VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

INFORMAČNÍ SYSTÉM PRO DROBNÉHO ZEMĚDĚLCE

INFORMATION SYSTEM FOR A SMALL FARMER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MAREK KUCHYNKA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. VLADIMÍR BARTÍK, Ph.D.**
SUPERVISOR

BRNO 2019

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2018/2019

Zadání bakalářské práce

Student: **Kuchynka Marek**
Program: Informační technologie
Název: **Informační systém pro drobného zemědělce**
Information System for a Small Farmer
Kategorie: Informační systémy

Zadání:

1. Seznamte se s principy tvorby webových informačních systémů, dostupnými prostředími a frameworky. Dále se seznamte s problematikou OLAP analýzy.
2. Analyzujte požadavky na informační systém, který kromě základních funkcí (evidence polí, plodin, prací a postřiků) bude také umožňovat jednoduchou OLAP analýzu nad daty, která IS bude poskytovat.
3. Navrhněte informační systém dle požadavků.
4. Navržený systém implementujte a ověřte jeho funkčnost na vhodném datovém vzorku.
5. Zhodnoťte dosažené výsledky a diskutujte další možné pokračování tohoto projektu.

Literatura:

- Naramore, E., Gerner, J. et al: PHP 6, MySQL, Apache: Vytváříme webové aplikace. Computer Press, 2009. ISBN: 978-8-0251-2767-4.
- Žára, O.: JavaScript - Programátorské techniky a webové technologie, Computer Press, 2015. ISBN: 978-80-251-4573-9
- Wrembel, R., Koncilia, C.: Data Warehouses and OLAP: Concepts, Architectures and Solutions. IGI GLobal, 2007. ISBN: 978-1-5990-4364-7.

Pro udělení zápočtu za první semestr je požadováno:

- Body 1-3.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce: **Bartík Vladimír, Ing., Ph.D.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2018
Datum odevzdání: 15. května 2019
Datum schválení: 16. října 2018

Abstrakt

Cílem této bakalářské práce bylo navrhnout a vytvořit webový informační systém pro drobného zemědělce, který bude později využit v reálném provozu. Podstatou celého systému je OLAP analýza příjmů a výdajů, které se zjistí na základě vykonaných prací na polích, nakoupeného materiálu a prodaných produktů. Pro implementaci byl zvolen jazyk PHP a český framework Nette společně s MySQL databází.

Abstract

The aim of the thesis was to design and create a web information system for a small farmer, which will be later used on a real farm. The core of the whole system is the OLAP analysis of incomes and expenses, which is based on the works done on fields, purchased materials and sold products. For implementation were chosen PHP, Nette – the Czech framework and MySQL database.

Klíčová slova

informační systém, webová aplikace, OLTP, OLAP, ETL, HTML, CSS, JavaScript, jQuery, PHP, Nette, Latte, MVC, MySQL, zemědělství

Keywords

information system, web application, OLTP, OLAP, ETL, HTML, CSS, JavaScript, jQuery, PHP, Nette, Latte, MVC, MySQL, farming

Citace

KUCHYNKA, Marek. *Informační systém pro drobného zemědělce*. Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Bartík, Ph.D.

Informační systém pro drobného zemědělce

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doktora Vladimíra Bartíka. Další informace mi poskytl Ing. Ladislav Souček jako soukromý zemědělec s dlouholetou praxí. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Marek Kuchynka
6. května 2019

Poděkování

Rád bych na tomto místě poděkoval panu Ing. Vladimíru Bartíkovi, Ph.D. za odborné vedení mé bakalářské práce, za pomoc a rady, které mi v průběhu řešení poskytl. Dále bych chtěl poděkovat panu Ing. Ladislavu Součkovi za odborné konzultace ohledně zemědělství a věcné připomínky k vytvářenému systému.

Obsah

1 Úvod **3**

2 Principy tvorby webových informačních systémů **4**
2.1 Front-end . . . 5
2.1.1 HTML . . . 5
2.1.2 CSS . . . 6
2.1.3 JavaScript . . . 6
2.1.4 Dart . . . 7
2.2 Back-end . . . 7
2.2.1 PHP . . . 7
2.2.2 ASP.NET . . . 8
2.2.3 Java . . . 8
2.2.4 Python . . . 8
2.2.5 C a C++ . . . 8
2.3 Databáze . . . 9
2.3.1 Dostupné SŘBD . . . 9
2.3.2 Jazyk SQL . . . 10
2.4 Klasifikace informačnch systémů . . . 10
2.4.1 Systémy OLTP . . . 10
2.4.2 Systémy OLAP . . . 11

3 Analýza a specifikace požadavků **13**
3.1 Uživatel . . . 13
3.2 Přehledy . . . 13
3.3 Skladové hospodářství . . . 14
3.4 Práce na poli . . . 14
3.5 Evidence použitých postřiků . . . 15
3.6 Diagram případů užití . . . 15
3.7 Existující řešení . . . 16

4 Použité technologie **18**
4.1 Front-end . . . 18
4.2 Back-end . . . 18
4.3 Databáze . . . 20
4.4 Knihovny . . . 20
4.5 Další software . . . 21
4.5.1 Composer . . . 21
4.5.2 Git . . . 21

4.5.3 Server Apache . . . 21

5 Návrh informačního systému **22**
5.1 Entitně-vztahový model . . . 22
5.1.1 Pole . . . 23
5.1.2 Sklad . . . 23
5.1.3 Práce . . . 23
5.2 Schéma relační databáze . . . 25
5.2.1 Transformace vztahů . . . 25
5.2.2 Generalizace a specializace . . . 25
5.2.3 Slabá entitní množina . . . 25
5.2.4 1. normální forma . . . 26
5.2.5 2. normální forma . . . 26
5.2.6 3. normální forma . . . 26
5.2.7 Boyceho-Coddova normální forma . . . 26
5.2.8 Další optimalizace . . . 26
5.3 Analytická část systému – OLAP . . . 28

6 Implementace **29**
6.1 Databáze . . . 29
6.2 Modely . . . 30
6.2.1 Entity . . . 30
6.2.2 Managery . . . 31
6.2.3 ETL a OLAP . . . 32
6.2.4 Pomocné třídy . . . 32
6.3 Presentery . . . 33
6.3.1 Router . . . 33
6.3.2 Životní cyklus presenteru . . . 34
6.3.3 Presentery v aplikaci . . . 35
6.3.4 Formuláře . . . 35
6.4 Pohledy (view) . . . 37
6.4.1 Šablony . . . 37
6.4.2 Makra . . . 38
6.4.3 Filtry . . . 39
6.4.4 Formuláře . . . 40

7 Testování **41**
7.1 Průběžné testování . . . 41
7.2 Konzultace . . . 41
7.3 Testování zemědělcem . . . 42

8 Závěr **43**

Literatura **44**

A Obsah přiloženého média **47**

B Instalace **48**

C Ukázky ze systému **50**

Kapitola 1

Úvod

Cílem této bakalářské práce je vytvořit informační systém pro drobného zemědělce, který by mu usnadnil administrativu, pomohl při rozhodování a zjednodušil vedení farmy zaměřené na rostlinnou výrobu. Systém bude využit v reálném provozu u soukromého zemědělce.

Informační systém bude implementován jako webová aplikace z důvodu přístupnosti z jakéhokoli zařízení a bezbroblémových aktualizací. Dalšími výhodami, popisem tohoto přístupu a způsoby vytváření webových aplikací se věnuje kapitola 2.

Zemědělec uvedl několik požadavků, které od systému očekává. Jeho hlavním cílem je analýza financí – které plodiny jsou nejvýnosnější a která pole jsou nejúrodnější. Zajímá ho také, zda se vyplatí utratit více peněz za hnojiva a více sklidit, nebo se spokojit s nižšími výnosy a ušetřit na hnojivech. Přitom chce jednoduchý systém, jenž bude obsahovat pouze funkce, které využije. Jeho požadavkům a existujícím řešením je věnována kapitola 3.

Po provedené analýze bylo třeba zvolit vhodné technologie pro vývoj aplikace, čímž se zabývá kapitola 4. Vítězem se stal jazyk PHP, který stále patří mezi nejoblíbenější jazyky pro tvorbu webových aplikací a na menší systém tohoto typu je ideální. Jako webový rámec celé aplikace byl zvolen Nette framework, jenž má původ v České republice. Databázovou vrstvu bude v aplikaci zajišťovat databáze MySQL, která byla vyhodnocena jako nejvhodnější a s PHP tvoří velmi dobré spojení.

Po analýze požadavků a výběru technologií nic nebrání vytvoření návrhu informačního systému. V kapitole 5 je popsán postup vytvoření entitně-vztahového modelu celé aplikace a poté jeho transformace na logické schéma relační databáze, které bude později využito právě při vytváření databázové vrstvy. Také zde bude navržena databáze pro analytickou část systému – OLAP.

V praktičtěji zaměřené části této práce nalezne čtenář popis implementace výsledného systému ve vybraných technologiích. V kapitole 6 je podrobně popsán Nette framework a návrhový vzor MVC. Zde lze vidět postupy a řešení, ke kterým bylo v průběhu vytváření aplikace přistoupeno a jaké další návrhové vzory byly použity. Speciální pozornost bude věnována formulářům, které jsou nedílnou součástí každého informačního systému.

Závěrešná kapitola 7 se věnuje způsobu, jakým způsobem probíhalo testování aplikace a její nasazení do ostrého provozu. Následuje shrnutí celé práce, zhodnocení vytvořeného systému a návrhy na další možné pokračování a rozšíření.

Kapitola 2

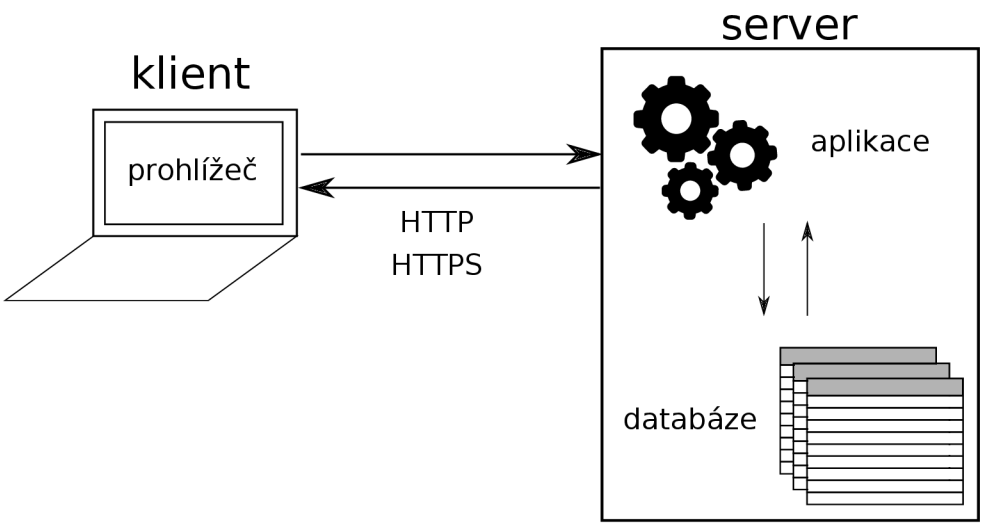
Principy tvorby webových informačních systémů

Nejdříve je třeba vysvětlit pojem „webový informační systém“ – jedná se o spojení webové aplikace a informačního systému. Pro lepší pochopení budou popsány obě části zvlášť.

Webová aplikace

Původním cílem iternetových prohlížečů bylo pouze prohlížení webových stránek, avšak v dnešní době se s rozvojem internetu a jeho dostupností stále více používají také pro přístup k webovým aplikacím, které dokáží nahradit celé programy. Hlavní výhodou těchto aplikací je, že je uživatel nemusí mít nainstalované lokálně na svém počítači – aplikace je totiž uložena na vzdáleném webovém serveru.

Na obrázku 2.1 je znázorněna obecná struktura webové aplikace. Na levé straně je klientský počítač s otevřeným prohlížečem, který komunikuje pomocí protokolu HTTP *(Hypertext Transfer Protocol)* nebo zabezpečeného HTTPS *(Hypertext Transfer Protocol Secure)* s webovým serverem, na němž je uložena aplikační logika a databáze.

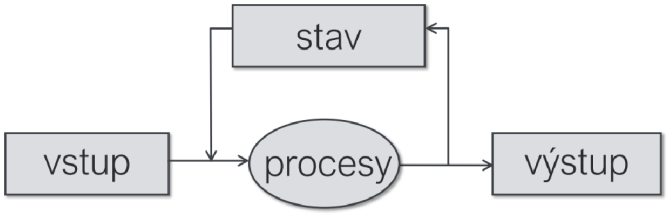


Obrázek 2.1: Obecná struktura webové aplikace (zdroj: autor)

Informační systém

> Informační systém lze chápat jako systém vzájemně propojených prostředků a procesů, které slouží k ukládání, zpracovávání a poskytování informací. Pojem procesy chápeme jako funkce, které zpracovávají vstupní informace, jež se v systému přemění na informace výstupní. Jinými slovy, procesy jsou funkce, které zajišťují sběr, přenos, uložení, zpracování a distribuci informací.[32]

Na obrázku 2.2 lze vidět obecnou strukturu informačního systému korespondující s definicí výše. V další části této kapitoly budou popsány konkrétní části webového informačního systému a dostupné technologie pro jejich řešení.



Obrázek 2.2: Struktura informačního systému (zdroj: [4])

2.1 Front-end

Pojem front-end lze přeložit jako „přední konec“, což značí část aplikace, kterou vidí uživatel (klient), proto je mnohdy označována jako klientská. Tato část by měla být propracována z hlediska vzhledu, přístupnosti a použitelnosti.[6] Na obrázku 2.2 odpovídají této části položky „vstup“ a „výstup“.

2.1.1 HTML

Hlavním stavebním prvkem všech webových aplikací z hlediska front-end části je bezpochyby značkovací jazyk HTML *(Hypertext Markup Language)*. Jazyk vznikl v roce 1990 odvozením od jazyka SGML. Za jeho zakladatele je považován Tim Berners-Lee.[25, str.17] V současné době je udržován organizací W3C a jeho nejaktuálnější verze 5.2 byla publikována 14. prosince 2017.[39]

Text dokumentu se uzavírá do jednotlivých značek *(tagů)*, čímž se určí jeho význam (sémantika).[24, str.19] Značky se dělí na párové a nepárové. Mezi nejvýznamnější značky patří `<h1>` (nadpis), `<p>` (odstavec), `<a>` (odkaz) nebo `<img>` (obrázek). Uvnitř značek se také mohou vyskytovat různé atributy, které se liší v závislosti na druhu značky (cíl odkazu, zdroj obrázku, třída, identifikátor atd.).

```
<!-- odkaz vedoucí na web example.com -->
<a href="https://www.example.com">klikni</a>
```

Speciální kategorií značek jsou formuláře a jednotlivé formulářové prvky (vstupní políčka, tlačítka, výběrové prvky atd.). Pomocí formulářů mohou uživatelé zadávat na stránku potřebné údaje, které jsou odeslány na server k dalšímu zpracování. Tím tedy vznikají interaktivní webové stránky umožňující uživateli obousměrnou komunikaci se serverem.[25, str.116]

2.1.2 CSS

Pro definici vzhledu webové stránky slouží pravidla CSS (*Cascading Style Sheets* – kaskádové styly). Autorem jazyka je Håkon Wium Lie, který jej vytvořil v roce 1994. V současné době se jazyk nachází ve verzi CSS3, jež byla vyvíjena od roku 1999.[3]

CSS je jazyk pro popis způsobu zobrazení stránek napsaných v jazycích HTML, XHTML nebo XML [24, str.19], a to na základě jednotlivých pravidel. Každé pravidlo se skládá ze selektoru a z množiny deklarací.[25, str.146] Selektor určuje, na který element v HTML/XML dokumentu se pravidlo použije a jednotlivé deklarace poté udávají vlastnosti dané části dokumentu (například barvu a velikost písma, barvu pozadí, vertikální a horizontální zarovnání, pozici prvků, ...).

```
a {color: red}      // všechny odkazy budou červené
```

Stylopis v CSS se může zapsat přímo k jednotlivým elementům, do hlavičky HTML stránky, nebo do externího souboru, jenž je do HTML stránky vložen.[35, str.27]

Nad samotným jazykem CSS jsou postaveny různé preprocesory, které si kladou za cíl usnadnit programátorovi práci při vytváření stylopisu. V jejich syntaxi lze používat proměnné, cykly, podmínky, matematické operace, zanořování selektorů apod. Po uložení takto napsaného souboru je třeba jej přeložit do čistého CSS souboru, protože nic jiného prohlížeče přečíst neumí. Mezi nejoblíbenější preprocesory patří Sass, Less a Stylus, ale existuje jich samozřejmě mnohem více.[7]

Dále lze využít různé rámce *(frameworky)* pro CSS. Mezi nejčastěji používané náleží Bootstrap, Foundation nebo Pure.css.[27]

2.1.3 JavaScript

Součástí klientské části aplikace je také skriptovací jazyk JavaScript. Jeho autorem je Brendan Eich, který jej vyvíjel pro společnost Netscape od roku 1995. Jazyk je objektově orientovaný, dynamicky typovaný a staví na prototypové dědičnosti.[42, str.12] Jeho zápis je velmi podobný jazykům rodiny C/C++ nebo Java.[24, str.20]

Kód zapsaný v JavaScriptu se umisťuje přímo do HTML stránky nebo se k ní externě připojí ze speciálního souboru.[42, str.28] Kód je přečten prohlížečem a následně je jím také vykonán. Program je tedy spouštěn přímo na počítači klienta, proto není možné například přistupovat k souborům nebo k databázi na serveru.

JavaScript se využívá hlavně pro tvorbu interaktivních webových stránek. Dokáže zareagovat na akce uživatele (např. kliknutí myší, zmáčknutí klávesy atd.) a přizpůsobit na základě těchto akcí obsah stránky nebo ovládat různé interaktivní prvky – tlačítka, textová pole apod.[24, str.20]

Práci s JavaScriptem může usnadnit například knihovna jQuery[1], která zjednodušuje zápis a zajišťuje funkční kód napříč různými prohlížeči. Dalším velmi významným rozšířením JavaScriptu je AJAX *(Asynchronous JavaScript And XML)*, jenž umožňuje nahrávání obsahu webových stránek bez jejich opětovného načítání. Nad samotným JavaScriptem existuje mnoho frameworků, k nejvýznamějším z nich se řadí React.js, Angular.js 2, Vue.js a další.[31] JavaScript lze spouštět také na serveru – projekt Node.js[2].

[1] https://jquery.com/
[2] https://nodejs.org/en/

2.1.4 Dart

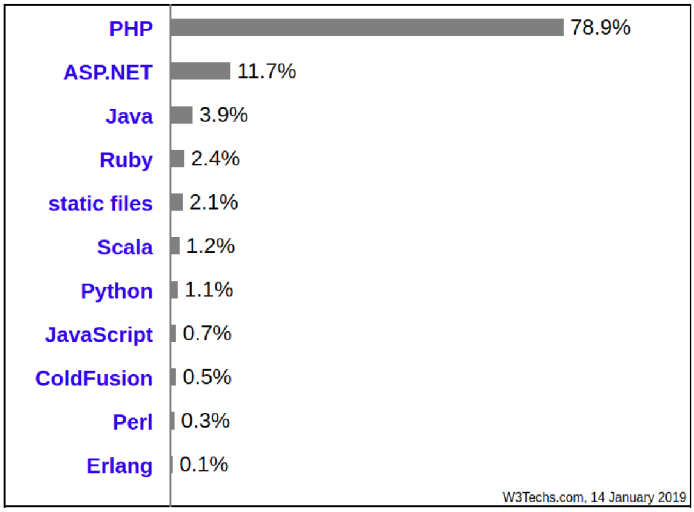
Jazyk Dart vyvíjený společností Google je relativně nový a zatím není příliš rozšířený. Jedná se o univerzální interpretovaný programovací jazyk, ve kterém lze psát aplikace pro mobilní telefony, prohlížeče nebo pro servery. Syntaxí připomíná jazyk C++, lze v něm programovat funkcionálně i objektově. Jelikož jeho interpret není implementovaný ve všech prohlížečích, byl vytvořen překladač do jazyka JavaScript, přesto nabízí vývojářům funkce, kterými JavaScript nedisponuje.[8]

2.2 Back-end

Definice pojmu back-end jsou dvě. Například [28] jej definuje jako část aplikace, která je přístupná pouze administrátorům (administrace e-shopu, publikování článků apod.). Jinde se dočteme, že se jedná o samotné jádro aplikace, jenž řídí, co bude ona aplikace dělat.[18, 6] V této práci se budeme držet druhé definice.

Back-end je tedy „zadní část“ aplikace, jenž v případě webových aplikací běží převážně na serveru, ale podle tloušťky klienta se může vyskytovat také na klientské straně. Nejčastějším případem je **tenký klient**, který pouze přijímá webovou stránku s vygenerovaným obsahem od serveru a zobrazuje jej uživateli. Naopak **tlustý klient** přijímá od serveru pouze data, se kterými poté provádí operace přímo v prohlížeči. Na obrázku 2.2 odpovídá back-endu položka „procesy“.

Z grafu na obrázku 2.3 lze zjistit nejoblíbenější technologie, které se používají na webových serverech. Na jednom serveru může být zastoupeno i více programovacích jazyků.[26]



Obrázek 2.3: Používané technologie na straně serveru (zdroj: [26])

2.2.1 PHP

Dominantou v serverových technologiích je jazyk PHP *(původně Personal Home Page, nyní PHP: Hypertext Prerocessor)*. Jazyk vytvořil v roce 1995 Rasmus Lerdorf. Jeho vývoj a syntaxe byl ovlivněn jazyky C, Perl a Java. V současné době se nachází ve verzi 7.3.[29, str.15-19]

Jedná se o multiplatformní skriptovací jazyk určený především pro programování dynamických webových stránek, ale lze jej využít i při tvorbě konzolových a desktopových aplikací. Oblíbeným se stal zejména pro svoji jednoduchost, univerzálnost a přenositelnost. Programy v jazyce PHP lze psát funkcionálně i objektově, je možné se připojit k různým typům databází a je také umožněna práce se soubory.[24, str.20-21]

Pro vývoj aplikací existuje nepřeberné množství PHP frameworků. Mezi nimi lze nalézt mimo jiné i českého zástupce Nette, který se v České republice používá nejvíce. Obecně nejpoužívanějším je framework Laravel, dále Symfony, CodeIgniter nebo Yii[20].

2.2.2 ASP.NET

ASP.NET navazuje na technologii ASP a je součástí .NET Frameworku, jenž je vyvíjen od roku 2002 společností Microsoft. Jedná se o soubor několika menších nástrojů, mezi které patří například ASP.NET Web Forms, ASP.NET MVC, ASP.NET Web Pages nebo ASP.NET Single Page Application.

V prostředí .NET lze programovat v několika různých jazycích podporujících CLR *(Common Language Runtime)*. Mezi tyto jazyky patří C#, Visual Basic .NET, Perl či Python. Zdrojový kód se překládá do spustitelných souborů.[19, str.25-41]

2.2.3 Java

Java patří k nejoblíbenějším programovacím jazykům[5], proto je pochopitelné, že se objevuje i na webu.

Jazyk Java začal vznikat v roce 1991 jako součást „Zeleného projektu“ ve společnosti Sun Microsystems a vydán byl oficiálně v roce 1995. Vytvořil jej James Gosling, vychází z jazyků C, C++, C# a Smalltalk a je objektově orientovaný. Zdrojový kód programu se překládá do tzv. bajtkódu, který je přenositelný na různé platformy, což bylo hlavním cílem jejího vývoje.[15, str.16-19]

2.2.4 Python

Python je vysokoúrovňový programovací jazyk, který vytvořil v roce 1991 Guido van Rossum. Poslední majoritní verze Python 3 byla vydána v roce 2008 a je nekompatibilní vůči své předešlé verzi Python 2. Python je multiparadigmatický a řadí se mezi skriptovací jazyky, ovšem jeho možnosti jsou mnohem širší, protože je v něm možné vytvářet plnohodnotné desktopové a mobilní aplikace. Díky své jednoduché syntaxi bývá označován za nejlepší jazyk pro začátečníky.[40]

Nejpoužívanější implementací je CPython, který je implementovaný v jazyce C. I přes to, že jsou využity knihovny jazyka C, řadí se k pomalejším jazykům, jež se nehodí pro psaní kritických aplikací. Pro programování webových aplikací se často používá v kombinaci s frameworky Django, Pyramid nebo TurboGears.[34]

2.2.5 C a C++

Kořeny jazyka C sahají do 70. let 20. století. Za jeho tvůrce jsou považováni Dennis Ritchie a Ken Thompson. Jedná se o nízkoúrovňový kompilovaný jazyk, který má velmi blízko k assembleru, a proto aplikace v něm napsané jsou velmi rychlé a efektivní.[12, str.20-21] V roce 1985 rozšířením jazyka C o třídy vznikl nový, objektově orientovaný jazyk C++.

Jeho autorem je Bjarne Stroustrup, který v Bellových laboratořích AT&T spolupracoval s Dennisem Ritchiem.

Jazyky C i C++ nachází své uplatnění především v desktopových a mobilních aplikacích, ale lze je použít také jako back-end na webovém serveru.

2.3 Databáze

„Databázi (anglicky *database* neboli datová základna) chápeme jako úložiště údajů, které jsou uloženy a zpracovávány nezávisle na aplikačních programech. Databáze zapouzdřují jednak vlastní údaje, ale i relační vztahy mezi jednotlivými prvky a objekty v databázi, schémata popisující struktury údajů a integritní omezení.“[17, str.25] Přístup k těmto datům a operace s databází obstarává program (software), jenž bývá nazýván „systém řízení báze dat“ (SŘBD). S tímto programem většinou spolupracuje back-end webové nebo jiné aplikace umožňující čtení, upravování, vkládání i odstraňování dat. Na obrázku 2.2 odpovídá databázi položka „stav“.

Matematiky byl zaveden pojem databázový model, což je prostředek pro popis databáze. Zpočátku (60. léta 20. století) se prosazoval hierarchický a síťový model. V 70. letech byl vytvořen relační model dat, který se využívá dodnes ve většině aplikací.[43, str.6] Na logické úrovni jsou data strukturována do normalizovaných tabulek (relací). Pohyb v tabulkách je velice jednoduchý – navigace po řádcích a sloupcích tabulky. S rozvojem objektově orientovaných jazyků se i do databázových systému dostává objektově orientovaný model, jenž se však v praxi příliš nevyužívá.[43, str.201-203]

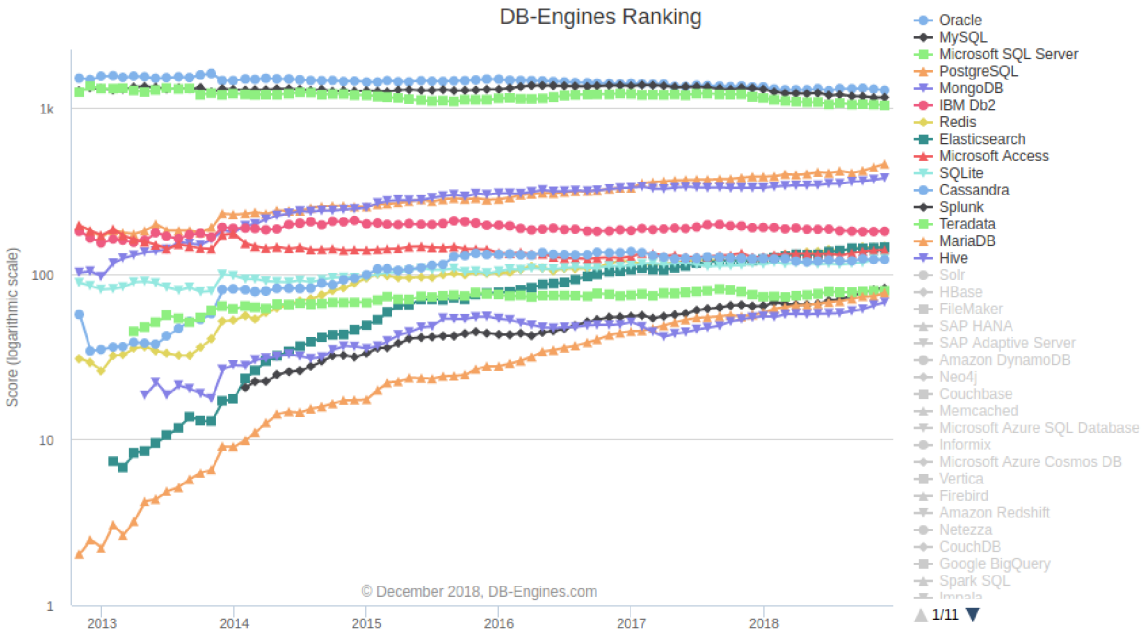
2.3.1 Dostupné SŘBD

Systémů pro řízení báze dat existuje mnoho. Graf na obrázku 2.4 ukazuje popularitu jednotlivých systémů na základě několika kritérií (počet zmínek na webových stránkách, četnost výskytů na Google Trends, četnost výskytů v technických diskuzích, počet pracovních nabídek se zmínkou o daném systému a počet zmínek na sociálních sítích), z čehož lze usoudit, že dominantní jsou na trhu systémy Oracle Database, MySQL a Microsoft SQL Server.

Oracle Database je moderní a multiplatformní databázový systém s pokročilými možnostmi zpracování dat s vysokým výkonem a snadnou škálovatelností. Jedná se o komerční produkt od Oracle Corporation vyvíjený od roku 1980. Implementuje kromě standardního jazyku SQL také proprietární rozšíření, imperativní programovací jazyk PL/SQL nebo objektové databáze.[33]

MySQL je nejpopulárnější databázový systém u vývojářů webových aplikací, k čemuž přispívá i skutečnost, že je k dispozici zdrama pod licencí GPL, je multiplatformní a podporuje jej většina webových hostingů. Byl vytvořen v roce 1995 společností MySQL AB, ale nyní patří pod společnost Oracle Corporation. Zakládá si na rychlosti i za cenu značných zjednodušení a omezení – například velmi dlouho nebyly podporovány databázové triggery a uložené procedury.[33]

Microsoft SQL Server se používá spíše pro podnikové sítě, než pro webové aplikace. Jedná se o komerční produkt vyvíjený firmou Microsoft od roku 1989. Kromě databázového systému obsahuje také analytický systém a řešení datových skladů.[33]



Obrázek 2.4: Popularita jednotlivých SŘBD (zdroj: [33])

2.3.2 Jazyk SQL

Pro manipulaci s daty v relačních databázích se využívá strukturovaný dotazovací jazyk SQL *(Structured Query Language)*, který byl poprvé standardizován v roce 1986.[30] Vznikl z jazyka SEQUEL *(Structured English Query Language)*, jehož cílem bylo vytvořit jazyk, jehož syntaxe by byla co nejblíže přirozenému jazyku. Díky tomu je jazyk SQL na pochopení velmi jednoduchý – jeho zápis se velmi dobře čte každému, kdo alespoň trochu ovládá anglický jazyk.

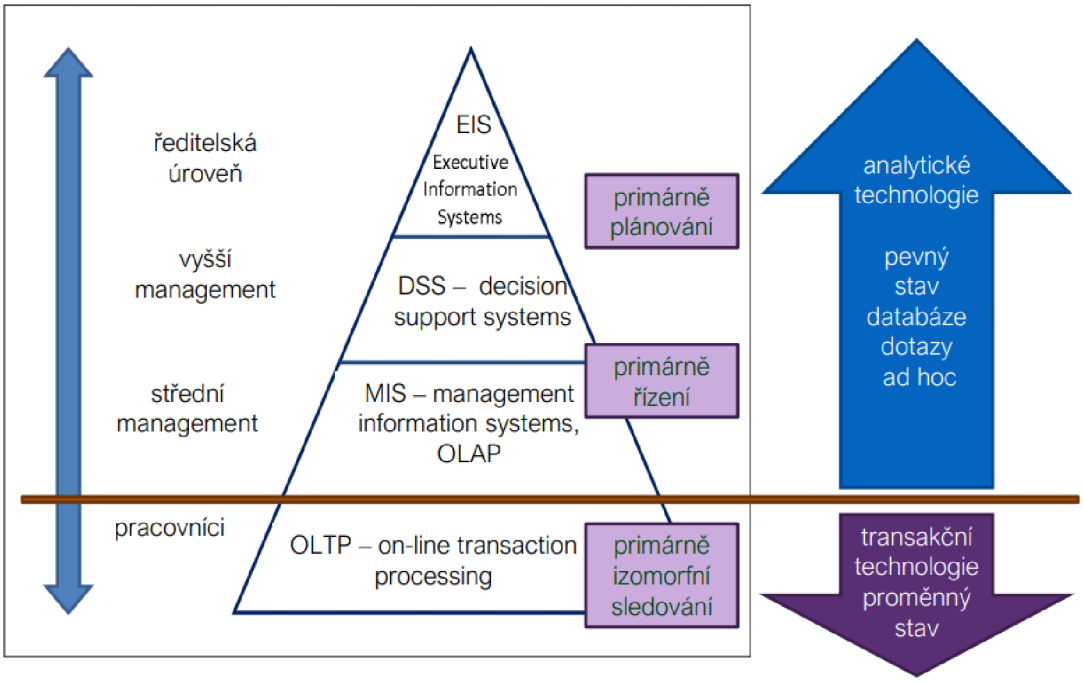
2.4 Klasifikace informačnch systémů

Na obrázku 2.5 lze vidět rozdělení informačních systémů dle úrovně rozhodování. Na nejnižší vrstvě se nachází transakční zpracování dat, tzv. OLTP *(Online Transaction Processing)* a nad ním se v několika formách objevují systémy pro analýzu dat, tzv. OLAP *(Online Analytical Processing)*.

2.4.1 Systémy OLTP

Primárním cílem systémů OLTP je automatizace každodenních činností. Ve firemní praxi jsou velmi často používané a oblíbené. Za cíl si také kladou zpracování velkého množství transakcí online – například bankovní převody, obchodování apod. Ke zdroji údajů tedy ve stejném čase přistupuje velké množství uživatelů najednou, kteří záznamy z databáze čtou, zapisují, případně odstraňují.[17, str.20]

Databáze OLTP jsou z důvodu jednoduchého dotazování a vyloučení redundance zpravidla normalizovány (vyhovují pravidlům tzv. normálních forem). Velmi často bývají systémy OLTP decentralizované, někdy dokonce v různých heterogenních systémech.[17, str.22]



Obrázek 2.5: Pyramidové schéma informačních systémů (zdroj: [4])

Z důvodu diskových kapacit provozních počítačů neukládají transakční systémy historické údaje.[17, str.24]

2.4.2 Systémy OLAP

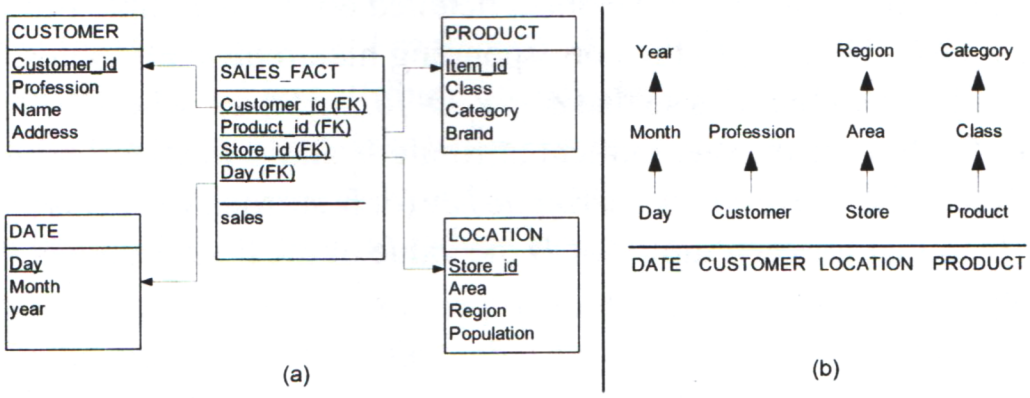
Typické využití systému OLAP je pro analýzu velkého množství dat. Výsledkem analýzy jsou souhrny a reporty, které jsou využívány manažery jako podklady pro řízení a rozhodování.[17, str.21]

Analýza velkého množství normalizovaných dat v relačních databázích by byla velmi neefektivní a značně pomalá, proto byla zavedena organizace údajů do multidimenzionálních struktur. V nich se nachází převážně nenormalizované tabulky, které lze rozdělit na tabulky faktů a tabulky dimenzí.[17, str.31]

Tabulky dimenzí se skládají z jedné či více klasifikačních hierarchií, každá je složená z několika hierarchických atributů. S tabulkou faktů je typicky svázána pomocí nejdetailnějšího z nich. Například „den“, „měsíc“ a „rok“ mohou klasifikovat dimenzi „datum“ a v tabulce faktů na ni bude odkazováno přes „den“.[41, str.138]

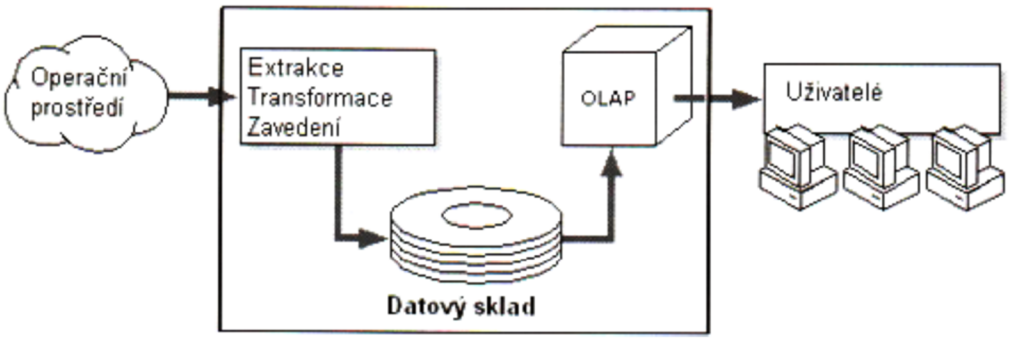
Tabulky faktů obsahují jeden nebo více měřitelných údajů, neboli faktů. Vždy jsou svázány s nějakými tabulkami dimenzí. Centrální uspořádání, kdy se uprostřed nachází tabulka faktů a kolem ní nenormalizované tabulky dimenzí, se nazývá „hvězda“.[41, str.138] Alternativním uspořádáním je schéma sněhové vločky, ve kterém jsou některé dimenze složené z mnoha relačně svázaných normalizovaných tabulek.[17, str.114-115]

Na obrázku 2.6 je znázorněn příklad datového skladu s uspořádáním tabulek typu „hvězda“.



Obrázek 2.6: (a) příklad datového skladu; (b) hierarchie dimenzí z příkladu (zdroj: [41])

Pro přenos dat mezi transakčními systémy (OLTP) a analytickými systémy (OLAP) existují nástroje a postupy zvané **ETL** *(Extraction, Transformation, Loading)*, případně **ETT** *(Extraction, Transformation, Transport)*, na obrázku 2.7 označeno česky jako „Extrakce, Transformace, Zavedení“. Tento proces je ve většině případů časově náročný, zahrnuje výběr dat, jejich ověření, čištění, integrování, časové označení a přemístění do datového skladu. Data v tomto skladu je také nutné pravidelně aktualizovat.[17, str.59-61]



Obrázek 2.7: Datový sklad (zdroj: [17])

Kapitola 3

Analýza a specifikace požadavků

Soukromý zemědělec podnikající v rostlinné sféře se v dnešní době zabývá více administrativou než prací na poli. Počítá zisky a ztráty, výnosy, zjišťuje, zda se mu vyplatí hnojit nebo ne, a hlavně podává mnoho různých hlášení úřadům – o použitých chemikáliích a hnojivech na polích, o pěstování plodin obsahující návykové látky apod.

Právě pro účely drobných zemědělců vzniká informační systém, jehož cílem je tyto činnosti zjednodušit. V následujících podkapitolách bude celé téma důkladněji popsáno a často bude pro přehlednost odkazováno na výsledný diagram případů užití, který je na obrázku 3.1.

3.1 Uživatel

Uživatelem informačního systému bude soukromý zemědělec. Předpokládejme, že i pokud by měl nějaké zaměstnance, bude zadávat data do informačního systému sám, takže není třeba více uživatelů a ani přihlašování do systému. V diagramu případů užití se tedy bude vyskytovat pouze jeden aktér „uživatel“.

Typicky bude systém využíván pro zadávání nových dat nebo pro analýzu dat stávajících. Zemědělec jej bude využívat z domova, přistupovat k němu bude primárně přes stolní počítač nebo notebook. Uživatelské rozhraní by mělo být jednoduché a ovládání lehce pochopitelné i pro technicky méně zkušeného uživatele.

3.2 Přehledy

Hlavní částí celého informačního systému by měl být celkový přehled o podniku. Tento úkol bude obstarávat část operačního systému OLAP. Zemědělec by chtěl vidět primárně příjmy a výdaje, jednoduše a přehledně zobrazitelné dle různých kritérií – po letech, polích, plodinách. Z těchto údajů by měl být zemědělec schopen bez problému identifikovat např. kterou plodinu je výhodné pěstovat, které pole je nejúrodnější nebo jaký osevní postup je nejlepší.

Nejvíce vypovídající hodnotu má dle zemědělce výnos z jednotky plochy, kterou je běžně v zemědělství hektar. Pokud by analýza probíhala pouze na základě polí jako celku, nebylo by k dispozici dostatečné srovnání, jelikož každé pole má jinou rozlohu.

V diagramu případů užití reprezentuje tuto analýzu položka „Analyzovat data“.

3.3 Skladové hospodářství

Aby bylo možné zaznamenávat nákupy, prodeje a využití plodin, chemikálií, paliv a jiných nezbytných prostředků, je třeba zavést alespoň zjednodušenou správu skladu. Zemědělec bude evidovat produkty, které nakoupil nebo prodal, a bude mít přehled o jejich aktuálním stavu, čímž vzniknou data o příjmech a výdajích. Nákupy a prodeje budou vázány vždy k nějaké firmě (dodavatel/odběratel).

U plodin by se ve skladu měl uchovávat navíc i název a odrůdy, díky tomu bude k dispozici dokonalejší přehled například o výnosech, ve kterých se mohou jednotlivé odrůdy značně lišit.

Tyto operace jsou v diagramu užití reprezentovány případy „Nakoupit zboží“, „Naskladnit plodinu“, „Prodat plodinu“ a pomocnými případy „Vyhledat na skladě“ a „Vyhledat firmu“.

3.4 Práce na poli

Největším zdrojem dat bude samozřejmě pole a práce na něm vykonávané. Každý den po práci zemědělec zadá do informačního systému nezbytné údaje o právě vykonané práci včetně všech výdajů (např. množství spotřebovaného paliva, využité postřiky, osivo).

Práce bude vždy vykonána na určitém poli a kvůli určité plodině. Z toho plyne také potřeba uchovávat evidenci polí a plodin, které na nich v průběhu let rostou. Každé jednotlivé pole bude reprezentováno jednoznačným názvem (např. „U rozhledny“, „Za rybníkem“), popřípadě číslem. Navíc bude u polí uloženo číslo DPB ze systému LPIS (bude popsáno dále). Každý rok na poli poroste jedna plodina. Práce nemusí být vykonána na celé ploše daného pole, proto je třeba rozlišovat celkovou plochu pole a obdělanou plochu (např. sklizeň 100 ha pole bude rozdělena do 5-ti dní).

Sezóna na poli samozřejmě není totožná s kalendářním rokem. Dle [38] v účetnictví záleží na podniku (zemědělci), kdy končí náklady pro sklizeň (zda první podmítkou po sklizni či hlubokou orbou) a které kultivační práce pro podzimní osetí náleží ke sklizni příštího roku. Proto by sám uživatel měl systému určit, kdy chce daný rok na poli ukončit.

Příkladem práce může být „ošetření pšenice na poli *Kopec* proti rzi pšeničné“. Výdajem bude určitě nafta do traktoru/postřikovače, postřik proti rzi a voda na přípravu postřiku. Pokud by zemědělec chtěl, může jít ve výdajích ještě do větších detailů a přidat také jednorázový ochraný oděv, rukavice atp.

Pokud bude prací **sklizeň**, kromě výdajů proběhne i naskladnění plodiny. Plodinu lze odvézt buď do vlastního sila, nebo rovnou do výkupu. Ve výkupu může být plodina daný den prodána, nebo zde bude uskladněna a prodej se uskuteční později, například za lepších platebních podmínek.

Evidence polí v diagramu případů užití je reprezentována položkami „Přidat pole“ a „Vybrat pole“, evidence prací na poli reprezentuje položka „Přidat práci na poli“ a vytvoření výdaje položka „Vytvořit výdaj“, která samozřejmě úzce souvisí se skladem a skladovým hospodářstvím. Pokud se bude jednat o sklizeň, využije se také rozšiřující případ užití „Naskladnit plodinu“.

3.5 Evidence použitých postřiků

V současné době se bez chemie na poli téměř žádný zemědělec neobejde. Ať už se jedná o chemii, která úrodu chrání před cizími organismy, nebo o hnojiva, která zvyšují nebo zkvalitňují úrodu, veškeré použité chemikálie se musí nahlásit příslušným úřadům. Aby nemusel zemědelec několikrát opisovat veškeré údaje o použitých prostředcích (kdy, na jakém poli, z jakého důvodu, v jakém množství atp.), bude v informačním systému, který je od zemědělce už jednou zaznamenal, zakomponován generátor všech potřebných hlášení. Systém bude generovat soubor ve formátu PDF, jenž stačí jen vytisknout a zanést na úřad, případně lze využít elektronickou poštu či datovou schránku.

U prací, u kterých byla použita nějaká chemie, se musí evidovat dávka na jednotku (hektar), účel použití, případně poznámku, zda byl postřik účinný. V hlášení nelze identifikovat pole jménem, které přidělil zemědělec. Je třeba specifikovat jednotlivá pole dle identifikátoru DPB ze systému LPIS[1].

V diagramu tomuto případu odpovídá položka „Vygenerovat hlášení postřiků“. Případ užití „Zobrazit použité postřiky“ slouží zemědělci jako archiv použitých postřiků například pro budoucí rozhodování, který postřik použít (bude zde zaznamenáno, na kolik procent byl postřik účinný).

3.6 Diagram případů užití

Při psaní této kapitoly a vytváření diagramu bylo čerpáno z [14, str.35-50].

Pro modelování typických interakcí uživatele se systémem se využívá jazyk UML a diagram případů užití *(use case)*. Jednotlivé případy užití zachycují funkčnost, která bude informačním systémem pokryta, čímž vymezují rozsah prací. Každý z nich popisuje jeden ze způsobů použití systému, tedy jednu jeho požadovanou funkčnost.

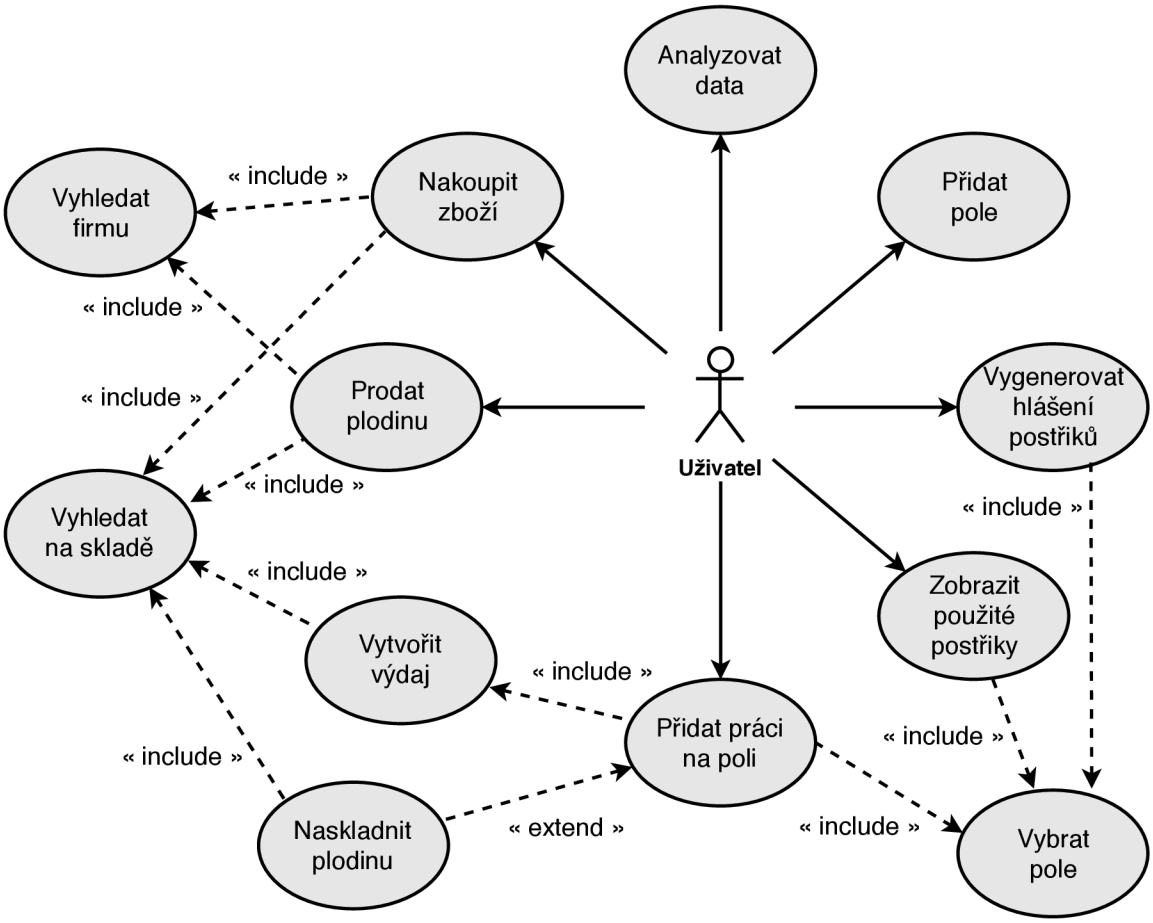
Základem diagramu jsou **aktéři**. Pod tímto pojmem rozumíme roli, ve které vystupuje uživatel v rámci komunikace se systémem. Je běžné, že jeden fyzický uživatel vystupuje ve více rolích v rámci systému. Aktéři spouštějí případy užití, jeden aktér může provádět řadu případů a obráceně, jeden případ užití může být prováděn více aktéry. Mezi aktéry lze také zavést generalizaci, kdy jeden aktér spouští stejné případy jako druhý a liší se pouze tím, že první může spustit některé další případy užití.

Případem užití rozumíme sadu scénářů, které ve výsledku vedou ke společnému cíli. Příkladem v navrhovaném systému může být jakýkoli nákup. Pokud se už od firmy nakupovalo, pouze se vybere ze seznamu. V opačném případě se založí nová. Podobně se postupuje u naskladňované položky – pokud je již položka stejného typu v databázi, pouze se vybere, jinak je nutné ji vytvořit. Všechny 4 kombinace scénářů vedou ke stejnému výsledku, kterým je nákup a naskladnění položky.

Kromě základního vztahu od aktéra k případů užití existují další dva typy vztahů mezi případy samotnými. Jedná se o relaci **include**, která označuje znovupoužitelnou část případů užití. Podřízený případ užití se využije vždy a je přímo volán z nadřazeného, jichž obvykle bývá více. Oproti tomu relace **extend** rozšiřuje případ užití o novou funkcionalitu, o které nemusí nadřazený případ vědět.

Z požadavků uživatele popsaných výše vznikl diagram případů užití 3.1.

[1] *land-parcel identification system*, DPB – díl půdního bloku, http://eagri.cz/public/app/lpisext/lpis/verejny2/plpis/



Obrázek 3.1: Diagram případů užití (zdroj: autor)

3.7 Existující řešení

Informačních systémů existuje nespočet, dokonce i těch specializovaných na zemědělství, avšak liší se kvalitou, komplexností a účelem použití.

Mezi nimi lze najít např. informační systém **ZeMan**® od firmy BM Servis[2]. Tento software se zaměřuje na celý podnik včetně správy zaměstnanců a mezd. Poskytuje také finanční řízení a prostředky pro správu živočišné výroby.

Do ERP systému (Enterprise Resource Planning, česky podnikový IS) **WinFAS** od firmy WinFAS software[3] lze doinstalovat modul pro zemědělce, který řeší rostlinnou i živočišnou výrobu, eviduje veterinární zákroky, zdravotní stav zvířat, osevní postupy a spolupracuje s daty z GPS a LPIS. Lze sledovat vozidla v reálném čase, evidovat pozemky a použitá hnojiva i postřiky.

Dalším podobným informačním systémem je **SQL Ekonom** od firmy Softbit Software[4], který poskytuje modul pro zemědělskou výrobu. Podporuje rostlinnou i živočišnou výrobu, propojení s LPIS a GQP moduly ve strojích pro automatické vytváření prací. Více se zaměřuje na chov prasat a skotu.

[2] http://www.bmservis.cz/is-zeman/
[3] https://www.winfas.cz/moduly-pro-zemedelce
[4] https://www.softbit.cz/informacni-system-zemedelska-vyroba

Pravděpodobně nejpropracovanějším je informační systém **IMES** od stejnojmenné společnosti[5]. Opět se jedná o univerzální řešení poskytující několik modulů, mezi kterými lze nalézt i modul pro zemědělství. Systém generuje veškerá potřebná hlášení o použitých chemikáliích, obsahuje propracovanou skladovou evidenci a plánování pro agronomy. Velmi zajímavým řešením je tzv. „nedokončená rostlinná výroba“, která zajišťuje, že je do účetnictví uměle připočten zisk z pole, které zatím nebylo sklizeno. Také se zde nachází správa živočišné výroby s automatickým výpočtem krmných dnů a dávek, třídění zvířat a zisků dle stájí či pastvin a vedením veterinárních zákroků u každého zvířete.

Dle ČSÚ [11] se pouze rostlinnou výrobou (stejně jako náš zemědělec) zabývá asi 30 % veškerých zemědělců a zemědělských podniků. Tato početná skupina nevyužije části výše popsaných informačních systémů pro rostlinnou výrobu, proto by pro ně bylo nevýhodné kupovat si celý informační systém, který toto obsahuje.

Navíc jsou všechna výše popsaná řešení komerčními produkty, jež jsou vytvořeny jako „běžné“ programy pro počítače, převážně k dispozici pouze pro platformu Windows. Autorovi není známo, že by existovala implementace informačního systému pro zemědělství jako webová aplikace.

[5] http://imes.cz/modul/zemedelstvi

Kapitola 4

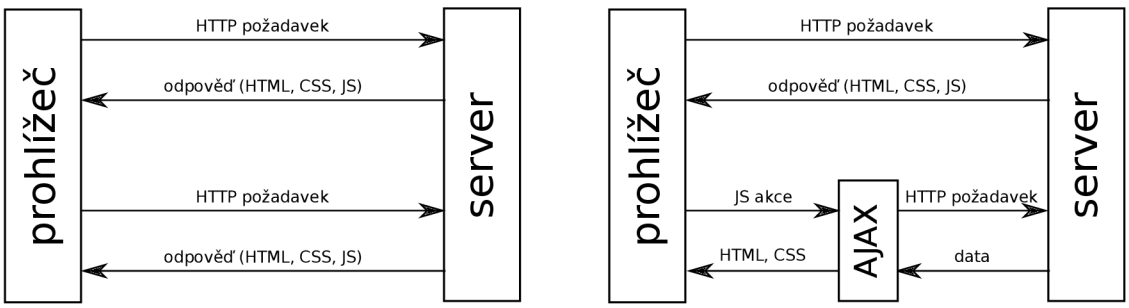
Použité technologie

Jelikož má být výslednou aplikací webový informační systém, bylo třeba zvolit vhodné technologie pro tento účel. Detailní popis všech použitých jazyků a technologií i jejich alternativ se nachází v kapitole 2.

4.1 Front-end

Pro front-end systému bude použit značkovací jazyk **HTML5** v kombinaci se stylopisem v jazyce **CSS3**. Nebude se využívat knihovny Bootstrap pro CSS, jelikož s ní autor nemá příliš dobré zkušenosti co se týká individuálního přizpůsobení vzhledu. Pro generování stránky je použit šablonovací systém **Latte**, který usnadňuje tvorbu HTML stránek s vnořenými PHP konstrukcemi a je součástí Nette frameworku (viz kapitola 4.2).

Dále bude na klientské straně využit skriptovací jazyk **JavaScript** a knihovna **jQuery**, která zjednodušuje jeho zápis a zajišťuje kombatibilitu napříč prohlížeči. JavaScript bude aplikován na kontrolu formulářových polí, dynamické skrývání nebo odkrývání částí obsahu a také pro načítání částí stránek nebo dat pomocí rozšíření **AJAX** – výhody tohoto řešení ukazuje obrázek 4.1.



Obrázek 4.1: Srovnání požadavků na server bez AJAXu a s AJAXem (zdroj: autor)

4.2 Back-end

Na straně serveru existuje více možností výběru, ovšem jak lze vidět na grafu na obrázku 2.3, nejpoužívanější technologií je jazyk **PHP**. Tento jazyk byl zvolen pro implementaci informačního systému, neboť s ním má autor nejvíce zkušeností, je dostupný téměř na

každém webovém serveru a existuje pro něj řada návodů a přehledná dokumentace i s příklady[1].

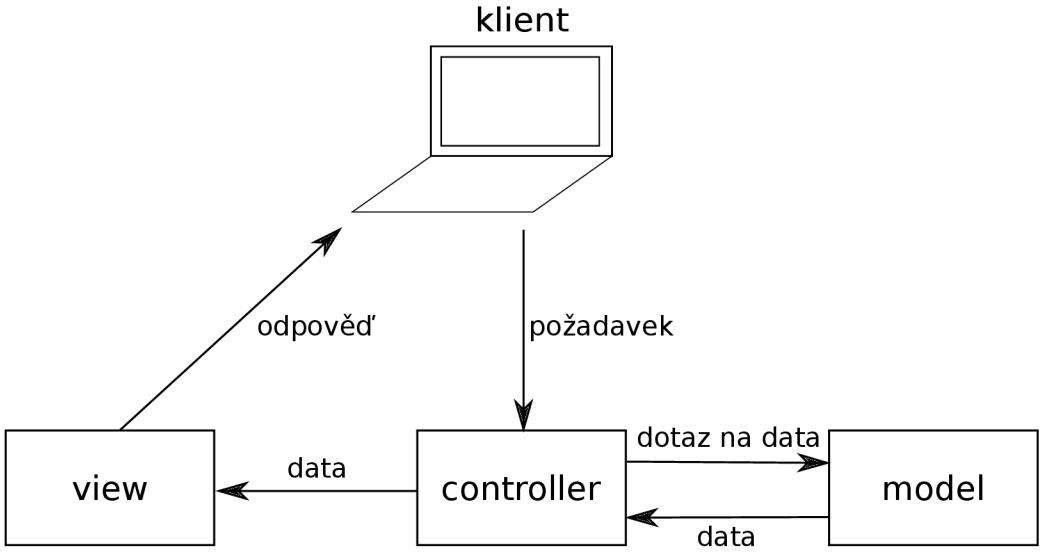
Jelikož se při vývoji webových aplikací mnoho věcí opakuje, bylo by velmi nevýhodné psát tato řešení několikrát. Proto vznikly různé frameworky (rámce), které je vhodné použít při vývoji aplikace větších rozměrů. Zvolen byl český framework **Nette**, jenž má přehlednou českou dokumentaci, rozsáhlou českou komunitu diskutující na fórech a obsahuje všechny potřebné funkce pro vývoj jednodušších i složitějších aplikací.

Velkou výhodou oproti nejčastěji používanému frameworku Laravel je například rozdělení vygenerované stránky do jednotlivých komponent a jejich následná invalidace či překreslení pomocí AJAXu. Laravel toto neřeší, jelikož se častěji používá pouze pro back-end a front-end se ve velkých projektech vytváří přes JavaScriptové frameworky.[13] Laravel je také pomalejší oproti například Symfony.[2] Narozdíl od Nette má sice obrovskou podporu, co se týče návodů v podobě videí (tzv „Laracasts“), ovšem tato forma dokumentace autorovi nevyhovuje, protože nelze používat vyhledávání a často je třeba kvůli jedné problematice zhlédnout několik videí, což je časově náročné.

Základem Nette je návrhový vzor MVC *(Model-View-Controller)*. Místo „controller“ užívá Nette slovo „presenter“. Popis návrhového vzoru [10]:

- **Model** - vrstva pracující s daty, je kompletně oddělena od zbytku aplikace a komunikuje pouze s presenterem
- **View** - front-end vrstva, vykresluje požadovaná data pomocí šablon a zobrazuje je uživateli
- **Presenter** - propojovací vrstva, zpracovává požadavky, dotazuje se modelu na data a vrací je zpět do view.

Proces zpracování požadavku je znázorněn na obrázku 4.2.



Obrázek 4.2: Schéma architekruty MVC (zdroj: autor)

[1] https://www.php.net/manual/en/

4.3 Databáze

Pro uchovávání dat aplikace bylo zvoleno **MySQL**, které je dostupné na většině webových hostingů zdarma a poskytuje veškeré potřebné funkce, jež jsou v této aplikaci vyžadovány. MySQL se v neplacené variantě zaměřuje na rychlost za cenu omezení pokročilé funkcionality, která ovšem pro běžné užití není nutná.[1]

Pro administraci databáze byl použit nástroj **phpMyAdmin**, který je napsaný v jazyce PHP a umožňuje jednoduchou správu MySQL databází.

4.4 Knihovny

Kromě výše zmiňovaných prostředků byly při vývoji aplikace použity některé volně dostupné knihovny. Každá z nich řeší konkrétní problematiku, je na ni odladěná a tudíž by bylo neefektivní vymýšlet ji znovu.

V tabulce 4.1 je výčet všech použitých knihoven s jejich stručným popisem. Ve zdrojových souborech je u načítání knihovny napsán vždy komentář s jejím názvem a odkazem na zdroj.

Název	**Popis**	**Licence**
SmallPop[2]	Malá oznamovací okénka *(flash zprávy)*	MIT
Select2[3]	Selectboxy s vyhledáváním, načítáním vzdáleného obsahu a dalším vylepšením	MIT
sorttable[4]	Řazení záznamů v tabulkách kliknutím na záhlaví pomocí JS	MIT
nette.ajax.js[5]	Nette – AJAX rozšíření	MIT
FormsReplicator[6]	Nette – přidávání neomezeného počtu „podformulářů“ do formulářů	BSD, GPLv3
DependentSelectBox[7]	Nette – závislé selectboxy	MIT
mPDF[8]	Generování PDF pomocí HTML a CSS	GPLv2

Tabulka 4.1: Přehled použitých knihoven (zdroj: autor)

Konkrétní případy použití jednotlivých knihoven budou popsány v kapitole 6, kde bude také podrobněji vysvětleno, k čemu konkrétní knihovna slouží, případně i stručně popsán způsob jejího používání.

[2] https://github.com/silvio-r/spop
[3] https://select2.org/
[4] https://kryogenix.org/code/browser/sorttable/
[5] https://github.com/vojtech-dobes/nette.ajax.js
[6] https://github.com/Kdyby/FormsReplicator
[7] https://github.com/NasExt/DependentSelectBox
[8] http://mpdf.github.io/

4.5 Další software

Pro vývoj aplikace bylo využito několik dalších programů, které však přímo nesouvisí s konkrétní aplikací, proto budou uvedeny zde.

4.5.1 Composer

Composer je nástroj pro správu závislostí (angl. *Dependency Manager*) v aplikacích napsaných v PHP. Umožňuje nadeklarovat, na kterých knihovnách je vytvořená aplikace závislá, následně je nainstaluje a aktualizuje. Také lze určit minimální potřebnou verzi těchto knihoven nebo verzi PHP. Často na něj spoléhají frameworky jako je Symfony, Laravel nebo i Nette. Veškerá konfigurace k projektu je uložena v kořenové složce v souboru `composer.json`, domovskou složkou pro nainstalované knihovny je složka `vendor`.[22]

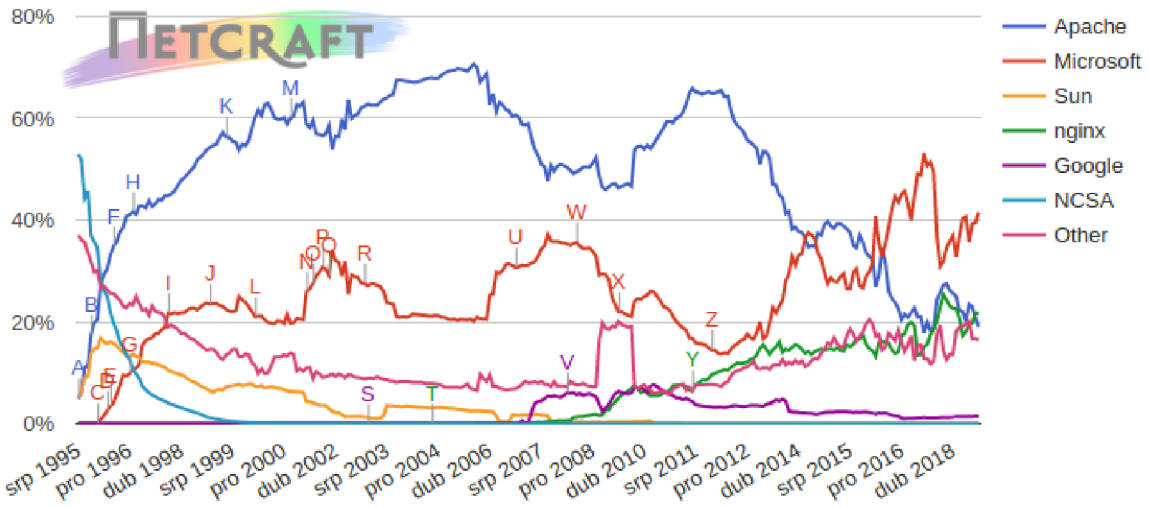
4.5.2 Git

Systém správy verzí git byl vyvinut v roce 2005. Za jeho tvůrce je považován Linus Torvalds, jeden z hlavních vývojářů linuxového jádra. Git umožňuje přístup k historii projektu, větvení, paralelní vývoj a nevyžaduje přístup k internetu. Veškeré změny lze kdykoli vrátit zpět nebo se k nim opět vracet.[37]

Pro sdílení repozitářů a spolupráci v týmech existuje spoustu veřejně dostupných serverů, které implementují webové rozhraní pro git. Mezi největší z nich patří GitHub, Bitbucket nebo GitLab.

4.5.3 Server Apache

Apache HTTP server byl poprvé vydán v roce 1995 jako open-source projekt. Jedná se o multiplatformní server, který nabízí všechny standardní HTTP služby.[36] Jak ukazuje graf na obrázku 4.3, dnes na něm funguje asi 20 % webových stránek (přibližně 74 milionů domén), ale ještě v roce 2012 to bylo 65 % všech webů. Apache server byl použit pro lokální vývoj aplikace a běží také na zakoupeném hostingu.



Obrázek 4.3: Podíl jednotlivých webových serverů na internetu (zdroj: [21])

Kapitola 5

Návrh informačního systému

Po provedené analýze požadavků na informační systém v kapitole 3 a výběru vhodných technologií v kapitole 4 je možné systém navrhnout a poté implementovat.

Pro návrh datové části informačního systému se používají dva modely – logický a fyzický datový model. **Logický model** (diagram entit nebo entitně-vztahový model) je nezávislý na implementaci. Oproti tomu **fyzický datový model** (schéma relační databáze) zohledňuje zvolenou relační databázi.[14, str.103-104]

Při popisu modelovacích technik UML *(Unified Modeling Language)* bylo v následujících kapitolách čerpáno z [14, str.103-121]. V kapitole 5.2 bylo při transformaci logického modelu na fyzický čerpáno z [14, str.110-121], [43]

5.1 Entitně-vztahový model

K návrhu databáze se využívá entitně-vztahový model (ER diagram). Jedná se o konceptuální model, který znázorňuje jednotlivé objekty reálného světa (entity) a vztahy mezi nimi.

Entita je identifikována názvem a obsahuje libovolný počet vlastností, neboli atributů. **Atributy** jsou jednoduché nebo složené, např. „jméno osoby“ je složený atribut obsahující jednoduché atributy „křestní jméno“ a „příjmení“.

Mezi entitami se nachází **vazby** (vztahy), které mohou být tří typů:

- Vazba 1..1 – nejjednodušší vazba vyjadřuje vztah jedné entity na právě jednu jinou entitu a opačně, velmi často vede ke sloučení entit
- Vazba 1..N – nejčastější typ, entita, u které je N (nebo *), je označována jako *detail*, a platí pro ni, že náleží právě k jedné *master* entitě, ovšem naopak k *master* entitě náleží libovolný počet (i nula) *detail* entit
- Vazba M..N – vyjadřuje, že k jedné entitě jednoho typu může náležet libovolný počet (i nula) entit druhého typu a opačně k jedné entitě druhého typu může náležet libovolný počet (i nula) entit prvního typu

V následujících kapitolách bude odkazováno na entitně-vztahový model, který zachycuje obrázek 5.1 na straně 24. Pro jednodušší manipulaci s daty mají všechny entity jako primární klíč automaticky generované ID.

5.1.1 Pole

Pole zemědělec identifikuje podle názvu, každé pole má rozlohu (plochu), pro kterou se v zemědělství jako základní jednotka používá hektar, a pro administrativní účely je popsáno číslem DPB ze systému LPIS.

Pro uchování roků na polích byla zvolena tzv. slabá entitní množina, která je identifikována rokem a váže se k entitě „Pole“. Ta musí držet informaci o zaseté plodině a odrůdě.

5.1.2 Sklad

Skladové položky jsou identifikovány názvem, každá položka má umístění a je určitého typu (plodina, postřik, hnojivo, osivo, palivo, jiné). Na skladě je množství v jednotkách, které mohou být pro každou ploužku jiné (kilogramy, litry, kusy).

Pokud je položka typu plodina, musí obsahovat také identifikaci plodiny (její název a odrůdu). Tento případ je v ER diagramu vyjádřen pomocí generalizace.

Položky se naskladní pomocí nákupů. U každého nákupu se bude evidovat dodavatel, datum, nakoupené množství a cena za jednotku. Jeden nákup bude obsahovat právě jednu položku, ta ale může být nakoupena v několika různých nákupech. Podobně se budou uchovávat prodeje plodin.

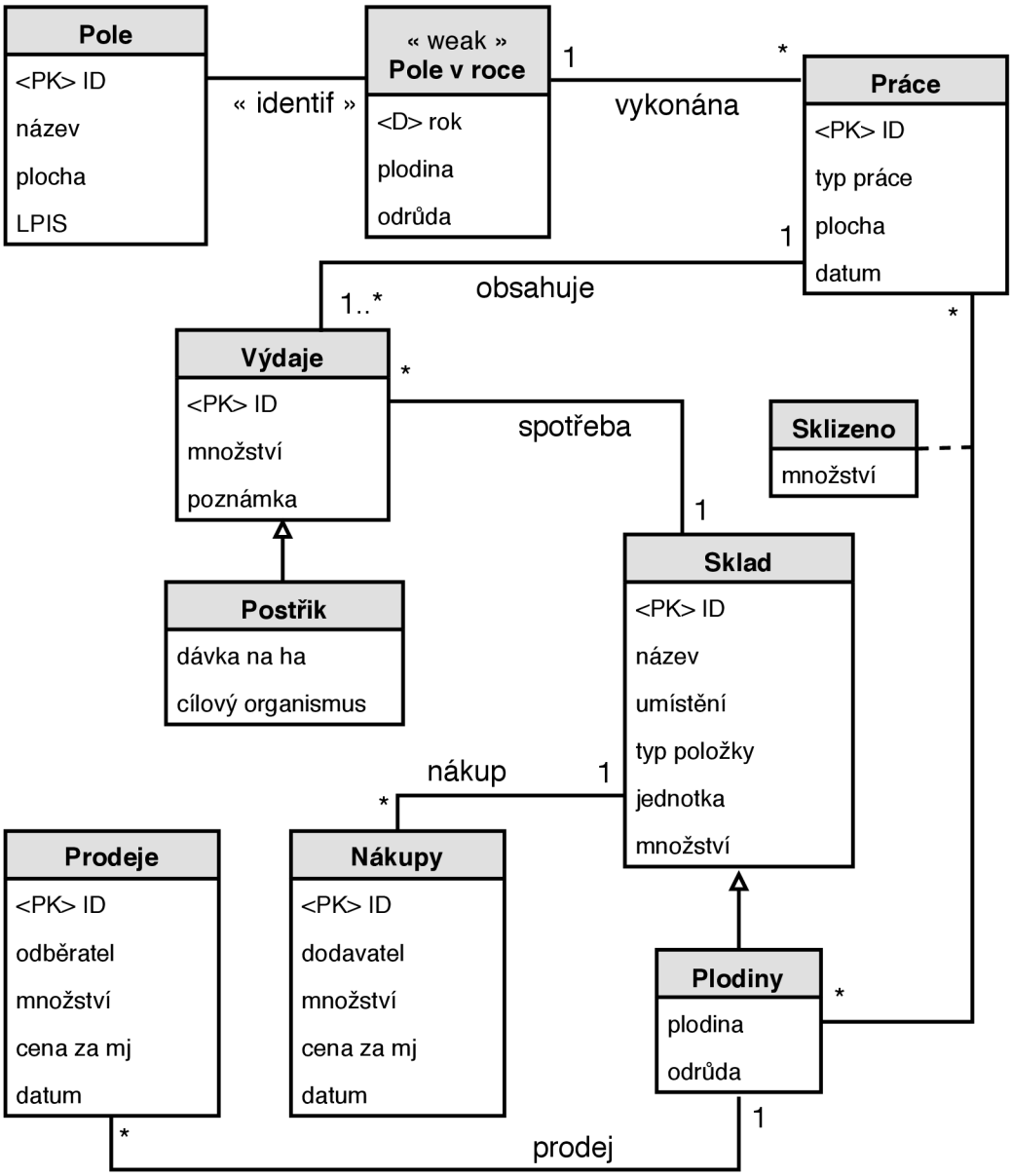
Bylo zvoleno ukládání ceny za položku z důvodu následného provádění analýzy a počítání výdajů na polích. Při tomto procesu se množství spotřebované při práci pouze vynásobí jednotkovou cenou, za kterou byl materiál pořízen. Pokud by se ukládala celková suma nákupu, bylo by nejdříve potřeba z této sumy a nakoupeného množství zjistit onu jednotkovou cenu, což by znamenalo jedno dělení navíc.

5.1.3 Práce

Každá práce obsahuje datum vykonání, typ (sklizeň, orba, podmítka, setí, hnojení, postřik, válení) a obdělanou plochu. Vždy musí být svázána právě s jedním polem v určitý rok, ovšem na poli může být proveden libovolný počet prací.

S prací souvisí výdaje, které vznikly v průběhu jejího vykonávání. U nich systém eviduje spotřebované množství a poznámku. V případě, že se jedná o postřik, obsahuje výdaj také dávku na hektar a údaj o cílovém organismu. Toto je v ER diagramu vyjádřeno pomocí generalizace. Práce musí obsahovat minimálně jeden výdaj a konkrétní výdaj se váže pouze k jedné práci. Výdaj je také spojen se skladovou položkou, kterou spotřebovává (odečítá množství ze skladu).

Pokud se jedná o sklizeň, váže se k práci množství sklizené plodiny, která se přidá na sklad. Jelikož může být jedna položka sklizena během několika prací a během práce může být sklizeno několik položek, je v ER diagramu tento vztah vyjádřen vazební entitou „sklizeno“, která obsahuje pouze sklizené množství.



Obrázek 5.1: Entitně-vztahový model (zdroj: autor)

5.2 Schéma relační databáze

Entitně-vztahový model není ideálním vzorem pro vytváření databáze. Jelikož velmi často slouží pro komunikaci se zákazníkem, musí zobrazovat data v lidsky srozumitelné formě a také obsahuje spoustu redundantních informací.

Při transformaci logického modelu dat na fyzický je třeba správně převést typy vztahů mezi entitami, vyřešit generalizace a specializace a splnit tzv. „normální formy“. Těmito kroky lze dospět až k výslednému schématu databáze, které je na obrázku 5.2 na straně 27.

5.2.1 Transformace vztahů

Vztahy mezi entitami v ER diagramu lze poměrně jednoduše převést do logického schématu podle následujících pravidel:

- Vztah 1..1 – v jedné z tabulek se vytvoří odkaz na záznam druhé tabulky, tzv. cizí klíč *(foreign key)* a atributy vztahu se přidají k libovolné z nich
- Vztah 1..N – v tabulce, u které se nachází N, se vytvoří cizí klíč do druhé tabulky a přebírá také všechny atributy vztahu
- Vztah M..N – vytvoří se vazební tabulka, která bude obsahovat pouze cizí klíče vedoucí do obou tabulek, případně atributy vztahu

5.2.2 Generalizace a specializace

Při transformaci generalizace/specializace máme tři možnosti (N označuje počet podtypů):

- Vytvořit tabulku pro nadtřídu a N tabulek pro podtřídy s odkazem na nadtřídu
- Vytvořit N tabulek pro podtřídy obsahující také atributy nadtřídy
- Všechno v jedné tabulce obsahující sjednocení

Výběr vhodné techniky je nutné přizpůsobit vždy konkrétnímu případu. Zvlášť pečlivě se musí promyslet, jaké dotazy se budou nad danou entitou nejčastěji vykonávat. Poté je třeba se rozhodnout, zda bude vhodnější použít více tabulek, zavést redundantní data a nebo jednu velkou tabulku.

Pro entity „výdaje“ i „sklad“ existuje pouze jedna specializace, proto se třetí řešení s jednou tabulkou zdá jako nejlepší možné a bude využito.

5.2.3 Slabá entitní množina

Transformace slabé entitní množiny do logického modelu lze vyřešit převodem na silnou entitní množinu se vztahem ke své identifikující množině 1..N. Primární klíč bude složený z cizího klíče na identifikující množinu a diskriminátoru.

Tato metoda má ovšem za následek, že v každé entitě, která se bude odkazovat na tuto slabou entitu, musí být oba dva atributy cizími klíči. Proto při převodu entity „pole v roce“ byl zaveden automaticky generovaný primární klíč jako u všech ostatních, diskriminátor se stal běžným atributem a vztah k identifikující množině byl zachován přes cizí klíč.

5.2.4 1. normální forma

První z normálních forem, kterou je třeba splnit, zajišťuje atomičnost dat. Relační databáze neumí ukládat datové struktury a tudíž je třeba složené atributy převést na jednoduché.

V entitně-vztahovém modelu 5.1 lze u entit „prodeje“ a „nákupy“ nalézt atribut **odběratel**, resp. **dodavatel**. Jsou to složené atributy obsahující IČ, název firmy a adresu. Adresa je opět složený atribut, který obsahuje ulici a číslo popisné, PSČ a město.

5.2.5 2. normální forma

Druhá normální forma řeší funční závislosti dat na části složeného primárního klíče. Pokud je klíč složený ze dvou a více atributů a jiný neklíčový atribut přímo nezávisí na celém primárním klíči (ale pouze na jeho části), je třeba tento atribut vytknout do samostatné tabulky.

Jelikož ani v jedné tabulce nebyl použit složený primární klíč, je tato normální forma automaticky splněna.

5.2.6 3. normální forma

Podle třetí normální formy nesmí být atribut, který není primárním klíčem, tranzitivně závislý na žádném klíči. Jinými slovy tato definice říká, že žádný neklíčový sloupec nesmí být přímo závislý na žádném jiném neklíčovém sloupci.

Ve schématu se objevila tato tranzitivita v atributech **odběratel** a **dodavatel**, konkrétně název a adresa jsou závislé na IČ firmy, které je závislé na čísle nákupu, resp. prodeje. Pokud tedy budou existovat například dva nákupy od jednoho dodavatele, bude se v databázi dvakrát vyskytovat celá jeho adresa, název i IČ. Proto je nutné vytvořit novou tabulku (například „firmy“), která bude obsahovat všechny informace o dodavatelích i odběratelích a bude se do ní odkazovat pomocí cizího klíče přes IČ.

5.2.7 Boyceho-Coddova normální forma

Boyceho-Coddova normální forma (BCNF, někdy označována jako 3,5.NF) zpřísňuje 3. normální formu. Definice zjednodušeně udává, že „všechny závislosti, které nejsou triviální, musí být dány závislostmi na celých kandidátních klíčích“[43, str. 104].

V modelu se žádná netriviální závislost nevyskytuje, proto je možné prohlásit, že splňuje BCNF.

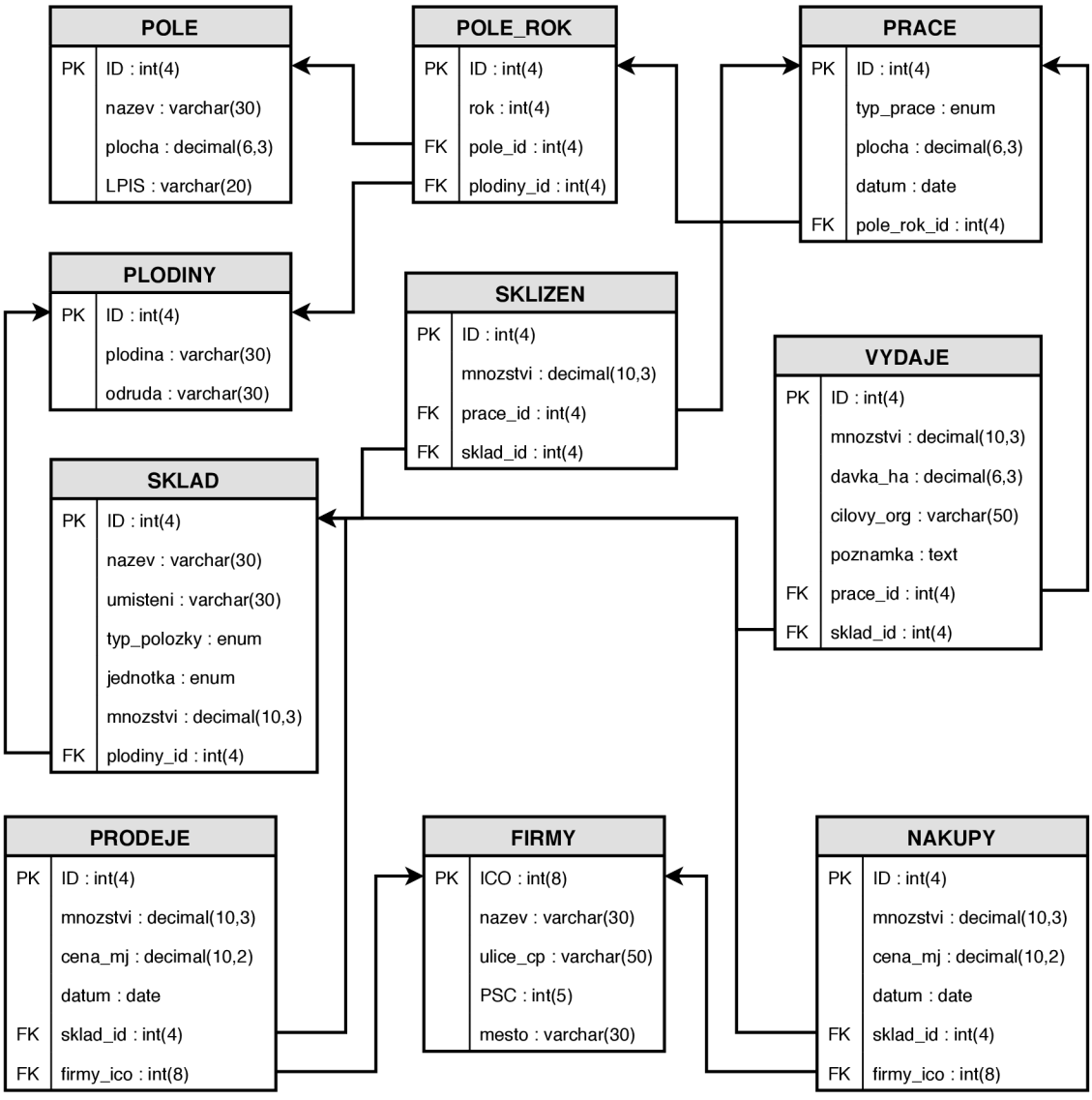
U této formy také většinou normalizace končí, ovšem existují minimálně další 2 normální formy. Čtvrtá normální forma řeší nezávislé vícehodnotové primární klíče a pátá normální forma zajišťuje, že tabulku již není možné bezeztrátově rozložit.

5.2.8 Další optimalizace

Po všech výše popsaných krocích byl logický návrh databáze téměř připraven, ale model trpěl ještě jedním nedostatkem, který se nepodařilo pomocí výše uvedených kroků odstranit. U entit „sklad“ a „pole v roce“ se nachází dvojice atributů **plodina** a **odrůda**. Téměř jistě se zde budou vyskytovat stejné údaje – například v případě, že na poli bude zaseta pšenice odrůdy „mulan“, určitě byla před setím na skladě jako osivo a po sklizni se zde objeví znovu jako plodina na prodej. Proto se vytvořila tabulka pro plodinu, obsahující tyto dva atributy a automaticky generovaný primární klíč, díky kterému se bude na plodiny

odkazovat jednodušeji. Tento krok vedl ke zmenšení dvou tabulek a zásadnímu omezení redundance dat.

Nette framework potřebuje pro pochopení vztahů mezi tabulkami speciální pojmenování atributů, mezi kterými se nachází cizí klíč. Atribut odkazující do jiné tabulky (cizí klíč) musí být pojmenován názvem tabulky a názvem pole, které obsahuje primární klíč. Jako oddělovač je možné použít například podtržítko a na pořadí nezáleží. Proto ve výsledném schématu databáze lze nalézt atributy jako **plodiny_id** (odkaz do tabulku plodiny přes ID plodiny), **firmy_ico** atp. Díky tomuto vylepšení umí Nette udělat automatické připojení tabulek (JOIN).



Obrázek 5.2: Schéma databáze – část OLTP (zdroj: autor)

5.3 Analytická část systému – OLAP

Analýza ekonomiky bude prováděna na základě příjmů a výdajů, které vznikly při vykonávání prací na polích. Tyto práce se vždy vážou ke konkrétní plodině na daném poli.

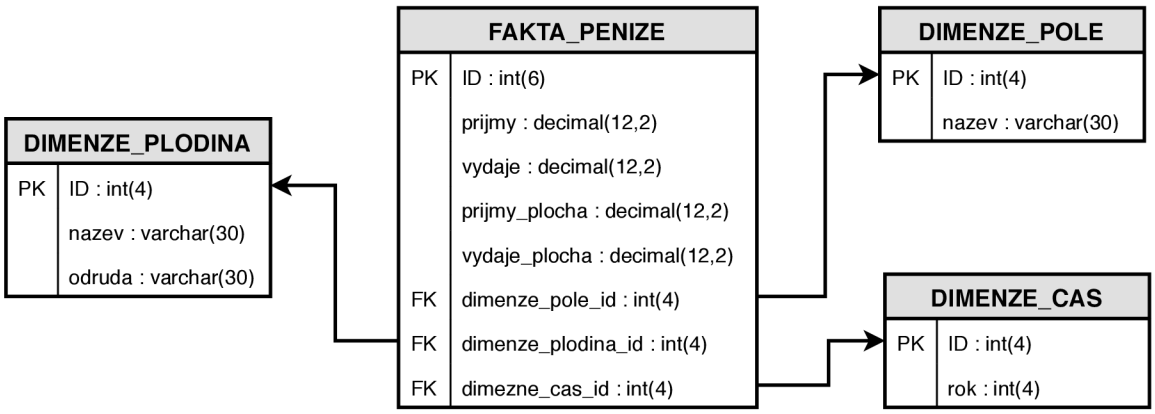
Dimenze budou celkem tři: **plodina**, **čas** a **pole**. U plodiny se bude uchovávat název a odrůda, přičemž analýza může probíhat nad druhem plodiny, tak i nad konkrétními odrůdami (zanoření). V dimenzi polí stačí uchovávat pouze název pole, kterým ho zemědelec nejsnáze identifikuje. Čas není třeba dělit na menší díly než roky. Provádět analýzu po měsících je zbytečné, jelikož jeden cyklus na poli (od přípravy, přes zasetí, chemickou ochranu, sklizeň a následné obdělání) trvá právě jeden rok.

Jak už bylo zmíněno, zajímá zemědělce finanční výsledek celého hospodaření. Proto vznikla tabulka faktů „peníze“, která bude obsahovat příjmy celkem, výdaje celkem, příjmy na plochu celkem a výdaje na plochu celkem. Uživatel si bude moci zobrazit tyto složky jednotlivě, nebo spočítané celkové sumy (příjmy−výdaje).

Atributy příjmy na plochu a výdaje na plochu byly zvoleny na přání zemědělce. Jelikož je každé pole jinak velké, používá se pro srovnávací účely právě výnos z jednotky plochy (hektaru). Více informací o konzultacích s farmářem bude uvedeno v kapitole 7 o testování.

Výsledné schéma OLAP části lze vidět na obrázku 5.3. Bylo zvoleno schéma „hvězda“, jelikož se zde nenachází tolik zanořování, aby se vyplatila „sněhová vločka“.

Proces **ETL**, který převádí data z OLTP části systému do OLAP části, je spouštěn na pokyn od uživatele, konkrétně při uložení práce na poli, která podle něj ukončuje aktuální rok a po níž následuje rok nový s jinou plodinou. Tato práce se uloží do databáze, následně se pro daný rok na poli a plodinu spočítají veškeré příjmy a výdaje (na základě prodejů a nákupů) a výsledné částky se uloží do tabulky faktů.



Obrázek 5.3: Schéma databáze – část OLAP (zdroj: autor)

Kapitola 6

Implementace

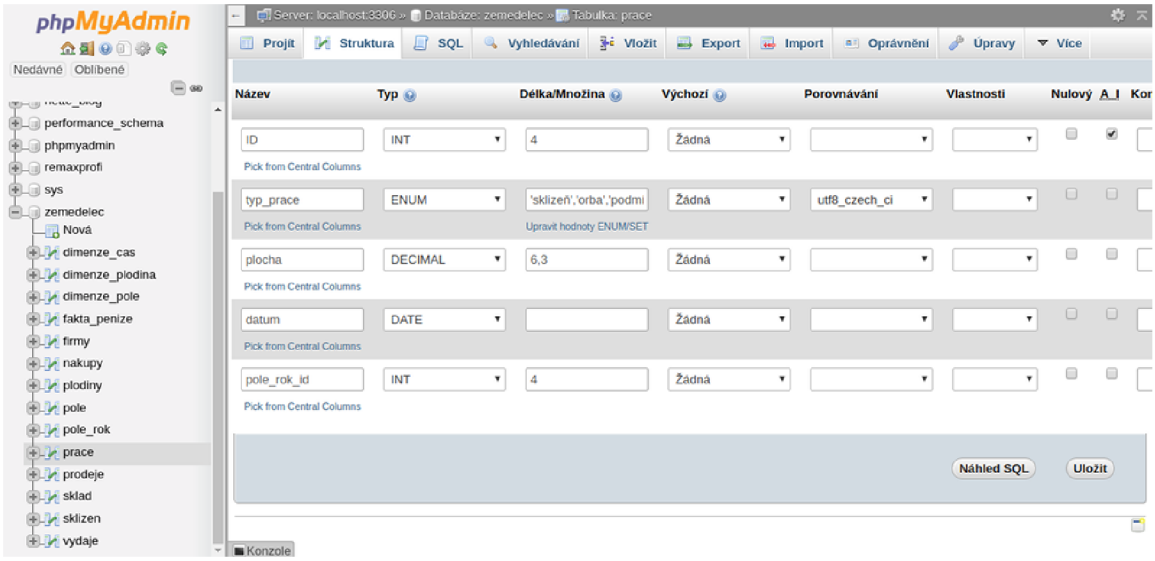
Na základě kapitol 3 a 5 lze nyní vytvořit výsledný informační systém za použití technologií vybraných v kapitole 4. Systém bude popsán po jednotlivých vrstvách směrem od databáze k uživateli a budou uváděny příklady zdrojového kódu u zajímavých řešení.

6.1 Databáze

Na základě diagramu 5.2 byla vytvořena MySQL databáze v uživatelsky příjemném prostředí phpMyAdmin (obrázek 6.1). Zde byla na lokálním stroji založena nová databáze a postupně po tabulkách nadefinována jejich struktura včetně vazeb mezi nimi.

Bylo použito úložiště **InnoDB**, které oproti výchozímu úložišti MyISAM podporuje transakce, uzamykání při souběžném přístupu operuje na úrovni řádků a obsahuje podporu pro cizí klíče. U obsáhlejších databází je pomalejší než MyISAM, ovšem tento problém je v menším informačním systému zanedbatelný.[23]

Jelikož bude informační systém nasazen také na veřejně přístupný webový server, je vhodné nechat si vygenerovat v nástroji phpMyAdmin inicializační SQL skript a ten poté nahrát na server. Tím je zajištěno, že obě databáze budou mít úplně stejnou strukturu.



Obrázek 6.1: Nástroj phpMyAdmin na lokálním serveru (zdroj: autor)

6.2 Modely

Nejnižší vrstvu MVC architektury Nette tvoří modely a všechny je lze nalézt ve složce `app/model`. Nad databází byly vytvořeny dvě vrstvy – **entity** a **managery**. Výhodou tohoto přístupu je, že vyšší vrstvy aplikace nemusí znát strukturu databáze a pracují přes managery přímo s entitami. Mapování entit a managerů k jednotlivým databázovým tabulkám zobrazuje tabulka 6.1.

Tabulka	Entita	Manager
pole	`Field`	`FieldManager`
pole_rok	`FieldYear`	
plodiny	`Crop`	`CropManager`
firmy	`Company`	`CompanyManager`
prodeje	`Transaction`	`TransactionManager`
nákupy		
sklad	`Store`	`StoreManager`
sklizen	`Harvest`	`WorkManager`
vydaje	`Expense`	
prace	`Work`	

Tabulka 6.1: Databázové tabulky – mapování na entity a odpovídající managery (zdroj: autor)

6.2.1 Entity

Každá entita obsahuje všechny atributy z databázové tabulky. Atributy entity jsou privátní a přistupuje se k nim pouze pomocí getterů a setterů, jak je v objektově orientovaném programování obvyklé. Díky tomuto přístupu lze kontrolovat, co je do entit přiřazováno a odhalit chyby, které by například později nepřijala databáze. Vztahy mezi entitami je možné reprezentovat přímo referencí na danou entitu, což významě ulehčuje a zpřehledňuje práci.

Například třída `Store` pracuje s položkou skladu. Obsahuje atributy totožné s databázovou tabulkou „sklad“ a navíc místo ID plodiny obsahuje objekt třídy `Crop`, což je opět třída odpovídající databázové tabulce „plodina“. Nyní lze přistupovat ke všem atributům pomocí getterů, např. `$storeItem->getLocation()`. Díky referenci na objekt plodiny lze její název zjistit pomocí `$storeItem->getCrop()->getName()`.

Pokud je třeba naopak ukládat informace, lze je nastavit entitě pomocí setterů. Například pro změnu množství na skladě lze jednoduše napsat `$storeItem->setAmount(6)`. Totéž lze aplikovat opět i na odkazované entity – například změna odrůdy plodiny se zapíše jako `$storeItem->getCrop()->setVariety("apache")`.

Kromě getterů a setterů obsahují entity také funkci `toJSON()`, která celý objekt serializuje do formátu JSON. Využití této funkce je hlavně při použití AJAXu, kdy je třeba do stránky vypsat informace o nějaké databázové entitě, ale nechceme kvůli tomu znovu načítat celou stránku.

6.2.2 Managery

Managery představují kombinaci dvou návrhových vzorů – *repository* a *mapper*. Mapper slouží k transformaci databázového záznamu (v případě Nette instance `ActiveRow`, viz dále) na entitu a naopak. Repozitář se používá při přístupu k databázi z jiných částí aplikace, využívá služeb mapperu a provádí s entitami převážně tzv. CRUD operace *(Create, Read, Update, Delete)*, ale vrací například i různé seznamy, výběry nebo konkrétní hodnoty.

Jak bylo popsáno výše, každý manager obsahuje také mapovací funkce. Ty jsou konkrétně dvě – `ActiveRow` na entitu a entita na asociativní pole. Tyto funkce jsou vidět na úryvku zdrojového kódu 6.1. Nette v databázových funkcích `insert()`, `update()` apod. vyžaduje asociativní pole, kde je klíčem název sloupce a hodnotou požadovaná data. Naopak z databáze vrací instanci třídy `Nette\Database\Table\ActiveRow`, kterou lze indexovat stejně jako asociativní pole názvem sloupce, nebo k ní lze přistupovat jako k objektu přes operátor šipky (např. `$row->nazev`). Nad objektem tohoto typu lze také přímo provádět akce `update()` a `delete()`, které se okamžitě propíší do databáze.

Na řádku 17 ve zdrojovém kódu 6.1 je vidět, že v konstruktoru skladové položky na místo plodiny přiřazuje mapovací funkce vždy `null`, což se praktikuje z důvodu optimalizací. Pokud by se vždy z databáze zjišťovalo, jestli a jakou má skladovací položka přiřazenou plodinu, znamenalo by to pokaždé dotaz navíc. V případě, že se bude získávat jedna konkrétní položka, by to nevadilo, ale pokud by byla tato funkce využitá při vytváření seznamu skladových položek, dotazů by bylo tolik, kolik by bylo těchto položek v seznamu. Proto je na volajícím této funkce, aby přiřadil ke skladovací položce plodinu dodatečně. Pokud bude dávat položky do seznamu, načte si z databáze všechny plodiny a poté je na základě ID připojí ke skladové položce. Tento princip se opakuje ve všech mapovacích funkcích napříč managery.

```
1  // prevod entita Store => databaze
2  private function storeToDB(Store $store) {
3    return [
4      "ID" => $store->getID(),
5      "nazev" => $store->getName(),
6      "umisteni" => $store->getLocation(),
7      "typ_polozky"=> $store->getType(),
8      "jednotka" => $store->getUnit(),
9      "mnozstvi" => $store->getAmount(),
10     "plodiny_id" => $store->getCrop() ? $store->getCrop()->getID() : null,
11   ];
12 }
13
14 // prevod databaze => entita Store
15 private function dbToStore(ActiveRow $row) {
16   return new Store($row["ID"], $row["nazev"], $row["umisteni"], $row["typ_polozky"],
17                    $row["jednotka"], $row["mnozstvi"], null);
18 }
```

Zdrojový kód 6.1: Ukázka mapovacích funkcí

Zdrojový kód výše byl pro lepší čitelnost zjednodušen. Ve skutečnosti nejsou názvy sloupců tabulek (ID, nazev, umisteni atd.) zapsány natvrdo ve všech funkcích. Každý manager má definované konstanty pro název tabulky a všechny její sloupce, poté se přistupuje všude do databáze pomocí těchto konstant. Pokud by se změnil název některého sloupce, stačí pouze změnit hodnotu dané konstanty.

Každý manager je zaregistrován jako služba *(service)*, takže je možné si nechat kdykoli předat jeho instanci v konstruktoru například presenteru. Tomuto předávání závislostí se

říká *Dependency Injection* a Nette ji hojně využívá. Stejným způsobem také každý manager dostane instanci databázového kontextu – všechny managery dědí od třídy `BaseManager`, který si v konstruktoru vyžádá `Nette\Database\Context`.

6.2.3 ETL a OLAP

Pro analytickou část systému vznikly speciálně dvě třídy. Obě navíc dědí od třídy `BaseAnalysis`, která kromě přístupu k databázovému kontextu uchovává názvy tabulek pro analýzu a jejich polí.

Třída `ETL` implementuje proces ETL *(Extract, Transform, Load)*, tzn. sbírá, transformuje a nahrává data z OLTP části systému do OLAP části. Obsahuje pouze jednu veřejnou metodu `runOnField(FieldYear $field)`, která jako parametr potřebuje instanci pole v roce, pro které provede výpočet příjmů a výdajů.

Nejdříve se zkontroluje, že na poli byla zasetá plodina, poté se ověří, že toto pole pro daný rok a s danou plodinou ještě nebylo počítáno, a pokud je třeba, vytvoří se v dimenzích záznamy pro rok, pole a plodinu. Na závěr se pro každou vykonanou práci spočítá celková suma výdajů, jež během ní vznikly, a příjmů, které byly vygenerovány při sklizních. Částka se u výdajů spočítá tak, že se spotřebované množství vynásobí jednotkovou cenou, která je určena nejbližším nákupem daného materiálu směrem zpátky. Například pokud byla 3. 10. zaseta pšenice, nalezne se nejbližší nákup osiva, který se uskutečnil před tímto dnem (např. 29. 9.), a z něj se vypočítá celková částka. Příjmy se spočítají podobně, jen se hledá nejbližší pozdější datum prodeje. Do databáze se uloží tyto dvě výsledné částky a navíc ještě vydělené plochou pole, aby bylo možné porovnávat příjmy a výdaje na plochu.

Pro přípravu výsledků na zobrazení slouží třída `AnalysisManager`, jež obsahuje několik veřejných metod. Každá z nich se zaměřuje na kombinaci dvou dimenzí a přijímá filtr pro tyto dimenze. Metoda pracující s plodinami vyžaduje množinu plodin, které si uživatel přeje zobrazit, metoda pracující s časem vyžaduje interval roků, které se mají zobrazit, a metoda pracující s poli vyžaduje množinu polí, které chce uživatel vidět. Všechny tyto metody vrací několik asociativních polí, mezi kterými je hlavička tabulky, jednotlivé řádky, řádek se součty, sloupec se součty a případně také detail, jenž se uplatní u plodin a jejich odrůd.

6.2.4 Pomocné třídy

Jak už bylo dříve zmíněno, všechny managery dědí od třídy BaseManager, která poskytuje databázový kontext v privátní proměnné `$db`. Kromě toho si také v konstruktoru pomocí DI *(Dependency Injection)* vyžádá instanci třídy `Transactor`. V této třídě je implementováno řízení dazabázových transakcí. V programu se objevuje hned několik příkazů, které jsou sdružené do transakcí. Ovšem tyto příkazy v sobě také mohou mít transakci a jelikož může být otevřena pouze jedna transakce v jeden čas a Nette nerozlišuje, kdo ji spustil, je třeba vytvořit čítač „běžících“ transakcí. Ten spočítá zanoření transakce a pokud se program vynoří zpět na nejvyšší úroveň, povolí ji uzavřít.

Mezi modely bylo také implementováno několik výjimek, jmenovitě jsou to tři třídy – `DataException`, `DatabaseException` a `LogicException`. Jak název napovídá, výjimka `DataException` je vyvolána při chybné práci s daty. K těmto chybám může dojít například při chybném formátu čísla, IČa, neexistující hodnotě výčtu atp. `DatabaseException` je vyvolána kdykoli se vyskytne chyba při práci s databází. Může se jednat o porušení integrity dat, o přístup k neexistujícímu záznamu apod. Výjimka `LogicException` se může vyskytnout při logické chybě při práci s programem. Například pokud bylo uzavřeno pole

bez osetí. Tyto chyby by se v produkčním nasazení neměly objevit, v nejhorším případě by měly být včas odchyceny a uživateli by se měla zobrazit smysluplná chybová hláška.

6.3 Presentery

V terminologii Nette představují presentery vrstvu controllerů v návrhovém vzoru MVC. Jejich úkolem je tedy přijímat požadavky od klienta, rozhodnout, co se provede a zavolat metody modelů. Ve finální fázi předají výsledky operací z modelů do vrtev pohledů *(view)*. Všechny presentery jsou uloženy ve složce `app/presenters` a jsou pojmenovány dle vzoru `<Název>Presenter`.

6.3.1 Router

Než se dostane klientský požadavek ke konkrétnímu presenteru, musí jej zpracovat směrovač *(router)*. Klientský požadavek má totiž tvar HTTP(S) dotazu, což je velmi primitivní protokol a aplikace z něj nezíská moc informací. Jednou z nejdůležitějších částí tohoto dotazu je cílová URL adresa, na niž se podívá právě implementovaný směrovač a rozhodne na základě svých pravidel, jaká akce presenteru se má vykonat.

Následuje ukázka vytvoření směrovače, které je v lehké modifikaci také použito ve vytvářeném informačním systému. Ve zdrojovém kódu 6.2 je nejdůležitější řetězec v konstruktoru cesty na řádku 3. Tento řetězec předepisuje tvar cesty, kterou směrovač přijímá a zároveň určuje, který presenter, která akce a případně s jakým parametrem se má vyvolat. Konkrétně v tomto případě označuje první část URL název presenteru, druhá část za lomítkem název akce a třetí část se do metody presenteru předá jako parametr `$id`. Uzavření částí cesty do hranatých závorek značí nepovinnost a parametr za znakem „=“ určuje výchozí hodnotu při explicitně nezadané.

```
1  public static function createRouter() {
2    $router = new RouteList;
3    $router[] = new Route("<presenter=Homepage>[/<action=default>[/<id>]]");
4    return $router;
5  }
```

Zdrojový kód 6.2: Vytvoření směrovače

Doména, na které aplikace běží, se do cest běžně neuvádí. Tím vzniká univerzální aplikace bez problému přenositelná na jiné domény. Pokud budeme uvažovat například testovací doménu `domena.xy`, bude tento směrovač přijímat následující URL:

- `domena.xy`
  presenter: HomepagePresenter, akce: default, parametr: null
- `domena.xy/store`
  presenter: StorePresenter, akce: default, parametr: null
- `domena.xy/store/add`
  presenter: StorePresenter, akce: add, parametr: null
- `domena.xy/store/edit/19`
  presenter: StorePresenter, akce: edit, parametr: 19

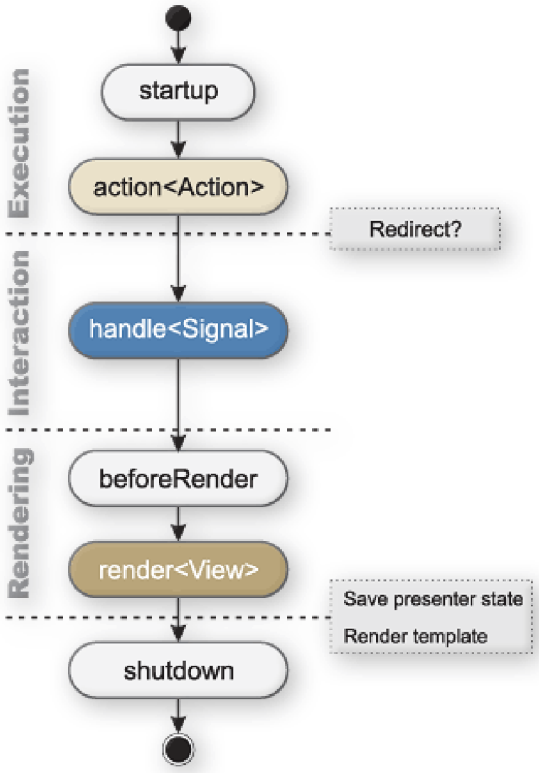
Ve výsledné aplikaci je ve směrovači navíc nastaven překlad URL adres do českého jazyka, což je zařízeno dalším parametrem konstruktoru cesty, který akceptuje asociativní

pole, kdy klíčem je název presenteru či akce dle pojmenování v aplikaci a hodnotou je český překlad. Výhodou tohoto způsobu je také to, že lze překlad vytvořit dodatečně, jelikož se zachová funkčnost původních pojmenování.

Silnou stránkou Nette je obousměrné směrování, což usnadňuje práci napříč celou aplikací. Nikde totiž není potřeba zadávat konkrétní URL (například v odkazech, v menu, při přesměrovávání v presenteru apod.). Stačí se odkazovat přímo na konkrétní presenter (`Store:`), na jeho akci (`Store:add`), nebo do odkazu vložit i parametr (`Store:edit 19`). Překlad na výsledná URL v pohledech zajišťuje Latte makro `{link "cesta"}` (viz kapitola 6.4.2).

6.3.2 Životní cyklus presenteru

V Nette má každý presenter až 6 základních metod, kterými při svém zavolání prochází. Graficky je životní cyklus presenteru znázorněn na obrázku 6.2 získaném z oficiální Nette dokumentace.



Obrázek 6.2: Životní cyklus presenteru (zdroj: [9])

Metoda `startup()` je zavolána ihned po vytvoření presenteru. Často se využívá pro kontrolu oprávnění uživatele, aby se zamezilo nechtěnému spuštění nějaké akce.

V metodě `action<Akce>()` je vhodné provádět operace, které přímo nesouvisí se zobrazováním obsahu. Jsou to například přesměrování, změna výchozí šablony (více v kapitole 6.4.1), načtení údajů do formulářů, zápis do databáze a nebo změna vykreslovací metody pomocí `$this->setView("nazev")`. Tato metoda tudíž může zásadně ovlivnit zbytek života presenteru.

Lehce speciální je metoda `handle<Signál>()`. Ta primárně obsluhuje AJAXové požadavky, které přijdou presenteru. Může se jednat o čistý AJAX požadavek, jehož výsledkem je odeslání JSON dat a ukončení vykonávání akcí presenteru, nebo se může jednat o překreslení částí zobrazené stránky (tzv. *snippet*).

Obecná metoda `beforeRender()` je volána vždy před vykreslením stránky bez ohledu na konkrétní akci presenteru. Používá se k předání parametrů do šablon, které jsou společné pro více pohledů (např. jméno přihlášeného uživatele, obsah košíku atd.). Také se zde implementují vlastní Latte filtry, jenž budou popsány v kapitole 6.4.3.

Nejvíce využívanou metodou je `render<Pohled>()`, která předá všechny potřebné proměnné a parametry do šablony a po svém ukončení spustí její vykreslení.

Na závěr je činnost presenteru ukončena pomocí metody `shutdown()`, která nemá žádné speciální využití.

6.3.3 Presentery v aplikaci

Jak již bylo zmíněno na začátku kapitoly, pojmenování i umístění presenterů se řídí konvencí `app/presenters/<Název>Presenter`. Jelikož je Nette velmi tvárný framework, lze zvolit jiné názvy, ovšem je třeba upravit konfigurační soubor `app/config/config.neon`, aby bylo možné presentery najít. V tabulce 6.2 jsou vypsány všechny vytvořené a používané presentery v aplikaci včetně jejich stručného popisu.

Název	Popis
`AnalysisPresenter`	Zobrazování analýzy, obsluha AJAX překreslování
`CompanyPresenter`	Výpis a administrace firem
`CropPresenter`	Obsluha AJAX požadavků na získání názvů a odrůd plodin
`FieldPresenter`	Výpis a administrace polí
`HomepagePresenter`	Výchozí stránka aplikace
`SprayPresenter`	Zobrazení využitých postřiků a generování hlášení
`StorePresenter`	Zobrazení položek na skladě, administrace skladu
`TransactionPresenter`	Výpis transakcí (nákupů a prodejů) a jejich vytváření
`WorkPresenter`	Přehled prací na polích a jejich vytváření
`ErrorPresenter`	Obecné chybové stránky a hlášky
`Error4xxPresenter`	Stránky s chybou třídy 4 (403, 404 atp.)

Tabulka 6.2: Presentery v aplikaci (zdroj: autor)

Kromě tříd popsaných výše existuje ještě jedna bázová třída `BasePresenter`, od které všechny ostatní presentery v aplikaci dědí. Třída obsahuje pouze jednu metodu a to konkrétně `beforeRender()`, ve které jsou definované vlastní Latte filtry, které jsou poté použity v šablonách, viz kapitola 6.4.3. Samotná bázová třída `BasePresenter` dědí od třídy `Nette\Application\UI\Presenter`, jak doporučuje Nette dokumentace [9]. Tato třída zajišťuje správné chování presenteru podle schématu popsaném v kapitole 6.3.2.

6.3.4 Formuláře

Formuláře tvoří základ interakce uživatele s webovou aplikací. Jedná se o sadu interaktivních prvků jazyka HTML, které uživatel vyplní, zvolí, zaškrtne atd. Po tzv. odeslání formuláře mohou být data zpracována na serveru nebo na straně klienta. Jelikož aplikace

předpokládá velmi jednoduchý frontend, jsou všechny formuláře odeslány na server, a to buď přímo se znovunačtením stránky, případně přesměrováním, nebo pomocí AJAXu.

Nette k formulářům přistupuje jako ke komponentám. Tento způsob má nespočet výhod, mezi které patří například znovupoužitelnost, zpřehlednění zdrojových kódů nebo oddělení definice vzhledu a funkčních požadavků.

Základní přístup je takový, že se formulář definuje v továrně (např. `StoreFormFactory`) a ta se zaregistruje v konfiguračním souboru jako služba. Poté si lze v kterémkoli presenteru pomocí DI vyžádat v konstruktoru instanci této továrny a nechat vytvořit libovolný počet formulářů daného typu. V továrně jsou definována jednotlivá pole formuláře, ale také zde může být implementován způsob validace formulářů a akce, která se má vykonat po úspěšném odeslání formuláře (např. přidání nebo modifikace záznamu v databázi).

Popsaný způsob je velmi jednoduchý a intuitivní, ovšem co se týče přizpůsobení vzhledu a rozložení jednotlivých formulářů, není úplně dostatečný. Proto byl využit komplikovanější způsob vytváření formulářů, který umožňuje lepší přizpůsobení jejich vzhledu.

Pro každý formulář bylo vytvořeno rozhraní, které má jen jednu metodu `create()` – toto požaduje framework Nette a přes ni se budou vytvářet formuláře. Rozhraní je vždy pojmenováno stylem `I<Název>FormFactory`, kde `Název` značí jméno formuláře (analysis, company, crop atd.). Toto rozhraní se společně s dalšími soubory týkající se konkrétního formuláře nachází ve složce `app/forms/<název>`.

Definice formuláře je ve třídě `<Název>Form`, která dědí od třídy `FormTemplate` a implementuje metody `validateForm()` pro validaci formuláře, `processForm()` pro zpracování formuláře a `createComponent<Název>()` pro definici polí formuláře.

Hlavní vylepšení ovšem spočívá ve třídě `FormTemplate`. V ní je implementována metoda `render()`, která zajistí, že se formulář vykreslí pomocí šablony `<Název>Form.latte`. V této šabloně lze definovat vzhled a rozložení prvků pro každý formulář zvlášť, stejně jako JavaScript potřebný pro ovládání některých prvků. Kromě této metody implementuje třída ještě další tři – `createForm()` pro vytvoření kostry formuláře (konkrétně přidání ochrany proti CSRF[1]) a metody `setActionAfterSuccess()` a `actionAfterSuccess()`, jimiž lze ovlivnit akci, která se vykoná po zpracování formuláře (typicky se v presenteru přiřadí zpráva informující uživatele o úspěchu nebo dojde k přesměrování).

Zpracování formuláře je velmi jednoduché, jelikož Nette předá do funkce pro validaci i pro zpracování asociativní pole se všemi vyplněnými políčky. Pokud tedy ve formuláři existuje například vstupní pole s názvem `name`, lze k jeho hodnotě přistoupit pomocí `$values["name"]`. Při zpracování formuláře se pracuje s entitami a managery pomocí jejich rozhraní, jak bylo popsáno v kapitole 6.2. Díky tomu jsou formuláře kompletně odstíněny od struktury databáze a práce s entitami je jednodušší. Například při vytváření nového záznamu stačí vytvořit novou entitu odpovídajícího typu, naplnit ji daty, která jsou automaticky zkontrolována a poté zavolat metodu manageru `add()`, která se postará o vložení entity do databáze včetně všech ostatních odkazovaných entit.

Ve formulářích byla použita dvě rozšíření, jež jsou volně dostupná přímo pro Nette. Jedním z nich je `FormReplicator`, který umožňuje vícekrát zkopírovat část formuláře (tzv. kontejner) a tyto kopie dynamicky přidávat a ubírat. Tohoto rozšíření bylo využito například ve formuláři pro transakce, kdy je možné nakoupit nebo prodat více položek od jednoho dodavatele ve stejný den, nebo u vytváření prací na polích pro zadávání příjmů a výdajů.

[1] Cross-Site Request Forgery je útok, při kterém útočník využije přihlášení oběti v systému a operuje v něm jejím jménem, více na: https://www.imperva.com/learn/application-security/csrf-cross-site-request-forgery/

Po stažení knihovny přes composer a zaregistrování rozšíření v konfiguračním souboru lze jednoduše při vytváření formuláře zavolat metodu `$form->addDynamic()`, která v parametrech potřebuje název skupiny kontejnerů, funkci pro jeho vytvoření a výchozí počet. Pro dynamické vytváření a rušení kontejnerů je nutné nadefinovat funkce, které jsou například zavolány po stisknutí tlačítka, implementující volání funkcí `$container->createOne()` nebo `$container->remove()`.

Při zpracování formuláře lze ke kontejeru přistoupit pomocí jeho jména přes asociativní pole hodnot formuláře, jak bylo popsáno dříve (např. kontejnery pod názvem `items` lze získat pomocí `$values["items"]`. Takto získáme pole, které je indexované pořadovým číslem kontejneru a obsahuje hodnoty položek uvnitř něj – např. `$values["items"][2]["name"]` obsahuje hodnotu vstupního pole `name`, která se nachází v kontejneru s indexem 2.

Dalším rozšířením pro formuláře je `DependentSelectBox`, které je použito ve formuláři pro vyhledávání využitých postřiků. Jedná se o výběr roků na poli, jehož nabídka je podmíněna vybraným polem. Toto rozšíření automaticky pomocí AJAXu změní nabídku roků při změně pole. Select box pro roky se vytvoří normálně pomocí `$form->addSelect()` a závislý select box z rozšíření pomocí `$form->addDependentSelectBox()` Na něm je poté ještě nutné zavolat metodu `setDependentCallback()`, kterou se přiřadí funkce pro načtení prvků.

6.4 Pohledy (view)

Nejvyšší vrstvou návrhového vzoru MVC jsou pohledy *(view)*. Je to vrstva nejblíže uživateli a tudíž se v ní nachází především definice vzhledu a jen málo logiky. V Nette je tato vrstva zastřešena šablonovacím systémem Latte, který pomáhá při zápisu částí PHP kódu do HTML stránky.

6.4.1 Šablony

Šablony jsou základním stavebním kamenem Latte. Zjednodušeně jsou to HTML soubory, které jsou obohaceny o speciální zápisy ze systému Latte, což lze poznat z jejich přípony – `.latte`.

Pokud není nakonfigurována jinak, existuje v Nette frameworku základní hierarchie šablon. Předpokládá se výchozí cesta `app/presenters/templates`. Pokud je zavolán presenter `Homepage` a jeho akce `default`, pokusí se Nette nalézt šablonu `Homepage/default.latte` nebo `Homepage.default.latte`. Společně s ní může také načíst rozvržení *(layout)*, do kterého by nahrál obsah šablony. Nejdříve je hledáno konkrétní rozvržení pro daný presenter (`Homepage/@layout.latte` nebo `Homepage.@layout.latte`), poté se použije z nadřazené složky společné pro všechny presentery (`@layout.latte`).

V aplikaci je použito jedno rozvržení, které obsahuje hlavičku HTML dokumentu včetně všech připojených CSS stylů a knihoven, skripty, definici rozvržení těla a odkazy menu. Tělo dokumentu se po načtení konkrétní stránky doplní pomocí makra `{include}`

Zajímavostí jsou tzv. *flash* zprávy, které se díky Nette přenáší mezi načtením stránek. Pomocí nich lze informovat uživatele o úspěchu či neúspěchu nějaké akce nebo mu poslat jiné upozornění, které vzniklo před přesměrováním. Do šablony je automaticky předáno pole těchto zpráv v proměnné `$flashes`. V něm je uchována informace o typu zprávy a její text. K jejich zobrazování byla použita JavaScriptová knihovna SmallPop. Po stažení skriptu a CSS souboru do webově přístupné složky stačí tyto soubory připojit přes HTML značku `<link>`, resp. `<script>`. Kdykoli je potřeba vyvolat vyskakovací okénko, stačí zavolat v Ja-

vaScriptu funkci `spop()`, jež má spoustu argumentů, mezi kterými lze nalézt text a typ zprávy, dobu automatického zavření, umístění, ikonu atp.

6.4.2 Makra

Síla systému Latte spočívá právě v makrech, což je zjednodušený zápis PHP kódu. Makro lze zapsat přímo do Latte souboru a při požadavku na konkrétní stránku se přeloží do odpovídajícího PHP, což se samozřejmě neděje při každém načtení stránky. Latte je optimalizováno a ukládá si výsledek překladu (předgenerované šablony) do složky dočasných souborů. Pouze pokud se šablona změní a administrátor vyprázdní složku dočasných souborů, Latte musí šablony při prvním načtení opět přegenerovat.

Makra se zapisují do složených závorek a zakládají si na intuitivním pojmenování. Lze zde proto nalézt klíčová slova jako `foreach`, `dump`, `if`, `include` atd[2]. Právě makro `include` je klíčovým prvkem pro šablonovací systém, jelikož dovoluje vložit stránku nebo část stránky do jiné. Například v těle `@layout.latte` je makro `{include content}`, kterým se do stránky vloží část stránky ze stejného nebo jiného souboru, jež bude definována pomocí párového makra `block`.

```
{block content}
  ...
{/block}
```

Většina maker má také svoji n-podobu. Toho je využito například na titulek stránky. V hlavičce HTML v souboru `@layout.latte` je definován titulek jako `{include title}` a na konkrétní stránce je první nadpis tvaru:

```
<h1 n:block="title">Titulek stránky</h1>
```

Makro `n:block` je zkrácený zápis pro makro `block` ukázané výše a za svůj obsah považuje obsah značky, ve které je zapsáno.

Ze spojení Latte a Nette lze těžit hlavně při generování odkazů. Jak už bylo popsáno v kapitole 6.3.1, existuje u Nette obousměrné směrování. V Latte lze proto zapisovat odkazy buď pomocí `{link Presenter:akce}`, nebo také zkráceně makrem `n:href`. Ukázka tohoto zápisu následuje ve zdrojovém kódu 6.3.

```
1 <nav>
2   <a n:href="Store:" n:class="$presenter->isLinkCurrent('Store:*') ? active">Sklad</a>
3   <a n:href="Field:" n:class="$presenter->isLinkCurrent('Field:*') ? active">Pole</a>
4   <a n:href="Transaction:sell" n:class="$presenter->isLinkCurrent('Transaction:sell')
        ? active">Prodeje</a>
5   ...
6 </nav>
```

Zdrojový kód 6.3: Odkazy v menu

V ukázce je také vidět další použité n-makro, konkrétně `n:class`. Toto makro vyhodnotí podmínku, kterou obsahuje, a pokud je splněna, bude mít odkaz nastavenou třídu `active`. Pokud podmínka splněna nebude, atribut `class` se ve výsledném HTML vůbec neobjeví. Co se týče podmínky, jedná se o dotaz na presenter. Metoda zjistí, zda se uživateli zobrazuje jakákoli (to značí hvězdička) akce konkrétního presenteru. Místo hvězdičky lze samozřejmě specifikovat i konkrétní akci, jak lze vidět na řádku 4.

[2]Přehled v dokumentaci Latte: https://latte.nette.org/cs/macros

Jedním z nejdůležitějších je makro pro výpis proměnné. Samotné PHP se snaží tento zápis co nejvíce zjednodušit, ovšem dospělo jen k tajemnému `<?= $var ?>`. Latte jej zkracuje na příjemné `{$var}`. Navíc oproti čistému PHP je obsah proměnné zbaven znaků se speciálním významem v daném kontextu a nahrazen sekvencí odpovídající tomuto znaku *(slangově escapování)*[3]. Pokud výchozímu chování chceme zamezit, lze použít například filtr `noescape`. Více o filtrech bude napsáno v následující kapitole 6.4.3.

Dalším zajímavým makrem je `snippet`. Používá se ve spojení s AJAXem a načítáním částí stránek. Elementy, které jsou uvnitř tohoto makra, je možné nechat samostatně překreslit invalidací dané části, přitom okolní obsah stránky zůstane nezměněn a stránka se nenačítá. Veškerá komunikace probíhá asynchroně a uživatel o ní vůbec nemusí vědět. Využití lze u výsledné aplikace nalézt například u formulářů ve spojení s doplňkem `FormReplicator`, který pro svoji činnost potřebuje odeslání formuláře, ovšem to může trvat i desítky milisekund, než se kontejner vykreslí. Místo toho je formulář odeslán pomocí AJAXu a při přidávání nebo odebírání kontejneru je překreslena pouze potřebná oblast. Podobně funguje analýza, kdy je celá analytická tabulka zabalena do jednoho „snippetu“ a načítá se se změnou dat nebo dimenzí.

6.4.3 Filtry

Mezi další přednosti Latte patří filtry. Jedná se o formátovací pravidla, která lze aplikovat při výpisu proměnných. Filtr je identifikován názvem a v samotném Latte je jich mnoho předdefinovaných[4]. Jeho název se píše za znak svislice po výpisu proměnné. Mezi svislítkem a názvem filtru nesmí být nikdy mezera. Příkladem může být výpis obsahu proměnné velkými písmeny – `{$var|upper}`.

Některé filtry potřebují další parametr, kterým je například formátovací řetězec, délka, oddělovač apod. Mezi taková makra patří `date`, jež formátuje datum. Řetězec je stejný jako u funkce `date()` v PHP. Dalším podobným filtrem je `number`, který může mít až 3 parametry – počet a oddělovač desetinných míst a oddělovač tisíců. Zápis a výsledek takových maker poté vypadá následovně:

```
{$var|date:"j. n. Y"}      =>    10. 4. 2019
{$var|number:2:",":" "}    =>    1 234,56
```

Toto jsou zápisy běžného českého datumu a částky, které se v aplikaci objevují několikrát. Je patrné, že je tento zápis příjemný a není problém si ho zapamatovat, ovšem opisovat jej na mnoha místech opakovaně je nebezpečné hned ze dvou důvodů. Není zaručeno, že se nikde neudělá chyba, a tudíž by mohla být aplikace nekonzistentní. Druhým důvodem je nemožnost hromadné změny, například pro různé jazykové mutace aplikace, po změně nějakých pravidel nebo na pokyn uživatele.

Latte však opět poskytuje řešení – uživatelské filtry. Aby nebylo nutné neustále vzpomínat, jak jsou zformátována čísla na předchozí stránce, nabízí se nadefinovat si formátování jedenkrát a poté používat nový filtr. Dokonce se nemusí pouze vytvářet „zkratky“ pro již existující filtry, ale lze napsat úplně nové. Této možnosti bylo využito i v informačním systému, kdy byly definovány ve třídě `BasePresenter` v metodě `beforeRender()` hned tři filtry.

Prvním z filtrů je `cs_date`, který formátuje objekt typu `DateTime` na české datum. Druhým filtrem je `cs_currency`, jenž zajistí správný formát české měny a navíc za číslo

[3] Více informací například na https://phpfashion.com/escapovani-definitivni-prirucka
[4] Přehled v dokumentaci Latte: https://latte.nette.org/cs/filters

přidá znak „Kč“. Posledním filtrem je `area`, který řeší formátování plochy. Za číslo ve správném formátu navíc přidá značku „ha“. Ukázky definic jsou ve zdrojového kódu 6.4. Ukázka byla zkrácená z důvodu lepší čitelnosti. V aplikaci se mezi znak měny či plochy a číslo vkládá nedělitelná mezera pomocí `html_entity_decode(" "))`. Stejně tak se nedělitelná mezera vkládá mezi oddělovač tisíců v číslech.

```
1  $this->template->addFilter("cs_date", function (\DateTime $content) {
2    return date("j. n. Y", $content->getTimestamp());
3  });
4
5  $this->template->addFilter('cs_currency', function ($content) {
6    return number_format($content, 2, ",", " ")." Kč";
7  });
```

Zdrojový kód 6.4: Vlastní Latte filtry

Jak lze z ukázek vidět, cokoli, co umí programátor zapsat pomocí jazyka PHP, může použít ve vlastním Latte filtru.

6.4.4 Formuláře

Jak již bylo popsáno v kapitole 6.3.4, formuláře jsou nejdůležitějším prvkem interakce mezi uživatelem a aplikací. Proto je důležité umět je správně a přehledně zobrazit. Pokud byl formulář v presenteru vytvořen, je možné jej nechat vykreslit v šabloně pomocí makra `control` (konkrétně `{control <název>}`). Pokud na vzhledu formuláře příliš nezáleží, lze ponechat výchozí nastavení vykreslování z frameworku Nette, které pro běžné účely postačí.

Pro vlastní definici rozložení prvků formuláře, je třeba použít další makra – `form` pro vypsání otevírací a uzavírací značky formuláře, `label` pro umístění popisku formulářového prvku a `input` pro vykreslení formulářového prvku, čehož bylo využito v šablonách jednotlivých formulářů, které se nachází v adresáři `app/forms/<název>`.

Pro dynamické skrývání a zobrazování prvků byl použit JavaScript, konkrétně knihovna jQuery, jež tuto práci zjednodušuje. Kromě toho pomáhá ve formulářích také s doplňováním nebo napovídáním hodnot, například při zadávání plodiny. Zde uživatel začne psát název plodiny (např. pšenice) a pod políčkem se zobrazí výběr z několika již dříve uložených plodin (např. pšenice jarní a pšenice ozimá). Když uživatel přistoupí k vyplňování odrůdy, nabízí se mu ty dříve uložené u konkrétní plodiny (například odrůdy ozimé pšenice: apache, julie, estevan). Načítání těchto hodnot probíhá pomocí AJAXu přes knihovnu `nette.ajax.js`, která rozšiřuje možnosti Nette právě v oblasti AJAX.

Ve formulářích je místo některých vybíracích políček *(select box)* použitá knihovna `select2`, která tyto vstupní pole vylepšuje a rozšiřuje o další možnosti. Mezi její hlavní přínosy patří:

- Vyhledávání v seznamu
- Dynamické načítání dat (AJAX)
- Přidávání položek za běhu
- Snadné přizpůsobení (jazykové i chování)

Nad běžným seznamem typu `select` lze jednoduše z JavaScriptu (po načtení knihovny) zavolat metodu `.select2()`, která má hodně parametrů přizpůsobující její chování a vzhled[5].

[5] Více o knihovně v dokumentaci https://select2.org/

Kapitola 7

Testování

Každý software by měl být před publikováním nebo uvedením do provozu řádně otestován. Testováním zjišťujeme informace o produktu, o jeho kvalitách a také (hlavně) o chybách v něm.

Dle [16] rozlišujeme pozitivní testy, které ověřují, zda program dělá to, co má, a negativní testy, které naopak spočívají v ověřování, jestli program nedělá to, co by dělat neměl.

V praxi se tedy jedná o zadávání správných vstupů s očekáváním správného chování a nebo zadáváním úmyslně špatných vstupů v očekávání zahlášení přívětivé chyby uživateli. Nikdy by ovšem neměla aplikace skončit v chybovém stavu, data by měla být vždy konzistentní a uživatel by měl mít přehled o tom, co a proč se děje.

7.1 Průběžné testování

Každý vývojář musí během vývoje aplikace průběžně testovat. Zde probíhalo testování manuálně zadáváním vzorových údajů do formulářů aplikace a zjišťovalo se, zda program dělá to, co by měl. K tomuto účelu bylo využito Tracy (dříve Laděnka)[1], které je součástí Nette frameworku. Zajišťuje přehledný výpis chybových hlášek PHP, zobrazuje vykonané databázové dotazy a přesměrování. Je možné si přes něj nechat vypsat obsah proměnné a dokáže napovědět, pokud se jedná o lehce opravitelnou chybu nebo překlep. Během tohoto testování bylo odhaleno množství chyb, ať už se jednalo o chyby při práci s databází, chyby v algoritmech, logické chyby, chybějící oprávnění při zápisu do souborů či překlepy.

Testování probíhalo převážně na lokálním stroji, na kterém byl spuštěn Apache server a na kterém také existovala MySQL databáze. V pozdějších fázích vývoje byla aplikace přesunuta také na server, na kterém bude po spuštění také běžet. Zde bude po odladění aplikace vypnuto zobrazování Tracy, aby uživatel neviděl ošklivé stránky popisující případné chyby, ale pouze programátorem nastavenou chybovou stránku. Tracy ovšem bude nadále zaznamenávat všechny chyby do složky `log`.

7.2 Konzultace

Kromě samotného testování aplikace probíhaly také konzultace se zemědělcem, jelikož vyvstalo několik otázek, které bylo třeba zodpovědět. Právě při těchto konzultacích vznikl například námět, že by se v analýze mělo počítat i se ziskem na jednotku plochy, ne pouze z pole jako celku.

[1] Dostupné na https://tracy.nette.org/cs/

Bylo také zjištěno, že u polí není dostatečné ukládat pouze jeho název, jelikož úřady z něj nepoznají, o které pole se jedná. Při hlášení postřiků je tedy nutné uvést číslo DPB (díl půdního bloku) podle systému LPIS. Tato změna byla zpětně zapracována do analýzy požadavků a okamžitě implementována do systému.

Dále zemědělec zmínil, že by mělo být možné nakoupit zboží i „zpětně po spotřebování“, protože často se nákup provede ráno před prací a než dodavatel vystaví fakturu, potřebuje zemědělec zapsat práci do systému. Je tedy umožněno spotřebovat položky ve skladu i do záporných hodnot a nákup, který je naskladní, zadat až později. O záporném stavu na skladě bude aplikace informovat na úvodní stránce, aby uživateli připomínala, že do aplikace nebyly vloženy všechny nákupy.

7.3 Testování zemědělcem

Při dokončení aplikace bylo přistoupeno k testům se samotným koncovým uživatelem výsledného systému. Zemědělec se pokoušel do aplikace zadávat vzorová data, která nashromáždil během posledních několika let a simuloval tím průběh života své farmy.

Před samotným testováním byl uživatel s výslednou aplikací seznámen a v průběhu testu se mohl na cokoli ptát. Pozorovalo se, zda opakovaně nedochází k nejasnostem při některém z úkonů. Pokud by se tak stalo, bylo by třeba zamyslet se a zjednodušit tuto činnost.

Všechna data byla ovšem nahrána bez problémů, tudíž nebylo co měnit. Databáze byla uložena jako vzor, se kterým bude aplikace distribuována – jedná se o dva SQL skripty, jeden vytvoří databázové tabulky a druhý do nich nahraje tato data.

Kapitola 8

Závěr

Cílem této bakalářské práce bylo vytvořit informační systém, který by byl schopen usnadnit zemědělci administrativu a plánování. Měl by také obstát v reálném provozu, což dle vyjádření zemědělce bylo splněno a aplikace jím bude od tohoto podzimu využívána.

Během vývoje bylo pracováno s několika různými technologiemi. Nejdůležitější z nich byl jazyk PHP pro vývoj logiky aplikace a nad ním postavený český webový rámec Nette ve verzi 2.4, která je v současné době nejaktuálnější. Bylo použito několik knihoven, které byly vytvořeny buď přímo pro Nette, nebo obecně například pro JavaScript.

Hlavní částí celé aplikace je analytická část systému, kde bylo třeba navrhnout strukturu databáze a implementovat uživatelsky přívětivé ovládání analytické tabulky včetně zanořování v dimenzi plodin. Tento cíl byl splněn, uživatel si může zobrazit výsledek hospodaření za jakékoli období, plodinu i pole, a to podle různých kritérií – příjmy, výdaje a celkem na pole a celkem na jeden hektar. Také lze aplikovat filtry na všechny dimenze, proto lze porovnávat např. několik polí mezi sebou.

Všechny požadavky vycházející z analýzy byly splněny, aplikace během testování koncovým uživatelem fungovala bez chyb a všechna vzorová data byla farmářem bez problému zadána.

Zemědělcem bylo navrženo několik rozšíření, která by měla být později v aplikaci implementována. Jedná se například o potřebu zasetí více plodin na jedno pole v případě, že by část špatně vzešla a bylo by třeba ji na jaře osít jinou plodinou.

Po vizuální stránce je aplikace dostačující, v budoucnu by ovšem uvítala drobná estetická vylepšení. Dále by také mohla být lépe přizpůsobena mobilním zařízením vzhledem k jejich stále stoupající oblíbenosti při práci s webem.

Plně funkční demonstrační aplikace je k dispozici veřejně na webové adrese `http://zemedelec.kuchynka.xyz/`. Zdrojové soubory lze nalézt na paměťovém médiu vloženém v tištěné verzi této bakalářské práce, jehož obsah je uveden v příloze A. Návod na instalaci aplikace na vlastní server včetně potřebných konfigurací se nachází v příloze B.

Literatura

[1] Arsenault, C. *The Pros and Cons of 8 Popular Databases* [online]. 2017 [cit. 16. ledna 2019]. Dostupné na: `https://www.keycdn.com/blog/popular-databases`.

[2] Barboyon, S. *PHP Benchmarks* [online]. 2017-2019 [cit. 16. ledna 2019]. Dostupné na: `http://www.phpbenchmarks.com/en/comparator/frameworks.html`.

[3] Bos, B. *20 Years of CSS* [online]. prosinec 2016 [cit. 4. prosince 2018]. Dostupné na: `https://www.w3.org/Style/CSS20/history.html`.

[4] Burget, R. *Informační systémy: Pojem informačního systému, data, informace.* Brno: FIT VUT, 2018. Přednáška.

[5] Chan, R. *The 10 most popular programming languages, according to the 'Facebook for programmers'* [online]. říjen 2018 [cit. 6. prosince 2018]. Dostupné na: `https://www.businessinsider.com/the-10-most-popular-programming-languages-according-to-github-2018-10`.

[6] Dostálová, Z. *Frontend vs. Backend* [online]. duben 2014 [cit. 4. prosince 2018]. Dostupné na: `https://www.czechitas.cz/cs/blog/zaciname-s-it/frontend-vs-backend`.

[7] Firth, A. *Which is the best CSS preprocessor?* [online]. září 2018 [cit. 4. prosince 2018]. Dostupné na: `https://www.creativebloq.com/features/best-css-preprocessor`.

[8] fnx.io. *Programovací jazyk Dart* [online]. 2018 [cit. 6. prosince 2018]. Dostupné na: `https://dartlang.cz/`.

[9] Grudl, D. *MVC aplikace & presentery | Nette Framework* [online]. 2008-2019 [cit. 5. dubna 2019]. Dostupné na: `https://doc.nette.org/cs/2.4/presenters`.

[10] Grudl, D. *Úvodní stránka blogu | Nette Framework* [online]. 2008-2019 [cit. 15. ledna 2019]. Dostupné na: `https://doc.nette.org/cs/2.4/quickstart/home-page`.

[11] Hanibal, J. *Struktura zemědělských podniků v rozlišení podle typů výrobního zaměření* [online]. duben 2018 [cit. 28. dubna 2019]. Dostupné na: `https://www.czso.cz/documents/10180/79535242/27016818k03cz.pdf/`.

[12] Herout, P. *Učebnice jazyka C.* 6. vyd. České Budějovice: KOPP, 2016. ISBN 978-80-7232-383-8.

[13] Janeček, M. *Laravel vs. Nette – tři měsíce poté* [online]. srpen 2016 [cit. 16. ledna 2019]. Dostupné na: https://blog.ikw.cz/laravel-vs-nette-t%C5%99i-m%C4%9Bs%C3%ADce-pot%C3%A9-78f1200497c9.

[14] Kanisová, H. a Müller, M. *UML srozumitelně*. 2. vyd. Brno: Computer Press, 2006. ISBN 80-251-1083-4.

[15] Keogh, J. *Java bez předchozích znalostí: průvodce pro samouky*. 1. vyd. Brno: CP Books, 2005. ISBN 80-251-0839-2.

[16] Kitner, R. *Co je testování softwaru?* [online]. 2015 [cit. 10. dubna 2019]. Dostupné na: https://kitner.cz/testovani_softwaru/co-je-testovani-softwaru/.

[17] Lacko, L. *Databáze: datové sklady, OLAP a dolování dat s příklady v Microsoft SQL Serveru a Oracle*. 1. vyd. Brno: Computer Press, 2003. ISBN 80-7226-969-0.

[18] Long, J. *I Don't Speak Your Language: Frontend vs. Backend* [online]. 2012 [cit. 4. prosince 2018]. Dostupné na: https://blog.teamtreehouse.com/i-dont-speak-your-language-frontend-vs-backend.

[19] MacDonald, M., Freeman, A. a Szpuszta, M. *ASP.NET 4 a C# 2010: tvorba dynamických stránek profesionálně*. 1. vyd. Brno: Zoner Press, 2011. ISBN 978-80-7413-131-8.

[20] Monus, A. *10 PHP Frameworks For Developers – Best of* [online]. duben 2018 [cit. 6. prosince 2018]. Dostupné na: https://www.hongkiat.com/blog/best-php-frameworks/.

[21] Netcraft. *December 2018 Web Server Survey* [online]. prosinec 2018 [cit. 17. ledna 2019]. Dostupné na: https://news.netcraft.com/archives/2018/12/17/december-2018-web-server-survey.html.

[22] Petrofčík, M. *Composer* [online]. červenec 2018 [cit. 17. ledna 2019]. Dostupné na: https://www.itnetwork.cz/php/ostatni/composer.

[23] Potter, J. *MySQL Performance: MyISAM vs InnoDB* [online]. srpen 2018 [cit. 3. dubna 2019]. Dostupné na: https://www.liquidweb.com/kb/mysql-performance-myisam-vs-innodb/.

[24] Procházka, D. *PHP 6: začínáme programovat*. 1. vyd. Praha: Grada Publishing, 2012. ISBN 978-80-247-3899-4.

[25] Písek, S. *HTML: začínáme programovat*. 3. vyd. Praha: Grada Publishing, 2010. ISBN 978-80-247-3117-9.

[26] Q-Success. *W3Techs - World Wide Web Technology Surveys* [online]. 2009-2019 [cit. 14. ledna 2019]. Dostupné na: https://w3techs.com/technologies/overview/programming_language/all.

[27] Shaleynikov, A. *Top 5 Most Popular CSS Frameworks that You Should Pay Attention to in 2017* [online]. říjen 2017 [cit. 4. prosince 2018]. Dostupné na: https://hackernoon.com/top-5-most-popular-css-frameworks-that-you-should-pay-attention-to-in-2017-344a8b67fba1.

[28] Shoptet. *Backend* [online]. 2008-2018 [cit. 4. prosince 2018]. Dostupné na: https://www.shoptet.cz/slovnik-pojmu/backend/.

[29] Sklar, D. *PHP 7: Praktický průvodce nejrozšířenějším skriptovacím jazykem pro web*. 1. vyd. Brno: Zoner Press, 2018. ISBN 978-80-7413-363-3.

[30] Skřivan, J. *Databáze a jazyk SQL* [online]. srpen 2000 [cit. 7. prosince 2018]. Dostupné na: https://www.interval.cz/clanky/databaze-a-jazyk-sql/.

[31] Smith, J. *9 Popular JavaScript Frameworks for 2018* [online]. červenec 2018 [cit. 4. prosince 2018]. Dostupné na: https://raygun.com/blog/popular-javascript-frameworks/.

[32] Šmíd, V. *Management informačního systému: Pojem informačního systému* [online]. 2018 [cit. 4. prosince 2018]. Dostupné na: https://www.fi.muni.cz/ smid/.

[33] solid IT gmbh. *Knowledge Base of Relational and NoSQL Database Management Systems* [online]. 2018 [cit. 7. prosince 2018]. Dostupné na: https://db-engines.com/.

[34] SteelKiwi. *Top 10 Python Web Frameworks to Learn in 2018* [online]. březen 2018 [cit. 7. prosince 2018]. Dostupné na: https://hackernoon.com/top-10-python-web-frameworks-to-learn-in-2018-b2ebab969d1a.

[35] Stejskal, J. *Vytváříme WWW stránky pomocí HTML, CSS a JavaScriptu*. 1. vyd. Brno: Computer Press, 2004. ISBN 80-251-0167-3.

[36] The Apache Software Foundation. *The Apache HTTP Server Project* [online]. 1997-2019 [cit. 17. ledna 2019]. Dostupné na: https://httpd.apache.org/.

[37] Valkovič, P. *Git – Historie a principy* [online]. srpen 2014 [cit. 17. ledna 2019]. Dostupné na: https://www.itnetwork.cz/software/git/git-tutorial-historie-a-principy.

[38] Vančurová, P. *Účtování rostlinných kultur v zemědělském podniku* [online]. duben 2014 [cit. 17. ledna 2019]. Dostupné na: https://www.dauc.cz/dokument/?modul=li&cislo=47237.

[39] W3C. *World Wide Web Consortium* [online]. [cit. 6. prosince 2018]. Dostupné na: https://www.w3.org/.

[40] W3Schools. *Introduction to Python* [online]. 1999-2018 [cit. 6. prosince 2018]. Dostupné na: https://www.w3schools.com/python/python_intro.asp.

[41] Wrembel, R. a Koncilia, C. *Data Warehouses and OLAP: Concepts, Architectures and Solutions*. Hershey: IRM Press, 2007. ISBN 1-59904-364-5.

[42] Žára, O. *JavaScript: Programátorské techniky a webové technologie*. 1. vyd. Brno: Computer Press, 2015. ISBN 978-80-251-4573-9.

[43] Zendulka, J. a Rudolfová, I. *Databázové systémy – IDS*. Brno: FIT VUT, 2005-2006. Studijní opora.

Příloha A

Obsah přiloženého média

- `/`
  - `aplikace/`
    - `app/` — zdrojové soubory aplikace
      - `config/` — konfigurační soubory
      - `forms/` — formuláře
      - `model/` — modely
      - `presenters/` — presentery a šablony
        - `templates/`
      - `router/` — směrovač
      - `bootstrap.php` — zaváděcí soubor aplikace
    - `log/` — logování chyb
    - `temp/` — dočasné soubory aplikace
    - `vendor/` — zdrojové soubory frameworku a knihoven
    - `www/` — webově přístupné zdroje
      - `css/` — CSS soubory
      - `images/` — obrázky
      - `js/` — JavaScript soubory
      - `index.php` — vstupní soubor aplikace
    - `composer.json` — závislosti pro Composer
    - `.htaccess`
    - `SQL-struktura.sql` — inicializační SQL skript
    - `SQL-data.sql` — vzorová data
  - `zprava/`
    - `bib-styles/` — bibliografické styly
    - `obrazky/` — obrázky a diagramy
    - `template-fig/` — loga VUT
    - `Makefile` — překlad zprávy
    - `xkuchy00-IS-drobneho-zemedelce-01-kapitoly.tex` — kapitoly
    - `xkuchy00-IS-drobneho-zemedelce-20-literatura.bib` — bibliograf. databáze
    - `xkuchy00-IS-drobneho-zemedelce-30-přílohy.tex` — přílohy
    - `xkuchy00-IS-drobneho-zemedelce.pdf` — zpráva v PDF
    - `xkuchy00-IS-drobneho-zemedelce-tisk.pdf` — zpráva v PDF – verze pro tisk
    - `xkuchy00-IS-drobneho-zemedelce.tex` — hlavní .tex soubor
    - `zadani.pdf` — zadání práce

Příloha B

Instalace

V této příloze bude popsán postup instalace aplikace na webový server. Nebude rozlišeno, zda se jedná o lokální server, nebo vzdálený, jelikož se konfigurace v ničem zásadně neliší. Pouze je třeba si dát pozor při výběru hostovaného serveru na podporu PHP ve vyšších verzích (minimálně 5.6), na možnost založení MySQL databáze, na dostatek místa pro aplikaci a případně na možnost využít nástroj Composer.

Umístění na server

Aplikaci lze na server nainstalovat dvěma způsoby – bez nástroje Composer nebo s ním. Výhodou jeho použití je přesun menšího objemu dat mezi paměťovým médiem a serverem, ovšem k zajištění jeho funkce je nezbytné připojení k internetu, odkud si stáhne potřebné soubory.

Obě metody spočívají v překopírování obsahu složky `aplikace/` z paměťového média na server. Pokud je použit Composer, lze vynechat složku `apliklace/vendor/`. Umístění na serveru není podstatné, ovšem existuje zde několik zásad, které je třeba splnit:

- Složka `www/` **musí** být přístupná z webu, ideálně by se do ní měl uživatel dostat přímo bez zadání jakékoli cesty
- Ostatní složky **nesmí** být veřejně přístupné (to zajišťují `.htaccess` soubory v nich)
- Složky `temp/` a `log/` musí být z aplikace zapisovatelné (zde záleží, pod jakým uživatelem či skupinou poběží proces, běžně stačí přidělit práva 775 při nastavené skupině `www-data`)

Databáze

Dále je nezbytné založit MySQL databázi a poté v ní vytvořit tabulky, případně nahrát vzorová data. Toto lze provést pomocí přiložených SQL skriptů `SQL-struktura.sql` a `SQL-data.sql` ve složce `app/`.

Informace o připojení k databázi je třeba aplikaci zadat do konfiguračního souboru `app/config/config.local.neon` do řetězce dsn. Po stažení zdrojů z paměťového média bude v konfiguračním souboru nastaveno přesně to, co lze vidět na ukázce B.1. Dále je nutné k databázi zřídit účet, který bude mít oprávnění číst a zapisovat do všech tabulek. Jeho přihlašovací jméno a heslo je třeba také vyplnit do tohoto souboru.

```
database:
      dsn: 'mysql:host=127.0.0.1;dbname=zemedelec'
      user: 'zemedelec'
      password: 'pmswduacjAzFUDsS'
      options:
            lazy: yes
```

Zdrojový kód B.1: Konfigurace připojení k databázi

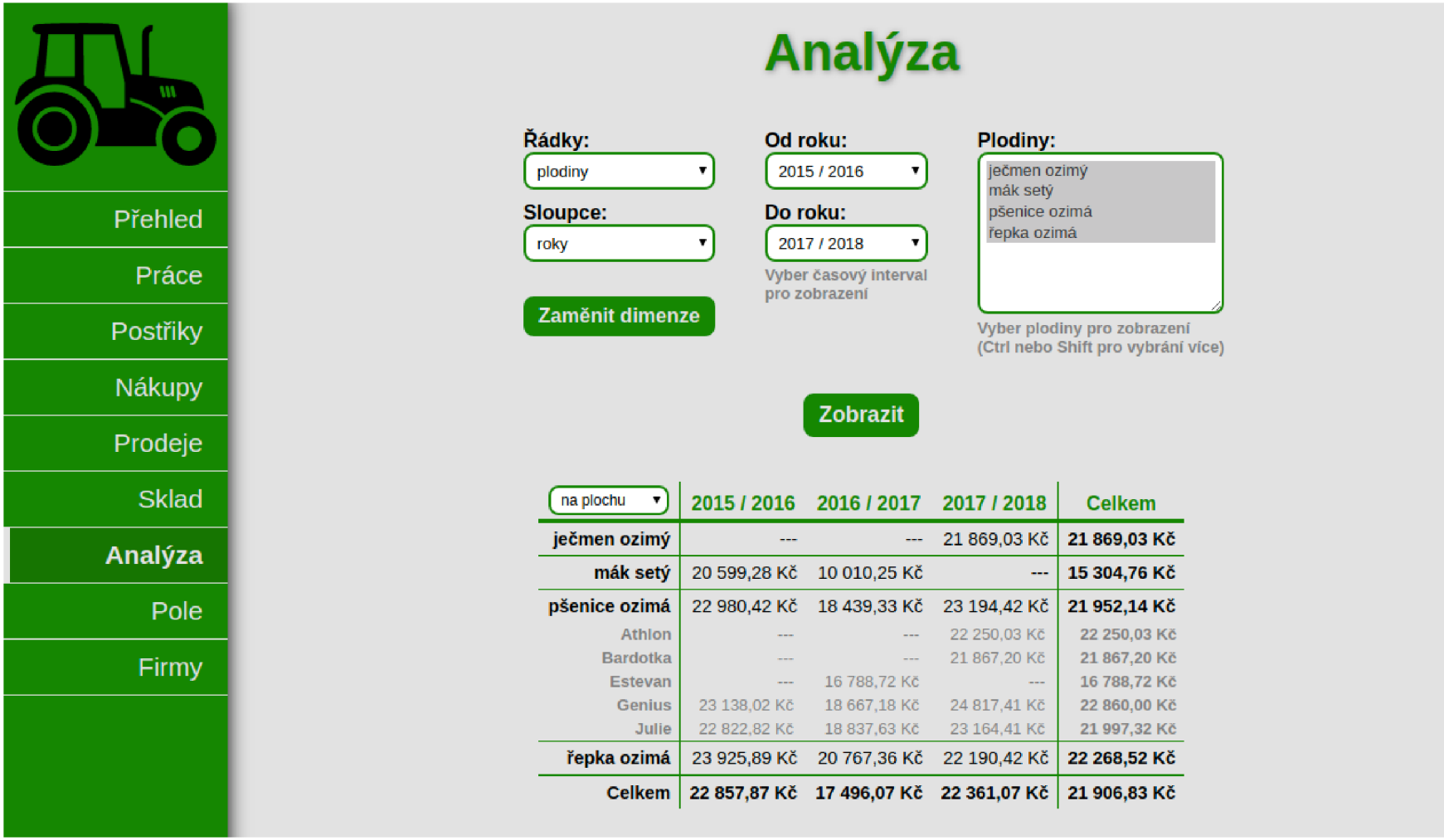
Přesměrování

Pokud je třeba upravit cestu ke vstupnímu souboru aplikace (což je `www/index.php`), lze toho docílit pomocí konfiguračního souboru `.htaccess`. Ten je umístěn v kořenové složce aplikace a předpokládá, že do ní se uživatel vždy dostane a bude nutné jeho požadavky přesměrovat do složky `www/`. Pokud je struktura serveru jiná, je nutné tento soubor upravit – na místa, kde se vyskytuje `/www/` je třeba vypsat skutečnou cestu k adresáři, ve kterém se nachází `index.php`. Předpokladem je ovšem běh aplikace na serveru Apache s možností použití `mod_rewrite` a zapnutým `RewriteEngine`. Většina webových hostingů toto podporuje již v základní konfiguraci a s lokálním Apache serverem není problém si tento doplněk doinstalovat, pokud ho základní balíček neobsahuje.

Příloha C

Ukázky ze systému

Obrázek C.1: Zadávání nové práce – postřik proti bejlomorce (zdroj: autor)



Obrázek C.2: Analýza dat v systému (zdroj: autor)

Hlášení použitých prostředků na ochranu rostlin

Období: 2015/2016
LPIS DPB: 0201/1 (620-1180)
Výměra celkem: 26,24 ha
Plodina: řepka ozimá (odrůda Arabella)

Jméno a příjmení: Marek Kuchynka
IČ: 12345679
Adresa: Božetěchova 2, 612 00 Brno

Datum	Název přípravku	Ošetřená plocha	Dávka	Množství	Účel aplikace	Poznámka
09. 09. 2015	Autor	26,24 ha	1,15 l/ha	30,00 l	jednoděložný a dvouděložný plevel	Dávka 300l/ha, 100% účinnost
09. 10. 2015	Nurelle D	18,90 ha	0,60 l/ha	10,80 l	dřepčíci, krytonosci	Dávka 280l/ha, účinný, aplikováno lokálně
01. 04. 2016	Nurelle D	26,24 ha	0,60 l/ha	16,00 l	krytonosci	Dávka 200l/ha, TM, účinný
01. 04. 2016	Lynx	26,24 ha	1,00 l/ha	26,00 l	černání stonků	Dávka 200l/ha, TM, účinný
16. 05. 2016	Biscaya 240 OD	26,24 ha	0,30 l/ha	8,00 l	bejlomorka	Dávka 250l/ha
25. 06. 2016	Roundup Flex	26,24 ha	2,00 l/ha	52,50 l	desikace	2 - 3 týdny před sklizní

Vygenerováno 06.05.2019 *Strana 1 z 1* *© Informační systém drobného zemědělce*

Obrázek C.3: Vygenerované hlášení postřiků (zdroj: autor)