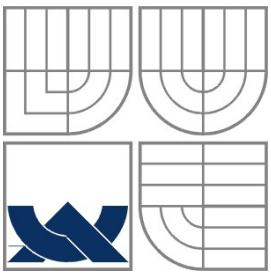
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

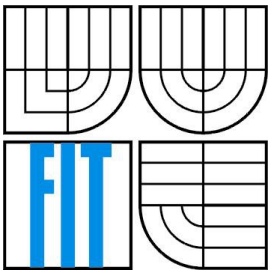
Fakulta informačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2016 Patrik Mrázek



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

ALGORITMUS OBSLUHY VIRTUÁLNÍ ČEKÁRNY

ALGORITHM FOR VIRTUAL WAITING ROOM HANDLING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Patrik Mrázek

VEDOUCÍ PRÁCE
SUPERVISOR

Prof. Dr. Ing. Pavel Zemčík

BRNO 2016

Abstrakt

Tato práce se věnuje problematice virtuálních čekáren, informuje o aktuálně dostupných produktech a programovacích prostředcích, hodnotí je a nabízí řešení nedostatků zjištěných v současném stavu. Výstupem je pak algoritmus pro obsluhu požadavků na rezervaci ve virtuální čekárně, řešící neefektivní využití otevírací doby, vznikající z neoptimálních požadavků přicházejících ze strany klienta. Práce dále popisuje vytvořené řešení z hlediska principu funkčnosti a analyzuje úspěšnost řešení zjištěnou na simulovaném případu.

Abstract

This thesis deals with a virtual waiting room topic. It informs about currently available products and programming tools. The thesis evaluates them and provides a solution to imperfections found in a current state. The output is an algorithm for handling booking requests in a waiting room. The algorithm solves an inefficient usage of business hours originating from unoptimal requests from a client side. This thesis also describes the created solution in terms of principal of functionality and analyses a success rate of the solution ascertained from a usage simulation.

Klíčová slova

čekárna, rezervace, algoritmus, efektivita, optimálnost, PHP, server

Keywords

waiting room, booking, algorithm, efficiency, optimality, PHP, server

Citace

MRÁZEK, Patrik. Algoritmus obsluhy virtuální čekárny. Brno, 2016. 32 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Zemčík Pavel.

Algoritmus obsluhy virtuální čekárny

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Prof. Dr. Ing. Pavla Zemčíka.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Patrik Mrázek
16.5.2016

Poděkování

Tímto chci poděkovat vedoucímu mé práce Prof. Dr. Ing. Pavlu Zemčíkovi za cenné rady, podněty, konzultace a ochotu. Bez něj by tato práce nemohla vzniknout.

Obsah

1 Úvod

S virtuálními čekárnami se dnes již většina z nás setkává běžně a možná i podvědomě. Místo toho, abychom se zašli objednat do čekárny osobně, nebo zvedli telefon, raději otevřeme prohlížeč a vytvoříme si rezervaci na našem oblíbeném sportovišti, u známého kadeřníka, kvalitního automechanika, nebo u důvěryhodného lékaře. Zároveň však virtuální čekárny usnadňují život i poskytovatelům těchto služeb. Šetří jim čas při vyřizování objednávek po telefonu, vytváří přehledný časový plán pro každý pracovní den a často nabízí jejich provozovatelům mnohá další usnadnění.

Virtuální čekárny se od svého zrodu s rozvojem technologií dostupných na webu vyvíjí. Začaly podporovat například platební systémy nebo upozorňování přes SMS. Věřím, že v budoucnu bude chtít čím dál tím více poskytovatelů služeb umožnit klientům objednat se tímto pohodlným způsobem a ti, kteří již tuto službu nabízejí ji budou chtít vylepšovat. I proto má další práce a zdokonalování v této oblasti smysl.

Sám jsem dříve pro své známé vytvořil několik jednoduchých rezervačních systémů a seznámil se tak s jejich potřebami v různých oborech a příležitostmi pro další rozvoj těchto systémů. To představuje jeden z hlavních důvodů, proč mě zadání této práce zaujalo a proč jsem si jej vybral. Rovněž mě na tomto tématu zaujalo, do jakého širokého množství oborů tato práce zasahuje. Tím mám na mysli obory související s výpočetní technikou, do kterých jsem měl během svého studia možnost nahlédnout, ale také téměř univerzální uplatnění ve všemožných odvětvích služeb, na které se člověk může objednat.

Tato práce si klade za cíl vytvořit část virtuální čekárny, která bude vyřizovat požadavky klientů na rezervace a zároveň se bude snažit je optimalizovat tak, aby mezi jednotlivými klienty nevznikaly zbytečně dlouhé pauzy. Vzniklý program je navržen tak, aby byl použitelný napříč různými virtuálními čekárnami, nicméně vznikl jako součást kompletní aplikace a vyhovuje tedy jejím specifickým požadavkům.

Další kapitola je věnována aktuálně dostupným rezervačním systémům na bázi virtuálních čekáren, které se zabývají danou problematikou. Zároveň zde zmíním programovací jazyky jevící se jako vhodní kandidáti pro vytvoření mé práce. Následující kapitola zhodnotí současný stav, navrhne řešení zjištěných problémů a stanoví cíle, jež budu ve své práci sledovat. Ve čtvrté kapitole se dočtete princip, jakým tato práce řeší dříve popsané problémy, jaké prostředky jsem užil a do jaké míry byla práce úspěšná v jejich řešení na simulovaném případu.

2 Shrnutí dosavadního stavu

Tato kapitola si klade za cíl poskytnout čtenáři přehled o současném stavu v oblasti virtuálních čekáren a programových prostředků vhodných pro jejich vytváření. Nejedná se však zdaleka o encyklopedický výčet, na který by v této práci ani nebyl dostatek místa. Jsou zde tedy vybrány a zmíněny ty věci, které mají bezprostřední vztah k mé práci.

2.1 Virtuální čekárny

2.1.1 Co jsou to virtuální čekárny?

Virtuální čekárny jsou software, který umožňuje poskytovatelům služeb objednávat své zákazníky na konkrétní čas a službu po internetu. Nahrazují tak objednání osobní nebo po telefonu. Typické uplatnění pak najdou všude tam, kde je jeden, či více zákazníků obsluhováni v jeden čas a ostatní čekají, až se na ně dostane řada. Typicky je lze nalézt u poskytovatelů typu autoservis, lékař, kadeřník, sportoviště, zábavné atrakce, úřady apod. Naopak se liší od rezervačních systémů pro ubytovací zařízení, která mají své jiné specifické požadavky. Rovněž rezervační systémy pro kina neodpovídají koncepci virtuálních čekáren.

Aplikace se typicky skládá z klientské a serverové části, přičemž klientská část poskytuje své rozhraní jak zákazníkům, tak poskytovatelům. Klientská část mívá podobu webového rozhraní, aplikace pro mobilní zařízení nebo zařízení k tomu určeného ve fyzické čekárně.

Virtuální čekárna může být napojena, či být součástí účetních, skladových a podobných systémů, SMS brány apod. pro usnadnění a zefektivnění práce.

2.1.2 Dostupné virtuální čekárny

Existuje nepřeberné množství rezervačních systémů typu virtuální čekárna. Jejich výčet zde není vzhledem k rozsahu práce možný, proto uvedu ty nejznámější zástupce, jejichž využití není limitováno na konkrétní typ poskytovatele.

Reservio

Reservio nabízí rezervační kalendář, jehož konkrétní funkce se odvíjí od zvoleného cenového balíčku a typu podnikání. K dispozici je více než 70 různých typů podnikání a 9 jazyků. Kalendář je dostupný prostřednictvím webového prohlížeče na stránkách poskytovatele na webu Reservio, je však možné si vytvořit vlastní motiv vzhledu. Komunikace je vždy šifrována. Je možné využívat SMS upozornění pro klienty.

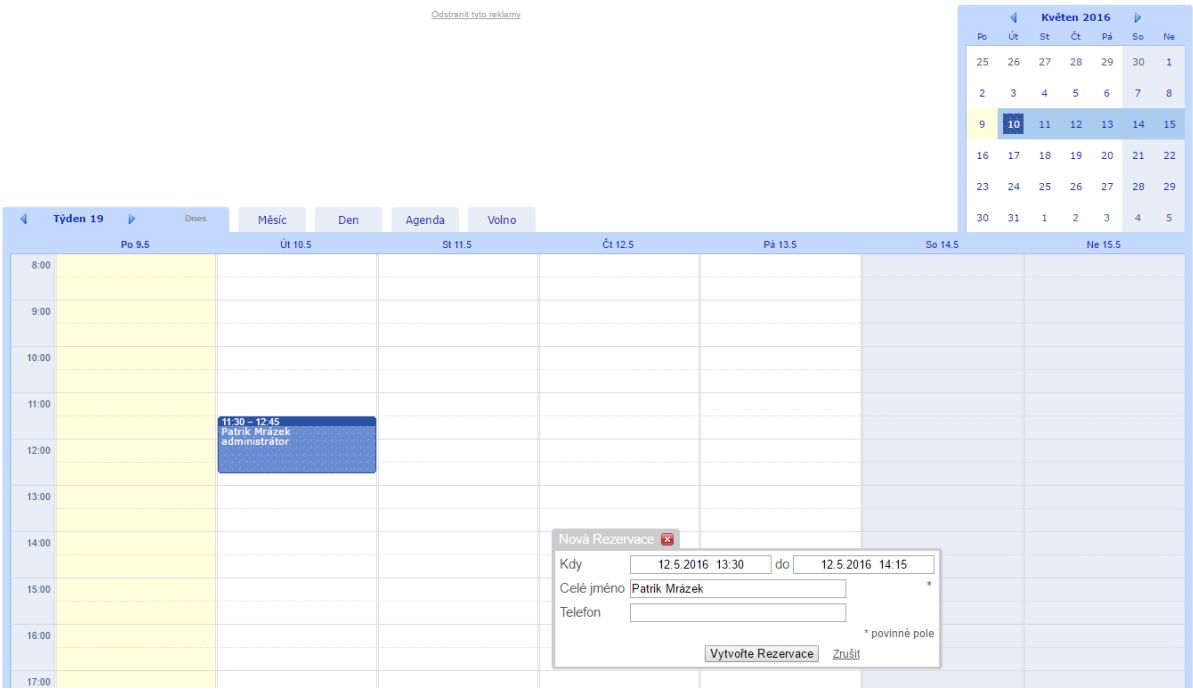
Klienti i správci mají vlastní účty, pod kterými se mohou přihlásit. Rezervační systém je pronajímán měsíčně. Cena se odvíjí od dostupných služeb a počtu rezervací v rozmezí 0 – 799,- Kč [1]

SuperSaaS

SuperSaas je svojí koncepcí velmi podobný Reservio, nabízí však mnohé funkce navíc. Opět zde najdeme konkrétní funkce kalendáře pro různé typy služeb, 29 jazyků, 26 měn, uživatelské účty, šifrovanou komunikaci, SMS a emailové notifikace. I způsob, jakým je kalendář dostupný je

obdobný. Každý poskytovatel má vytvořenou webovou stránku pro svoje rezervace. Barevné motivy jsou opět přizpůsobitelné.

Navíc SuperSaas obsahuje platební bránu, pokročilá pravidla pro rezervace, statistická vyhodnocení a algoritmy pro lepší využití otevírací doby. Systém je znovu pronajímán měsíčně s rozpětím cen od 0 – 750,- Kč v závislosti především na počtu rezervací. [2] Rezervační kalendář systému SuperSaaS i s formulářem pro vytvoření nové rezervace vidíme na obrázku 2.1



Obrázek 2.1:Rezervační kalendář systému SuperSaaS

Planyo

Planyo je další robustní systém, který přizpůsobuje své funkce konkrétnímu typu služeb. Všechny obvyklé funkce známé z předchozích systémů naleznete i zde. Upozorňování přes email a SMS, velké množství platebních bran (více než 40), uživatelské účty, zabezpečenou komunikaci, statistická data a přizpůsobování vzhledu pomocí šablon a kaskádových stylů.

Liší se však způsobem, jakým je rezervační kalendář klientům dostupný. Na výběr je zde aplikace pro Android a iOS a klasické webové rozhraní. Webové rozhraní se však integruje přímo do webové stránky poskytovatele. I cenový model je od předchozích zástupců poněkud odlišný. Licencovat software je možné třemi způsoby.

1. Bez měsíčních poplatků, poplatek se přičte k ceně rezervace, omezení ve využití
2. Měsíční pronájem, konkrétní cena se vypočítává převážně podle počtu rezervovatelných prostředků
3. Procentuální srážka z každé uskutečněné rezervace
   Software je zdarma k vyzkoušení na 30 dní. [3]

Booked

Dříve známý jako phpScheduleIt, je jediný zástupce dostupný bez omezení zdarma, s otevřeným zdrojovým kódem pod licencí GPL. Je tedy plně přizpůsobitelný, až na úroveň zdrojového kódu.

Z toho vyplývá poněkud odlišný přístup k problému. Nenajdeme zde platební brány, SMS brány, ani široké množství typů kalendářů dle poskytovaných služeb. Ostatní běžné funkce však jsou přítomny a to včetně jazyků, uživatelských účtů, statistických dat o využití, upozorňování klientů atd.

Zajímavostí je možnost propojit Booked se službami jako LDAP a Active Directory a možnost přidělovat jednotlivým uživatelům kvóty pro rezervace. [4]

bookingbug

Pravděpodobně nejrobustnější rezervační systém s virtuálními čekárnami. Větší část svého působení zaměřuje na náročné aplikace s velkým počtem přístupů jako jsou úřady, nemocnice apod. Nabízí však variantu uzpůsobenou i pro menší firmy a podnikatele, podobně jako předchozí zástupci. Zajímavostí je aktualizace zobrazovaných informací v reálném čase. Bookingbug rovněž dává svoji několikanásobně oceněnou technologii k dispozici vývojářům pro vytváření jejich vlastních rezervačních systémů skrze webové REST API a SDK založeném na technologii JavaScript.

Licencování je zde opět řešeno formou měsíčního pronájmu v závislosti na počtu personálu, rezervovatelných prostředků a vytvořených rezervací v přibližném rozmezí 15 – 50 GBP.[5]

Reservanto

Reservanto se zdá být typickým zástupcem, nepříliš se lišícím od většiny zde uvedených. Funkce jsou opět přizpůsobeny jednotlivým druhům služeb. Na výběr je 17 schémat. Nabízí standardní funkce jako platební bránu, SMS a emailová upozornění, uživatelské účty a statistiky. Navíc je tu webové REST API.

Reservanto se mírně liší cenovou politikou, i přestože je pronajímán měsíčně s cenami od 0 do 1249,- Kč. Základní verze, která je zdarma, totiž nabízí neomezené služby, zákazníky, služby, platební bránu a SMS notifikace. [6]

FITSYSTEM

Jedná se o univerzálně navržený systém bez funkce přizpůsobování se konkrétní službě. Je postaven právě na principu virtuálních čekáren. Dostupný je formou webového rozhraní a to jako kalendář, nebo vyskakovací okno s rezervačním formulářem. Funkce jako platební brány, uživatelské účty, upozorňování, statistiky apod. jsou opět přítomny.

FITSYSTEM je vybudován modulárně, přičemž každý modul má stanovenou svoji cenu za měsíc využívání. Je tak možné používat základní systém zcela zdarma bez omezení rezervací, klientů nebo aktivit. Cena kompletního systému se všemi moduly se pak pohybuje kolem 1800,- Kč měsíčně. [7]

Rezervačník

Rezervačník je systém virtuálních čekáren zaměřený speciálně na sportoviště. Konkrétně je přizpůsoben pro 8 typů sportovišť. Nabídka funkcí je oproti ostatním zástupcům menší. Nenajdeme zde statistiky využití, platební brány ani typické upozorňování zákazníků. Systém je typicky provozován na webu Rezervačníku. Cena se pohybuje od 0 – 990,- Kč podle počtu rezervací. Varianta zdarma nenabízí podporu. [8]

2.2 Programové prostředky

2.2.1 Programové prostředky pro web

Virtuální čekárny musí klientům umožnit objednat se po internetu, a tak i přestože může klientská aplikace existovat ve variantách pro mobilní zařízení, je vždy nabízeno i webové rozhraní dostupné skrze webový prohlížeč. To do značné míry omezuje volbu programových prostředků. Stejně jako v případě virtuálních čekáren i u programových prostředků určených pro vývoj webových aplikací nalezneme obrovské množství vhodných kandidátů se specifickým zaměřením. Proto je tato kapitola věnována jen těm nejrozšířenějším zástupcům využitelných v mé práci. Nutno také podotknout, že vedle zde zmíněných způsobů vykonávání kódu existují obvykle i alternativní verze, např. kompilátor pro PHP apod. Uvažujeme však nejběžnější způsob užití.

2.2.2 ASP.NET

ASP.NET je součástí .NET Framework společnosti Microsoft pro budování webových aplikací. Je vybaven nástroji pro usnadnění řešení problémů, kterým webový vývojář musí čelit, jako je nasazování systému, konfigurace, nebo dodržení stejného vzhledu a funkčnosti na všech zařízeních a ve všech webových prohlížečích podporovaných verzí. Jedná se tedy o velmi kompletní sadu nástrojů, nejen o samotný programovací jazyk.

Pro .NET Framework typicky, se i kód v jazyce ASP.NET překládá nejprve do jazyka CIL, což je jazyk nízké úrovně, nezávislý na platformě. Tento kód je posléze přeložen do nativního strojového kódu a vykonán pomocí CLR jenž je společný všem jazykům .NET. CLR se zároveň stará o automatickou správu paměti, typovou bezpečnost, strukturované zpracování chyb a poskytuje fond vláken pro nejběžnější asynchronní operace. Překlad pochopitelně neprobíhá pokaždé celý znovu při každém požadavku na webovou stránku, jen při prvním požadavku.

ASP.NET je objektově orientovaný. Přestože první verze poskytovaly pouze slabý objektový model nad některými často využívanými komponenty, dnešní verze jsou již plně objektové. Během vývoje byly také zaznamenány nové trendy v oblasti webových aplikací. Jedním z nich je užívání technologie AJAX pro dynamické změny obsahu na stránce bez nutnosti jejího přenačtení. Na tuto změnu odpověděl Microsoft abstraktní vrstvou ASP.NET AJAX. Druhým ze zaznamenaných trendů je model webové aplikace MVC. Ten Microsoft reflektoval skrze ASP.NET MVC. Potlačuje tak sice některé myšlenky a praktiky, které ASP.NET nastavilo, prosazuje však pojetí MVC, ktere se pro některé stalo příhodnějším modelem webové aplikace a otevírá nový pohled na věc.

Pro provoz webu postaveného na ASP.NET je pak zapotřebí placený Microsoft Server s nainstalovaným IIS. [9]

2.2.3 JSP

JSP neboli JavaServer Pages je další technologie pro tvorbu dynamických webových stránek, jak již název napovídá, založená na programovacím jazyce Java. Stojí za ní tedy, stejně jako za jazykem Java, společnost Sun Microsystems. V nejjednodušším pojetí jsou části kódu jazyka Java vkládány do HTML stránky a před odesláním požadované stránky klientovi vykonány na serveru, v praxi však lze využít naplno potenciál objektového pojetí jazyka Java.

Pro provoz stránek založených na JSP je zapotřebí webového serveru s nainstalovaným JSP kontejnerem. Překlad se provádí opět na vícekrát. JSP soubory se zdrojovým kódem jsou převedeny do jazyka Java. Tento převedený kód se pak přeloží do bajtkódu nezávislého na cílové architektuře. Bajtkód je následně interpretován virtuálním strojem Java. I zde se využívá již jednou přeložených

tříd, aby celý překlad nemusel probíhat pokaždé znovu při každém požadavku na webovou stránku. Princip je tedy obdobný jako u ASP. [10]

2.2.4 PHP

Původně osobní projekt vznikající v roce 1995 pro potřeby osobních domovských stránek. Právě tak vznikl i název jazyka původně značící „Personal Homepage“. PHP však upoutalo pozornost a po vícerých úpravách a vylepšeních se již v roce 1998 ukázalo, že PHP je nainstalováno na více než jednom milionu domén.

To byl však teprve začátek. Autoři se rozhodli pro přepracování mnohých problematických částí. Výsledkem byla v roce 2002 nová verze jazyka s pořadovým číslem 4. Nabídla výrazně rychlejší zpracování skriptů, vrstvu serverové abstrakce a další novinky. Tato verze byla provozována na přibližně 15 milionech domén. Další, pátá verze pak přinesla velké změny v oblasti objektově orientovaného zápisu. Ten byl sice již přítomný od verze 3, ale pouze v omezené míře. [11]

Provoz stránek založených na PHP vyžaduje webový server s nainstalovanou podporou PHP. I zde najdeme jistou podobnost při vykonávání kódu. Kód se přeloží do bajtkódu nezávislého na architektuře. Až tento kód se interpretuje. Ani v případě PHP pak neprobíhá celý překlad znovu při každém požadavku na načtení stránky. Přeložený kód se ukládá do mezipaměti. [12]

Aktuální verze PHP je v době vzniku této práce 5.6.2. Paralelně však již existuje alfa verze PHP 7. To slibuje další zásadní změny včetně rapidního nárůstu výkonu. [13]

Pro PHP existuje celá řada aplikačních rámců. Jejich cílem je mimo jiné podporovat znovupoužitelnost kódu, usnadnit a zpřehlednit vývoj webových aplikací nebo prosazovat dříve zmíněnou filozofii k tvorbě webu – MVC [14]

3 Zhodnocení současného stavu a plán práce

3.1 Zhodnocení současných virtuálních čekáren

Přizpůsobitelnost

Z široké nabídky rezervačních systémů na bázi virtuálních čekáren je dobře patrný trend přizpůsobovat vzhled a funkčnost konkrétní aplikaci a vytvořit takovýchto přednastavených šablon velké množství pro okamžité nasazení. Cílem je nejspíše zvýšit intuitivnost užití pro uživatele a využít v každé konkrétní aplikaci maximum ze systému. Pakliže systém toto přizpůsobení nenabízí, je obvykle velmi úzce zaměřen. Z mého pohledu je takový krok logický a patrně i prověřený.

Co se týká přizpůsobitelnosti vzhledu, zde již trend tak jasný není. Dá se sice očekávat, že poskytovatelé služeb, kteří si zakládají na své značce budou chtít uzpůsobit vzhled svého rezervačního systému, některé rezervační systémy však tuto možnost nemají. Jiné pak poskytují možnost ovlivnit pouze barevné schéma, u některých lze vytvořit vlastní stylový předpis.

Cena

Jen minimum systémů je dostupných zdarma, převážně s omezeními. Většina je pak pronajímána měsíčně, i když Planyo nabízí více modelů licencování. Cena většiny systémů je navíc velmi podobná. Z mého pohledu chybí systém, který by byl dostupný zdarma a zároveň byl široce použitelný poskytovateli bez nutnosti hlubší znalosti správy webu.

Efektivita při plánování

I přes technickou pokročilost mnohých zmíněných systémů je stále opomíjenou kategorií efektivita při plánování. Je přirozené, že lidé se při výběru času příliš neřídí cílem využít efektivně otevírací dobu. Vznikají tak mezi jednotlivými klienty nevyužitelné časové mezery. Ty pak v důsledku znamenají méně obsloužených klientů, delší čekací doby a ztrátu peněz. Klienti navíc u vytížených poskytovatelů vyžadují zkrátit čekací doby. Vyplývá to z výsledků průzkumu v čínské, velmi vytížené nemocnici.[15]

Jediný zástupce v předchozím výčtu, který se dle dostupných informací touto problematikou zaobírá je SuperSaaS. Ten daný problém řeší tím, že umožňuje manuálně schvalovat termíny, nebo využít algoritmy, které se snaží stejných výsledků docílit.

Dle mého názoru se jedná o jeden ze strategických, ne-li klíčových prvků virtuálních čekáren. Toto tvrzení dokládá i dříve zmíněný průzkum. Pouze jediný uvedený rezervační systém však tento problém řeší.

3.2 Zhodnocení programových prostředků

Bylo by zbytečné řešit, který jazyk je lepší. V každém případě se jedná o vyspělé technologie s určitými technickými výhodami a pochopitelně i nevýhodami. Všechny nepochybně splňují

požadavky pro tvorbu mé práce. Rovněž úvahy o tom, který jazyk je intuitivnější nebo se v něm bude vyvíjet snáze by byly značně neobjektivní. Chceme-li však vzít v potaz cenový aspekt, technologie ASP.NET má zvýšené nároky kvůli licencování.

PHP je v poslední době velmi oblíbený jazyk a jeho obliba narůstá. Stalo se tak jednou z nejrozšířenějších platforem pro provozování webu na světě. Celosvětově je instalován na více jak třetině webových serverů. [11] To z něj činí vhodného kandidáta pro široké rozšíření mezi běžné poskytovatele služeb, pakliže tento vlastní webovou prezentaci využívající PHP. Pro PHP také hovoří fakt, že z výše zmíněných technologií mám s touto již předchozí zkušenosti. Zvolil jsem jej tedy jako programový nástroj pro tvorbu mé práce.

3.3 Cíl práce

Cílem mé práce bude vytvořit algoritmus pro obsluhu požadavků ve virtuální čekárně v jazyce PHP 5.5 tak, aby byl použitelný jako knihovna v libovolném rezervačním systému typu virtuální čekárna vybudovaném ve stejném jazyce.

Klient bude zajisté od systému očekávat, že mu ušetří čas při rezervování, snadný přístup na různých zařízeních, přehledné a intuitivní ovládání, možnost platit online, nebo mít svůj účet s kreditem a také, že ho bude informovat o důležitých změnách. Klient pak zajisté ocení, když bude jeho požadavek ve většině případů vyřízen kladně a pokud ne, věřím, že kladně uvítá návrhy, na kdy je možné rezervaci vytvořit.

Poskytovatel na druhou stranu bude pravděpodobně očekávat úsporu času při vyřizování rezervací, efektivnější využití otevírací doby a zvýšení tržeb nebo nové klienty a také napojení na případné další systémy, plánování času a návrat prostředků do pořízení a udržování systému vložených.

Ne všechna očekávání lze v mé práci splnit. Některá pak přímo souvisí s klientskou aplikací, nebo s jinou částí rezervačního systému, který není součástí mé práce. Bylo by však vhodné při návrhu postupovat tak, aby jich bylo splněno co největší množství, popřípadě nebylo bráněno realizaci těch, které souvisí s jinou částí systému.

Algoritmus se bude chovat tak, že se bude snažit svým chováním omezit vznik nevyužitelných časových oken v otevírací době poskytovatele. Toho se pokusí docílit tím, že bude schvalovat příchozí požadavky na registrace, popřípadě je mírně přesouvat, bez nutnosti souhlasu, v časovém rámci stanoveném poskytovatelem, nebo zamítat a navrhovat nové.

Bude tedy k dispozici také funkce pro navrhování termínů. Ta bude využita jak již byl zmíněno pro návrh vhodnějších termínů v případě, že nebude požadovaný čas schválen. Její další využití tkví v navrhování termínů klientům, kterým bude muset být z rozličných důvodů jejich rezervace zrušena. Např. pro nemoc, poruchu vybavení, dovolenou,... Další využití najde navrhování termínů ve funkci pro obsluhu přednostních případů.

Mimo jiné by měly rozhodovací algoritmy uvažovat i pravděpodobnost, s jakou se požadavky na jednotlivé služby mohou vyskytnout. Cílem této schopnosti je další zefektivnění algoritmu.

Zároveň musí celý aparát počítat s možností paralelní obsluhy klientů. Je tedy možné, pokud to poskytovatel podporuje, v jeden okamžik obsloužit dva a více klientů, přičemž každá obslužná fronta může poskytovat jiné služby. To je důležitá vlastnost, kterou není možné simulovat pouhým zdvojením poskytovatele, aby bylo možné klienty variabilně přesouvat mezi obslužnými frontami,

je-li to vhodné. Systém musí pochopitelně rovněž uvažovat otevírací hodiny a případné přestávky během dne.

Výhodou taktového řešení bude možnost integrovat jej do celé řady systémů bez ohledu na typ poskytovaných služeb. Vzhledem k rozšíření PHP na webových serverech pak bude možné provozovat řešení téměř všude. Zároveň knihovna bude nabízet funkce, které uživatelé a poskytovatelé zajisté ocení, ale v dnes dostupných rezervačních systémech se nevyskytují, nebo jen v malé míře a rozhodně nejsou běžné. Očekávám také rychlou práci algoritmů, která nebude server zbytečně zatěžovat.

Nevýhodou tohoto řešení bude, že při integraci do jiných systémů bude nutné knihovnu přizpůsobit tak, aby spolupracovala s jejich databázovým modelem a komunikovala řádně s klientskými aplikacemi. Nastavení knihovny tak, aby splňovala dobře svoji práci, bude nutné uzpůsobit podle konkrétní sady služeb daného poskytovatele. Ke zjištění vhodných hodnot může v budoucnu sloužit simulace.

3.4 Návrh ověření řešení

Pro ověření úspěšnosti algoritmu, přesněji řečeno, do jaké míry je schopný zefektivnit využití otevírací doby a zvládat příchozí požadavky, bude vytvořena simulace. Bude vygenerována série požadavků na rezervaci služby se pseudonáhodnými časy a typy služeb. Čistě pro názornost využiji příklad autoservisu, bylo by však pochopitelně možné použít obecný model.

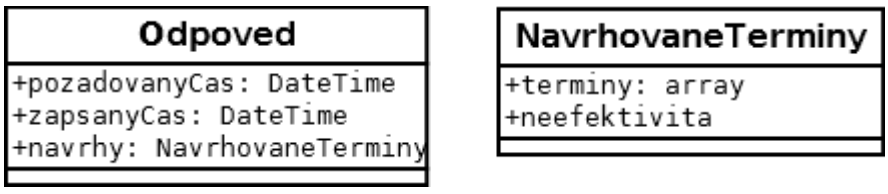
Simulace pak proběhne s různými nastaveními maximální doby, o kterou je možné klienta bez schválení přesunout, různým minimálním ohodnocením uživatelem požadovaného termínu, který bude přijat a bude z něj vytvořena rezervace a různou velikostí časových oken, tedy minimálního časového úseku, s kterým může algoritmus pracovat. Celá simulace se pak spustí znovu, tentokrát však bez využití optimalizačního algoritmu.

Při simulaci budou sledovány procentuální hodnoty využití otevírací doby, uspokojení požadovaného času zákazníka a počet přeplánovaných požadavků. Výsledky budou vyhodnoceny a shrnuty do závěru.

3.5 Parametry výstupu

Vytvářená knihovna bude nabízet různé druhy akcí mající, dle očekávání, velmi různorodý výstup. Při obsloužení požadavku klienta na rezervaci mu bude buď vrácen konkrétní čas, na který mu byla rezervace vytvořena, nebo mu bude zasláno několik návrhů. Taková odpověď bude zastřešena navrácením instance třídy `Odpoved`. Ta bude obsahovat původně požadovaný čas a podle výsledné operace buďto výsledný, zapsaný čas, nebo návrhy.

Návrhy budou reprezentovány v třídě `NavrhovaneTerminy`, která informaci o navrhovaných termínech nese v proměnné `$terminy`, která obsahuje pole instancí třídy `Termin`. Ostatní proměnné a metody třídy `NavrhovaneTerimny` jsou používány pouze interně. Obě třídy používané pro výstup se nachází ve jmenném prostoru `model` a jsou znázorněny diagramem tříd na obrázku 3.1.



Obrázek 3.1: Diagram tříd znázorňující třídy používané pro výstup

Informace o přeplánování termínů by pak měla být při nasazení realizována jako notifikace pro dotčené uživatele. Způsob doručení notifikace se pak bude lišit podle preferencí uživatele a poskytovatele. Může například nabývat forem SMS, emailové zprávy nebo upozornění na mobilním zařízení skrze rezervační aplikaci. Pro účely testování budou informace tištěny na výstup programu.

Požadavek na zařazení akutního případu vrací hodnotu `true`, byl-li případ zařazen, nebo vyvolá výjimku, není-li možné požadovaný čas dodržet. Dochází-li při zařazení k přeplánování jiných rezervací, budou tyto informace, v rámci testování, tištěny na výstup programu, obdobně jako v předchozím případě.

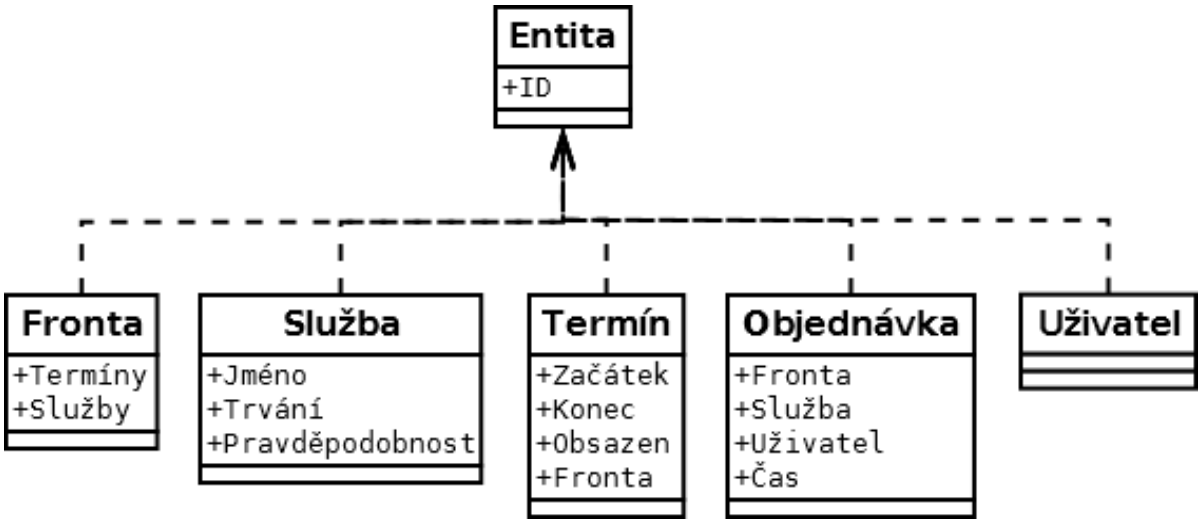
4 Popis vlastní práce

Práce je napsána v jazyce PHP 5, ten umožňuje jak objektový, tak procedurální zápis. [11] Rozhodl jsem se moji práci vypracovat objektovým zápisem. Ten je v PHP 5 mnohem propracovanější a pokročilejší, než v jeho předchozích verzích. [11] Dalšími důvody pak jsou větší míra intuitivnosti, flexibilita a znovupoužitelnost objektově orientovaného návrhu. [16]

4.1 Princip fungování práce a implementace

4.1.1 Databáze

Jelikož tato práce nevznikla jako samostatný útvar, ale je součástí kompletního rezervačního systému, spolupracuje s databází, kterou navrhoval jiný člen týmu. Ten poskytuje svoje rozhraní pro práci s databází. Toto rozhraní však není součástí mé práce, proto jej pouze simuluji. Část struktury databáze, se kterou pracuji, pak odráží třídy v jmenném prostoru `model`. Tyto třídy dědí od abstraktní třídy `Entita`. Ta poskytuje prvky společné všem třídám jmenného prostoru `model`, pro tuto chvíli pouze jednoznačný identifikátor `$ID`. Třídy reflektující užívanou část databáze znázorňuje diagram tříd na obrázku 4.1.



Obrázek 4.1:Diagram tříd reflektující část struktury databáze, se kterou algoritmus pracuje

Přestože tyto třídy reflektují strukturu databáze navrženou jiným členem týmu, je struktura dostatečně obecná na to, aby ji bylo možné použít v libovolném rezervačním systému, nanejvýš s drobnými úpravami. Třída `Uzivatel` je pro zatím prázdná. Je zde přítomna kvůli potřebě zasílat uživateli notifikace. Tyto funkce však mohou být řádně implementovány až v kombinaci se zbytkem rezervačního systému.

Databáze je simulována vzorovými daty uloženými v souborech typu JSON v adresáři `/vzorovaData`. Volba notace pro uložení vzorových dat byla spíše nahodilá, data by mohla být reprezentována stejně tak dobře například pomocí XML. Vzorová data pak obsahují mimo

jiné seznam služeb poskytovatele. Pro lepší názornost jsem zvolil služby autoservisu a také jednotlivé, částečně obsazené fronty na jeden den tak, aby na nich bylo možné otestovat navržené případy určené k ověření funkčnosti celého díla. Bude-li třeba načíst data pro další dny, budou tato data obsahovat prázdné, tedy nezaplněné fronty.

4.1.2 Zpracování požadavku

Třída `AnalyzaPozadavku` obsahuje čtyři veřejné metody (vyjma konstruktoru), ty reprezentují čtyři základní funkce, které algoritmus poskytuje. Jsou to metody `analyzujTermin`, `akutniPripad`, `preplanujTerminy`, `navrhTerminu`. Jejich účel je vystižen názvem a detailněji pak popsán v kapitole 3.3. V této kapitole bude zmíněn způsob, jakým se jednotlivé metody chovají. V následující kapitole pak budou popsány klíčové algoritmy a způsob jejich implementace.

Analýza termínu zvoleného klientem

Metoda `analyzujTermin` přijímá jako parametr datum a čas, na který se chce klient objednat a identifikátor požadované služby. Nejdříve načte fronty pro dané datum a odfiltruje fronty, které nenabízí požadovanou službu. Pro zbývající fronty vytvoří instanci třídy `AnalyzaFronty`. Tato třída slouží k analyzování konkrétní čekací fronty, jež obsahuje časová okna pro jeden den.

Časové okno je zde chápáno jako nejmenší časový úsek, s kterým může systém pracovat a jeho velikost je individuální podle potřeb každého poskytovatele služeb. Každé časové okno nese dvě časové informace, jeho začátek a jeho konec. Díky tomu lze rozlišit pauzy během otevírací doby i samotné otevírací hodiny zvlášť pro každou čekací frontu.

V každé vytvořené instanci třídy `AnalyzaFronty` je volána metoda `analyzujFrontu`. Metoda najde pro požadovaný čas odpovídající časové okno. Každá služba musí mít přiřazenou svoji délku. Toho je zde využito ke zjištění, zda je pro požadovaný čas možné klienta objednat. Pokud to možné je, zjistí dále metoda, zda pro tento případ nedojde k neefektivitě ve využití otevírací doby. O algoritmu a přesném postupu, jak se toto zjistí se zmíním v dalších kapitolách.

Pakliže by k neefektivitě došlo, zkusí algoritmus neefektivitu omezit přesunem požadovaného času v rámci malého časového úseku stanoveného poskytovatelem. Např. 5 – 15 minut.

V tuto chvíli již tedy máme pro každou frontu, daný čas a službu výsledek. Ten může nabývat následujících stavů:

- Termín je volný, k neefektivitě nedochází
- Termín je volný, k neefektivitě nedochází za předpokladu, že požadovaný čas mírně pozměníme
- Termín je volný, každopádně však dojde k neefektivitě
- Termín je obsazený

Výsledek je nyní ohodnocen celým nezáporným číslem. Čím vyšší je číslo, tím lepší řešení tato fronta nabízí. Výsledek je ohodnocen na základě toho, zda-li byl splněn čas požadovaný klientem, kolik volných časových oken se stane v případě obsazení termínu nevyužitelnými (např. protože pro žádnou službu již nebude volných oken dostatečné množství) a jakou mají tato okna délku. Dále pak na četnosti s jakou se rezervovaná služba vyskytuje v závislosti na celkovém počtu služeb.

$$b = 100 - n * t/5 * 2 - |r| * t/5 + 5p$$

Vzorec 4.1: Vzorec pro ohodnocení řešení

Vzorec 4.1, kterým je řešení ohodnoceno, byl zkonstruován experimentálně. Jsou v něm však patrné závislosti, na kterých staví. Nejdříve si objasníme užité proměnné. b je počet bodů, kterým bude řešení ohodnoceno. Proměnná n představuje počet časových oken, která se stanou nevyužitelnými v případě, že bude řešení realizováno, t je délka časového okna v minutách a r reprezentuje posun v časových oknech, o které byl klientův původní návrh posunut. Proměnná r může nabývat i záporných hodnot, pokud má být přesun proveden na dřívější čas. Nakonec p udává pravděpodobnost, s jakou se rezervovaná služba vyskytuje. Tato informace je uložena v databázi a reprezentována desetinným číslem mezi 0 a 1.

Konstanta 100 je zde použita jako výchozí počet bodů. Prvně jsou odečteny body za vzniklá nevyužitelná časová okna. Délka časového okna může být u každého poskytovatele jiná, je však vhodné, aby neměla vliv na výsledný počet bodů. Proto jsou odečítané body vynásobeny délkou časového okna v minutách a poděleny konstantou 5. Za každých 5 nevyužitelných minut je tak odečten 1 bod. Vliv takových oken na hodnocení by však měl být větší, proto je vynásoben dvěma.

Dále odečteme body za přesun oproti času požadovanému klientem. Je zřejmé, že termín, který klientův čas respektuje by měl být ohodnocen lépe, než ten, který klienta přesouvá. Protože přesun je reprezentován časovými okny, je i zde užito násobení délkou okna a následné dělení číslem 5, čímž se každých 5 minut odchylky oproti požadovanému času projeví hodnocením nižším o jeden bod.

Nakonec jsou přičteny body za pravděpodobnost s jakou se služba vyskytuje. Vzhledem k faktu, že pravděpodobnost nabývá hodnot od 0 do 1, neovlivnila by příliš výsledné bodové hodnocení. Proto je násobena koeficientem 5.

Ohodnocením řešení úloha třídy `AnalyzaFronty` končí. Ve třídě `AnalyzaPozadavku` se nyní vybere řešení, které bylo nejlépe ohodnoceno. Pokud žádné řešení nedostalo minimální požadovaný počet bodů, jsou klientovi navrhnuty jiné termíny. Je-li i přes prvotní zamítnutí klientova požadavku navrhnut termín se stejným časem, je takový termín považovaný za automaticky přijatý. O principu navrhování termínů se zmíním později. Minimální počet bodů, které musí řešení obdržet, aby mohlo být vybráno je individuální a nastavitelné. Nejmenší přípustná bodová hodnota je však 1. Nižší hodnocení, tedy 0 bodů znamená, že termín je obsazený a nelze jej zarezervovat.

Toto řešení, založené na ohodnocování konkrétních případů a vybírání toho nejlépe ohodnoceného, je známé z umělé inteligence. Využívají ho například některé metody pro hraní her, třeba Minimax nebo Alfa-Beta. [17]

Návrh termínů

Navrhování termínů se využívá v případech, kdy algoritmus zamítne klientem požadovaný čas a navrhne mu tak svoje vlastní, dále v případě, že je potřeba přeplánovat rezervace, například z důvodu poruchy, dovolené apod. Klientům se tak může zaslat upozornění s novými navrženými termíny. Posledním případem, kdy této funkce využijeme je přeplánování klientů z důvodu nutnosti obsloužit akutní případ.

Metoda `navrhTerminu` implementuje jednu z klíčových vlastností této práce. Jako parametry přijímá datum a čas od kterého má proběhnout návrh termínů a identifikátor služby, pro kterou jsou

termíny navrhovány. Volitelně pak přijímá počet termínů, který při volání metody chceme naplánovat (výchozím počtem jsou 3 termíny). Nakonec přijímá volitelně informaci o tom, zda navržený termín bude blokován pro ostatní návrhy.

Postup je do jisté míry obdobný jako u schvalování termínu požadovaného uživatelem. V první řadě se pro dané datum načtou fronty. Ty jsou pak filtrovány podle toho, zda nabízí požadovanou službu, či nikoliv. Pro fronty, které požadovanou službu nabízí jsou pak vytvořeny instance třídy `AnalyzaFronty`. V každé instanci je volána metoda `navrhniTermin`.

Návrh termínu je uzpůsoben tak, aby předcházel neefektivnostem. Pokud je k dispozici větší počet časových oken, nežli je třeba pro zařazení požadované služby, je vždy vybrán termín ze začátku a konce takové série oken. Dříve obsazená okna tak navazují na nově navrhovaný termín a nevznikají nevyužitelná časová okna.

V každé frontě jsou termíny postupně navrhovány a ohodnocovány obdobně jako v případě analýzy termínů. Rozdíl je pouze v tom, že nelze uvažovat vyhovění času požadovaného klientem, protože při navrhování termínů takový čas neexistuje. V rámci fronty jsou termíny s nižším ohodnocením zahazovány a nahrazovány termíny s lepším ohodnocením. Má-li nový termín stejné ohodnocení jako předcházející termín, jsou k nabídnutí vybrány oba.

Ve třídě `AnalyzaPozadavku` se tyto navržené termíny z každé fronty ukládají. Ve chvíli, kdy je dosaženo počtu navrhovaných termínů je navrhování u konce a termíny nabídnuty. Pokud se pro načtené fronty nepodaří navrhnout dostatečný počet termínů, načtou se fronty pro další dny. Tento postup se opakuje, dokud se od původního data neodchýlíme o maximální počet dní určený konstantou.

Přeplánování termínů

Přeplánování termínů slouží, jak již bylo zmíněno, k obsluze případů, kdy není možné v danou dobu obsloužit některou frontu, ať už z jakéhokoliv důvodu. Postup je zde velmi triviální a popíšu jej tedy pouze krátce.

Metoda `preplanujTerminy` přijme parametry určující začátek a konec období, kdy nebude možné obsluhovat klienty. Volitelně pak přijme pole front, kterých se to týká. Pokud tento parametr nebude poskytnut, budou uvažovány všechny fronty, které spadají do intervalu určeného prvními dvěma parametry.

Pro každou zvolenou frontu pak budou načteny všechny rezervace spadající do daného intervalu. Tyto rezervace budou zrušeny a budou jim navrhnuty nové termíny skrze již dříve popsanou metodu `navrhTerminu`. Zde využijeme parametr této metody, který zajišťuje, že navržené termíny budou blokovány pro další navrhování. Tím zabráníme tomu, aby byl dvěma a více klientům navrhnut stejný termín.

Akutní případ

Metoda pamatující na obsloužení přednostního případu `akutniPripad` může do jisté míry připomínat chování metody pro přeplánování termínů. Rozdíly opravdu nejsou na první pohled patrné. Zatímco přeplánování termínů navrhuje pro každý termín, který spadá do zadaného intervalu nový vhodný termín, akutní případ se nejdříve snaží využít případné nevyužité mezery v otevírací době jiných front.

Chování metody `akutniPripad` je zpočátku obdobné ostatním zmíněným metodám. Přijme parametry obsahují datum a čas, kdy má být přednostní případ obsloužen a službu, která mu má být

poskytnuta. Dále pak načte fronty pro daný datum a odfiltruje fronty, které nenabízí danou službu. Následně zkontroluje, zda pro zadaný čas není v některé z filtrovaných front volno. Je-li volno, vytvoří v takové frontě rezervaci a to bez využití optimalizačního algoritmu.

Není-li volno, podívá se algoritmus do front, které danou službu nenabízí, zda do nich není možné, na jejich původní čas, přesunout rezervace z jiné fronty, která danou službu nabízí, ale je obsazená jiným klientem. Tím by se uvolnilo místo pro akutní případ. Toto řešení je pro klienty velmi příjemné, neboť akutní případ bude obsloužen a přesunutého klienta se změna nijak nedotkne – bude stále objednán na stejný čas, jen bude obsloužen jinou frontou, než jakou mu původně algoritmus přiřadil.

Poslední možností je, že pro vytvoření okna pro obsloužení přednostního klienta bude nutné přesunovat klienty. Algoritmus v takovéto situaci vybere pro zařazení akutního případu takovou frontu, kde bude počet přeplánovaných rezervací co možná nejmenší. V této frontě pak vezme rezervace, které zasahují, byť jen z části do času obsluhy akutního případu, zruší je a navrhne jejich klientům nové termíny.

4.1.3 Klíčové algoritmy

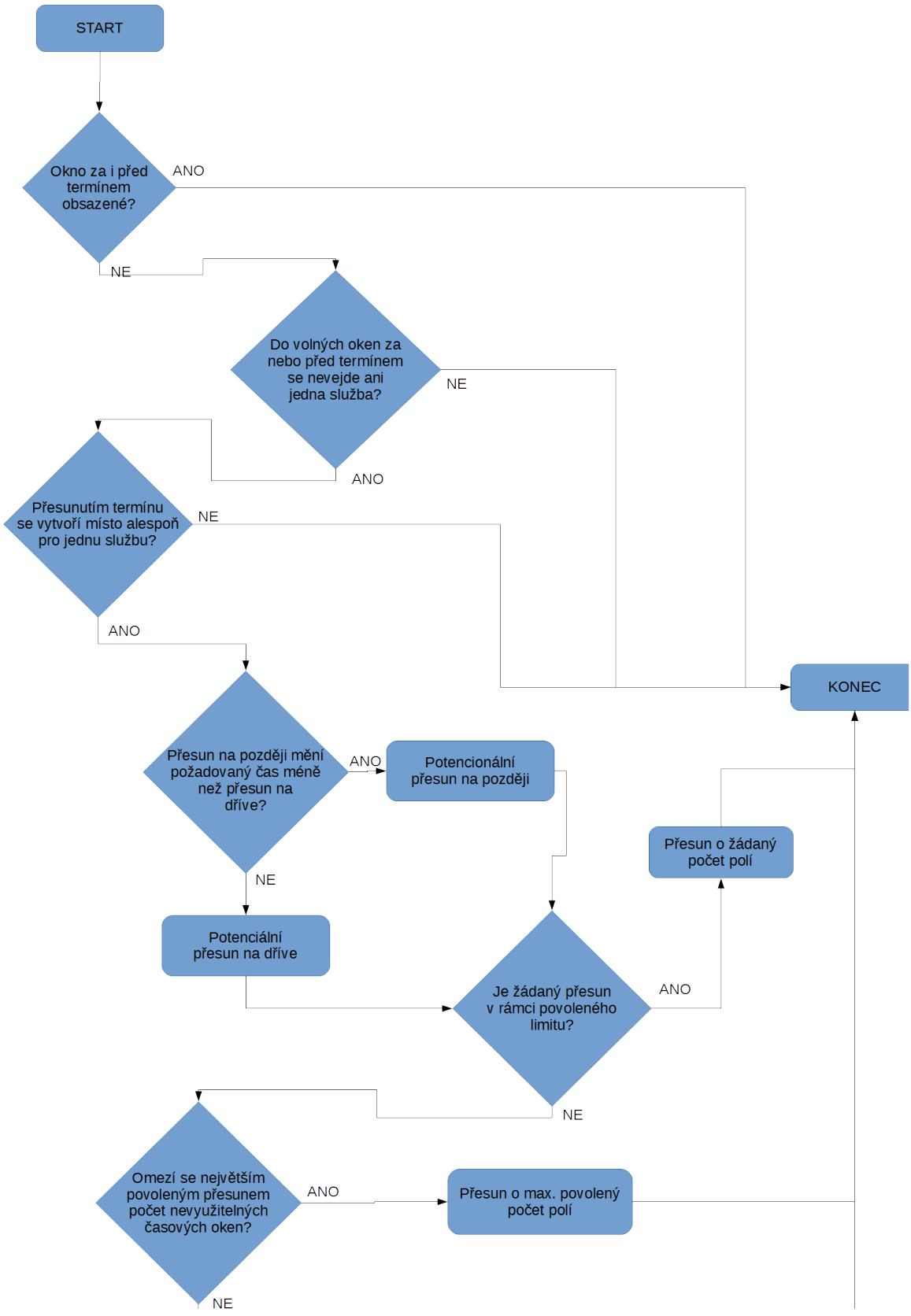
V této kapitole se věnuji detailnímu popisu klíčových algoritmů použitých v mé práci. Jedná se o algoritmus pro analyzování požadavku klienta a navrhování termínů. Tyto algoritmy, jak se domnívám, nejsou natolik triviální, aby stačil jejich zběžný popis z předchozí kapitoly.

Postupy zde popsané jsou vždy užity pro jednu frontu. Implementaci tedy nalezneme ve třídě `AnalyzaFronty`. Jak je nakládáno s výstupy z jednotlivých front zmiňovala v dostatečné míře již předchozí kapitola.

Analýza termínu zvoleného klientem

V prvním kroku vyhledáme časové okno odpovídající času požadovaného klientem. O to se stará metoda `terminProDanyCas` nacházející se ve třídě `Fronta`. Pokud fronta požadovaný čas neobsahuje, vyvolá výjimku. Ta se propaguje do třídy `AnalyzaPozadavku`, kde je zachycena a ošetřena. Následuje jednoduché zjištění, zda je požadované časové okno volné, nebo již obsazené a spočtení volných za sebou jdoucích časových oken před a po vybraném oknu. To probíhá iterací nad časovými okny v aktuálním směru. Ve směru od okna dále se musí shodovat koncový čas tohoto okna a počáteční čas následujícího. Tuto operaci obstarávají metody třídy `Fronta` `volnychTimeslotuPredTerminem` a `volnychTimeslotuPoTerminu`. Je-li tato podmínka porušena, okna na sebe nenavazují a může se jednat například o pauzu na oběd. Nyní zjistíme trvání požadované služby na základě jejího identifikátoru. V tomto kroku již nekontrolujeme, zda fronta požadovanou službu nabízí. To je zajištěno tím, že instance třídy `AnalyzaFronty` se vytváří pouze pro fronty, které danou službu nabízí. Posledním krokem před započetím analýzy možné neefektivnosti je pak ověření toho, že požadovaná služba má v časovém rozvrhu dostatek prostoru pro její zařazení.

Neefektivitu analyzujeme na základě počtu volných, po sobě jdoucích časových oken před požadovaným termínem a počtu volných, po sobě jdoucích časových oken za posledním oknem, které by služba v případě jejího zařazení obsadila. Pro uchování této informaci je zavedena proměnná `$volnychTimeSlotuZaSluzbou`.



Obrázek 4.2:Vývojový diagram algoritmu analyzující nevyužitelná časová okna

Dalším důležitým faktorem při posuzování možné neefektivity je délka nejkratší služby v časových oknech, kterou daná fronta nabízí. Její zjištění je triviální, ukládáme ji do proměnné `$delkaNejkratsiSluzby`.

Následuje samotné analyzování nevyužitelných časových oken, popř. přesun termínu, který se je pokusí eliminovat. Algoritmus této činnosti je znázorněn na obrázku 4.2 a dále detailně vysvětlen.

Při ověřování neefektivity v první řadě ověříme ideální případ, kdy je v časovém rozvrhu dostatek prostoru pro zařazení požadované služby, ale žádná volná časová okna již před ani za službou nezbudou. V takovém případě úloha algoritmu končí.

Druhým případem k ověření je, zda se do volných oken za a před zvoleným termínem vejde alespoň jedna služba, tedy minimálně ta nejkratší. Ani v tomto případě nutně k neefektivitě nedochází a algoritmus končí. Pokud se však alespoň z jedné strany nevejde ani nejkratší služba, pokouší se algoritmus takovýto požadavek optimalizovat.

Sečtením volných časových oken před a po službě získá algoritmus informaci o tom, zda přesunem požadovaného termínu lze získat dostatek prostoru pro zařazení další, byť nejkratší služby. V případě, že se další služba již nevejde ani po přesunu, opět úloha algoritmu končí.

Je však potřeba ještě jednou zdůraznit, že to nutně neznamená, že k neefektivitě dojde. Jedná se pouze o výstup z jedné fronty, který bude podle počtu nevyužitelných časových oken ohodnocen. Vybrána bude varianta s nejlepším ohodnocením a pokud ani toto ohodnocení nedosáhne na minimální požadované, jež je individuální a přizpůsobitelné, nebude taková varianta schválena a přistoupí se k návrhu jiných termínů.

Nyní se budu věnovat situaci, kdy se zjistí, že přesunem klientem požadovaného termínu získáme možnost eliminovat nevyužitelná časová okna. V takovém případě se nejprve rozhodne, zda přesunout termín na později, nebo na dříve. Výběr proběhne podle toho, na kterou stranu by byl přesun proveden o méně časových oken a blížil se tak více času požadovaného klientem. Zákazníky však není možné bez jejich souhlasu přesouvat neomezeně. Proto je potřeba navíc zkontrolovat, zda se přesun bez souhlasu klienta potřebný k zamezení vzniku nevyužitelných časových oken vejde do limitu nastaveného poskytovatelem. Pokud ano, navrhne fronta přesun. Pokud ne, ověří se, zda přesunem o maximální povolený počet časových oken dojde k alespoň částečnému zlepšení, tedy že nevyužitelných časových oken bude menší množství, než jaké by bylo, pokud by se vyhovělo času zvoleného klientem. Je-li výsledek pozitivní, navrhne se čas s maximálním možným přesunem. V opačném případě se čas nijak nemodifikuje.

Ať už bylo algoritmem zvoleno jakékoliv řešení, je nakonec ohodnoceno a nabídnuto třídě `AnalyzaPozadavku`, kde bude vybráno nejlépe ohodnocené řešení, nebo budou navrhnuta řešení jiná.

Návrh termínů

Obdobně jako při analýze termínu i zde první krok představuje nalezení časového okna, které odpovídá prvnímu parametru metody, tedy času, od kterého je navrhování termínů zahájeno. Zde je však princip mírně odlišný. Časové okno se vybere první takové ve frontě, které je větší, než počáteční čas. To umožňuje vyhledávat i ve frontách, které jsou časově až za zadaným dnem. Respektuje to však zároveň podmínku, že v zadaný den se vybere časové okno až od dané hodiny. O tento úkon se stará poměrně jednoduchá metoda ve třídě `Fronta`, `pocatecniTerminProNavrh`. Nenalezený termín vyvolá výjimku propagující se obdobně jako v případě analýzy termínu do třídy

`AnalyzaPozadavku`, kde je zachycena a ošetřena. Také zde zjistíme délku požadované služby na základě jejího identifikátoru. Samotné navrhování termínů nyní může začít.

Jednotlivá časová okna jsou procházena a je zkoumáno, zda je okno volné. Pokud ne, pokračuje se dalším oknem. Pokud však okno volné je, spočítá se počet volných za sebou jdoucích oken od aktuální pozice. Tuto operaci provede již dříve zmíněná metoda `volnychTimeslotuPoTerminu` ve třídě `Fronta`. Z této informace a délky požadované služby se zjistí, zda je možné požadovanou službu zařadit. Pakliže ne, přeskočí se časová okna až na konec úseku, do kterého nebylo možné službu zařadit, protože vyžadovala delší čas. Pokud by však službu bylo možné zařadit, zjistí se, zda-li by došlo jejím zařazením ke vzniku nevyužitelných časových oken. Přesněji řečeno, zda-li po zařazení služby nezbudou časová okna, do kterých by se nevešla ani nejkratší služba nabízená frontou. Tuto funkci má na starost metoda `analyzujNeefektivituNavrhu` ve třídě `AnalyzaFronty`. Počet takto nevyužitelných oken je uložen do proměnné `$neefektivitaAktualnihoNavrhu` a je dále použit k porovnávání s dalšími návrhy. Speciální hodnota této proměnné -1 definována konstantou `NEEFEKTIVITA_ZADNA` je vyhrazena pro případ, kdy služba padne do volných časových oken přesně tak, aby po jejích stranách nezbylo volné ani jedno okno. Oproti tomu konstanta s hodnotou 0, `NEEFEKTIVITA_MOZNA`, signlizuje, že počet nevyužitelných oken je 0, ale za službou je stále velký počet volných oken, minimálně takový, aby se do něj vešla aspoň nejkratší služba. V tomto případě je navíc přidán návrh z konce sledu po sobě jdoucích, volných časových oken. Z jednoho sledu volných časových oken tak vzniknou dva návrhy. Jeden standardně na jeho začátku a druhý na jeho konci.

Porovnáním proměnných `$neefektivitaAktualnihoNavrhu` a `$neefektivitaVybranehoNavrhu` se zjistí, zda nový návrh vytvoří více, méně, či stejně nevyužitelných oken. Podle toho se rozhodne, jak bude s novým návrhem naloženo. Je-li hodnota `$neefektivitaAktualnihoNavrhu` menší, stává se tento návrh vybraným, předchozí návrhy jsou zahozeny a hodnota `$neefektivitaVybranéhoNavrhu` je přepsána hodnotou `$neefektivitaAktualnihoNavrhu`. Pokud jsou hodnoty stejné, je nový návrh pouze přidán k vybraným. V případě, že hodnota je větší, je návrh zahozen a další návrhy budou vznikat až od časového okna za úsekem po sobě jdoucích neobsazených oken, do kterého spadá první okno právě zahazovaného návrhu.

Pokud ještě žádný návrh nebyl uskutečněn, je proměnná `$neefektivitaAktualnihoNavrhu` nastavena na maximální hodnotu celého čísla na provozované architektuře. Takové číslo bude vždy mnohem vyšší, než kterákoliv reálná hodnota a jakýkoliv návrh tedy bude mít neefektivitu menší.

Na konci cyklu tak máme jeden či více návrhů z dané fronty, které mají stejný a zároveň z dané fronty nejmenší počet vzniklých nevyužitelných časových oken. To platí samozřejmě pouze za předpokladu, že se ve frontě alespoň jedno volné místo pro požadovanou službu nachází. Vybrané termíny z každé fronty jsou zpracovány ve třídě `AnalyzaPozadavku`. Jsou seřazeny vzestupně podle počtu nevyužitelných časových oken a klientovi je nabídnut požadovaný počet návrhů ze začátku takto seřazených návrhů.

4.2 Ověřování funkčnosti

Ověřování funkčnosti díla probíhalo již od jeho vývoje, testováním očekávaného chování jednotlivých metod jako uzavřených celků, při snaze ověřovat vždy i krajní nebo nestandardní případy a pokrýt tak všechny větve programu. Po dokončení prací na zdrojovém kódu byla následně vytvořena vhodná vzorová data, na nichž bylo ověřeno korektní chování knihovny jako celku.

Poslední fáze ověřování funkčnosti pak byla i níže popsaná simulace, která odhalila několik dalších, doposud skrytých, chyb v nestandardních případech, na které dřívější testy nepamatovaly. V tuto chvíli tak nejsou známa žádná neočekávaná chybová hlášení, ani nekorektní chování.

4.3 Simulace užití

Jako způsob ověření, do jaké míry dokáže knihovna napomoci k lepšímu zaplnění otevírací doby, byla zvolena simulace. Vzniklo tak několik dalších metod a úprav pro její provedení a další vzorová data. Simulace byla prováděna s různými parametry, výsledky ukládány a následně analyzovány. Výsledky jsou dle mého názoru zajímavé. Nejdříve však k implementaci.

4.3.1 Implementace simulace

Ve třídě `AnalyzaPozadavku` vzniklo několik dalších metod a přibyl jeden parametr konstruktoru. Tím je volitelný parametr `$simulace` nabývající hodnot `true` a `false`. Přepíná tak chování třídy do simulačního režimu. Ten se liší především tím, že sbírá statistická data v průběhu obsluhy požadavků, po každém požadavku nenačítá znovu fronty a návrh termínů probíhá pouze v omezené míře, aby se zjistilo, zda návrh neodpovídá zamítnutému požadavku.

Ústřední metodou pro provedení simulace je metoda `simulujPozadavky()`. Jak název napovídá, metoda vytváří pseudonáhodné požadavky na rezervace. Časy jsou však vždy voleny z množiny časových oken takových, aby se pseudonáhodně vygenerovaná služba do daného úseku vešla. Chováním tak metoda simuluje uživatele, který využívá klientskou aplikaci, která nesmyslné požadavky nedovolí odeslat.

Toho bylo docíleno tak, že je nejprve z množiny všech dostupných služeb pseudonáhodně vygenerována služba, poté jsou nalezeny všechna časová okna, do kterých může být rezervace v daný den umístěna. Z těchto oken je pak pseudonáhodně vybráno právě jedno. Generování požadavků končí ve chvíli, kdy nebylo po páté možné vygenerovanou službu nikam zařadit. Fronty jsou tak pro daný den s největší pravděpodobností zaplněny. Po ukončení generování požadavků jsou ze sesbíraných dat vypočteny sledované hodnoty a vráceny v řetězci.

Pro generování pseudonáhodných čísel bylo užito vestavěné funkce PHP, funkce `rand`;

4.3.2 Provedení simulace

O provedení simulace se stará skript `simulace.php`. Ten obsahuje funkci `simulujPozadavky`, která podle zadaných parametrů opakovaně vytváří instance třídy `AnalyzaPozadavku` a volá v nich metodu `simulujPozadavky`. Ve výchozím nastavení toto provede skript 100×. Všechna sesbíraná data nakonec uloží do souboru typu CSV do adresáře výsledky. Používá při tom formát CSV podporovaný MS Excel – buňky jsou od sebe odděleny středníkem. Pojmenování souboru vychází ze zvolených parametrů pro snazší dohledávání. Provedení 100 cyklů simulace netrvá na průměrně výkonném osobním počítači déle, než několik desítek sekund, v závislosti na nastavených parametrech. Obvykle však simulace skončí během několika vteřin.

Simulace byla provedena s 9 různými sadami nastavení, s dvěma nezaplněnými frontami, vždy v průběhu jednoho dne. Otevírací doba obou front byla nastavena od 8:00 do 12:00 a od 13:00 do 16:00. Použitá množina služeb odpovídá službám v přiložených vzorových datech, tedy 10 služeb autoservisu s různou délkou od 30 minut do 4 hodin. Bylo by však možné použít i obecný model. Sady nastavení zahrnovaly následující parametry: délka časového okna, minimální hodnocení termínu pro jeho přijetí a maximální povolený přesun uživatele při rezervaci, oproti jeho původnímu požadavku, bez jeho souhlasu. Sady nastavení byly rozčleněny na 3 části podle využité délky časového okna. V každé části pak byla právě jedna sada nastavení, která optimalizační algoritmus nevyužívala. Sledovány pak pro každé opakování simulace byly procentuální hodnoty využití časových oken, počtu požadavků klienta, kterým bylo vyhověno bez přesunu a počtu požadavků klienta, které byly zamítnuty, tedy ty, kterým byl navržen nový termín.

4.4 Vyhodnocení výsledků simulace

Začněme s nejmenší délkou časového okna (do), 5 minut a vypnutým optimalizačním algoritmem. Zde je zbytečné uvádět procentuální hodnoty časů, kterým bylo vyhověno a které musely být přeplánovány. Jejich hodnoty činily očividně 100% resp. 0%. Hodnoty zaplnění front uvedené v tabulce 4.1 jsou však zajímavé.

průměr	77,2321%
medián	78,5714%
minimum	32,1429%
maximum	92,8571%
zaplnění aspoň 90%	2%
rozptyl	72,2177

Tabulka 4.1: Tabulka zjištěných hodnot zaplnění front pro délku časového okna 5 minut a vypnutý optimalizační algoritmus

Hodnota minimálního zaplnění zde však nedává smysl a je způsobena možnou nepřesností při ukončování generování požadavků.

Po zapnutí optimalizačního algoritmu s minimálním přijímaným ohodnocením termínu (mh) 95 bodů (což je poměrně striktní hodnota) a maximálním umožněným přesunem (mp) 15 minut se situace, jak je vidno z tabulky 4.2, příliš nezlepšila. Výrazně se však zvýšil počet případů, kdy zaplnění stouplo nad 90%. Četnost případů, kdy hodnota v příslušném sloupci dosáhla aspoň 90% zde reprezentuje řádek s popisem „aspoň 90%“. Vzhledem k tomu, že celkový počet simulací byl 100 tak četnost odpovídá i procentuální hodnotě.

	uspokojení času klienta	zaplnění front	přeplánovaných požadavků
průměr	52,5152%	77,7679%	22,3564%
medián	52,9412%	76,7857%	22,2222%
minimum	25,0000%	55,3571%	0,0000%
maximum	91,6667%	96,4286%	52,6316%
aspoň 90%	1%	11%	
rozptyl	177,6007	87,3964	147,9358

Tabulka 4.2: Hodnoty zjištěné pro do = 5 minut, mh = 90 bodů a mp = 15 minut

Další ověřovaný případ dal algoritmu více možností. Snížením minimálního bodového hodnocení a prodloužením maximálního přesunu se hodnoty výrazně změnily. Oproti předchozí sadě nastavení přibylo uspokojení času zvoleného klientem a naopak ubylo přeplánovaných požadavků, dokonce tak, že za celou dobu nebyl přeplánován jediný požadavek. Algoritmus si tak „vystačil“ s přesunováním klientů v rámci umožněného přesunu. Další detaily jsou k nalezení v tabulce 4.3.

	uspokojení času klienta	zaplnění front	přeplánovaných požadavků
průměr	70,1975%	85,5357%	0,0000%
medián	71,4286%	87,5000%	0,0000%
minimum	42,8571%	55,3571%	0,0000%
maximum	94,4444%	98,2143%	0,0000%
aspoň 90%	4%	30%	
rozptyl	125,4236	62,0855	0

Tabulka 4.3: Hodnoty zjištěné pro do = 5 minut, mh = 80 bodů a mp = 30 minut

Takové chování bylo předvídatelné. Nárůst zaplnění front o přibližně 7% v průměru pak znamená, že algoritmus je při vhodném nastavení schopen plnit účel, pro který byl vytvořen.

Následuje srovnání s hodnotami zjištěnými ze simulací s délkou časového okna 10 minut. Nejdříve je uveden případ s vypnutým optimalizačním algoritmem. Procentuální hodnoty zaplnění front jsou uvedeny v tabulce 4.4.

průměr	82,6310%
medián	83,3333%
minimum	67,8571%
maximum	97,6190%
zaplnění aspoň 90%	11%
rozptyl	40,9041

Tabulka 4.4: Tabulka s hodnotami zjištěnými pro do = 10 minut a vypnutým optimalizačním algoritmem

Ze zjištěných hodnot vyplývá, že délka časového okna ovlivňuje zaplnění front výrazným způsobem. Průměrné zaplnění front narostlo oproti případu s délkou časového okna 5 minut při vypnutém optimalizačním algoritmu o více než 5%. I ostatní hodnoty se oproti tomuto případu výrazně zlepšily.

Následuje simulace s délkou časového okna 10 minut a zapnutým optimalizačním algoritmem. Maximální čas přesunu byl nastavena na 20 minut a minimální hodnocení rezervovaného termínu na 80 bodů. Zjištěné hodnoty následují v tabulce 4.5.

	uspokojení času klienta	**zaplnění front**	**přeplánovaných požadavků**
průměr	78,4046%	85,7857%	0,0000%
medián	78,5714%	85,7143%	0,0000%
minimum	53,3333%	67,8571%	0,0000%
maximum	100,0000%	97,6190%	0,0000%
aspoň 90%	15%	28%	
rozptyl	112,9765	45,6582	0

Tabulka 4.5: Tabulka hodnot zjištěných simulací s nastaveními do = 10 minut, mp = 20 minut, mh = 80 bodů

Také zde je možno vidět určité zlepšení v zaplnění front. Přibližně však o pouhá 3 procenta a to i navzdory benevolentně nastaveným parametrům. Další simulace s délkou okna 10 minut měla i přes odlišné parametry velmi obdobné výsledky.

Poslední délka časového okna užitá v simulaci je 15 minut. Hodnoty zaplnění front získané z prvního případu s vypnutými optimalizacemi jsou v tabulce 4.6.

průměr	84,0536%
medián	85,7143%
minimum	62,5000%
maximum	100,0000%
zaplnění aspoň 90%	21%
rozptyl	62,5163

Tabulka 4.6: Hodnoty získané s délkou časového okna 15 minut a vypnutými optimalizacemi

I zde navýšení délky časového okna výrazně prospělo průměrnému zaplnění front. Poprvé se zde podařilo dosáhnout zaplnění front 100%, i když pouze v jednom případě ze sta. Zaplnění aspoň 90% pak v 21 případech.

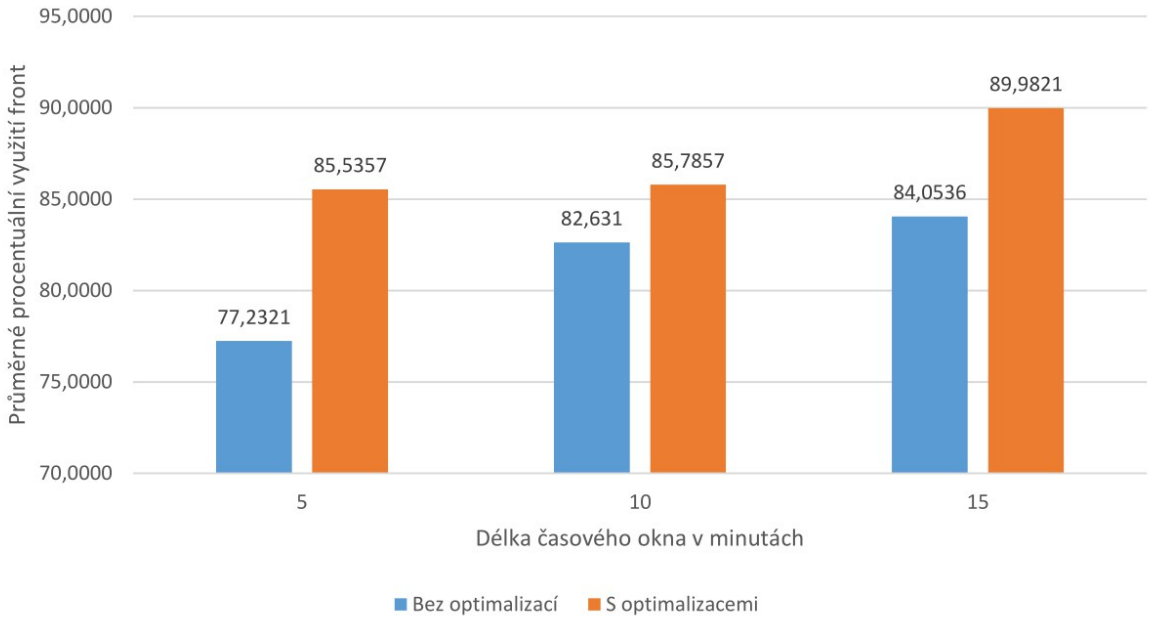
Poslední zde zmíněnou simulací bude případ s délkou časového okna 15 minut a zapnutým optimalizačním algoritmem s hodnotami maximální délky přesunu 15 minut a minimálního hodnocení termínu 85 bodů. Hodnoty jsou k nalezení v tabulce 4.7.

	uspokojení času klienta	zaplnění front	přeplánovaných požadavků
průměr	83,7234%	89,9821%	0,0000%
medián	83,3333%	91,0714%	0,0000%
minimum	56,2500%	58,9286%	0,0000%
maximum	100,0000%	100,0000%	0,0000%
aspoň 90%	30%	59%	
rozptyl	103,1367	56,8173	0

Tabulka 4.7: Hodnoty získané ze simulace s parametry do = 15 minut, mp = 15 minut, mh = 85 bodů

Tento případ nabízí nejlepší kombinaci z naměřených hodnot. Oproti vypnutým optimalizacím se průměrná hodnota zaplnění front s optimalizačním algoritmem zvýšila o téměř 6%. Úplného zaplnění se pak podařilo dosáhnout ve čtyřech případech ze sta. Aspoň 90% zaplnění pak v 59% případů, což je výrazný nárůst oproti simulaci bez optimalizací. Simulace s o něco volněji nastavenými parametry dopadla jen nepatrně hůře.

To, do jaké míry tedy algoritmus ovlivnil průměrné zaplnění front lze vidět na souhrnném grafu 4.1. Z grafu vyplývá, že algoritmus může při vhodně zvolených parametrech příznivě ovlivnit využití otevírací doby. Zde o necelých 13% oproti nejhoršímu řešení bez užití optimalizací.



Graf 4:1: Souhrnný graf znázorňující procentuální zaplnění front před a po užití optimalizací pro různé délky časového okna

Mezi další vlastnosti chování algoritmu, které lze vypozorovat, patří, že s delšími časovými okny stoupá i míra uspokojení času požadovaného klientem. Samozřejmě, že bez optimalizací je míra uspokojení 100%. Počet přeplánovaných pak přímo úměrně souvisí s minimálním bodovým hodnocením termínu. Minimální hodnocení však musí být, dle získaných dat, poměrně striktní, aby k nějakým přeplánováním vůbec došlo. S tím souvisí také závislost minimálního bodového ohodnocení termínu a maximálního umožněného přesunu. Hodnoty obou parametrů by měly být

nastaveny vždy tak, aby se neomezovaly. Bude-li totiž například minimální bodové ohodnocení nabývat příliš vysokých hodnot a maximální povolený přesun bude dostatečně velký, může se stát, že body odečtené za velký navržený přesun způsobí ohodnocení termínu menším počtem bodů, než je minimální přijímaná hodnota a termín tak bude zahozen.

4.5 Budoucí práce a rozšíření

Přestože se zdá být knihovna jako celek funkční a účinná, poskytuje zároveň i prostor pro zlepšení. Různé algoritmy, přestože běh knihovny nepovažuji za pomalý, by mohly být optimalizovány nebo by mohly využívat vyrovnávací paměť pro rychlejší běh. Jednou z klíčových komponent mé práce je vzorec 4.1, který hodnotí řešení z jednotlivých front. Přestože vzorec funguje bez problémů, bylo by zajisté možné jej ještě upravit tak, aby fungoval efektivněji a využíval lépe nižší část hodnot ze škály hodnocení.

Další komponenta, kterou je možné propracovat více do detailu, je využívání pravděpodobnosti, s jakou se vyskytují jednotlivé služby. V současnosti pracuje systém s její nejjednodušší variantou, kdy je pravděpodobnost s jakou se služba vyskytuje uložena v databázi jako desetinné číslo v intervalu <0;1> a je užita ve vzorci 4.1 k ohodnocení termínu. Má práce by však mohla být rozšířena o aparát ke sledování četností, s jakou jsou jednotlivé služby objednávány. Toto sledování by mohlo probíhat i s přihlédnutím k různým vlivům, které mohou četnost objednávání jednotlivých služeb ovlivnit. Například roční období nebo měsíce, dny v týdnu a státní svátky, předstih, s kterým se služba obvykle objednává apod. Zapracování těchto rozšíření by pravděpodobně vedlo k dalšímu navýšení zaplnění front.

Knihovna aktuálně také nepodporuje služby, které trvají déle, než je souvislý časový úsek, kdy má poskytovatel otevřeno. Pokud tedy kupříkladu nějaká služba autoservisu zabere celý den, nebude kvůli pauze na oběd nikdy zařazena. Takové služby by bylo vhodné explicitně značit a přidat pro ně podporu.

Další rozšíření se nabízí v podobě uživatelského rozhraní pro obsluhu simulací, spojené s automatizovaným vyhodnocováním výsledků. Taková simulace by pak mohla konkrétnímu poskytovateli služeb navrhnout parametry pro nastavení knihovny na míru tak, aby došlo co možná k nejlepšímu využití otevírací doby na základě zadaných služeb, dob jejich trvání a otevírací doby. Simulace by pak mohla proběhnout znovu po nějaké době používání systému se sesbíranými daty o pravděpodobnosti, s jakou se služby vyskytnou a parametry tak ještě poupravit pro ještě lepší zaplnění front.

5 Závěr

Cílem této práce bylo vytvořit algoritmus, který bude sloužit jako základ serverové části aplikace pro obsluhu požadavků virtuální čekárny. Tohoto cíle se podařilo dosáhnout. Po seznámení se s existujícími řešeními, způsoby, jakým se chovají a jaké jsou vlastně potřeby moderních virtuálních čekáren byla vytvořena koncepce žádaného algoritmu a následně byla v jazyce PHP implementována knihovna, která takové požadavky obsluhuje. Umožňuje objednat se na určitý čas, pamatuje na paralelní nebo přednostní obsluhu požadavků, počítá s délkou obsluhy dle konkrétní služby a navíc umožňuje navrhovat termíny pro konkrétní službu tak, aby došlo k efektivnímu využití otevírací doby a zároveň obsahuje algoritmus pro obsluhu požadavků na rezervaci pro efektivnější využití otevírací doby. Implementované řešení bylo demonstrováno a ověřeno pomocí simulace na příkladě autoservisu. Výsledky simulace byly vyhodnoceny a diskutovány spolu s možnostmi dalšího pokračování práce.

Na základě simulace a využití optimalizačních algoritmů bylo teoreticky možné zvýšit využití otevírací doby smyšleného menšího autoservisu v průměru o téměř 13%. Takové zlepšení by mohlo v praxi pro poskytovatele služeb znamenat výrazné navýšení tržeb.

Díky této práci jsem si mnohem lépe uvědomil, jak důležitou roli mohou sehrát optimalizační algoritmy i na místech, kde na ně doposud nejsme zvyklí. Naučil jsem se také odlišně nahlížet na potřebu simulací v relativně malých prostředích. V práci bych rád pokračoval v duchu dříve zmíněných rozšíření a vylepšení, přičemž bych ocenil zájem dalších lidí o vývoj a také brzké rozšíření do nasazeného rezervačního systému.

Literatura

[1] Reservio, s.r.o. *Reservio: Online rezervační systém zdarma.* [online]. © 2012-2016 [cit. 2016-04-26]. Dostupné z WWW: <https://www.reservio.com/cs/>

[2] *Online rezervační systém a kalendář SuperSaaS.* [online]. © 2016 [cit. 2016-04-26]. Dostupné z WWW: <https://www.supersaas.cz>

[3] Xtreeme GmbH. *Online reservation system & booking software Planyo.* [online]. © 2016 [cit. 2016-04-28]. Dostupné z WWW: <http://www.planyo.com>

[4] Booked (formerly phpScheduleIt). *SourceForge.* [online]. 29.1.2016 [cit. 2016-04-28]. Dostupné z WWW: <https://sourceforge.net/projects/phpscheduleit/>

[5] BookingBug Inc. *BookingBug: Online Booking System & Appointment Software.* [online]. © 2016 [cit. 2016-04-28]. Dostupné z WWW: <https://www.bookingbug.co.uk>

[6] Mersite Systems s.r.o. *Reservanto.cz: Online rezervační systém zdarma.* [online]. [2015] [cit. 2016-04-28]. Dostupné z WWW: <http://www.reservanto.cz/>

[7] MECA, Petr a Lukáš Meca. *Online rezervační systém FITSYSTEM.* [online]. © 2016 [cit. 2016-04-28]. Dostupné z WWW: <http://www.fitsystem.cz/>

[8] *Rezervační systém pro sportoviště.* [online]. © 2013 [cit. 2016-04-28]. Dostupné z WWW: <http://http://www.rezervacnik.cz/>

[9] MACDONALD, Matthew, Adam FREEMAN, Mario SZPUSZTA a Jan POKORNÝ. *ASP.NET 4 a C# 2010: tvorba dynamických stránek profesionálně. Kniha 1*. Brno: Zoner Press, 2011, 880 s. : il. ISBN 978-80-7413-131-8.

[10] BURD, Barry. *JSP: Javaserver pages, podrobný průvodce*. Praha: Computer Press, 2003, 381 s. ISBN 80-7226-804-X.

[11] GUTMANS, Andi, Stig Sæther BAKKEN a Derick RETHANS. *Mistrovství v PHP 5*. Vyd. 2. Brno: Computer Press, 2007, 655 s. : il. ; 23 cm. ISBN 978-80-251-1519-0.

[12] KOTLÁŘ, Pavel. *Možnosti optimalizace výkonu LAMP (linux/apache/mysql/php)*. Vysoké učení technické v Brně. Fakulta informačních technologií, 2009, Dostupný z WWW: <https://dspace.vutbr.cz/xmlui/handle/11012/53892>

[13] The PHP Group. *PHP: Hypertext Preprocessor.* [online]. 29.4.2016 [cit. 2016-05-02]. Dostupný z WWW: <www.php.net>

[14] LUDWIG, Jakub. *PHP framework pro tvorbu jednoduchých informačních systémů*. Vysoké učení technické v Brně. Fakulta informačních technologií, 2012.

[15] ZHANG, Minmin, Congxin ZHANG, Qinwen SUN, Quancai CAI, Hua YANG a Yinjuan ZHANG. Questionnaire survey about use of an online appointment booking system in one large tertiary public hospital outpatient service center in China. *BMC Medical Informatics and Decision Making* [online]. London: BioMed Central, 2014, **14**, 49 [cit. 2016-05-02]. DOI: 10.1186/1472-6947-

14-49. Dostupný z WWW:
<http://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/1472-6947-14-49>

[16] KOLÁŘ, Dušan. *Principy programovacích jazyků a OOP: IPP*. Brno: Fakulta informačních technologií, 2008, 99 s.

[17] MAŘÍK, Vladimír, Olga ŠTĚPÁNKOVÁ a J LAŽANSKÝ. *Umělá inteligence I.díl*. Praha, 1993, 264 s. ISBN 80-200-0496-3.

Seznam použitých zkratek

GPL	General Public Licence
LDAP	Lightweight Directory Access Protocol
REST	Representational state transfer
API	Application Programming Interface
SDK	Software development kit
CIL	Common Intermediate Language
CLR	Common Language Runtime
AJAX	Asynchronous JavaScript and XML
MVC	Model-view-controller
IIS	Internet Information Services
JSON	JavaScript Object Notation
XML	Extensible Markup Language
MS	Microsoft

Seznam příloh

Příloha 1 – Obsah CD

- `zpráva.php` — písemná zpráva
- `zpráva.odt` — zdrojový tvar písemné zprávy
- `zdrojoveKody/` — zdrojové texty knihovny
  - `model/` — zdrojové texty knihovny
  - `vysledky/` — adresář pro ukládání výstupů ze simulace
  - `vzorovaData/` — JSON data simulující databázi
  - `AnalyzaPozadavku.php` — třída zastřešující funkce knihovny
  - `simulace.php` — skript pro spouštění simulací
  - `demo.php` — skript s ukázkami užití knihovny
- `docs/` — API dokumentace
- `vysledky/` — výstupy ze simulací zmíněných ve zprávě

Příloha 2 – Užití knihovny

1. K využití funkcí z knihovny je nejprve zapotřebí webového serveru s nainstalovaným interpretem PHP. Připravená řešení pro různé platformy nabízí například server XAMPP.
2. Při instalaci webového serveru je vytvořen adresář, kde budou uloženy zdrojové soubory webu. Tento adresář je dále ovlivněn konfigurací serveru. Sem je potřeba zkopírovat zdrojové soubory z CD, tedy adresář `zdrojoveKody/` včetně podadresářů.
3. Skript `demo.php` obsahuje začleněné zdrojové soubory a vytvořenou instanci třídy knihovny. Dále také demonstruje užití jednotlivých funkcí. Skript je řádně komentovaný a ukázky v něm obsažené jsou samozřejmě modifikovatelné pro další ověření funkčnosti.
4. Skript `simulace.php` obsahuje jednoduchý kód určení ke spouštění simulací s využitím vypracované knihovny s okomentovanými příklady užití. Výsledky ukládá ve formátu CSV do adresáře `vysledky/`. Je nutné nastavit oprávnění systému souborů tak, aby měl skript práva do adresáře zapisovat.
5. Pakliže běží služby webového serveru, je možné spustit skripty z bodu 3 a 4 například otevřením libovolného webového prohlížeče a požadavkem na odpovídající stránku. Například http://localhost/zdrojoveKody/demo.php