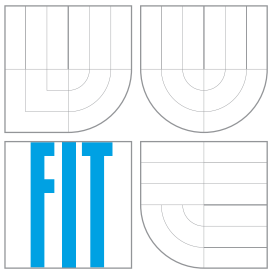
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

PRÁCE S QR KÓDY VE WEBOVÝCH APLIKACÍCH

WORK WITH QR CODES IN WEB APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL MELICHAR

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. VLADIMÍR BARTÍK, Ph.D.

BRNO 2016

Zadání bakalářské práce/18682/2015/xmelic17

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2015/2016

Zadání bakalářské práce

Řešitel: **Melichar Michal**
Obor: Informační technologie
Téma: **Práce s QR kódy ve webových aplikacích**
Work with QR Codes in Web Applications
Kategorie: Web

Pokyny:

1. Seznamte se s principy tvorby webových aplikací.
2. Prostudujte dostupné knihovny pro práci s QR kódy ve webových aplikacích a srovnejte jejich vlastnosti, popř. výhody a nevýhody.
3. Analyzujte požadavky na informační systém vinařské firmy, v rámci něhož bude podpora pro generování a zobrazování QR kódů implementována.
4. Navržený systém implementujte jako webovou aplikaci a ověřte funkčnost na vhodně zvoleném vzorku dat.
5. Zhodnoťte dosažené výsledky a další možné pokračování tohoto projektu.

Literatura:

- Waters, J.: QR Codes for Dummies, John Willey and Sons, 2012.
- Gutmans, A., Rethans, D., Bakken, S.: Mistrovství v PHP 5, Computer Press, 2012.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1-3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bartík Vladimír, Ing., Ph.D.**, UIFS FIT VUT
Datum zadání: 1. listopadu 2015
Datum odevzdání: 18. května 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Bakalářská práce se zabývá problematikou návrhu a realizace informačního systému vinařské firmy. Práce je rozdělena na kapitoly Základní pojmy, Práce s QR kódy, Návrh informačního systému a Implementace. Kapitoly Základní pojmy a Práce s QR kódy obsahují základní informace o QR kódech a jejich využití v praxi, přehled nástrojů pomocí kterých můžeme generovat a číst QR kódy, dále je zde přehled základních vlastností frameworků Nette a Bootstrap. Kapitoly Návrh informačního systému a Implementace se věnují návrhu a realizace aplikace podle analýzy požadavků dané firmy. K analýze a modelování byly využity diagramy ER a Use Case. Jako databázový systém bylo zvoleno MySQL, pro funkční rozhraní byl použit Nette Framework a pro design a responzivní rozhraní Bootstrap. Součástí práce jsou zdrojové soubory a návrhové diagramy.

Abstract

The bachelor thesis deals with the problem of designing and implementation of an information system in a wine producing company. The thesis is divided into four chapters, namely a Basic concepts, a Work with QR codes, a Design of information system and an Implementation. The chapters Basic concepts and Work with QR codes contain fundamental information on QR codes and their use in practice, including the overview of tools suitable for generating and reading of QR codes, as well as we can find in this part the overview of the basic features of the Nette and Bootstrap frameworks. The chapters Design of information system and Implementation are dedicated to the design and realization (implementation) of an application based on the analysis of needs of the chosen company. The diagrams ER and Use Case have been used for the analysis and modelling. MySQL has been used as the database system, Nette Framework has been used for the functional interface, as well as Bootstrap has been used for the responsive interface. The source files and project diagrams are an integral part of the thesis.

Klíčová slova

webová aplikace, informační systém, QR kódy, Nette Framework, Bootstrap, PHP, MySQL, Live Validace

Keywords

web application, information system, QR codes, Nette Framework, Bootstrap, PHP, MySQL, Live Validation

Citace

MELICHAR, Michal. *Práce s QR kódy ve webových aplikacích.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Bartík Vladimír.

Práce s QR kódy ve webových aplikacích

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Vladimíra Bartíka, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal. Veškeré údaje, kromě veřejně dostupných, jsou smyšlené a neidentifikují např. konkretní osobu či vozidlo.

.....
Michal Melichar
16. května 2016

Poděkování

Tímto bych chtěl poděkovat Ing. Vladimíru Bartíkovi, Ph.D. za velmi cenné rady a připomínky při vypracování bakalářské práce. Dále Ing. Karlovi Průšovi za možnost vytvoření informačního systému a v neposlední řadě také všem zaměstnancům firmy Vinicola, s.r.o., kteří svými podněty přispěli k finální podobě informačního systému.

Obsah

Seznam obrázků

Seznam zdrojových kódů

Kapitola 1

Úvod

QR kódy jsou součástí našeho moderního způsobu života. Najdeme je všude kolem nás. Na obalech výrobků, na reklamních letácích, anebo ve webových prezentacích. V dnešní době má téměř každý u sebe chytrý mobilní telefon s fotoaparátem, který nám umožňuje naskenovat QR kód. Proto je tento způsob poskytování dat velmi oblíbený a svým způsobem také moderní. QR kód nám umožňuje pouhým naskenováním zobrazit požadované informace, které nám jeho autor chtěl sdělit. Může se jednat o URL adresu, e-mailovou adresu, telefonní číslo, anebo také třeba pokyny pro online platbu. Využití je opravdu rozmanité a jednoduché pro každého.

Dříve se zcela běžně ukládaly informace v rozsáhlých kartotékách, dnes se k tomu používají stále častěji databáze. Nechat spravovat záznamy počítačem je mnohem efektivnější a rychlejší, než stejnou práci svěřit lidem. Uložené záznamy v databázích musíme ale určitým způsobem dostat k uživatelům. A v tuto chvíli přicházejí ke slovu informační systémy, které nám dovolují zrychlit vyhledání požadované informace, přehledně ji zobrazit a dále s ní pracovat. Dále se nám otevírá také možnost mít všechny údaje, data a soubory pohromadě v jedné službě. Velmi často tak informační systémy můžeme vidět formou webové aplikace. Takový informační systém je poskytován z webového serveru přes počítačovou síť Internet případně Intranet. Mezi hlavní výhody takového řešení patří dostupnost informačního systému na jakékoliv běžně používané platformě, jelikož webový prohlížeč je téměř všude přítomný. Dále nám odpadá nutnost konfigurace zařízení uživatelů, kterých může být pouze několik ale také několik desítek či tisíc.

Kapitola 2

Základní pojmy

V této kapitole jsou uvedeny základní pojmy a technologie, které jsou využity při návrhu a implementaci informačního systému. Nachází se zde úvod k teorii informačního systému, základní popis jazyka UML, E-R diagramu, architektury MVC, frameworků Nette a Bootstrap.

2.1 Informační systém

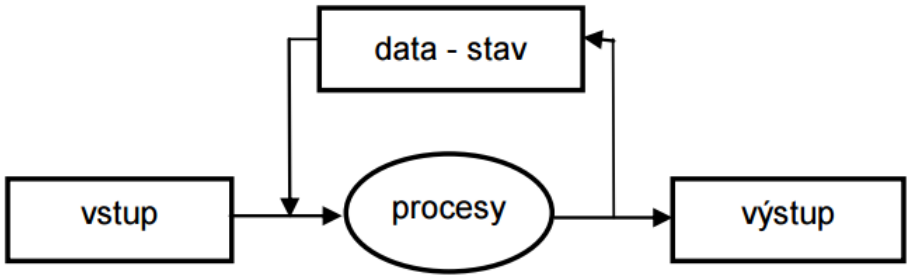
Pro začátek je vhodné si definovat, co to vlastně informační systém je a co si pod tímto pojmem představit. Pojem se skládá ze dvou slov, a to informační a systém.

Význam slova informace se dá interpretovat několika způsoby podle toho, ze kterého odvětví se na toto slovo díváme. V oblasti informačních technologií se dá říci, že informace jsou interpretovaná data, která mají určitý význam. Datům význam nepřisuzuje informační systém, ale uživatel až v době jejich použití. [21]

Systém můžeme chápat jako množinu prvků (definovaných za určitým účelem) a vazeb mezi nimi, které jsou definovány na nosiči. Jako celek vykazují určitou funkci eventuálně chování. Nosičem rozumíme množinu prvků daného systému, které mají mezi sebou vazby. Jednotlivé prvky systému nazýváme zdroje. [21]

Podle jiné definice se dá informační systém chápat jako soubor lidí, technických prostředků a metod, které mají za cíl sběr, přenos, zpracování a uchování dat. Tohoto se využívá při prezentaci informací pro potřeby uživatelů, kteří jsou aktivní v systémech řízení. [25]

Na obrázku 2.1 můžeme vidět schéma informačního systému.



Obrázek 2.1: Schéma informačního systému. Převzato z [21].

Informační systém má podle obrázku 2.1 několik stavů. Stavy Vstup a Výstup představují data, která přicházejí do systému ke zpracování a data, nad kterými systém provedl určité operace a transformace. [21]

- **Data** v rámci informačního systému uchovávají stav systému. Tím rozumíme, že stavem systému jsou vlastně hodnoty zdrojů daného systému.
- **Procesy** provádějí změny v informačním systému nejčastěji pomocí transakcí.

2.2 Webová aplikace

Webová aplikace je druh aplikace, kterou poskytuje webový server přes počítačovou síť Internet nebo Intranet. Intranet je vlastně obdoba sítě Internet, pouze s tím rozdílem, že se jedná o vnitropodnikovou síť. Pro spuštění postačí webový prohlížeč, který je v dnešní době snad samozřejmostí u většiny běžně používaných platforem. Není tedy potřeba instalovat aplikaci ani provádět její nastavení na počítačích uživatelů. Do kategorie webových aplikací můžeme zařadit např.: [19, 2]

- informační systémy,
- internetové obchody,
- e-mailové služby,
- diskuzní fóra,
- chaty,
- inzertní servery.

Velkou výhodou webových aplikací je, že jejich zavedení je velmi jednoduché, dokáží pracovat bez ohledu na použitou platformu případně verzi operačního systému. Usnadnila se také distribuce nových verzí, kdy stačí pouze novou verzi aplikace nahrát na server. Podpora pro uživatele je také mnohem snadnější a levnější. Technici nemusí jezdit za uživateli, aby jim pomohli s instalací nebo nastavením aplikace. V případě chyby není potřeba distribuovat novou verzi mezi všechny uživatele, ale chyba může být opravená pouze na serveru.

Naopak mezi nevýhody můžeme uvést závislost na internetovém připojení, delší odezvu aplikace v porovnání s aplikací běžící na uživatelově PC nebo určité odlišné specifikace webových prohlížečů. Dále musíme pamatovat na možnost webových aplikací využívat Adobe Flash, AJAX, JavaScript, které lze v prohlížeči vypnout. Tím by mohlo dojít k narušení vzhledu stránky nebo její funkčnosti. Dále by webové aplikace měly být optimalizované pro větší spektrum rozlišení z důvodu, že v dnešní době jsou mimo PC také v oblibě smartphony, které mají jinou uhlopříčku než běžné PC nebo notebooky. Při výpadku serveru, na kterém běží daná webová aplikace, dojde také k výpadku přístupu všech klientů. Stejná situace nastane při výpadku připojení k internetu, trochu jiná situace může nastat, pokud se server nachází v intranetu.

2.2.1 Tenký klient

Webové aplikace se od architektury klient/server liší tím, že používají webový prohlížeč jako tzv. tenkého klienta. Tenký klient je tedy počítačový program nebo počítač, který silně závisí

na funkcionalitě jiného počítače a komunikuje s ním pomocí bezstavového protokolu HTTP. Tímto počítačem je právě server. Účelem takového klienta je tedy připojit se na server, odeslat a přijmout data/požadavky/odpovědi a následně je vykreslit uživateli. Veškeré výpočty a operace, které mění např. obsah databáze, se provádějí na straně serveru. Pro úplnost uvádím některé výhody a nevýhody, které plynou z takového řešení: [20]

- + klient nepožaduje výkonný HW,
- + snadná aktualizace, podpora uživatelů, správa,
- + přenositelnost,
- - server vyžaduje mnohem výkonnější HW,
- - možnost práce s aplikací/systém na serveru pouze v režimu online,
- - horší odezva aplikace, která je závislá na parametrech připojení k serveru.

2.3 Jazyk UML

UML (Unified Modeling Language) je „grafický“ jazyk, který se používá při vývoji softwaru. Stal se standardem v oblasti analýzy, vizualizace, návrhu a dokumentace programových systémů. Podporuje také objektově orientovaný přístup. Nabízí standardní způsob zápisu např. pro návrh systému nebo databázové schéma. V dnešní době už většinou neprogramuje jeden programátor celý systém. Je tedy nutná spolupráce více programátorů, a zde se objevuje velká výhoda jazyka UML, která nám usnadňuje implementaci systému i jeho dokumentaci. Dále nám umožňuje přehledně ukázat změnu v zadání se všemi závislostmi, které z toho plynou. Díky standardizaci zápisu návrhu s jazykem UML, si každý programátor v týmu vyloží návrh stejně. Můžeme jazyk UML také využívat při komunikaci se zákazníkem, kdy zjišťujeme, co přesně zákazník požaduje. Je možné v něm vytvářet různé modely systému, které poté podrobíme schvalování, testování nebo přepracování. Až po následném schválení může být daný systém naprogramován v cílovém programovacím jazyce. [26, 22]

2.3.1 Use Case diagram

První fáze v životním cyklu softwaru je specifikace požadavků. Use Case diagram neboli diagram případů užití, slouží k popisu chování systému a jedná se o jeden z diagramů chování, který nalezneme v UML. Účelem diagramu je tedy popsat funkčnost jakou od systému očekáváme z hlediska uživatele, a popisuje nám typy uživatelů, kteří se systémem pracují. Tzn. říká nám co má daný systém umět, ale neříká nám už jakým způsobem toho docílíme. Use Case diagram je tvořen případy užití, aktéry a vztahy mezi nimi. [26, 22, 28]

Use Case

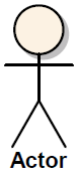
Use Case neboli případy užití představují v systému akce, které může daný aktér provádět. Příkladem může být např. akce smazat, editovat, nebo vytvořit novou položku. Jedná se tedy o jednu funkcionalitu, kterou systém umí poskytnout konkrétnímu aktérovi. Use Case pracuje s určitou mírou abstrakce. Případy užití vycházejí z požadavků zákazníka. Můžeme si všimnout, že máme akci editovat položku, za touto akcí jsou však skryté další kroky, které je potřeba vykonat, aby bylo možné úspěšně provést editaci položky. Můžeme to rozdělit

na kroky, které se provedou před samotnou editací, jedná se např. o ověření přihlášení a role uživatele, nalezení a vypsání dat dané položky. Dále na kroky po dokončení editace (myšleno v rámci uživatelské rozhraní), je potřeba provést např. validaci upravených dat, data zapsat do databáze. Všechny tyto kroky ovšem v Use Case diagramu uvedené už nebudou. Use Case se obvykle znázorňuje elipsou, která obsahuje název akce. Příklad můžeme vidět na obrázku: 2.2.

Obrázek 2.2: Ukázka znázornění Use Case (případu užití).

Actor

Actor neboli aktér představuje uživatele nebo externí systém, který komunikuje s jednotlivými případy užití. Aktérem může být tedy např. ředitel, nebo E-mailový server. Název aktéra nám vyjadřuje, jakou roli má v systému. Aktéra v Use Case diagramu znázorňujeme jako postavu z čar. Jednoduché znázornění můžeme vidět na obrázku: 2.3.

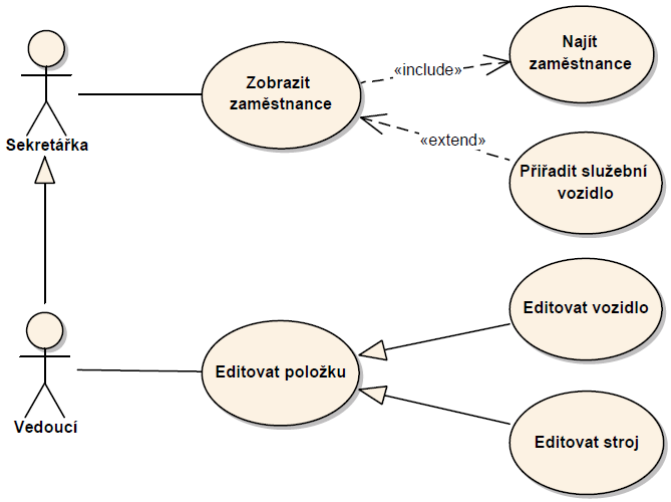


Obrázek 2.3: Ukázka znázornění Actora (aktéra).

Jednouchý Use Case diagram

Samozřejmě Use Case diagram není tvořen pouze jednotlivými aktéry a případy užití. Musí se zde nacházet také vazby mezi nimi. Ukázka jednoduchého Use Case diagramu, který znázorňuje základní možné vazby je na obrázku: 2.4.

Máme zde dva aktéry a to sice Sekretářka a Vedoucí. Podle obrázku 2.4 si můžeme všimnout, že role Vedoucí rozvijí roli Sekretářka. To nám umožňuje říci, že role Vedoucí bude mít možnost provádět stejné akce jako role Sekretářka, aniž bychom museli pro obě role definovat tytéž akce. Tomuto použití se říká zobecnění účastníka tzv. actor generalization. Podobný postup můžeme vidět u akce Editovat položku. Tomuto chování se naopak říká zobecnění případu užití tzv. use case generalization. Dále vidíme mezi případy užití Zobrazit zaměstnance a Najít zaměstnance vazbu *include*. Tato vazba nám říká, že pokud chceme zobrazit detail zaměstnance, tak budu nutné nejdříve najít zaměstnance. Poté máme mezi případy užití Zobrazit zaměstnance a Přidat služební vozidlo vazbu *extend*. Tato vazba nám naopak říká, že při zobrazení uživatele můžeme, ale nemusíme přidat uživateli služební vozidlo.



Obrázek 2.4: Ukázka jednoduchého Use Case diagramu.

Na základě obrázku 2.4 jde vidět, že nakreslení Use Case diagramu je poměrně jednoduché, přičemž má přesné a silné vyjadřovací schopnosti. Na první pohled jde vidět, jaké aktéry bude systém obsahovat a jaké operace mohou provádět. Můžeme také zobrazit určité podmíněné i nepodmíněné akce, kterou mohou být provedené.

2.4 Návrhové modely a E-R diagram

2.4.1 Návrhové modely

Konceptuální model nám slouží k analýze požadavků na data a návrhu struktury databáze. Nezáleží přitom, jakým způsobem budou data uložena. Zda např. v MySQL databázi nebo čistě v textovém souboru. Můžeme říci, že konceptuální model není ovlivněn způsobem řešení. [32]

Cílem logického modelu je vytvoření schématu relační databáze neboli logické schéma databáze, které obsahuje strukturu jednotlivých tabulek a je nezávislé na konkrétní databázové platformě. Nenalezneme zde tedy informace o způsobu uložení dat nebo SQL příkazu pro vytvoření jednotlivých tabulek v databázi. Používá se především v době, kdy ještě není známa použitá databázová platforma. Úrovní abstrakce se řadí mezi konceptuální a fyzický model. [32]

Fyzický model představuje fyzické uložení tabulek (může se jednat např. o použitý datový typ, indexy, sekvenční soubory) - popisuje způsob uložení dat. Výsledkem je fyzické schéma databáze. Tento model závisí na konkrétním SŘBD (systému řízení báze dat), přičemž se snaží o dosažení co nejlepšího výkonu databáze. [32]

2.4.2 E-R diagram

Entity-Relationship diagram neboli E-R diagram je typ diagramu, vytvořený podle Entity-relationship modelu. Jedná se vlastně o konceptuální model. Popisuje data „v klidu“. Neukazuje, jaké operace mohou probíhat s daty. Model obsahuje následující základní prvky. [22]

- **Entita:** popisuje objekt z reálného světa, o kterém chceme mít v databázi uložené informace. V rámci objektově orientovaného programování můžeme chápat entitu jako třídu. V databázi je entita prezentována tabulkou. Entita bývá pojmenována podstatným jménem např. stroj, vozidlo. Prioritou je zvolit jednoduchý a výstižný název.

- **Vztah (relationship):** jedná se o asociaci mezi jednotlivými entitami, tzn. dvě entity jsou spolu logicky spojeny. V databázi je to realizováno pomocí vztahu primárního klíče s cizím klíčem. Každý vztah v E-R diagramu je znázorněn spojem mezi dvěma entitami, má své jméno a také kardinalitu - což je vlastně počet maximálních možných vztahů daného typu. V určitých případech nám kardinalita určuje také minimální počet možných vztahů. Uvedu-li kardinalitu *, znamená to, že dolní ani horní hranice není omezená. Počet vztahů může být tedy klidně 0, anebo také např. 999. Naopak uvedu-li kardinalitu 1..*, znamená to, že musí být alespoň 1 vztah a maximálně může být opět 999. Existuje také kardinalita 1, 0..1. Zápis 0..* je totéž jako zápis *.

- **Atribut:** je vlastnost entity, která nás zajímá a chceme její hodnotu mít uloženou v databázi. V databázi je poté prezentována atributem dané tabulky. Atributy mohou být buď jednoduché, nebo složené. Jednoduché atributy nabývají skalárních hodnot. Složené atributy jsou složené z několika jiných atributů. Přičemž může být složen buď z jednoduchých atributů, složených atributů, nebo jejich kombinací.

2.4.3 Jednoduchý E-R diagram

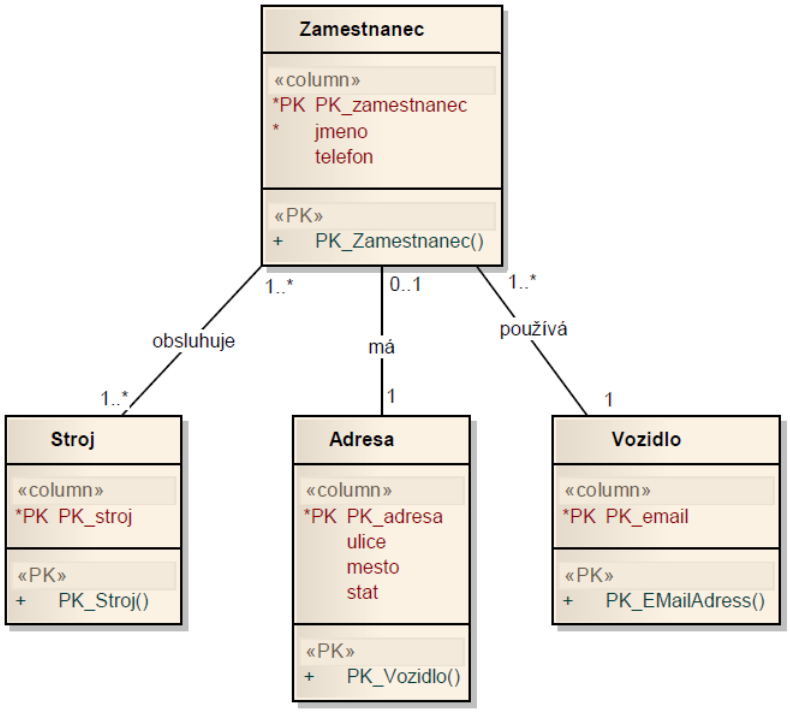
V následujícím příkladu nebudou uvedeny všechny možnosti E-R diagramu. Budou uvedeny pouze použité věci v E-R diagramu v bakalářské práci.

Na obrázku 3.2 máme celkem čtyři tabulky (ZAMESTNANEC, STROJ, ADRESA, VOZIDLO). Každá tabulka má svůj primární klíč, který slouží k jednoznačné identifikaci záznamu v konkrétní tabulce. Dále v tabulce ZAMESTNANEC můžeme vidět atributy JMENO a TELEFON. Podle hvězdičky u atributu JMENO, poznáme, že se jedná o povinný atribut - tudíž v databázi nesmí nabývat hodnoty NULL. Atribut TELEFON je naopak nepovinný a může nabývat hodnoty NULL.

Vztah mezi tabulkami ZAMESTNANEC a VOZIDLO má jméno *používá*, kardinalitu 1..* a 1. Jedná se tedy o vazbu M:1. To můžeme chápat následovně: jedno vozidlo používá více zaměstnanců a jeden zaměstnanec používá jedno vozidlo. Jedná se o velmi jednoduchou vazbu. Při implementaci se přidá pouze jeden atribut do tabulky ZAMESTNANEC, který bude obsahovat odkaz na primární klíč z tabulky VOZIDLO.

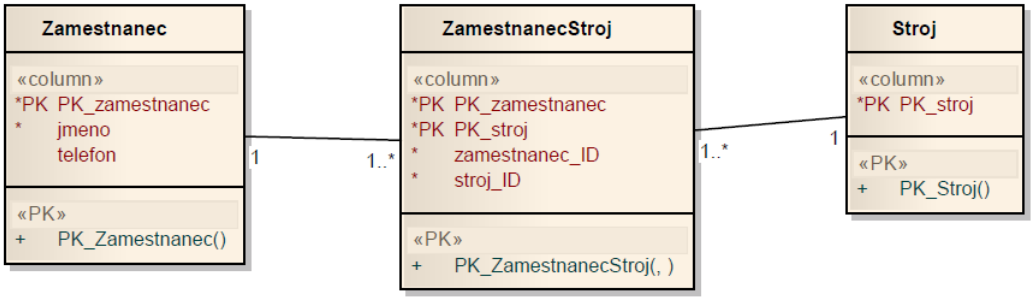
Další na řadě je vztah mezi tabulkami ZAMESTNANEC a ADRESA. Vztah má jméno *má*, kardinalitu 1..0 a 1. Jedná se tedy o vztah 1:1. Hodnota 0 v kardinalitě vypovídá o tom, zda daný odkaz na záznam do další tabulky musí existovat. Vazba 1:1 se běžně nepoužívá. Smysl má, pokud se jedná o složený atribut. Dále může být vazba použita, pokud se jedná o důležitý atribut. V našem případě nechceme plést atributy, které obsahuje tabulka ADRESA do tabulky ZAMĚSTNANEC, a chceme je mít tedy oddělené. Případně pokud touto vazbou jsme schopni předejít definování stejných parametrů v dalších tabulkách. Další možností využití této vazbu je, jestliže si nejsme zcela jisti, že v budoucnu nebude potřeba mít více adres pro zaměstnance. Implementace v databázi by vypadala podobně jako u předešlého vztahu, pouze s rozdílem, že by bylo na nás, do které tabulky umístíme nový atribut odkazující na primární klíč záznamu druhé tabulky.

Poslední na řadě je vztah mezi tabulkami ZAMESTNANEC a STROJ. Vztah má jméno *obsluhuje*, kardinalitu 1..* a 1..*. Jedná se tedy o vztah M:N. Implementace je to v tomto



Obrázek 2.5: Ukázka jednoduchého E-R diagramu.

případě trošku složitější, protože nelze pomocí dvou tabulek provést vazbu M:N. K tomu je potřeba přidat pomocnou tabulku, která bude obsahovat složený primární klíč z hodnot primárních klíčů tabulek Zamestnanec a Stroj. Dále musíme přidat dva atributy, kdy každý atribut bude obsahovat hodnotu primárního klíče jedné z tabulek Zamestnanec a Stroj. Ukázku takové implementace můžete vidět na obrázku: 3.4.



Obrázek 2.6: Ukázka implementace vztahu M:N.

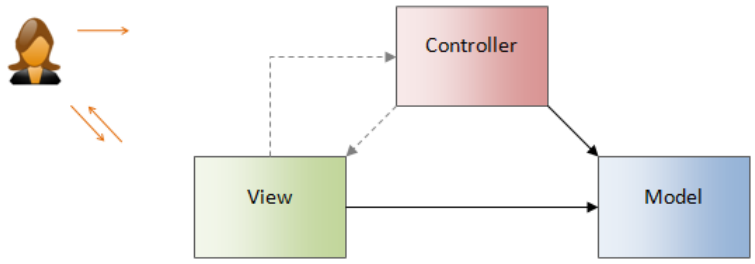
2.5 Architektura MVC

MVC je softwarová architektura, která má za cíl oddělit datový model, uživatelské prostředí a řídící logiku na sobě navzájem nezávislé komponenty, které mezi sebou komunikují.

Díky tomu je celý kód mnohem více přehledný, snáze se udržuje a modifikuje. Aplikace, která využívá MVC architekturu je rozdělena na tři části. Sice na Model, View (pohled) a Controller (kontrolér), z toho vznikla zkratka MVC. Touto architekturou se snažíme řešit problém tzv. „špagetového kódu“, kdy v jedné třídě se nachází aplikační logika, práce s databází a také vykreslení výstupu. Takový přístup je značně nepřehledný, u složitějších a rozsáhlejších aplikací nepřípustný. Princip MVC architektury se dá popsat následovně.

1. Uživatel provede akci na pohledu.
2. Tuto akci zachytí kontrolér.
3. Kontrolér se rozhodne jak na akci reagovat. Může se jednat o změnu hodnot pomocí modelu nebo ovlivnit přímo pohled.
4. Pohled zobrazí provedené změny uživateli.

Jak vypadá MVC architektura s interakcí uživatele můžeme vidět na obrázku 2.7. [17, 7]



Obrázek 2.7: MVC s interakcí uživatele. Převzato z [17].

2.5.1 Model

Model můžeme chápat jako datový základ, který navíc obsahuje aplikační logiku aplikace. Můžeme zde mít např. práci s databázovou vrstvou (získání potřebných dat z databáze) nebo složitější výpočty. Jak bylo na začátku řečeno, model je nezávislá komponenta, tzn. že nic neví o pohledu a kontroléru. Má za úkol pouze získat data, zpracovat je a předat je dále. Zpracováním dat můžeme rozumět např. úpravu získaných dat do požadovaného formátu, ošetření správného formátu hesla, případně jeho porovnání. Modelem rozumíme model celé webové aplikace, který je běžně tvořen více objekty. [17]

2.5.2 View

View neboli Pohled nám slouží k zobrazení výstupu požadavku uživateli v patřičném formátu. Nejčastěji se můžeme setkat se šablonou, která obsahuje HTML stránku a data zobrazuje přehledněji pro uživatele. Zatímco model můžeme mít pouze jeden, pohledů bývá většinou několik. Příkladem může být pohled, který vykreslí tabulku, další např. seznam, anebo formulář. Pohledy obvykle využívají dědičnosti, a to z důvodu zamezení neustále opakujícího se kusu kódu. Např. aby se v každém pohledu znovu nevytvářelo menu. Místo toho můžeme definovat další pohled, který obsahuje právě naše menu, a tento pohled pouze

připojíme do dalších pohledů. To celé přispívá k mnohem větší přehlednosti kódu a ke zjednodušení následujících úprav. [17, 7]

2.5.3 Controller

Controller neboli kontrolér je prvek, který propojuje model a pohled. Zpracovává požadavky uživatele, na jejich základě volá patřičné akce modelu a poté zpracované data předá pohledu, aby je mohl vykreslit. [7]

2.6 Nette Framework

Nette Framework je framework pro tvorbu webových aplikací v PHP 5, který je šířen jako svobodný software s licencí NewBDS nebo GPL. Jeho rozhraní a syntaxe se snaží být co nejvíce srozumitelná. Zaměřuje se především na eliminaci bezpečnostních rizik a znovupoužitelnost kódu. Můžeme v něm vytvořit např. blogy, informační systémy a e-shopy. Mezi hlavní výhody patří: [3]

- šablonovací systém Latte,
- Tracy „laděnka“, která pomáhá hledat přehledně chyby v kódu,
- široká komunita a kvalitní dokumentace,
- Open-source licence,
- podpora především: AJAX, AJAJ, MVC, HTML5.

2.6.1 Bezpečnost frameworku

Nette Framework nám poskytuje komplexní zabezpečení našich webových aplikací. Samotný kodér se nemusí o zabezpečení starat. Díky tomu nedochází k narušení zabezpečení, např. pokud kodér zapomene některé části ošetřit.

Cross-Site Scripting (XSS)

Jedná se o princip narušení webových stránek, který využívá bezpečnostní chyby ve skriptech - zejména neošetřené vstupy. Útočník v tomto případě dokáže podstrčit vlastní kód, čímž může dojít např. k pozměnění, získání citlivých údajů o uživatelích nebo obejití bezpečnostních prvků aplikace. Nebezpečí hrozí, pokud nám útočník podstrčí JavaScriptový kód, který se ihned provede. Obranou je důkladný a naprostý převod znaků, které mají v kontextu vypisovaných dat zvláštní význam. V jednoduchém příkladu útoku 2.1 posíláme přes metodu GET parametr ID. Útočník nám místo tohoto parametru podvrhne vyskakovací ALERT okno. Tento příklad je triviální a spíše nám ukazuje princip, na kterém útok funguje. Mnohem častější bývá podstrčení adresy, na které se nachází soubor obsahující JavaScriptový kód, který už může měnit a získávat mnohem důležitější a z bezpečnostního hlediska citlivé informace. [29]

Nette Framework nás proti tomuto útoku chrání důkladnou kontrolou a ošetřením všech vstupů pomocí speciální technologie Context-Aware Escapin. Cílem této technologie je ošetřit všechny vstupy za kodéra tak, aby nedošlo k jejich opomenutí. Vše probíhá automaticky na pozadí, kodér se nemusí o nic starat a nic nepozná. [8]

Zdrojový kódu 2.1: Vložení scriptu do URL

```
www.test.cz/index.php?id=<script>alert('Utok byl uspesny.');
```

Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) je další z metod útoku na webové aplikace. Spočívá v požadavku, který uživatel nevykonal, např. smazání nějaké položky. Ve většině případů se nejedná vyloženě o útok k získání přístupu do aplikace, jedná se spíše o zneužití akce právě přihlášeného konkrétního uživatele. Pro tento typ útoku je potřeba, aby útočník znal strukturu aplikace a tedy měl na co zaútočit. Příkladem může být určitý informační systém. Útočník přiměje uživatele, který je přihlášený do toho systému, aby navštívil stránku, která provede útok. Tato stránka poté může generovat HTTP Request (v HTTP metoda GET), který provede změnu v systému, např. editace, změna nebo přidání nové položky (záznamu). [15]

Nette Framework nás chrání generováním autorizačního tokenu a následně jeho ověřováním. Této ochrany dosáhneme pouze jediným příkazem: `$form->addProtection()`. [8]

URL attack, control codes, invalid UTF-8

Všechny uvedené případy představují pokus útočníka podstrčit webové aplikaci škodlivý vstup např. pomocí hodnot parametrů předaných pomocí URL. Způsob jak těmto útokům předejít je ochrana všech vstupů webové aplikace na úrovni bajtů. To je pro kodéra velmi pracné a náročné. Nette Framework tuto ochranu řeší za nás, a to zcela automaticky. Nemusíme se tedy vůbec o nic starat, nic nastavovat a přesto budou všechny vstupy ošetřené.

Session hijacking, session stealing, session fixation

HTTP protokol je bezstavový. Tzn. neuchovává žádné informace mezi jednotlivými HTTP Requesty. Z tohoto důvodu byl protokol HTTP rozšířen o HTTP cookies, což jsou krátké textové řetězce, které umožní uchovat určité informace o stavu. Z důvodu malé délky těchto řetězců a ukládání na straně klienta - nikoliv severu, není vhodné do nich zapisovat rozsáhlejší informace. K tomuto účelu slouží session. Webovému serveru je tak umožněno uložit si data přiřazená ke konkrétnímu uživateli a následně se na ně pouze odkazovat. Nedochází tak znovu k přenosu těchto dat. Session ID (SID) je uloženo do HTTP cookies a podle tohoto identifikátoru je uživateli umožněno přistoupit k uloženým datům. Dokud je SID stále stejné, server předpokládá, že komunikuje stále se stejným uživatelem. Z bezpečnostního hlediska nepatří SID do URL adresy, ale přenáší se pouze pomocí cookies. Je tedy důležitá konfigurace na straně serveru tak, aby případné SID v URL ignoroval.

Nette Framework se postará za nás o správné nakonfigurování serveru. Např. v případě přihlášení nového uživatele vygeneruje nové SID. Jedinou podmínkou je povolení funkce `ini_set()` na straně serveru. [8] Konfigurace session v tomto informačním systému je popsán v kapitole 5.10.

2.6.2 Latte

Latte je velice schopný šablonovací systém pro jazyk PHP, který je součástí frameworku Nette. Za cíl si klade zpříjemnit a zjednodušit práci a zabezpečit vstupy před možnou

zranitelností, která je popsána v kapitole 2.6.1. Šablonovací systém jako takový není určen pro implementaci aplikační logiky, ale slouží k použití v šablonách a vykreslování dat, které dostane. Syntaxe se snaží být jednoduchá a přehledná. Latte šablonovací systém vyniká svou rychlostí. Samotný kód šablony se přeloží do PHP kódu a ukládá do cache na disku. Díky tomu je použití Latte stejně výkonné jako kdybychom vytvářeli šablony v čistém PHP. Oproti čistému PHP ovšem získáme vyšší bezpečnost a přehlednost celého kódu. Každá chyba se nám přehledně zobrazí v „laděnce“, viz. následující podkapitola 2.6.3. [6]

Filtry v Latte jsou vlastně funkce, které slouží k úpravě a přeformátování dat do jiné podoby. Filtry se zapisují pomocí svislé čáry např. `<h1>retezec|filtr</h1>`. Pro zajímavost zde uvádím několik filtrů:

- `lower` => převod textu na malá písmena,
- `firstUpper` => převod prvního písmene na velké,
- `date` => umožňuje nám formátovat datum,
- `Bytes` => přeformátuje velkost v bytech do čitelnějšího formátu.

2.6.3 Tracy

Tracy je knihovna, která je součástí Nette Framework a slouží k debugovaní a zpracování chyb. Mezi její hlavní výhody patři:

- rychlé odhalení a logování chyb,
- vypisování obsahu proměnných, výsledků funkcí a nejrůznějších mezi stavů,
- měření času, využití alokované paměti, dotazů do databázové vrstvy.

„Laděnku“ je potřeba aktivovat. Nejdříve do projektu připojíme knihovnu `use Tracy\Debugger;` a následně ji ještě povolíme `Debugger::enable();`. Povolení „laděnky“ se doporučuje provést hned za načtení frameworku v souboru `bootstrap.php`. Zde stačí v příkazu `setDebugMode();` nastavit hodnotu `TRUE`. [5]

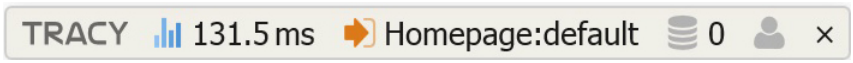
Po povolení „laděnky“ se nám zobrazí plovoucí Debugger bar panel v pravém dolním rohu stránky. Můžeme zde vidět dobu provádění, název aktuálního presenteru a jeho akce, počet dotazů do databáze a identitu přihlášeného uživatele. Pohled na základní Debugger bar obrázek 2.8.

Mnohem zajímavější pohled nám ale „laděnka“ poskytne, pokud si zobrazíme informace např. *System info*. Zobrazí se nám poté mnohem detailnější informace. Můžeme zde vidět informace, jako je doba provádění, množství přidělené paměti, počet připojených souborů, nebo třeba verzi PHP a verzi Tracy.

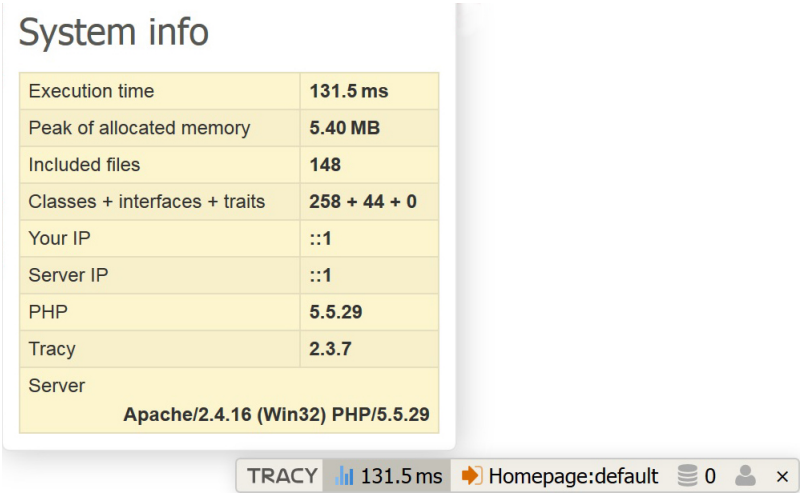
Jak můžeme vidět, tak „laděnka“ je velice mocný nástroj, který nám může velice pomoci a ušetřit čas při odhalování a opravovaní chyb, jelikož nám chyby přehledně zobrazí a řekne nám, co se jí nelíbí.

2.6.4 MVC aplikace a presentery

Nette Framework používá architekrutu MVP. Což se dá přeložit jako Model - View - Presenter. Presentery jsou vlastně obdobou Controlleru u MVC. V této kapitole se zaměřuji spíše na MVP u Nette Frameworku. Změny v MVP oproti MVC se dají popsat následovně.



Obrázek 2.8: Debugger bar z knihovny Tracy.



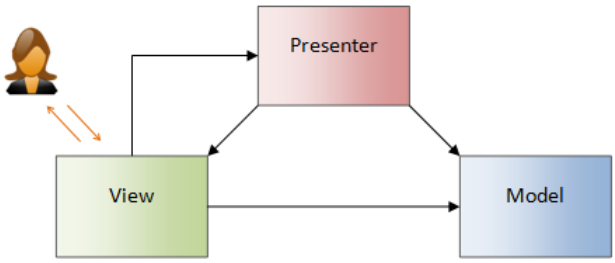
Obrázek 2.9: Detail položky System info.

1. Uživatel provede akci na view (dále už pohled), tuto akci plně kontroluje pohled.
2. Pohled má přímou vazbu na presenter. Díky čemu může volat metody z presenteru.
3. Presenter přímo pracuje s pohled. V jednoduchých případech je možné obsadit práci s databází přímo do presenteru.

Je sice možné provádět práci s databázovou vrstvou přímo v presenetru, ovšem v takovém případě by se nejednalo o správný návrh architektury aplikace. Takové užití se dá považovat za rozumné pouze buď jako ukázka, anebo ve velmi jednoduchých webových aplikací, u kterých víme, že je už dále nebudeme nikdy rozšiřovat. Přesto je vždy doporučováno řídit se architekturou MVP a důsledně přesunout práci s databází do modelu. V ideálním případě by se pohled měl starat pouze o vykreslení výstupu a může obsahovat podmínky `if - else` případně cyklus pro procházení polí. Model by se měl starat o aplikační logiku a práci s databází. Presenter by naopak měl obstarat správné data pro pohled, které získá pomocí akcí Modelu. Dále by měl určit, jaký pohled se použije. Princip MVP architektury s interakcí uživatele můžeme vidět na obrázku 2.10. [18]

Adresářová struktura

Distribuce Nette Frameworku obsahuje tzv. Sandbox. Což není nic jiného než základní kostra aplikace, která se řídí MVP architekturou. Najdeme zde mimo jiné základní autorizátor tvořený modelem, přihlašovací formulář, presentery starající se o zpracování chybových zpráv, obsluhu přihlášení, zobrazení úvodní stránky, dále jsou k nim obsaženy odpovídající pohledy. Základní souborová struktura společně s popiskami jednotlivých složek je v příloze A.



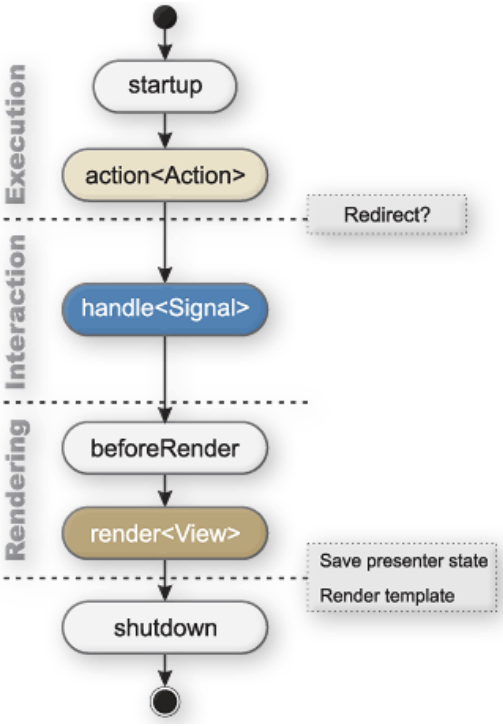
Obrázek 2.10: MVP s interakcí uživatele. Převzato z [18].

Všechny požadavky, které provede uživatel, se předávají přes soubor `index.php`, který následně předá řízení aplikaci do zavádějícího souboru `bootstrap.php`. Pokud bychom změnili adresářovou strukturu, je potřeba provést změny také v souboru `bootstrap.php`. [7]

Životní cyklus presenteru

Každý presenter má svůj životní cyklus. Podle toho musíme také volat akce, které chceme vykonat. Životní cyklus presenteru je znázorněn na obrázku 2.10. Presenter má definované následující akce - metody. [7]

- **startup()** - Po vytvoření presenteru se zavolá tato metoda, která má za úkol inicializovat proměnné a prostředí.
- **action<jméno>()** - Jedná se o podobnou metodu jako je metoda `render<jméno>()`. Hlavní rozdíl je v tom, že metoda `action<jméno>()` se volá dříve než metoda `render<jméno>()`. Jako příklad mohu uvést následující. Po provedení určité akce dojde k přesměrování na metodu `render<jméno>()`, která potřebuje přijmout parametry. Zároveň má dojít k vykreslení formuláře, který obsahuje přijaté parametry. Bohužel formulář bude prázdný. To je z toho důvodu, že metoda `render<jméno>()` se volá, až po vytvoření formuláře. Z tohoto důvodu je potřeba použít metodu `action<jméno>()`, která se zavolá dříve, než je formulář vytvořen a proto se také může formulář naplnit přijatými parametry. Zároveň metoda `action<jméno>()` umožňuje rozhodnout, který pohled bude následně vykreslen.
- **handle<jméno>()** - Metoda zpracovává signály, které jsou presenteru zaslány např. z pohledu. Signály mohou sloužit k zavolání metody v modelu a změně hodnoty v databázi. Využití je zejména v ohledu, jestliže výsledek akce nepotřebujeme zobrazit, a nebo k tomu použijeme aktuální šablonu. Na této úrovni se také vytvářejí a zpracovávají formuláře.
- **beforeRender()** - Metoda se volá před samotnou metodou `render<jméno>()`. Slouží např. k nastavení šablon.
- **render<jméno>()** - Metoda, která slouží k naplnění šablony daty.
- **shutdown()** - Metoda se zavolá při ukončení životního cyklu daného presenteru. Presenter se dá ukončit kdykoliv během jeho životného cyklu. Obvykle to je z důvodu, že nechceme vykreslit odpovídající šablonu.



Obrázek 2.11: Životní cyklus presenteru. Převzato z [7].

2.7 Bootstrap

Bootstrap je jednoduchý a dostupný CSS framework. Dalo by se říci, že je to vlastně připravená sada hotových stylů, pomocí kterých lze vytvořit jednoduché, ale přesto přehledné webové aplikace bez zapojení grafika. Pro plnohodnotné využití frameworku je potřeba připojit JavaScriptový soubor. Bootstrap nám poskytuje styl např. pro tlačítka, vysouvací seznamy, alerty, modální okna, formuláře. Po stažení Bootstrapu máme k dispozici základní vzhled. Existují pro něj ovšem také nejrůznější styly. Můžeme najít placené - velmi propracované, které se zcela liší od základního vzhledu. Najdeme styly také zcela zdarma - ty většinou vycházejí z výchozího stylu a mají pouze jinou škálu barev. Přesto se jedná o zajímavou alternativu k běžnému vytváření vzhledu pomocí CSS. Pro použití Bootstrapu musíme znát základy HTML. Styl se k HTML prvkům připojuje pomocí atributu `class=' '`. Tento atribut poté obsahuje názvy jednotlivých tříd frameworku a tím dojde k připojení stylů k prvkům. [4]

Mezi hlavní přednosti patří:

- jednoduché použití,
- přehledná dokumentace s příklady,
- možnost ovlivnit nastavení výchozího stylu (vygenerováním vlastního nastavení Bootstrapu),
- bezproblémová spolupráce např. se Nette Frameworkem,
- dostupné styly, které umožňují rychlou změnu designu,

- ušetření času při tvorbě vzhledu webové aplikace,
- responzivní web.

Bootstrap se dá také použít pro responzivní řešení webové aplikace. V tomto případě si je ale třeba uvědomit několik věcí. Bootstrap nám může velmi pomoci, můžeme využít celý framework a nebo si vytáhnout pouze části, které potřebujeme. Může se jednat např. o formuláře nebo navigační lištu. Bootstrap už nijak neřeší rychlost načítání nebo různé verze komponent pro různé velikosti displeje. [24]

Kapitola 3

Práce s QR kódy

V této části jsou uvedeny základní informace o QR kódech a jejich specifikace. V kapitolách Specifikace 3.2, Chybová korekce dat 3.3 čerpám ze stránek společnosti, která vytvořila QR kódy [14]. Pokud byla některá informace převzata odjinud, tak obsahuje odkaz na citaci, která se týká dané informace.

3.1 Základní informace

Zkratka QR znamená quick response, to můžeme přeložit jako rychlá odpověď. Pokud se zamyslíme, tak název tento kód vystihuje zcela přesně. Stačí kód přečíst např. mobilním telefonem a hned se nám zobrazí údaje z QR kódu. Může se jednat třeba o text, anebo nejčastěji URL adresu. Výhoda zakódování URL adresy spočívá v tom, že stačí QR kód naskenovat a přejít na nabídnutou URL adresu. Odpadá tak opisování URL adresy, které u složitějších případů může být zdlouhavé, a nemusíme se přitom vyhnout chybě. QR kód se využívá nejčastěji na plakátech, obalech výrobků, časopisech nebo třeba také na tričkách.

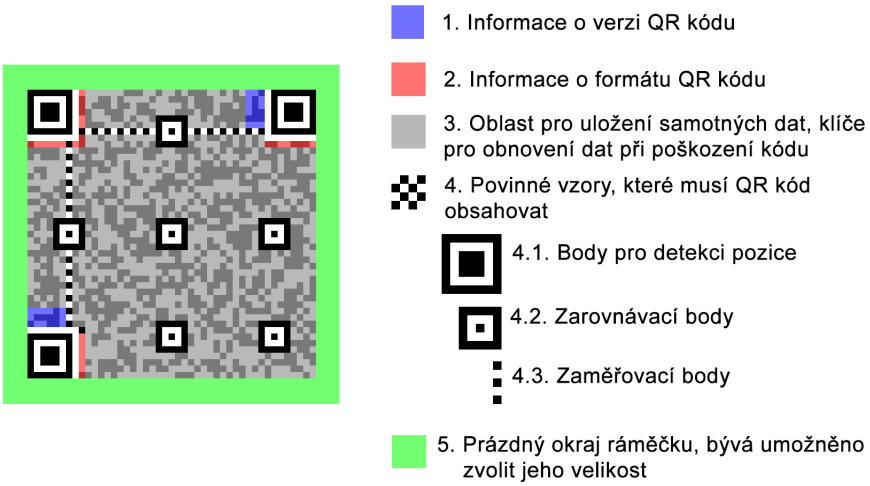
QR kód je typ 2D kódu, který se skládá ze čtverečků dvou různých barev. Nejčastěji můžeme vidět barevnou kombinaci černé a bílé barvy. Není ale problém vytvořit ani QR kód z modré a bílé barvy, červené a bílé barvy, nebo zcela jiných barevných kombinací.. Důležité je ovšem si uvědomit, že zvolená barevná kombinace musí mít co největší barevný kontrast. Je to z toho důvod, abychom byli schopni daný QR kód přečíst a hlavně jej dokázali přečíst bez chyb. Dokážeme si představit, jaké následky by mohla mít nevhodně zvolená barevná kombinace např. při QR platbě. Kdyby nám čtečka místo částky 500,- načetla 5000,-. To by nám asi moc příjemné nebylo. Proto se ve většině případů volí barevná kombinace černé a bílé barvy. Jiné barevné kombinace se používají spíše tam, kde se nám klasická kombinace nehodí třeba po designové stránce.

Při používání QR kódu je potřeba si uvědomit, že je zbytečné jej za každou cenu dávat všude. Zamysleme se, jaké mají využití, jestliže jsou na letáku, který odkazuje na URL adresu, kde jsou stejné informace jako na letáku. QR kódy by měly přinášet určitou přidanou hodnotu k tomu, kde se vyskytují. Příkladem může být etiketa vína, kde QR kód bude obsahovat odkaz na WWW stránku, na které si přečteme podrobnější informace o daném víně. V takovém případě má jejich využití zajisté smysl a přínos. Určitým problémem může být bezpečnostní riziko jejich používání. Může nastat situace, kdy někdo vloží do QR kódu odkaz vedoucí na stránku se škodlivým kódem, který bude okamžitě proveden po přechodu na tuto stránku. Modelová situace může nastat, pokud někdo např. na veřejných plakátech přelepí QR kód za svůj se škodlivým obsahem. Následky budou už záležet pouze na kreati-

vitě škůdce. Ve chvíli kdy vidíme někde QR kód, tak vlastně netušíme, co se v něm nachází. To stejné platí, pokud otevíráme URL adresu tak netušíme, kam vede. Pouze se můžeme domnívat, že má co dočinění s obsahem, kde tento QR kód byl zobrazen. [30]

3.2 Specifikace

Specifikace QR kódu jsou v současné době řízeny standardem ISO 18004:2015. QR kód obsahuje několik pevně daných zón, které nesou informace o verzi kódu, typu dat a jeho formátu. Jedná se např. o tři velké čtverce, které využívá čtečka a slouží ke správnému zaměření kódu. Díky tomu je možné přečíst kód jakkoliv otočený (v jakémkoliv úhlu od 0° do 360°), zároveň je dosažena vyšší rychlost čtení. V dalších oblastech najdeme už samotné zakódované data. Struktura QR kódu je znázorněna, společně s popisky co jednotlivé sekce znamenají, na obrázku: 3.4.



Obrázek 3.1: Struktura QR kódu. Převzato a upraveno z [10].

Mezi nejčastější používané formáty dat, které jsou podporované generátory, patří následující.

- Kontaktní údaje (vizitka) - umožňuje uložit informace jako jsou např. jméno, příjmení, telefoní číslo, E-mail, adresa.
- Kalendářová událost,
- URL adresa,
- E-mail - umožňuje uložit informace jako jsou adresa příjemce, předmět, zpráva.
- Telefonní čísla,
- GPS souřadnice,
- Prostý text,
- SMS zpráva - umožňuje uložit informace jako telefonní číslo a předmět SMS zprávy.

Některé generátory podporují i další formáty dat, jako jsou údaje o Facebookovém profilu, anebo Wifi přihlašovací údaje.

Běžný čárový kód typu EAN, UPC, Code 2/5 nám umožňuje uschovat cca 10 numerických znaků [16]. Existují také čárové kódy pro průmyslové využití, které nám umožňují zakódovat více znaků a mohou mít variabilní délku. Příkladem jsou kódy Codabar - 16 znaků, Code 39 - 43 znaků nebo Code 128 - 128 znaků ASCII [16]. QR kód nám umožňuje uložit několikanásobně více dat, viz. tabulka 3.1.

Typ kódovaného obsahu	**Maximální počet znaků**
numerické znaky	7089
alfa-numerické znaky	4296
8bitová data	2953
Kanji znaky	1817

Tabulka 3.1: Množství možných uložených dat v QR kódu.

Další rozdíl oproti EAN kódu je v jeho velikosti. Běžné čárové kódy nám dovolují zakódovat obsah pouze v horizontálním směru. Naopak QR kód zakóduje obsah jak v horizontálním tak i vertikálním směru. Díky tomu můžeme také zakódovat mnohem více dat. Ukázku můžeme vidět na obrázku: 3.2, kde kódy EAN 13 i QR obsahují oba posloupnost čísel 012345678910. Můžeme vidět, že QR kód má zcela jinou strukturu. Běžné čárové kódy jsou tvořeny pomocí čar, QR kód naopak pomocí bodů (čtverců o různé velikosti) [31].

Obrázek 3.2: Porovnání EAN a QR kódu se stejným zakódovaným řetězcem.

QR kódy existují v několika verzích. Celkem máme 40 velikostí QR kódů. Liší se od sebe počtem modulů. Modul je nejmenší prvek QR kódu. Černý čtvereček odpovídá logické jedničce a bílý logické nule. Nejmenší verze QR kódu má 21x21 modulů, naopak největší verze má 177x177 modulů. S každou vyšší verzí QR kódu se počet modulů zvětšuje o 4 v horizontálním a vertikálním směru. Dalo by se tedy říci, že verze QR kódu nám určuje, kolik můžeme uložit dat a množství (velikost) kontrolních klíčů. Čím více dat chceme uložit do QR kódu, tím větší musí být QR kód.

Existuje několik typů QR kódů. Liší se od sebe počtem modulů, počtem kontrolních bodů, nebo strukturou. Přehled níže dále zmíněných QR kódu najdeme na obrázku: 3.3. Mezi nejčastěji používané typy můžeme zařadit následující.

- **QR kód Model 1 a Model 2** - Jedná se o základní verzi QR kódu, zároveň také největší. Model 1 je původní verze a jeho maximální velikosti je 14, což odpovídá

73x73 modulům. Model 2 je vylepšení původního návrhu, umožňuje použít největší verzi 40, která odpovídá 177x177 modulů.

- **Micro QR kód** - Hlavní rozdíl oproti QR kódu Model 2 je v počtu zaměřovacích bodů. Výše zmíněný kód potřebuje tři zaměřovací body. Zatímco Micro kódu stačí pouze jeden. Díky tomu je možné dosáhnout mnohem menší plochy při tisku. Dále je možné u tohoto kódu snížit šířku okraje. Kvůli omezení v počtu zaměřovacích bodů a menší plochy, největší verze obsahuje pouze 17x17 modulů.
- **Frame QR** - Jedná se o typ QR kódu, který v sobě má „plátno“. Na tomto plátně mohou být umístěny např. obrázky nebo další text. Díky této vlastnosti je nám umožněno zvýšit vyjadřovací schopnost QR kódu a také uživatele více zaujmout. Samozřejmě tento kód zvládne uložit mnohem méně dat než Model 2.

Dále můžeme ještě narazit na tyto QR kódy.

- **iOR Code** - Kód může mít místo klasických čtvercových modulů, moduly obdelníkové. Dokáže pojmout mnohem více dat než běžný QR kód. Maximální velikost je 61, což odpovída 422x422 modulů.
- **SQRC** - Má omezené typy dat, které je možné zakódovat. Hodí se spíše pro správu interních informací.

Obrázek 3.3: Přehled typů QR kódu. Převzato a upraveno z: [14].

3.3 Chybová korekce dat

QR kód má chybovou schopnost korekce dat. Pokud dojde k určitému poškození QR kódu, tak je pořád možné zakódované data přečíst. Příklad můžete vidět na obrázku 3.4. Příklad 1 nám znázorňuje poškození znečištěním QR kódu. Samotné přečtení není tak rychlé jako v případě nepoškozeného kódu, ale přesto je možné obsah stále přečíst. Příklad 2 znázorňuje už značnější znečištění QR kódu. V tomto případě už přečtení možné není. Příklad 3 nám představuje poškození QR kódu za situace, kdy nám část kódu chybí. Situace je velmi podobná jako v případě Příkladu 1. Samotné přečtení může trvat o něco déle, ale obsah nakonec je možné přečíst. Příklad 4 znázorňuje chybějící větší část kódu. V tomto případě už nelze obsah přečíst. Zda se nám nakonec podaří QR kód přečíst rozhoduje míra poškození kódu a také kvalita čtečky. V případě mobilního telefonu se jedná také o kvalitu fotoaparátu a schopnost kvalitně zaostřit.

Existují celkově 4 úrovně schopnosti korekce dat. Tyto úrovně jsou označeny písmeny L, M, Q a H a odpovídá to asi 7%, 15%, 25% a 30% z plochy kódu. Je třeba si uvědomit, že čím větší míru korekce dat použijeme, tím můžeme uložit méně informací. Je to z tohoto důvodu, že společně se zakódovanými daty jsou uloženy také informace pro jeho obnovu (správné přečtení).

Obrázek 3.4: Jednotlivé typy poškození QR kódu.

3.4 Jak vytvářet QR kódy

3.4.1 Internetové generátory

Pokud chceme vytvořit jednorázově QR kód a nepotřebujeme jeho zakódovaný text nikam ukládat, můžeme využít některý z online generátorů. Na internetu jich můžeme nalézt spoustu. Uvádím jen krátký přehled.

- Server QRgenerator.cz nám nabízí příjemný generátor kódu. Umožňuje nastavit rozměry výsledného obrázku, který obsahuje vygenerovaný QR kód. Dále umožňuje nastavit úroveň korekce dat. Jako typy dat můžeme použít prostý text, E-mail, telefonní číslo, URL adresu, SMS zprávu, GPS polohu, WiFi, vizitku. Vygenerovaný QR kód je ve formátu PNG.
- Další možnost je server qikni.cz, který nám umožňuje vygenerovat QR kód jako obrázek ve formátu PNG, anebo si můžeme vygenerovaný QR kód nechat zaslat na E-mailovou adresu. Také umožňuje zvolit úroveň korekce dat. Typy dat jsou velmi podobné jako v předchozím příkladu. Rozdíl je pouze v tom, že můžeme vygenerovat také událost v kalendáři (např. schůzka), anebo platbu. Typ dat událost nám umožňuje vložit hodnoty jako je název události, datum a čas konání, místo, popis. Typ dat platba slouží k zakódování platebním údajů pro určitou platbu. Najdeme zde údaje jako číslo účtu, částka, variabilní/specifický/konstantní symbol, datum splatnosti nebo zprávu pro příjemce.
- Velice zajímavý generátor poskytuje server Qoqr.me. Kromě funkcí, které měly výše dva uvedené generátory, poskytuje barevné nastavení QR kódu, dále můžeme uložit vygenerovaný QR kód ve formátu SVG/EPS/PNG/JPEG.

Implementace do uživatelských aplikací

Implementovat QR kódy do uživatelských aplikací jde provést několika způsoby. Každý způsob má své výhody i nevýhody. Nejdříve přehled jaké možnosti se nám naskytují:

- Google Charts,
- JavaScript,
- Knihovny pro programovací jazyky. Pro PHP se jedná pro PHP o QR Code knihovnu, pro Javu o knihovny ZXing API, QRGen API.

3.4.2 Google Charts

Google Charts umožňuje velmi jednoduché generování. Na stránce stačí zobrazit formulář, který načte vstupní text, vloží jej do URL adresy, na které se následně zobrazí vygenerovaný QR kód. Příklad takové adresy může být: 3.1.

Zdrojový kódu 3.1: URL adresa vygenerovaného obrázku

```
http://chart.apis.google.com/chart?cht=qr&chs=250x250&chl=Test&
    chld=H|0
```

Základní kořen URL adresy je : `http://chart.apis.google.com/chart?`. Poté následuje parametr `cht=qr`, tím se určí, že se jedná o QR kód. Znak `&` představuje oddělovač. Parametr `chs=` určuje, jaký rozměr bude mít obrázek s vygenerovaným kódem. Dále máme parametr `chl=`, který obsahuje náš vstupní řetězec. Poslední parametr v našem příkladu je `chld=`. Tento parametr nám umožňuje zadat úroveň korekce dat. V příkladu máme zadanou úroveň H což je 30%. Můžeme si povšimnout, že dále následuje symbol |, který nám určuje rámeček kolem QR kódu. Další parametr, který je možné zadat, ale není zmíněn je `choe=`. Tento parametr slouží k určení formátu kódování dat. Může mít hodnoty: UTF-8 - výchozí hodnota, Shift_JIS, ISO-8859-1. V případě tohoto parametru a `choe=` se jedná o nepovinné parametry. Zbytek parametrů je už povinný a musí být zadán. [11]

Nevýhodou tohoto řešení je nutnost přístupu na internet. Bez něj se nedá vygenerovat QR kód. To může být v některých případech problém, jedná-li se např. o vnitropodnikový server.

3.4.3 JavaScript

Tento typ řešení má několik výhod. Mezi ně patří rychlost generování. Z důvodu, že se nemusí čekat na odpověď ze strany serveru. Dále se nepřenášejí žádné data třetí straně. Také není potřeba být připojený k internetu. Další výhoda je, že si můžeme zvolit formát vygenerovaného QR kódu. Tzn. QR kód může být zobrazen jako klasický obrázek, a nebo se může vykreslit pomocí obyčejné tabulky - zde je poté nevýhoda, že QR kód nejde lehce nabídnout uživateli ke stažení. Zároveň nedochází k neustálému odesílání požadavků na server a generování se provádí na straně klienta. Není poté ani problém přidat vykreslení (vygenerování) QR kódu po zadání každého znaku. Změnu tak vidíme okamžitě.

Naopak nevýhoda spočívá v nutnosti vykonat na stránce skript. Pokud tedy si uživatel zakáže v prohlížeči provádění JavaScriptů, tak vygenerování nebude možné. Další nevýhodou je nutnost alespoň základní znalosti práce s JavaScriptem.

Pro tento ty generování existuje několik hotových skriptů, které můžeme použít. Bohužel ne všechny podporují český jazyk, což nám nemusí v určitých případech vyhovovat. Níže uvádím přehled několika hotových JavaScriptů, které lze využít pro generování QR kódu.

- **educastellano/qr-code** - Velice povedený nástroj. Podporuje český jazyk. Umožňuje nám generování do HTML tabulky, formátů SVG a PNG. Umožňuje nastavit velikost a rámeček výsledného QR kódu. Je licencovaný pod MIT licencí.
- **davidshimjs/qrcodejs** - Tento skript podporuje český jazyk. V minulosti tomu tak, bohužel nebylo. Umožňuje nastavit velikost výsledného vygenerovaného obrázku v pixelech, míru chybové korekce dat, barevnou kombinaci. Je tedy možné zvolit barevnou kombinaci např. modré a bílé barvy. Výsledný QR kód je generován do HTML5 plátna canvas a podporuje tagy tabulky v DOM. Následně je QR kód umožněn stáhnout uživateli ve formátu PNG. Je licencovaný opět pod MIT licencí.
- **neocotic.com/qr.js** - U posledního uvedeného skriptu bohužel nefunguje správně český jazyk. Hodí se tedy pouze na použití bez českých znaků. Umožňuje vykreslit výsledek do plátna canvas, nebo jako obrázek JPEG a PNG. Dále se dá nastavit úroveň korekce dat a velikost. Můžeme také upravit barevnou kombinaci QR kódu, výchozí barvy jsou nastaveny na černou a bílou barvu. Licence je GPL verze 3.

3.5 Knihovny pro programovací jazyky

Pro PHP existuje knihovna PHP QR Code, která nemá problém s českým jazykem. Odpadá tedy problém s vypnutým JavaScriptem na straně uživatele. Knihovna nám umožňuje generovat QR kód, ale také čárový kód. Vygenerovaný kód můžeme uložit ve formátu PNG a JPEG, nebo bitové tabulky. Výstup můžeme zobrazit jako obrázek, nebo vykreslit pomocí plátna canvas případně SVG. Knihovna je implementována čistě v PHP. Pří generování jsou data ukládány do cache paměti a tím je docíleno vyšší rychlosti. Máme k dispozici také např. ladící informace nebo protokolování chyb. Knihovna nám dovoluje nastavit velikost QR kódu i jeho rámec. Knihovna má LGPL licenci. [13]

Pro jazyk Java existuje rozhraní ZXing, která kromě klasických QR kódu podporuje také tvorbu např.: UPC-A, EAN-8, EAN-13, dále průmyslové čárové kódy - Code 39, Code 128, Codabar, z 2D kódu: Data Matrix, Aztec. Z tohoto rozhraní existuje třeba port pro jazyk C++, nebo modul pro Qt framework. [12]

3.6 Jak číst QR kódy

Nesmírnou, snad by se dalo říci i hlavní, výhodou QR kódu je, že se dají naskenovat a přečíst obyčejným chytrým telefonem. Stačí pouze mít nainstalovanou potřebnou čtečku (aplikaci). Taková aplikace je v dnešní době dostupná téměř pro každou mobilní platformu. Často bývá už předinstalováná od výrobce zařízení. Díky tomu se masově rozšířily mezi širokou veřejnost. Jak můžeme vidět z níže uvedeného přehledu, tak čteček existuje opravdu velké množství. Jejich volba je zcela na uživateli a také na tom, jakou funkčnost od čtečky očekává. Některé čtečky nám dovolují přečíst pouze QR kód, jiné navíc i čárové kódy a další nám umožní i generovat vlastní QR kódy.

3.6.1 Android platforma

Všechny aplikace zmíněné v tomto odstavci, najdete v oficiálním obchodě Google Play a jsou ke stažený zdarma.

- **Google Goggles od vývojáře Google Inc.** - v tomto případě se ani tak nejedná o samotnou čtečku QR kódu, jako spíše o analyzátor pořízeného obrazu. Ten kromě aktuálního naskenování pomocí fotoaparátu, může být uložen také v telefonu (např. stažen do telefonu z internetu). Kromě čtení QR a čárových kódů totiž umožňuje hledat např. památky, obrazy, atd... Umožňuje nám nastavit velikost snímané plochy, zapnout přisvětlování LED diodou, nebo zda se má každý naskenovaný obrázek uložit do telefonu. Kromě samotného přečtení kódu, analyzuje také všechno ostatní, co je zrovna naskenováno na obrázku. Nevýhodou je nutnost připojení telefonu na internet. Dále neobsahuje automatické přečtení kódu, pokud zrovna nějaký skenujeme. Je nutné zaostřit na kód a poté jej ručně vyfotit.

- **QR čtečka od vývojáře Seznam.cz, a.s.** - čtečka od českého vývojáře. Obsahuje historii naskenovaných kódů. Jedna z jejich hlavních výhod je uživatelské prostředí v českém jazyce. Dále si poradí s QR, DataMatrix kódem a podle výrobce s několika typy čárových kódů. Podle testování bez problémů přečte čárové kódy typu EAN 8, EAN 13, UPC-A a UPC-E. Po naskenování nám zobrazí obsah kódu, dále je možné odeslat přečtený obsah pomocí SMS zprávy, E-mailu nebo sdílet na Facebook. Můžeme také po naskenování zobrazit skenovaný kód.

- **QR Code Reader od vývojáře Scan** - čtečka s jednoduchým rozhraním. Umožňuje nastavit, zda chce při skenování použít přisvětlovací LED diodu, vybrat obrázek z mobilního telefonu a najít na něm kód. Dále poskytuje historii již naskenovaných kódu. Zajímavostí je, že přestože se jedná o čtečku QR kódu, tak umožňuje naskenovat také běžné čárové kódy např. EAN 8, EAN 13, UPC-A, UPC-E. S vizuálně podobným DataMatrix kódem si už poradit nedokáže. Přečtený obsah můžeme odeslat pomocí E-mailu, nebo si jej zkopírovat do schránky.

3.6.2 Windows Mobile platforma

Aplikace uvedené v tomto odstavci, najdete v oficiálním obchodě Windows Store. Ke stažení jsou zdarma.

- **qr kód od vývojáře Trafalgar Law** - aplikace slouží ke skenování a sdílení kódů. Umožňuje nám také vytvářet vlastní QR kódy, které můžeme následně sdílet pomocí E-mailu, sociálních sítí, a nebo SMS zprávy. Podle vývojáře dokáže přečíst QR, DataMatrix kód, dále čárové kódy ve formátu UPC-A, UPC-E a EAN 8, EAN 13.

- **QR Scanner RS od vývojáře EXYAIZED** - čtečka s podporou QR kódu a čárových kódu UPC-A, UPC-E, EAN 8 a EAN 13. Podporuje automatické naskenování kódu po jeho zaměření.

- **QR Code Reader od vývojáře ShopSavvy Inc.** - jednoduchá čtečka, která poskytuje přečtení QR kódu. Nepotřebuje přístup k internetu. Podporuje formáty dat QR kódu: URL adresa, E-mail, SMS zpráva, událost v kalendáři, vizitka, GPS souřadnice, prostý text a WiFi údaje.

- **QR čtečka od vývojáře Seznam.cz, a.s.** - funkcionalita je stejná jako v případě verze pro Android.

3.6.3 iOS platforma

Uvedené aplikace jsou dostupné z oficiálního obchodu iTunes.

- **QR code od vývojáře Marco Tini** - jedná se o jednoduchou čtečku pouze QR kódů, která je dostupná zdarma.
- **QR Code & Bar Code scan and management od vývojáře TowmsChang** - aplikace podporuje čtení QR, DataMatrix a Aztec kódů. Dále čárových kódů EAN 8, EAN 13, Code39, Code 39 Mod 43, Code 93, Code 128. Je dostupná zdarma. Umožňuje také vytváření QR kódů.
- **QR Code and Barcode Reader & Generator od vývojáře Muhammed Hassan** - další aplikace, která kromě běžného čtení QR a čárových kódů umožňuje také vytvářet vlastní QR kódy. Naskenované údaje ukládá do historie. K dispozici je zdarma.

3.6.4 BlackBerry platforma

Všechny aplikace zmíněné v tomto odstavci, najdete v oficiálním obchodě BlackBerry World a jsou ke stažení zdarma. Jelikož nevlastním telefon s touto platformou, tak se jedná spíše jen o přehled několika vybraných aplikací pro čtení QR kódu.

- **QR Code od vývojáře Sentio Ltd** - čtečka podporuje pouze QR kódy. Použití základní aplikace je zdarma. V případě verze PRO můžeme také vytvářet a generovat vlastní QR kódy.
- **NeoReader od vývojáře NeoMedia Technologies, Inc.** - aplikace podporuje čtení QR, DataMatrix, Aztec, EAN a UPC kódů. Je zcela zdarma a její používání není časově omezeno. Dále zvládá automatické naskenování kódu po zaměření mobilním telefonem.
- **QR Code Scanner od vývojáře Newgen Photographics** - čtečka podporuje pouze QR kódy. Aplikace podporuje automatické skenování, také obsahuje historii naskenovaných QR kódů. Aplikace je zdarma na vyzkoušení po dobu 5 dnů.

Kapitola 4

Návrh informačního systému

Tato kapitola se zabývá analýzou, návrhem a implementací informačního systému jako webové aplikace. Při návrhu webové aplikace jsem vycházel z diagramů Use Case a E-R. Vlastnosti těchto diagramů byly popsané v teoretické části v kapitolách 2.3 a 2.4.2.

4.1 Profil společnosti

Společnost Vinicola, s.r.o., pro kterou byl v rámci bakalářské práce vytvořen informační systém, se zabývá dovozem vína. V poslední době se také činnost společnosti rozrostla o výrobu vlastního vína. Hlavní sídlo společnosti se nachází ve městě Břeclav. Aktuálně zaměstnává ve svém hlavním sídle přibližně 11 zaměstnanců. Tito zaměstnanci mají různé pozice, jako je ředitel, zástupce ředitele, administrativní pracovnice, ekonomický pracovník, skladník a uklízečka. Každá taková pozice má jiné požadavky na informační systém a jeho funkčnost. Společnost zaměstnává také obchodní zástupce, kteří zde nejsou uvedeni. Je to z toho důvodu, že společnost má obchodní zástupce rozmístěné po celé České Republice a v sídle firmy se téměř nevyskytují. Z tohoto důvodu, tedy není nutné, aby měli zvláštní oprávnění, případně aby měli vůbec přístup do informačního systému. Společnost v dnešní době provozuje webové stránky se svou webovou prezentací a dále e-shop.

4.2 HW a SW vybavení společnosti

Společnost vznikla v roce 1997 a v současné době již používá několik softwarových řešení, které jí pomáhají ve správě produktů, jenž nabízí. Jedná se např. o účetní program POHODA, BarTender pro návrh a tisk štítků nebo etiket, nebo celní program CLA, který běží ještě pod DOSem. Samotná evidence např. vozidel, strojů, zaměstnanců je prováděna pomocí tabulkových procesorů (Microsoft Excel), v některých případech jsou záznamy pouze v papírové verzi.

Aktuálně se ve společnosti nachází 6 stolních PC, které jsou vybaveny operačním systémem Windows 7 Professional nebo Windows 8.1 Pro. Dále jsou ve společnosti 3 notebooky se systémem Windows 7 Professional a Windows 10 Pro. Všechny počítače mají nainstalované webové prohlížeče Internet Explorer, Chrome a Mozzilla Firefox. Zaměstnanci firmy používají také mobilní telefony, které využívají operační systém Android nebo Windows Phone. Společnost má také vlastní interní server bez přístupu na internet, na kterém bude umístěný implementovaný informační systém.

4.3 Analýza požadavků na informační systém

Navrhovaný informační systém nemá za cíl nahradit již používané softwarové řešení. Jeho cílem je poskytnou základní evidenci a ve smysluplných případech nahradit práci s tabulkovými procesory. Výsledkem má být tedy informační systém, který nám poskytuje zjednodušeně následující možnosti.

- Evidence zaměstnanců společnosti, nasmlouvaných externích zaměstnanců a firem, nemocenských dovolených, služebních cest, dovolených, zboží a jejich dodavatelů, strojů a jejich dodavatelů, externích vozidel (jedná se o přepravce).
- Evidence zboží, u kterých jsou uschovány skladové informace, šarže, limitované edice. Zboží se váže k přehledu objednávek, které společnost provedla u svých dodavatelů.
- Umožňuje zaměstnancům objednávat obědy. Uživatel s odpovídajícími právy si může zobrazit počet objednaných jídel, druhy jídel, a kteří zaměstnanci si tyto obědy objednali.
- Evidence vozidel, u kterých je možné evidovat servisní záznamy.
- K většině výše uvedeným položkám umožnit nahrát předem nestanovené množství souborů v různých formátech.

Výsledek analýzy požadavků na informační systém můžeme vidět na Use Case diagramu, který najdeme v příloze B. Z analýzy nám vyplynuly tyto role: ředitel/majitel, administrativní pracovnice, skladníci, ostatní zaměstnanci. Tyto čtyři role nám stačí k pokrytí přístupových práv všech zaměstnanců. Jelikož ekonomický pracovník má velmi podobné možnosti práce s informačním systém, tak může mít stejné práva jako administrativní pracovnice. Stejný případ nastává u ředitele a zástupce ředitele. Uklízečky poté spadají do kategorie ostatní zaměstnanci.

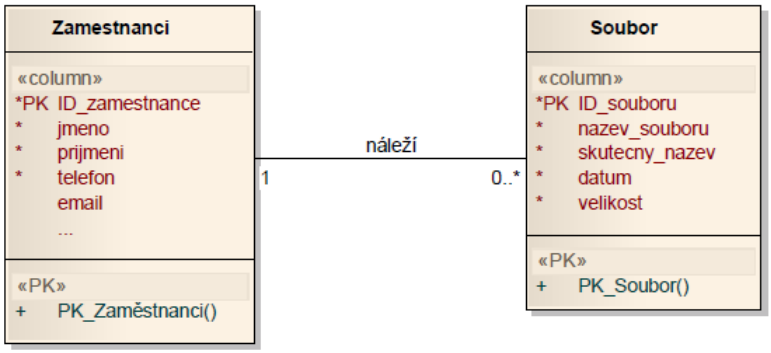
4.4 Návrh databáze

Schéma návrhu databáze si můžeme prohlédnout na E-R diagramu, který najdeme v příloze C. E-R diagram neobsahuje jednu tabulku. Konkrétně se jedná o tabulku `Soubor`.

Tato tabulka slouží k uložení informací o souboru. V E-R diagramu není zaznamenána z důvodu, že pro úplnost by bylo nutné přidat závislosti téměř na všechny ostatní tabulky. Tím by nám vznikl zcela nepřehledný E-R diargam. Další důvodem nezaznamenat tabulku bylo, že se nejedná ani tak o návrh databáze vycházející z analýzy požadavků, jako spíše o konkrétní implementaci řešení. Strukturu této tabulky můžeme vidět na obrázku 4.1. Tabulka `Soubor` má s tabulkou `Zamestnanci` vztah 1:M. Tento vztah jsme schopni uskutečnit v databázi. Ovšem tabulka `Soubor` nemá pouze tuto vazbu, má také vazby na další tabulky, kterými jsou `Dodavatel_potreby`, `Stroje`, `Zbozi`, `Vozidla`, `Servisni_naklady`, `Firmy_zamestnanci_externi`, `Vozidla_ext`, `Zamestnanci`, `Nemocenska`, `Dovolena`, `Sluzebni_cesta`, `Dodavatel`, `Objednane_zbozi`. Jelikož může mít jeden záznam vztah k více souborům, nemůžeme přidat atribut soubor, který ponese hodnotu primárního klíče z tabulky `Soubor`, k danému záznamu. Reprezentovat tyto vztahy by bylo možné pomocí přidání atributů do tabulky `Soubor`. Každý atribut by poté nabýval buď hodnoty `NULL` - v případě, že by soubor nenáležel k dané tabulce, anebo hodnoty primárního klíče v záznamu tabulky, ke které by daný záznam v tabulce náležel. V našem případě by se jednalo

o přidání třinácti nových atributů. Další způsob reprezentace vztahů, by bylo přidat do tabulky `Soubor` dva atributy. Jeden by nesl hodnotu primárního záznamu, ke které se váže daný soubor a druhý atribut by nesl jméno tabulky, které se týká hodnota primárního klíče v prvním přidaném atributu. Tato řešení nebyla zvolena. Bylo zvoleno řešení, které vychází z reprezentace vztahu M:N v databázi. Pro každou tabulku, z výše vyjmenovaných třinácti tabulek, byla přidána pomocná tabulka. V této tabulce je složený primární klíč, který se skládá z hodnot primárního klíče tabulky `Soubor` a tabulky, která obsahuje záznam, k němuž se vztahuje záznam v tabulce `Soubor`.

Jako příklad: k určitému zaměstnanci, uložíme soubor se smlouvou. Pomocná tabulka se tedy jmenuje `Zamestnanec_soubory`, jejíž primární klíč je složen z primárního klíče daného zaměstnance a primárního klíče daného souboru.



Obrázek 4.1: Náhled na tabulku Soubor se vztahem na tabulku Zaměstnanci.

Dále u tabulek `Dodavatel`, `Dodavatel_potreby`, `Firmy_zamestnanci_externi`, `Objednane_zbozi`, `Sarze`, `Stroje`, `Vozidla`, `Vozidla_ext`, `Zamestnanci`, `Zbozi` není uveden atribut `aktivni`. Tento atribut slouží ke „smazání“ položky v databázi. Podrobnější vysvětlení, jakým způsobem se mažou záznamy z databáze, je uvedeno v kapitole 5.7.

Kapitola 5

Implementace

V této kapitole jsou uvedeny použité technologie, při vývoji informačního systému. Dále je zde uvedeno, jakým způsobem jsem některé, z mého pohledu, zajímavé věci implementoval.

5.1 Použité technologie

Vývoj informačního systému probíhal na verzi PHP 5.5.29. Jako databázový systém bylo zvoleno MySQL, který byl použit ve verzi 5.6.27. Pro správu databáze bylo použito rozhraní phpMyAdmin ve verzi 4.5.1. Informační systém byl napsán pomocí Nette Frameworku ve verzi 2.3.8. Po designové stránce byl využit Bootstrap framework ve verzi 3.3.6.

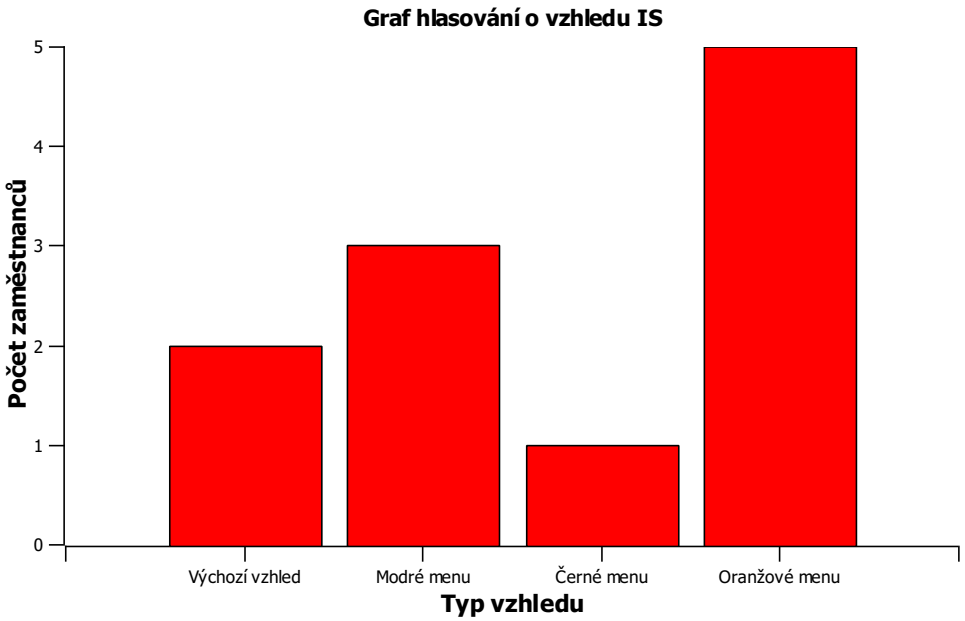
5.2 Uživatelské prostředí

Úvodní prostředí informačního systému můžeme vidět v příloze D, které se skládá z navigační lišty, tlačítka pro přihlášení a banneru, který se vztahuje ke společnosti Vinicola s.r.o. Vzhled informačního systému byl sladěn do oranžové barvy, přičemž byl zvolen jednoduchý (spíše čistější) design. Tato barva se nejvíce blíží vzhledu webových stránek společnosti a také nejvíce vyhovovala zaměstnancům firmy. Zaměstnancům společnosti bylo nabídnuto několik barevných kombinací, které vycházely ze stylu dostupných pro Bootstrap, a hlasováním měli vybrat styl vzhledu. Výsledek hlasování můžete vidět na obrázku 5.1. Jsou zde uvedeny pouze vzhledy, které dostaly alespoň jeden hlas.

Barevnou kombinaci Bootstrapu je možné změnit pouze nahrazení souboru `bootstrap.min.css` za jiný soubor se stylem. Díky tomu bylo možné nabídnout zaměstnancům funkční ukázku vzhledu informačního systému. Zaměstnanci si tedy mohli vyzkoušet práci s informačním systémem v různé barevné kombinaci. Což umožnilo mnohem objektivněji zvolit cílový vzhled informačního systému.

5.3 Přihlašování uživatele

Uživatel se dostane k přihlášení do informačního systému po stisknutí tlačítka `Přihlášení`. Otevře se modální okno, které umožňuje zadat přihlašovací jméno, heslo a navíc zaškrtnout `Zapamatuj přihlášení`. Vzhled stránky při přihlašování najdeme v příloze E. Přihlašovací formulář podporuje Live Validaci. Přihlašovací jméno i heslo musí být vyplněno, dále obě položky musí mít minimálně pět znaků. Bez tohoto nedojde k odeslání formuláře. Podrobnější informace k Live Validaci jsou uvedeny v kapitole 5.6.



Obrázek 5.1: Graf hlasování o vzhledu informačního systému.

Samotné heslo v databázi je uloženo jako výsledek hashovací funkce, která vrací řetězec o délce šedesáti znaků. Při procesu autentizace je využit autentifikátor ze Sandboxu. Byl upraven tak, aby vracel, při správně zadaném heslu, id zaměstnance, jeho roli v systému, jméno a příjmení. Zároveň ověřuje, zda nedochází k přihlášení z již deaktivovaného přihlašovacího jména.

5.3.1 Autentifikátor

Autentizace je proces, při kterém dochází k ověření, že přihlašovaný uživatel je opravdu uživatel, za kterého se vydává. V tomto informačním systému se provádí autentizace pomocí uživatelského jména a hesla.

Přihlašování funguje následovně. Autentifikátor na základě zadaného uživatelského jména a pokud zaměstnanec nemá atribut `aktivni` nastaven na `FALSE`, vybere z databáze záznam o uživateli. Následně porovná pomocí funkce `verify()`, zda došlo ke shodě. Funkce `verify()` je dostupná v `Nette\Security\Passwords`. Pokud dojde ke shodě hesel, autentifikátor vrací výše zmíněné údaje. V opačném případě vyvolá výjimku podle toho, zda nebyla nalezena identita uživatele, anebo zda neodpovídalo heslo.

Po úspěšném přihlášení uživatele do systému, se zobrazí menu, které obsahuje položky, podle toho jaké má uživatel oprávnění. V příloze F je ukázáno, jak vypadá menu pro přihlášení uživatele. V této ukázce je uživatel přihlášený v roli `boss`.

Pokud systém umožňuje přihlášení, musí také umět odhlášení. Po kliknutí na tlačítko `Odhlášení`, dojde k zavolání funkce `actionOut()`, kterou najdeme v `HomepagePresenter`. Funkce provede odhlášení uživatele, odeslání flash zprávy o úspěšném odhlášení a přesměrování na základní stránku (presenter). Druhou možností je odhlášení po časové neaktivitě. Pokud uživatel vybere `Zapamatuj přihlášení`, dojde k nastavení odhlášení za 12 hodin. Pokud tomu tak neučiní, dojde k odhlášení za 50 minut.

5.3.2 Autorizátor

Autorizace je proces, při kterém dochází k ověření, zda má uživatel dostatečné práva k provedení akce. Může se jednat např. o zobrazení určitých záznamů, editaci záznamu, smazání souboru.

Role nám umožňují přesnější řízení oprávnění. Není potřeba u každé akce nastavovat oprávnění pro každého uživatele. Z toho plynou problémy, pokud bychom měli více uživatelů. Kód by se začal stávat nepřehledný a zbytečně by docházelo k opakování částí kódu. Díky rolím můžeme rozdělit uživatele do skupiny, které následně můžeme využívat při definování přístupu k akci. Na role dále navazuje tzv. Permission ACL. Jedná se o definování přístupového seznamu. Nette Framework obsahuje autorizátor. Přístupový seznam najdeme v souboru `Authorizator.php`.Jedná se o třídu `Nette\Security\Permission`. Díky tomu můžeme definovat role (skupiny uživatelů), jednotlivé presentery a následně akce - metody, které presenter obsahuje. Autorizátor se už za nás postará, aby přístup k dané akci měl pouze povolený uživatel. V informačním systému jsou použité celkem čtyři role.

- `boss` - Role určená pro ředitele, nebo zástupce ředitele. Dovoluje provádět v systému veškeré akce.
- `warehouseman` - Role určená pro skladníky. Tato role nemá oprávnění editovat ani mazat položky.
- `secretary` - Role určená pro administrativní pracovnice, případně ekonomického pracovníka. Umožňuje provádět velmi podobné akce jako role `boss`. Pouze s tím rozdílem, že nemá oprávnění editovat citlivé údaje. Může se jednat např. o editaci zaměstnanců, nebo přidání nového zaměstnance.
- `other` - Role určená např. pro uklízečku, nebo brigádníky. Dovoluje pouze přístup k objednání obědu, změně hesla a využití generování QR kódu.

Pokud bychom se v budoucnu rozhodli přidat další roli, tak stačí pouze ji definovat v autorizátoru a nadefinovat akce - metody presenteru, ke kterým má přístup. Příklad jednoduchého dodefinování role, presenteru a pravidla je uveden v příkladu zdrojového kódu 5.1. Na prvním řádku provedeme definování nové role uživatelů. Druhý řádek nám umožní definovat nový presenter do přístupového seznamu. Poslední řádek slouží k přiřazení akce presentru k roli uživatele. Tím docílíme povolení přístupu k provedení dané akce.

Zdrojový kódu 5.1: Příklad definování nového oprávnění

```
$permission->addRole('vendor');
$permission->addResource('Wine');
$permission->allow('vendor', 'Wine', 'show');
```

5.4 Struktura aplikace

Při implementaci informačního systému jsem se snažil dodržovat architekturu MVP, která byla popsána v kapitole 2.6.4.

Aplikace obsahuje tedy model, který se stará o získání dat z databáze, případně provede zpracování dat a vrací je presenteru. Tento model je tvořen abstraktní třídou *BaseManager*,

které je umístěna v souboru `BaseManager.php`. Tato třída obsahuje funkce pro získání dat z databáze, pro seřazení dat, nastavení hodnoty `aktivni`, vložení nových hodnot, aktualizaci hodnot, anebo pro formátování data. Dále má přímý přístup do databáze, který se jí předává pomocí konstruktoru. Dále obsahuje proměnnou `protected $database`. Tato proměnná slouží k nastavení jména tabulky. Všechny výše uvedené funkce poté pracují s tabulkou, jejíž název si vezmou z této proměnné. Téměř každý presenter má poté v modelu svou třídu, která rozšiřuje abstraktní třídu *BaseManager*. Struktura takové třídy může vypadat, tak že v ní pouze nastavíme do proměnné jméno tabulky, se kterou daný presenter bude pracovat. Dále můžou být v této třídě definovány funkce modelu, se kterýma pracuje pouze tento konkrétní presenter. Funkce v této třídě většinou pracující se složitějšími dotazy do databáze, které jsou unikátní pro určitou tabulku, případně spojení tabulek. Z tohoto důvodu nejsou tyto funkce definovány v abstraktní třídě *BaseManager*.

Presentery se starají o určitou část v informačním systému. Jeden presenter např. zabezpečuje obsluhu vozidel, druhý zase zaměstnanců. Opět se zde setkáváme s abstraktní třídou *BasePresenter*, která se nachází v souboru `BasePresenter.php`. Tato abstraktní třída dostává v konstruktoru objekt `user`, který představuje právě přihlášeného uživatele. Dále máme v abstraktní třídě funkci, která v případě neautorizovaného přístupu k akci, přesměruje na určitý pohled, který obsahuje informace o tom, co se stalo. Funkce pro ověření přihlášení uživatele, formátování data, které se vkládá do databáze. Najdeme zde také funkci (`formatDate()`), která ověří, zda zadané datum existuje nebo ne. K tomuto účelu využíváme funkci PHP `checkdate()`, která zohledňuje i přestupné roky. Můžeme zde narazit také na funkci `formatDateDatabase()`. Jedná se vlastně o obdobu funkce `formatDate()`. Pouze s tím rozdílem, že se nekontroluje existence data. Používá se k formátování výpisu data, při nastavování výchozí hodnoty v editačním formuláři. Poslední funkce (`accessLatte()`), která je v této třídě slouží k dotazování na přístupový seznam ze šablony, který je popsaný v kapitole 5.3.2. Toto je jediná funkce presenteru, která se přímo volá se šablony.

Každý presenter je svázán s pohledem, anebo skupinou pohledů. Tyto pohledy se starají o vykreslování určitých akcí a také volají v případě interakce uživatele metody presenteru. Jedná se např. o reakci při odeslání formuláře, nebo přesměrování v případě stisknutí tlačítka. V pohledech je pro vykreslení použit framework Bootstrap. Nedochází zde tedy k žádné práci s databází a starají se pouze o vykreslení dat, které jim poskytne presenter.

5.5 Práce s databází

Nette Framework nám poskytuje velmi kvalitní nástroje pro práci s databází. Pro začátek je potřeba nastavit přistup do databáze. To se provede v aplikační konfiguraci v souboru `config.local.neon`. Zde nastavíme položky jako dsn, user, password. Díky tomu umožníme Nette frameworku vytvořit připojení s databází, které mohou využívat další třídy.

Základní funkcionalitu nám poskytuje třída `Nette\Database\Context`. Tato třída poskytuje velmi užitečnou metodu `query()`. Díky ní můžeme jednoduše psát MySQL dotazy. To se hodí v případě složitějších MySQL dotazů, kdy propojujeme více tabulek a získáváme z nich požadované informace. Z mého pohledu je takový zápis mnohem přehlednější, než v případě použití pokročilejší vrstvy `Nette\Database\Table`. Příklad složitějšího MySQL dotazu je uveden v ukázce zdrojového kódu 5.2.
MySQL dotaz má za úkol získat jméno, příjmení a ID zaměstnanců, kteří jedou na služební cestu. Dotaz celkově zpracovává informace ze tří tabulek: `Zamestnanci`, `Sluzebni_cesta` a pomocné tabulky `tabzamestnanec_cesta`, která reprezentuje vztah M:N. Spojení tabulek se provede pomocí příkazu `JOIN`. Nejdříve k tabulce `Zamestnanci` se připojí tabulka

Zdrojový kódu 5.2: Příklad složitějšího dotazu pomocí třídy Context

```
$this->database->query("SELECT Z.jmeno, Z.prijmeni, Z.
    ID_zamestnanec
                FROM tabzamestnanec AS Z
                JOIN tabzamestnanec_cesta AS ZC ON Z.ID_zamestnanec
                    = ZC.zamestnanec_id
                JOIN tabsluzebni_cesta AS SC ON ZC.sluzebni_cesta_id
                     = SC.ID_sluzebni_cesta
                WHERE SC.ID_sluzebni_cesta = ?", $oneValue->
                    ID_sluzebni_cesta)->fetchAll();
```

`tabzamestnanec_cesta` , následně se připojí další tabulka `Sluzebni_cesta`. Podmínka `WHERE` určuje, že se vyberou záznamy pouze takové, které odpovídají jednomu konkrétnímu primárnímu klíči v tabulce `Sluzebni_cesta`, čili jedné konkrétní služební cestě. Po dokončení dotazu je zavolána metoda `fetchAll()`, která načte všechny vrácené záznamy a atributy, které lze následně procházet pomocí cyklu `foreach`.

Pokročilá databázová vrstva `Nette\Database\Table` zjednodušuje dotazy do databáze. V tomto informačním systému je využívána k jednodušším dotazům. Příklad použití této vrstvy je v příkladu zdrojového kódu 5.3. Nad databázovým připojením stačí zavolat metodu `table()`, v které zadáme jméno tabulky, a následně nám vybere všechny záznamy v dané tabulce. V ukázce je dále metoda `where()`, která má stejný význam jako podmínka `WHERE` v MySQL. Ve výsledku tedy dojde k výběru všech záznamů v tabulce `Stroje`, které jsou aktivní.

Zdrojový kódu 5.3: Příklad jednoduchého dotazu pomocí třídy Tablet

```
$this->database->table(''tabstroje'')->where('aktivni', TRUE);
```

5.6 Formuláře a jejich Live Validace

Formuláře tvoří velkou část aplikace. Bez nich by nebylo možné přidávat nové záznamy, ani je editovat. Formuláře v informačním systému používají takzvané „továrničky“. Formulář je tedy tvořen tovární třídou, kterou můžeme použít ve více presenterech. Definice formuláře se provede pouze v tovární třídě. Vytvoření takového formuláře se provede zavoláním metody `create()`. Pokud provedeme změnu v tovární třídě, změny se projeví všude, kde byl tento formulář vytvořen pomocí „továrničky“. To je rozdíl oproti tomu, kdybychom formulář vytvářeli jeho definicí ve více presenterech nebo metodách. Museli bychom potom změnu provést v každé jeho definici.

Způsob jakým můžeme provést vytvoření formuláře pomocí „továrničky“, je uveden v ukázce zdrojového kódu 5.4. V této bakalářské práci jsou použité dva způsoby vytvoření formuláře za pomoci „továrničky“.
První dva řádky zdrojového kódu 5.4 jsou ukázka, jakým způsobem se vytvoří formulář pomocí „továrničky“, jestliže konstruktor v tovární třídě očekává parametry. Tím může být přístup k databázi. Toho je využito, pokud potřebujeme do formuláře načíst data databáze.

Zdrojový kódu 5.4: Příklad vytvoření formuláře pomocí „továrničky“

```
/** @var GoodsOrderedFormFactory @inject */
public $factory;
$form = $this->factory->create();
//nebo
$form = (new PasswordFormFactory())->create();
```

Jedná se třeba o formulář přidání objednaného zboží u dodavatele. V takovém formuláři načítáme seznam dodavatelů, ze kterých si jednoho vybereme. V takovém případě potřebujeme získat závislosti a to je provedeno komentářem nad veřejnou proměnnou `$factory`. Do této proměnné se předává instance třídy, která je definována v rámci komentáře, v našem případě se jedná o třídu *GoodsOrderedFormFactory*. Tento způsob předávání závislostí je specifický pro Nette Framework. [9] Poslední řádek v ukázce zdrojového kódu 5.4 slouží k vytvoření formulář pomocí „továrničky“, jehož tovární třída neočekává žádné parametry. V našem případě třída nemá konstruktor. Dojde k vytvoření formuláře z tovární třídy *PasswordFormFactory*.

5.6.1 Live Validace

Nette Framework poskytuje velmi propracovanou podporu formulářů, včetně jejich validace. Tato validace bohužel probíhá až po odeslání formuláře. Validační pravidla se definují hned za prvky formuláře. Další možností jsou validační funkce, které dovolují mnohem důkladnější validaci prvků, třeba i navzájem závislých. Validační pravidla můžou kontrolovat třeba jednoduchou závislost prvků na sobě, kontrolovat typ prvku (zda se jedná např. o číslo), nastavit aby prvek byl povinný, anebo omezit délku zadávaných dat. Nette Framework obsahuje základní validaci pomocí JavaScriptu. Při tomto typu validace dojde, po stisknutí tlačítka pro odeslání formuláře, k odchytnutí této události validačním skriptem, který provede kontrolu validačních pravidel. Pokud nějaký prvek nesplňuje zadané pravidlo, tak dojde k zobrazení ALERT okna, které obsahuje nadefinovaný popis chyby.

Live Validace umožňuje provádět validaci formuláře v reálném čase. Jakmile uživatel vloží data do prvku formuláře, ihned vidí, zda zadal data ve správném formátu. Validace se tedy neprovádí až v okamžiku odeslání formuláře, ale průběžně v době vyplňování formuláře. Jak taková validace vypadá pro uživatele je vidět v příloze G, kde je předvedena na přihlašovacím formuláři. Jak je vidět, uživatelské jméno i heslo musí mít aspoň pět znaků. Uživatel zadal pouze čtyři znaky, a proto formulář není validní. Rámeček prvků formuláře, které neprošly validací, se obarví červenou barvou a pod ním se objeví červeným nápisem, k jaké chybě došlo. Jakmile uživatel svou chybu opraví (tak aby splňoval validační pravidla), chybová hláška ihned zmizí. Tímto způsobem jsou ošetřeny všechny formuláře v informačním systému.

5.6.2 Implementace Live Validace

Nette Framework neobsahuje implementaci Live Validace, proto je možné využít již hotové řešení od jiných autorů. V tomto informačním systému je použitá Live Validace od autora Robyer [27]. Obsluha validačních pravidel se nachází v souboru `live-form-validation.js`. Při Live Validaci se využívají validační pravidla, které poskytuje Nette Framework a sa-

motná validace probíhá pomocí AJAXu. Pro správnou funkčnost, je potřeba do vygenerované HTML stránky doplnit meta informace, které můžeme zpracovat pomocí JavaScriptu.

Zajištění vygenerování meta informace do HTML tagu, je obsaženo v souboru `ValidatedForm.php`. Je zde třída *ValidatedForm*, která rozšiřuje třídu *TextInput*, od níž podědíme funkci `getControl()`, zároveň také implementuje `ISignalReceiver`. Validační pravidla jsou uložena v meta tagu *data-nette-rules*. Získáme tedy tyto informace a následně se zpracují pomocí vlastní serverové validace. Nette formulářové komponenty neobsahují běžnou funkčnost standartního Controlu. Proto nelze použít např. zavolání signálu (handle), nebo metodu `link()`. Tohle lze obejít pomocí rozšíření Nextras/Forms, kdy při použití `ComponentControlTrait` dojde k nakopírování funkcionality standartního Controlu. Potom už není problém použít běžné metody, jakými disponuje Control. Funkce `getControl()` dále využívá signál `handleValidatedForm`, který řídí obsluhu jednotlivých validačních pravidel. Signál si nejdříve nastaví proměnnou `$validni` na `TRUE`. Pokud se během zpracování pomocí JavaScriptu zjistí, že některé validační pravidlo není splněno, tak se nastaví proměnná `$validni` na `FALSE`. Následně se výsledek zpracování, případně chybová zpráva, odešle AJAXem.

Samotné zpracování validačních pravidel řeší JavaScript, který se nachází v souboru `MeFormValidation.js`. Odešle se AJAXový Request, ve kterém nahradíme řetězec `__replace_me__` za hodnotu, kterou chceme validovat. Validace musí být prováděna asynchronně. Z důvodu, že nemůžeme přerušit zadávání do formulářových prvků, do doby než se provede validace hodnoty v předešle zadaném prvku. Soubory `ValidatedForm.php` a `MeFormValidation.js` byly převzaty a upraveny od autora Jana Škráška [23].

5.6.3 Získání a zpracování dat z formuláře

Vytvoření formuláře se provádí v metodě `createComponent....()`. V této metodě se přidá také obslužná činnost při odeslání formuláře. To můžeme vidět v ukázce zdrojového kódu 5.5. Jako druhý parametr pole, se zadá název funkce, která provede zpracování zadaných dat z formuláře.

Zdrojový kódu 5.5: Příklad přesměrování po úspěšném odeslání formuláře

```
$form->onSuccess[] = array($this, 'completedSuccess...');
```

V obslužné funkci formuláře, se získají zadané data ve formuláři, které mohou být dále zpracovávány. Může se jednat např. o úpravu dat do požadovaného formátu pro databázi, o kontrolu existence data, anebo mohou být data uloženy do databáze. Data z formuláře získáme pomocí příkazu `$form->getComponent('znacka')->getValue()`. Kde ve funkci `getComponent()` zadáme název prvku formuláře, ze kterého chceme získat hodnotu. Funkcí `getValue()` provedeme získání hodnoty. Pokud by prvek formuláře byl např. select box, tak bychom místo funkce `getValue()` použili `getSelectedItem()`. Funkce vrátí vybraný prvek.

5.7 Smazání záznamů v databázi

Smazání záznamu v databázi se provádí z pohledu aplikace dvěma způsoby. Musíme rozlišovat, zda chceme záznam opravdu trvale smazat z databáze, anebo zda jej chceme pouze skrýt ve výpisu, aby se tedy tvářil jako smazaný. Uživatel o zvolené metodě vůbec netuší,

vše se odehrává na pozadí. Oba použité způsoby jsou reprezentovány stejným tlačítkem a v danou chvíli je vždy dostupný pouze jeden ze způsobů.

5.7.1 Trvalé smazání záznamu

Trvalé smazání záznamu z databáze provádíme, pokud nechceme zachovat historii, nebo pokud pro nás má daný záznam jen omezenou časovou platnost. Příkladem mohou být obědy. Jídelníček má pro nás smysl pouze po dobu dvou týdnů. Proč tedy uchovávat záznamy déle. Podobný případ nastává u nemocenské dovolené, služební cesty, nebo dovolené. Pokud takový záznam chceme smazat, tak to je z důvodu, že došlo ke zrušení dané události. Proto je tedy zbytečné takový záznam dále uchovávat. Funkcí, které provádějí smazání záznamu v databázi, je v informačním systému více. Všechny ale pracují na stejném principu, který je předveden v ukázce zdrojového kódu 5.6. Základem je vždy provedení v databázi operace `DELETE`. V té se pouze určí tabulka, v které se nachází záznam a hodnota primárního klíče záznamu, kterou požadujeme smazat. Takto smazaný záznam už nelze obnovit zpět.

Zdrojový kódu 5.6: Princip trvalého vymazání záznamu z databáze

```
$this->database->query("DELETE
                        FROM $table
                        WHERE $table.$column = $id");
```

5.7.2 Dočasné smazání záznamu

Ke smazání záznamu, stylem skrytí položky ve výpisu, slouží v tabulkách atribut `aktivni`, který je datového typu `BOOLEAN`. Při vytvoření nového záznamu je automaticky nastavena hodnota na `TRUE`, čímž je záznam aktivní. Pokud chce uživatel smazat danou položku, zavolá se funkce `setAktivni()`, která je dostupná v modelu. Funkce očekává ID mazané položky a hodnotu typu `BOOLEAN` - v tomto případě `FALSE`. Takto smazaný záznam se už neobjeví mezi dalšími aktivními záznamy v informačním systému. V aplikace je možnost zobrazit si všechny smazané položky dané typu (např. vozidla nebo stroje). Zůstává tedy zachována historie. Pro příklad bychom smazali zaměstnance ze systému. Následně bychom si chtěli zobrazit seznam souborů k nějaké položce. Pokud by soubor přidal v minulosti zaměstnanec, který byl již ze systému odstraněn, nevěděli bychom, kdo soubor přidal. Další možný případ využití může být, pokud chceme uchovat záznam, ale nechceme, aby se nám zobrazoval v běžném výpisu. Může se jednat o zboží, které už neprodáváme, ale informace o něm chceme mít stále v databázi. Navíc takto smazaný záznam, je možné opět obnovit. Stačí k tomu změnit hodnotu atributu `aktivni` na `TRUE`. Proto tedy funkce `setAktivni()` má parametr typu `BOOLEAN`. Podle toho jaká hodnota se tohoto parametru vloží, se provede buď obnovení, anebo smazání záznamu.

5.8 Nahrávání a mazání souborů

K většině položek lze uložit libovolné množství souborů. Ukázka rozhraní je vidět v příloze H. Nahoře se nachází ALERT upozornění, které oznamuje uživateli, zda se soubor nahrál v pořádku nebo došlo k chybě. Pokud je vše v pořádku, upozornění má zelenou barvu. Došloli k chybě, upozornění má červenou barvu. Zároveň upozornění obsahuje název souboru,

který se nahrával. Níže máme oblast, která slouží k vybrání nahrávaného souboru a následně tlačítko pro jeho uložení. Pod touto oblastí najdeme přehled všech nahraných souborů, které se vztahují k dané položce. Můžeme zde vyčíst název souboru, jeho velikost, datum a čas uložení souboru, anebo zaměstnance, který soubor nahrál. Dále zde najdeme tlačítko pro stažení souboru a v závislosti na oprávnění do systému je zde tlačítko pro smazání souboru.

Informace o souboru jsou ukládány do tabulky `Soubor`, jak bylo popsané v kapitole 4.4. Pokud požadujeme zobrazit soubory např. k určitému zaměstnanci, stačí kliknout na tlačítko *Seznam souborů*, které se nachází v detailu konkrétního zaměstnance. Poté dojde k přesměrování na presenter `File` a jeho metodu `actionNew()`. V této metodě se naleznou všechny soubory, které patří k danému zaměstnanci, pomocí funkce modelu `getArrayFiles()`. Následně dojde k přesměrování na šablonu, která se postará o vykreslení získaných dat.

Ukázka zdrojového kódu funkce `getArrayFiles()` je v příloze I.1. Pro lepší názornost uvedu princip této funkce, jestliže chceme zobrazit soubory, které se vztahují k určitému vozidlu. Nejdříve pomocí dotazu do databáze funkce získá všechny záznamy v tabulce `Soubor`, které se vztahují k primárnímu klíči našeho vozidla. Následně pomocí cyklu `foreach` iteruje přes jednotlivé získané záznamy. V tomto cyklu se iteruje dalším cyklem `foreach` přes jednotlivé atributy každého záznamu a zapisuje si je výsledného dvourozměrného pole (`$pole_result`). Kdy každý řádek v tomto poli odpovídá jednomu vrácenému záznamu a zároveň každý prvek v řádku pole, odpovídá jednomu atributu záznamu z databáze. Jakmile tento druhý cyklus `foreach` skončí, dotáže se znovu do databáze, konkrétně do tabulky `Zamestnanci`, ze které získá jméno a příjmení zaměstnance, který nahrál soubor. Tyto údaje opět uloží do pole `$pole_result` pomocí cyklu `foreach`. Nakonec vrátí celé pole, které už může zpracovat šablona (neboli pohled).

Nelze spoléhat na uživatele, že nebude nahrát soubory se stejným názvem. Proto bylo potřeba ošetřit duplicitu názvu souborů. Každý presenter v aplikaci pracující s takovými daty, ke kterým mohou být uloženy soubory, má vlastní složku na disku, do níž si ukládá soubory. Jelikož ani toto řešení nelze považovat za uspokojivé, do uložiště se proto ukládá soubor pojmenovaný náhodně vygenerovaným řetězcem o délce 35 znaků, pomocí funkce `random()`. Tuto funkci poskytuje třída `Nette\Utils\Strings`. Část funkce, která se stará o uložení souboru, je v ukázce zdrojového kódu 5.7. Nejdříve se získá přístup do session sekce *File*, v které se nacházejí potřebné údaje. Dále se z formuláře získají podrobnosti o nahraném souboru, obsahující jméno souboru, typ, velikost v bajtech, umístění v dočasné složce a příznak zda nedošlo k chybě. Následně dojde k získání celého jméno souboru, které se rozdělí na název a typ souboru. Vygeneruje se náhodný řetězec, který bude sloužit k pojmenování souboru v uložišti. Současně dojde k připojení původní koncovky souboru. Poté pokud se soubor nahrál v pořádku, se přesune pod novým názvem do koncového uložiště. Dále už následuje pouze přidání záznamu do databáze, kam se uloží mimo jiné původní název souboru, aby bylo možné jej uživateli nabídnout ke stažení pod původním názvem.

5.8.1 Stažení a smazání souborů

Při akci stažení nebo smazání souborů není potřeba vykreslovat další šablonu. Vystačíme si s aktuální šablonou a vyvoláním akce, která zpracuje požadavek. K tomu jsou v Nette frameworku určeny signály (metody `handle...()`). Více jsou signály popsány v kapitole 2.6.4. Tyto signály jsou uloženy v souboru `FilePresenter.php`.

Zdrojový kódu 5.7: Uložení souboru

```
$session_file = $this->session->getSection('File');

$file = $form['file']->getValue();
$file_name_ext = $file->getName();
list($nazev, $pripona) = explode('.', $file_name_ext);
$file_name = Strings::random(35).'.'.$pripona;

if ($file->isOk()){
  $file->move(WWW_DIR.'/file/'.$session_file->folder.'/'.
     $file_name);
}
```

Pro stažení souboru slouží signál `handleDownload()`. Ukázka zdrojového kódu obsahující tělo signálu je v ukázce 5.8. Signál očekává jako parametry primární klíč záznamu souboru, skutečný název souboru, tímto názvem se pojmenuje soubor při stahování, a vygenerovaný název, pod kterým je soubor uložen v uložišti. Ve funkci se nejprve vygeneruje cesta k souboru, která obsahuje také název souboru. Poté dojde k ověření, zda požadovaný soubor je k dispozici. Mohlo se stát, že soubor někdo smazal ručně. Poté by nebylo možné stáhnout soubor. Pokud soubor neexistuje v uložišti, odešle se ALERT zpráva s náležitým upozorněním a typem zprávy. Následně dojde ke smazání záznamu z databáze pomocí funkce `delete()` z modelu. Pokud je všechno v pořádku použije se Nette funkce `sendResponse()`, které očekává jako parametry cestu k souboru a název souboru, kterým se má pojmenovat souboru při stahování. V této metodě je potřeba v jakém formátu se odešlou požadované data. V našem případě se jedná o formát `FileResponse`. Tím říkáme, že se jedná o soubor. Pro příklad by mohl být další formát např. `JsonResponse` - což by se jednalo o JSON formát. Tímto server odešle soubor a prohlížeč uživatele zobrazí výzvu pro uložení souboru.

Zdrojový kódu 5.8: Stažení souboru

```
$session_file = $this->session->getSection('File');
$cesta = WWW_DIR.'/file/'.$session_file->folder.'/'.$name_random;

if(!file_exists($cesta)){
  $this->flashMessage('Alert zprava', 'warning');
  $this->model_tmp->delete($id);
  ...
}

$this->sendResponse(new Nette\Application\Responses\FileResponse(
   $cesta, $name_real));
```

Pro správnou funkčnost nahrávání souborů v informačním systému je potřeba upravit konfiguraci PHP. Informační systém má maximální povolenou velikost nahrávaného souboru

omezenou na 25MB. PHP má ve výchozím nastavení povolenou velikost 8MB. Podobná situace nastává u omezení velikosti odesílaných dat pomocí metody POST. Doporučuji tedy upravit hodnotu v parametru `upload_max_filesize` na velikost 25M. Což odpovídá 25MB. Dále upravit hodnotu v parametru `post_max_size` na velikost 33M. To odpovídá 33MB. Jelikož tento parametr v sobě zahrnuje velikost souboru + další data, tak jeho velikost musí být větší než hodnota parametru `upload_max_filesize`. Pokud bychom razantně zvýšili velikost nahraného souboru nad 25MB, bylo by vhodné upravit také parametr `memory_limit`. Tento parametr říká, kolik paměti si může aplikace zabrat. Jelikož nahrávané soubor procházejí přes přidělenou paměť, mohlo by dojít k tomu, že paměť bude vyčerpána a soubor se nepodaří úspěšně nahrát. Ve výchozím nastavení má PHP nastavenou hodnotu tohoto parametru na 128M, tedy 128MB. Tato hodnota, je pro naši velikost přenášených souborů dostatečná.

Pro smazání souboru slouží signál `handleDelete()`. Ukázka zdrojového kódu obsahující tělo signálu je v ukázce 5.9. Smazání souboru má v tomto případě dva kroky. Nejprve se pomocí funkce `delete()` z modelu, provede smazání záznamu v databázi. Následně pomocí funkce `unlink()` dojde k dokončení procesu smazání, odstraněním souboru v uložišti. Zbytek kódu je podobný jako v případě stahování souboru. Také je potřeba vygenerovat adresu k souboru. Kontrola na existenci souboru se zde už neprovádí.

Zdrojový kódu 5.9: Smazání souboru

```
$this->model_tmp->delete($id);
...
unlink($cesta);
...
```

5.9 Předávání parametrů a sessions

Hodnoty můžeme předávat buď pomocí parametrů metod, anebo pomocí sessions. V tomto informačním systému jsou zvoleny obě možnosti. Pokud se jedná o jednoduché předání např. dvou hodnot, které využiju pouze jedenkrát, tak jsou tyto hodnoty předány pomocí parametrů metody. Zároveň jsou viditelné v URL adrese. Příkladem může být zobrazení detailu položky. Kdy společně s přesměrováním na metodu `show()` se předá také parametr ID dané položky. Výjimku může tvořit navíc předání řídící proměnné. Jedná se o proměnnou, která svou hodnotou určuje na jaký presenter a metodu se budeme vracet. Příklad takového chování je v ukázce zdrojového kódu 5.10.

Na řádku číslo dva je ukázka volaní metody presenteru ze šablony. V makru `link` se nachází název preseneteru a volané metody (`Cars:show`). Následuje první parametr, který představuje ID vozidla, u kterého požadujeme zobrazit podrobnosti. Druhý parametr obsahuje v tomto případě hodnotu 1 nebo 0, podle toho zda se jedná o aktivní, anebo odstraněné vozidlo. Na pátek až devátém řádku je vidět volaná metoda presenteru, která může mít až tři parametry. Pokud parametr `$id_back` nezadáme, bude mít hodnotu `NULL`. V metodě dojde k předání parametrů do své šablony (show), která je využívá. Zápisem `$this->template->promenna` říkáme, že se jedná o proměnnou, která bude k dispozici v šabloně. Na řádku dvanáct až šestnáct je znázorněno zpracování předaných parametrů do šablony. Podle hodnoty parametru `$rozhodnuti` dojde k vygenerování patřičného popisu

Zdrojový kódu 5.10: Ukázka předávání paremetrů metodě

```
1  // Zdrojovy kod ze sablony
2  <a href="{link Cars:show $car->ID_vozidla, $car->aktivni}">
       Zobrazit podrobnosti</a>
3
4  // Zdrojovy kod v presentru
5  public function renderShow($id, $rozhodnuti, $id_back){
6    $this->template->rozhodnuti = $rozhodnuti;
7    $this->template->id_back = $id_back;
8    ...
9  }
10
11 // Zdrojovy kod v sablone Show
12 {if $rozhodnuti == 1}
13       <a n:href="Cars:">Seznam vozidel</a>
14 {elseif $rozhodnuti == 0}
15   <a n:href="Cars:deleted">Seznam odstranenych vozidel</a>
16 {/if}
```

v tlačítku (které slouží pro návrat zpět) a také k nastavení akce, která se má provést po jeho stisknutí. Pokud v tomto případě bude mít atribut `$rozhodnuti` hodnotu `TRUE`, tak se jedná o aktivní vozidlo, tudíž budeme přesměrování na seznam aktivních vozidel. Pokud by měl atribut `$rozhodnuti` hodnotu `FALSE`, tak se jedná o odstraněné vozidlo, tudíž budeme přesměrování na seznam odstraněných vozidel.

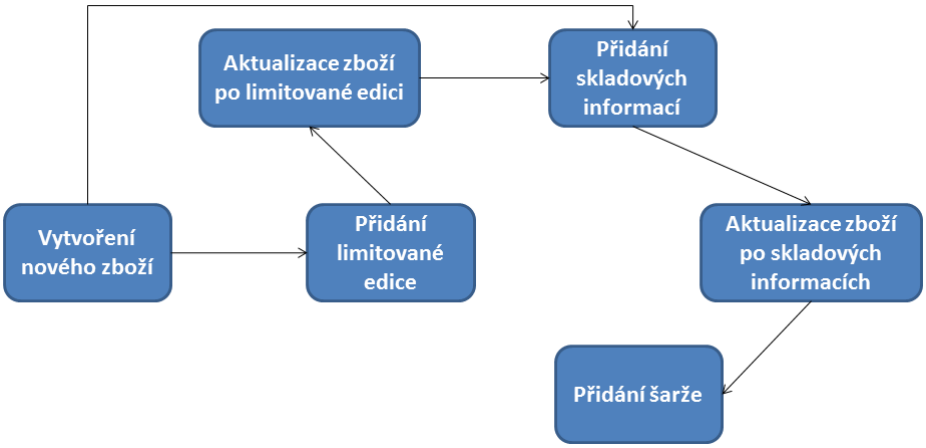
5.10 Sessions

Jiná situace nastává, pokud potřebujeme předávat více parametrů při víceúrovňovém formuláři. Případně je-li potřeba předat více parametrů najednou. K tomuto jsou použity sessions. Každý presenter používající sessions, má v sessions svoji sekci. Další vlastností je, že hodnoty předávaných dat nejsou vidět v URL adrese. Což je rozdíl oproti předávání hodnot pomocí parametrů metod. Zmínka o sessions byla už v kapitole zabývající se bezpečností Nette frameworku 2.6.1 a ve spojení se soubory v kapitole 5.8.

Příkladem víceúrovňový formulář nalezneme při přidávání nového zboží. Nejprve se objeví formulář, který slouží k uložení základních údajů o zboží. Po vyplnění a zpracování hodnot se do sessions v sekci Goods, uloží ID nově přidaného zboží. Poté dojde k přesměrování na presenter a metodu `LimitEdition:new`. Zde se zobrazí formulář sloužící k zadání informací o limitované edici zboží. Pokud formulář vyplníme a uložíme, tak se zpravují jeho hodnoty, do stejné sekce sessions se uloží ID limitované edice. Následně dojde k přesměrování na `Goods:update`. Zde se ze sessions vyberou oba ID záznamy a dojde k přidání ID limitované edice do záznamu daného zboží. Bylo by možné tyto údaje ID přenášet pomocí parametrů metod. Přenášet ale hodnoty, které mají být někam trvaleji zapsány, není moc vhodné. Proto je lepší raději využít sessions. Dále přidání nové položky pokračuje přesměrováním na `Stock:new`, kde se nachází formulář pro zadání skladových informací. Stejně tak, bychom byli sem přesměrování i v případě, že bychom přeskočili přidávání limitované

položky. Tento formulář už přeskočit nelze. Po jeho vyplnění a zpracování hodnot se uloží do sessions do sekce Goods ID nově přidaných skladových informací a dojde k přesměrování na `Goods:updateStock`. Zde stejně jako v předešlém případě, dojde k editaci záznamu položky zboží a přidání cizího klíče, odkazující na primární klíč nově přidaných skladových informací. Poté následuje přesměrování na poslední formulář a to do `BatchNumber:new`, který slouží k vytvoření nových šarže zboží. Zde se ze sessions sekce Goods získá ID zboží, kterého se týká nově vytvářená šarže a nastaví se jako výchozí hodnota do select box. Rovnou tedy vidíme, k jakému zboží vytváříme novou šarži. Poté dojde ke zpracování dat, jejich uložení do databáze a přesměrování na detail nově vytvořeného zboží.

Schéma kroků víceúrovňového formuláře je na obrázku 5.2.



Obrázek 5.2: Schéma kroků víceúrovňového formuláře pro vytvoření nového zboží.

5.11 Úprava vzhledu Bootstrapu

Souboru `bootstrap.css` definuje vzhled výchozí vzhled prvků z Bootstrapu. Soubor `bootstrap.min.css` je jeho odlehčenou variantou, ze které jsou odstraněny všechny přebytečné znaky, takže jeho zpracování je o něco rychlejší. Bylo nutné provést editaci souboru `bootstrap.min.css`, jelikož došlo k problému s prvkem `navbar-nav`. Editace tohoto souboru se neprovádí přímo, ale je nutné provést editaci hodnoty v souboru `variables.less` a následně tento soubor přeložit na soubor typu CSS. U prvku `navbar-navdojde` při určitém zmenšení k jeho „zborcení“, tak aby možnost použití menu zůstala zachovaná. Ve výchozím nastavení dojde ke „zborcení“ (parametr `@grid-float-breakpoint`), při velikosti obrazovky menší než `@screen-sm-min`. Což odpovídá velikosti 768 pixelů. Bohužel toto nastavení je limitováno určitým počtem tlačítek, jenž lze použít. Také záleží na délce názvu v tlačítkách. Co se stane, pokud máme více tlačítek a nedojde ke „zborcení“ menu, je předvedeno na obrázku 5.3. Na horní části je navigační lišta v normálním stavu, při dostatečné šířce okna. Na dolní části je znázorněna situace, kdy se zmenší velikost okna, ale zároveň je větší než 768 pixelů, takže nedojde ke „zborcení“ navigační lišty. Pro editaci a následné přeložení souboru `variables.less`, jsem použil online editor Live LESS Theme Customizer [1]. Velikost proměnné `@grid-float-breakpoint` byla nastavena na `@screen-md-min`, což odpovídá 992 pixelům. Tato úprava sebou nese jedno úskalí a to sice, pokud se provede ak-

tualizace na vyšší verzi Bootstrapu, bude nutné opět upravit soubor `bootstrap.min.css`, jelikož bude obsahovat výchozí nastavení „zborcení“ navigační lišty.

Obrázek 5.3: Náhled na menu.

5.12 Jídelníček

Informační systém umožňuje zaměstnancům zobrazit jídelníček na aktuální týden, na další týden, objednat obědy na další týden, zobrazit na aktuální týden objednané obědy a při patřičném oprávnění, zobrazit seznam objednaných obědů. Náhled jak vypadá rozhraní objednání obědů, je možné vidět v příloze K.1. Rozdíl oproti rozhraní aktuálního týdne je v tom, že neobsahuje tlačítka pro objednání a zrušení objednaného obědu. Zaměstnanec si na každý den, může objednat pouze jeden oběd. Pokud již má na daný den objednaný oběd, systém mu nedovolí oběd na daný den další, nejdříve musí zrušit objednávku. V každém dni se objednaný oběd podbarví zelenou barvou. V případě, že dojde k chybě, je zaměstnanec informován ALERT oknem.

Data vztahující se jídelníčku mají význam pouze po dobu dvou týdnů. Navíc informační systém zobrazí pouze jídelníček na dva týdny. Z tohoto důvodu, je zajištěno automatické smazání starých dat po vložení nových. Nový jídelníček je zadáván pomocí víceúrovňového formuláře. V první úrovni se zadají informace, v jakém rozsahu data jídelníček platí. Poté následuje druhá úroveň, která vždy představuje nabídku jídel na jeden den. Po odeslání formuláře, dojde ke zpracování zadaných hodnot a jejich uložení do databáze. Následně jsme přesměrování opět na druhou úroveň formuláře. Pokud už nechceme přidávat nabídku jídel na další den, můžeme normálně z formuláře odejít pomocí tlačítka `Odejít`.

5.12.1 Výpis jídelníčku na jeden týden

Při zobrazení nabídky jídel na daný týden, se nejprve v presenteru zavolá funkce `getWeek()` z modelu (jedná se o soubor LunchManager.php). V této funkci se zjistí aktuální datum pomocí PHP funkce `StrFTime()`. Následuje dotaz do databáze, kde se vybere záznam z tabulky `Obed_tyden`, tak aby získané datum pasovalo do vybraného rozpětí data daného záznamu. Poté presenter opět volá funkci z modelu a to `menu_one_day()`, která získá přehled nabízených jídel na jeden den. Tato funkce je volána pro všechny dny v týdnu. Funkce nejdříve získá z databáze všechny jídla pro určitý den. Poté jsou pomocí cyklů `foreach` uloženy do dvourozměrného pole. Přičemž každý řádek v poli odpovídá jednomu záznamu z databáze a zároveň jeden sloupec odpovídá jednomu atributu záznamu z databáze. Ke každému záznamu jídla v dvourozměrném poli se doplní informace, zda si zaměstnanec, který si prohlíží jídelníček, dané jídlo objednal. Nakonec se pro každé jídlo získá z databáze seznam alergenů, které jsou následně uloženy do pole. Poté už pouze šablona vykreslí předané data z presenteru. Před zavoláním funkce `menu_one_day()` pro další den, je potřeba získat

datum následujícího dne. K tomu slouží funkce z modelu `cislo_dne_next()`. Této funkci se předá naposledy použité datum. Funkce si následně rozdělí datum na dny, měsíce a roky. Nejdříve iteruje den a ověří existence data pomocí PHP `checkdate()`. Pokud datum neexistuje, tak se dále iteruje měsíc, následuje rok. Funkce vrací následující datum po datu, které dostala jako parametr.

Velice podobný postup je, při zobrazení jídelníčku na další týden. Rozdíl je pouze v použité funkci, která určí patřičný týden. Zde se použije funkce `getNextWeek()`, která má pozměněný MySQL dotaz, tak aby našla záznam, který má větší začátek týdne, než je aktuální datum.

5.12.2 Přehled objednaných jídel na týden

Oprávněná osoba si může zobrazit seznam objednaných jídel na daný týden. Přehledně se zobrazí den, v rámci něj pouze objednané jídla a množství, kolikrát byly objednány. Zároveň je pro každé jídlo, možné zobrazit v modálním okně, kteří zaměstnanci si jej objednali. Stejně jako v případě zobrazení jídelníčku, je potřeba získat týden, kterého se to týká. Dále je nutné se vždy posunout na následující den. K získání informací, které jídla byly daný den objednány, slouží funkce z modelu `order_menu_day()`. Funkce nejdříve získá z databáze seznam objednaných obědů. Ke každému záznamu, dále získá počet objednání a následně seznam zaměstnanců, kteří si dané jídlo objednali. Všechny tyto údaje si uloží do dvourozměrného pole, které také vrací. Princip práce a struktura dvourozměrného pole je stejná, jako v předešlé podkapitole, zabývající se výpisem jídelníčku na jeden týden.

5.12.3 Objednání a zrušení jídla

Objednání i zrušení objednávky jídla mají na starost signály v presenteru. Konkrétně `handleOrder()` a `handleCancel()`. Oba signály pro práci s databází využívají funkce modelu. V případě signálu `handleOrder()` je provedeno vložení nového záznamu o objednávce jídla pomocí funkce `insertMyExt()`. U signálu `handleCancel()` se odstranění objednávky provede pomocí funkce `deleteLunch()`.

5.12.4 Vytvoření nového jídelníčku

Vytvoření nového jídelníčku má dvě úrovně. V první úrovni se nachází formulář, do kterého zadáváme datum od kdy a do kdy platí jídelníček. Po jeho zpracování a uložení dat do databáze, jsme přesměrování na druhou úroveň, která slouží k přidání seznamu jídel na jeden den. Získané hodnoty z formuláře se v prezenteru uloží do pole a to se předá ke zpracování funkci z modelu `insertOneLunch()`. Tato funkce postupně ukládá jednotlivé zadané hodnoty do databáze. Přitom využívá pomocné funkce `insertMyExt()` a `insertMyExtAuxiliary()`, které slouží k uložení informací o jednotlivých jídlech do databáze. Poté následuje použití funkce `insertAuxiliary()`, která má na starost vytvoření vztahu v databázi, mezi jídlem a použitými alergeny.

5.12.5 Automatické smazání jídelníčku

Databázová struktura jídelníčku obsahuje celkem sedm tabulek. Smazání dat je tedy o něco komplikovanější. Využívá se zde smazání dat pomocí databázové vrstvy a pomocí aplikační vrstvy. Smazání starého jídelníčku začne ve chvíli, kdy se vytvoří nový záznam v tabulce `Obed_tyden`. Tzn. jakmile se provede zpracování první úrovně víceúrovňového formuláře.

Zavolá se z modelu funkce `updateCountLunch()`. Tato funkce nejdříve spočítá, kolik se nachází záznamu v tabulce `Obed_tyden`. Pokud najde více než dva záznamy, tzn. že nejstarší záznam je v tuto chvíli už nepotřebný, získá seznam primárních klíčů obědů z tabulky `Obedy`, které patří k nejstaršímu záznamu z tabulky `Obed_tyden`. Čímž se zjistí, které obědy bude potřeba smazat. Následně bude pro každý oběd získán z databáze seznam jednotlivých jídel, ze kterých se skládá nabídka jídel na daný den. Jejich primární klíče se uloží do pole. Poté se smaže v databázi nejstarší záznam v tabulce `Obed_tyden`. Díky kaskádovému mazání v databázové vrstvě, dojde ke smazání záznamů také z tabulek `Obedy`, `Obed_polozka_obedu`. Přejde se opět do aplikační vrstvy, kde bude provedeno smazání jednotlivých položek obědů (z tabulky `Polozka_obedu`), jejíž primární klíče jsou uloženy v poli, pomocí cyklu `foreach`. Díky kaskádovému mazání dat, se smažou záznamy z tabulek `Zamestnanec_polozka_obedu` a `Alergeny_polozka_obedu`. V případě naposledy dvou zmíněných tabulek, se jedná o pomocné tabulky, které reprezentují vztah M:N mezi tabulkami `Zamestnanec`, `Polozka_obedu` a `Alergeny`, `Polozka_obedu`.

5.13 Implemetace QR kódů

Pro implementaci QR kódu v informačním systému byla zvoleno generování pomocí JavaScriptu a to pomocí knihovny *qrcode.js*, která byla popsána v kapitole 3.4. Nejdříve bylo potřeba připojit tuto knihovnu do informačního systému. Ukázka připojení knihovny do projektu, je předvedena v ukázce zdrojového kódu 5.11. Poté je potřeba připojit script, který se postará o správné generování, pomocí výše zmíněné knihovny. Ukázka skriptu je v příloze K.1.

Nejdříve se nastaví vlastnosti QR kódu. Jedná se o velikost výsledného obrázku v pixelech a míru chybové korekce dat. Úroveň chybové korekce dat, byla zvolena úroveň H, což odpovídá 30% z plochy kódu. Takový míra byla zvolena s ohledem na využití QR kódu. Jelikož ve většině případů bude umístěn na etiketě výrobku, bude využíván k uložení URL adresy. Tudíž byla upřednostněna vyšší pravděpodobnost naskenování kódu, před množstvím dat, které do něj lze uložit. Poté už následuje funkce, které se předá text, který má být v QR kódu zakódován. Následně se zavolá funkce z připojené knihovny, která se postará o vygenerování QR kódu.

Zdrojový kódu 5.11: Ukázka připojení knihovny

```
<script src="{$basePath}/js/qrcode.js"></script>
```

Hodnota, která má být uložena v kódu, se zadává do klasického formuláře, do prvku *textarea*. Obsluha formulářového prvku je odlišná podle toho, zda se jedná o generování QR kódu v záložce detailu zboží, anebo v záložce generátor QR kódu. Jednou z možností vygenerování QR kódu v detailu zboží, je po kliknutí do vstupního pole formulářového prvku. Tohle chování je z důvodu, že vygenerovaný QR kód se nikam neukládá, jelikož samotný QR kód je zbytečné ukládat do fyzického uložiště. Uživatel si jej uloží až v případě potřeby. Do databáze se uloží pouze řetězec, který byl v QR kód zakódován. Obsahuje-li daná položka zboží již uloženou hodnotu QR kódu, tato hodnota se zobrazí ve vstupním poli formulářového prvku. Jelikož je možné pouze zkontrolovat, jaký text byl zakódován, je zbytečné zároveň generovat i QR kód. V záložce generátor QR kódu, se QR kód začne generovat, až když uživatel začne do vstupního pole formulářového prvku psát text. V obou

případech chování, se QR Kód generuje po každém napsaném znaku. Je tedy možné průběžně sledovat, jak se QR kód mění.

Vygenerování obraz QR kódu je možné stáhnout ve formátu PNG. Standardně nabídnutý název obrázku je index.

Kapitola 6

Zhodnocení informačního systému

Informační systém byl průběžně testován vývojářem. Jednalo se zejména o ošetření zpracování chybných dat, které mohly být do formulářů zadány. Dále byla testována práce s databází, zda informační systém správně získává dat, ukládá, modifikuje a prezentuje. Informační systém byl také konzultován se zaměstnanci společnosti Vinicola s.r.o., díky kterým získal informační systém výslednou podobu a funkčnost.

Informační systém byl po vývojářském testování předán společnosti. Aktuálně probíhá, tzn. schvalovací testování, kdy je informační systém nasazen při reálném používání. Cílem je jednak předvedení systému zaměstnancům, jejich soužití se systémem, ale také případné odladění dalších chyb v systému. Pro testování v rámci společnosti, je informační systém nasazen na serveru, který má společnost k dispozici. Zaměstnanci společnosti považují systém za přínosný. Vzhledově se jim líbí jednoduchost a zároveň přehlednost systému. Jako přínos považují centralizaci údajů na jednom místu, s možností nahrávání neomezeného počtu souborů k důležitým položkám. Místo doteď používaných tabulek, ke které museli navíc mezi sebou sdílet. Zároveň jsou také velmi spokojeni s přehledným a jednoduchým objednáváním obědů.

Po schválení a případném doladění, bude informační systém plně nasazen. Zároveň budou vytvořeny patřičné uživatelské účty a databáze bude naplněna skutečnými daty. Z hlediska požadavků na funkce systému, informační systém splnil zadání i očekávání. Zároveň byl dodatečně zapracován požadavek na implementaci jídelního lístku, který ze strany společnosti vzešel, až v průběhu implementace samotného systému.

Celý informační systém byl navrhnut s ohledem na snadné rozšíření do budoucna. Postupně podle požadavků společnosti bude přidávána další funkcionalita, tak aby informační systém poskytoval co největší efektivitu a měl přínos pro společnost.

6.1 Návrhy na vylepšení

Během schvalovacího testování byli zaměstnanci společnosti požádání o vyjádření, co by v informačním systému zlepšili, nebo přidali. Zaměstnanci z pracovních pozic skladník a uklízečka, by další rozšíření nepřidávali. Byli spokojeni s aktuální podobou. Pracovníci na pozici administrativní pracovník, by uvítali všude dostupný formulář pro rychlé vyhledávání položek, řazení tabulek kliknutím na jejich záhlaví. Pracovníci na ostatních pozicích neměli aktuálně potřebu v informačním systému, provádět vylepšení.

Po konzultaci se zaměstnanci společnosti, je do budoucna plánováno rozšíření informačního systému o evidenci nákupu hroznů, výdajů a příjmů společnosti a paletové hospodář-

ství. Paletové hospodářství je vlastně evidence palet, která zahrnuje kolik palet si společnost Vinicola s.r.o. vypůjčila, kolik a komu půjčila palety, nebo kolik poškozených palet přijmula.

Kapitola 7

Závěr

Cílem bakalářské práce bylo seznámit se s principy tvorby webových aplikací, problematikou použití QR kódů ve webových aplikacích. Následně navrhnout databázi, databázový systém, implementovat samotný informační systém a připravit jej pro reálné nasazení ve společnosti Vinicola.s.r.o.

Práce je rozdělena na několik kapitol. A to sice na základní pojmy, práci s QR kódy, návrh infomačního systému, jeho implementaci a zhodnocení. V kapitole Základní pojmy, se věnuji popisu prostředků, které byly využity pro návrh a implementaci informačního systému. Jsou zde popsány principy UML, architektury MVC a MVP, Nette a Bootstrap frameworky. Kapitola práce s QR kódy pojednává o teoretické části QR kódů. Jsou zde uvedeny základní specifikace, přehled jakými způsoby lze vytvářet a číst QR kódy. V kapitole Návrh informačního systému jsou uvedeny požadavky na informační systém ze strany společnosti Vinicola s.r.o., návrhu databáze a profil společnosti. V kapitole Implementace je popsán způsob implementace jednotlivých částí informačních systému. Jedná se např. o formuláře, autentifikátor a autorizátor, práce s databází, nebo smazání souborů. V závěrečné kapitole Zhodnocení informačního systému je popsán způsob testování, hodnocení zaměstnanci společnosti, dále jsou zde uvedeny návrhy na vylepšení systému a jeho plánované rozšíření.

Součástí bakalářské práce jsou také zdrojové soubory informačního systému. Pro jeho zprovoznění je potřeba mít funkční server a použít databázový MySQL systém. Zdrojové soubory obsahují dva MySQL skripty. Jeden je pro vytvoření struktury databáze (soubor `MySQL - vytvoreni_databaze.sql` a druhý (soubor `MySQL - data.sql`) slouží pro nahrání testovacích dat. Soubor obsahující testovací data a předdefinovaný jídelníček na dva týdny. Aby se jídelníček zobrazil v informačním systému, je potřeba v tomto souboru upravit rozsah data v tabulce `Obed_tyden` na adekvátní datum. Dále je potřeba upravit data vkládaných záznamů do tabulky `Obedy`, tak aby odpovídaly rozsahům dat v předešle zmíněné tabulce. Pokud toto nebude provedeno a pokusíte se zobrazit jídelní lístky, informační systém vypíše hlášku, že pro daný týden není definovaný jídelníček.

Závěrem mohu říci, že se podařilo splnit vytyčené cíle, implementovat informační systém v takovém rozsahu, v jakém bylo společností Vinicola s.r.o. požadováno a do budoucna je naplánováno jeho rozšíření.

Literatura

[1] Live LESS Theme Customizer [online], version 3.3.6.
URL http://bootstrap-live-customizer.com/

[2] *Web Applications: What are They? What of Them?* [online]. acunetix, viděno: 2016-04-23; [cit. 2016-05-01].
URL http://www.acunetix.com/websitesecurity/web-applications/

[3] *Seznámení s Nette Frameworkem* [online]. Nette Foundation, 2016-01-30 [cit. 2016-05-01], version 2.3.
URL https://doc.nette.org/cs/2.3/getting-started

[4] *Components* [online]. Bootstrap, 2016-02-26 [cit. 2016-05-01], version 3.3.6.
URL http://getbootstrap.com/components/

[5] *Debugování a zpracování chyb* [online]. Nette Foundation, 2016-02-26 [cit. 2016-05-01].
URL https://tracy.nette.org/

[6] *Latte* [online]. Nette Foundation, 2016-02-26 [cit. 2016-05-01].
URL https://latte.nette.org/

[7] *MVC aplikace & presentery* [online]. Nette Foundation, 2016-02-26 [cit. 2016-05-01], version 2.3.
URL https://doc.nette.org/cs/2.3/getting-started

[8] *Zabezpečení před zranitelnostmi* [online]. Nette Foundation, 2016-02-26 [cit. 2016-05-01], version 2.3.
URL https://doc.nette.org/cs/2.3/vulnerability-protection

[9] *Získávání závislostí* [online]. Nette Foundation, 2016-02-26 [cit. 2016-05-01].
URL https://doc.nette.org/cs/2.3/di-usage

[10] *QR kód* [online]. acunetix, 2016-04-29 [cit. 2016-05-01].
URL https://cs.wikipedia.org/wiki/QR_k%C3%B3d

[11] Google Developers [online]. [cit. 2016-05-01].
URL https://developers.google.com/chart/infographics/docs/qr_codes

[12] Official ZXing ("Zebra Crossing") project home [online]. [cit. 2016-05-01].
URL https://github.com/zxing/zxing

[13] PHP QR Code [online]. [cit. 2016-05-01].
URL http://phpqrcode.sourceforge.net/

[14] QRcode.com [online]. [cit. 2016-05-01].
URL http://www.qrcode.com/en/

[15] AUGER, R.: *The Cross-Site Request Forgery (CSRF/XSRF) FAQ* [online]. CGISecurity.com, 2010-04-28 [cit. 2016-05-01], version 1.62.
URL http://www.cgisecurity.com/csrf-faq.html

[16] BENADIKOVÁ, A.; Štefan MADA; WEINLICH, S.: *Čárové kódy: autimatická identifikace*. Praha: Grada, 1994 [cit. 2016-05-01], ISBN 80-85623-66-8, 272 s.

[17] BERNARD, B.: *Úvod do architektury MVC* [online]. *Devel.cz Lab s.r.o.*, 2009-05-07 [cit. 2016-05-01], ISSN 1803-5620.
URL https://www.zdrojak.cz/clanky/uvod-do-architektury-mvc/

[18] BERNARD, B.: *Prezentační vzory z rodiny MVC* [online]. *Devel.cz Lab s.r.o.*, 2009-05-11 [cit. 2016-05-01], ISSN 1803-5620.
URL https://www.zdrojak.cz/clanky/prezentacni-vzory-zrodiny-mvc/

[19] CONALLEN, J.: *Building Web Applications with UML*. Boston: Addison-Wesley, druhé vydání, 2003 [cit. 2016-05-01], ISBN 0-201-73038-3, 468 s.

[20] ČERMÁK, M.: *Vícevrstvá architektura: tenký, tlustý a chytrý klient* [online]. CLEVER AND SMART, 2010-05-24 [cit. 2016-05-01], aktualizováno: 2012-11-02.
URL http://www.cleverandsmart.cz/vicevrstva-architektura-tenky-tlusty-a-chytry-klient/

[21] HRUŠKA, T.; KŘIVKA, Z.: *Informační systémy (IIS,PIS): Pojem informačního systému, Data, Procesy, Transakce [online]*. 2012 [cit. 2008-11-28], 164 s.
URL https://www.fit.vutbr.cz/study/courses/WAP/private/opory/OporaIIS2PISPojemDataProcesyTransakce.pdf

[22] KŘENA, B.; KOČÍ, R.: *Úvod do softwarového inženýrství IUS* [online]. Brno: FIT VUT, 2010-12-21 [cit. 2016-05-01].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IUS-IT/texts/IUS_opora.pdf?cid=8697

[23] ŠKRÁŠEK, J.: GitHub, Inc., 2014-10-27.
URL https://github.com/hrach/posobota-2014-server-validation

[24] MICHÁLEK, M.: *Kdy vám Bootstrap pomůže a kdy ne?* [online]. Vzhůru dolů, 2014-02-10 [cit. 2016-05-01].
URL http://www.vzhurudolu.cz/blog/11-bootstrap-pomuze

[25] MOLNÁR, Z.: *Podnikové informační systémy*. Praha : ČVUT, 2009, ISBN 9788001043806, 195 s.

[26] ČÁPKA, D.: *1. díl - Úvod do UML* [online]. ITnetwork.cz, viděno: 2016-04-16; [cit. 2016-05-01].
URL http://www.itnetwork.cz/navrhove-vzory/uml/uml-uvod-historie-vyznam-a-diagramy

[27] ROBYER: *Live Form Validation (for Nette Forms 2.3)* [online]. GitHub, Inc.
URL https://github.com/Robyer/nette-live-form-validation

[28] SCHMULLER, J.: *Myslíme v jazyku UML*. Grada, 2001, ISBN 8024700298, 360 s.

[29] TICHÝ, J.: *Cross-site scripting* [online]. PHP Guru.cz, 2008-02-22 [cit. 2016-05-01]. URL http://www.phpguru.cz/clanky/cross-site-scripting

[30] TREJBAL, T.: Proč si dávat pozor na qr kódy? [online]. 2012-03-11 [cit. 2016-05-01]. URL http://www.qr-kody.cz/qr/pozor-na-qr-kody.html

[31] WINTER, M.: *Scan Me: Everybody's Guide to the Magical World of QR Codes*. Napa: Westsong Publishing, první vydání, 2011 [cit. 2016-05-01], ISBN 978-0-9659000-3-4, 144 s.

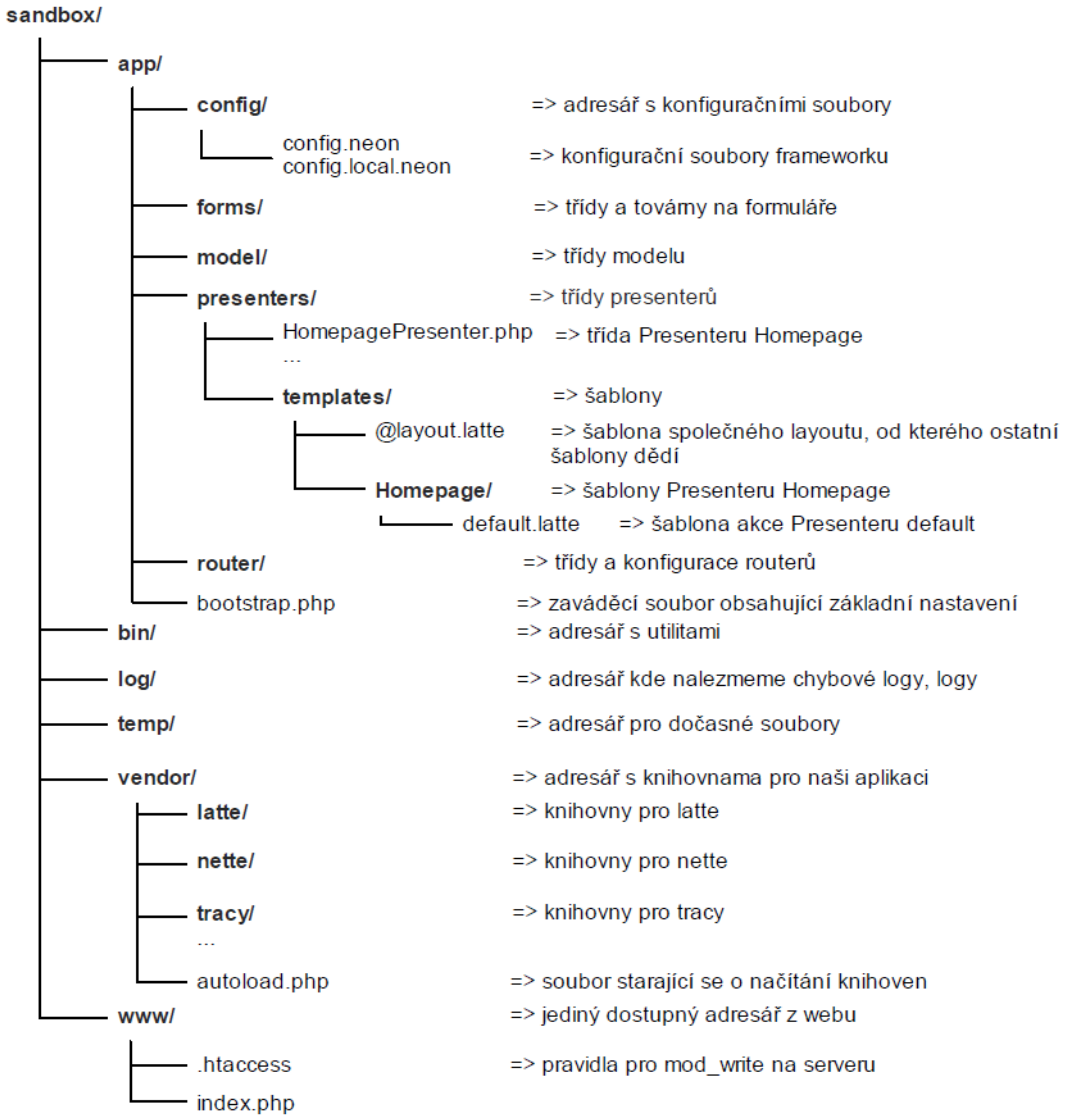
[32] ZENDULKA, J.; Rudolfová, I.: *Databázové systémy IDS* [online]. Brno: FIT VUT, 2006-07-18 [cit. 2016-05-01].
URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IDS-IT/texts/IDS_predn.pdf?cid=9355

Přílohy

Seznam příloh

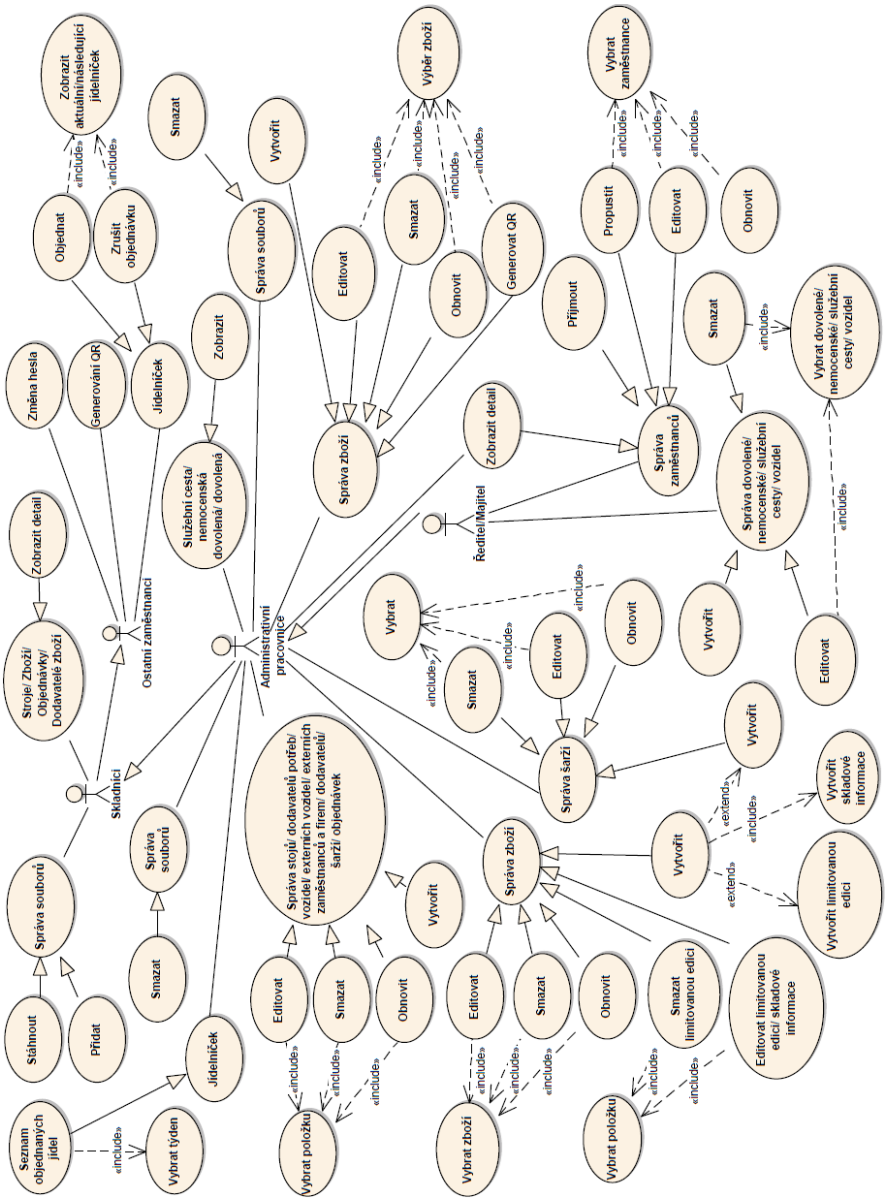
Příloha A

Schéma struktury Sandboxu Nette Frameworku



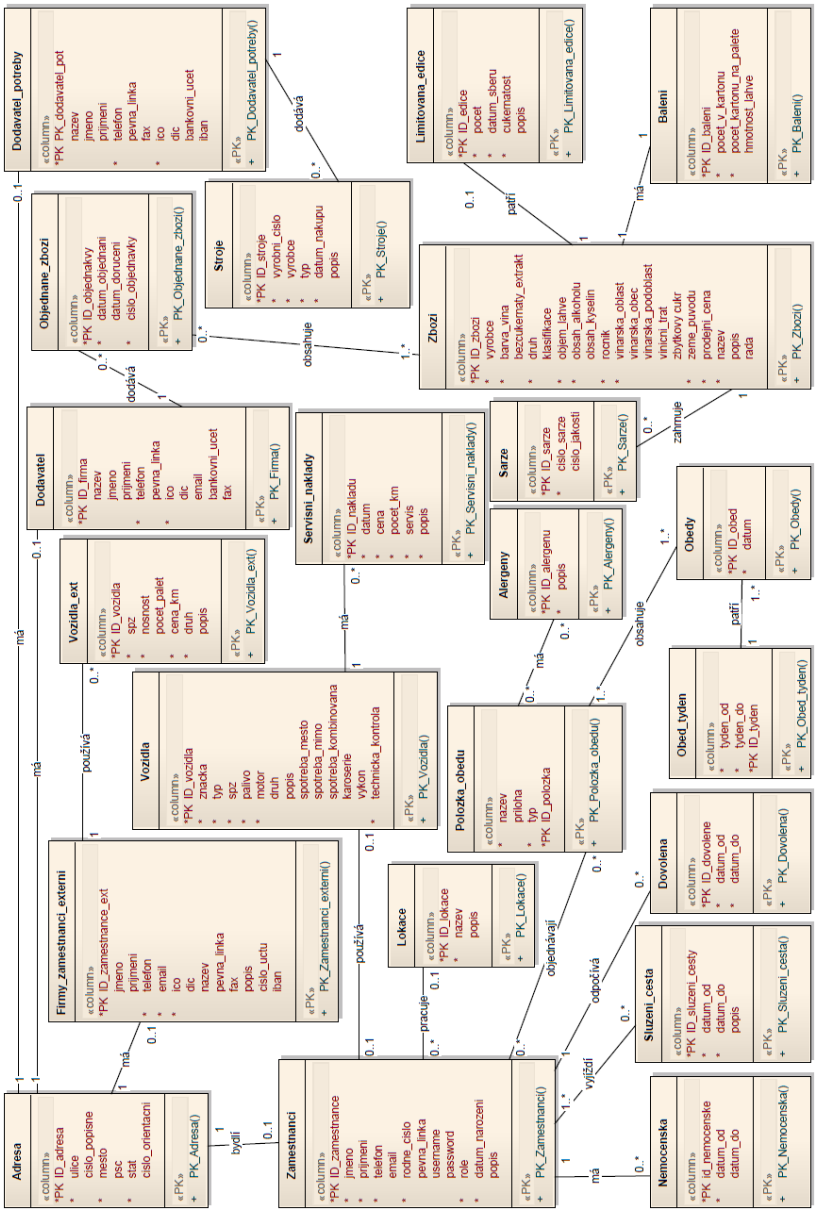
Příloha B

Analýza návrhu IS



Příloha C

Návrh databáze IS



Příloha D

Základní prostředí IS

Příloha E

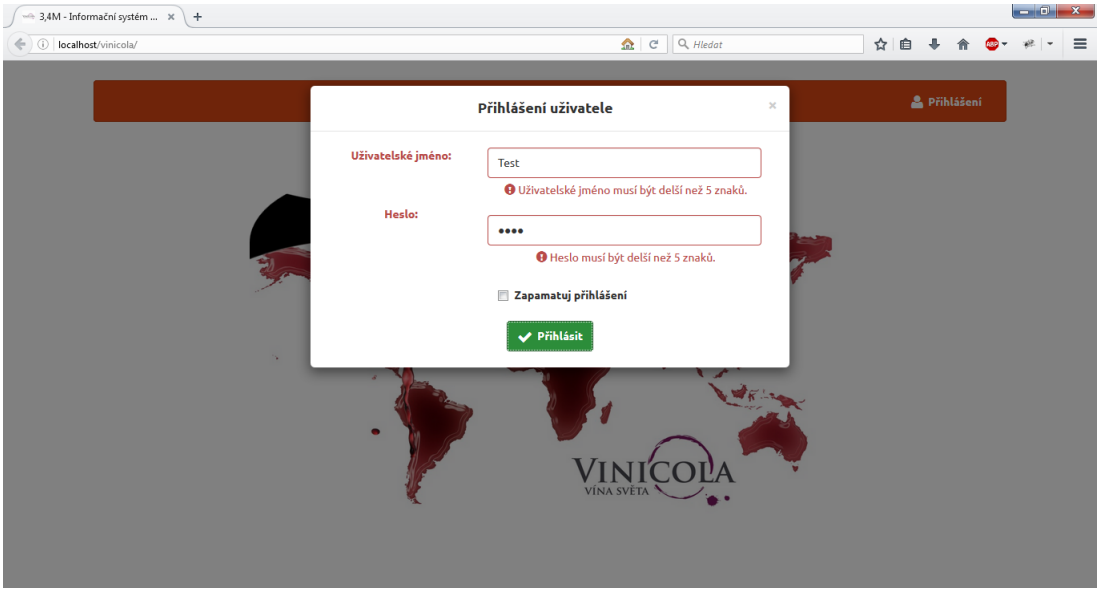
Přihlašování uživatele

Příloha F

Přihlášený uživatel

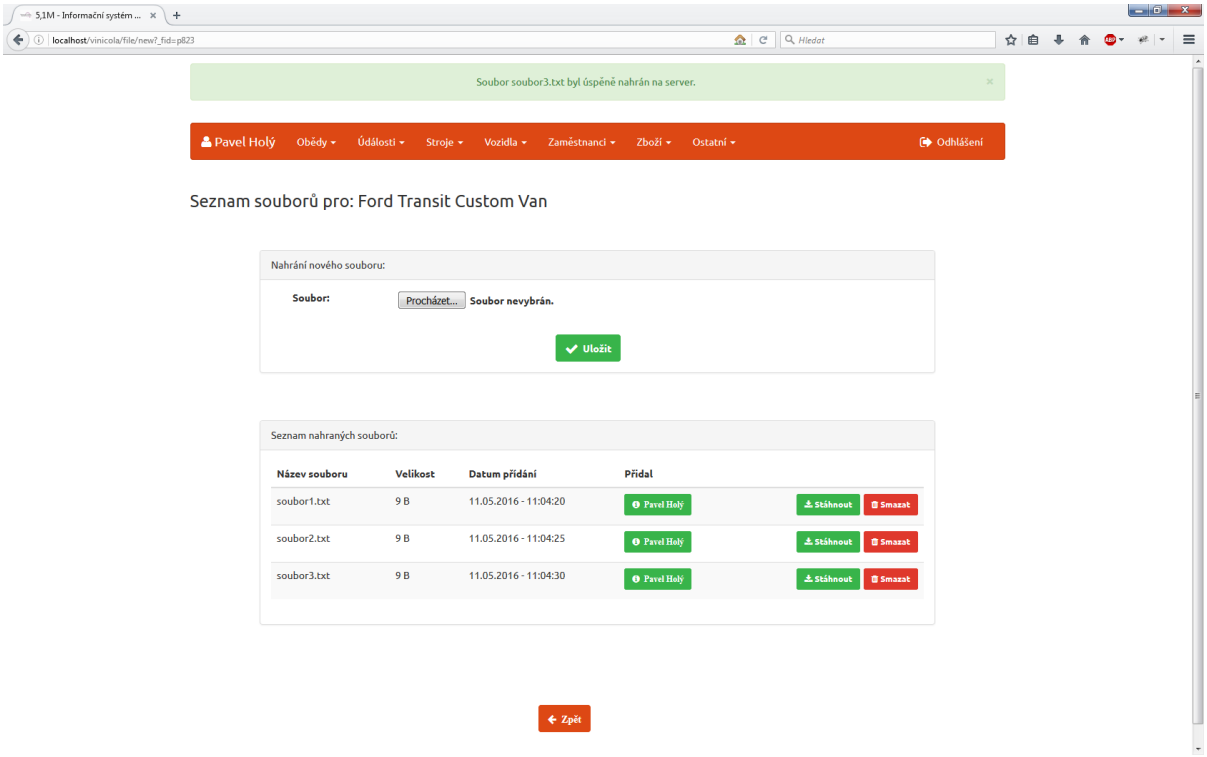
Příloha G

Náhled na Live Validaci formuláře



Příloha H

Přehled a nahrávání souborů



Příloha I

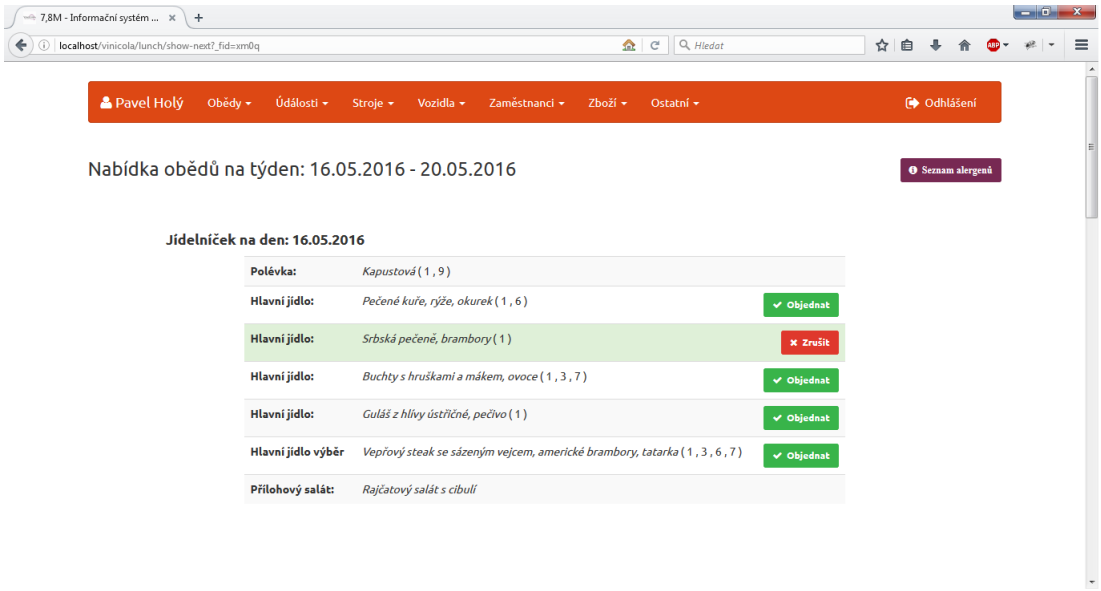
Uložení souboru

Zdrojový kódu I.1: Uložení souboru

```
$retults = $this->database->query("SELECT S.*
                FROM tabsoubor AS S
                JOIN $table AS SS ON S.ID_souboru = SS.souboru_id
                WHERE SS.polozky_id = $id
                ORDER BY S.skutecny_nazev");

$pole_result = array();

foreach($retults as $retult){
  $i = 0;
  foreach ($retult as $tmp){
    $pole_result[$retult->ID_souboru][$i++] = $tmp;
  }

  $result_2 = $this->database->query("SELECT Z.jmeno, Z.prijmeni,
      Z.aktivni
              FROM tabzamestnanec AS Z
              WHERE Z.ID_zamestnanec = $retult->zamestnanec_id");

  if($result_2){
    foreach($result_2 AS $tmp2){
      $pole_result[$retult->ID_souboru][$i++] = $tmp2->jmeno." ".
         $tmp2->prijmeni;
    }
  }
}

return $pole_result;
```

Příloha J

Objednání obědu



Příloha K

Generování QR kódu

Zdrojový kódu K.1: Generování QR kódu

```
<script type="text/javascript">
var qrcode = new QRCode("qr-code", {
  width: 200,
  height: 200,
  correctLevel : QRCode.CorrectLevel.H
});

function vygenerovatQr(text) {
  qrcode.makeCode(text);
}
</script>
```