Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Elektronický rezervační systém ubytovacího zařízení

Kryštof Truschka

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Katedra informačního inženýrství

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Kryštof Truschka

Informatika

Název práce

Elektronický rezervační systém ubytovacího zařízení

Název anglicky

The Electronic Booking System for the Recreation Center

Cíle práce

Cílem práce je navrhnout a implementovat rezervační systém pobytového a rekreačního střediska Líchovy. Toto středisko spadá pod sdružení Prosaz, zabývající se sociální rehabilitací zdravotně postižených občanů. Systém bude umožňovat online rezervaci pobytu klientů a usnadňovat a zefektivňovat proces rezervace a další s ním spojené úkony. Systém bude realizován jako webová aplikace.

Metodika

V první fázi bude proveden sběr požadavků na rezervační systém na základě rozhovorů s uživateli a pozorováním dosavadních zavedených postupů práce uživatelů. Následovat bude analýza požadavků a návrh sytému. Samotný vývoj bude realizován objektově v jazyce PHP s použitím databázového systému MySQL. Posledním krokem bude nasazení systému na server vybrané organizace.

Doporučený rozsah práce

50 – 60 stran

Klíčová slova

informační systém, webová aplikace, UML, PHP, MySQL

Doporučené zdroje informací

ARLOW Jim, NEUSTADT Ila. UML 2 a unifikovaný proces vývoje aplikací : objektově orientovaná analýza a návrh prakticky. Brno : Computer Press, 2007. str. 567 s. ISBN 978-80-251-1503-9.

BORONCZYK, Timothy. PHP 6, MySQL, Apache : vytváříme webové aplikace. Programmer to programmer. Brno : Computer Press, 2009. str. 816 s. ISBN 978-80-251-2767-4.

BRUCKNER, Tomáš. Tvorba informačních systémů : principy, metodiky, architektury. Management v informační společnosti. Praha : Grada, 2012. str. 357 s. ISBN 978-80-247-4153-6.

LECKY-THOMPSON, Ed, NOWICKI, Steven D. a MYER, Thomas. Professional PHP 6. Wrox programmer to programmer. Indianapolis : Wiley, 2009. str. 703 s. ISBN 978-0-470-39509-7.

SCHMULLER, Joseph. Myslíme v jazyku UML : knihovna programátora.Praha : Grada, 2001. str. 359. ISBN 80-247-0029-8.

Předběžný termín obhajoby

2015/06 (červen)

Vedoucí práce

Ing. Marek Pícka, Ph.D.

Elektronicky schváleno dne 25. 3. 2015

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 25. 03. 2015

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Elektronický rezervační systém ubytovacího zařízení" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 31. 3. 2015 _____

Poděkování

Tímto bych rád poděkoval Ing. Marku Píckovi, Ph.D. za vedení mé diplomové práce a pomoc při jejím vzniku. Dále bych chtěl poděkovat pracovníkům sdružení Prosaz, zejména Antonínu Hudimu a Kamilu Klímovi za ochotnou spolupráci při tvorbě rezervačního systému.

Elektronický rezervační systém ubytovacího zařízení

The Electronic Booking System for the Recreation Center

Souhrn

Práce se zabývá tvorbou webového rezervačního systému pro organizaci Prosaz. Dosud zaměstnanci používali ke správě rezervací jen základních nástrojů kancelářského balíku Microsoft Office. Zákazníci mohli rezervovat pobyt jen telefonicky. Úkolem nového rezervačního systému je umožnit zákazníkům vytvářet online rezervace pobytu. Zaměstnancům pak systém usnadní vyřizování a správu ubytování. Data o každé rezervaci již nebudou ukládána odděleně do souborů Excel, ale do centrální MySQL databáze. Systém bude naprogramován v jazyce PHP. Jelikož se bude jednat o internetovou aplikaci, bude dostupná jak pro zákazníky, tak pro administrátory téměř odkudkoli. V první fázi byla provedena analýza, při níž bylo zjišťováno, jak v současné době funguje proces rezervace a ubytování zákazníků. Následně bylo navrhnuto jak proces upravit a vylepšit s využitím nového rezervačního systému. Systém byl vytvořen na autorově vlastním frameworku, který byl v průběhu práce vylepšen. Programování rezervačního systému probíhalo za spolupráce s jeho budoucími uživateli, kteří ho měli v průběhu vývoje možnost testovat a připomínkovat. V poslední fázi byl systém nahrán na webhostingovou službu sdružení Prosaz, finálně otestován a uveden do provozu.

Klíčová slova: rezervační systém, webová aplikace, PHP, MySQL, OOP, framework, MVC

Summary

This thesis focuses on creating a web booking system for the Prosaz organization. Up to now, the employees have managed bookings using basic tools of Microsoft Office. Customers could book a stay by phone only. The first goal of the new booking system is to enable customers to create bookings online. The other goal is to simplify handling and management of bookings by the employees. The data of each booking will no longer be stored separately in an Excel file but rather in a central MySQL database. The system is programmed in PHP. As it is an internet application, it is accessible for both customers and administrators from almost anywhere. In the first phase, an analysis was performed. In this stage, it was examined how the booking process currently worked. Subsequently, an adaptation and improvement of the process was designed, using a new booking system. The system was created on the author's self-programmed framework, improved in the course of the work. The system was programmed in cooperation with future users who had an opportunity to test and comment the system in its development stage. In the last phase, the system was uploaded to the webhosting service of the Prosaz organization. After the final testing, the system was successfully put into operation.

Keywords: booking system, web application, PHP, MySQL, OOP, framework, MVC

Obsah

Seznam obrázků

Seznam ukázek zdrojového kódu

1. Úvod

Naše století je dobou raketového rozmachu informačních technologií a internetu. Počítače vstupují do všech oblastí života společnosti i jednotlivce. Postupně dochází k digitalizaci knihoven, na internetu je obchodní rejstřík, katastr nemovitostí i rejstřík trestů. Většina obchodníků má své e-shopy, po internetu si můžeme objednat termín schůzky na úřadě, koupit si jízdenku na vlak či dokonce dovolenou v zahraničí.

Na elektronickou komunikaci přecházejí státní instituce, orgány veřejné správy, samozřejmě soukromé firmy a nakonec i neziskové organizace. Komunikace online je výhodná jak pro poskytovatele nějaké služby, tak pro klienta. Poskytovatel získá online od klienta všechny potřebná data a může je snadno převést do svého informačního systému. Klient si zase může objednávku vyplnit kdykoli a nemusí se ohlížet na úřední hodiny poskytovatele služby, protože příslušný objednací formulář je možné vyplnit v kteroukoli hodinu, kterýkoli den.

Jak jsem uvedl, na digitální formy evidence a komunikace přecházejí postupně všechny subjekty, včetně neziskových organizací. Ty zpravidla nemívají příliš velké prostředky na nákup drahých informačních systémů, proto se jim vyplatí pořídit si menší aplikaci, ušitou na míru, např. v rámci práce studentů v oboru informačních technologií.

Cílem této práce je navrhnout a implementovat rezervační systém pobytového rekreačního střediska Líchovy. Toto středisko spadá pod občanské sdružení Prosaz, zabývající se především sociální rehabilitací zdravotně postižených občanů. Systém bude umožňovat online rezervaci pobytu a usnadní a zefektivní proces rezervace a další s ním spojené úkony. Systém bude realizován jako webová aplikace.

V první fázi bude proveden sběr požadavků na rezervační systém na základě rozhovorů s uživateli a pozorováním dosavadních zavedených postupů práce uživatelů. Následovat bude analýza požadavků a návrh systému. Samotný vývoj bude realizován

objektově v jazyce PHP[1] s použitím databázového systému MySQL. Posledním krokem bude nasazení systému na pracovišti vybrané organizace a jeho zhodnocení.

Výběr tématu

K rozhodnutí vytvořit rezervační systém pro středisko Líchovy dospěl autor práce po absolvování předmětu *Internetové technologie*. V rámci vypracování zápočtových projektů byli studenti rozděleni do skupin a měli vytvořit webové aplikace pro občanské sdružení Prosaz podle zadaných témat. Autor práce byl členem týmu, který měl vytvořit právě rezervační systém. Rezervační systém byl sice naprogramován, ale nikdy nebyl nasazen do provozu. Hlavním důvodem byla neochota většiny členů skupiny na projektu spolupracovat. Systém neobsahoval celou řadu funkcí, které by byly v reálném prostředí potřeba. Kvůli špatné komunikaci s uživateli systému nebyly správně pochopeny a zapracovány všechny jejich požadavky. Při tvorbě nebyla dodržována jednotná hierarchie umístění souborů, pojmenování tříd, proměnných a podobně. Systém nevyužíval šablony, proto se kód PHP mísil s HTML[2], což způsobovalo značnou nepřehlednost a komplikace při provádění změn. Další nevýhodou byl nedokonalý návrh databáze, který by při delším používání způsoboval neadekvátní hromadění dat. Proto se autor práce rozhodl navrhnout a naprogramovat systém, který by mohl být ve středisku reálně aplikován.

Aplikace bude naprogramována v jazyku PHP s využití databázového systému MySQL. Hlavním důvodem jsou zkušenosti autora s tvorbou webových stránek a aplikací (zejména e-shopů) na těchto platformách. Dalším důvodem je široká podpora těchto technologií napříč webhostingy.

Proč právě PHP a MySQL?

PHP je multiplatformní, to znamená, že ho lze provozovat na většině webových serverů a většině existujících operačních systémů. PHP je zdarma a může být provozováno na linuxových serverech, což šetří náklady za hosting. PHP je dále aktivně rozvíjeno a stále vycházejí nové verze. Má velkou uživatelskou základnu, propracovanou dokumentaci

1 PHP (PHP: Hypertext Preprocessor) - skriptovací programovací jazyk, který je určený především pro programování internetových stránek a webových aplikací.
2 HTML (HyperText Markup Language) – nejběžnější značkovací jazyk používaný pro tvorbu webových stránek.

a ochotně spolupracující komunitu. Pro PHP existuje velké množství frameworků[3], doplňků a tříd, které byly vytvořeny uživateli a mohou být dále použity ve vytvářených projektech.

MySQL je relační databázový systém dostupný pod licencí GPL[4]. MySQL je hojně rozšířen a v základu dostupný na většině hostingových služeb, včetně přístupu do databáze přes webové rozhraní.

PHP se serverovým programem Apache a databází MySQL tvoří dohromady tzv. triádu, trojici programů nejčastěji instalovanou k na webhostingových serverech pro databázové aplikace.

[3] Framework (aplikační rámec) je software, která slouží k podpoře vývoje dalších aplikací. Může obsahovat podpůrné programy, knihovny, podporu pro návrhové vzory nebo doporučené postupy při vývoji.
[4] GNU General Public License, GNU GPL - je licence pro svobodný software, která vyžaduje, aby byla odvozená díla dostupná pod toutéž licencí.

2. Cíl práce a metodika

Cílem práce je vytvořit rezervační systém pro ubytovací zařízení v podobě webové aplikace a provést jeho zavedení do provozu. Hlavní motivy pro zavedení rezervačního systému jsou dva: umožnění online rezervací ubytování zákazníkům a usnadnění a zefektivnění práce zaměstnanců. V teoretické části budou zejména popsány některé pohledy na tvorbu softwaru a architektura MVC, na které bude systém vytvořen. Samotná tvorba rezervačního systému bude rozdělena do několika následujících kroků:

Analýza

Jedním z cílů analýzy bude pochopit cíle organizace a požadavky uživatelů. Uživatelé vysvětlí svoji úlohu a dosavadní způsob práce. Společnými silami pak autor ve spolupráci s uživateli navrhne, jakým způsobem jejich práci pomocí informačního systému zefektivnit. Pro účely analýzy budou použity vytvořeny některé UML[5] diagramy. Budou identifikovány doménové třídy a jejich statický vztah bude zobrazen v analytickém diagramu tříd. Funkčnost z pohledu uživatelů bude zobrazena pomocí diagramu případů užití. Dále bude posouzeno, jestli je vhodné použít klasický přístup k vývoji softwaru nebo přístup podle agilních principů, které se v poslední době začínají na poli webových aplikací stále častěji používat.

Návrh

V této fázi bude vybrán framework či architektura, která bude brát ohledy na jednoduchou rozšiřitelnost funkcionality ve fázích vývoje i do budoucna. Dále bude na základě třídního modelu navržena struktura uložení dat v MySQL databázi. Vybrána bude také HTML šablona, která bude použita při vytváření grafického rozhraní aplikace. Při výběru bude kladen důraz na jednoduchost jejího použití, moderní vzhled, podporu různých platforem.

[5] UML (Unified Modeling Language) - je v softwarovém inženýrství jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů.

Implementace

V této fázi začnou práce na programování. Celá aplikace bude naprogramována objektově v jazyce PHP s využitím HTML 5, CSS[6] 3 a JavaScriptu[7]. Nejdříve dojde k úpravám frameworku, následně bude zahájeno programování modulů. Protože při vývoji bude použit agilní přístup, bude dbán důraz na komunikaci s uživateli. Postupně jim budou předváděny návrhy grafického rozhraní ve formě wireframů[8] a statických výstupních obrazovek v HTML. Dále budou vznikat funkční prototypy, které budou testovány v rámci vývoje i samotnými uživateli.

Probíhat budou pravidelné schůzky, na kterých bude zhodnocen dosavadní postup a uživatelé budou moci ohodnotit a připomínkovat nově vzniklé moduly aplikace. Pokud vzniknou nové požadavky, bude společně diskutováno jejich zavedení do systému. Po dokončení a schválení zadavatelem bude aplikace přenesena na produkční server, kde bude provedeno finální testování. Pokud se během testování najdou chyby, budou okamžitě opraveny. Po otestování budou z databáze vymazána testovací data a aplikace bude nasazena do ostrého provozu.

[6] CSS (Cascading Style Sheets) -kaskádové styly jsou jazykem pro popis zobrazení a formátování elementů v dokumentu napsaném ve značkovacím jazyce.
[7] JavaScript - scriptovací jazyk používaný zejména na webu. Skripty se obvykle spouští na straně klienta a umožňují dynamické chování internetové stránky.
[8] Wireframe webu (WF), neboli drátěný model je skica webu, která se běžně používá k rozvržení základní struktury webové stránky.

3. Teoretická východiska

Rezervační systém bude naprogramován v jazyce PHP. Na úvod bude stručně shrnuta historie vzniku tohoto jazyka. Dále budou krátce popsány některé aspekty nastavování serveru Apache a architektura MVC, na které bude systém postaven. Před vlastním vývojem aplikace se musí stanovit postup, jakým bude software vyvíjen. Nejdříve bude popsáno, jak je možné nasbírat požadavky na nově vznikající systém. Následně budou opsány některé principy agilního způsobu vývoje a také vývojový rámec scrum, který je aplikuje.

3.1. PHP

Počátky PHP sahají do roku 1994, kdy Rasmus Lerdorf naprogramoval jednoduchý systém pro počítání návštěvnosti jeho webových stránek napsaný v jazyce C. Sadu těchto skriptů pojmenoval *Personal Home Page Tools*. Postupem času systém vylepšoval a nový model dokázal pracovat s databází. V roce 1995 byl uvolněn zdrojový kód jako Open Source.

V polovině roku 1995 systém spojil s dalším svým programem *"FI" Form Interpreter* (neboli interpret formulářů) a tím vznikla verze pojmenovaná PHP/FI. V tomto období se systém začínal rozšiřovat a svůj kód začali přidávat další lidé.

Další verze, která se začala podobat PHP, jak je známe dnes, vyšlo ve verzi 3.0 v roce 1998. Název byl zkrácen na PHP (*Hypertext Preprocessor*). Zároveň PHP nyní začalo fungovat i pod operačním systémem Windows a byla přidána podpora objektově programování.

V roce 2000 vyšla verze 4.0, která začala využívat nové jádro *Zend Engin*. Oproti verzi 3 bylo spolehlivější a rychlejší. Přidána byla další vylepšení jako podpora dalších webových serverů, *HTTP session, output buffering*, lepší zabezpečení a několik nových jazykových konstrukcí.

V roce 2004 byla vydána verze 5.0 snovou verzí *Zend Engine*, vylepšeným objektovým modelem a další řadou vylepšení a rozšíření. Při vývoji verze 6, došlo k problémům při implementaci unicode, kvůli odkladům bylo rozhodnuto, že další verze ponese název PHP 7. [1]

Výhodou PHP je rozsáhlý soubor funkcí v základní knihovně PHP. Podpora množství databázových systémů, multiplatformnost, podpora na hostingových službách, velké množství knihoven a kódu, které je možné využít a také poměrně strmá křivka učení jazyka. PHP je interpretovaný jazyk, což znamená, že až do okamžiku svého spuštění je uchován ve zdrojovém tvaru. Výhodou tohoto přístupu je přenositelnost mezi operačními systémy a snazší úpravy aplikace (stačí upravit zdrojový kód a nemusí se provádět kompilace). Nevýhody oproti kompilovaným jazykům jsou nutnost mít interpret a pomalejší zpracování. [2, s. 16,17]

3.2. Apache

Apache HTTP server je velmi rozšířený multiplatformní webový server založený na principu otevřeného zdrojového kódu. Podporuje velké množství funkcí, které jsou často dostupné jako moduly, které rozšiřují jeho jádro.

Webový server Apache je zpravidla nakonfigurován administrátorem pomocí souboru `httpd.conf`. K tomuto souboru však často nemají přístup administrátoři jednotlivých webů umístěných na serveru. Proto existuje také konfigurační soubor `.htaccess`, který může ve většině případů vytvořit a editovat správce konkrétního webu. Soubor `.htaccess` je textový soubor určený k nastavení webového serveru Apache. Pomocí něj může webmaster například zakázat přístup do určitých adresářů, nastavit přesměrování na určitou stránku podle toho, jaký stavový kód vrací protokol HTTP[9] na klientský požadavek, a mnoho dalších funkcí. Nastavení provedená v souboru `.htaccess` se vztahují na adresář a jeho podadresáře. Webový server proto může obsahovat více těchto souborů, rozmístěných v rámci adresářové struktury podle potřeby. Zvláštností tohoto souboru je jeho název - `.htaccess` neobsahuje příponu a na linuxových/Unixových systémech je označen jako skrytý. [3]

3.3. Architektura MVC

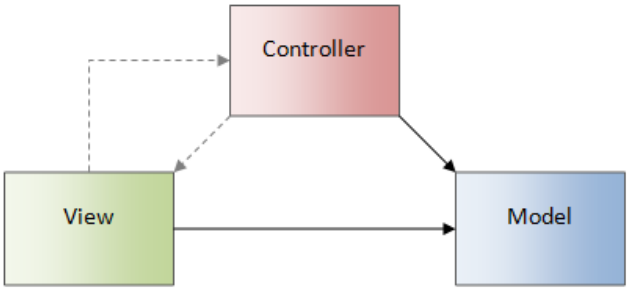
Původní návrh architektury MVC (Model-View-Controller) představil Trygve Reenskaug již v roce 1979. Od té doby se však pohled na MVC a jeho implementaci změnil, i samotný autor přiznal, že si neuvědomoval všechny aspekty a před lety začal pracovat na

[9] HTTP (Hypertext Transfer Protocol) – internetový protokol určený pro výměnu informací v podobě požadavek/odpověď, který používá transportní služby protokolu TCP.

přepracovaném návrhu. [4] Je vcelku zajímavé, jak se pohled na MVC postupem času mění a jak velké jsou rozdíly mezi tím, jak je MVC chápáno různými autory.

MVC je možné vnímat obecně, jako princip mající za cíl přemostit mezeru mezi *lidským mentálním modelem* a *modelem počítačovým*, jehož ideálu zatím nebylo dosaženo. Nebo jako přístup ke tvorbě aplikací. Dále pak jako jednotlivé konkrétní variace MVC. [5] [4]

Základní myšlenkou MVC je: Architektura MVC dělí aplikaci na 3 logické části tak, aby je šlo upravovat samostatně a dopad změn byl na ostatní části co nejmenší. Tyto tři části jsou Model, View (pohled) a Controller (kontroler). Model reprezentuje data a business logiku aplikace, pohled zobrazuje uživatelské rozhraní a kontroler má na starosti tok událostí v aplikaci a obecně aplikační logiku. [5]



Obrázek 1 - Komponenty MVC (obecné) a jejich závislosti. Převzato z [5]

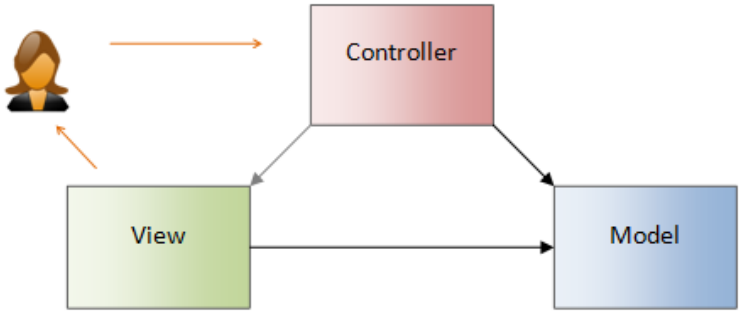
Důležitou vlastností MVC je, že model, kontroler a pohled závisí na modelu, naopak model nezávisí ani na jedné z těchto komponent. V praxi mohou existovat ještě vazby mezi pohledem a kontrolerem, závisí na konkrétní implementaci.

MVC vnímané obecně má dvě hlavní varianty, a to MVC a MVP (Model-View-Presenter). V obou variantách se o vykreslení stará pohled, liší se však v tom, kde jsou zachytávány vstupy. U „widgetových“ systémů[10] umí uživatelský vstup ošetřit komponenty samy – například tlačítko umí reagovat na událost *Click*, textové pole dokáže zachytávat, co uživatel píše na klávesnici a podobně. Protože dnešní klientské technologie jsou prakticky všechny „widgetové“, nemá v nich klasické MVC příliš uplatnění a typicky

[10] „widgetové“ systémy - komponentové frameworky typu Java Swing, Windows Forms, WPF, Silverlight, Flex, ASP.NET Web Forms a podobně

se použije MVP. U „newidgetového“ typu, tyto komponenty neexistují jinde a o zachycení vstupu se stará kontroler. V tomto případě se často používá MVC.

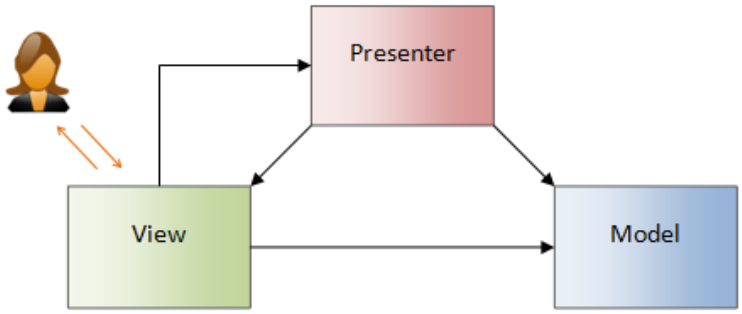
MVC architektura



Obrázek 2 - Vazby komponent a interakce s uživatelem v MVC. Převzato z [6]

Uživatel vykoná v uživatelském rozhraní nějakou akci, kontroler ji zachytí a rozhodne, jak bude zpracována. Většinou změní hodnoty modelu a aktualizuje pohled. Pohled zobrazí výstup uživateli.

MVP architektura



Obrázek 3 - Vazby komponent a interakce s uživatelem v MVP. Převzato z [6]

Největším rozdílem oproti MVC je, že uživatelské vstupy nejsou zachytávány kontrolerem ale pohledem skrze ovládací prvky (tlačítka, vstupní pole apod.) Pohled má typicky přímou vazbu na Presenter. Primární motivací pro oddělení pohledu a presenteru už není nutnost ošetření vstupu, důvody jsou čistě architektonické (prezentační vrstva s MVP se udržuje lépe než monolitický pohled). Vrstva pohledu zpracovává uživatelský vstup. Například při kliknutí na ovládací prvek volá příslušnou metodu presenteru. Stejně jako u MVC by však pohled neměl obsahovat aplikační logiku.

Vzor Supervising Controller

Tento vzor odpovídá původnímu MVP vzoru. Ta zajišťuje mapování mezi pohledem a modelem většinou k tomu používá *data binding* nebo vzor *Observer*. Tento postup dovoluje vytvořit vazby mezi prvky modelu a pohledu. Pokud dojde ke změnám modelu je automaticky aktualizován také pohled.

Vzor Passive View

V této modifikace architektury MVP neexistuje vazba z pohledu na model. To znamená, že presenter neřeší jen odpovědi na uživatelské vstupy, ale také aktualizuje pohled. Cílem je zjednodušit testování. Protože se veškerá logika přesouvá do prezenteru a pohled se maximálně zjednoduší, je možné se soustředit na testování prezenteru.

MVC v prostředí webu

Použití MVC ve webovém prostředí je vcelku přirozenou záležitostí. Protože z principu web neumí sloučit vstup a výstup (zjednodušeně výstup je HTML, vstup URL). Je třeba rozlišovat klasické webové aplikace a ty, které jsou z velké části postaveny na AJAXu[11]. Klasické aplikace generují na straně serveru HTML a na straně klienta se web chová víceméně staticky, naopak u AJAXových aplikací se část prezentační logiky přesouvá na stranu klienta. V dalším textu bude popsána implementace, kde největší část prezentační logiky je na straně serveru. [6]

Využití MVC při tvorbě webové aplikace má řadu výhod. Kód, který je rozdělen do více komponent, se tak stává přehlednější, lépe se testuje a usnadňuje přidávání dalších modulů do aplikace. K jednomu kontroleru může být implementováno více pohledů, to umožňuje vytvořit nad stejným kontrolerem různé výstupy a uživatelská rozhraní, která se liší například podle toho, jestli se jedná o backend nebo frontend aplikace. Stejně tak jeden kontroler může využívat více tříd z modelové vrstvy. Další výhodou tohoto rozdělení je například to, že o tvorbu pohledů se při práci v týmu může starat kodér, který ovládá značkovací jazyk, ale stačí mu jen základní znalosti o programování. Musí však vědět, jaká

[11] AJAX (Asynchronous JavaScript and XML) technologie vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich kompletního znovunačítání za pomoci asynchronního zpracování a JavaScriptu.

data a v jakém formátu jsou příslušným kontrolerem předána do šablony, na které pracuje. Dále budou popsány jednotlivé komponenty s důrazem na použití ve webových aplikacích.

Model

Vrstva modelu obsahuje data a business logikou. Modelová vrstva je nezávislá na kontroleru a pohledu. Poskytuje jen své vstupní a výstupní rozhraní a provádí operace nad daty. „Jeho funkce spočívá v přijetí parametrů zvenku a vydání dat ven. Model neví, odkud data v parametrech přišla a ani jak budou výstupní data zformátována a vypsána.“ [7] Typickým příkladem tříd modelové vrstvy jsou doménové třídy, viz dále. Například vybírání a ukládání dat z databáze a vytváření jakýchkoli dotazů by se mělo objevit jen ve vrstvě modelu.

Pohled

Pohled je část kódu, který se stará o generování výstupu, přičemž nezáleží, jestli výstupem bude HTML, XML[12] nebo PDF[13]. Například, když bude výstupem HTML, mohou být součástí pohledu šablony obsahující HTML kód, které definují to, jak budou data zobrazena uživateli. Pohled přijímá data od kontroleru a ty umisťuje mezi tagy značkovacího jazyka. Šablona také obsahuje podmínky a cykly, nutné k zobrazení výstupních dat.

Kontroler

Kontroler je mezičlánek mezi modelem a pohledem. Úloha kontroleru spočívá v shromáždění požadavků (většinou pochází od uživatele) a rozhodnutí, co se bude dále vykonávat. Jinými slovy přijímá GET, POST požadavky, relace apod. a podle nich rozhoduje, jaké metody budou volány. Vybrané metody pak zpracují uživatelská data a dále komunikují s modelovou vrstvou, kde jsou provedeny další operace a jejich výsledky jsou vráceny zpět do kontroleru. Ten je vyhodnotí a následně předá do pohledu.

[12] XML (Extensible Markup Language) – rozšiřitelný značkovací jazyk. Jeden z nejpoužívanějších formátů pro sdílení strukturovaných dat.
[13] PDF (Portable Document Format) – formát dokumentů zajišťující zobrazení nezávisle na používaném softwaru a hardwaru.

3.4. Sběr požadavků

Požadavky popisují žádané chování systému a jeho vlastnosti. Kvalitní sběr požadavků od uživatelů má přímý vliv na dodání softwaru s dohodnutou funkcionalitou, v daném termínu a za stanovenou cenu. Aby se výsledný informační systém co nejvíce přiblížil požadavkům uživatele, musí se zjistit, co uživatelé chtějí, co očekávají a hlavně co skutečně potřebují. Aby bylo jasné, co a jak bude do systému nakonec implementováno, je nejdříve nutné posbírat požadavky. Následně je analyzovat a upřesnit.

Přitom musí být nalezena shoda na požadavcích všech zainteresovaných stran, proto je většinou nutné tento proces provádět několikrát a požadavky postupně dolaďovat.

Dělení požadavků

Podnikatelské požadavky udávají, proč zákazník systém chce, cíle, kterých chce prostřednictvím systému dosáhnut. Všechny další funkce i požadavky dále směřují k naplnění podnikatelských požadavků. Tyto požadavky jsou formulovány většinou na úrovni vyššího managementu zákazníka.

Uživatelské požadavky popisují cíle uživatelů a to, co bude uživatel se systémem schopen dělat.

Funkční požadavky říkají, jakou funkcionalitu musí vývojáři do systému implementovat, aby uživatelé mohli splnit své úkoly a tím i podnikatelské požadavky.

Systémové požadavky zahrnují požadavky na systém složený z více subsystémů, kombinaci požadavků na software případně hardware.

Parametrické požadavky pochází z prostředí mimo systém, ale přesto ho ovlivňují. Jsou to standardy, vládní nařízení vnitropodnikové předpisy, výpočetní algoritmy apod. Při sbírání funkčních požadavků se musí i na tato pravidla brát zřetel, z nich totiž plynou například některá omezení, která musí být do systému zapracována.

[8, s. 29-31]

Rozdělení uživatelů do tříd

Prvním krokem při sbírání požadavků je identifikace tříd uživatelů. Pro sběr požadavků je potřeba rozdělit je do uživatelských tříd například podle toho, jak často budou systém používat, jakou zkušenost mají s aplikační doménou a výpočetní technikou, jaké funkce budou používat a jaká budou mít oprávnění. Přitom jedinec může spadat do více uživatelských tříd. Požadavky uživatelů z jednotlivých tříd se liší podle toho, jak a k čemu budou systém využívat. Je také možné vybrat přednostní třídy, na jejichž požadavcích závisí podnikatelské cíle projektu.

Z každé třídy je vhodné vybrat zástupce, se kterým bude analytik spolupracovat na vytvoření požadavků i během dalšího vývoje informačního systému. Tento zástupce by měl být skutečným uživatelem s dobrou znalostí aplikační domény a měl by být schopen mluvit za celou uživatelskou třídu, komunikovat s ostatními uživateli této třídy, řešit případné rozpory a předávat informace analytikovi.

Tito zástupci by měli chápat vizi celého projektu a měli by být přesvědčeni o jeho užitečnosti a významu pro ulehčení práce své i svých spolupracovníků. Zároveň by měli mít důvěru vedení. Vedení by naopak mělo brát v potaz, že právě oni znají své potřeby nejlépe a nemělo by zbytečně zasahovat do jejich pravomocí. [9, s. 99]

Následně získáme od zástupců tříd jejich požadavky, například při rozhovorech. Z nich bychom měli být schopni pochopit jednotlivé uživatelské úkoly a podnikové cíle. Poté můžeme pokračovat v analýze informací od uživatelů a odlišení jejich cílů od funkčních požadavků, parametrických požadavků, podnikatelských požadavků, navrhovaných řešení a nadbytečných informací. U jednotlivých požadavků bychom měli také zjišťovat jejich důležitost a pořadí, ve kterém mají být implementovány. Všechny požadavky by měly být zdokumentovány v podobě psané specifikace, případně doplněné vhodnými modely. Na závěr bychom měli ověřit, že zdokumentovaným požadavkům rozumí všichni stejně. Během průběhu projektu mohou požadavky přibývat a měnit se. Řešitel má za úkol najít shodu na požadavcích se zákazníkem a tuto shodu v průběhu projektu udržovat. [8, s. 33-34]

Zdroje požadavků

Existují různé způsoby, jak nasbírat požadavky na výsledný systém, jejich volba závisí na typu a rozsahu realizovaného systému. Zde budou stručně popsány jen způsoby využité pro sběru požadavků při vytváření tohoto projektu.

Nejdůležitějším zdrojem jsou rozhovory a diskuze s budoucími uživateli. Z rozhovoru by analytik měl odhalit skutečné potřeby uživatele. Uživatel totiž často přesně neví, co vlastně chce. Uživatel popisuje analytikovi postup své práce a dalších uživatelů, kteří budou systém používat. Při rozhovoru analytik zjistí, jaké jsou nedostatky v dosavadním postupu, co uživatele nejvíce zdržuje, jaké věci mu nejvíce vadí, co očekává od nového systému. Vyjasnit by se měly také výjimečné stavy, které mohou při procesu nastat. Analytik může zjistit, že některé funkce jsou neefektivní a může navrhnout zlepšení. Po zdokumentování jsou nasbírané požadavky znovu předloženy uživateli a ten zkontroluje, jestli byly opravdu analytikem správně pochopeny, nedostatky jsou případně opraveny.

Dalším cenným zdrojem může být odzkoušení a dokumentace stávajících nebo konkurenčních systémů. Mnohdy je pro analytika užitečné pozorovat uživatele při práci, kde si ověří poznatky získané při rozhovorech. Má poté lepší přehled o tom, co přesně uživatel při práci potřebuje, jaký je zavedený pracovní postup. Při tom může odhalit další požadavky, které uživatel při rozhovorech nezmínil, najít nedostatky v zavedených postupech a systémech a poté navrhnout, jak toto zlepšit v novém systému. [8, s. 98]

Pohled na sběr požadavků

Klasické metodiky vycházejí z předpokladu, že budování informačních systémů lze popsat, plánovat a měřit. Klasický přístup k vývoji softwaru je založen na tom, že požadavky tedy i budoucí funkcionalita jsou popsány v počáteční fázi projektu a později jsou víceméně neměnné. Zatímco zdroje (náklady) a čas se mohou v průběhu projektu kvůli jejich nepřesnému odhadu měnit. [10, s. 120] Tento pohled na vývoj softwaru klade důraz na kompletní sesbírání požadavků, po kterém následuje analýza a vývoj. Při tom může nastat problém, kdy se požadavky v průběhu vytváření softwaru změní. Například dojde ke změnám v okolním prostředí, které vytvoří nová očekávání od vyvíjeného softwaru. Nebo se zákazníkovi některé části aplikace nelíbí, protože si je představoval jinak, i když jeho formálně zadané požadavky splňují. [10, s. 50]

Agilní přístupy na rozdíl od klasických počítají s tím, že se požadavky na funkcionalitu v průběhu projektu mění a dolaďují. Důraz je kladen především na implementaci. To však neznamená, že analýza může být zanedbána, je jen těsněji spojena s procesem implementace. Vývoj je rozdělen na kratší iterace, které zahrnují jak analýzu problému, tak implementaci řešení. To zpravidla probíhá v týmu, kde je kladen důraz na spolupráci a komunikaci. Práce jsou průběžně konzultovány se zástupcem zákazníka, který je buď součástí týmu, nebo s ním úzce spolupracuje. [10, s. 50-51]

3.5. Agilní metodiky

V současné době se na poli vývoje, zejména webových aplikací, začínají prosazovat agilní přístupy k jejich tvorbě. „Umožnují vytvořit řešení velmi rychle a pružně jej přizpůsobovat měnícím se požadavkům. Tyto metodiky prosazují myšlenku, že jedinou cestou, jak prověřit správnost navrženého systému, je vyvinout jej (nebo jeho část) co nejrychleji, předložit zákazníkovi a na základě zpětné vazby jej upravit.“ [9, s. 43] Jinými slovy cílem agilní metodiky je: „co nejdříve, co nejlevněji dodat zákazníkovi produkt, který splňuje všechny požadavky, je kvalitní a umožňuje provádět i následnou údržbu a podporu“ [10, s. 13] Agilní metodiky jsou proto vhodné pro projekty, kde je jejich zadání nejasné nebo se často mění.

Agilní metodiky nejsou tak náročné na některé formální a byrokratické procesy v průběhu projektu a vycházejí z toho, že nejdůležitějším nositelem informace je zdrojový kód. [10, s. 120]

3.5.1. Hlavní principy agilních metodik

V roce 2001 vznikl manifest [11], který popisuje zásady agilních metodik. Jako hlavní rozdíl oproti dřívějším postupům by se dalo vyzdvihnout, že dává přednost:

- **Jednotlivci a interakci** před procesy a nástroji
- **Fungujícímu softwaru** před vyčerpávající dokumentací
- **Spolupráci se zákazníkem** před vyjednáváním o smlouvě
- **Reagování na změny** před dodržováním plánu

Tyto body je možné podrobněji rozepsat do dvanácti zásadních myšlenek agilních metodik:

Užitná hodnota pro zákazníka

Hlavním cílem je uspokojovat potřeby zákazníka, a to průběžným dodáváním softwaru, který pro něj bude mít užitnou hodnotu. Tuto hodnotu tvoří především fungující software a perfektně vytvořené modely a dokumentace.

Změny jsou vítané

Změny v požadavcích jsou očekávány i v pozdních fázích vývoje, to dává možnost přizpůsobit se měnícím se požadavkům zadavatele. Zároveň je implementováno skutečně jen to, co je aktuálně potřeba. Tak se snižuje riziko, že by se programovalo něco, co se bude ještě v průběhu vývoje měnit.

Časté dodávky

Vývoj se provádí iterativně, kdy se na konci iterace dodá zadavateli fungující část softwaru. Agilní metodiky zkracují dobu jednotlivých iterací.

Spolupráce se zákazníky

Řešitel (resp. řešitelský tým) úzce spolupracuje se zákazníkem, zákazníci se v podstatě stávají členy týmu a přejímají na sebe část zodpovědnosti za úspěšnost projektu. Každodenní komunikace umožňuje velmi dynamicky upřesňovat, přidávat, případně měnit hrubé požadavky, které byly definovány v počáteční fázi projektu.

Motivace

Aby členové týmu dosahovali optimálních výsledků, musí být správně motivováni, musí věřit v úspěch projektu, mít podporu vedení a dobré pracovní podmínky.

Komunikace

Přímá konverzace je preferovanou formou komunikace mezi členy týmu. Je rychlejší a levnější spolu mluvit než vytvářet obsáhlou dokumentaci. Dokumentace podle agilního přístupu slouží spíše k pochopení problému a nenese hlavní užitnou hodnotu pro zákazníka.

Posuzování úspěchu

Funkční software je hlavním cílem, a proto je také hlavní mírou, podle které můžeme měřit dosavadní pokrok a úspěch projektu.

Udržitelný vývoj

Aby byl projekt úspěšný, měli by být vývojáři i uživatelé schopni udržet stálé tempo. Pracovníci, kteří se na projektu podílejí, nesmí být přetěžováni. Neodpočinutí lidé ztrácejí produktivitu a dělají chyby. Proto by se měla jasně stanovit pracovní doba a práce přesčas by měla být využívána jen ve výjimečných případech.

Perfektní návrh i řešení

Návrh není samostatná fáze před implementací, ale mění se v průběhu projektu, tak jak přicházejí změny. Právě dobrý návrh a řešení umožní rychlé promítnutí změn do již naprogramovaných částí softwaru.

Jednoduchost

Důraz je kladen na jednoduché postupy řešení, ty se totiž mění snadněji než ty složité. Vykonávat by se mělo jen nutné minimum potřebné k dosažení cíle. Implementace jednotlivých požadavků by se měla odkládat až na dobu, kdy je to opravdu nutné, protože požadavky mohou podlehnout změně.

Samoorganizace týmu

Tým s dobrou vnitřní komunikací, důvěrou a rozumnou pracovní zátěží, který není centrálně řízen, ale spolupráce vzniká přirozeně, podporuje kreativitu svých členů.

Efektivita

Tým průběžně hledá odpovědi na otázky, kde je efektivní a kde a jak by mohl svou efektivitu zvýšit a produkovat tak software rychleji a kvalitněji.

[10, s. 121-23] [9, s. 41-43]

Existuje celá řada agilních metodik, nejznámějšími jsou nejspíše scrum a extrémní programování. Vývoj v extrémním programování probíhá v krátkých iteracích, důraz je kladen na testování softwaru a refaktoring kódu. Využívá také párové programování, kdy dva programátoři spolupracují. Jeden píše kód a druhý na něj dohlíží a pomáhá mu. Dalším příkladem může být metodika Feature Driven Development[14], která je založena na iterativním vývoji, který je řízen užitnými vlastnostmi produktu. Dále by mohl být

[14] Feature Driven Development (FDD) - Vývoj řízený vlastnostmi

jmenován například Test Driven Development[15], kde se nejdříve napíší testy, které ověřují, zda kód, který bude zajišťovat funkcionalitu, dělá to co má. Následuje psaní vlastního kódu, jeho testování a refaktoring. Výše uvedené a další metodiky zde nebudou blíže popisovány V následujících odstavcích však bude přiblížen procesní rámec scrum.

3.6. Scrum

Scrum je agilní procesní rámec, který se používá k řízení vývoje složitých produktů od začátku devadesátých let. U jeho vzniku stáli Ken Schwaber a Jeff Sutherland, kteří základně nastavili délku iterace na 30 dní a zavedli backlog a další artefakty. [12]

„Scrum není proces pro samotný vývoj produktů, je to spíše procesní rámec, uvnitř kterého lze používat jiné procesy a techniky.“ [13, s. 3]

3.6.1. Scrum tým

Scrum tým se skládá z vlastníka produktu, Scrum mastera a vývojového týmu, je sebeorganizující a multifunkční. Sebeorganizující znamená, že si volí, jak práci odvedou a nikdo je neřídí zvenčí. Multifunkční znamená, že k dokončení úkolů nepotřebují nikoho dalšího z okolí týmu.

Vlastník produktu (Product owner) – Osoba, která má konečné slovo v tom, co se bude implementovat a s jakou prioritou. Stará se o správu produktového backlogu, tak aby byl vždy dostupný pro vývojový tým a jeho položky byly srozumitelné. Důležité je, aby tato osoba měla vizi a znala cíle celého projektu.

Scrum master - Chrání vývojový tým před zbytečným rozptylováním, odstiňuje ho od okolního světa. Vede tým směrem k sebeorganizaci a multifunkčnosti, udržuje jeho výkonnost a podporuje ho, aby vytvářel produkt s vysokou přidanou hodnotou. Scrum master také pomáhá vlastníkovi produktu s udržením optimální formy produktového backlogu a moderuje všechny scrum meetingy (schůzky ve scrumu).

Vývojový tým – samoorganizující tým lidí spolupracujících na vytváření přírůstku produktu při každém sprintu. Týmy jsou multifunkční, to znamená, že disponují

[15] Test Driven Development (TDD) - Vývoj řízený testy

potřebnými znalostmi a dovednostmi k vytvoření přírůstku produktu. Velikost týmu závisí na situaci, někde se uvádí, že počet členů by měl být 3 až 9. [13]

3.6.2. Artefakty scrumu

Produktový backlog

Produktový backlog (produktový katalog) je seznam všeho, co může být v produktu potřeba. Obsahuje seznam všech vlastností, funkcí, požadavků, rozšíření, oprav chyb a změn, které bude produkt v příštích vydáních obsahovat. Ty jsou zaznamenány jako položky backlogu[16] většinou ve formě *user stories*[17]. Na produktovém backlogu je pro všechny zúčastněné strany viditelná práce, která má být vynaložena k dosažení cíle. Je jediným zdrojem požadavků. Není však neměnný, na začátku obsahuje jen známé a dobře pochopené požadavky. V průběhu času se však vyvíjí tak, jak se vyvíjí produkt, technologie i prostředí, ve kterém bude používán. Položky backlogu obsahují popis, prioritu a časový odhad pracnosti.

Backlog sprintu

Backlog sprintu obsahuje položky z produktového backlogu, které budou implementovány v aktuálním sprintu. Vývojový tým řídí svůj postup sledováním zbývající práce. Backlog se v průběhu sprintu mění, jak se upřesňují poznatky o tom, co je potřeba ke splnění cíle sprintu, také se upřesňují odhady pracnosti položek. O to se stará pouze vývojový tým.

3.6.3. Průběh procesu s použitím scrumu

Před první plánovací schůzkou se vytvoří produktový backlog, nemusí být plně detailní, protože se bude v průběhu projektu upřesňovat. Všechny důležité položky by měly mít nastavenu prioritu, aby bylo možné některé zařadit do prvního sprintu. Následně proběhne plánovací schůzka sprintu. Po ní je zahájen první sprint, což je jedna časově omezená iterace, jejímž výsledkem by měla být hotová část produktu. V rámci sprintu se každý den pořádají krátké schůzky vývojového týmu. Na konci sprintu je svolána schůzka, kde se provádí vyhodnocení sprintu a prezentuje se výsledek. Také proběhne retrospektivní schůzka sprintu, kde se tým baví o tom, jak by se mohla vylepšit nebo zefektivnit jeho

[16] Položka produktového backlogu (Product Backlog Item, PBI)
[17] User stories – krátké jednoduché popisy vlastností systému z pohledu osoby, která tuto funkčnost požaduje, obvykle uživatele nebo zákazníka

práce. Následně se naplánuje a proběhne další sprint. Jednotlivé kroky budou podrobněji popsány dále.

Sprint

Během sprintu vývojový tým implementuje přírůstek produktu. Délka trvání sprintu je do jednoho měsíce. Toto omezení snižuje možná rizika způsobená změnou zadání projektu. Také je minimálně jednou za měsíc celý proces kontrolován a přizpůsobován. Na dokončený sprint navazuje ihned další. Během sprintu se nesmí měnit zadání práce. Součástí sprintu je popis toho, co bude vytvořeno, vytváří flexibilní plán, který určuje postup prováděných prací, dále zahrnuje samotnou práci a její výsledek.

Plánovací schůzka

Této schůzky se účastní celý scrum tým a je časově ohraničena, podle délky následujícího sprintu. Během schůzky se řeší, jaký přírůstek bude dodán na konci sprintu. Vlastník produktu vysvětluje cíl sprintu a položky produktového backlogu, které by měly zajistit jeho splnění. Každá položka má prioritu, do následujícího sprintu jsou zařazeny položky s nejvyšší prioritou. O počtu zařazených položek z produktového backlogu do backlogu sprintu rozhoduje vývojový tým, protože jen ten může nejlépe určit, co je možné během sprintu stihnout. K plánovací schůzce mohou být přizváni i další lidé, se kterými je potřeba prokonzultovat určité aspekty produktu. „Na konci plánovací schůzky by měl být vývojový tým schopen vlastníkovi produktu a Scrum masterovi vysvětlit, jakým způsobem hodlá jako sebeorganizující tým dosáhnout cíle sprintu vytvořit očekávaný přírůstek.“ [13, s. 9-10]

Denní schůzky

Tyto schůzky probíhají každý den v rámci sprintu a trvají 15 minut. Účastní se jí vývojový tým a Scrum master, který ji řídí. Slouží k vytvoření plánu na dalších 24 hodin. Během schůzky je rekapitulováno, jaká práce byla vykonána od poslední schůzky a co je potřeba vykonat do další schůzky. Pokud má někdo z členů překážku, která by mohla ohrozit splnění cílů sprintu, tak ji zde přednese. Díky této schůzce se kontroluje, jestli tým spěje k dosažení cílů sprintu a sebeorganizaci. Po této schůzce se tým nebo někteří členové mohou sejít k dalším debatám nebo k úpravám plánu zbývající práce.

Vyhodnocení sprintu

Vyhodnocení probíhá při neformální časově ohraničené schůzce, které se účastní scrum tým i další zúčastnění (např. uživatelé). Probírá se zde výsledek předchozího sprintu a reviduje se produktový backlog. Vlastník produktu shrne, které položky produktového backlogu byly zavedeny a které ne. Vývojový tým diskutuje o tom, co se během sprintu dařilo a jak byly řešeny případné problémy, prezentuje dosažené výsledky a odpovídá na dotazy ohledně nových přírůstků. Vlastník produktu diskutuje o stavu produkt backlogu a odhaduje datum dokončení. Celá skupina navrhuje, co bude řešeno dál a vytváří tak odrazový můstek pro další plánovací schůzku.

Retrospektiva sprintu

Tato schůzka probíhá mezi vyhodnocením jednoho a plánováním dalšího sprintu, čas je omezen na tři hodiny při měsíčním sprintu. Schůzky se účastní scrum tým. Během schůzky se rekapituluje předchozí sprint s ohledem na lidi, jejich vztahy a procesy. Hledají se věci, které fungovaly dobře a ty, které je možné zlepšit. Následně se vytvoří plán, jak zlepšení zavést. „Retrospektiva sprintu přináší týmu metodickou a formalizovanou příležitost k zaměření se na kontrolu a adaptaci“. [13, s. 13]

4. Vlastní práce

4.1. Analýza

4.1.1. Výběr postupu vývoje systému

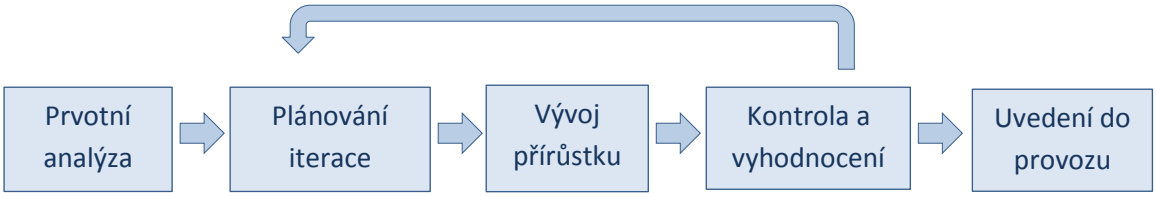
Vývoj softwaru patří mezi oblasti, které se vyvíjejí nejrychleji. Změny probíhají v technologické oblasti, technickém vybavení, vývojových nástrojích, rozšiřující se dostupnosti výpočetní techniky. Roste také konkurence mezi společnostmi zabývajícími se vývojem softwaru. V prostředí webových aplikací jsou tyto změny tím citelnější. Jedním ze základních požadavků bývá zavést webovou aplikaci v co možná nejkratší době. Klasické metodiky vývoje softwaru však kladou velký důraz na detailní analýzu, robustní návrh a široké testování. Tento postup vede intuitivně k předpokladu vytvoření kvalitního softwarového produktu. V dnešní době ale takový přístup nemusí stačit, protože průchod všemi těmito fázemi může být neúměrně časově náročný. Často není cílem vytvořit dokonalou aplikaci, ale aplikaci splňující požadavky, která bude nasazena co nejdříve.

Zákazníci nadále chtějí kvalitní software, ale v kratším termínu. Mnohdy však nemají ucelenou představu o všem, co by měl software umět. Agilní přístup k vývoji klade důraz na velmi těsnou spolupráci mezi zákazníkem a vývojáři. U webových aplikací je také možné dodat základní verzi a funkcionalitu přidávat za provozu. Vývojové týmy se zmenšují, agilní přístup je vhodný i pro týmy, které mají jen několik málo členů.

Někdy není jednoduché použít hned konkrétní agilní metodiku, ať z důvodů zavedených postupů nebo kvůli tomu, že žádná z již hotových metodik zcela nevyhovuje. Není však důvod, proč do procesu vývoje nezačlenit jen fragmenty z agilních metodik, které se zdají být přínosem pro vývoj aplikací. [10, s. 272]

V teoretické části byl popsán procesní rámec Scrum. Z popisu je vidět, že se zaměřuje na práci a komunikaci v týmu. Jelikož systém, který je předmětem této práce, bude vyvíjen jen jednou osobou, nemůže být Scrum použit. Nicméně není důvod, proč nepoužít některé principy, které jsou ve scrumu aplikovány.

Příkladem může být využití krátkých iterací, které podobně jako sprint začínají plánovací schůzkou za účasti vývojáře a uživatelů. Na této schůzce se dohodne, co se bude v následující iteraci implementovat. Na konci iterace se opět zúčastněné osoby sejdou, předvede se demo a zhodnotí se dosažené cíle. V rámci této schůzky se také naplánují další kroky pro následující iteraci. Následující schéma zobrazuje proces vývoje systému.



Obrázek 4 -Proces vývoje systému

4.1.2. Sběr požadavků od uživatelů

Po dohodě s vedením organizace Prosaz byli vybráni dva zaměstnanci, kteří budou na projektu úzce spolupracovat. Jako nejvhodnější způsob získávání informací byly zvoleny osobní konzultace. Před první schůzkou s uživateli si autor prostudoval webové stránky střediska Líchovy.

Pro inspiraci byly také na internetu vyhledány komerční webové rezervační systémy a vyzkoušeny jejich demoverze, pokud byly dostupné. Při této činnosti byl tvořen seznam základních dotazů, které bude potřeba objasnit hned v úvodní fázi.

Cílem první schůzky bylo, aby se autor práce seznámil s tím, jak v současné době probíhá objednání ubytování ze strany zákazníka, jaké úkoly mají zaměstnanci a jaké funkce bude do systému potřeba zařadit.

Zaměstnanci nejdříve popsali vlastními slovy, jak středisko Líchovy funguje a jak probíhají rezervace ubytování. Názorně také ukázali, jak vypadá šablona faktury v Excelu a postup jejího vyplnění. Autor práce si poznamenával důležité části a následně se ptal na věci, které nebyly zmíněny nebo které zcela nepochopil.

V následujícím přehledu jsou zaznamenány otázky a odpovědi na ně:

1. Průběh rezervace

Popis současného stavu zaměstnanci

Rezervace probíhá v současné době tak, že host zavolá na jedno z telefonních čísel uvedených na internetových stránkách. Zaměstnanec, který hovor přijme, musí ověřit, jestli je v období, ve kterém chce být zákazník ubytován, volné místo. Obsazenost ubytovacích prostor se zapisuje do sešitu aplikace MS Excel. Podobně si může host smluvit ubytování elektronickou poštou. Po domluvě vytvoří zaměstnanec fakturu (opět je použita šablona v MS Excel), kterou zašle poštou budoucímu hostu.

1.1. Vytváří ji zaměstnanec jen po telefonické domluvě?

V současné době telefonicky nebo e-mailem.

1.2. Jak to chcete v budoucnu – online formulář (automaticky), telefon, e-mail, na místě (přes zaměstnance)?

Chceme zachovat možnost vytváření rezervace telefonicky a e-mailem. Často se stává, že zákazník při objednávce e-mailem neuvede všechny potřebné údaje, to zbytečně zvětšuje objem komunikace, která probíhá mezi hostem a zaměstnance a prodlužuje dobu vyřízení rezervace. Objednávka je v tomto případě vytvářena zaměstnancem na základě údajů poskytnutých hostem. Na webových stránkách by mělo být uvedeno, jaké informace bude zaměstnanec od hosta vyžadovat, aby mohla být rezervace provedena co nejrychleji. Od rezervačního systému bychom požadovali, aby mohl zákazník sám vytvořit rezervaci přes webové stránky.

1.3. Lze ubytovat bez rezervace?

Zcela ve výjimečných situacích je možné se ubytovat bez rezervace a zaplatit v hotovosti správci zařízení oproti pokladnímu dokladu.

1.4. Vybírá zákazník společně s ubytováním i jídlo? Jaké má možnosti typu stravování?

Hosté vybírají typ stravování při rezervaci, možnosti jsou následující: polopenze, plná penze. Dále se jídlo dělí podle velikosti porce na dětské a dospělé, přičemž cena dětských porcí je v současné době poloviční oproti porcím pro dospělé. Po domluvě je možné nechat si připravit vegetariánské jídlo nebo například jídlo pro

hosty s bezlepkovou dietou. Tyto požadavky však musí znát kuchaři předem, aby měli možnost se na ně připravit. Při vytváření rezervace chceme, aby uživatel mohl zadat počty jednotlivých druhů jídel a aby mohl někam přiložit poznámku, pokud vyžaduje některá dietní jídla apod.

1.5. Jsou ceny za stravu pevné nebo se liší dle jídelníčku?

Cena těchto jídel se neliší od běžných jídel. Jídelníček je vytvářen předem a hosté si nemohou volit, jaké konkrétní jídlo dostanou.

1.6. Je cena ubytování pro děti stejná jako pro dospělé?

Ano, ceny se neliší.

1.7. Je možné přidávat k jednotlivým rezervacím slevy?

Ano, občas k jednotlivým rezervacím přidáváme slevu. Sleva je většinou poskytována zaměstnancům, stálým zákazníkům nebo při sezónních akcích. Její výši stanovuje zaměstnanec. Možnost vytvoření slevy bychom chtěli zachovat i v novém systému.

1.8. Máte otevřeno celý rok? Bude chtít zakázat rezervace v nějakém období?

Středisko Líchovy je otevřeno většinou v období mezi květnem a říjnem. Mimo toto období ubytování není standardně možné. Budeme proto požadovat, aby se na určitá období daly rezervace zakázat.

1.9. V každém pokoji je stejně hostů po celou dobu trvání pobytu, nebo se počet může měnit?

V drtivé většině případů se počet nemění. Pokud ano, je tato situace řešena individuálně. Například poskytnutím slevy. Od rezervačního systému nebudeme požadovat, aby zohledňoval proměnlivý počet osob na pokoji.

1.10. Při každém pobytu je jeho délka stejní pro všechny rezervované pokoje? (Může se stát že, člověk objedná 2 pokoje a jeden bude obsazen do soboty a druhý do pondělí?)

Ano, doba je stejná. Pokud by měla nastat zmíněná situace, musí zákazník vytvořit více rezervací.

1.11. Budou zákazníci automaticky informováni e-maily při změně stavu objednávky?

Ano, možnost automatického zasílání e-mailů bychom uvítali.

1.12. Je potřeba formulář na odeslání individuálního e-mailu zákazníkovi?

Pokud bude potřeba řešit nějaké nejasnosti ohledně rezervace, preferujeme použití vlastního e-mailového klienta nebo věc řešíme telefonicky.

2. Fakturace a platba

2.1. Jak probíhá fakturace?

Faktura je vytvořena v Excelu. Po vytvoření je zaslána zákazníkovi poštou. Po zaplacení celé částky zákazník opět poštou obdrží potvrzení, které předkládá při nástupu do ubytovacího střediska.

2.2. Musí se platit vždy předem?

Ano.

2.3. Jaké jsou platební metody a jaké budete chtít - převodem, hotovostí, kartou ...

Platby jsou prováděny předem převodem na účet. V současnosti se neuvažuje o zavedení jiných platebních metod.

2.4. Částka, kterou zákazník potvrdí při rezervaci, je konečná, nebo se k ní může něco přičíst během pobytu?

Ano, fakturovaná částka je konečná. Další případné služby platí host v hotovosti na místě. Pokud by došlo například k poškození majetku, bude toto také řešeno na místě. Tyto situace ovšem řešíme individuálně a nechceme, aby byly součástí rezervačního systému.

2.5. Jak se platí storno poplatky?

Pokud klient zruší ubytování před zahájením, je povinen uhradit organizaci Prosaz stornopoplatky. Jejich výše je procentuální částka z celkové ceny rezervace a liší se podle doby, která zbývá do zahájení pobytu. Vzhledem k tomu, že celá částka za ubytování je hrazena předem, je zákazníkovi vrácena platba za pobyt snížená o stornopoplatky. Na tuto částku je vystavován dobropis. Ve většině případů jsou ale stornopoplatky promíjeny a je vracena celá částka.

Budeme chtít, aby bylo možné vygenerovat k vystavené faktuře také dobropis. Administrátor přitom zadá, kolik procent z fakturované částky bude dobropisováno.

2.6. Vystavujete fakturu nebo nějaký jiný doklad?

Ano při rezervaci je vystavována faktura a potvrzení o zaplacení pobytu, které host předkládá při nástupu do ubytovacího střediska. Oba tyto dokumenty jsou zasílány zákazníkovi poštou. Dále jsou tisknuty stravenky, které host potřebuje, aby mu bylo vydáno jídlo. Stravenky jsou zasílány spolu s potvrzením o zaplacení pobytu.

Žádoucí změnou by bylo automatizovat proces generování potřebných dokumentů a omezit počet listin, které jsou posílány prostřednictvím pošty. Ve většině případů by se měly tyto dokumenty zasílat elektronicky, což by ušetřilo čas zaměstnance, který se o to musí starat. Zároveň by se zredukovaly náklady na tisk a rozesílání dokumentů. Musí však být brán ohled i na to, že někteří hosté nejsou ochotni nebo schopni používat prostředky elektronické komunikace. Někteří hosté například nemají internet, e-mailovou adresu nebo jsou zvyklí rezervaci vytvářet telefonicky. I na tyto klienty musí být pamatováno a musí být umožněno vytvoření rezervace také pro ně.

2.7. Na tyto doklady máte nějaký systém, používáte papírové formuláře nebo šablony z textového editoru?

Fakturu vytváří zaměstnanec zadáním údajů, které získá od zákazníka vyplněním šablony v podobě sešitu aplikace MS Excel, tato šablona obsahuje vzorce pro výpočet celkové ceny pobytu, zároveň obsahuje seznam některých klientů pro zjednodušené vkládání fakturační adresy apod.

2.8. Budete chtít z nového systému nějaké doklady generovat?

Ano, bylo by vhodné, kdyby systém uměl generovat faktury a dobropisy.

2.9. Kolik stojí ubytování? Mění se ceny nebo vzorec pro výpočet ceny pobytu?

Ceny se liší podle toho, jestli jsou hosté ubytováni v chatě nebo v pokoji. Za chatku se platí fixní částka, za pokoj se platí podle počtu obsazených lůžek. Ke změnám cen nedochází příliš často, ale vyloučit se nedají. Proto budeme chtít, aby se ceny daly měnit z administrace. Je nepravděpodobné, že bychom měnili vzorec pro

výpočet cen pro chaty a pokoje, není proto třeba, aby se jeho změna prováděla z administrace

2.10. Jak je to s DPH?

Organizace Prosaz není plátcem DPH.

2.11. Platí zákazník vždy v korunách? Bude se počítat s nasazením Eura?

Ano, platby jsou prováděny vždy v korunách. Platby v jiných měnách včetně Eura do systému v současnosti zavádět nechceme.

2.12. Platí host nějaké další poplatky za ubytování?

Ano, hosté platí ubytovací a rekreační poplatek. Jejich výše je stanovena zákonem nebo vyhláškou. Výše těchto poplatků se mohou v budoucnu měnit. Proto musí administrace umožňovat jejich změnu. Některé skupiny lidí jsou od těchto poplatků osvobozeny. Při rezervaci musí být proto možné uvést, kolik hostů platit poplatek nebude.

2.13. Do kdy musí host po potvrzení objednávky zaplatit? Když nezaplatí včas, má být objednávka automaticky stornována?

Většinou se počítá s tím, že host zaplatí do čtrnácti dnů od vystavení faktury. Chtěli bychom si však nechat volnost při zhodnocení toho, jestli rezervaci zrušit, či nikoli.

2.14. Je potřeba generovat variabilní symbol pro platbu převodem?

Variabilní symbol je stejný jako číslo faktury. Potřebovali bychom, aby se číslo faktury dalo zadávat ručně a nebylo generováno podle nějakých pravidel.

3. Zákazník

3.1. Jaké potřebujete údaje od zákazníka? Jméno, adresa, e-mail, telefon? Může být zákazníkem i organizace?

Od hosta je vyžadováno: jméno, příjmení, adresa (ulice, město, číslo popisné, PSČ, stát), u organizací pak také: název organizace, IČO, DIČ. Telefon není povinný, ale chceme, aby měl zákazník možnost jeho zadání. Při vytváření rezervace zákazníkem musí být zadána také e-mailová adresa. V případě vytváření rezervace zaměstnancem chceme nechat zadání e-mailové adresy volitelné, protože někteří hosté elektronickou poštu nepoužívají.

3.2. Zaznamenává číslo dokladu totožnosti? Ověřuje se někdy?

Ne, žádný takový údaj neevidujeme. Totožnost hosta může být ověřena při zahájení ubytování, kdy host předkládá potvrzení o zaplacení pobytu. V něm je mimo jiné uvedeno jeho jméno.

3.3. Bude se zákazník muset nutně registrovat (vyplnění hesla pro opětovnou rezervaci), nebo stačí jednorázová rezervace bez další možnosti přihlášení?

Bylo by vhodné, aby fungovaly paralelně obě možnosti, neboli vytvoření rezervace s registrací i bez registrace.

3.4. Může registrovaný zákazník po přihlášení ještě prohlížet případně editovat vytvořenou rezervaci?

Chtěli bychom, aby vytvořenou registraci mohl upravovat již jen administrátor. Pokud zákazník obdrží informace o rezervaci e-mailem, není nutné, aby mohl své rezervace prohlížet.

3.5. Máte hosty ze zahraničí? Evidujete u nich nějaké další údaje?

Ano, ale jen mizivé množství. Pokud přijede zahraniční host, bude zaevidován při příjezdu mimo rezervační systém.

3.6. Musí zákazník souhlasit s nějakými smluvními podmínkami?

Ano, musí potvrdit souhlas.

4. Ubytování

4.1. Co všechno uvádět u pokoje?

Uvádí se číslo pokoje, počet lůžek, cena za lůžko, penále za neobsazené lůžko, typ (chatka, pokoj).

4.2. Existují ještě další možnosti ubytování - přistýlky, stany?

Použití přistýlky je vzácné, pokud by někdo z nějakého důvodu přistýlku vyžadoval, musí kontaktovat zaměstnance, který by ji zahrnul do objednávky jako doplňkovou službu. U stanů by byla podobná situace jako u přistýlek, standardně se stanování nenabízí.

4.3. Jak dlouho dopředu je možné si rezervovat pokoj?

Počítali bychom s rezervací maximálně rok dopředu.

4.4. Stačí v administraci pro účty zaměstnanců jen jeden stupeň oprávnění, nebo bude se systémem pracovat více uživatelů s různými právy?

Rezervační systém bude spravovat více zaměstnanců, ale není potřeba, aby měli odlišná oprávnění. Bude stačit, aby se každý správce mohl přihlásit svým uživatelským jménem a heslem.

4.5. Potřebujete tisknout nějaké sestavy, třeba pro pokojské?

Kromě faktur a podobných dokladů tiskneme při vyřizování rezervace také stravenky.

Stravenky máme předtištěné na barevných papírech, podle toho, jestli jsou určeny pro snídani oběd nebo večeři. Pokud jsou stravenky na poloviční porci, je to na nich také napsáno. Na jeden papír formátu A4 se vejde 28 stravenek. Stravenky jsou po zaplacení pobytu nastříhány a každá je opatřena razítkem s datem, na které je vystavena, poté jsou poštou zaslány zákazníkovi společně s potvrzením o zaplacení pobytu.

4.6. Potřebujete nějaké statistické výstupy?

Jako poskytovatelé ubytování máme povinnost sbírat některá data a předávat je Českému statistickému úřadu. Zejména je potřeba za každý měsíc spočítat, kolik bylo za jednotlivé měsíce ubytováno hostů, počet tzv. *pokojodnů* a počet přenocování. K výpočtům nyní používáme Excel. Automatické generování těchto statistických dat by nám ušetřilo mnoho práce.

5. Systémové a jiné požadavky

5.1. Má zaměstnanec, který vyřizuje objednávky, k dispozici PC s internetem?

Ano, zaměstnanci, kteří spravují objednávky, mají počítače s připojením k internetu. V samotném ubytovacím středisku je však jen základní možnost připojení k internetu prostřednictvím mobilní telefonní sítě.

5.2. Máte vlastní webový server nebo hosting s PHP MySQL?

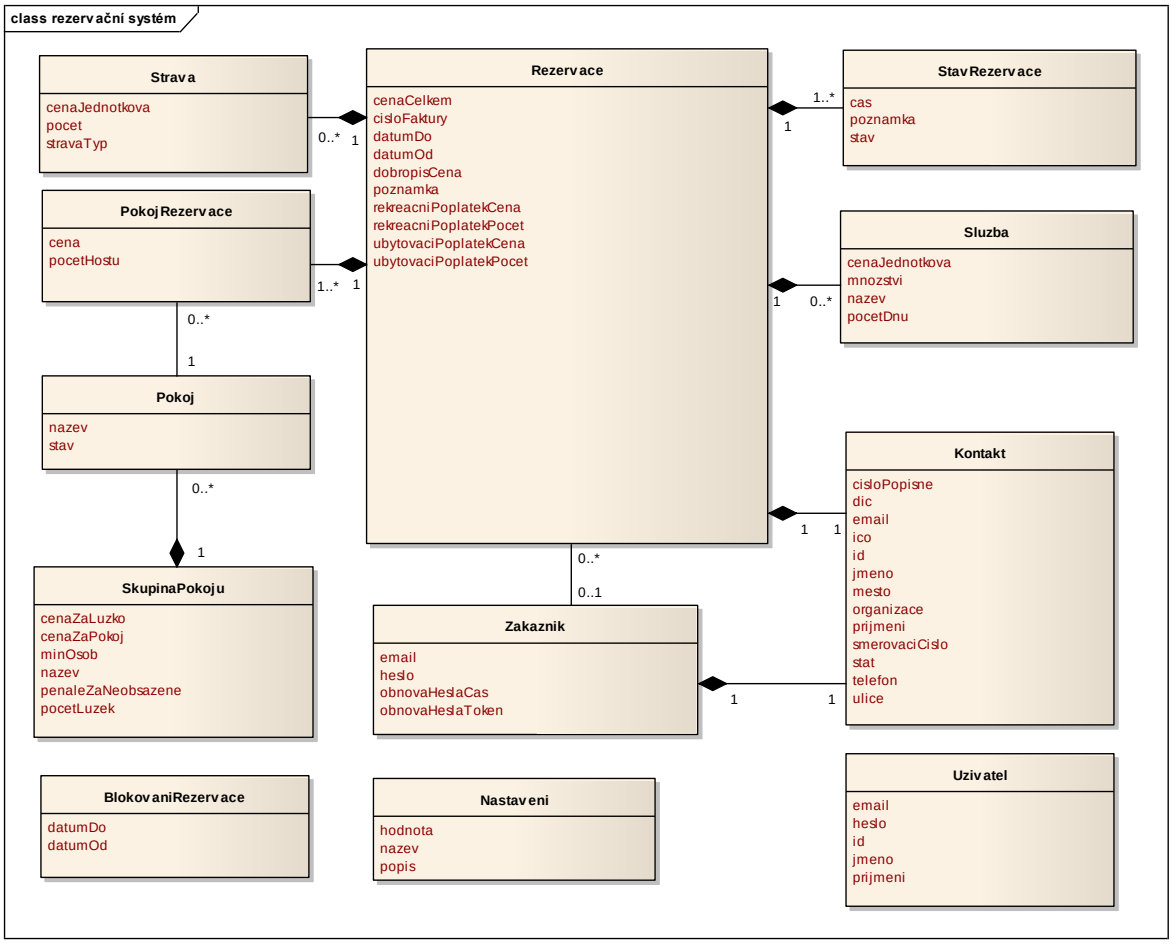
Vlastní server nemáme, ale naše webové stránky jsou hostovány na serveru podporujícím PHP a MySQL.

5.3. Speciální požadavky na přístupnost. Bude požadována podpora mobilních zařízení?

Chceme, aby systém mohli používat i lidé se sníženou pohyblivostí. To znamená, že například menu by se mělo rozbalit po kliknutí a zůstat rozbalené i po přejetí na jiné místo. Dále bychom chtěli, aby se text dal zvětšovat pomocí standardní funkce prohlížeče, bez dopadu na funkčnost stránky. Prohlížení stránky na mobilních zařízeních není nutností, ačkoli bychom tuto funkcionalitu uvítali.

4.1.3. Konceptuální model tříd

Následující diagram zobrazuje identifikované třídy rezervačního systému, vazby mezi nimi a také klíčové atributy.



Obrázek 5- Konceptuální doménový model tříd

Třída Rezervace

Třída `Rezervace` obsahuje základní údaje o rezervaci, jako počátek a konec pobytu, celkovou cena, poplatky, číslo faktury, dobropisovanou částka a poznámku pro administrátora. V každé rezervaci je jeden nebo více pokojů.

Třída PokojRezervace

Obsahuje údaje o tom, kolik bude v jakém pokoji hostů a jaká je cena za rezervovaný pokoj a zprostředkovává vazbu mezi rezervací a pokoji.

Třída Pokoj

Označuje konkrétní pokoj, který má svůj název a stav. Každý pokoj je v právě jedné skupině pokojů.

Třída SkupinaPokoju

Má atributy název, cena za lůžko, cena za pokoj, počet lůžek, penále za neobsazené lůžko a minimálně osob. Skupina pokojů může obsahovat žádný nebo více pokojů.

Třída StavRezervace

Každá rezervace má jeden nebo více stavů. Atributy jsou čas přidání stavu, nepovinná poznámka a typ stavu rezervace.

Třída Strava

Obsahuje vybranou stravu k rezervaci. Její atributy jsou počet porcí, typ stravy, cena za jednu porci. Rezervace může obsahovat žádný nebo několik typů stravy.

Třída Kontakt

Ke každé rezervaci je vázán jeden kontakt. Kontakt obsahuje jméno příjmení, adresu a případně firemní údaje zákazníka. Každý registrovaný zákazník má také právě jeden kontakt.

Třída Uzivatel

Tato třída obsahuje jméno, příjmení administrátora a také jeho přihlašovací údaje v podobě e-mailové adresy a hesla.

Třída Nastaveni

Obsahuje globální nastavení systému. Jejími atributy jsou název nastavení, hodnota a popis.

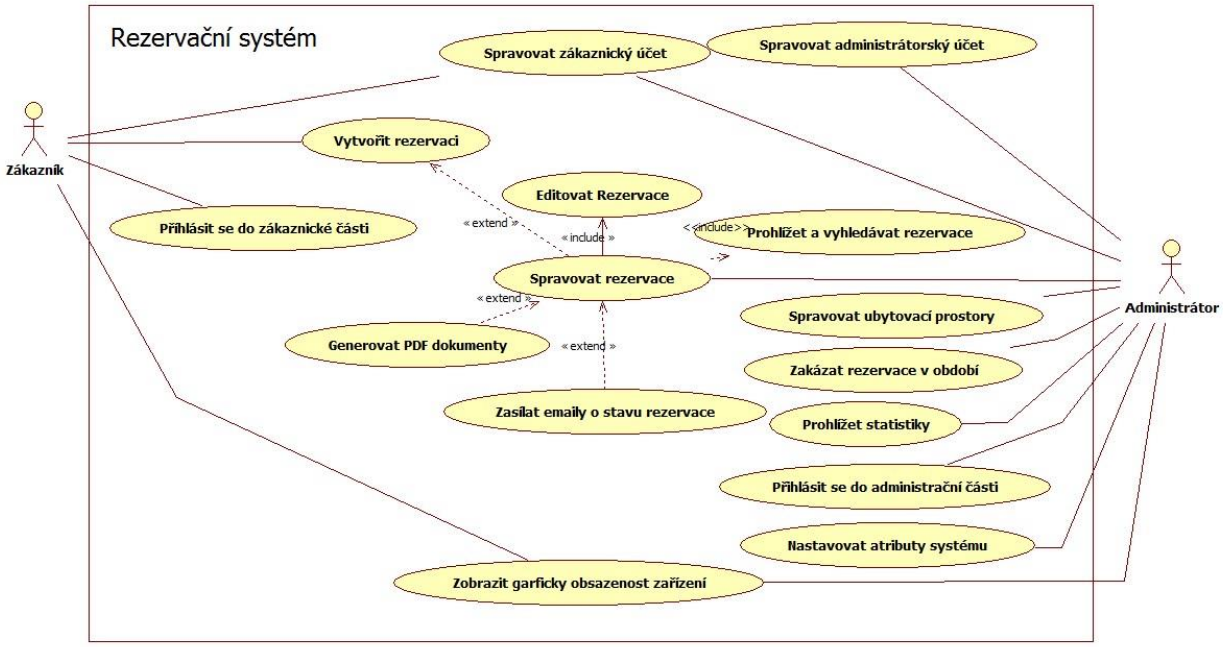
Blokování rezervace

Obsahuje, v jakém rozmezí je zakázáno vytváření rezervací.

4.1.4. Případy užití

Pomocí případů užití lze popsat, jak uživatelé systém používají. Případ užití popisuje část funkcionality systému, kterou využívá uživatel (aktér) a která plní určitý cíl. S jejich použitím uživatel lépe pochopí, co může od systému očekávat, protože popisují systém z jeho pohledu. Lépe se také hledá, jestli byla při sbírání požadavků nalezena všechna funkcionalita, kterou uživatel od systému potřebuje. S využitím případů užití můžeme nasbírat funkční požadavky, tyto požadavky budou vycházet přímo z potřeb uživatelů, nestane se tak, že by byly zahrnuty některé funkce, které by ve výsledku byly nadbytečné a nepoužívané. [8, s. 140]

Diagram případů užití rezervačního systému je uveden níže, následuje stručný popis jednotlivých případů užití. Obdélník, který představuje hranice systému. Z vnějšku systému stojí aktéři, zakreslení jako „panáčci“. Elipsy znázorňují jednotlivé případy užití, tedy to, co aktér od systému potřebuje. Jednotlivé případy užití jsou propojeny s příslušnými aktéry.



Obrázek 6 - Use Case diagram systému

Zákazník

Zákazník – zákazníkem je zde osoba, která má zájem o ubytování.

Spravovat zákaznický účet – zákazník se bude moci zaregistrovat. Registrace bude probíhat zadáním přihlašovacích a kontaktních údajů. Pokud bude zákazník přihlášen, bude mít možnost tyto údaje také editovat.

Vytvořit rezervaci – zákazník bude mít možnost vytvořit rezervaci, to znamená vybrat termín, pokoje, druh stravování, zadat fakturační údaje a odeslat rezervaci ke zpracování administrátory.

Přihlásit se do zákaznické části – přihlásit se pod svým účtem. Při vytváření objednávky budou použity fakturační údaje spojené s jeho účtem.

Zobrazit graficky obsazenost zařízení – tato funkce pomůže při plánování pobytu rychle zjistit, kdy jsou v ubytovacím středisku volné kapacity.

Administrátor

Administrátor – zaměstnanec organizace, který se stará o vyřizování rezervací.

Spravovat Rezervace

Vytvořit rezervaci - vytvářet rezervace podobně jako zákazník, jen s rozšířenými možnostmi.

Editovat Rezervace – upravovat již vytvořené rezervace a měnit jejich stav.

Generovat PDF dokumenty – automaticky vytvářet dokumenty určené pro tisk nebo zaslání zákazníkovi.

Zasílat emaily o změně stavu rezervace – při změně stavu rezervace zasílat zákazníkovi informační e-mail s přiloženými dokumenty.

Prohlížet a vyhledávat rezervace – snadno vyhledávat a filtrovat vytvořené rezervace podle zvolených kritérií.

Spravovat administrátorský účet – měnit přihlašovací údaje vytvářet nové administrátorské účty.

Spravovat ubytovací prostory – přidávat a upravovat pokoje a nastavovat cenu za ubytování.

Zakázat rezervace v období – na vybrané časové úseky zakázat vytváření nových rezervací.

Prohlížet statistiky – zobrazit vybrané statistické údaje za jednotlivá období.

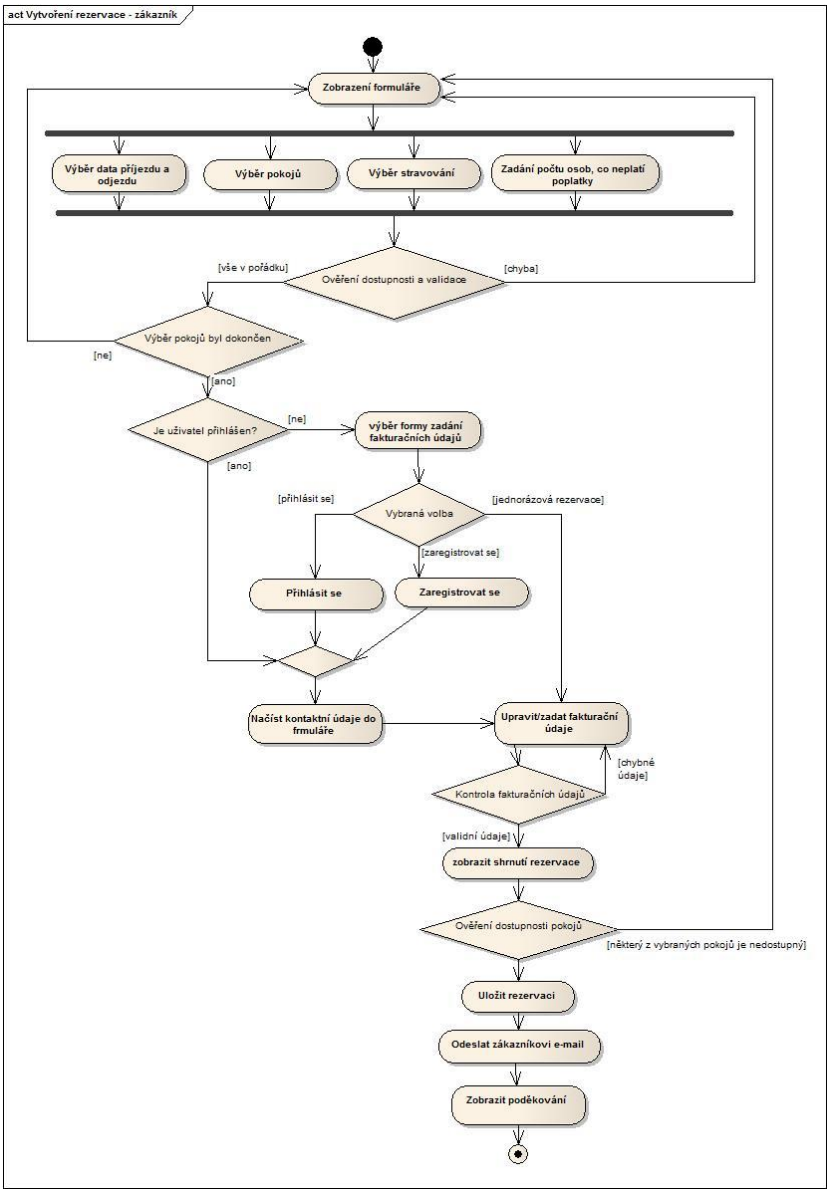
Přihlásit se do administrační části – přihlášení pomocí vytvořeného administrátorského účtu.

Nastavovat atributy systému – upravovat globální nastavení, jako je kontakt na organizaci, ale například i cenu stravování.

Zobrazit graficky obsazenost zařízení – Podobně jako u zákazníka, jen se zobrazením více detailů užitečných jen pro administrátora.

4.1.5. Diagram aktivit

Pro popis dynamických aspektů systému je možné využít diagramy aktivit. Ty se používají při modelování byznys procesů a pracovních postupů, pro popis procedurální logiky, či pro modelování logiky scénářů případů užití. Na následujícím diagramu je jako příklad uveden postup vytvoření nové rezervace zákazníkem.



Obrázek 7 - Diagram aktivit - vytvoření rezervace zákazníkem

4.1.6. Rozpočet projektu

Neziskové organizace mají oproti komerčnímu sektoru mnohem méně prostředků. Z autorovy osobní zkušenosti vyplývá, že pro vedení menší neziskové organizace, v tomto případě občanského sdružení, bývá problém investovat do vybavení, které přímo nesouvisí s posláním organizace, tedy i do ITC. Pro sdružení Prosaz je tedy ideální variantou nechat si vypracovat rezervační systém v rámci této diplomové práce zdarma. Náklady na vytvoření systému tak budou nulové, pokud není počítán čas zaměstnanců, kteří budou na projektu spolupracovat, popřípadě náklady spojené s nastavením webhostingu, které jsou však minimální.

4.2. Návrh

Předmětem projektu je webový rezervační systém. Systém je rozdělen na dvě hlavní části frontendovou a backendovou. Frontend je určen především koncovým zákazníkům – zájemcům o ubytování. Umožňuje prohlížet, jaké jsou volné ubytovací kapacity ve vybraných termínech. Provádí se zde registrace nového uživatele. Umožňuje zákazníkovi vytvářet online rezervace. Backend slouží zaměstnancům, ti mají přehled o rezervacích, které vytvořili zákazníci, dále mohou přidávat nové rezervace a upravovat stávající. Součástí je také seznam registrovaných klientů s jejich kontaktními údaji. Administrátor může vytvářet a spravovat ubytovací prostory, měnit jejich kapacitu a vytvářet pravidla pro výpočet ceny za ubytování. Zaměstnanec může v průběhu procesu rezervace ubytování klientem měnit stav rezervace, když je rezervace v určitém stavu, je zákazník informován e-mailovou zprávou. Součástí je také generování dokumentů spojených s ubytováním, které si mohou zaměstnanci vytisknout nebo je přes systém zaslat zákazníkovy. Do této sekce mají přístup jen registrovaní zaměstnanci. Podrobněji budou jednotlivé funkce a moduly systému popsány v kapitole implementace.

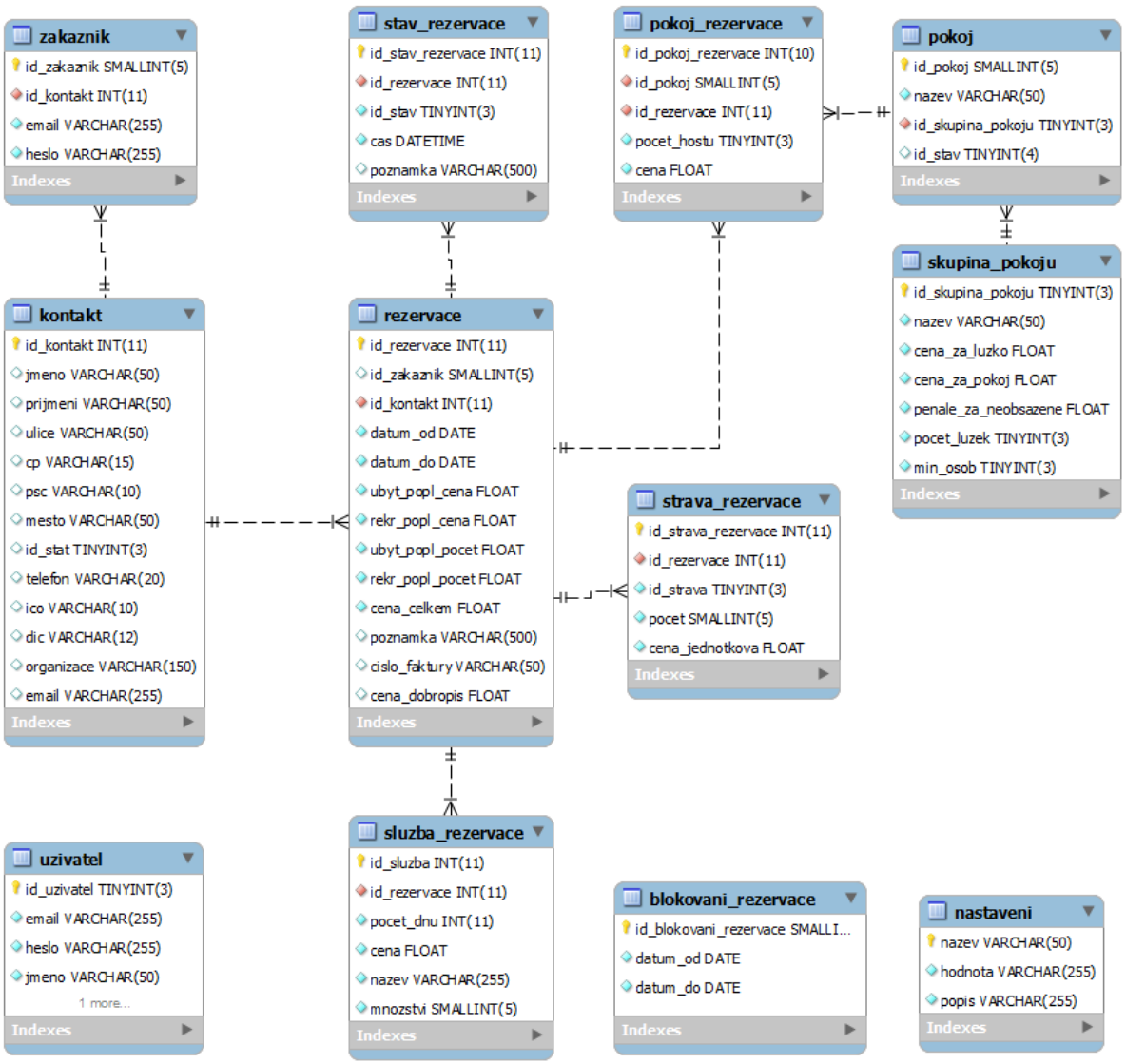
4.2.1. Databáze

Při výběru formátu úložiště bylo rozhodováno mezi MyISAM a InnoDB. MyISAM je sice při dotazech typu SELECT (výběr z databáze) rychlejší, nepodporuje však transakce a referenční integritu. Proto bude na všech tabulkách nastaven formát úložiště na InnoDB.

Transakce udržují databázi v konzistentním stavu. Slučují sadu několika databázových operací do jednoho funkčního celku. Pokud například budou do dvou tabulek vkládány záznamy, které jsou na sobě závislé, je potřeba, aby se uložily buď oba, nebo žádný (v tom případě je zobrazeno chybové hlášení).

Referenční integrita se definuje pomocí cizích klíčů. Podřízená tabulka má nastaven cizí klíč na hodnotu primárního klíče v nadřazené tabulce. Při přidávání nebo změně záznamu v podřízené tabulce je kontrolováno, jestli má cizí klíč hodnotu odpovídající nějakému primárnímu klíči v nadřazené tabulce. Jestliže ne, dojde k chybě. Pokud je záznam v nadřazené tabulce mazán nebo upravován, je ověřováno, jestli v podřízené tabulce není odpovídající záznam. Pokud zde je, může být vyvolána chyba nebo může dojít ke změně cizího klíče v podřízené tabulce, dle nastavení referenční integrity.

Struktura navržené databáze



Obrázek 8 - ER diagram navržení databáze

4.2.2. Framework

Jako základní stavební prvek byl použit framework, který autor práce naprogramoval pro jiný webový projekt. Tento framework je postaven na architektuře MVC a jeho starší verze bezproblémově funguje v reálném provozu. V průběhu práce byl původní framework v některých částech optimalizován, bylo opraveno několik chyb a přidáno mnoho vylepšení, které usnadňují práci při programování samotné aplikace. Diagram tříd frameworku je umístěn v Příloha 1 - Diagram tříd frameworku.

4.2.3. Souborová struktura aplikace

Rezervační systém je rozdělen na dvě části: frontendovou a backendovou. Tomu je přizpůsobena i souborová struktura. Hlavní složka, do které je systém nahrán, obsahuje soubory a složky, které využívá frontend, kromě toho obsahuje složku `admin`. V té jsou nahrány všechny komponenty backendové části ve stejné struktuře jako u frontendu. Navíc obsahuje složky `_kernel`, `_rozsireni` a `_projekt`, které jsou využívány oběma částmi.

Frontendová část

`index.php` – načítá konfigurační sobor, spouští autoloading, spouští hlavní kontroler

`Kernel.class.php` – hlavní kontroler frontendové části

`_conf` – obsahuje konfigurační soubory

`_modul` – obsahuje složky s jednotlivými moduly

- `_class` – obsahuje pomocné třídy modulu
- `_css` – obsahuje soubory s automaticky načítanými kaskádovými styly modulu
- `_js` – obsahuje soubory s automaticky načítanými javaScripty modulu
- `_sablony` – šablony modulu

`_sablony` – šablony, které mohou používat všechny moduly

`_tmp` – dočasné úložiště soborů

- `_smartyKompilace` – složka se zkompilovanými šablonami systému Smarty
- `_smartyCache` – složka s cache šablonovacího systému Smarty

`_css` – globální složka pro uložení css soborů

`_js` – globální složka pro uložení javaScriptových souborů a pluginů[18]

`_grafika` - obrázky a fotografie

`admin` – složka backendové části

Backendová část

[18] Plugin je software, který nepracuje samostatně, ale jako doplňkový modul jiné aplikace a rozšiřuje její funkčnost.

Tato část má stejnou strukturu jako frontendová, navíc přibyly následující součásti:

`_kernel` – obsahuje třídy potřebné pro chod frameworku

`_rozsireni` – v podsložkách obsahuje rozšíření třetích stran zpracovávané na straně serveru

- `dompdf` – konvertor HTML do PDF
- `email` – PHPMailer – knihovna pro odesílání e-mailů z PHP [15]
- `smarty` – šablonovací systém

`_faktury` – složka s vygenerovanými PDF dokumenty

`_projekt` – obsahuje třídy rezervačního systému sdílené napříč moduly

Modulárnost

Aplikace je rozdělena na menší logické celky (moduly). Modul je relativně nezávislý segment aplikace zajišťující určitou část funkcionality. Například modul „Uživatel“ zprostředkovává vytváření, mazání a další úkoly spojené se správou uživatelských účtů. Každý modul má vlastní složku. Ta obsahuje jeho kontroler, také může obsahovat další soubory, které se k němu vztahují. V podsložce `_class` jsou pomocné a doménové třídy, které jsou automaticky zaváděny autoloaderem po jejich zavolání. Ve složce `_js` mohou být javaScriptové soubory a v podsložce `_css` soubory kaskádových stylů. Tyto soubory jsou při zpracování šablony automaticky přidány do její hlavičky. Ve složce `_sablony` jsou umístěné šablony templatovacího systému Smarty pro tento modul. Rozdělení na moduly aplikaci zpřehledňuje a také usnadňuje přidání nové funkcionality.

4.2.4. Vzhled a chování

Standardní aplikace jsou většinou vyvíjeny na konkrétní platformu, případně je vytvořeno více verzí. U webových aplikací by nemělo záležet na jakém operačním systému nebo prohlížeči webovou aplikaci uživatel použije. Pravdou bohužel je, že rozdíly mezi prohlížeči stále existují. Ne všechny prohlížeče výslednou stránku zobrazí stejně. Může se jednat o různou interpretaci a podporu kaskádových stylů, HTML, skriptů a podobně. Dalším faktorem je rozlišení obrazovky, a to nejen na osobních počítačích. V dnešní době musíme počítat s tím že, stránky mohou být zobrazeny i na různých mobilních zařízeních. Je takřka nemožné docílit toho, aby se všem návštěvníkům zobrazily stránky naprosto

korektně. Proto je třeba při tvorbě zjistit, jaké budou hlavní cílové skupiny uživatelů a do jaké šíře se snažit o podporu různých prohlížečů a rozlišení a na jakých platformách. Pokud například budeme tvořit web pro mladé lidi, stojí za úvahu podpora zobrazení na mobilních zařízeních.

Podle statistik ze StatCounter byl nejpoužívanějším prohlížečem ke konci roku 2014 Chrome s 43 %, na druhém místě byl Firefox s 28 %, Internet Explorer používalo 18 % uživatelů. Ostatní prohlížeče nebyly zastoupeny více jak pěti procenty. [16]

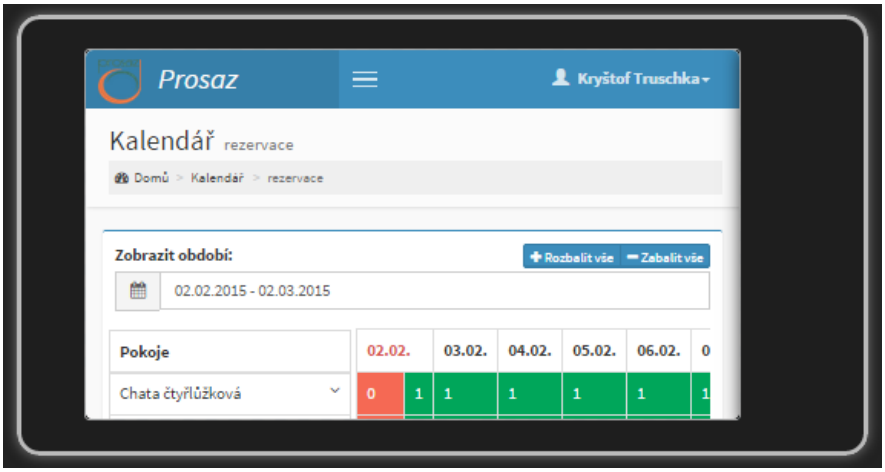
Po domluvě se zadavatelem bylo dohodnuto, že bude brán ohled jen na novější verze prohlížečů. Aplikace bude otestována v novějších verzích Chromu a Firefoxu. Internet Explorer bude podporován od verze 9.

Backendová část

Zadavatel nemá na vzhled administračního rozhraní žádné speciální požadavky. Kvůli urychlení projektu bude použita předpřipravená sada šablon pro administrační rozhraní. Tyto šablony jsou tvořeny HTML soubory, které obsahují návrh rozhraní, formuláře, tabulky, ikony a další grafické a funkční prvky, které se používají ve většině webových projektů. Na kodérovi je vybrat elementy, které se mu v projektu budou hodit a vytvořit vlastní šablony pro konkrétní stránky. Tento způsob ušetří velké množství času při vytváření kostry stránek i jednotlivých prvků. Šablony jsou ve formátu HTML a přiložené jsou i soubory s kaskádovými styly, případně skripty, které zajišťují dynamické chování na straně klienta.

Na internetu je dostupné velké množství šablon. Existují jak placené verze, tak šablony s volnou licencí. Pro tento projekt bylo vybíráno ze šablon, které jsou zdarma. Jako finální byla vybrána šablona Admin LTE vytvořená v HTML 5 a CSS3 s responsivním designem[19] založeném na Bootstrap 3. [17]

[19] Responzivní design – způsob vytváření stránky tak, aby ji bylo možné zobrazit na různých typech zařízení (počítačích, mobilních telefonech, tabletech apod.).



Obrázek 9 - Emulace zobrazení stránky na iPhone

Úpravy Admin LTE:

Některé skripty a soubory kaskádových stylů jsou ve výchozím nastavení načítány ze vzdálených serverů. V některých případech ale tento způsob zpomaloval dobu načítání stránek. Proto byly tyto sobory staženy a uloženy do adresářové struktury tak, aby mohly být načítány ze stejného serveru, na kterém bude samotná aplikace. Výjimkou jsou písma, která se načítají ze serveru společnosti Google. Google Fonts API vygeneruje CSS soubor, který je optimalizovaný pro prohlížeč uživatele a vybere správné formáty písma. [18]

Admin LTE obsahuje řadu užitečných javaScriptových pluginů. Některé však nebyly již delší dobu aktualizovány a v mezičase vyšly novější verze s vylepšenou funkcionalitou. Tam, kde to bylo vhodné, byly staženy a implementovány novější verze. K některým pluginům musela být přidána také lokalizace do českého jazyka.

Frontendová část

Při návrhu grafického rozhraní frontendové části byl kladen důraz na jednoduchost a přehlednost. Postačovat bude jednodušší struktura než v backendové části, proto bude vytvořena zcela nová šablona postavená na frameworku Bootstrap. Budou také využity některé javaScriptové pluginy z backendové části.

4.3. Implementace

Implementace probíhala tak, že byla naprogramována určitá část systému, například moduly nebo jejich skupina. Následně byla celá aplikace nahrána na testovací server, který byl z internetu zpřístupněn zaměstnancům zadavatele, kteří na projektu spolupracovali. Ti následně nově vytvořenou část otestovali a sepsali své názory a poznámky, které poté zaslali autorovi práce. V dohodnutém termínu pak proběhla schůzka, kde se o těchto připomínkách diskutovalo. Také byly probírány implementační detaily funkčních částí, které budou naprogramovány v nejbližší době.

V této části budou vysvětleny některé stěžejní funkce frameworku, který je jádrem celého systému. Také budou popsány konkrétní moduly rezervačního systému, které byly naprogramovány.

4.3.1. Nastavení serveru

Rezervační systém vyžaduje MySQL databázi a Apache server s těmito parametry:

- povolené nastavení serveru pomocí konfiguračního souboru `.htaccess`
- doporučená verze PHP je 5.5 a vyšší, minimálně PHP 5.3.7
- povolený `mod_rewrite`
- další doporučené moduly (`mod_deflate`, `mod_filter`, `mod_expires`, `mod_setenvif`, `mod_headers`, `mod_header`, `mod_mime`)

Pro správnou funkčnost aplikace je vyžadováno provést určitá nastavení serveru. Především přepisování URL[20] pomocí `mod_rewrite`. Modul `mod_rewrite` se používá přesměrování URL do souborového systému nebo na jinou URL pomocí předem nastavených pravidel. Každé pravidlo může obsahovat další podmínky založené na URL, serverových proměnných, hlavičce HTTP apod. Tyto pravidla mohou být nastavena v souboru `httpd.conf` nebo `.htaccess`. [19]

V použitém frameworku je pomocí `mod_rewrite` při každém HTTP požadavku prověřována požadovaná URL adresa. Je zjišťováno, zda se nejedná o adresu souboru, který má koncovku z výčtu povolených typů např. jpeg, jpg, css a podobně. Pokud je

[20] URL (Uniform Resource Locator) - řetězec s definovanou strukturou, který specifikuje umístění zdrojů informací. URL definuje protokol, doménovou adresu serveru a umístění zdroje na serveru.

podmínka splněna, kontroluje se, jestli soubor na serveru existuje. Pokud neexistuje, je zobrazena chybová stránka a je vrácen stavový kód HTTP 404. Pokud existuje, je vrácen příslušný soubor. Všechny ostatní požadavky jsou přesměrovány na `index.php`, kde se načítá framework a další parsování URL se provádí již v PHP.

Některé další části konfigurace v `.htaccess` byly v tomto projektu převzaty z ukázkového konfiguračního souboru projektu Boilerplate[21]. Ten obsahuje řadu nastavení vylepšujících výkon a bezpečnost webových stránek. [20]

4.3.2. Konfigurace aplikace

Jako první krok při spuštění aplikace se v souboru `index.php` načte konfigurační soubor `konfigurace.php`. Frontendová i backendová část mají každá vlastní. Tento soubor slouží k základní konfiguraci aplikace. Nastavují se zde zejména cesty k vybraným složkám, přístupové údaje k databázi, nastavení modulů a podobně. Tato globální nastavení jsou uložena v konstantách a lze je využívat napříč aplikací. V konfiguračním souboru se také pomocí funkcí `ini_set()` upravuje nastavení PHP v průběhu vykonávání skriptu. Mimo to se načítá také společný konfigurační soubor `globalniKonfigurace.php`, který je uložen ve složce `admin/_conf`, ve kterém jsou uloženy přístupové údaje k MySQL případně SMTP[22] serveru.

4.3.3. Kernel

Třída „Kernel“ slouží jako hlavní kontroler a je volána ihned po načtení konfiguračního souboru a spuštění autoloadingu. Stará se například o spuštění ukládání výstupů do bufferu, zapnutí session[23], výběr modulu, se kterým se bude pracovat. Kontroluje, jestli je přihlášen uživatel, pokud modul vyžaduje autentizaci. Předává autoloaderu název modulu, aby mohl načíst soubory potřebné k jeho zavedení. Vytváří instanci třídy `Sablona`, do které jsou v průběhu skriptu zasílána data a na konci zobrazena uživateli. Spouští kontroler vybraného modulu. Na konci zpracování zasílá požadavek odeslání výstupu ze šablony uživateli.

[21] Boilerplate projekt - HTML 5 šablona pro tvorbu rychlých a přizpůsobitelných stránek s řadou doplňků.
[22] SMTP (Simple Mail Transfer Protocol) – internetový protokol určený pro přenos e-mailových zpráv.
[23] HTTP session - HTTP je bezestavový protokol, avšak skoro každá aplikace potřebuje ukládat stav mezi požadavky. Session v protokolu HTTP dává webovému serveru možnost uložit si informace o uživatelích, kteří k němu přistupují, a to o každém zvlášť.

4.3.4. Autoloader

Aby při každém volání nové třídy nebylo potřeba ručně připojovat soubor, který tuto třídu obsahuje, např. pomocí funkce `require`, byla vytvořena třída „Autoloader“. V této třídě je pomocí funkce `spl_autoload_register()` registrována funkce `autoload($trida)`. Ta je spuštěna při vytváření každé nové instance všech tříd nebo například při volání metody abstraktní třídy. Tato metoda zjišťuje, jestli byla třída již zahrnuta, pokud ne, načte příslušné soubory.

V metodě `nastavZahrnutecesty()` je pomocí funkce `set_include_path()` upravena konfigurační direktiva `include_path`. Ta definuje, jaké složky budou prohledávány při volání PHP funkcí `require()`, `include()`, `fopen()` apod. V těchto složkách jsou také vyhledávány soubory při autoloadingu.

Ve výchozím nastavení jsou to složky `_kernel`, `_projekt` a složky rozšíření nastavené v konfiguračním souboru. Kromě takto staticky nastavených cest se zahrnuje také cesta k aktuálnímu modulu. To zajišťuje metoda `pridejModul($modul)`, která přidá cesty ke složce s modulem a jeho podsložce `_class` a také načte kontroler modulu. Pokud modul neexistuje, je automaticky načten modul „404“, který zobrazí hlášení, že požadovaná stránka neexistuje.

Aby autolader fungoval správně, je potřeba dodržovat určitou konvenci pojmenování a umístění souborů s potřebnými třídami. Soubory obsahující třídy modelu a pomocné třídy jsou proto pojmenovány `NazevTridy.class.php`, přitom je potřeba dbát na soulad použití velkých a malých písmen v názvu souboru a názvu třídy. Pokud by toto nebylo dodrženo, tak by na serveru s určitými operačními systémy aplikace nemusela fungovat. Kontroler modulu musí být v kořenové složce modulu a musí být pojmenován `nazevModulu.inc.php`. Název třídy musí být následující - `KontrolerNazevModulu`.

4.3.5. Dynamické přístupové metody

O dynamické vytváření přístupových metod se stará třída `Accessor`. Je to abstraktní třída, to znamená, že nemůže být samostatně instanciována, ale může být děděna. Tato třída rozšiřuje potomky o možnost využívání přístupových metod k nastavení a získání hodnot jejich atributů.

```
class Zamestnanec extends Accessor
{
  protected static $plat = 30000;
  protected $jmeno = null;
  protected $prijmeni = null;

  public function nastavPrijmeni($prijmeni)
  {
   if (strlen($prijmeni) < 4){
    Throw new Exception ("Příjmení musí být delší než 4 znaky.");
   }
   $this->prijmeni = $prijmeni;
  }
}

$karel = new Zamestnanec();
```

výpis kódu 1 – Příklad třídy, která dědí přístupové metody ze třídy Accessor

Ukázková třída `Zamestnanec` (viz výpis kódu 1) má tři atributy: `$jmeno`, `$prijmeni` a statickou proměnnou `$plat`. V proměnné `$karel` je instancí třídy `Zamestnanec`. Většinou není žádoucí, aby se k atributům mohlo přistupovat zvenčí přímo, například: `$karel->jmeno = "Karel";`.

Byla by tak ztracena kontrola nad obsahem nastavovaných proměnných. Proto mají proměnné viditelnost nastavenou na `protected` (chráněná), to znamená, že k nim má přístup pouze samotná třída, rodič nebo potomek. Zvenčí jsou dostupné pouze přes přístupové metody, jako je `nastavPrijmeni()` ve třídě `Zamestnanec`. Tato metoda například umožní zkontrolovat, jestli má příjmení alespoň 4 znaky. Ne všude je však kontrola vyžadována a bylo by zdlouhavé psát přístupové funkce pro všechny proměnné. Proto vznikla třída `Accessor`, která zavádí jednotné dynamické přístupové metody pro všechny chráněné proměnné (lze použít i pro veřejné). Navíc můžeme tyto metody ve třídě, která dědí z `Accessor` přetížit, jako je tomu v případě funkce `nastavPrijmeni()`. Stejně tak je možné modifikovat výstup pomocí přidání metody `vratPrijmeni()`.

Použití:

Přístupová funkce pro nastavování proměnných (`setter`) začíná klíčovým slovem „nastav“ doplněným názvem proměnné s počátečním velkým písmenem. Pro získání proměnné (`getter`) začíná slovem „vrat“ doplněným opět názvem proměnné s velkým počátečním písmenem. Následující kód ukazuje, jak je možné funkce použít jak u statických, tak dynamických proměnných. Metody `nastavJmeno()` a `nastavPlat()` i oba gettery jsou v tomto případě vytvářeny třídou `Accesor`.

```
$karel->nastavJmeno("Karel");
$karel->nastavPrijmeni("Novák");
Zamestnanec::nastavPlat(20000);
echo $karel->vratJmeno() ." - ". Zamestnanec::vratPlat() ." Kč";
```

výpis kódu 2 – Příklad použití přístupových metod

4.3.6. Doménové třídy

Doménové třídy jsou třídy, které popisují entity z reálného světa. Název doménové poukazuje na to, že jejich instance popisují entity z domény, kterou se aplikace zabývá.

Příkladem může být třída `Uživatel`, které obsahuje atributy jméno, příjmení, e-mail, heslo. Každá instance třídy reprezentuje jednoho konkrétního uživatele. V aplikaci tuto třídu využijeme téměř u všech procesů, kde budeme muset pracovat s údaji o uživateli.

Doménová třída by se dala srovnat s tabulkou v relační databázi, kde sloupce v relační tabulce odpovídají atributům třídy, řádky tabulky zase instancím doménové třídy. Doménová třída většinou pracuje s daty, které jsou perzistentně uložena v databázi. Tyto třídy zajišťují základní funkce CRUD[24], tedy vytvoření, čtení, modifikaci a mazání. Obsahují tedy databázové dotazy potřebné pro zajištění těchto funkcí a navenek poskytují rozhraní, které umožnuje volání těchto funkcí. Kromě toho jsou zde implementovány další funkce zajištující části aplikační logiky.

Bylo řečeno, že jedna třída odpovídá jedné tabulce, ale ve skutečnosti může jeden doménový objekt obsahovat data z více tabulek, někdy je totiž pro uložení v databázi vhodné tento objekt tvořící logický celek dekomponovat do více tabulek.

[24] CRUD (Create, Read, Update, Delete) je zkratka používaná v programování, shrnuje operace vytvoření, čtení, modifikaci a mazání.

Doménové objekty většinou nestojí samostatně, ale mají vazby na další objekty. V relační databázi jsou tyto vazby realizovány použitím cizích klíčů a vazebních tabulek. Doménová třída má také na starosti vytvořit objektově orientovanou reprezentaci těchto vazeb mezi objekty. Použitý framework využívá doménové třídy, jak zde byly popsány ve vrstvě modelu.

Implementace doménových objektů

Doménové třídy jsou v použitém frameworku potomky třídy `BO2`. Tato třída zajišťuje základní operace CRUD nad instancí objektu, bude popsána dále. Poskytuje také obecné přístupové metody k atributům doménových objektů, které dědí ze třídy `Accessor`.

Pokud potomek bude využívat zděděné funkce pro CRUD, musí obsahovat tři statické atributy:

- `$mapovaniDB`, který reprezentuje vazby mezi atributy třídy a souvisejícími sloupci v databázi. Jako klíče zde vystupují názvy atributů třídy a příslušné hodnoty jsou názvy sloupců v databázi.
- `$tabulkaDB`, ve kterém je uložen název databázové tabulky, se kterou se bude pracovat.
- `$kolekce` (dále označován jako **hlavní kolekce třídy**), která slouží jako kontejner pro vytvořené instance dané třídy s přiřazeným atributem `$id`.

Další důležitým atributem je `$id`, ten má také každý doménový objekt. `$id` zde má zvláštní význam, protože je v něm uložena hodnota primárního klíče z databáze. To znamená, že každá instance, která má nenulový atribut `$id` má odpovídající záznam v databázi. Toto `$id` je tedy v rámci všech instancí jedné třídy unikátní.

Dále musí být v konstruktoru volán konstruktor rodičovské třídy, protože se nedědí automaticky. V následujícím výpisu kódu je uveden příklad velmi jednoduché doménové třídy.

```php
class BlokovaniRezervace Extends BO2
{
    protected $id = null;
    protected $datumOd = null;
    protected static $tabulkaDB = "blokovani_rezervace";
    protected static $mapovaniDB = array(
    "id" =>"id_blokovani_rezervace",
    "datumOd" =>"datum_od",
    );
    protected static $kolekce;

    function__construct($id=null)
    {
       parent::__construct($id);
    }
}
```

výpis kódu 3 - Příklad implementace doménové třídy

Třída BO2 – Business object

Z této třídy dědí všechny doménové objekty. Protože je tato abstraktní třída jednou z nejdůležitějších, budou popsány její stěžejní metody. V konstruktoru `__construct($id=null)` třídy se rozhoduje, jestli se jedná o nově vytvářenou instanci, která není ještě uložená v databázi (`$id==null`), nebo se její proměnné budou naplňovat z databáze. Pokud je vyplněno `$id` je zavolána metoda `vrat()` (viz výpis kódu 4), která načte příslušné údaje z databáze. Tato metoda pracuje s atributy `$mapovaniDB` a `$tabulkaDB`.

```php
protected function vrat()
{
    $parametry = array("id"=>$this->id);
    $dotaz = "SELECT ".implode(",",array_values(static::$mapovaniDB))." FROM
    " . static::$tabulkaDB . " WHERE ". static::$mapovaniDB["id"]. " = :id
    LIMIT 1";
    $vysledek = DB::dotazJeden($dotaz,$parametry);
    if(!$vysledek) {
     Throw new Chyba ("Položka [".get_class($this)."] s tímto id
     neexistuje.");
    }
    $this->nactiAtributy($vysledek);
}
```

výpis kódu 4 – Metoda vrat()

Tato metoda vygeneruje dotaz podobný následujícímu a odešle ho ke zpracování do databázového systému.

```
SELECT id_blokovani_rezervace,datum_od FROM blokovani_rezervace WHERE
id_blokovani_rezervace = :id LIMIT 1
```

Pokud existuje záznam v databázi, je volána metoda `nactiAtributy()` (viz výpis kódu 5), která přiřazuje výsledky z databáze v asociativním poli v proměnné `$radek` k odpovídajícím instančním proměnným.

```
protected function nactiAtributy($radek)
{
     foreach (static::$mapovaniDB as $promenna => $sloupecDB)
      {
         if ($promenna == "id")
         {
            $id = ($radek[$sloupecDB])? $radek[$sloupecDB] : null;
         $this->nastavId($id);
         }
         elseif($radek[$sloupecDB] === null)
         {
         eval('$this->'. $promenna .' = null;');
         }
         else
         {
         $this->$promenna = $radek[$sloupecDB];
         }
      }
```

výpis kódu 6 - Metoda nactiAtributy()

Atribut `$id` zde má zvláštní význam, protože je v něm uložena hodnota primárního klíče z databáze a ve frameworku se s ním počítá, jako s jednoznačným identifikátorem objektu. Proto se při nastavování atributu `$id` volá metoda `nastavId()` (viz výpis kódu 6). Tato metoda zajistí, že se nově načtený objekt z databáze přidá do hlavní kolekce třídy. Nejdříve se zjistí, jestli je v atributu `$kolekce` již vytvořena kolekce pro ukládání

```
protected function nastavId($id)
{
  $this->id = $id;
  if(!static::$kolekce){
    static::$kolekce = new Kolekce(get_class($this));
  }
  static::$kolekce->pridej($this,$id);
}
```

výpis kódu 5 - Metoda nastvId()

instancí, pokud ne, tak ji vytvoří. Dále do kolekce přidá nově vytvořenou instanci.

Další metodou je metoda `ulozit()`, která instanci uloží do databáze. Pokud je nastaven atribut `$id`, znamená to, že již v databázi existuje a bude proveden „update“. Pokud ne, provede se „insert“ a zároveň se zavolá funkce `nastavId($id)`, která nastaví atribut `$id` na hodnotu primárního klíče nově vzniklého záznamu.

Metoda `smazat()` smaže existující záznam z tabulky. Před provedením příslušného databázového dotazu je volána funkce `kontrolaID()`. Tato metoda ověřuje, jestli je nastaven atribut `$id`. Pokud není, znamená to, že záznam nemůže být smazán, protože ještě nebyl v databázi vytvořen.

4.3.7. Kolekce

Často je potřeba mít uloženo více objektů v nějakém seznamu, to se dá řešit jejich uložením do pole. V případě použitého frameworku je zastřešeno nadstavbou v podobě třídy `Kolekce`, která má oproti poli řadu výhod.

Nová kolekce se vytváří pomocí příkazu `$kolekce = new Kolekce ($trida);`. Volitelný parametr `$trida` obsahuje název třídy, jejíž instance je povoleno do kolekce přidávat.

Třída `Kolekce` obsahuje pole v atributu `$polozky`, do kterého se ukládají další objekty. Objekty se do kolekce přidávají metodou `pridej($obj, $klic = null)`, kde prvním atributem je instance objektu a druhým klíč, pod kterým se do pole vloží. Tento klíč musí být unikátní v celém poli, jinak dojde k vyhození výjimky. Pokud parametr `$klic` není vyplněn, bude použit jako klíč atribut `$id` přidávané instance.

Kolekce také zahrnuje některé metody, které provádí standardní operace nad polem položek. Například vracení počtu všech položek, vraceni první položky, vráceni klíčů pole položek apod. Dále obsahuje metodu, které nastavuje callback funkci, která bude rozebrána dále.

Třída kolekce implementuje rozhraní `IteratorAggregate`, To spolu s deklarací metody `getIterator()` umožňuje traverzování kolekce. Kolekce je možné přímo procházet pomocí cyklus `foreach` podobně jako pole.

Lazyloading

Lazyloading neboli líná instanciace slouží k tomu, aby se data nenahrávala hned, ale až když jsou potřeba.

Bylo již řečeno, že u doménových objektů se většinou přiřazují k atributům data po vytvoření instance (pokud existují v databázi). Jsou to ale data jen z jedné tabulky. Ve skutečnosti jsou ale objekty často provázané. Nejlepší bude vše vysvětlit na třídě `Rezervace`, která je součástí rezervačního systému.

Obrázek 10 - Vazba tabulky rezervace na pokoj_rezervace

Z ER diagramu je vidět, že mezi rezervací a pokoji, které jsou v ní zarezervovány (`pokoj_rezervace`) je vazba 1:N. To znamená, že v jedné rezervaci může být více pokojů. Musí se tedy zajistit, aby se daly k rezervaci tyto pokoje v PHP přiřadit. Právě k tomu výborně poslouží kolekce. Ve třídě `Rezervace` se vytvoří atribut `$pokojRezervace` v konstruktoru třídy se pak definuje, že se jedná o kolekci a zároveň se nastaví callback.

```
$this->pokojRezervace = new Kolekce('PokojRezervace');
$this->pokojRezervace->nastavCallback('kolekceSouvisejicich', $this);
```

Tato kolekce zůstává prázdná, dokud se na instanci třídy rezervace nezavolá nějaká funkce, která bude pracovat s instancemi uloženými v kolekci.

Například:
`$rezervace->vratPokoje()->vratPolozky();`

Metoda `vratPolozky()` vrací instance z pole v kolekci. Při jejím prvním volání je spuštěna callback funkce kolekce a dojde k jejímu naplnění příslušnými instancemi třídy `PokojRezervace`, které jsou načteny z tabulky `pokoj_rezervace`.

Výhodou je, že kolekce je naplněna vždy jen jednou, pokud se znovu zavolá `$rezervace->vratPokoje()->vratPolozky()`, vrátí se již naplněná kolekce a nemusí se přistupovat do databáze. Callback funkce není spouštěna jen při `vratPolozky()`, ale i při volání všech ostatních metod, které pracují s jejími položkami například `vratPocet()`.

Celý princip spočívá v tom, že při volání každé z těchto metod je zkontrolováním atributu `$nacteno`. V něm je uložena logická hodnota `true`, pokud již byla předtím spuštěna callback funkce. Pokud je `$nacteno` nastaven na `false`, je callback funkce spuštěna a `$nacteno` je nastaven na `true`.

V příkladu byla callback funkce nastavena tak, aby byla volána metoda `kolekceSouvisejicich()` (viz výpis kódu 7) na instanci třídy `Rezervace`. Tato metoda je součástí třídy `BO2` a třída `Rezervace` ji tudíž dědí.

```
public function kolekceSouvisejicich(Kolekce $kolekce)
{
   if($this->id)
   {
      $trida = $kolekce->vratPovoleneObjektyTridy();
      $dotaz = "SELECT " .$trida::$mapovaniDB["id"] . " FROM "
      .$trida::$tabulkaDB. " WHERE ".static::$mapovaniDB["id"]." = :id";
      $parametry = array("id" => $this->id);
      $poleID = DB::dotazJedenSloupec($dotaz,$parametry);
      $trida::Kolekce($kolekce,$poleID);
   }
}
```

výpis kódu 7 – Metoda kolekceSouvisejicich()

Konkrétně tato metoda vybere z tabulky `pokoj_rezervace` všechny řádky které mají cizí klíč `id_rezervace` roven atributu `$id` rezervace, pro kterou kolekci vytváříme (tedy jejímu primárnímu klíči). Z nich se pak vytvoří instance a uloží do kolekce. Zde byla použita již předpřipravená metoda `kolekceSouvisejicich()`. Není ale problém vytvořit jinou individuální metodu, třeba s dotazem s více podmínkami. Důležité je, aby tato metoda přijímala jeden parametr, a to kolekci, která se bude plnit.

Hromadné plnění kolekcí

Na předchozím příkladu je ještě možné dovysvětlit, jak se ve frameworku vytváří hromadně instance. V metodě `kolekceSouvisejících()` (viz výpis kódu 7) se kolekce plní voláním `$trida::Kolekce($kolekce,$poleID);`.

`$trida` je zde `PokojRezervace` a `$trida::Kolekce()` je metoda, kterou dědí doménové objekty z `BO2`. Tato metoda přijímá jako první parametr kolekci, kterou bude plnit. Dále `$poleID`, které obsahuje pole unikátních identifikátorů (atribut `$id`), objektů, které budou do kolekce zařazeny. Pokud je voláno `PokojRezervace::Kolekce($kolekce,array(1,2));` budou do kolekce zařazeny dvě instance `PokojRezervace` - jedna s `$id=1` a druhá s `$id=2`. Tato metoda funguje tak, že v první řadě je pro všechny hodnoty v `$poleID` prohledána hlavní kolekce třídy[25], ze které se budou vytvářet instance. Je zjišťováno, jestli zde již není instance s požadovaným `id` vytvořena. Pokud již existuje, může být přidána do kolekce a již nemusí být znovu načítána z databáze. Zároveň je z `$poleID` hodnota tohoto identifikátoru odstraněna. Na konci cyklu jsou v proměnné `$poleID` identifikátory těch instancí, které ještě nebyly vytvořeny. Poté je sestaven dotaz, který vybere z příslušné tabulky řádky s primárními klíči v `$poleID`, vytvoří z nich objekty a přidá do kolekce. Tento proces slouží k tomu, aby nebyly z databáze zbytečně načítány objekty, které již existují.

4.3.8. Vytvoření jedné instance doménové třídy

Ve frameworku se k vytváření nových instancí doménových objektů nepoužívá příkaz `$instance = new Trida($parametry)`. Místo toho se používá třída `N`, která

[25] hlavní kolekce třídy – kolekce, ve které jsou všechny instance dané třídy s nastaveným atributem $id

vrací požadovanou instanci. Zápis je: `N::Trida($id)`. Třída `N` obsahuje magickou metodu `__callStatic($trida,$arg)`. Tato metoda je spouštěna při volání jakékoliv statické metody třídy `N`. Pokud je zavoláno `N::Trida($id)`, získá z názvu volané metody název třídy, pro kterou je potřeba vytvořit instanci. Identifikátor instance je v parametru `$id`. Nejdříve je zkontrolováno, jestli třída existuje. Pokud ano, je prohledána hlavní kolekce třídy a je zjištěno, zda existuje instance s atributem `$id`, který je roven parametru `$id`. Pokud existuje, je vrácena tato instance. Jestliže ne, je vytvořena a vrácena nová.

```
class N
{
   public static function__callStatic($trida,$arg)
   {
      $id = (isset($arg[0]))? $arg[0] : null;
      if(!class_exists($trida))
      {
       throw new Chyba("Třída '$trida' neexistuje");
      }
      if($id and $trida::vratKolekce() and
         $trida::vratKolekce()->existuje($id))
      {
       return $trida::vratKolekce()->vrat($id);
      }
      else
      {
       return new $trida($id);
      }
   }
}
```

výpis kódu 8 - Třída "N"

Tento způsob zajišťuje to, že každá instance s nastaveným atributem `$id` bude vytvořena pouze jednou. Pokud je jednou instance vytvořena, není pro její získání znovu potřeba přistupovat do databáze, pokud k ní bude přistupováno přes třídu `N`.

4.3.9. Šablonovací systém Smarty

Architektura MVC přímo vybízí k použití nějakého šablonovacího (templatovacího) systému v prezenční vrstvě. Samozřejmě by se daly vytvářet jen jednoduché `phtml`[26] šablony. Část vývojářů se kloní k názoru, že PHP je již sám o sobě šablonovací engine

[26] phtml šablona – Myšleno jako šablona vytvořená z mixu HTML a PHP. Pro tyto soubory se často používá koncovka phtml.

a nemá cenu ho dále rozšiřovat, zvláště vzhledem k dodatečnému spotřebování výkonu při použití šablonovací nástavby.

Na druhou stranu nám však šablonovací systém umožní snadno oddělit aplikační logiky obsahující PHP od prezenční vrstvy, obsahující HTML a jednoduché příkazy nutné k zobrazení. Proto se autor práce nakonec přiklonil k použití zavedeného open source šablonovacího systému pro PHP – Smarty [21].

Jednotlivé šablony pro Smarty se v rámci tohoto projektu vytváří v HTML, do kterého se vkládají značky nebo bloky kódu ve speciální syntaxi. Tyto šablony Smarty využijí k vygenerování výsledného HTML kódu. Soubory se šablonami mají koncovku `tpl`. Ve Smarty je možné vytvářet také funkce, cykly, vkládat subšablony, modifikovat výstup pomocí přednastavených funkcí a podobně. Data do šablony jsou předávána např.:

```
$smarty->assign('jmeno',$jmenoUzivatele);
```

Kde `$smarty` je instance třídy Smarty V šabloně pak může být například blok `<b>{$jmeno}<b>`. Při zpracování šablony je za `$jmeno` dosazena hodnota proměnné `$jmenoUzivatele`.

Smarty při prvním spuštění skriptu zkompiluje šablonu do PHP a uloží ji pro další použití na webový server. Při dalším zpracování používá již zkompilovanou verzi, což urychluje dobu zpracování.

Další zajímavou funkcí je cachování. Smarty může před odesláním výstupu v podobě HTML kódu uložit tento výstup spolu s dalšími interními informacemi (např. kdy byla cache vytvořena, jaká šablona byla použita apod.) do souboru na webovém serveru. Při dalším volání skriptu se může rovnou zobrazit tento výstup a nemusí se provádět další operace, které byly potřebné k vytvoření tohoto výstupu. To se hodí především u stránek, které mají vysokou návštěvnost a obsah je poměrně statického charakteru. Příkladem může být internetový obchod, kde se na hlavní stránce zobrazuje 20 produktů. Bez použití cachování by se s každým přístupem musely provádět výběry z databáze a další operace spojené s výpisem oněch produktů. Pokud ovšem bude stačit, že se informace aktualizují jen jednou za den, nastaví se životnost cache na 24 hodin. Do té doby se bude zobrazovat

pořád stejný výstup a část PHP skriptu nebude muset být vykonávána. Po vypršení cache se proces zopakuje.

Na následujícím příkladu je uveden rozdílný zápis při použití `phtml` a šablony Smarty.

```
<div>
{foreach from=$kolekceUzivatelu->vratPolozky() key=uzivatelId item=uzivatel}
  {$uzivatel->vratPrijmeni()|upper} {$uzivatel->vratJmeno()} <br />
{/foreach}
</div>
```

výpis kódu 10 - Šablona "phtml"

```
<div><?php
foreach ($kolekceUzivatelu->vratPolozky()  as $uzivatelId => $uzivatel )
  {
echo strtoupper($uzivatel->vratPrijmeni())." ".$uzivatel->vratJmeno();
?><br/><?php
  }
?></div>
```

výpis kódu 9 - Šablona systému Smarty

Ačkoli jsou Smarty užitečným nástrojem, jejich nevýhodou je dokumentace, ta často není kompletní, či nekoresponduje s aktuální verzí Smarty.

4.3.10. Třída Šablona

Třída `Sablona` slouží jako další nadstavba Smarty. Většina stránek se skládá ze tří dílčích šablon ve formátu `tpl`: hlavičky, patičky a těla. Třída `Sablona` je skládá dohromady, přičemž u každé části je možné nastavit, jaký `tpl` soubor má být využit, jestli se má vytvořit cache, případně na jak dlouho.

První instance třídy `Sablona` je vytvářena ve třídě `Kernel` a je zde uložena do statické proměnné `$sablona`. Tato šablona je hlavní a je v průběhu skriptu naplněna daty a na konci vypíše HTML kód.

Jako výchozí `tpl` šablony jsou pro jednotlivé části přednastaveny soubory `hlavicka.tpl`, `paticka.tpl` a pro tělo `nazevModulu.tpl`. To znamená, že výchozí šablona pro tělo modulu „uzivatel“ je `uzivatel.tpl`. Tyto soubory jsou načítány ze společného adresáře šablon a z adresáře šablon modulu. Proto je vhodné umísťovat šablony používané ve více modulech do společné složky a šablony, které patří

k modulům, do podsložky `_sablony` přímo k modulu. Pokud existují dvě šablony se stejným názvem, má vždy přednost šablona modulu.

Některé moduly mohou používat více `tpl` šablon. Například modul „uzivatel“ může mít jednu šablonu pro vypsání všech uživatelů a druhou pro editaci konkrétního uživatele. V kontroleru modulu lze jednoduše zvolit soubor, který bude použit.

```
Kernel::$sablona->nastavTelo('uzivatel.editace.tpl');
```

V architektuře MVC předává data pro pohled kontroler. Data jsou do šablony přidávána kontrolery modulů prostřednictvím metody `assign()`. Tímto způsobem je možné do šablony předávat proměnné, může se jednat například o proměnné typu řetězec, pole či objekty.

```
Kernel::$sablona->assign('uzivatel', $uzivatel);
```

Ve třídě `Sablona` je nastaveno, aby Smarty na všechny vypisované proměnné aplikovalo PHP funkci `htmlspecialchars()`, která zamění speciální znaky za jejich HTML entity. Toto napomáhá proti XSS[27] útokům. Pokud je proměnnou potřeba zobrazit bez escapování speciálních znaků, lze v šabloně použít příznak nofilter např.: `{$obsahClanku nofilter}`.

Automatické načítaní skriptů a stylů

Cesta k JavaScriptovým a CSS soborům, které jsou využívány napříč většinou modulů (např. jQuery[28], bootstrap), jsou uloženy ve standardní šabloně hlavičky `hlavicka.tpl`. Často je ale potřeba načíst i další skripty a styly, které se váží jen ke konkrétnímu modulu. Toho lze docílit tak, že se soubory s těmito skripty umístí do podsložky `_js` konkrétního modulu. Tato složka je při zpracování třídy `Sablona` prohledána a cesty ke všem souborům s koncovkou `js` jsou zahrnuty do HTML kódu hlavičky. Obdobně lze přidat i kaskádové styly.

[27] XSS (Cross-sitescripting) - metoda napadení webových stánek využívající bezpečnostní chyby ve skriptech. Útočníkovi umožňuje vložit vlastní skript do webové stránky.
[28] jQuery - multiplatformní JavaScriptová knihovna, která klade důraz na interakci mezi JavaScriptem a HTML.

4.3.11. Komunikace s databází

Třída DB zprostředkovává spojení a komunikaci s databází. V rámci tohoto projektu se počítá s tím, že bude využívána jen jedna databáze. Proto třída `DB` udržuje jedno spojení, které je inicializováno ve třídě `Kernel`. Pro práci s databází je využíváno objektové rozhraní databází PDO (*PHP Data Objects*). PDO umožňuje jednotným způsobem pracovat s rozličnými databázemi.

Instance PDO se vytváří voláním metody `DB::pripoj($host, $uzivatel, $heslo, $databaze, $port=null)` a následně je udržováno ve statické proměnné třídy `DB` `$spojeni`.

Výhodou PDO je ochrana proti `SQL Injection`[29]. Dotazy je ale potřeba připravit a až poté vykonat. Metody třídy `DB` pro zpracování dotazů to zajistí, ale musí obdržet parametry ve správném formátu. Existuje několik možností, jak PDO předat dotazy před přípravou a příslušné parametry. V tomto projektu je použit následující zápis. Do dotazů se vkládají místo hodnot pojmenované parametry začínající dvojtečkou. Hodnoty jsou zasílány separátně ve formě asociativního pole, kde klíčem je pojmenování parametru z dotazu a hodnotou řetězec, který za něj bude dosazen.

```
$dotaz = "UPDATE zakaznik SET email=:email, heslo=:heslo WHERE
id_zakaznik=:id";
$parametry = array("email" =>"email@email.cz", "heslo" =>"hJdfU54U", "id" =>
1);
DB::dotaz($dotaz,$parametry);
```

výpis kódu 11 - Ukázka vytvoření databázového dotazu s oddělenými parametry

Třída `DB` obsahuje předpřipravené metody, které vracejí výsledky v často používaných formátech. Například metoda `dotazJeden($dotaz, $parametry)` vrátí první řádek z databáze jako asociativní pole. Další funkce vrací například všechny načtené řádky v poli asociativních polí apod.

PDO nabízí funkce pro řízení transakcí. Přes třídu `DB` se transakce spouští voláním `DB::spustitTransakci()` potvrzení změny pomocí `DB::commit()` a zrušení

[29]SQL injection je technika napadení databázové vrstvy programu vložením vlastního kódu do původního SQL dotazu prostřednictvím neošetřeného vstupu.

změn prostřednictvím `DB::rollBack()`. MySQL databáze musí být ve formátu transakce InnoDB, který podporuje.

4.3.12. Generování PDF dokumentů

Často je potřeba vytvořit dokumenty, které budou na všech zařízeních zobrazeny stejně, a to pomocí nějakého rozšířeného zdarma dostupného prohlížeče. Ideálním formátem je PDF, který tyto předpoklady splňuje. Pro vytváření dokumentů PDF v rezervačním systému je použit PHP konvertor dompdf [22] , který převádí HTML do PDF. Výhodou je jeho snadné použití. Následující ukázka vytvoří pomocí Smarty HTML obsah, který je předán do konvertoru. Ten ho převede do PDF a nabídne ke stažení.

```
$sablona = new Sablona();
$sablona->assign('objednavka', $objednavka);
$html = $sablona->fetch();
$dompdf = new PDF();
$dompdf->load_html($html);
$dompdf->render();
$dompdf->stream("faktura.pdf");
```

výpis kódu 12- Příklad generování PDF ze Smarty šablony

Pro správný převod do PDF musí být zdrojový HTML kód validní. Ačkoli je částečně podporováno HTML 5 a CSS 3, ne vždy je výstup zobrazen správně. Proto je důležité řádně otestovat, jestli se výsledné dokumenty zobrazují správně. Pokud ne, je třeba upravit HTML, ze kterého je dokument PDF generován. Z těchto důvodů se dompdf hodí především pro převod HTML s jednoduchou strukturou. Dalším problémem jsou vysoké nároky na paměť a dlouhá doba převodu u větších nebo složitějších dokumentů. Původně byl konvertor využíván také ke generování stravenek. Na jedné stránce bylo 27 stravenek umístěných v tabulce. Pokud bylo třeba vygenerovat 350 stravenek, byla doba převodu přes půl minuty a bylo využito přes 128MB paměti. Přitom tyto parametry bývají na webhostinzích často omezeny. Kvůli tomu se nyní stravenky tisknou přímo z HTML.

4.3.13. Zabezpečení

Hesla nejsou do databáze ukládána v podobě, jakou zadal uživatel, ale nejdříve je vytvořen jejich hash a teprve ten uložen. Mělo by být zajištěno, aby ani provozovatel aplikace nemohl zjistit heslo v původní podobě, i kdyby měl přístup ke zdrojovým kódům aplikace. Běžné hashovací funkce, jako MD5, SHA-1 jsou příliš rychlé, proto je možné heslo

v poměrně krátkém čase prolomit. [23] Proto byla použita PHP funkce password_hash, v současné době využívající algoritmus bcrypt. Funkce password_hash je dostupná od PHP verze 5.5, pro verzi PHP 5.3.7 je použita knihovna password_compact [24], která zavadí zpětnou kompatibilitu. [25]

Ochrana proti CSRF

CSRF (Cross Site Request Forgery) je druh útoku, kdy je zneužívána identita uživatele. Útočníkovi dovoluje provést takovou akci, kterou by mohl provést za normálních okolností jen autorizovaný uživatel. Existuje mnoho možností, jak tento útok provést. Zde bude uveden jednoduchý příklad: uživatel je přihlášen na nějaké stránky, kam může vkládat příspěvky. Útočník ví, že mazání konkrétního příspěvku se provádí z administrace při kliknutí na odkaz `www.domena.cz?smazatPrispevek=idPrispevku`.V tom případě může uživateli, přes kterého chce útok provést, zaslat tento odkaz a navést ho, aby na něj klikl. Nebo ho třeba zamaskovat do obrázku (např.: `<img src='www.domena.cz?smazatPrispevek=3 width='0' height='0'>`) a vložit někam, kde se uživateli zobrazí. Při pokusu o načtení obrázku se zavolá URL v parametru `src`. Pokud je uživatel při volání této url přihlášen do systému, dojde ke smazání příspěvku. Obecně by se pro vykonávání akcí jako mazání, vkládání apod. neměla používat metoda GET. Je však nutno podotknout, že CSRF útok se dá provést i při použití POST požadavku. Například odesláním podvrženého formuláře na cílovou adresu při příchodu oběti na stránku vytvořenou útočníkem. [26] [27]

V rezervačním systému se pro tyto zásadní akce používají požadavky typu POST. Navíc byla zavedena ochrana pomocí tokenů v modulech, kde je nutné přihlášení. Do každého formuláře, který odesílá metodou POST je vložen skrytý „input“. Ten obsahuje token, vygenerovaný při přihlášení uživatele, který je zároveň uložen do SESSION. Vkládání inputu do formulářů se provádí automaticky pomocí filtru, který je spuštěn před kompilací šablony Smarty. Při zachycení POST požadavku se kontroluje shoda tokenu odeslaného z formuláře a uloženého v session. Bez znalosti tokenu nemá útočník možnost CSRF použít.

Session hijacking a session fixation

Seesion hijacking je ukradení platného identifikátoru relace. Session Fixation je útok, který spočívá v tom, že útočník podvrhne oběti svůj session identifikátor a počká, až se oběť přihlásí. [28, s. 367] V obou případech se díky znalosti tohoto podvrženého identifikátoru může začít za oběť vydávat. Identifikátory relací je v systému povoleno přenášet pouze v cookies. Pro zvýšení bezpečnosti byly upraveny některé konfigurační direktivy týkající se sessions.

4.3.14. Optimalizace výkonu

Testování rychlosti načítání stránek bylo prováděno pomocí webového nástroje PageSpeed Insight od Google. Tento nástroj prověří nastavení serveru, HTML strukturu stránky, načítání externích zdrojů, jako CSS, Javasctipt a obrázky. Výkon je hodnocen na stupnici 0 až 100 bodů, zvlášť pro počítače i mobilní zařízení. Skóre 85 bodů a více indikuje velmi dobře optimalizovanou stránku.

Pro zvýšení rychlosti byly externí JavaScriptové soubory, u kterých to dávalo smysl, sloučeny do jednoho. To samé bylo provedeno s CSS soubory, ty byly navíc v některých případech minimalizovány (odstraněním mezer a sloučením do jednoho řádku).

Dalšího zrychlení bylo docíleno povolením gzip[30] komprese obsahu stahovaného ze serveru. Soubory jsou na straně serveru komprimovány a na straně uživatele rozbaleny, což šetří množství přenášených dat. Moderní prohlížeče automaticky žádají přes protokol HTTP o komprimovaný obsah. U větších souborů s textovým obsahem dosahuje míra komprese 70-90% [29] Nastavení komprimace se provádí přes konfigurační soubory `.htaccess` nebo `httpd.conf`. Podmínkou je povolený modul `mod_deflate` na Apache serveru.

Ke zvýšení výkonu bylo také přenastaveno cachování souborů v prohlížeči. Některé soubory, které již byly staženy v minulosti, není třeba stahovat znovu, protože se mohou ukládat na straně uživatele. Pro potřeby cachování je nutné nakonfigurovat Etagy, které jsou součástí HTTP hlavičky. Etag je generován na straně serveru a obsahuje jednoznačné určení verze souboru. Pomocí Etagu je možné porovnat, jestli se soubor na serveru

[30] Gzip - aplikační software užívaný pro kompresi dat založený na algoritmu DEFLATE

a v cache prohlížeče shoduje. Pokud jsou sobory stejné, není je potřeba přenášet znovu. Také by měla být nastavena maximální doba, po kterou je cache platná. Tato doba je opět uvedena v HTTP hlavičce a je jí možné různě uzpůsobovat pomocí konfigurace Apache serveru [30]

Po provedení změn bylo na testovacím serveru naměřeno pomocí PageSpeed Insight pěkné skóre od 75 bodů pro mobilní zařízení a od 85 pro počítače.

4.4. Popis modulů aplikace

4.4.1. Modul Pokoje

Tento modul slouží ke správě pokojů a skupin pokojů. Na úvodní obrazovce modulu je seznam vytvořených skupin pokojů a konkrétních pokojů s možností přidávání a editace.

Skupina zastřešuje všechny pokoje stejného typu, například všechny dvoulůžkové pokoje, a ty dědí její vlastnosti. Pokud dojde např. ke změnám cen za lůžko, stačí změnit ceny u skupin pokojů a nemusí být měněny u jednotlivých pokojů v této skupině. Pro každou skupinu pokojů je možné nastavit:

Název - Podle něj by měla být skupina odlišitelná od ostatních (např. Pokoj dvoulůžkový s koupelnou).
Cena za pokoj - Cena, kterou za jednu noc zákazník zaplatí vždy, bez ohledu na počet osob ubytovaných v pokoji.
Cena za lůžko - Cena za každé obsazené lůžko a noc.
Počet lůžek - Počet lůžek v pokoji.
Penále za neobsazené lůžko -Penále, které se platí za každé neobsazené lůžko a noc v případě, že je počet hostů menší než počet lůžek v pokoji.
Minimálně ubytovaných osob – Minimální počet osob, které mohou být ubytovány v pokoji.

Cena za noc v pokoji ze skupiny se vypočítá podle vzorce:

```
$cena = $this->cenaZaPokoj + (($this->cenaZaLuzko * $pocetHostu) +
(($this->pocetLuzek - $pocetHostu) * $this->penaleZaNeobsazene));
```

Jeden z požadavků byl, aby se za chatu platila pevná částka a za ubytování v pokojích podle počtu obsazených, respektive neobsazených lůžek. Toho lze dosáhnou nastavením parametrů skupiny pokojů dle tabulky. Samozřejmě je možné vytvářet i další různé kombinace podle potřeby.

Název skupiny	Cena za pokoj	Cena za lůžko	Počet lůžek	Penále za neobsazené lůžko
Chata	vyplněna	0	vyplněno	0
Pokoj	0	vyplněna	vyplněno	vyplněno

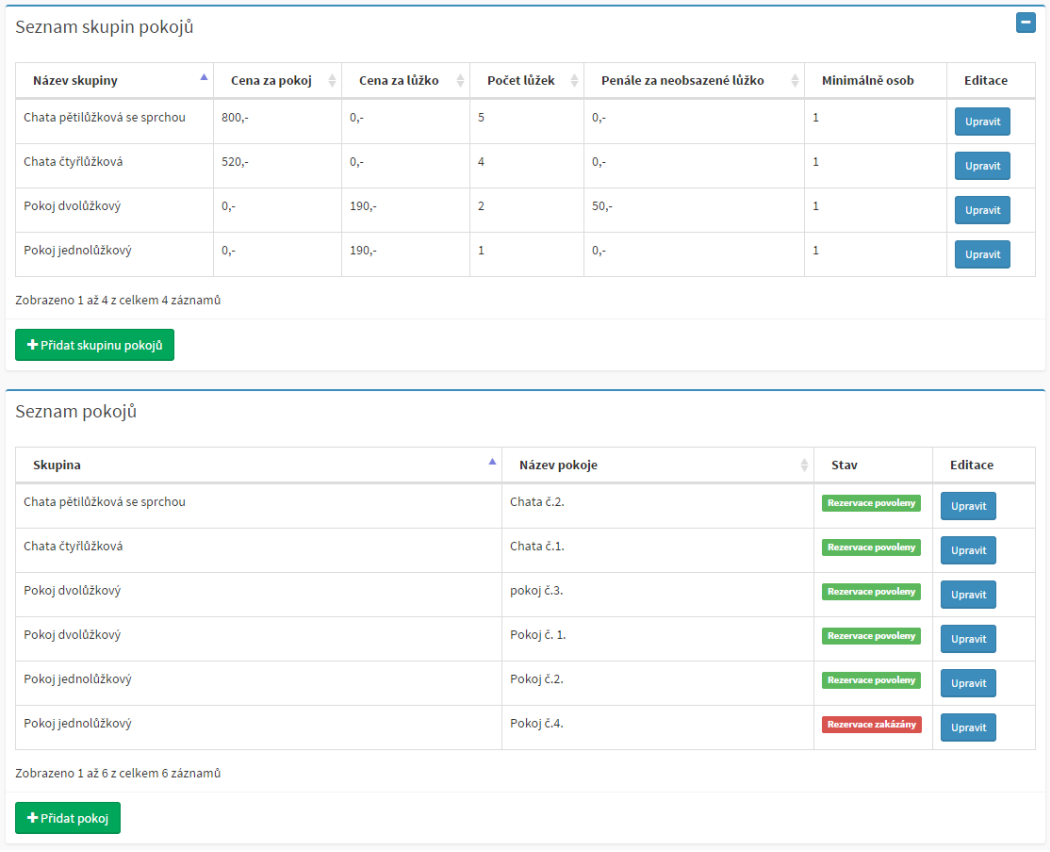
Tabulka 1- příklad nastavení cen pokojů a chat

Kromě skupin pokojů může administrátor přidávat i konkrétní pokoje. Každý pokoj má tři měnitelné atributy:

Název - (např. pokoj č. 4)

Skupina – zařazení v jedné z existujících skupin pokojů.

Stav – Pokoj může být ve stavu *„rezervace povoleny“* nebo *„rezervace zakázány“*. Pokud jsou rezervace zakázány, nelze pro tento pokoj vytvářet nové rezervace. Stávající rezervace nejsou změnou stavu ovlivněny.



Obrázek 11 -Náhled obrazovky s přehledem skupin pokojů a konkrétních pokojů

4.4.2. Vytvoření rezervace

Vytvoření a editace rezervace jsou rozděleny do více provázaných modulů:

- Ubytování a stravování
- Fakturační údaje
- Shrnutí
- Doplňkové služby
- Stav rezervace

Zákazník využívá při vytváření rezervace jen první tři moduly. V první fázi vybírá pokoje, dobu pobytu a typ stravování. V druhém vyplňuje fakturační a kontaktní údaje a ve třetím potvrdí a odešle celou rezervaci.

Výběr ubytování

Výběr doby pobytu se provádí prostřednictvím vstupního pole, které zobrazí kalendář s možností zadání rozsahu od zahájení do ukončení pobytu. Pokud uživatel ještě dobu pobytu nezadal, je nastaven sedmidenní pobyt počínající aktuálním datem. V modulu „Obecná nastavení“ může administrátor zvolit, kolik dní může maximálně zbývat do začátku pobytu. Toto opatření je zde proto, aby nemohly být vytvářeny rezervace začínající např. až za dva roky. Ve stejném modulu lze také nastavit maximální počet nocí v rámci rezervace. Začátek doby rezervace je také limitován aktuálním datem. Není možné vytvářet rezervace v období, které již uplynulo.

Níže je zobrazena tabulka, ve které jsou skupiny pokojů a cenové varianty podle počtu ubytovaných osob. Například ve skupině čtyřlůžkový pokoj může být ubytován jeden až čtyři hosté. Ceny za noc se však mohou lišit podle toho, kolik lidí bude v pokoji ubytováno. Rozdílnou cenu za noc způsobuje penále za neobsazené lůžko.

Zobrazen je také počet volných pokojů pro vybrané období. V celé skupině je vždy stejný počet volných pokojů bez ohledu na variantu, protože i v případě, že bude v pokoji jeden člověk, je pokoj obsazený.

Existují dva způsoby výběru pokojů, ve kterých budou hosté ubytování:

1. automatický výběr pokojů

Uživatel volí jen počet pokojů ve skupině a variantě podle počtu ubytovaných osob. Konkrétní pokoje jsou systémem vybrány automaticky tak, že jsou prohledány všechny volné pokoje a přiřazen je vždy první dostupný.

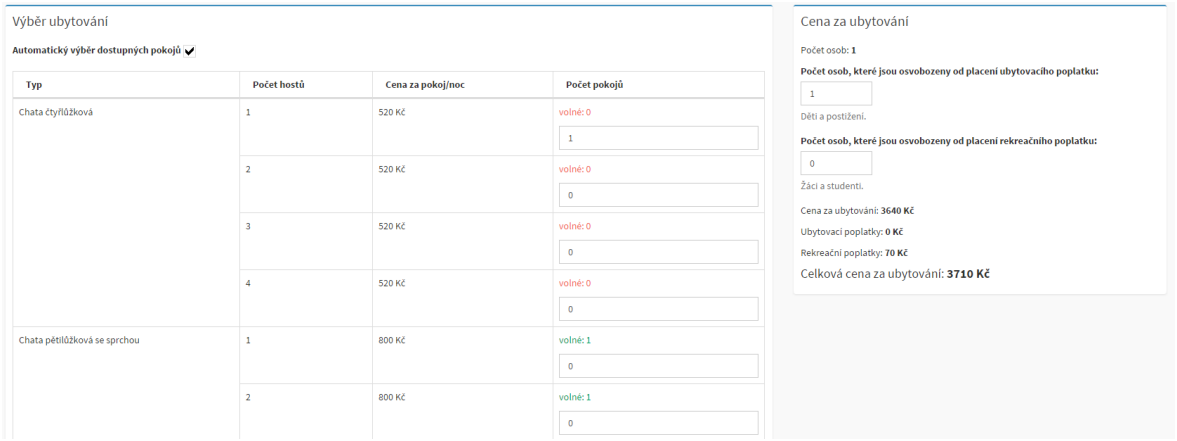
2. manuální výběr konkrétních pokojů

Při tomto způsobu vybírá konkrétní pokoje uživatel. Pro každou skupinu a variantu jsou zobrazeny pokoje, které do ní spadají. Uživatel má možnost vybrat konkrétní volný pokoj. Pokoje obsazené v jiných rezervacích jsou zobrazeny, ale není možné je vybrat. Každá varianta ve skupině obsahuje stejné pokoje, konkrétní pokoj jde však vybrat jen jednou v rámci všech variant, aby se nestalo, že „Pokoj č. 1“ bude vybrán ve variantě pro jednu i dvě osoby.

V backendové části si může uživatel zvolit, který ze způsobů bude používat. Může použít automatický pro rychlé vytvoření rezervace nebo vybere pokoje ručně podle svých preferencí, případně požadavků hostů (např. dva sousedící pokoje).

Ve frontendové části je možný jen automatický výběru pokojů, protože hosty ve většině případů nezajímá, jaký konkrétní pokoj dostanou, ale vybírají podle počtu lůžek apod. Pokud má host speciální požadavky, může je připsat do poznámky a administrátor vybere vhodné pokoje ručně, pokud budou volné.

Cena za ubytování je zobrazena napravo od výběru pokojů. Uživatel zadává také, kolik osob bude platit ubytovací a rekreační poplatky. Počet těchto osob nesmí být větší než celkový počet ubytovaných hostů.



Obrázek 12 - Náhled obrazovky s výběrem ubytování

Výběr stravování

Uživatel vybírá, jaký typ stravy a pro kolik osob bude požadovat. Kontrolováno je také to, jestli počet strávníků není větší než počet ubytovaných osob. Celková cena za stravování je zobrazena v pravé části stránky.

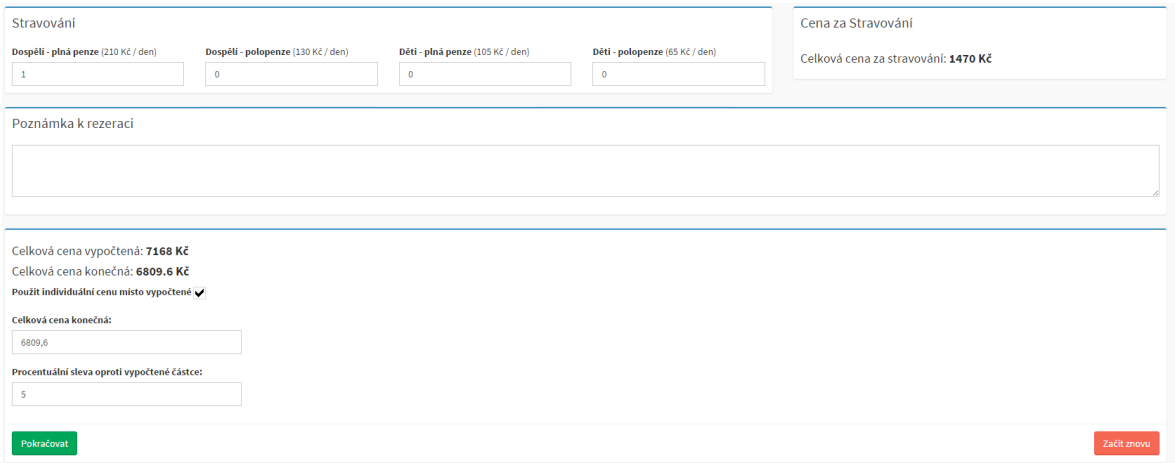
Při každé akci, jako je výběr pokoje, změna doby pobytu apod., je kontrolována dostupnost pokojů a jsou přepočítány ceny. Uživatel má tak neustále přehled, kolik bude jeho pobyt stát, jaké pokoje má vybrány a kolik volných pokojů ještě zbývá. Pokud v průběhu výběru pokojů dokončí rezervaci stejných pokojů jiný uživatel, bude uživatel, který ještě rezervaci nedokončil, na tuto skutečnost upozorněn a pokoje již nebude moci vybrat.

V backendové části může administrátor také měnit konečnou cenu. Zaškrtnutím „*Použít individuální cenu místo vypočtené*“ bude ignorována vypočtená cena rezervace a místo toho bude použita cena nastavená administrátorem. Administrátor může konečnou cenu změnit dvěma způsoby:

1. Zadat konečnou cenu v Kč
2. Změnit hodnotu v poli „*Procentuální sleva oproti vypočtené částce*“, čímž nastaví konečnou cenu na cenu vypočtenou, sníženou o tuto procentuální slevu.

Po těchto změnách je konečná cena zafixována na hodnotě zadané v poli „*Konečná cena*“ a nemění se při jiných změnách v rezervaci. V poli „*Procentuální sleva oproti*

vypočtené částce“ je však vždy vidět aktuální procentuální rozdíl mezi vypočtenou a konečnou cenou.



Obrázek 13 - Náhled obrazovky s volbou stravování a konečné ceny

4.4.3. Fakturační údaje zákazníka

V této části administrátor vyplňuje fakturační údaje zákazníka.

- E-mail (nepovinné)
- Jméno
- Příjmení
- Ulice
- Číslo popisné
- Poštovní směrovací číslo
- Město
- Stát
- Telefon (nepovinné)
- Název organizace (nepovinné)
- IČO (nepovinné)
- DIČ (nepovinné)

V horní části stránky je rolovací nabídka, která obsahuje registrované zákazníky. Pokud je zákazník již registrován, může ho administrátor vybrat a načíst do aktuální rezervace jeho registrační údaje. Po načtení do formuláře může tyto údaje dále upravovat.

Provedené změny nijak neovlivní registrační údaje zákazníka, ale pouze fakturační údaje vybrané rezervace.

4.4.4. Shrnutí rezervace

Pro přístup do této části musí být u vytvářené rezervace vybrány pokoje a fakturační údaje. Tato část slouží k zobrazení detailů rezervace a také ke kontrole údajů vyplněných v předchozích krocích. Obsahuje fakturační údaje, uveden je rozpis rezervovaných pokojů, poplatky, typ a ceny stravování, včetně podrobnějšího rozpisu účtovaných položek. Protože tato stránka obsahuje pro administrátora rychlý přehled o rezervaci, je také optimalizována k tisku.

Až v tomto kroku dochází k ukládání nově vytvářené rezervace do databáze. Dokud uživatel neklikne na tlačítko uložit, jsou informace o rezervaci uloženy v session. Po uložení do databáze je rezervace vytvořena a získává vlastní identifikační číslo.

Po uložení do databáze má administrátor možnost vygenerovat fakturu. Před samotným vygenerováním se zobrazí informační okno, do kterého administrátor zadá číslo faktury. Toto číslo musí být jedinečné napříč všemi rezervacemi. Po potvrzení je faktura uložena do PDF souboru na server. Administrátor si ji může zobrazit a poté stáhnout nebo vytisknout. Pokud je rezervace editována, je potřeba fakturu vygenerovat znovu a tím aktualizovat PDF soubor.

Pokud již existuje faktura, je k ní možné vytvořit dobropis. Po kliknutí na tlačítko „*Dobropis*“ si může administrátor nechat zobrazit nebo smazat existující dobropis. Také může vytvořit nový nebo přepsat stávající. Předtím musí zadat, kolik procent bude vráceno z celkové ceny rezervace. Procenta se přepočítají na koruny a částka, která je označena jako cena dobropisu, je uložena k rezervaci. Vygenerovaný dobropis je uložen v PDF dokumentu na serveru.

Jestliže je součástí rezervace také stravování, může administrátor tisknout stravenky. Po kliknutí na „Vytisknou stravenky“ se na pozadí zavolá stránka pro generování stravenek a je vyvoláno dialogové okno tisku z prohlížeče.

4.4.5. Stav rezervace

Každá z uložených rezervací má jeden ze stavů:

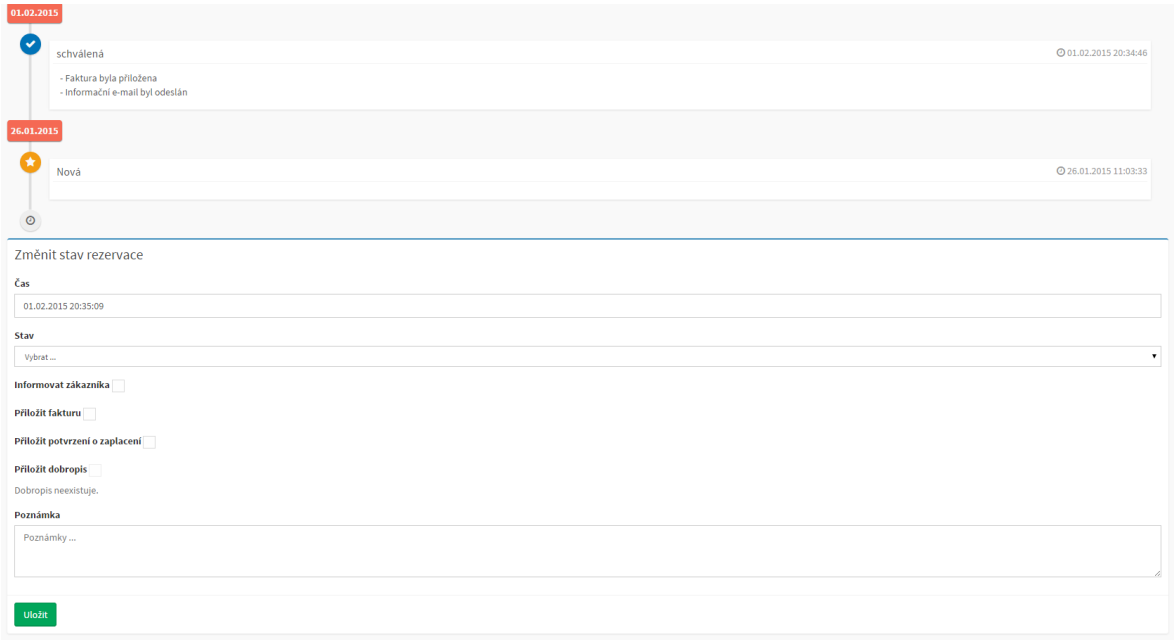
- nová – zákazníkem vytvořená rezervace
- potvrzená – rezervace vytvořená administrátorem nebo rezervace vytvořená zákazníkem a potvrzená administrátorem
- zaplacená – administrátor již potvrdil, že vyfakturovaná částka byla zaplacena
- stornovaná – zrušená administrátorem

Pokud vytvoří rezervaci zákazník, musí být nejprve zkontrolována administrátorem, ten může například reagovat na požadavky, které zadal zákazník do poznámky při vytváření rezervace, nebo změnit vybrané pokoje, pokud automatický výběr z frontendu nebyl optimální. Poté administrátor vygeneruje fakturu a může změnit stav rezervace na potvrzená.

Při přidání stavu administrátor vyplňuje pole:

- Čas změny – automaticky je předvyplněn aktuální čas, ale může být změněn
- Stav
- Poznámka – nepovinné interní poznámky ke změně stavu

Pokud je vyplněn kontaktní e-mail na zákazníka, může administrátor zaškrtnout také políčko „*Informovat zákazníka*“. Pokud tak učiní, bude současně se změnou rezervace zaslán zákazníkovi informační e-mail. Také může zaškrtnutím dalších polí vybrat, jaké dokumenty budou k e-mailu přiloženy. Na výběr má: fakturu, potvrzení o zaplacení a dobropis. Dobropis a faktura musí být před odesláním již vytvořeny na serveru, potvrzení o zaplacení pobytu se na server neukládá a generuje se na vyžádání. Po výběru stavu pomocí roletkového menu jsou vždy vybrány dokumenty, které se budou posílat podle předpřipravené šablony. Například při výběru stavu „*potvrzená*“ je vybráno *„zaslat fakturu“*.

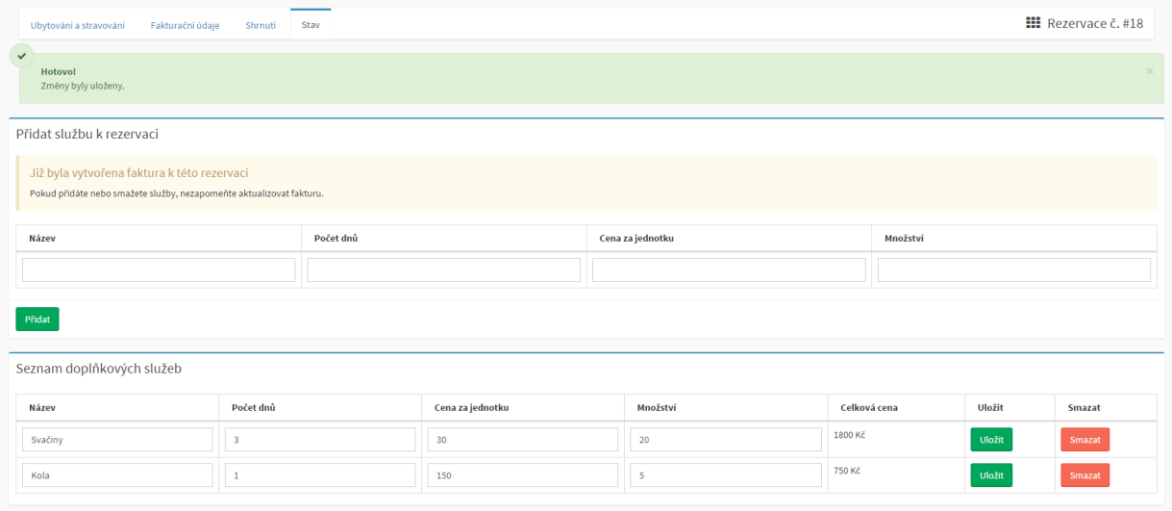


Obrázek 14 - Náhled obrazovky s editací stavu rezervace

4.4.6. Doplňkové služby

V některých případech jsou kromě ubytování poskytovány ještě další služby, které jsou individuální a o jejichž existenci není zatím zmínka ani na webových stránkách. Jsou určeny především pro větší zákazníky, jako například letní tábory. Jedná se kupříkladu o přípravu balíčků se svačinami nebo o pronájem sálu. Jelikož o těchto službách jedná zákazník přímo se zaměstnancem, není možné, aby je sám zákazník přidával během vytvoření rezervace. Přidávat služby může jen administrátor přes modul doplňkové služby. K rezervaci může přidat libovolné množství služeb. U každé z nich musí vyplnit:

- Název
- Počet dnů poskytování služby
- Cenu za jednotku
- Množství za den



Obrázek 15 - Náhled obrazovky s doplňkovými službami

4.4.7. Kalendář

Tento modul slouží ke grafickému zobrazení rezervací a volných pokojů v zadaném období. Jako výchozí období v administračním režimu je měsíc od aktuálního dne. Jiné období je možné jednoduše vybrat pomocí pole „Doba pobytu“, které zobrazí výběrový formulář javaScriptového pluginu „daterangepicker“.

Kalendář je tvořen tabulkou, ve které je zobrazeno, kolik volných pokojů je ve skupině pokojů určitý den. V prvním sloupci jsou názvy skupin pokojů, tento sloupec je pevně umístěn v levé části tabulky a je tudíž vždy vidět. Zbytek tabulky je opatřen horizontálním posuvníkem. Tabulka tak nebude přesahovat rozměry stránky bez ohledu na to, jak dlouhé období bude vybráno. Uživatel se může ve vybraném časovém rozsahu pohybovat pomocí lišty pod tabulkou.

Některé buňky jsou rozděleny na půl, a to vždy v den, kdy rezervace začíná a končí. Důvodem je skutečnost, že ubytování začíná vždy odpoledne prvního dne a pokoje musí být vyklizeny dopoledne v den odjezdu. Mohou tedy vzniknout dvě rezervace na stejný den a pokoj, ale pobyty se nesmějí překrývat.

Dny, kdy není ve skupině žádný volný pokoj, jsou označeny červeným pozadím. Dny, kdy je alespoň jeden pokoj volný, mají zelené pozadí. Pro období, kdy je možnost nových rezervací zablokována, je pozadí modré.

Pokud chce uživatel zobrazit, jaké konkrétní pokoje jsou obsazeny, stačí kliknout na skupinu pokojů v prvním sloupci. Po klinutí se zobrazí v řádcích pod danou skupinou všechny pokoje, které do ní spadají. Po dalším kliknutí na skupiny se pokoje ve skupině skryjí. Podle potřeby je možné zobrazit nebo skrýt pokoje ve všech skupinách kliknutím na jedno z tlačítek v horní části. Rozbalování a sbalování skupin je řešeno pomocí javaScriptu, identifikátory rozbalených skupiny jsou navíc ukládány do cookies[31]. Při obnovení stránky nebo příchodu z jiné stránky jsou cookies prohledány a automaticky jsou rozbaleny ty skupiny, které byly rozbaleny naposledy.

Obsazenost pokojů je opět barevně vyznačena, podobně jako u skupin. Místo počtu volných pokojů je zde uvedeno, kolik bude v pokoji hostů. Pomocí ikony je také označeno, v jakém stavu se rezervace aktuálně nachází. Po najetí na buňku se zobrazí popisek se jménem, příjmením, případně názvem organizace zákazníka. Kliknutím na odkaz v buňce se přejde na souhrnné informace o rezervaci.

Obrázek 16 - Náhled obrazovky s kalendářem

4.4.8. Statistika

Modul statistika slouží k vytvoření podkladových dat pro Český statistický úřad. Uživatel může zvolit, pro který rok chce statistiku zobrazit. V tabulce jsou pak měsíční data.

Počet osob: součet osob, které se ubytovaly daný měsíc; pokud pobyt zasahuje do dalšího měsíce, je započítán jen do měsíce, kdy pobyt začal.

[31] Cookie - v protokolu HTTP označuje data, která server pošle prohlížeči a ten je uloží na počítač uživatele. Při každé další návštěvě téhož serveru pak prohlížeč tato data posílá zpět serveru.

Počet přenocování: celkový počet přenocování hostů ubytovaných v zařízení ve sledovaném období. Výpočet se provádí tak, že se vyberou všechny rezervace ve sledovaném období. U každé rezervace se vypočítá počet přenocování, který se vynásobí počtem hostů v rezervaci. Konečnou hodnotu ukazatele získáme součtem vypočtených hodnot pro vybrané rezervace.

Pokojodny: Termín *pokojoden* vyjadřuje jedno přenocování jedné či více osob na jednom pokoji, což znamená, že se jedná o údaj nezávislý na počtu osob v pokoji. *Pokojodny* se pro určité období vypočítají tak, že se z každé rezervace vybere jen počet nocí spadajících do tohoto období. Počet nocí se vynásobí počtem pokojů v rezervaci. Tím získáme *pokojodny* pro jednu rezervaci. *Pokojodny* pro celé ubytovací zařízení získáme součtem *pokojodnů* všech rezervací v období.

Pod tabulkou je sloupcový graf, který zobrazuje celou situaci během roku v grafické podobě.

4.4.9. Blokování rezervací

V průběhu roku mohou být období, kdy je ubytovací středisko zavřené. Zejména je to mimo sezónu, ale mohou nastat další situace, kdy bude potřeba zajistit, aby bylo na určité období zakázáno vytvářet rezervace. K tomuto účelu slouží modul „Blokování rezervací“.

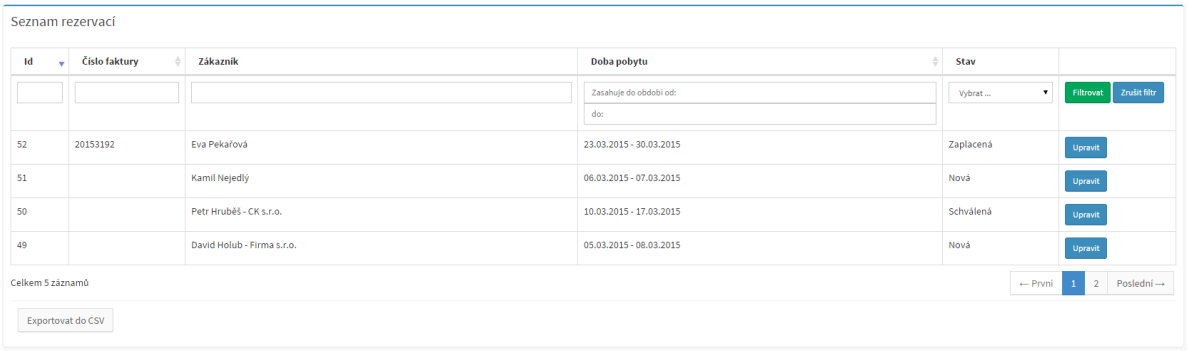
Administrátor vybere pomocí formuláře pro výběr rozsahu počáteční a koncový den a měsíc blokování rezervace. Blokované rozmezí nezahrnuje roky, platí tedy pro všechny roky a není nutné ho každý rok měnit, pokud se doba uzavření nezmění. Jediným omezením je, že období blokování nemůže zasahovat do více let. Pokud je to potřeba, je možné ho rozdělit na dvě období. Jednotlivé blokace se můžou bez problémů překrývat.

Pod formulářem pro přidávání je v tabulce seznam období se zablokovanými rezervacemi, úpravy se provádějí přímo na této stránce - změnou rozsahu v příslušném období a kliknutím na tlačítko „Uložit“. Nepotřebné blokace je možné smazat.

4.4.10. Přehled rezervací

Tento modul slouží zejména k vyhledávání mezi rezervacemi. Rezervace lze filtrovat vyplněním vstupních polí v záhlaví tabulky s rezervacemi a stisknutím klávesy „enter“ nebo kliknutím na „Filtrovat“. Filtrovat je možné podle identifikačního čísla, čísla faktury,

jména, příjmení, názvu organizace zákazníka, doby začátku a konce pobytu a stavu rezervace. Filtry je možné kombinovat, zobrazeny jsou jen výsledky odpovídající všem filtrovacím podmínkám. Pomocí kliknutí na záhlaví tabulky je možné seřadit rezervace podle id, čísla faktury a počátečního data pobytu. Sestupný nebo vzestupný směr třídění je zobrazen šipkou u třídícího atributu. Nastavení filtru se ukládá do session, proto si uživatel může filtr nastavit dle potřeb a procházet mezi stránkami. Pokud se vrátí na „přehled rezervací“, bude filtr ve stavu, jako když ho uživatel použil naposledy. Vymazat všechny filtrovací pravidla může uživatel kliknutím na *„Zrušit filtr“*.



Obrázek 17 - Náhled obrazovky s přehledem rezervací

4.4.11. Správa administrátorů a zákazníků

Administrátor může editovat zákaznické a administrátorské účty přes moduly „zákazník“ a „uživatel“. Primární obrazovku modulu tvoří přehled registrovaných zákazníků respektive uživatelů, ve kterém může vyhledávat a používat filtry. Administrátor může vytvářet nové účty, editovat a mazat stávající. U administrátorského účtu může nastavit jméno, příjmení, e-mail a heslo. U zákazníka navíc i fakturační a kontaktní údaje. Heslo se zadává vždy dvakrát, aby se minimalizovala možnost překlepu. E-mail slouží mimo jiné také k přihlášení do systému.

4.4.12. Frontendová část

Úvodní strana

Na úvodní straně je zobrazen kalendář s přehledem volných pokojů ve skupinách pokojů. Kalendář je podobný tomu v administraci, pouze není možné zobrazit konkrétní pokoje, protože tato informace není pro zákazníka ve většině případů směrodatná.

Přihlášení a registrace

Zákazník se může registrovat a svůj účet používat ke zrychlení vytváření rezervací. Své údaje v něm může kdykoliv upravovat. Moduly pro přihlášení a registraci nebudou blíže popisovány.

Vytváření rezervace

Vytvoření rezervace probíhá podobně jako v administraci, zde k tomu stačí pouze tři kroky. Svůj postup mezi kroky vidí zákazník na liště v horní části stránky.

Nejdříve zákazník vybírá ubytování a stravování. Oproti administrační části však může vybrat jen skupiny pokojů, konkrétní pokoje mu budou přiděleny automaticky. Logicky také nemůže přidávat k rezervaci žádné slevy.

Větší změna nastává při výběru fakturačních údajů. Pokud je zákazník přihlášen, zobrazí se rovnou předvyplněné údaje z jeho uživatelského účtu. Pokud přihlášen není, má jednu ze tří možností, jak fakturační údaje k rezervaci přiřadit. Výběr možnosti provede ve druhém kroku zaškrtnutím příslušného „radio buttonu“.

1. **Vytvořit jednorázovou rezervaci**
   V tomto případě nemusí být registrován, jen doplní fakturační údaje
2. **Přihlásit se**
   pokud má již vytvořený zákaznický účet, vyplní e-mail a heslo a přihlásí se. Tím se automaticky vyplní fakturační údaje pro tuto rezervaci údaji z jeho uživatelského účtu.
3. **Registrovat se**
   Při výběru této možnosti se zobrazí registrační formulář. Po úspěšné registraci dojde k automatickému přihlášení a načtení fakturačních údajů do rezervace.

Po výběru ubytování a fakturačních údajů je zpřístupněn poslední krok, kde je zobrazen kontrolní souhrn rezervace. Po potvrzení je rezervace uložena do databáze a zákazníkovi je zaslán e-mail s informacemi o vytvořené rezervaci. Stejný e-mail může být také zaslán na další účty, které jsou zadány v modulu „obecné nastavení“ v administraci. Administrátoři jsou díky tomu rychle informováni o nově vzniklých rezervacích.

Obnovení zapomenutého hesla

Pokud registrovaný uživatel zapomene své heslo, může získat přístup ke svému účtu zpátky vytvořením nového hesla. V modulu obnova hesla zadá svůj registrační e-mail a odešle formulář. Pokud existuje účet s tímto e-mailem, je vygenerován náhodný token, který je zaslán v podobě odkazu na zadaný e-mail. Token je také v zašifrované podobě uložen do databáze. Po kliknutí na odkaz z emailu a ověření platnosti tokenu může uživatel zadat nové heslo ke svému účtu. Pokud si uživatel v mezičase na heslo vzpomene, může ho bez obav používat a nemusí heslo měnit. Pro zvýšení zabezpečení je platnost tokenu pro obnovu hesla 24 hodin.

4.5. Uvedení systému do provozu

4.5.1. Nasazení

Sdružení Prosaz nemá vlastní webový server, ale využívá externích webhostingových služeb. Aby byly ušetřeny náklady, byl rezervační systém umístěn na server spolu s internetovými stránkami sdružení.

Internetové stránky využívaly redakční systém Joomla ve verzi 1.0. Podpora této verze skončila již v roce 2009, proto byl připravován přechod na nejnovější verzi redakčního systému Wordpress spojený s celkovou rekonstrukcí webových stránek.

Na serveru bylo původně nainstalováno PHP verze 5.2. Rezervační systém vyžaduje pro správnou funkčnost PHP ve verzi 5.3.7 nebo vyšší. Po komunikaci s administrátorem webhostingu bylo zjištěno, že podpora PHP starších verzí PHP bude brzy ukončena. Proto bylo rozhodnuto, že dojde rovnou k migraci na server s PHP 5.5.

Po odeslání požadavku na změnu verze PHP byl administrátorem webhostingu zpřístupněn nový PHP server, zároveň byla vytvořena nová databáze. Na tento server byla překopírována rozpracovaná verze nových stránek založená na Wordpressu. ze současného serveru zadavatele. Následně byl na server nahrán také rezervační systém. Tímto způsobem byla zachována funkčnost současných stránek a zároveň mohl být otestován rezervační systém na novém serveru. Přístup na nový server vyžaduje úpravy souboru

`host` ve Windows a to do doby než budou změněny DNS záznamy tak, aby z domény prosaz.cz ukazovali na IP adresu nového serveru. Tato změna bude provedena ihned poté, co budou zaměstnanci připraveny nové stránky organizace.

4.5.2. Testování

Ačkoli byla aplikace testována autorem práce i zadavateli v rámci vývoje při každém představení funkčního celku, bylo také nutné překontrolovat kompletní webovou aplikaci. Toto celkové testování probíhalo nejdříve na testovacím serveru a poté i na produkčním serveru, kde je aplikace finálně provozována. Testování probíhalo za použití několika internetových prohlížečů. Prověřen byl celý proces práce se systémem, tak jak ho budou standardně používat uživatelé. Odzkoušeny byly ovšem i scénáře, kdy by se někdo pokoušel systém poškodit nebo zneužít. V rámci testování byl v Excelu vytvořen dokument, v němž bylo popsáno, co v jakém modulu a s jakým výsledkem bylo testováno, případně zda byla provedena oprava. V průběhu testování bylo skutečně nalezeno několik chyb. Většinou se jednalo o chyby drobnějšího rázu, které byly neprodleně opraveny.

Zároveň byla ověřena validita všech HTML výstupů. Kód, který odpovídá standardům, zvyšuje pravděpodobnost, že stránka bude zobrazena správně při užití různých prohlížečů. Testování probíhalo pomocí online služby Markup Validator poskytované konsorciem W3C[32]

[32] World Wide Web Consortium (W3C) - mezinárodní konsorcium, jehož členové společně s veřejností vyvíjejí webové standardy pro World Wide Web

5. Závěr

Cílem práce bylo navrhnout, implementovat a uvést do provozu nový rezervační systém pobytového střediska. Zavedení systému mělo dva hlavní cíle. Prvním cílem bylo umožnit zákazníkům vytvářet rezervace ubytování online, tedy kdykoli, bez kontaktu s pracovníky Prosazu. Druhým důležitým cílem bylo ulehčit zaměstnancům proces vytváření a správy rezervací.

Celý systém byl vytvářen v těsné spolupráci se zaměstnanci organizace, kteří ho budou v praxi používat. Tento postup se ukázal jako velice efektivní. Zaměstnanci ochotně spolupracovali a jejich poznámky a nápady byly při vývoji velmi užitečné. Díky časté komunikaci byl systém vytvářen velmi plynule a nedocházelo k nutnosti přepracování modulů, které by bylo způsobeno nesprávným pochopením požadavků. Ve výsledku se podařilo vytvořit systém, který odpovídá praktickým potřebám uživatelů.

Systém byl naprogramován v jazyce PHP s využitím relační databáze MySQL, jakožto velmi rozšířených platforem široce podporovaných webhostingy. Rezervační systém byl postaven na moderní architektuře MVC, která je v současné době hojně využívána k tvorbě webových stránek a aplikací. Díky této architektuře bude usnadněno případné budoucí rozšiřování systému. Zaměstnancům společnosti byla vysvětlena funkce šablon ve vrstvě pohledu. To jim umožní samostatně provádět některé menší změny, přestože nemají příliš velké znalosti programování, ale ovládají HTML. Základním stavebním kamenem systému je vlastní framework, který byl v průběhu práce vylepšen a rozšířen. Tento framework může být dále použit při vytváření dalších webových projektů. Použití frameworku výrazně urychlilo programování a redukovalo redundantní části kódu. Díky modulárnosti je možné aplikaci dále rozšiřovat podle požadavků, které mohou vzniknout při reálném provozu aplikace.

Při tvorbě grafického rozhraní byl brán ohled na trend používání mobilních zařízení při práci s internetem. Díky použití frontendového frameworku Bootstrap jsou stránky optimalizovány i pro zobrazení na těchto zařízeních. Kromě toho se při zvětšení textu pomocí prohlížeče mění automaticky struktura stránky, tato vlastnost přispívá k pohodlnému ovládání aplikace i pro uživatele, kteří mají problémy se zrakem.

Pokud se při reálném provozu objeví nějaké chyby, autor práce se zavázal, že sjedná nápravu. Pokud by v budoucnu vznikly požadavky na další funkcionalitu, bude autor práce po domluvě rád spolupracovat na rozšíření aplikace.

Zdrojové kódy rezervačního systému jsou uloženy na DVD, které je přiloženo k této práci. Pokyny pro instalaci aplikace jsou uvedeny v příloze 6.

6. Seznam použitých zdrojů

1. History of PHP. *PHP* [online]. [cit. 2015-03-05]. Dostupné z: http://php.net/manual/en/history.php.php

2. BRÁZA, J. *PHP 5 začínáme programovat*. Praha: Grada, 2005, 244 s. ISBN 80-247-1146-X.

3. JANOVSKÝ, D. Jak psát web. *Soubor.htaccess* [online]. [cit. 2014-19-12]. Dostupné z: http://www.jakpsatweb.cz/server/htaccess.html

4. GRUDL, D. Nette Framework: MVC & MVP. *zdroják.cz* [online]. 2009 [cit. 2014-12-02]. Dostupné z: http://www.zdrojak.cz/clanky/nette-framework-mvc--mvp/

5. BERNARD, B. Úvod do architektury MVC. *zdroják.cz* [online]. 2009 [cit. 2014-12-05]. Dostupné z: http://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

6. BERNARD, B. Prezentační vzory z rodiny MVC. *zdroják.cz* [online]. [cit. 2014-12-05]. Dostupné z: http://www.zdrojak.cz/clanky/prezentacni-vzory-zrodiny-mvc/

7. ČÁPKA, D. ITnetwork.cz. *MVC architektura* [online]. [cit. 2015-01-12]. Dostupné z: http://www.itnetwork.cz/mvc-architektura-navrhovy-vzor

8. E.WIEGERS, K. *Požadavky na software*. Brno: Computer Press , a.s. 2008, 448 s. ISBN 978-80-251-1877-1.

9. BUCHALCEVOVÁ, A. *Metodiky vývoje a údržby informačních systémů: kategorizace, agilní metodiky, vzory pro návrh …*. Praha: Grada, 2005, 163 s. ISBN 80-247-1075-7.

10. KADLEC, V. *Agilní programování: Metodiky efektivního vývoje software*. Brno: Computer Press, 2004, 278 s. ISBN 80-251-0342-0.

11. *Manifesto for Agile Software Development* [online]. 2001 [cit. 2014-12-12]. Dostupné z: http://agilemanifesto.org

12. KNESL, J. Agilní vývoj: Scrum. *Zdroják.cz* [online]. 2009 [cit. 2015-01-06]. Dostupné z: http://www.zdrojak.cz/clanky/agilni-vyvoj-scrum/

13. SCHWABER , K. a SUTHERLAND. Průvodce Scrumem. *ScrumGuides* [online]. 2013 [cit. 2015-02-03]. Dostupné z: http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-CS.pdf#zoom=100

14. KNIBERG, H. *Scrum and XP from the Trenches*. 2007. C4Media, Publisher of InfoQ.com. ISBN 978-1-4303-2264-1. Dostupné také z: http://wwwis.win.tue.nl/2R690/doc/ScrumAndXpFromTheTrenchesonline07-31.pdf

15. PHPMailer. *GitHub* [online]. [cit. 2014-10-25]. Dostupné z: https://github.com/PHPMailer/PHPMailer/

16. StatCounter Global Stats. *Browser, OS, Search Engine including Mobile Usage Share* [online]. [cit. 5-01-2015]. Dostupné z: http://gs.statcounter.com/#browser-CZ-monthly-201312-201412

17. *Free Admin Template For Bootstrap - AdminLTE* [online]. [cit. 2014-11-02]. Dostupné z: http://almsaeedstudio.com/

18. *Google Fonts* [online]. [cit. 2014-12-09]. Dostupné z: https://www.google.com/fonts

19. Apache. *Apache Module mod_rewrite* [online]. [cit. 2014-08-07]. Dostupné z: http://httpd.apache.org/docs/current/mod/mod_rewrite.html

20. Github. *Apache HTTP server boilerplate configs* [online]. [cit. 2015-02-03]. Dostupné z: https://github.com/h5bp/server-configs-apache

21. Smarty. *PHP Template Engine* [online]. [cit. 2014-11-02]. Dostupné z: http://www.smarty.net/

22. GitHub. *dompdf* [online]. [cit. 2014-11-06]. Dostupné z: https://github.com/dompdf/dompdf

23. PHP triky. *Ukládání hesel bezpečně* [online]. [cit. 2015-02-03]. Dostupné z: http://php.vrana.cz/ukladani-hesel-bezpecne.php

24. GitHub. *password_compat* [online]. [cit. 2015-05-02]. Dostupné z: https://github.com/ircmaxell/password_compat

25. Soom. *Hashování hesel pomocí PHP 5.5 Password Hashing API* [online]. [cit. 2015-02-03]. Dostupné z: http://www.soom.cz/clanky/1121--Hashovani-hesel-pomoci-PHP-55-Password-Hashing-API

26. Soom. *Cross Site Request Forgery* [online]. [cit. 2015-02-04]. Dostupné z: http://www.soom.cz/clanky/484--Cross-Site-Request-Forgery

27. VRÁNA, J. PHP triky. *Automatická obrana proti CSRF* [online]. 3. 3. 2015. Dostupné také z: http://php.vrana.cz/automaticka-obrana-proti-csrf.php

28. VRÁNA, J. *1001 tipů a triků pro PHP*. Brno: Computer Press, 2010, 456 s. ISBN 978-80-251-2940-1.

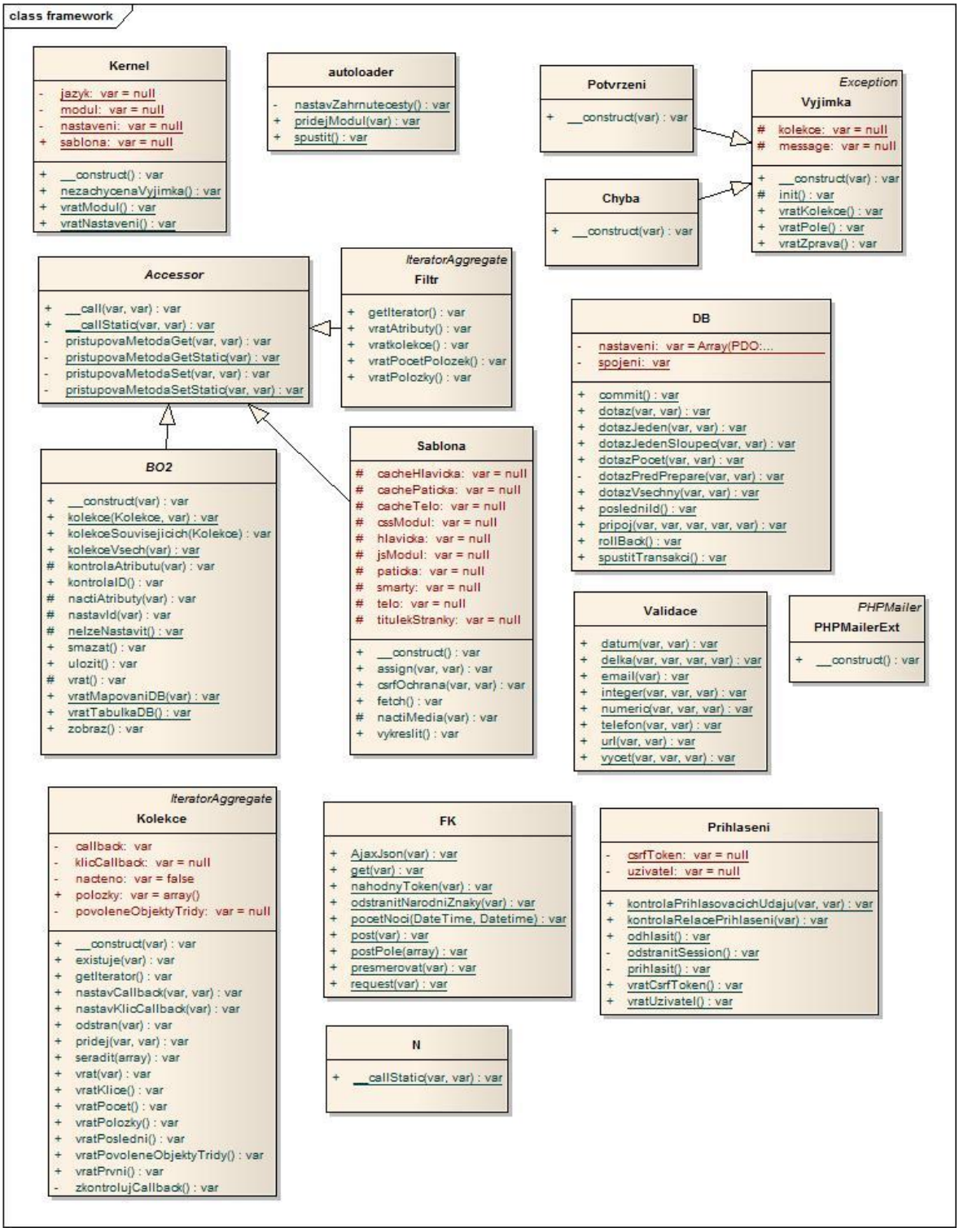
29. Google Developers. *Optimizing encoding and transfer size of text-based assets* [online]. [cit. 2015-02-03]. Dostupné z: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/optimize-encoding-and-transfer#text-compression-with-gzip

30. Yahoo Developer Network. *Best Practices for Speeding Up Your Web Site* [online]. [cit. 2015-02-02]. Dostupné z: https://developer.yahoo.com/performance/rules.html

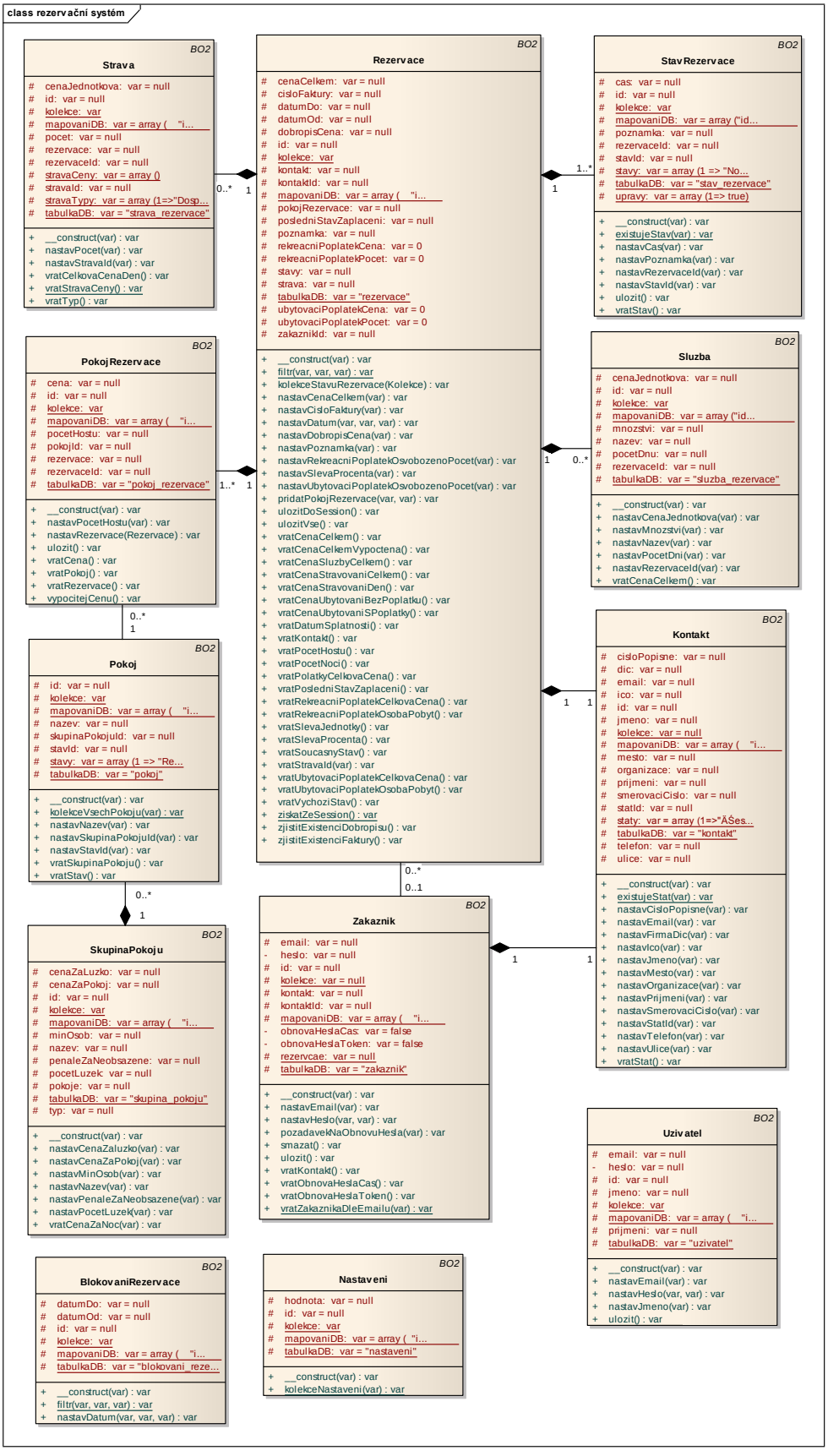
31. LECKY-THOMPSON, E. S. D. NOWICKI a T. MYER. *Professional PHP 6*. Wrox programmer to programmer. Indianapolis: Wiley, 2009, 703 s. ISBN 978-0-470-39509-7.

7. Přílohy

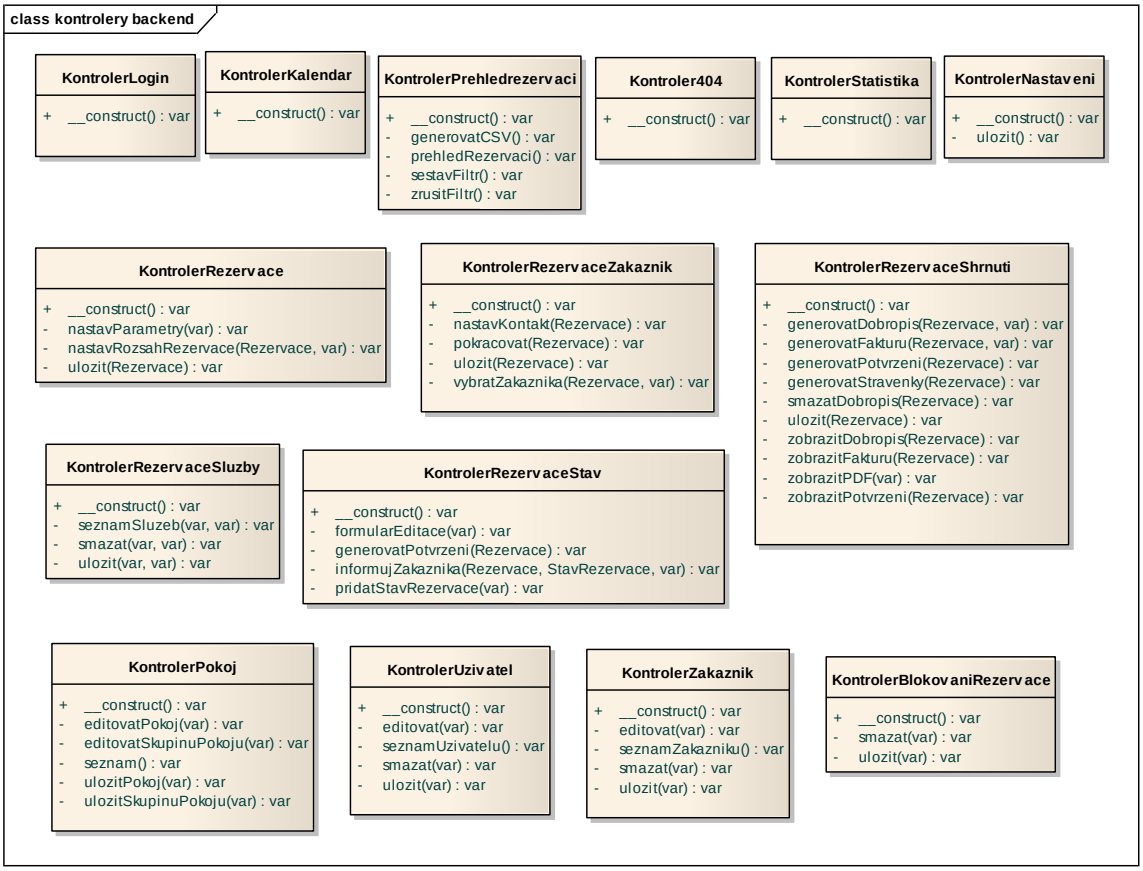
Příloha 1 - Diagram tříd frameworku



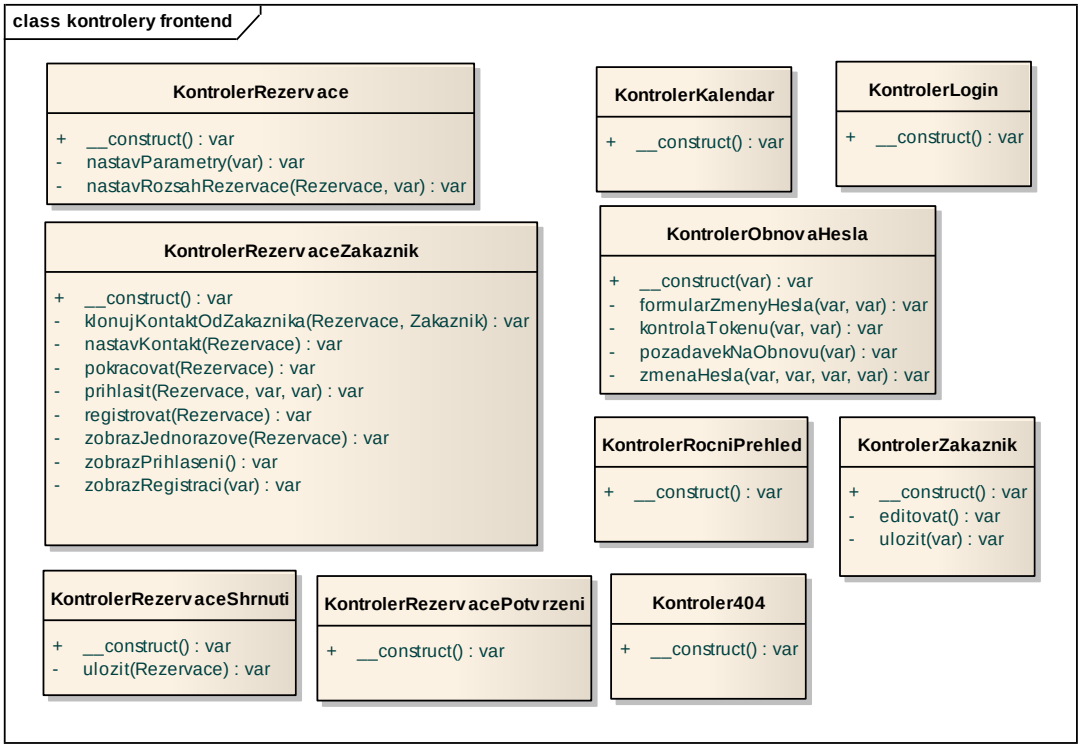
Příloha 2 - Implementační diagram doménových tříd



Příloha 3 - Diagram tříd kontrolerů backendové části



Příloha 4 - Diagram tříd kontrolerů frontendové části



Příloha 6 - Instalace rezervačního systému

Instalace a spuštění ukázkové verze pomocí XAMPP

Tato verze obsahuje již vytvořené rezervace a další prvky sloužící k demonstraci funkcionality. Pro spuštění rezervačního systému prostřednictvím přenositelné verze XAMPP postupujte dle následujícího návodu.

1. Zkopírujte složku `xampp` z přiloženého DVD do **kořenového adresáře** vašeho počítače například `E:\xampp` nebo `W:\xampp`

2. Ve zkopírované složce vyhledejte soubor `xampp-control.exe` a spusťte ho. Otevře se kontrolní panel XAMPP.

3. V kontrolním panelu zapněte moduly Apache a MySQL kliknutím na tlačítko „Start“ u příslušného modulu. Spuštění modulu je indikováno zeleným podbarvením názvu modulu.

Pokud vše proběhlo v pořádku, měla by se po zadání adresy `http://localhost/prosaz` do internetového prohlížeče zobrazit frontendová část aplikace.

Administrační část se nalézá na adrese `http://localhost/prosaz/admin`
Přihlašovací údaje do administrační části jsou:
e-mail: admin1@prosaz.cz
heslo: t!tan1

phpMyAdmin – webové rozhraní pro administraci databáze
`http://localhost/phpmyadmin/`
jméno: root
heslo: S5Ka98WyJScX6Ef

Poznámka: e-maily nejsou z této verze odesílány, ale ukládány v textové podobě do složky `xampp\mailoutput`.

Instalace na existující webový server

Aplikace může být také nakopírována na již existující Apache server. V tom případě je na něj potřeba překopírovat soubory z adresáře `xampp\htdocs\prosaz`, který se nalézá na přiloženém DVD a naimportovat do MySQL databáze tabulky a data potřebná pro chod rezervačního systému. SQL soubory jsou umístěné ve složce `sql`. Chcete-li používat demoverzi s předvyplněnými rezervacemi, naimportujte soubor `prosaz_sandbox.sql`. Pokud chcete použít čistou produkční verzi, naimportujte soubor `prosaz.sql`.

Podle nastavení serveru a umístění rezervačního systému v souborové struktuře bude možná nutné editovat soubory: `.htaccess` a `admin/.htaccess`, konkrétně přenastavit `RewriteBase`. Dále nastavit cesty v `_conf/konfigurace.php` a `admin/_conf/konfigurace.php`. Přístupové údaje k MySQL databázi se nastavují v souboru `admin/_conf/globalniKonfigurace.php`.

Příloha 7 - Příloha na DVD

Na přiloženém DVD je ve složce `xampp` umístěna portable verze XAMPP pro Windows. XAMPP obsahuje Apache server PHP a MySQL.

Zdrojové kódy rezervačního systému jsou ve složce `xampp\htdocs\prosaz`.
Ve složce `sql` jsou soubory pro vytvoření databázové struktury.