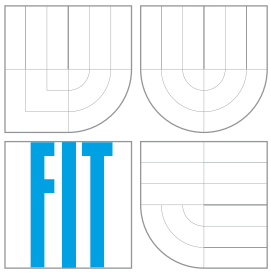
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KLIENT VIRTUÁLNÍ ČEKÁRNY PRO POSKYTOVATELE SLUŽEB

VIRTUAL WAITING ROOM CLIENT FOR SERVICE PROVIDER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAROSLAV HOLCMAN
AUTHOR

VEDOUCÍ PRÁCE — MARTIN KOLÁŘ, M.Sc.
SUPERVISOR

BRNO 2016

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Holcman Jaroslav**
Obor: Informační technologie
Téma: **Klient virtuální čekárny pro poskytovatele služeb**
Virtual Waiting Room Client for Service Provider
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s potřebami tvorby "virtuálních čekáren" například pro lékařské ordinace, půjčovny, úřady apod. se zaměřením na stranu poskytovatelů služeb.
2. Vytvořte koncepci klienta "virtuální čekárny" jako části klient-server aplikace s mobilními klienty tak, aby poskytovatel služeb měl dobrý přehled o stavu čekárny a mohl vybírat zákazníky.
3. Diskutujte vlastnosti vytvořené koncepce klienta pro poskytovatele služeb a rozeberte výhody a nevýhody.
4. Implementujte klienta čekárny a demonstrujte její funkčnost na vhodném příkladě.
5. Diskutujte dosažené výsledky a možnosti pokračování práce.

Literatura:

- Dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3 zadání

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: **Kolář Martin, M.Sc.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Cílem této práce je navrhnout a implementovat klientskou aplikaci s grafickým uživatelským rozhraním pro poskytovatele služeb v rámci projektu Virtuální čekárna. Grafické uživatelské rozhraní umožňuje poskytovateli pracovat s frontami zákazníků, objednávat zákazníky na určené termíny a celkově pracovat s daty, která jsou uložena na serveru. Uživatelské rozhraní je vytvořeno pomocí technologie WPF z .NET Frameworku od firmy Microsoft. Text této práce popisuje projekt Virtuální čekárny jako celku, dále podrobněji klienta pro poskytovatele služeb a také technologie, které byly použity pro vytvoření tohoto klienta.

Abstract

The goal of this thesis is to develop and implement client application with graphical user interface for service providers as a part of the project Virtual waiting room. Graphical user interface enables the provider to manage queues of costumers, appoint costumers for specific terms and work with data stored on the server. The user interface is developed using the WPF technology from the .NET Framework, created by Microsoft. This thesis describes the Waiting room project, the client for service provider and also technologies used to develop this client.

Klíčová slova

virtuální čekárna, uživatelské rozhraní, WPF, .NET, C#, GUI, Microsoft Windows

Keywords

virtual waitingroom, user interface, WPF, .NET, C#, GUI, Microsoft Windows

Citace

HOLCMAN, Jaroslav. *Klient Virtuální čekárny pro poskytovatele služeb*. Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Kolář Martin.

Klient Virtuální čekárny pro poskytovatele služeb

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana M.Sc. Martina Koláře. Další informace mi poskytli Prof. Dr. Ing. Pavel Zemčík a další kolegové zapojeni do projektu Virtuální čekárna. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jaroslav Holcman
17. května 2016

Poděkování

Na tomto místě bych rád poděkoval panu Martinu Kolářovi, M.Sc. za vedení a vstřícný přístup při tvorbě této práce. Dále bych rád poděkoval panu profesoru Zemčíkovi a ostatním kolegům zapojeným do projektu Virtuální čekárny za poskytnuté informace a ochotnou spolupráci.

Obsah

Kapitola 1

Úvod

Virtuální čekárna je systém vznikající pro využití různými poskytovateli služeb, kteří v rámci svých podniků využívají čekárny s frontami zákazníků. Systém bude tvořit virtuální prostředí pro evidenci těchto zákazníků, umožní zákazníkům se do fronty přihlásit vzdáleně bez nutnosti jejich přítomnosti v čekárně a poskytovatelům zjednoduší každodenní provoz.

Klient pro poskytovatele služeb bude aplikace s grafickým uživatelským rozhraním, které bude sloužit právě pracovníkům na straně poskytovatele pro sledování stavu fronty v čekárně, vyvolávání zákazníků k obsluze, jejich objednávání na budoucí termíny a případně i sběr a analýzu statistik o každodenním chodu pracoviště, což může následně vést k dalšímu zefektivnění poskytovaných služeb.

V této práci je nejprve rozebráno použití vyvolávacích systémů zaváděných v reálných čekárnách a jejich výhody a nevýhody. Dále je popsána koncepce vznikající Virtuální čekárny, jejích částí a následně podrobnější analýza problémů spojených se vznikem klienta pro poskytovatele služeb. Poté je popsán návrh řešení těchto problémů, stejně tak i návrh uživatelského rozhraní celé aplikace a její implementace včetně popisu použitých technologií, vyhodnocení dosaženého výsledku a další možnosti budoucího směru vývoje.

Kapitola 2

Virtuální čekárna

Prakticky na všech úřadech, ale i u doktorů, kadeřníků, v autoservisech a podobně, se tvoří fronty čekajících lidí. Tito čekající lidé musí být ze strany poskytovatele služeb (úředník, lékař, holič,...) evidováni a následně nějakým způsobem voláni do kanceláře (případně do ordinace, k přepážce apod.). Ke zrychlení, zpřesnění a zdokonalení tohoto procesu se zavádějí různé systémy, které pomáhají poskytovatelům evidovat a volat odpovídající zákazníky. Obecně se nazývají vyvolávací systémy.

Virtuální čekárna je projekt vytvářený několika studenty FIT VUT v rámci jejich bakalářských prací. Každý ze studentů má za úkol vytvořit určitou část výsledného systému. Tato kapitola přibližuje obsah tohoto projektu jako celku, stručně popisuje jeho jednotlivé části a také jeho potenciální význam pro budoucí využití.

2.1 Vyvolávací systém

Vyvolávací systémy jsou zaváděny pro zdokonalení způsobu vyvolávání zákazníků v reálných čekárnách, například na úřadech. Následující příklad popisuje právě typické použití vyvolávacího systému na úřadě.

2.1.1 Příklad vyvolávacího systému

Zákazník po příchodu na úřad přistoupí k tiskárně pořadových lístků. Tato tiskárna je vybavena displayem (často dotykovým), pomocí kterého zákazník zvolí svůj důvod návštěvy úřadu. Tímto úkonem se zařadil do virtuální fronty některé z přepážek (může jít i o více přepážek), která je za zvolenou službu zodpovědná. Zákazník si poté odebere vytisknutý lístek s pořadovým číslem a čeká v čekárně, než je zavolán. Pracovník úřadu má k dispozici rozhraní, které mu dává informace o stavu jeho fronty. Pomocí tohoto rozhraní také volá dalšího zákazníka, který je na řadě. V té chvíli se na obrazovce v čekárně objeví číslo zákazníka, který je právě volán k přepážce, a zpravidla také číslo dané přepážky (ukázka viz obrázek 2.1). Tato událost je typicky doprovázena i zvukovým signálem, který upoutá pozornost čekajících zákazníků, a snižuje se tak možnost přehlédnutí momentu, kdy je zákazník volán. Zákazník poté může přistoupit k přepážce.

Z předchozího příkladu je zřejmé, že vyvolávací systém se skládá z více částí, kdy každá plní v rámci systému svoji konkrétní úlohu. Neodmyslitelnými částmi takovéhoto systému jsou:

- Tiskárna pořadových lístků obsahující také display a ovládání (případně dotykový display), aby zákazník mohl vybrat důvod své návštěvy.
- Obrazovka, kde se zobrazují čísla volaných zákazníků spolu s čísly odpovídajících přepážek (ukázka viz obrázek 2.1).
- Rozhraní pro pracovníka úřadu, který si volá další zákazníky k volné přepážce. Může jít o vestavěný systém nebo o aplikaci na PC.
- Server, který spravuje data celého systému a který celou činnost systému řídí. Typicky obsahuje i rozhraní pro konfiguraci systému pověřeným pracovníkem.

Tento příklad je pro zjednodušení uveden v rámci úřadu. V lékařské ordinaci nebo v jiných provozovnách poskytovatelů služeb by však podobný systém pracoval takřka stejně, až na některá případná specifika daného poskytovatele.

Obrázek 2.1: Ukázka vyvolávací obrazovky v čekárně

Některé ze zavedených vyvolávacích systémů jsou uzavřené právě na daném pracovišti a klienti se systémem interagují právě pomocí popsaných prvků umístěných v čekárně. Existují ale i systémy, které umožňují sledování stavu a případné rezervace termínů přes internetové stránky poskytovatele služeb. Takové systémy poté při volání zákazníků k přepážkám musí zohledňovat objednané a neobjednané zákazníky. S tímto přístupem se počítá i v rámci tvorby naší Virtuální čekárny. Navíc budou zahrnuty i klientské aplikace pro přenosná zařízení, která budou za tímto účelem k dispozici zákazníkům.

2.1.2 Výhody a nevýhody vyvolávacích systémů

Z pohledu zákazníka je výhodou zejména skutečnost, že nemusí stát ve frontě, která se tvoří u přepážky, ale pouze čeká na to, až se na obrazovce objeví jeho pořadové číslo.

Čekání je tak pro zákazníka příjemnější, protože místo stání ve frontě si může v čekárně sednout a případně se věnovat jiné činnosti, než na něj přijde řada. V případě neobjednaných zákazníků v čekárnách podobných těm u lékařů nemusí mít zákazník (pacient) neustále přehled o tom, kdo je ve frontě před ním a za ním. Systém sám dokáže určit pořadí podle příchodů zákazníků do čekárny.

Vyvolávací systémy však přináší výhody především poskytovatelům služeb. Například v lékařské ordinaci při použití takovéhoto systému již sestra nemusí vycházet do čekárny, kde sama eviduje příchozí pacienty a volá je do ordinace. Systém tyto úkony provádí za ní. Dochází tím k rychlejšímu a pohodlnějšímu výkonu každodenní činnosti poskytovatelů služeb.

Vyvolávací systém je také velmi vhodný ke sběru a analýze statistik o chodu pracoviště. Systém dokáže bez problému ukládat veškeré informace o počtu odbavených zákazníků, průměrné době čekání, vytíženosti přepážek apod. Tyto informace mohou výrazně přispět poskytovateli ke zdokonalení poskytovaných služeb.

Další výhodou pro poskytovatele je i možnost relativně rychle reagovat na situaci v čekárně. Pokud jsou všechny přepážky vykonávající určitou činnost obsazené a vytvořila se u nich dlouhá fronta, někteří zákazníci mohou být přesunuti k jiné přepážce, která je volná. Jednou z možností je, že pracovník sám přeposílá zákazníky k jiné přepážce (případně pracovník u volné přepážky si volá zákazníky od přetížené). Možné je však také nastavení systému takovým způsobem, že některé přepážky při přetížení automaticky mají za úkol vykonávat i činnosti, které jim za běžného provozu nepřísluší. Systém je tak mnohem pružnější a dokáže reagovat na příchody zákazníků v rušných hodinách.

Výhodou pro poskytovatele je také spokojenost zákazníků. K tomu je ovšem zapotřebí, aby systém skutečně dobře fungoval a zlepšoval kvalitu poskytovaných služeb. S tím se pojí také jedna z nevýhod takovéhoto systému. Pokud systém nepracuje správně nebo často selhává, může docházet ke zmatkům, prodloužení čekací doby zákazníků a následně i k jejich nespokojenosti.

Nevýhodou je zcela jistě i nutnost poskytovatele zaškolit pracovníky v práci se systémem a samozřejmě i odpovídající upozornění zákazníků na zavedení tohoto systému. V tomto případě ale spíše nejde o závažný problém, jelikož systémy nejsou nikterak složité na ovládání ani ze strany zákazníků, ani ze strany obsluhujících pracovníků. Navíc správně fungující systém dokáže zlepšit prostředí pro všechny zúčastněné.

2.2 Projekt Virtuální čekárna

Projekt Virtuální čekárna má za cíl vytvořit podobný systém, jaký je popsaný v předcházející kapitole, a nabídnout ho k využití různým poskytovatelům služeb. Předpokladem pro opravdové využití vytvořeného systému je možnost aplikace takovéhoto systému v reálných čekárnách. Proto se projekt inspiruje již fungujícími a ověřenými skutečnostmi z existujících systémů a na ně postupně navazuje. Projekt se neomezuje na jednu konkrétní cílovou skupinu poskytovatelů služeb, ale naopak jedním z cílů je vytvořit univerzální systém, který by mohl být využit širokým spektrem různých poskytovatelů. Jako ideální místo využití Virtuální čekárny se jeví lékařské ordinace a jejich čekárny. Nicméně systém by svojí univerzalitou měl být bez problémů dostupný i pro poskytovatele dalších služeb, například autoservisy nebo kadeřnictví.

2.2.1 Popis Virtuální čekárny jako celku

Výsledný systém Virtuální čekárny má fungovat jako služba nabízená k použití jiným poskytovatelům různých služeb pro zákazníky (lékař, kadeřník,...). Celý systém Virtuální čekárny bude využívat centrálního serveru, který bude zpřístupněn všem poskytovatelům služeb, kteří budou Virtuální čekárnu využívat. To je také jeden z rozdílů oproti dříve popsaným vyvolávacím systémům, kde typicky každý z poskytovatelů vlastní svůj server, na kterém systém běží právě pro potřeby tohoto poskytovatele. Použití centrálního serveru ulehčí zákazníkům práci s klienty v mobilních zařízeních. Klienti se mohou automaticky připojit na centrální server, který jim odpoví a poskytne informace o požadovaných poskytovatelích, jejich čekárnách a frontách. Zákazník tak bude moci pohodlně používat klientskou aplikaci pro všechny potenciální fronty, které ho zajímají. V případě samostatných serverů by bylo třeba využít samostatných aplikací pro každého poskytovatele služeb nebo ručně zadat adresu každého konkrétního poskytovatele, aby aplikace věděla, kam se připojit.

Virtuální čekárna počítá s tím, že zákazníci budou mít možnost sledovat stav fronty a přihlašovat se do ní vzdáleně, tedy jednak z internetových stránek a také z navržených klientských aplikací pro přenosná zařízení. Také klient na straně poskytovatele služeb bude uzpůsoben tak, aby pracovník kromě aktuálního stavu čekárny a volání dalších zákazníků měl možnost i objednávat (a rušit) zákazníky na požadované termíny pomocí jedné aplikace.

2.2.2 Detaily specifikace Virtuální čekárny

Systém se sestává z několika vzájemně nezávislých částí, které spolu ovšem komunikují. Základem je server, který je napojen na databázi. Poskytovatel má k dispozici klientskou aplikaci, která mu umožňuje interagovat s jeho frontou. Zákazníci se mohou do fronty zařadit jednak pomocí klienta v čekárně poskytovatele služeb nebo přes klientskou aplikaci na webu nebo přenosném zařízení. Každý ze zákazníků může pomocí své aplikace sledovat a přihlašovat se do více front, v každé frontě ovšem vystupuje pod jiným identifikačním číslem a musí se tedy vždy objednávat jednotlivě. Celý systém bude využívat komunikační protokol navržený přímo pro komunikaci mezi prvky tohoto systému. Diagram navrženého systému Virtuální čekárny je na obrázku 2.2 níže.

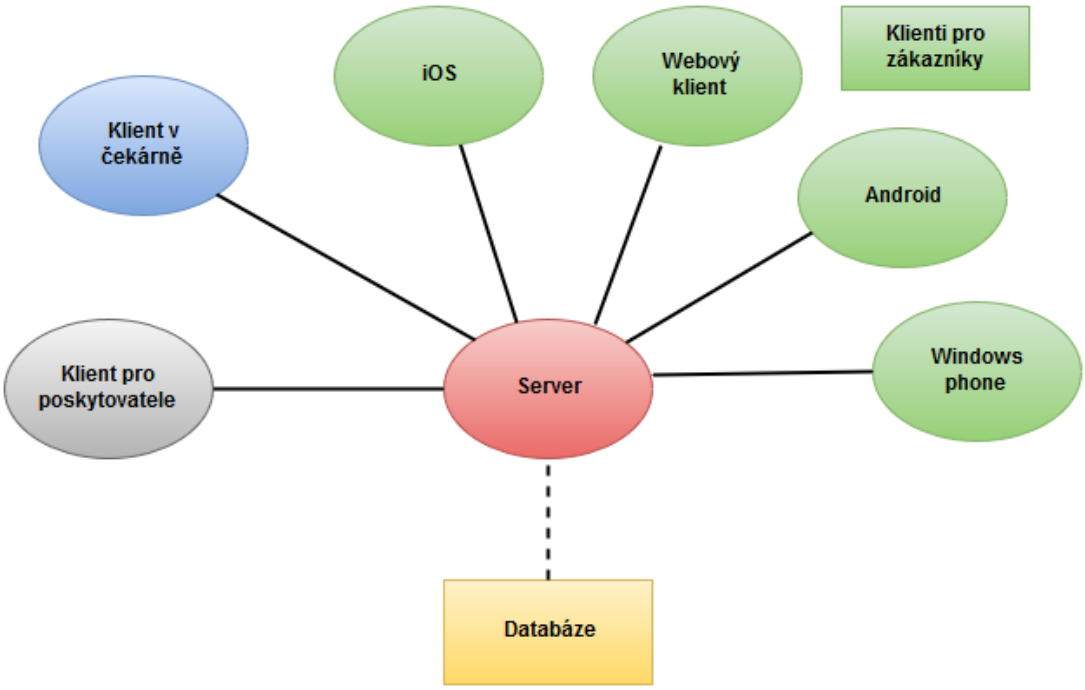
2.3 Prvky Virtuální čekárny

Celý systém Virtuální čekárny se bude skládat z více prvků. Základem je klient-server architektura s komunikací po síti. Server zajišťuje veškerou správu dat a komunikuje s klienty, kteří mohou mít v celém systému různou úlohu. Jednotlivé části systému jsou popsány dále v této kapitole.

2.3.1 Server Virtuální čekárny

Server Virtuální čekárny je centrálním článkem celého systému. Server má přímý přístup k databázi s veškerými daty, komunikuje s jednotlivými klienty a tato data jim zpřístupňuje.

Základní funkcí serveru Virtuální čekárny bude odpovídat na požadavky od klientů. Celá komunikace je navržena na protokolu http (standardně port 80), kdy server čeká na dotazy od klientů a na ně odpovídá. Konkrétní dotazy se mohou lišit podle jednotlivých typů klientů, ale základní druhy dotazů na server jsou tři: získání informací, zařazení do fronty a vyřazení z fronty.



Obrázek 2.2: Diagram systému Virtuální čekárny

V rámci protokolu komunikace mezi klienty a serverem je třeba rozlišovat mezi dotazem a odpovědí. Každý z dotazů (směr klient - server) je stavěn jako url požadavek. Server odpovídá pomocí http hlavičky buď `200 OK` v případě úspěšně vyřízeného dotazu nebo `404 NOT FOUND` v případě chyby. V těle odpovědi se poté nachází zpráva pro klienta ve tvaru XML.

Server také musí zabránit nekonzistenci dat v databázi. Je třeba zajistit, aby algoritmy pracující s uloženými daty a přijímanými daty od klientů vždy správně pracovaly, jinak hrozí nefunkčnost celého systému.

2.3.2 Klienti pro zákazníky

Rozšířením přenosných zařízení (smartphony, tablety) přichází na řadu také nové způsoby jejich využití. V dnešní době již velké procento lidí vlastní takováto zařízení a má tedy přístup k funkcím, které tato zařízení poskytují. Podnikatelé a firmy si jsou této skutečnosti samozřejmě vědomi a mají snahu toho využít. Vzniká tedy velké množství různých aplikací právě pro přenosná zařízení, které se snaží oslovit cílového zákazníka a získat ho tak na svoji stranu. Máme tak již k dispozici aplikace například pro rychlé zjištění sportovních výsledků, počasí, internetové bankovnictví v mobilu a mnoho dalších.

V projektu Virtuální čekárny také vznikají aplikace, které by mohli zákazníci použít. Pomocí aplikace bude možné získat informace o momentální situaci v čekárně, což může neobjednaným zákazníkům značně pomoci. Aplikace také umožní zjistit obsazenost jednotlivých termínů v nadcházejících dnech a objednat se dle přání zákazníka na kterýkoliv termín, který ještě není zcela obsazen. Poskytovatel služeb bude mít samozřejmě možnost takto objednané zákazníky zrušit, případně bude vyžadováno, aby objednané zákazníky explicitně potvrdil.

Klienti budou zákazníkům k dispozici na hlavních mobilních platformách, tedy Android, Windows Phone a iOS. Další možností je využití webových stránek, které také vznikají v rámci Virtuální čekárny jako jedna z variant klienta pro zákazníky poskytovatelů služeb.

2.3.3 Klient v čekárně

Nedílnou součástí celého systému Virtuální čekárny bude klient umístěný přímo v čekárně poskytovatele služeb. Tento klient slouží zákazníkům k tomu, aby nahlásili poskytovateli svůj příchod do čekárny.

Do čekárny mohou přijít jak objednaní zákazníci, tak i neobjednaní. Objednaný zákazník se identifikuje a tím je přiřazen přímo k termínu, na který se objednal. Poskytovatel v takovou chvíli má informaci o tom, že objednaný zákazník již je v čekárně a může být zavolán. Neobjednaný zákazník pomocí klienta v čekárně vybere důvod své návštěvy a je zařazen do odpovídající fronty. Bez ohledu na to, zda je příchozí zákazník objednaný či ne, obdrží své pořadové číslo, nejspíše formou vytištěného papírku. Pomocí tohoto čísla je poté volán a toto číslo se objeví na obrazovce spolu s konkrétní ordinací (případně přepážkou), která daného zákazníka obsluhuje. Tento postup se v zásadě neliší od již fungujících systémů popsaných v kapitole 2.1.

Konkrétní podoba klienta v čekárně bude aplikace na pevně umístěném tabletu se systémem Android.

2.3.4 Klient pro poskytovatele služeb

Posledním klientem v rámci systému je klient pro poskytovatele služeb. Tento klient má zajistit provozovateli možnost získat informace o stavu své fronty a to jak v daném okamžiku přímo v čekárně, tak i přehled objednaných zákazníků v dalších dnech. Klient získává od serveru informace o stavu fronty a zobrazuje je poskytovateli, který následně může vykonat odpovídající akci, např. zavolat si čekajícího zákazníka.

Klient pro poskytovatele je důkladněji popsán v kapitole 4.

Kapitola 3

Technologie

Zvolenou platformou pro vývoj klientské aplikace je Microsoft Windows. Hlavním důvodem je předpoklad, že většina potenciálních provozovatelů služeb, kteří by mohli mít zájem o Virtuální čekárnu, pravděpodobně využívá právě jednu z verzí tohoto operačního systému. Vývoj bude probíhat na operačním systému Microsoft Windows 10, ale předpokládá se kompatibilita i se staršími verzemi operačních systémů Windows, které jsou firmou Microsoft stále podporovány [12] — tedy Windows 7 a Windows 8 (respektive 8.1). Pro Windows Vista se kompatibilita zajišťovat nebude, nicméně je možné, že i na tomto systému aplikace poběží. Jelikož cílem je vytvořit desktopovou aplikaci, která bude sloužit jako grafické uživatelské rozhraní (anglická zkratka GUI — graphical user interface), tak prvním úkolem je výběr vhodné technologie, která umožní daného cíle co nejlépe dosáhnout. V této kapitole postupně popisuji jednak dostupné technologie pro tvorbu desktopových aplikací, a poté také jednotlivé technologie, které jsem při vývoji klienta Virtuální čekárny použil.

3.1 Technologie pro vývoj desktopových aplikací

Pro vývoj desktopových aplikací s grafickým uživatelským rozhraním bylo vytvořeno mnoho technologií. Níže jsou popsány některé z nich, všechny velmi známé a hojně využívané.

Jednou z možností pro vývoj aplikace je jazyk Java s použitím knihovny Swing. Java není překládána přímo do strojového kódu, ale místo toho je výsledkem tzv. bajtkód. Ten je nezávislý na architektuře zařízení a aplikace napsané v Javě tedy mohou běžet na jakémkoliv zařízení, na kterém je k dispozici interpret, tedy Java Virtual Machine (JVM). Velká výhoda této technologie je tedy v tom, že výsledné řešení je vysoce multiplatformní. Swing je knihovna pro jazyk Java, která umožňuje pohodlné psaní GUI aplikací.

Dalším možným způsobem tvorby GUI aplikace je využití programovacího jazyka C++ spolu s frameworkem Qt. Qt je multiplatformní knihovna určená právě pro tvorbu uživatelských rozhraní pro aplikace psané v C++ (je možné jeho spojení s více jazyky, ale použití s C++ je typické). V rámci Qt je využíváno tzv. signálů a slotů. Jde o přístup, kdy každá událost (např. stisknutí tlačítka, přejetí myši,...) vytvoří signál, který je obsloužen odpovídajícím slotem — speciální metodou. Pro vývoj pomocí technologie Qt je k dispozici integrované vývojové prostředí Qt Creator. Qt i Qt Creator jsou v současné době open-source technologie dostupné zdarma.

C++ je možné využít také ve spojení s Win32 API (application programming interface — česky rozhraní pro programování aplikací) pro tvorbu výkonnostně náročných aplikací pro Windows. C++ umožňuje programátorovi přímou práci s pamětí, spravovat životnost

objektů a bezprostředně využívat DirectX, především Direct3D. Případně je možné v rámci C++ kódu použít instrukce ze sady SSE nebo AVX, které jsou zaměřené především na výpočetní výkon, a mohou tak velmi výrazně urychlit chod aplikace. Tímto způsobem je možné vytvořit velmi výkonné aplikace, ale vývojová doba je delší než u ostatních popsaných technologií. Pro potřeby klientské aplikace Virtuální čekárny se nepředpokládá nutnost velkého výpočetního výkonu a ostatní technologie nabízejí mnohem jednodušší, přímočařejší a příjemnější prostředí pro vývoj, Win32 tedy k implementaci nepoužiji.

Pro vývoj přímo na platformě Windows je také možné použít rozhraní .NET Framework, který je vyvíjen společností Microsoft právě k tvorbě aplikací na jejich operačních systémech. V rámci .NET Frameworku je k dispozici WPF (Windows Presentation Foundation) pro tvorbu desktopových aplikací. WPF postupně nahrazuje starší knihovnu WinForms.

Z uvedených technologií jsem si nakonec vybral poslední jmenovanou. WPF umožňuje přímočarý a pohodlný vývoj cílové aplikace pro zvolenou platformu Microsoft Windows. Více o technologii .NET, její součásti WPF a dalších vývojových prostředcích využitých při tvorbě klienta Virtuální čekárny v rámci této bakalářské práce je uvedeno ve zbytku této kapitoly.

3.2 .NET Framework

Microsoft vytvořil platformu .NET [6] především pro své operační systémy Windows, na kterých je také nejvíce využívána. Je ovšem možné využívat .NET i na jiných operačních systémech. Nabízí prostředí pro vývoj a běh aplikací, a to jak spouštěcí rozhraní aplikací, tak i potřebné knihovny. Programátor není při vývoji pomocí jedné z dostupných technologií přímo závislý na použití konkrétního programovacího jazyka. Nejpoužívanějšími jazyky pro vývoj .NET aplikací jsou C# a Visual Basic. Je možné však použít i některý z jiných podporovaných jazyků. Jsou jimi například C++/CLI (jazyk C++ upravený firmou Microsoft pro použití v rámci .NET frameworku), F# (funkcionální jazyk) nebo J# (.NET jazyk podobný Javě). Všechny tyto jazyky mohou využívat společné knihovny dostupné v .NET Framework. Níže je uveden seznam některých součástí platformy .NET:

- ASP.NET — slouží k tvorbě webových aplikací.
- ADO.NET — knihovna sloužící pro práci s databázemi.
- Windows Forms - tvorba grafických uživatelských rozhraní, v dnešní době už je využití Windows Forms spíše na ústupu.
- WPF (Windows Presentation Foundatiton) — tvorba grafických uživatelských rozhraní, postupně nahrazuje starší Windows Forms.
- WCF (Windows Communication Foundation) — vývoj webových služeb, případně komunikační infrastruktury mezi aplikacemi.
- LINQ (Language Integrated Query) — dotazovací jazyk, určený pro přístup k datům bez ohledu na jejich zdroj.

Velmi důležitou součástí .NET Frameworku je CLR (Common Language Runtime) [6]. Jedná se o virtuální stroj, který spouští .NET aplikace. V .NET Frameworku nezáleží na tom, jaký programovací jazyk je použit při tvorbě aplikace. Zdrojový kód je přeložen odpovídajícím překladačem do tzv. společného mezijazyka — CIL (Common Intermediate

Language). Ten je poté přeložen těsně před spuštěním aplikace do nativního kódu odpovídajícího konkrétní architektuře, na které má aplikace běžet. Tento proces je nazýván překlad Just-In-Time (JIT) a jde o podobný proces, který je využit i pro interpretaci aplikací vyvíjených v Javě. Právě díky tomuto způsobu překladu není tvorba .NET aplikací omezena na použití konkrétního jazyka a vývojář má možnost používat kterýkoliv z mnoha podporovaných jazyků. Díky využití CIL je také možné používat různé části kódu psané v různých jazycích a objekty popsané v různých jazycích spolu mohou komunikovat. Každá z částí bude přeložena zvlášť právě do CIL a poté již nic nebrání překladu do nativního kódu stroje. CLR se za běhu aplikace také stará například o správu paměti, ošetření výjimek, typovou bezpečnost a správu vláken.

Další velkou částí .NET Frameworku jsou knihovny tříd. Knihovny tříd jsou úzce spojeny s CLR. Nabízí vývojářům možnost využít již implementované datové typy a funkcionalitu, bez ohledu na programovací jazyk, který je právě využíván. Pomocí tříd obsažených v těchto knihovnách je možné snadno a rychle pracovat například s řetězci nebo soubory.

V rámci .NET Frameworku byl definován tzv. Common Type System (CTS) [17] — česky společný typový systém. CTS definuje, jakým způsobem jsou deklarovány a používány datové typy v CLR. Díky CTS je možné na platformě .NET využívat jednotlivé moduly psané v různých programovacích jazycích. Velmi také napomáhá typové bezpečnosti a rychlosti vykonání kódu. V neposlední řadě také poskytuje knihovnu s primitivními datovými typy, např Boolean, Char nebo Int32.

Nejnovější verze .NET Frameworku je 4.6.1 vydaná v listopadu 2015.

3.3 WPF a C#

WPF (Windows Presentation Foundation) je součást .NET Frameworku, určená pro tvorbu grafických uživatelských rozhraní pro aplikace na systémech Microsoft Windows. WPF postupně nahrazuje starší technologii Windows Forms, která ovšem stále může být používána.

3.3.1 WPF

Technologie WPF [7, 16, 18] používá k vykreslování grafických prvků DirectX, které přistupuje přímo k hardwaru (nejčastěji grafické kartě). Na rozdíl od dříve používaného GDI (Graphics Device Interface) díky tomu přenechává velkou část práce právě na grafické kartě a snižuje tak zatížení procesoru. Veškeré grafické prvky ve WPF jsou popisovány pomocí vektorové grafiky, což umožňuje použití cílové aplikace na zařízeních s různým rozlišením a velikostí obrazovky bez výrazných problémů se zobrazením uživatelského rozhraní.

WPF odděluje vývoj grafického rozhraní od vnitřní logiky aplikace. K vytvoření grafického rozhraní je použit jazyk XAML (Extensible Application Markup Language). K programování back-endu aplikace je možno využít více .NET jazyků, nejčastěji je však používán jazyk C#. O obou těchto jazycích je více informací níže.

Důležitým prvkem v technologii WPF je vázání dat (tzv. Data binding) [7]. WPF umožňuje načítat data do aplikace z různých zdrojů a poté nezávisle na zdroji tato data určitým způsobem zobrazovat uživateli. Pomocí data bindingu je zajištěno jednak načtení dat ze zdroje do aplikačních objektů, odkud mohou být zobrazena a uživatelem upravena, tak také opětovné uložení změn zpátky do zdroje. V rámci WPF aplikace je možné rozlišit celkem čtyři druhy data bindingu:

- One time — data jsou jednou stažena ze zdroje do cílových objektů a veškeré změny zdroje jsou posléze ignorovány.
- One way — aplikace má práva pouze číst data ze zdroje, pokud jsou zdrojová data změněna, tyto změny jsou vždy reflektovány v aplikaci.
- Two way — čtení dat ze zdroje funguje stejně jako u one way, ale všechny změny dat učiněné v rámci aplikace jsou následně zapsány také zpět do zdroje.
- One way to source — aplikace má pouze zápisová práva ke zdrojovým datům.

Konkrétní zdroj dat ani způsob data bindingu nemá žádný vliv na způsob, jakým budou data v aplikaci zobrazována.

3.3.2 XAML

XAML (Extensible Apllication Markup Language) [10] je jazyk založený na XML, vytvořený firmou Microsoft a k dispozici pro použití v rámci .NET Frameworku. U WPF slouží XAML zejména k definování grafických prvků uživatelského rozhraní, jejich vlastností, přiřazení událostí (například metody obsluhy stisknutí tlačítka) apod. XAML je kompletně deklarativní jazyk, který odděluje návrh GUI od aplikační logiky. Není však nezbytně nutné ho využít, vše co lze pomocí XAML popsat je možné popsat i pomocí jiných .NET jazyků.

Při deklarování jednotlivých prvků je využíváno stromové hierarchie, kdy existuje vždy jeden kořenový prvek. Jednotlivé elementy popsané pomocí XAML kódu jsou navázány přímo na instance objektů v CLR. Na obrázku 3.1 je ukázka XAML kódu, který vytváří okno a v něm tlačítko s popisem “Click Me!”. Pod XAML kódem je zobrazeno i toto okno s vykresleným tlačítkem.

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    Title="Window with Button"
    Width="250" Height="100">

  <!-- Add button to window -->
  <Button Name="button">Click Me!</Button>

</Window>
```

Obrázek 3.1: Ukázka XAML kódu a výsledného prvku

3.3.3 C#

C# [2, 8] je vysokoúrovňový objektově orientovaný jazyk, který byl vytvořen firmou Microsoft pro použití v rámci .NET Frameworku. Inspirací pro tento jazyk jsou především jazyky C++ a Java. C# využívá CTS (Common Type System), všechny typy v C# jsou tedy odvozené od třídy System.Object. Například každý typ dědí metodu ToString(), která vypisuje textovou podobu obsahu daného typu. Z hlediska použití jazyka C# při tvorbě WPF aplikací je důležitá existence klíčového slova partial, pomocí kterého je možné definovat různé části jedné třídy ve více zdrojových souborech. Tato vlastnost je velmi užitečná v případě, že některá část třídy je generována automaticky, zatímco další část je psaná programátorem, což je přesně případ tvorby WPF aplikací pomocí Visual Studia.

V jazyce C# není možné vytvářet globální proměnné a funkce. Náhradou mohou být statické metody a jejich třídy a atributy. Také zde není k dispozici vícenásobná dědičnost, ale každá třída může implementovat více rozhraní. C# je case sensitive (v názvech se rozlišují velká a malá písmena), narozdíl od dalšího rozšířeného jazyka z rodiny .NET — Visual Basic. Za zmínku stojí také tzv. nullovatelné (nullable) typy. Nullovatelným může být proměnná jakéhokoliv datového typu, pokud je tak v kódu označena. Výsledkem toho je, že mezi povolené hodnoty daného typu je přidána hodnota null. S tím souvisí operátor koalescence, který při přiřazení testuje zdrojovou proměnnou právě na hodnotu null, a případně do cílové proměnné přiřazuje hodnotu z alternativního zdroje.

Od verze 3.0 je k dispozici programátorovi také klíčové slovo var. Toho lze využít při deklarování proměnných jako výsledku nějaké operace, kdy není třeba rozepisovat cílový datový typ, ale místo něj použít právě toto klíčové slovo. Kompilátor poté podle operace na pravé straně přiřazení automaticky určí datový typ takto deklarované proměnné. Využití tohoto klíčového slova může poměrně výrazně zrychlit psaní kódu, nicméně také může způsobit horší čitelnost kódu.

```
using System;

class Program
{
    static void Main()
    {
        Console.WriteLine("Hello, world!");
    }
}
```

Zdrojový kód 3.1: Ukázka Hello World programu v jazyce C#

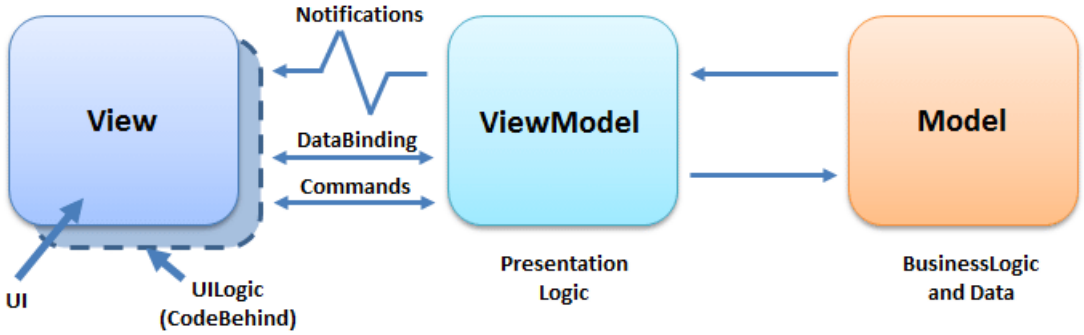
3.3.4 MVVM — Model-View-ViewModel

Pro tvorbu aplikací s grafickým uživatelským rozhraním jsou často využívány různé návrhové vzory, které oddělují logiku aplikace od grafické vrstvy aplikace. Návrhové vzory zároveň umožňují jednodušeji rozdělit práci mezi více různých vývojářů, zpřehledňují výsledný kód a také umožňují kvalitnější údržbu hotové aplikace. Mezi nejznámější návrhové vzory pro tvorbu uživatelských rozhraní patří MVC (Model-View-Controller) a MVP (Model-View-Presenter). Microsoft vytvořil nový návrhový vzor pro tvorbu aplikací pomocí WPF, který vychází především z MVC a má některé rysy společné s MVP. Návrhový vzor se jmenuje Model-View-ViewModel [1, 4, 9], zkratkou MVVM.

View je vrstva aplikace, kterou uživatel vidí na obrazovce. Jedná se o definici grafického uživatelského rozhraní, jeho prvků a reprezentace dat. Ve View se nenachází žádná logika, kterou má na starost Viewmodel. View také není zodpovědný za udržování stavu aplikace, to za něj také obstarává ViewModel. Ve WPF je View právě vrstva popisovaná pomocí jazyka XAML.

Model obsahuje základní struktury dat, tak jak budou v aplikaci používána. Často se jedná o data načtená z databáze, ale zdroje mohou být i jiné (například klient poskytovatele služeb, který bude výsledkem této práce, získává veškerá data pomocí síťové komunikace se serverem). Z pohledu MVVM je pro Model důležité to, že obsahuje data, se kterými aplikace pracuje, nikoliv však popis práce s těmito daty, ani způsob jejich zobrazování uživateli.

ViewModel spojuje obě předchozí vrstvy. Stará se o aplikační logiku, zpřístupňuje data z Modelu do View a následně propaguje změny v datech zpět z View do Modelu. Data binding popsaný výše je zodpovědný právě za provázání dat mezi částmi View a ViewModel. ViewModel je obdobou Controlleru v MVC a Presenteru v MVP.



Obrázek 3.2: Diagram MVVM a propojení jednotlivých částí

3.4 Microsoft Visual studio

Visual studio je vývojové prostředí vytvořené firmou Microsoft pro tvorbu širokého spektra aplikací na různých platformách Microsoft. Visual studio samozřejmě obsahuje editor kódu s podporou zvýrazňování syntaxe, kompilátory vestavěných jazyků, debugger a tzv. Solution explorer, který umožňuje procházet souborovou strukturu projektů. Součástí je rovněž nástroj IntelliSence, který se snaží automaticky doplňovat názvy (tříd, metod, proměnných,...), čímž zrychluje a usnadňuje programátorovu práci. Implicitně podporované programovací jazyky jsou C/C++, Visual Basic .NET a C#. Podpora dalších jazyků je možná, ale musí být přidána.

Visual studio také obsahuje vizuální designery, které v reálném čase zobrazují grafickou podobu vyvíjených komponent a aplikací. Jedním z designerů je WPF designer. Ten zobrazuje aktuální stav uživatelského rozhraní definovaného pomocí XAML.

Nejnovější verzí Visual studia je Visual studio 2015 vydané v červenci 2015.

Kapitola 4

Klient pro poskytovatele

Hlavním cílem této bakalářské práce je navrhnout a implementovat klienta Virtuální čekárny pro poskytovatele služeb. Obecný popis celého projektu Virtuální čekárny je uveden v kapitole 2. Celá tato kapitola navazuje na krátký úvod (viz 2.3.4) z popisu jednotlivých prvků Virtuální čekárny, popisuje úlohu klienta pro poskytovatele služeb a návrh cílové aplikace. Součástí této kapitoly je také popis komunikačního protokolu využívaného v rámci systému Virtuální čekárny, jelikož protokol je nedílnou a závaznou částí projektu, kterou všichni klienti (tedy i klient pro poskytovatele) musí zohlednit při své implementaci.

4.1 Popis klienta

Klient pro poskytovatele služeb je jeden z klientů v rámci klient-server architektury Virtuální čekárny. Klient bude sloužit jako grafické uživatelské rozhraní pro pracovníky na straně poskytovatele.

Jedním z cílů je efektivně a přehledně zobrazit pracovníkovi dostupná data, se kterými bude nadále pracovat. V současné době se předpokládá především zobrazení aktuálního stavu reálné čekárny s možností volat zákazníky, kteří jsou ve frontě. Dále bude aplikace umožňovat zobrazení obsazenosti termínů v dalších dnech. Důležitým faktorem je skutečnost, že každá přepážka (ordinace, kancelář,...) může být zodpovědná za více různých front, odpovídajících více různým službám vykonávaných daným poskytovatelem. Proto je třeba zohlednit zobrazení různých front a pohodlné přepínání mezi těmito frontami. Poskytovatel (respektive pověřený pracovník na straně poskytovatele - úředník, doktor,...) může samozřejmě měnit předem objednané termíny zákazníků, kteří se objednali pomocí zákaznických klientských aplikací.

Předpokladem do budoucna je také možnost zobrazení statistiky a analýzy využití přepážek pro pracovníky s odpovídajícím stupněm oprávnění přístupu, stejně jako možnost upravovat základní nastavení služeb u jednotlivých přepážek apod. Samozřejmostí je také možnost změny konfigurace základních nastavení zodpovědnou osobou, a to co nejpřímočařejší cestou.

Aplikace získává data pomocí síťové komunikace ze serveru. Nepracuje tedy přímo s žádnou databází. Veškerá data spojená s daným poskytovatelem jsou uložena na serveru a poskytovatel k nim má přístup až po spuštění aplikace, autentizaci a stažení požadovaných dat. Při každé lokální změně dat virtuální čekárny je třeba, aby aplikace odeslala tyto změny také na server, který je bude moci uložit do databáze. Při tomto procesu může nastat problém, kdy zároveň s poskytovatelem provede změnu stejných dat (například objednání

zákazníka na určitý termín v určité frontě) také některý ze zákaznických klientů. Vyřešit tento problém bude úkolem serveru, ale poskytovatel by měl mít při jakýchkoliv změnách jeho vlastních front přednost před změnami vytvořenými zákazníky.

Komunikace se serverem, získání dat, zpřístupnění těchto dat poskytovateli a odeslání zaznamenaných změn zpět na server jsou tedy hlavní úkoly klienta pro poskytovatele služeb. Kromě toho musí klient zvládat i lokální správu dat, například v momentech, kdy vypadne spojení se serverem. Na rozdíl od klientů pro zákazníky, kteří při ztrátě spojení jednoduše neumožní objednání, by klient u poskytovatele měl poskytnout alespoň omezenou funkčnost. Klient by v takovém případě měl být schopen pracovat alespoň s těmi daty, které měl k dispozici v okamžiku výpadku spojení a být schopný k nim lokálně přidávat další, ručně zadaná poskytovatelem. Z tohoto důvodu musí klient zvládat určitou logiku práce s daty, nejenom odesílání zpráv o změnách a následné stažení nových dat ze serveru. Problém poté ovšem může nastat při obnovení spojení, kdy zvláště při delší době výpadku může dojít ke značným rozdílům mezi daty na serveru a lokálními daty klienta na straně poskytovatele.

4.2 Komunikační protokol

Komunikační protokol Virtuální čekárny je navržený pro komunikaci mezi serverem a jednotlivými klienty. Protokol Virtuální čekárny bude pracovat na protokolu HTTP s využitím HTTP hlaviček `200 OK` a `404 NOT FOUND`. Předpokládaným portem pro využití při komunikaci je standardní HTTP port 80.

4.2.1 Dotaz na server

Základem komunikace je formát dotaz-odpověď, kdy klient posílá požadavek na server, server požadavek zpracuje a odešle klientovi odpověď. Obecně může být dotaz dvojího typu. Může se jednat o dotaz na data, kdy server dle udaných parametrů načte požadované hodnoty z databáze a odešle je zpět klientovi, který si je posléze sám zpracuje. Druhou formou dotazu je určitý požadavek, kterým klient server žádá o provedení odpovídající akce. Může se jednat o požadavek na zařazení do fronty na určitý termín, případně o vyřazení z již přihlášeného termínu apod. V rámci celého systému Virtuální čekárny se mohou vyskytnout další speciální typy dotazů, například zpráva od klienta v čekárně, že konkrétní zákazník dorazil na objednaný termín a čeká na zavolání do ordinace.

Formát dotazu je určen pomocí url, podle vytvořené specifikace. Při dotazu na určitá data je třeba v url dotazu zadat jednak typ dat, která jsou požadována od serveru, stejně jako identifikátor určující konkrétní informaci. Naproti tomu požadavek na určitou akci (např. zařazení do fronty) obsahuje pouze jednotlivé hodnoty, podle kterých server pozná, jaká akce se má vykonat.

Jednotlivé požadavky se taktéž mohou lišit podle jednotlivých typů klientů. Například při zařazení zákazníka do fronty se zákazník pomocí svého klienta zařazuje přímo, naproti tomu pokud zákazníka objednává poskytovatel, tak musí požadavek obsahovat jednak identifikaci poskytovatele, poté i identifikaci klienta, a nakonec termínu, na který ho objednává. Klient poskytovatele má také zvlášť určené speciální dotazy, pomocí kterých se ptá na stav v čekárně a volá zákazníky do ordinace. Níže je uvedeno několik příkladů dotazů na server:

- `adresa_serveru/company/id=2100` — dotaz na seznam front firmy s id 2100,
- `adresa_serveru/myqueues/id=101/` — dotaz na obsah fronty s id 101,
- `adresa_serveru/waitingroom/id=308/` — dotaz na stav čekárny s id 308,
- `adresa_serveru/myqueues/101/appointment/3/` — požadavek na zařazení do fronty 100 na termín 3.

Toto jsou ukázky pouze některých typů dotazů, celý protokol obsahuje více možných způsobů dotazování na server. Dá se také předpokládat, že protokol bude postupem času doplňován, případně upravován, podle potřeby jednotlivých částí Virtuální čekárny při jejich vzájemném propojování.

4.2.2 Odpověď serveru

Po přijetí dotazu a jeho zpracování odesílá server odpověď zpět klientovi. Odpovědi je možné rozdělit podle HTTP hlavičky. Pokud byl požadavek vyřízen správně, pak je hlavička odpovědi `200 OK`, v opačném případě je hlavička odpovědi `404 NOT FOUND`. Tělo odpovědi poté tvoří XML, které obsahuje informace pro klienta.

Pro dotaz na určitá data obsahuje XML strukturu těchto dat, kterou poté klient zpracuje tak, aby potřebná data mohl zobrazit uživateli. Při vyřizování požadavku (zařazení do fronty,...) se v XML nachází informace o tom, zda byl požadavek správně vyřízen a pokud ano, tak například i id, kterým je zákazník v rámci daného termínu identifikován (při požadavku na zařazení do fronty). Na obrázku 4.1 je ukázka XML, které přišlo jako odpověď od serveru. Odpověď obsahuje informace o termínech fronty s id 101.

4.2.3 Protokol pro klienta poskytovatele služeb

Klient poskytovatele služeb plní jedinečnou úlohu v celém systému Virtuální čekárny a je třeba pro jeho speciální požadavky vytvořit v rámci protokolu odpovídající požadavky a odpovědi. Klient musí být schopen se přihlásit a odhlásit ze systému, musí mít možnost přihlásit a odhlásit i konkrétního pracovníka, který právě klienta obsluhuje. Dalším specifikem oproti ostatním typům klientů je nutnost stažení aktuálního stavu čekárny. Posledním unikátním typem dotazu na server od klienta poskytovatele služeb je vyvolání zákazníka v čekárně, případně jeho opětovné vyvolání, pokud nereaguje, nebo přesunutí do jiné fronty k jiné přepážce. Níže je seznam některých z uvedených požadavků na server od klienta poskytovatele služeb:

- `adresa_serveru/counter/reg/0/compay/2100/` — přihlášení přepážky s id 0 firmy s id 2100 do systému,
- `adresa_serveru/counter/210112/opnumber/1111/` — načtení fronty zákazníků přepážky s id 210112,
- `adresa_serveru/counter/210011/queue/101/sn/15/` — vyvolání klienta s id 15 z fronty s id 101 k přepážce s id 210011,
- `adresa_serveru/counter/done/210011/` — dokončení obsluhy na přepážce s id 210011.

```xml
<response>
    <item key="101">
        <name>přepážka 2</name>
        <uri>adresa_serveru/myqueues/id=101/</uri>
        <service>sádrování</service>
        <dates>
            <id>1</id>
            <since>10:00 21.05.2016</since>
            <to>11:00 21.05.2016</to>
            <act_capacity>4</act_capacity>
            <max_capacity>5</max_capacity>
            <info> Detaily termínu - seznam přihlášených zákazníků </info>
        </dates>
        <dates>
            <id>2</id>
            <since>11:00 21.05.2016</since>
            <to>12:00 21.05.2016</to>
            <act_capacity>1</act_capacity>
            <max_capacity>5</max_capacity>
            <info> Detaily termínu - seznam přihlášených zákazníků </info>
        </dates>
        <dates>
            <id>3</id>
            <since>12:00 21.05.2016</since>
            <to>13:00 21.05.2016</to>
            <act_capacity>3</act_capacity>
            <max_capacity>5</max_capacity>
            <info> Detaily termínu - seznam přihlášených zákazníků </info>
        </dates>
    </item>
</response>
```

Obrázek 4.1: Příklad XML odpovědi od serveru

Tyto dotazy slouží jako ukázka specifické části protokolu pro klienta poskytovatele služeb. Klientská aplikace, která vznikne jako součást této práce se musí umět pomocí daného protokolu dorozumět se serverem a dokázat zpracovat odpovědi, kterými server po těchto dotazech vrátí.

4.2.4 Push-notifikace

V budoucnu je plánováno využít při komunikaci v rámci Virtuální čekárny tzv. push-notifikace. Principem push-notifikací je stav, kdy klient naslouchá a server bez předchozího dotazu od klienta zašle zprávu s informací, že došlo k nějaké změně dat na serveru. Klient si poté klasickým dotazem zažádá o nová data. Výhoda použití této koncepce je zejména v okamžitém doručení změn a v odstranění nutnosti se periodicky ptát serveru, jako tomu je u současného stavu komunikace Virtuální čekárny.

4.3 Návrh klienta

V rámci návrhu je třeba vzít v úvahu dříve popsané detaily a problémy a najít k nim odpovídající řešení. Následující text popisuje právě možné řešení problémů spojených s klientem virtuální čekárny a návrh různých částí aplikace.

4.3.1 Struktura aplikace

Aplikace bude vytvořena pomocí návrhového vzoru MVVM popsaného v kapitole (3.3.4). Bude vytvořeno několik projektů, odpovídajících logickým částem celé aplikace a dohromady tvořících jedno celkové řešení (anglicky Solution, používáno v rámci Microsoft Visual studia). Projekty budou reflektovat části Model, View a ViewModel tvořící návrhový vzor MVVM. Pro další služby (např. komunikace se serverem) vznikne oddělený projekt dostupný ostatním částem. Jednotlivé projekty budou vytvářet jmenné prostory celé aplikace, která se v budoucnu bude pravděpodobně upravovat a rozšiřovat.

4.3.2 Datové modely

Datový model aplikace slouží jako šablona dat, která bude za běhu aplikace třeba ukládat a spravovat. Uživatelské rozhraní aplikace bude posléze datový model využívat k patřičnému zobrazení informací uživateli.

Z hlediska datového modelu se dá aplikace rozdělit do dvou hlavních částí — současný stav čekárny a fronty s termíny, kam se zákazníci objednávají. V rámci aktuální fronty v čekárně je situace poměrně jednoduchá. Je třeba uchovávat pouze seznam jednotlivých zákazníků nacházejících se v čekárně a dostupné informace o každém z nich (jméno, čas objednání, požadovaná služba,...).

Pro přehled objednaných zákazníků bude model o něco málo složitější. Aplikace bude potřebovat mít k dispozici seznam všech front, za které daná ordinace zodpovídá, a musí mít možnost mezi nimi v případě potřeby rychle přepínat. V rámci modelu tedy je k dispozici seznam jednotlivých front. Každá z těchto front obsahuje informace o sobě (id, název) a poté také seznam všech dostupných termínů v rámci této fronty. Každý termín má mimo jiné svůj unikátní identifikátor, kapacitu a čas začátku a konce. Je třeba také u termínů udržovat počet a seznam již objednaných zákazníků, aby poskytovatel měl jak přehled nad stavem svých front, tak i možnost tento stav měnit.

4.3.3 Návrh GUI

Grafické uživatelské rozhraní klienta pro poskytovatele umožňuje pracovníkům zobrazovat a spravovat data popsaná v 4.3.2. Jelikož aplikace je určena pro desktopové počítače, předpokládá se tedy minimální velikost obrazovky odpovídající běžným monitorům desktopových počítačů. Na základě toho je možné navrhnout aplikaci takovým způsobem, že většina dat bude na obrazovce zobrazena v jeden okamžik. Především se jedná o souběžné zobrazení stavu čekárny provozovny poskytovatele služeb a náhledu termínů pro objednávání zákazníků na další dny. V rámci hlavního okna aplikace bude také zahrnuto zobrazení informace o zákazníkovi u přepážky (případně v ordinaci, v kanceláři,...).

Problém přepínání mezi jednotlivými frontami lze vyřešit více způsoby. Jednou z možností je seznam všech dostupných front na straně okna. Tento seznam by však nejspíše zabíral zbytečné množství prostoru a zužoval by tak zbytek okna aplikace pro zobrazení dalších dat. Řešením tohoto problému by mohlo být schování seznamu a jeho zobrazení

pouze po kliknutí na odpovídající ovládací prvek. Takové řešení ovšem bude mít za následek snížení intuitivnosti, rychlosti a pohodlnosti práce. Jinou možností uspořádání je udělat ze seznamu dostupných front horizontální lištu záložek. Horizontální lišta nebude zužovat okno, pouze ho nepatrně sníží. Jednotlivé fronty budou rychle k dispozici intuitivním a přehledným způsobem. Horní část okna tedy bude obsahovat záložky odpovídající jednotlivým frontám.

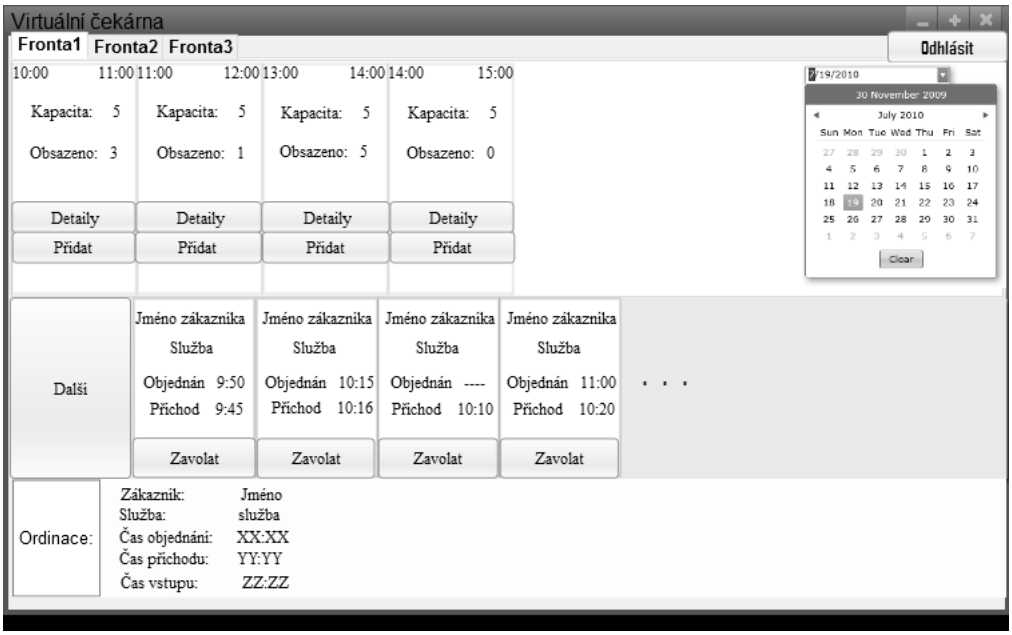
Tyto záložky korespondují s termíny na objednávání zákazníků, ty bude tedy vhodné umístit právě pod záložky. Základním způsobem zobrazení termínů bude jejich seznam v jednotlivých dnech. K dispozici bude kalendář, kde si pracovník bude moci vybrat konkrétní den, který si přeje zobrazit. Zobrazené termíny budou informovat o jejich začátku a konci, kapacitě termínu a aktuální obsazenosti. U každého termínu je třeba také zobrazit detaily a především seznam již objednaných zákazníků. Kvůli tomu není třeba vytvářet nové okno, ale stačí pouze překreslit odpovídající termín a zobrazit tyto detaily přímo v hlavním okně.

Většinu zbytku okna obsadí zobrazení zákazníků v čekárně. Přestože přepážka může mít zodpovědnost za více různých front (tedy služeb), reálná fronta k dané přepážce je pouze jedna a bude tak tedy i zobrazena. Aplikace bude řadit zákazníky podle určitého algoritmu, který je popsán v kapitole 5.2.4. Pracovník má poté k dispozici tlačítko "Další", pomocí kterého může volat zákazníky postupně k přepážce právě v takovém pořadí, jaké bylo určeno algoritmem řazení zákazníků. Pracovník má samozřejmě také možnost si určit pořadí zákazníků sám a volat si je podle svého uvážení místo automaticky určeného pořadí. U každého zákazníka je třeba zobrazit jméno, požadovanou službu, čas příchodu a v případě objednaného zákazníka i čas objednání.

Pod frontou čekajících zákazníků bude ještě umístěn panel s informacemi o stavu přepážky. Pokud je obsazená, tak bude panel obsahovat informace o právě přítomném zákazníkovi. V rámci toho bude také možné měřit čas obsluhy pro budoucí statistické zpracování a analýzu.

Obrázek 4.2 zobrazuje schéma navrženého GUI.

Ne všichni poskytovatelé služeb však potřebují celoobrazovkové uživatelské rozhraní. Například při použití systému na úřadě se dá předpokládat, že nebude třeba zákazníky objednávat na budoucí termíny, ale pracovník spíše potřebuje pouze vyvolat dalšího zákazníka z fronty a zbytek obrazovky využije pro jiné účely. Proto je vhodné vytvořit minimalistickou verzi uživatelského rozhraní, které bude zobrazovat pouze tlačítko "Další" a frontu zákazníků v čekárně, s možností zúžení pouze na tlačítko "Další". Mezi oběma dostupnými zobrazeními by mělo být možno za běhu aplikace přepínat a výchozí nastavené zobrazení bude možno upravit podle požadavku poskytovatele v konfiguračním souboru.



Obrázek 4.2: Schéma návrhu GUI

Kapitola 5

Implementace řešení

Na základě vytvořeného návrhu jsem začal s implementací aplikace pomocí technologií popsaných v kapitole 3. Podle návrhového vzoru MVVM bylo třeba postupně vytvořit datový model obsahující třídy, které odpovídají používaným datům v rámci aplikace. Poté bylo třeba vytvořit potřebné Views (český ekvivalent je Pohledy, ale tento překlad se běžně nepoužívá), které budou použity pro vytvoření uživatelského rozhraní aplikace podle návrhu popsaného v kapitole 4.3. Třetím krokem byla implementace odpovídajících ViewModelů (ViewModels), které tyto dvě vrstvy propojí. Všechny části aplikace jsou popsány dále v této kapitole.

Struktura aplikace je rozdělena na dílčí jmenné prostory, kterým odpovídá složková struktura souborů se zdrojovými kódy. Základní rozdělení je do pěti jednotlivých projektů v rámci řešení (Solution) `WaitingRoom`. Každý z těchto projektů ještě obsahuje vnitřní strukturu rozdělení zdrojových souborů podle logických částí, které jsou v nich řešeny. Datové třídy tvořící datový model aplikace jsou obsaženy v projektu `WaitingRoom.Model`, projekt `WaitingRoom.ViewModels` obsahuje ViewModely a v projektu `WaitingRoom.Desktop` jsou definovány Views a další komponenty, které tvoří grafické uživatelské rozhraní. Zbylé dva projekty obsahují podpůrné třídy pro běh aplikace, které jsou použity především v rámci ViewModelů. Jsou to projekty `WaitingRomm.Services` a `WaitingRoom.StaticResources`. Podrobnější popis jednotlivých projektů a problémů, které řeší, je uveden dále.

5.1 Datový model

Datový model je dle návrhu v kapitole 4.3.2 rozdělen na dvě hlavní části, a to třídy pro reprezentaci termínů pro objednávání a pro reprezentaci čekárny. Třídy `ServiceRoster`, `ServiceTerm` a `Appointment` obsahují atributy pro udržení potřebných dat v rámci objednávání termínů. `Appointment` zobrazuje každé jednotlivé objednání zákazníka. `ServiceTerm` je obrazem konkrétního termínu, na který je možné se přihlásit a obsahuje také seznam objednávek. Seznam termínů spolu s dalšími informacemi jsou obsahem třídy `ServiceRoster`. Tato třída je posléze využita ve ViewModelu k uchování všech dostupných dat ohledně termínů.

Z hlediska čekárny stačí pouze třída `Costumer`, která reprezentuje čekajícího zákazníka. Ve ViewModelu je tato třída využita v kolekci, která reprezentuje frontu čekajících zákazníků.

Všechny datové třídy mají společného předka `BaseModel`, který implementuje rozhraní `INotifyPropertyChanged` [15]. Implementace tohoto rozhraní se používá pro reflektování

změn dat v uživatelském rozhraní. Při využití bindingu (3.3.1) jsou poté prvky GUI notifikovány pokaždé, když dojde ke změně hodnoty odpovídajícího atributu v instanci datové třídy. Toho je docíleno vyvoláním události `PropertyChanged`, na kterou GUI reaguje aktualizováním prvků, kterých se změna týká.

5.2 Aplikační logika

Po vytvoření datového modelu jsem začal pracovat na aplikační logice. Zároveň s ní vznikalo i zjednodušené uživatelské rozhraní, na kterém bylo možné průběžně testovat funkčnost implementovaných metod.

5.2.1 ViewModely

ViewModely využívají datových tříd popsaných pomocí modelu, zpřístupňují je pro zobrazení v rámci uživatelského rozhraní a zároveň se starají o reakci na podněty přicházející právě z prvků GUI. ViewModely jsou v aplikaci vytvořeny čtyři. `LoginViewModel` je využit pro přihlašovací obrazovku a úspěšné přihlášení aplikace (a zároveň pracovníka) na server Virtuální čekárny. Přihlášení pracovníka do systému má dvě postupné fáze, které ovšem pro pracovníka samotného zůstávají skryty v jednom úkonu. Pro přihlášení není vyžadováno přihlašovací jméno, ale pouze heslo. Po spuštění aplikace dojde k přihlášení přepážky na server pomocí identifikátoru konkrétní přepážky uloženého v konfiguračním souboru. Pokud server odpoví kladně, tak je odesláno pracovníkem zadané heslo, pomocí kterého je do systému přihlášen i pracovník sám, dojde k přepnutí uživatelského rozhraní do hlavního okna a od této chvíle je možné s aplikací pracovat.

Hlavní okno aplikace využívá tři různé ViewModely. Celé okno využívá `MainViewModel`. Ten má na starost obsluhu tlačítek v menu na odhlášení a zobrazení konfigurace. Jak již bylo zmíněno výše, aplikace využívá konfigurační soubor. Jeho obsahem jsou adresa a port serveru Virtuální čekárny, identifikátor přepážky, na které daná aplikace běží, a další informace. Zbytek okna je spravován ostatními ViewModely, které jsou v `MainViewModel` odkazovány. `ServicesViewModel` je napojen na zobrazení seznamu termínů a objednávání, `QueueViewModel` se zase stará o data v rámci čekárny a ordinace. Oba tyto ViewModely zajišťují inicializaci svých dat při startu aplikace, mají k dispozici třídy se službami (např. pro komunikaci se serverem) a implementují další potřebné metody, například algoritmus řazení zákazníků v čekárně.

5.2.2 Příkazy

Příkazy (commands) [3, 11] jsou součástí ViewModelů a ve WPF aplikacích podle návrhového vzoru MVVM slouží pro reakci aplikace na uživatelské podněty z uživatelského rozhraní, typicky například stisknutí tlačítka. Příkazy se pomocí bindingu naváží na tlačítka a WPF si při stisknutí tlačítka zavolá metodu `Execute`, která se vykoná. Podobné funkcionality se dá docílit pomocí vyvolání události, která je následně provedena v code-behind daného okna. Tento přístup se ovšem neslučuje s návrhovým vzorem MVVM. Pro vytvoření nového příkazu je třeba implementovat rozhranní `ICommand` [11]. Součástí rozhraní `ICommand` je také metoda `CanExecute`, která podle návratové hodnoty (`true` nebo `false`) určuje, zda příkaz může být vykonán, tedy zda stisknutí tlačítka bude mít kýžený efekt.

V aplikaci jsem využil generické třídy `BaseCommand`, ze které všechny ostatní příkazy dědí. Tato třída obsahuje odkaz na ViewModel, ve kterém je příkaz využit, a tudíž v metodě `Execute` je možné pracovat s daty, která ViewModel spravuje. V aplikaci jsou příkazy využity například pro přepínání mezi zobrazením obsahu hlavního okna a okna konfigurace, také pro volání zákazníků do čekárny a u dalších tlačítek použitých v uživatelském rozhraní. V ukázce zdorojového kódu 5.1 je příklad implementace `CallNextCostumerCommand` a právě jeho metod `CanExecute` a `Execute`.

```
public override bool CanExecute(object parameter)
{
    return this.ViewModel.Costumers.Count != 0;
}

public override bool Execute(object parameter)
{
    var costumer = this.ViewModel.Costumers.First();
    this.ViewModel.Costumers.RemoveAt(0);

    costumer.OfficeEnteringTime = new Time((uint)DateTime.Now.Hour,
        (uint)DateTime.Now.Minute);

    this.ViewModel.CostumerInOffice = costumer;
}
```

Zdrojový kód 5.1: Ukázka implementace Command

5.2.3 Komunikace se serverem

Jelikož klient pro poskytovatele služeb je klientskou aplikací v rámci klient-server architektury, je třeba implementovat funkční síťovou komunikaci se serverem pomocí dostupného protokolu (4.2). Pro tento účel jsem vytvořil statickou třídu `Communicator` v projektu `WaitingRoom.Services`, která implementuje statické metody přístupné pro další třídy. Při startu programu jsou do třídy `Communicator` z konfiguračního souboru nahrána data potřebná k přístupu na server, tedy adresa serveru, port a identifikátor přepážky. Jednotlivé metody jsou poté určeny pro jednotlivé způsoby komunikace, tedy například pro přihlášení přepážky a pracovníka do systému nebo pro stažení obsahu čekárny.

Protože server na požadavky od klienta reaguje pomocí odpovědí ve formátu XML, je třeba tyto odpovědi patřičně zpracovat a získat potřebná data. K tomuto účelu je v projektu `WaitingRoom.Services` k dispozici statická třída `XmlParser`, která zpřístupňuje statické metody právě pro zpracování XML dat. Pomocí této třídy je také při startu programu zpracováván konfigurační soubor, který je rovněž ve formátu XML. Jelikož u většiny odpovědí je dopředu známo, co by se v ní mělo nacházet, využívám pro zpracování příchozích XML dat dostupnou třídu `XmlDocument` [5] a jeho metody `SelectNodes` a `SelectSingleNode`. Tyto metody přistupují přímo k explicitně uvedeným uzlům.

Z důvodu toho, že často je třeba zpracovat opakovaně stejný typ uzlu v jedné odpovědi, je využíváno klíčového slova jazyka C# `yield`. Pomocí tohoto klíčového slova je možné vracet z metody iterativně stejný typ odpovědi (pomocí generického rozhraní `IEnumerable`) a v místě volání metody tyto odpovědi v cyklu zpracovávat. Příklad využití této vlastnosti jazyka C# je v ukázce zdrojového kódu 5.2. V třídě `ServicesViewModel` je třeba získat ze serveru seznam dostupných služeb. Je tedy zavolána metoda `FetchServiceRostersIds` ze třídy `Communicator`. Tato metoda je volána v cyklu a každý průchod cyklem zpracovává

jednu návratovou hodnotu. Podobně tato metoda využívá metody `ParseListOfServices` ze statické třídy `XmlParser`, která zpracovává XML odpověď serveru a pomocí `yield return` postupně vrací všechny hodnoty, které z odpovědi získala.

```
// download list of accesible queues
foreach(var id in Communicator.FetchServiceRostersIds())
{
    ServiceRosters.Add(new ServiceRoster(id));
}


public static IEnumerable<int> FetchServiceRostersIds()
{
    // get list of services
    var Response = TalkToServer(ServerName + "/myqueues/");

    // parse response and add services to service rosters
    foreach(var id in XmlParser.ParseListOfServices(response))
    {
        yield return id;
    }
}


public static IEnumerable<int> ParseListOfServices(string xmlResponse)
{
    var xml = new XmlDocument();
    xml.LoadXml(xmlResponse);

    // get all <item> nodes
    var nodes = xml.SelectNodes("/response/item");

    // get service ids from nodes
    foreach(XmlNode node in nodes)
    {
        yield return int.Parse(node.Attributes["key"].Value);
    }
}
```

Zdrojový kód 5.2: Ukázka použití klíčového slova yield v C# kódu

5.2.4 Řazení zákazníků

Aby nemusel pracovník stále volat čekající zákazníky ručně, a tedy přemýšlet nad jejich pořadím, je třeba použít nějaký způsob jejich seřazení ve frontě. Pracovník již poté pouze volá dalšího zákazníka, který je momentálně na řadě. Při řazení zákazníků je třeba zohlednit několik faktorů, nicméně výsledná situace je nakonec poměrně jednoduchá. Algoritmus řazení musí zohlednit objednané a neobjednané zákazníky. V případě objednaných zákazníků budou logicky řazeni podle času objednání na dané termíny. V rámci jednoho termínu jsou však již všichni objednaní na stejný čas a bude tedy rozhodovat čas příchodu do čekárny. Objednaní zákazníci jsou tedy seřazeni poměrně jednoduše. Do této fronty je nyní ještě potřeba zařadit zákazníky neobjednané. Algoritmus se takovéto zákazníky snaží vždy zařadit na konec aktuálního termínu za všechny objednané zákazníky. Může se ovšem stát, že termín je plný a neobjednaný zákazník tudíž bude přeřazen až za další skupinu zákazníků

objednaných na další termín. V takovém případě je především na přítomném pracovníkovi, zda si zavolá již déle čekajícího neobjednaného zákazníka na úkor objednaných. Druhým řešením do budoucna by mohlo být nastavení maximální čekací doby podle přání poskytovatele služeb a při překročení této doby automatické zařazení neobjednaného zákazníka do fronty bez ohledu na objednané zákazníky.

Samotný algoritmus řazení pracuje tak, že zákazníci ve frontě jsou rozděleni na tři skupiny. Jednu skupinu tvoří objednaní zákazníci, jejichž objednaný termín právě probíhá. Druhou skupinu tvoří zákazníci, kteří jsou sice objednaní, ale jejich termín ještě nezačal, tudíž jsou v čekárně dříve. Poslední skupinou jsou neobjednaní zákazníci. Všechny tři tyto skupiny jsou nezávisle na sobě seřazeny podle času příchodu do čekárny a poté vloženy do společné seřazené fronty. Nejdříve zákazníci objednaní na současný termín, poté neobjednaní zákazníci, kteří tím vyplní případné prázdné místo daného termínu a nakonec zákazníci, kteří jsou objednaní na později. Jak je popsáno výše, tak může nastat situace, že neobjednaní zákazníci jsou sice do fronty zařazeni před zákazníky objednané na pozdější termín, ale protože do konce současného termínu nestihnou být obslouženi, tak jsou přeřazeni na konec dalšího termínu. Teoreticky tedy může nastat situace, že neobjednaný zákazník je ve frontě stále posouván na pozdější termíny, ale k jeho obsloužení dojde až za dlouhou dobu. S touto situací samotný řadící algoritmus v současné době nepočítá, ale je jeden z možných budoucích směrů vývoje aplikace, pro lepší praktické využití.

5.2.5 Práce offline

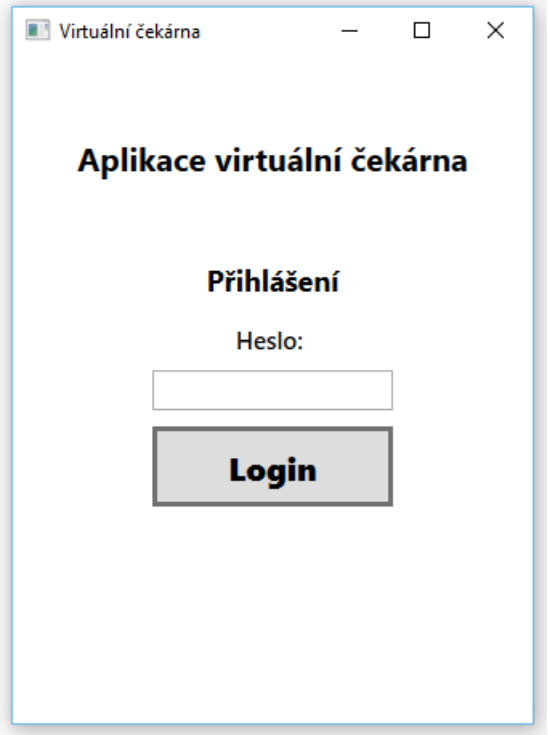
Aplikace sice ke svému běhu potřebuje funkční komunikaci se serverem, může však nastat situace, kdy tato komunikace na určitou dobu nebude k dispozici. Pro takovou situaci je aplikace připravena obsluhovat aktuálně dostupná data od serveru. Pracovník tedy bude mít možnost dále vyvolávat zákazníky (v případě přímého spojení s klientem v čekárně) a také může objednávat zákazníky na připravené termíny. Není ovšem zatím implementováno následné přenesení těchto dat na server a ošetření případných kolizí.

5.3 GUI

Grafické uživatelské rozhraní vznikalo společně s tvorbou ViewModelů, na které jsou jednotlivé jeho části navázané. Jedná se o vrstvu View z návrhového vzoru MVVM. Veškeré prvky využité v rámci GUI jsou umístěny v projektu `WaitingRoom`. Pro potřeby uživatelského rozhraní byla vytvořena tři okna. Prvním z nich je přihlašovací okno v souboru `LoginWindow.xaml`, jehož hlavní část je zobrazena na obrázku 5.1. Ostatní dvě okna jsou hlavní celoobrazovkové okno v souboru `MainWindow.xaml` a zmenšené okno zobrazující pouze frontu a vyvolávací tlačítko v souboru `MinimalizedWindow.xaml`.

Hlavní okno aplikace je tvořeno několika různými komponentami definovanými ve složce `UserControls`. Jsou zde vytvořeny komponenty pro zobrazení horního menu, rozpisu termínů služeb pro objednávání, čekárnu, ordinaci a konfigurační obrazovku. Všechny tyto komponenty jsou využity v rámci hlavního okna aplikace, kde jim je předán i odpovídající datový kontext (atribut DataContext), který poté využívají pro zobrazení dat pomocí data binding.

Jelikož v části rozpisů termínů služeb je třeba přepínat mezi více službami, byl pro jeho zobrazení použit prvek `TabControl` [13]. Ten umožňuje přepínání mezi položkami pomocí záložek, kdy obsahem každé položky jsou jiná data. Tento prvek také umožňuje navázání dat z kolekce (`ObservableCollection` využívána ve WPF aplikacích pro uchování seznamu



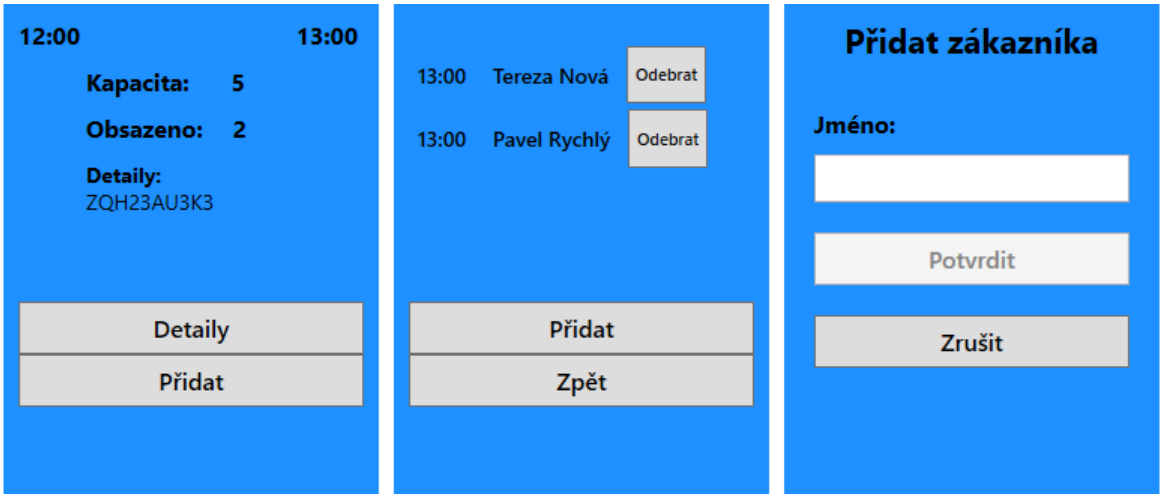
Obrázek 5.1: Obraz přihlašovacího okna

stejných datových položek), která může mít proměnný počet položek. To je výhodná vlastnost, protože počet služeb určených pro danou přepážku není nijak omezen. Na obrázku 5.2 je ukázka záložek prvku `TabControl`.

Obrázek 5.2: Ukázka záložek prvku `TabControl`

Obsahem každé položky prvku `TabControl` je seznam dostupných termínů zobrazený pomocí prvku `ListBox` [13]. Každý z těchto termínů má tři různé formy zobrazení, mezi kterými je možno přepínat pomocí tlačítek. Jedná se o základní zobrazení informací o termínu (začátek, konec, kapacita,...), dále o detail termínu s jednotlivými zákazníky, kteří jsou na něj přihlášeni, a nakonec je možnost objednání nového zákazníka na určitý termín. Příklad všech tří možností je na obrázku 5.3.

Zobrazení jedné ze tří možností je vyřešeno tak, že jednotlivé možnosti jsou najednou vykresleny na různých vrstvách vodorovných s plochou obrazovky (využití atributu `ZIndex`). ViewModel rozpisu služeb obsahuje indikátory, které ze tří zobrazení má být právě použito, a pomocí převodníků (converters) [14] je na základě hodnot těchto indikátorů určeno, co bude aktuálně zobrazeno na místě konkrétního termínu. Stejného postupu je využito i v případě zobrazení nastavení konfigurace v rámci hlavního okna. V ukázkách 5.3 a 5.4 je příklad implementace a použití převodníku `BoolToVisibilityConverter`, jehož metoda `Convert` na základě hodnoty typu bool určí, zda bude konkrétní prvek viditelný nebo ne.



Obrázek 5.3: Jednotlivé pohledy termínů k pro objednání

```
Grid.ZIndex=1
Visibility="{Binding ShowDetails,
                     Converter={StaticResource BoolToVisibilityConverter
                         }}"
```

Zdrojový kód 5.3: Ukázka použití převodníku hodnot v XAML kódu

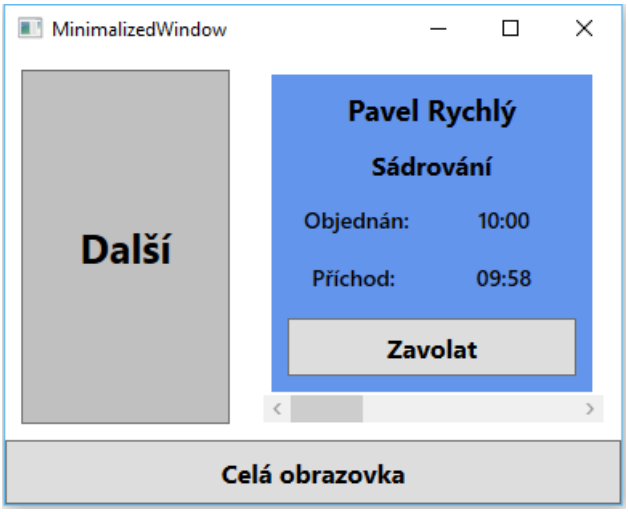
```
public class BoolToVisibilityConverter : IValueConverter
{
    public object Convert(object value, Type targetType,
        object parameter, CultureInfo culture)
    {
        var isValueTrue = (bool) value;
        return isValueTrue ? Visibility.Visible : Visibility.Collapsed;
    }
}
```

Zdrojový kód 5.4: Ukázka implementace převodníku hodnot typu bool

Část obrazovky obsahující frontu v čekárně je tvořena jedním tlačítkem, které vyvolává zákazníky z fronty, a samotnou frontou vykreslenou opět pomocí prvku `ListBox`. Pod frontou je ještě zobrazen aktuální stav přepážky spolu s tlačítkem pro ukončení obsluhy. Ukázka celého hlavního okna uživatelského rozhraní je na obrázku 5.4.

Na obrázku 5.5 je poté znázorněna ukázka zmenšeného okna s aktuálním začátkem fronty zákazníků a tlačítkem pro vyvolávání zákazníků ze začátku fronty.

Obrázek 5.4: Ukázka hlavního okna uživatelského rozhraní



Obrázek 5.5: Ukázka zmenšeného zobrazení fronty zákazníků

Kapitola 6

Závěr

Cílem této bakalářské práce bylo navrhnout a implementovat klienta Virtuální čekárny pro poskytovatele služeb. Jedná se o aplikaci s grafickým uživatelským rozhraním, která na straně poskytovatele služeb slouží pro přístup k datům uloženým na serveru a interakci s nimi. Předpokládá se budoucí využití výsledné aplikace v rámci skutečného systému, který bude přístupný pro poskytovatele služeb pro potřeby jejich čekáren. Jednotlivé části systému vznikly v rámci bakalářských prací několika studentů FIT VUT v Brně.

Po analýze dostupných informací o reálných čekárnách a potřebách aplikace v rámci celého systému byla aplikace navržena a implementována pro platformu Microsoft Windows pomocí technologie .NET a její součásti WPF (3). V průběhu vývoje byl postupně vytvořen datový model (5.1), code-behind uživatelského rozhraní (5.2) a také samotné uživatelské rozhraní (5.3). Klient komunikuje se serverem, který mu poskytuje data. Umožňuje zobrazit stav termínů pro objednání zákazníků, objednat zákazníka a případně také rušit objednávky zákazníků na konkrétní termíny. Další funkcí klienta je zobrazení stavu čekárny poskytovatele služeb a volání čekajících zákazníků. Byl vytvořen řadící algoritmus (5.2.4), který řadí zákazníky v čekárně a ulehčuje tak pracovníkovi práci.

Uživatelské rozhraní je možno zobrazit ve dvou režimech a to buď jako celoobrazovkové okno, které poskytuje plnou funkcionalitu nebo jako minimalizované okno zobrazující pouze určitý úsek fronty v čekárně a tlačítko na zavolání dalšího zákazníka. Pro případ dočasného výpadku spojení se serverem byla také implementována logika umožňující správu aktuálně dostupných dat bez pomoci serveru.

Aplikace samotná bude nadále doplňována a rozšiřována. Nejdříve je nutné dokončit implementaci zbytku komunikace se serverem podle zpřístupněného komunikačního protokolu (4.2) a případná úprava uživatelského rozhraní pro více různých požadavků různých poskytovatelů. V rámci komunikace se serverem je do budoucna také plánováno využití push-notifikací. Dalším možným budoucím rozšířením je sběr a analýza statistik, například o průměrné době čekání zákazníků, době obsluhy, délky front apod.

Literatura

[1] DAJBYCH Václav: *MVVM: Model-View-ViewModel. In: DOTNETPORTAL.cz [online].* 2009-04-21 [cit. 2016-03-23].
URL http://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel

[2] DRAYTON, Peter; ALBAHARI, Ben; NEWARD, Ted: *C# v kostce.* Grada publishing a.s., 2003, ISBN 80-247-0443-9.

[3] KERR, Dave: *Commands in MVVM. In: Code project [online].* 2012-12-03 [cit. 2016-04-25].
URL http://www.codeproject.com/Articles/274982/Commands-in-MVVM

[4] LIKNESS, Jeremy: *Model-View-ViewModel (MVVM) Explained. In: Code project [online].* 2010-08-08 [cit. 2016-03-23].
URL http://www.codeproject.com/Articles/100175/Model-View-ViewModel-MVVM-Explained

[5] MAREŠ, Amadeo: *1001 tipů a triků pro C#.* Computer Press a.s., 2008, ISBN 978-80-251-2125-2.

[6] MICROSOFT: *Overview of the .NET Framework [online].* [cit. 2016-03-19].
URL
https://msdn.microsoft.com/en-us/library/zw4w595w%28v=vs.110%29.aspx

[7] MICROSOFT: *Introduction to WPF [online].* [cit. 2016-03-22].
URL
https://msdn.microsoft.com/en-us/library/aa970268%28v=vs.100%29.aspx

[8] MICROSOFT: *C# Language Fundamentals [online].* [cit. 2016-03-23].
URL https://msdn.microsoft.com/library/orm-9780596521066-01-03.aspx

[9] MICROSOFT: *Implementing the Model-View-ViewModel Pattern [online].* [cit. 2016-03-23].
URL https://msdn.microsoft.com/en-us/library/ff798384.aspx

[10] MICROSOFT: *XAML Overview (WPF) [online].* [cit. 2016-03-23].
URL
https://msdn.microsoft.com/en-us/library/ms752059%28v=vs.100%29.aspx

[11] MICROSOFT: *Commanding Overview [online].* [cit. 2016-04-25].
URL
https://msdn.microsoft.com/en-us/library/ms752308%28v=vs.100%29.aspx

[12] MICROSOFT: *Windows lifecycle fact sheet [online]*. Last updated: January 2016 [cit. 2016-03-19].
URL http://windows.microsoft.com/en-us/windows/lifecycle

[13] MOSER, Christian: *Built-in Controls of WPF. In: Christian Mosers WPF Tutorial.net [online]*. Last updated: 2008-10-18 [cit. 2016-04-29].
URL http://www.wpftutorial.net/Controls.html

[14] MOSER, Christian: *ValueConverters. In: Christian Mosers WPF Tutorial.net [online]*. Last updated: 2010-02-19 [cit. 2016-04-29].
URL http://www.wpftutorial.net/ValueConverters.html

[15] MOSER, Christian: *MVVM: Model-View-ViewModel. In: Christian Mosers WPF Tutorial.net [online]*. Last updated: 2011-03-03 [cit. 2016-04-21].
URL http://www.wpftutorial.net/INotifyPropertyChanged.html

[16] NOBLE, Sam; BOURTON, Sam; JONES, Allen: *WPF Recipes in C# 2008, A Problem-Solution Approach*. Apress, 2008, ISBN 978-1-4302-1084-9.

[17] SAILLESH, Pawar: *Common Type System (CTS) in .NET. In: C# corner [online]*. 2015-07-01 [cit. 2016-03-19].
URL http://www.c-sharpcorner.com/UploadFile/cb1429/cts-common-type-system-in-net/

[18] ČÁPKA, David: *Okenní aplikace v C# .NET WPF. In: ITnewtwork.cz [online]*. [cit. 2016-03-22].
URL http://www.itnetwork.cz/csharp/wpf

Přílohy

Seznam příloh

Příloha A

Obsah DVD

- readme.txt - soubor s popisem obsahu DVD a návodem k aplikaci
- \src - složka se zdrojovými kódy
- \bin - složka se spustitelnou aplikací
- \thesis_ src - složka se zdrojovými kódy písemné zprávy v LaTeX
- \thesis - složka s písemnou zprávou v pdf