely autory frameworku. Samotný převod souboru se provádí příkazu v příkazové řádce. Pro zautomatizování byl vytvořen dávkový soubor, který je zobrazen ve výpisu Ukázka kódu 6.

Ukázka kódu 6 Obsah dávkového souboru pro převod souboru.

```
@echo off
```

```
"C:\Program Files (x86)\Microsoft SDKs\Windows\v8.0A\bin\NETFX 4.0
Tools\ResGen" Resources.cs-CZ.txt ..\Resources.cs-CZ.resx
pause
```

První část příkazu je cesta k nainstalovanému nástroji ResGen. Vzhledem k mezerám v cestě je třeba celou cestu uzavřít do uvozovek. Cesta není zcela univerzální a může se lišit podle toho, kde je daný nástroj nainstalován. Cesta uvedená ve výpisu je výchozí pro 64-bitový operační systém. Nástroj posléze přijme zdrojový textový soubor (Resources.cs-CZ.txt), který je předpokládán ve stejném umístění, jako dávkový soubor, a výstupní soubor uloží do složky, která je nadřazená právě aktuální složce do souboru Resx (Resources.cs-CZ.resx). Pro úplnost je třeba doplnit, že ve skutečném dávkovém souboru je celý příkaz uveden pouze na jednom řádku, což je zde znázorněno pouze jedním číslem řádku a celý soubor tak má podle čísel skutečně pouze tři řádky.

8.2 Zabezpečené přihlášení

Vzhledem k tomu, že má mít nová instalace aplikace přednastaveno několik různých účtů, bylo rozhraní DPAPI (kapitola 7.3.1) odmítnuto. Místo něho bude v aplikaci přihlašování prováděno pomocí funkce PBKDF2 (kapitola 7.3.2). Výběr konkrétní KDF (Key Derivation Function) byl zvolen mimo jiné i podle doporučení organizace NIST (National Institute of Standards and Technology). NIST funkci PBKDF2 doporučuje ve své zprávě o uchovávání hashovaných hesel [24] za předpokladu, že bude jako vstupní parametr funkce použit co možná největší počet iterací.

8.2.1 Počet iterací

Test byl proveden pro počty iterací 1000, 5000, 10000, 50000 a 100000. Minimální doporučený počet iterací byl ve zmíněném dokumentu 1000, proto tato hodnota byla použita jako minimální hodnotou. Test byl proveden na dvou různých strojích s 64-bitovým operačním systém Windows 10. První použitý stroj byl osobní počítač s označením „počítač“ a následující konfigurací:

Procesor: Intel Core i5-3570K, 3,4 GHz

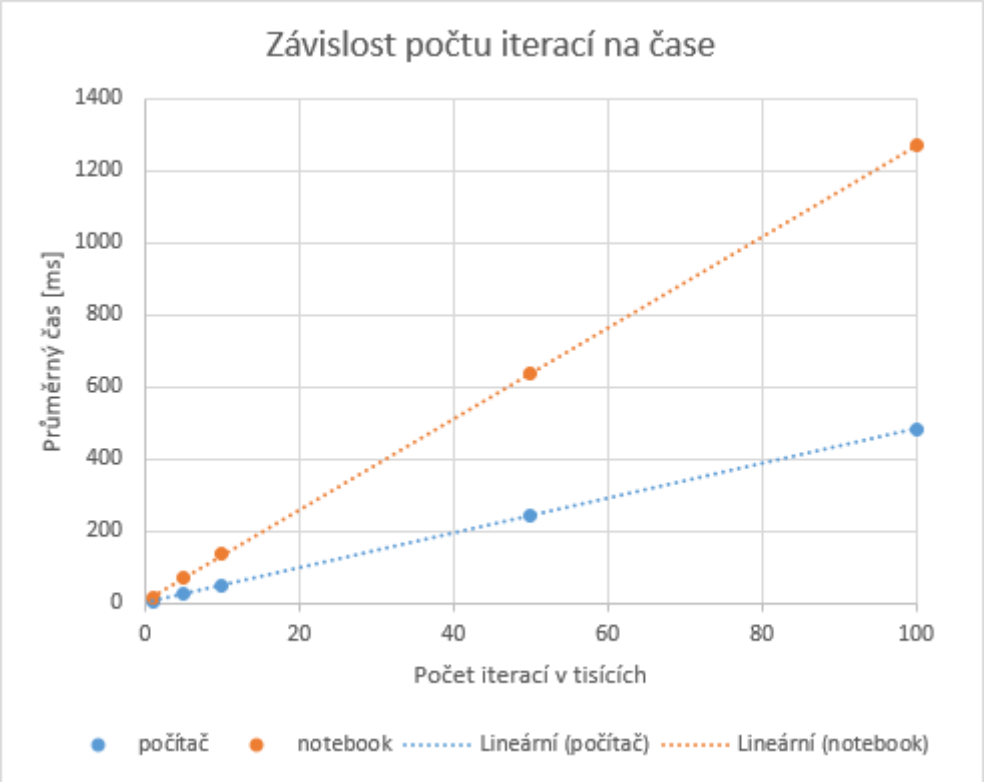
RAM: 8 GB

Druhý testovací stroj byl notebook DEL Latitude D630 pod označením „notebook“ s následující konfigurací:

Procesor: Intel Core 2 Duo T7500, 2,2 GHz

RAM: 4 GB

Každý počet iterací byl změřen celkem padesátkrát.



Obrázek 5 Zobrazení závislosti počtu iterací na čase. [zdroj: vlastní]

Z grafu na obrázku Obrázek 5 lze vyčíst, že vztah mezi počtem iterací a časem potřebným k výpočtu hashe, je lineární. Vzhledem k tomu, že průměrný čas pro běh jedné iterace by ani z konstrukce PBKDF2 neměl být závislý na celkovém počtu iterací, lze z naměřených hodnot pro dané stroje interpolovat další hodnoty pro požadovaný počet iterací.

Po prezentaci výsledků zadavateli bylo dohodnuto, že výsledný počet iterací bude nastaven na 100000.

8.2.2 Velikost kryptografické soli

V kapitole 7.3.2 je napsáno, že minimální velikost kryptografické soli by měla být 16 Bytů. Toto doporučení je také v aplikaci dodrženo, protože to znamená, že pro každé heslo je pro slovníkový útok bez znalosti kryptografické soli třeba vypočítat celkem 2^{128} výsledných hashů.

8.2.3 Velikost výsledného hashe

Poslední parametr funkce je velikost výsledného hashe. Maximální velikost by neměla překročit velikost, kterou generuje hashovací algoritmus použitý v implementované verzi PBKDF2. Toto doporučení vychází z vnitřní funkčnosti funkce. Pokud je výsledná požadovaná velikost větší, než je výsledek hashovacího algoritmu, je nejprve vypočítán hash a následně jsou dopočítány bity do požadované velikosti pomocí získaného hashe. Útočníkovi tak stačí do požadované délky PBKDF2 zadat pouze délku, kterou vygeneruje použitý hashovací algoritmus a výsledek porovnávat s odpovídajícím počtem bitů hashe. Ve výpisu Ukázka kódu 7 jsou zobrazeny výsledné hashe, které se v zadání liší pouze v požadované výsledné velikosti.

Ukázka kódu 7 Výsledný hash PBKDF2-HMAC-SHA-1 pro 20 a 30 Bytů.

```
"Výsledný hash 20 Bytů"
0x46, 0x3c, 0x46, 0xb9, 0xd8, 0x0d, 0x36, 0x24, 0x41, 0xc4,
0x6f, 0xc6, 0xa1, 0x2d, 0x13, 0x56, 0x40, 0xf5, 0x07, 0x4f

"Výsledný hash 30 Bytů"
0x46, 0x3c, 0x46, 0xb9, 0xd8, 0x0d, 0x36, 0x24, 0x41, 0xc4,
0x6f, 0xc6, 0xa1, 0x2d, 0x13, 0x56, 0x40, 0xf5, 0x07, 0x4f,
0x42, 0x12, 0x8c, 0xe8, 0x66, 0xfa, 0x52, 0xdc, 0x36, 0x5e
```

Použitý SHA-1 algoritmus má výslednou délku 160 bitů (20 Bytů). Jak lze zjistit jednoduchým porovnáním, je prvních 20 Bytů obou hashů stejných (řádek číslo 2 je stejný se řádkem číslo 6 a řádek číslo 3 s řádkem číslo 7). Pokud by měl tedy program používající algoritmus SHA-1 nastavenou výslednou velikost na 30 Bytů, bude se jednat pouze o prodloužení doby pro běžného uživatele, nikoliv o zvětšení obtížnosti pro případného útočníka.

8.2.4 Třída Rfc2898DerivedBytes

.NET Framework obsahuje ve jmenném prostoru *System.Security.Cryptography* třídu *Rfc2898DeriveBytes*, která je implementací PBKDF2-HMAC-SHA-1. Tato třída bude využita v produkčním kódu. Velikost výsledného hashe bude nastavena na maximální doporučenou velikost - tedy 20 Bytů (viz kapitola 8.2.3).

Ukázka kódu 8 Metoda pro zadání nového hesla.

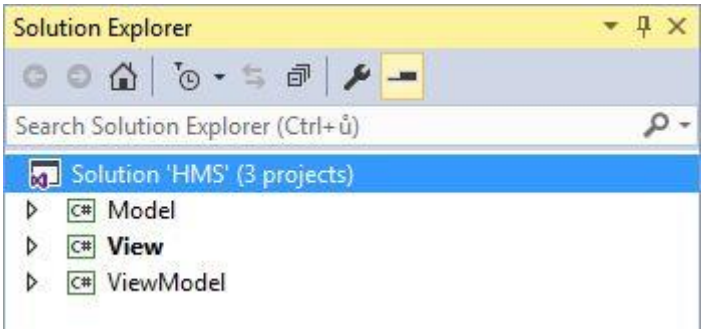
```
private const int SaltSize = 16;
private const int IterationCount = 100000;
private const int HashSize = 20;

public Password CreateNewPassword(string password)
{
      var deriveBytes = new Rfc2898DeriveBytes(password, SaltSize,
IterationCount);
      var derivedPassword = new Password();
      derivedPassword.Hash = deriveBytes.GetBytes(HashSize);
      derivedPassword.Salt = deriveBytes.Salt;
      return derivedPassword;
}
```

Ve výpisu Ukázka kódu 8 je zobrazena metoda pro odvození hashe a kryptografické soli ze zadaného hesla uživatele a také hodnoty konstant použitých v metodě. Testy pro vhodné nastavení počtu iterací byly prováděny pomocí drobné obměny této metody. Metoda vrací instanci třídy *Password*, která obsahuje vlastnosti Hash a Salt, do kterých jsou uloženy hodnoty získané po aplikování PBKDF2 na heslo zadané uživatelem. Na řádku číslo 7 je vytvoření nové instance třídy Rfc2898DeriveBytes. Třída se stará zároveň o vygenerování kryptografické soli v požadované velikosti (zde 16 Bytů).

8.3 MVVM

Jak je zobrazeno na obrázku Obrázek 6, celé řešení aplikace je ve Visual Studiu rozděleno do tří projektů, které kopírují jednotlivé části návrhového vzoru Model-View-ViewModel. Projekty Model a ViewModel jsou vytvořeny jako dll knihovny a lze je tak využít i pro případné další aplikace. Například aplikaci pro Windows Store. Projekt View je WPF aplikace, která představuje rozhraní pro uživatele.



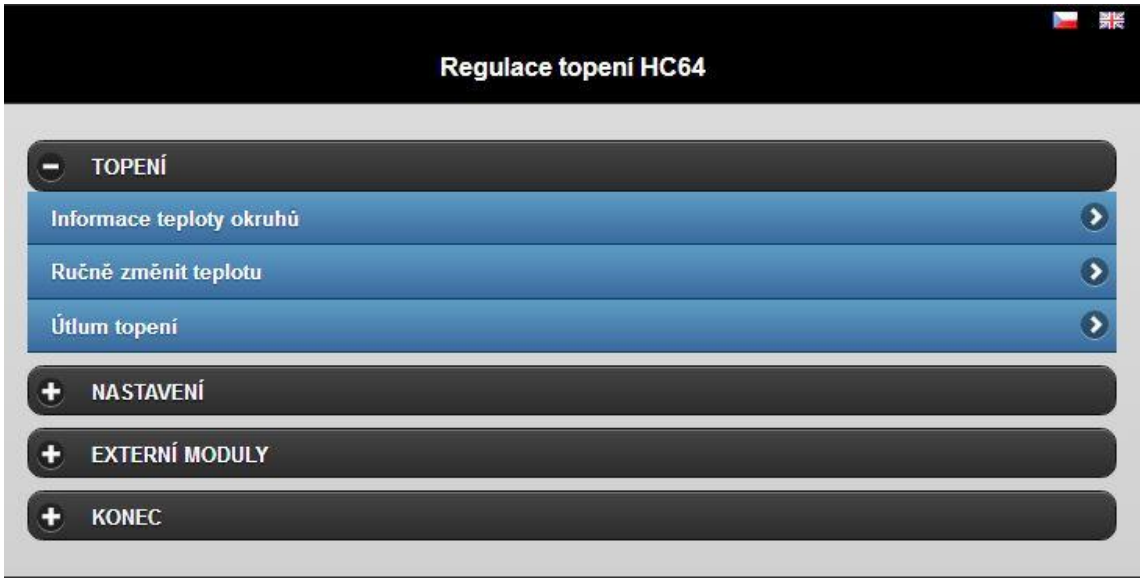
Obrázek 6 Základní struktura řešení aplikace [zdroj: vlastní]

Rozdělení na projekty též umožňuje lepší kontrolu nad tím, zda jsou reference mezi jednotlivými částmi v požadovaném směru (viz Obrázek 3). Pokud neexistuje vážný důvod, proč by to tak být nemělo, pak by měl projekt View obsahovat pouze referenci na projekt ViewModel a projekt ViewModel by měl mít referenci pouze na projekt Model. Projekt Model by pak neměl mít referenci ani na View, ani na ViewModel.

8.4 Vzhled

Jak je napsáno v kapitole 5, zadavatel jako jeden z požadavků vyspecifikoval vzhled podobný již vytvořenému webovému rozhraní, které bylo vytvořeno pomocí knihovny jQuery Mobile. Menu tohoto rozhraní vypadá tak, jak je zobrazeno na obrázku Obrázek 7. Celé webové rozhraní je vytvořeno tak, aby bylo snadno ovladatelné především z malých dotykových obrazovek (chytré telefony a tablety). Po výběru položky v menu, se zobrazí konkrétní obrazovka přes celou šířku okna (displeje) tak, jak je vidět na obrázku Obrázek 8. Tím je celá plochu využitá údaji, které chce v daném okamžiku uživatel vidět.

Vzhledem k tomu, že je aplikace tvořena pro obrazovky notebooků a stolních počítačů, bylo se zadavatelem dohodnuto, že bude v aplikaci zobrazeno jak menu, tak obsah vybrané položky zároveň.

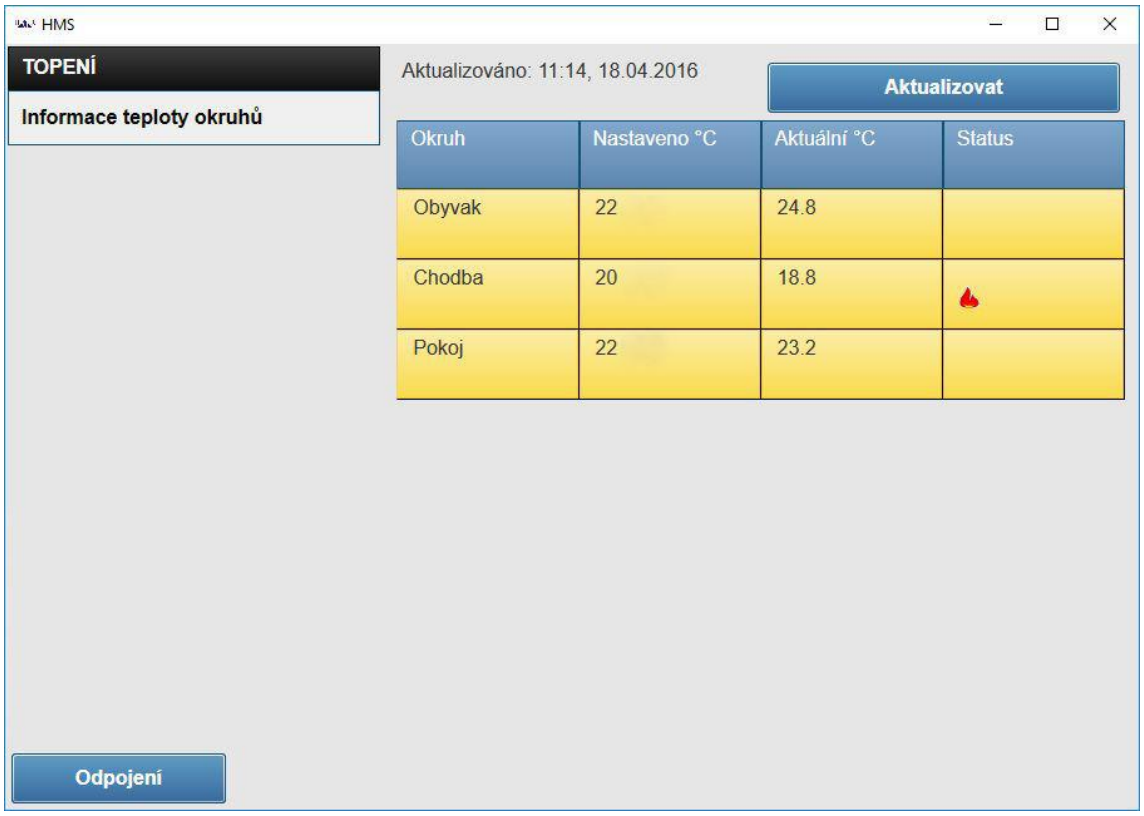


Obrázek 7 Menu webového rozhraní jednotky HC64 [zdroj: vlastní]

Obrázek 8 Informace o místnostech ve webovém rozhraní [zdroj: vlastní]

Menu je zobrazeno v levé části okna a detail konkrétní položky v pravé části. Vzhledem k podobnosti s webovým rozhraním, které je tvořeno tak, aby šlo ovládat prsty, je možné ovládat na dotykové obrazovce i tuto aplikaci. Aplikaci lze spustit na jakémkoliv zařízení s operačním systémem Windows XP a novějším. Velká část těchto zařízení přitom je vybavena dotykovou obrazovkou. Může se jednat například o *All In One* počítače, notebooky s dotykovou obrazovkou, nebo některé tablety. Možnost ovládat aplikaci pohodlně dotyky, je díky tomu velkou výhodou.

Obrazovka s informacemi o místnostech tak ve výsledné aplikaci vypadá tak, jak je znázorněno na obrázku Obrázek 9.



Obrázek 9 Zobrazení informací o místnostech v programu [zdroj: vlastní]

Toto rozložení je na širší obrazovce přehlednější a uživatel může měnit položky bez toho, aby se musel nejprve vrátit zpět domu a následně vybrat další položku.

9 Závěr

Cílem této práce bylo vytvořit komunikační rozhraní s řídící jednotkou vytápění HC64, které by bylo možné spustit na osobních počítačích s operačním systém Windows XP a novějším. Zadavatel zároveň vyspecifikoval způsob připojení k jednotce a zadal požadavek na vzhled aplikace. Mezi požadavky patřilo též využití programovacího jazyku C# a .NET Frameworku.

V úvodu práce bylo představeno několik způsobů řízení a regulace vytápění objektů různými zdroji tepla. Bylo zde také zdůrazněno, že některé zdroje tepla vyžadují specifický způsob regulace a tím pádem není možné jednoznačně rozhodnout, jaký způsob regulace je obecně nejvhodnější. Vzhledem k velkým rozdílům v tepelných ztrátách (u rodinných domů zvláště mezi novostavbami a staršími budovami), není možné určit obecně nejvhodnější způsob vytápění, protože v domech s větší tepelnou ztrátou může být ekonomicky výhodné použít jiný zdroj tepla, než u nízkoenergetických novostaveb.

Dále byly představeny požadavky zadavatele na novou aplikaci. Patřili mezi ně jak vzhledové, tak funkční, ale i technologické požadavky. Po jejich prostudování byly představeny technologie, které je nejlépe splňují. Jako nejvhodnější verze .NET Frameworku byla s ohledem na požadavek podpory operačního systému Windows XP zvolena verze 4.0, která je nejnovější verzí, jež daný operační systém podporuje. Byly zde popsány některé rozdíly mezi návrhovými vzory Model-View-ViewModel, Model-View-Presenter a Model-View-Controller.

Velká část práce byla věnována představení technologie Windows Presentation Foundation a návrhovému vzoru Model-View-ViewModel. Tato kombinace nejlépe splňovala požadavky na výsledný vzhled aplikace, ale i některé další funkční požadavky. Návrhový vzor Model-View-ViewModel lze uplatnit i v dalších technologiích, jako je například framework pro webové aplikace ZK, který je psaný v programovacím jazyku Java, nebo JavaScriptová knihovna Knockout.

Výsledek práce splňuje požadavky zadané zadavatelem. Kvůli některým omezujícím požadavkům, nebylo sice možné využít nejnovější verze některých dostupných

technologií, nicméně výsledkem je aplikace, která je vzhledově i funkčně stejná na všech operačních systémech Windows, počínaje Windows XP.

10 Seznam použité literatury

10.1 Literární zdroje

[1] HALL, Gary McLean. *Pro WPF and Silverlight MVVM: effective application development with Model-View-ViewModel.* New York: Distributed to the book trade worldwide by Springer Science Business Media, 2010, xiii, 257 p. Expert's voice in WPF. ISBN 14-302-3162-9.

[2] HORALEK, Josef a Vladimir SOBESLAV. Intelligent Heating Regulation. In: *Advanced Computer and Communication Engineering Technology*. Springer International Publishing Switzerland, 2016, s. 807-817. DOI: 10.1007/978-3-319-24584-3_68. ISSN 1876-1100. Dostupné také z: http://link.springer.com/10.1007/978-3-319-24584-3_68

[3] NATHAN, Adam a Daniel LEHENBAUER. *Windows presentation foundation unleashed*. Indianapolis, Ind.: Sams, c2007, xi, 638 p. ISBN 978-067-2328-916.

[4] SMITH, Josh. *Advanced MVVM*. Josh Smith, 2013. ISBN 0985784547.

[5] VISCONTI, Andrea, Simone BOSSI, Hany RAGAB a Alexandro CALÒ. On the Weaknesses of PBKDF2. In: *Cryptology and Network Security.* Switzerland: Springer International Publishing, 2015, s. 119-126. DOI: 10.1007/978-3-319-26823-1_9. ISSN 0302-9743. Dostupné také z: http://link.springer.com/10.1007/978-3-319-26823-1_9

10.2 Internetové zdroje

[6] Binding.Mode Property. In: *Binding.Mode Property* [online]. [cit. 2015-08-14]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.windows.data.binding.mode(v=vs.100).aspx

[7] Binding.UpdateSourceTrigger Property. In: *Binding.UpdateSourceTrigger Property* [online]. [cit. 2015-08-14]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.windows.data.binding.updatesourcetrigger(v=vs.100).aspx

[8] BJELLÅS, Sondre. No Code-Behind for MVVM. *Technology & Robotics* [online]. [cit. 2015-08-13]. Dostupné z: http://sondreb.com/blog/post/no-code-behind-for-mvvm.aspx

[9] Exception.HResult Property. In: *Exception.HResult Property* [online]. [cit. 2015-08-10]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.exception.hresult(v=vs.100).aspx

[10] Exception.HResult Property. In: *Exception.HResult Property* [online]. [cit. 2015-08-10]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.exception.hresult(v=vs.110).aspx

[11] Introduction to Model/View/ViewModel pattern for building WPF apps. *Tales from the Smart Client: John Gossman's observations on Avalon development* [online]. 2005 [cit. 2015-08-17]. Dostupné z: http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx

[12] Meet the Nest Thermostat. *Nest* [online]. Palo Alto, c2016 [cit. 2016-03-27]. Dostupné z: https://nest.com/thermostat/meet-nest-thermostat/

[13] *MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b3* [online]. 2012 [cit. 2015-08-10]. Dostupné z: http://www.modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf

[14] MODBUS PROTOCOL. In: *Modbus Specifications and Implementation Guides* [online]. [cit. 2015-08-10]. Dostupné z: http://www.modbus.org/specs.php

[15] *Msdn forums* [online]. [cit. 2015-08-13]. Dostupné z: https://social.msdn.microsoft.com/forums/vstudio/en-us/home

[16] *Produkty a systémy pro regulaci vytápění, chlazení a ovládání rolet* [online]. 2016 [cit. 2016-03-29]. Dostupné z: http://bmr.cz/index.php/ke-stazeni/bmr-2016-katalog-modry-pdf?download=95:bmr-2016-katalog-zeleny-regulace-vytapeni

[17] *Programmers* [online]. [cit. 2015-08-13]. Dostupné z: http://programmers.stackexchange.com/

[18] ProtectedData Class. In: *ProtectedData Class* [online]. [cit. 2015-08-10]. Dostupné z: https://msdn.microsoft.com/en-us/library/system.security.cryptography.protecteddata(v=vs.100).aspx

[19] RDE410/EH. *Siemens* [online]. Praha, 2014 [cit. 2016-03-29]. Dostupné z: http://w5.siemens.com/web/cz/cz/corporate/portal/home/produkty_a_sluzby/IBT/mereni_a_regulace/regulatory/programovatelne_termostaty/Pages/RDE410EH.aspx#

[20] SHAMAM, Tomer. WPF Localization – On-the-fly Language Selection. In: *Essential XAML: The Hitchhiker's Guide to XAML Apps* [online]. 2007, 06-Feb-2009 [cit. 2015-08-18]. Dostupné z: http://blogs.microsoft.co.il/tomershamam/2007/10/30/wpf-localization-on-the-fly-language-selection/

[21] *Stack Overflow* [online]. [cit. 2015-08-13]. Dostupné z: http://stackoverflow.com/

[22] STRAHL, Rick a LEROUX BUSTAMANTE. *WPF Localization Guidance* [online]. 2009 [cit. 2015-08-18]. Dostupné z: http://wpflocalization.codeplex.com/downloads/get/73227

[23] Supported client operating systems. In: *.NET Framework System Requirements* [online]. [cit. 2015-08-10]. Dostupné z: https://msdn.microsoft.com/en-us/library/8z6watww(v=VS.110).aspx

[24] TURAN, Meltem Sönmez, Elaine BARKER, William BURR a Lily CHEN. NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *Recommendation for Password-Based Key Derivation: Part 1: Storage Applications.* 2010. SP 800-132. Dostupné také z: csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf

[25] Windows Data Protection. In: *Windows Data Protection* [online]. 2001 [cit. 2015-08-10]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms995355.aspx

[26] What is Modbus? In: *Simply Modbus - About Modbus* [online]. [cit. 2015-08-10]. Dostupné z: http://www.simplymodbus.ca/faq.htm

[27] What's new in the .NET Framework 4.5: Core new features and improvements. In: *What's New in the .NET Framework* [online]. [cit. 2015-08-10]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms171868(v=VS.110).aspx

11 Seznam obrázků

12 Seznam ukázek kódů

13 Přílohy

1) CD obsahující spustitelný program, kód programu a tuto práci.