

Informační systém pro podporu drobné rostlinné výroby

Diplomová práce

Vedoucí práce:

doc. Ing. František Dařena, Ph.D.

Bc. Lukáš Dohnálek

Brno 2016

Na tomto místě bych rád poděkoval vedoucímu mé práce panu doc. Ing. Františku Dařenovi, Ph.D. za ochotu a odborné rady v průběhu zpracování a také rodině za její podporu během mého studia.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Informační systém pro podporu drobné rostlinné výroby** vypracoval/a samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací*.

Jsem si vědom/a, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 28. Prosince 2015

Abstract

This thesis focuses on design and implementation of an ERP system, dedicated to support small farmers and their enterprises in crops production in the Czech republic. ERP system is divided into three bases. First of them aims on working with land blocks, the other one focuses on stock operations and the third one focuses on economical operations of small farmers. Based on the processed data, ERP system provides control of costs and yields of single land blocks. Furthermore, the ERP is dedicated for creation of crop rotation plans and a compliance of which it helps to keep track of. The system also provides desired data for management of farming control. The ERP system was being developed in cooperation with an entrepreneur Karel Procházka, further referred to as an authority, who was testing the final application. The ERP system was developed as an web application.

Keywords

Information system, crop production, ASP .Net, AngularJs, Repository pattern, Dapper, Html 5.

Abstrakt

Tato diplomová práce je zaměřena na návrh a implementaci informačního systému, určeného pro podporu činnosti drobných zemědělců věnujících se rostlinné výrobě v ČR. Informační systém je rozdělen do tří základních částí. První se zaměřuje na práci s půdními bloky, další část se věnuje skladové činnosti a třetí ekonomické činnosti drobných zemědělců. Na základě zpracovaných dat informační systém poskytuje kontrolu nákladovosti a výnosnosti jednotlivých půdních bloků. Dále je IS určen pro tvorbu osevních plánů, jejichž dodržování pomáhá kontrolovat. Systém také poskytuje potřebná data pro nařízené zemědělské kontroly. Informační systém byl vyvíjen za spolupráce s OSVČ Karlem Procházkou dále zadavatelem, který testoval výslednou aplikaci. Tento systém je vytvořený jako webová aplikace.

Klíčová slova

Informační systém, Rostlinná výroba, ASP .Net, AngularJS, Repository pattern, Dapper, Html 5.

Obsah

1	Úvod a cíl práce	10
1.1	Úvod.....	10
1.2	Cíl práce.....	10
2	Informační systémy	12
2.1	Informace, data, znalost.....	12
2.2	Přístupy k vývoji informačních systémů.....	12
3	Rostlinná výroba v ČR	14
3.1	Osevní postupy.....	14
3.2	Příprava půdy.....	15
3.3	Přípravky na ochranu rostlin.....	16
3.4	Nitrátová směrnice.....	17
3.5	Osevní plán pro hospodářský rok.....	17
3.6	Současný stav IS pro drobné zemědělce v ČR.....	18
4	Metodika práce	20
4.1	UML.....	20
4.1.1	Use case diagram.....	21
4.2	Vícevrstvá architektura software.....	21
4.3	Repository pattern a Dapper.....	21
4.3.1	Extenzní metody.....	22
4.4	Technologie použité pro implementaci GUI.....	22
4.4.1	Html.....	23
4.4.2	Css.....	23
4.4.3	JavaScript.....	24
4.4.4	AngularJS.....	25
5	Návrh IS pro drobné zemědělce	27
5.1	Specifikace požadavků informačního systému.....	27
5.1.1	Funkční požadavky systému.....	27

5.1.2	Nefunkční požadavky	28
5.2	Návrh IS.....	29
5.2.1	Návrh funkcionality informačního systému	29
5.2.1.1	Popis základního use case informačního systému	30
5.2.1.2	Popis use case Vytvoření osevního plánu.....	34
5.3	Návrh datové vrstvy	36
5.3.1	Konceptuální model datové vrstvy	36
5.3.1.1	Popis vlastností entit konceptuálního modelu	37
6	Realizace IS	40
6.1	Databáze aplikace.....	40
6.2	Datová vrstva	41
6.2.1	Doménový model	42
6.2.2	Manipulace s daty.....	42
6.2.2.1	Použití návrhového vzoru RepositoryPattern	43
6.3	Aplikační vrstva	45
6.4	Prezentační vrstva	47
6.4.1	Struktura prezentační vrstvy.....	47
6.4.1.1	Owin	47
6.4.1.2	Owin middleware.....	47
6.4.1.3	WebApi controllers.....	48
6.4.1.4	Autentizace	52
6.4.1.5	Spuštění aplikace.....	52
6.4.2	Implementace uživatelského rozhraní	52
7	Diskuze	64
8	Závěr	65
9	Literatura	66
10	Přílohy	68
A.	Diagram aktivit Registrace klienta	68
B.	Diagram aktivit přihlášení do systému.....	69
C.	Diagram přidání nového stroje do IS.....	70

D.	Diagram vytvoření kategorie plodin.....	71
E.	Diagram aktivit vytvoření půdního bloku	72
F.	Diagram aktivit vložení nové položky skladu	73
G.	ERD databáze	74
H.	Ukázka implementace modelů entit	75
I.	Ukázka implementace šablony modálního dialogu	76
J.	Ukázka rozhraní pro tvorbu osevního plánu.....	77
K.	Ukázka rozhraní pro kontrolu osevního plánu před uložením	78
L.	Ukázka rozhraní detailu osevního plánu	79
M.	Ukázka rozhraní pro editaci dodavatelů	80
N.	Ukázka rozhraní skladu	81
O.	Ukázka rozhraní pro správu faktur	82

Seznam obrázků

Obrázek 1	Použití návrhového vzoru Repository pattern.....	22
Obrázek 2	Diagram Two-way data binding.....	26
Obrázek 3	Use case diagram akcí uživatele.....	30
Obrázek 4	Diagram aktivit přidání prováděné práce.....	32
Obrázek 5	Use case vytvoření osevního plánu.....	35
Obrázek 6	Crow's foot notace.....	36
Obrázek 7	Zjednodušený konceptuální model datové vrstvy.....	37
Obrázek 8	Struktura datové vrstvy aplikace.....	41
Obrázek 9	Souborová struktura části prezentační vrstvy.....	53
Obrázek 10	Ukázka formuláže pro registraci.....	55
Obrázek 11	Hlavní obrazovka aplikace.....	58
Obrázek 12	Příklad dialogového okna.....	59
Obrázek 13	Tvorba osevního plánu.....	61
Obrázek 14	Editace položky osevního plánu.....	62

1 Úvod a cíl práce

1.1 Úvod

Využívání informačních systémů přináší pro majitele firem určitou konkurenční výhodu. IS jsou navrhovány tak, aby svým klientům usnadňovaly práci, šetřily finanční zdroje a poskytovaly užitečné informace pro strategické rozhodování. Velké množství firem dnes pro usnadnění provozu využívá informační systémy. Podnikatelé mají k dispozici velké množství informačních systémů, které již na trhu jsou. Nebo mají možnost nechat si firemní informační systém vyvinout, pokud se jim to vyplatí. Na trhu jsou velké společnosti, nabízející různě modulované informační systémy jako jsou SAP, Helios, Abra, atd. Nasazení takového informačního systému pro střední a velké firmy je určitě přínosem. Jak bylo zmíněno, informační systémy lze využít v různých oborech podnikání. V posledních letech nacházejí informační technologie stále větší uplatnění i v oboru zemědělství. Pro větší podniky trh nabízí určité množství informačních systémů, nebo různé druhy modulů do stávajících ekonomických systémů zaměřených na usnadnění práce v zemědělství. Tyto systémy nabízí velké množství funkcí a jejich nasazení do podniku se také stává přínosem.

V České republice je poměrně velké množství podnikatelů věnujících se rostlinné výrobě. Dle strukturálního šetření v zemědělství zpracovaného českým statistickým úřadem v roce 2013 zde působí přes dvacet tisíc zemědělských subjektů. Jedná se převážně o drobné zemědělce, kteří se této činnosti věnují již celou řadu let nebo mladé začínající podnikatele. Drobný zemědělec musí o svých půdních blocích shromažďovat velké množství informací, které musí dále poskytovat zemědělským kontrolám. Jak bylo zmíněno, dnes je k dispozici řada programů a informačních systémů určených pro zemědělské podniky, nicméně tyto systémy jsou zaměřeny pro nasazení ve středních a velkých podnicích. Nasazení takových informačních systémů je velice nákladné a jejich efektivní použití poměrně náročné. Podnikatelé věnující se drobné rostlinné výrobě obhospodařují průměrně 20-40 ha půdy a informace potřebné pro kontroly zpravidla uchovávají papírovou formou, protože je pro ně nasazení informačního systému cenově nedostupné nebo nedokáží informační systém naplno využít. To jsou hlavní důvody, proč se drobní zemědělci brání pořízení informačního systému pro svoji vlastní potřebu.

1.2 Cíl práce

Diplomová práce je zaměřena na návrh a implementaci informačního systému, který bude určený pro podporu činnosti drobných zemědělců. Tento informační systém bude vytvořený za spolupráce s OSVČ Karlem Procházkou, kde bude výsledný systém testován a potencionálně využíván.

Cílem této práce je na základě analýzy činnosti drobného zemědělce vypracovat návrh informačního systému, který bude odpovídat potřebám drobných zemědělců. Ten bude následně implementován a otestován v reálném provozu.

Základní činností systému bude poskytování potřebných informací týkajících se půdních bloků. Systém bude nabízet správu půdních bloků, které zemědělec může rozdělit na různé půdní hony. Součástí správy půdních bloků bude sestavování osevního plánu pro daný půdní blok. Na základě osevního plánu bude zemědělec informován o celkových nákladech na dokončení osevního plánu. Následně, po zaznamenání dat o sklizni, také o výnosu jednotlivých půdních bloků. Dále bude systém nabízet evidenci skladu. IS tak bude při tvorbě osevního plánu zemědělce informovat o stavu skladovaného materiálu a případné nutnosti přikoupení materiálu pro splnění osevního plánu. Dále bude IS poskytovat informace o technice, která je využívána drobným zemědělcem. Bude tak mít pohodlnou možnost plánovat plnění osevních plánů a mít přehledné informace o nákladech na provoz této techniky. Nedílnou součástí systému bude správa ekonomické činnosti drobného zemědělce. Ta bude zaměřena na tvorbu a uchovávání faktur, které musí drobný zemědělec evidovat. Na základě informací o osevních plánech tak bude informační systém poskytovat data pro kontroly prováděné státní rostlinolékařskou správou.

Práce nejprve seznámí čtenáře s problematikou drobné rostlinné výroby. Čtenář zde nalezne informace, které jsou potřebné pro vypracování návrhu informačního systému zaměřeného na drobnou rostlinnou výrobu. Dále práce obsahuje popis technologií, které jsou potřebné pro vývoj informačního systému jako webové aplikace. Následně se práce věnuje návrhu a implementaci konkrétní aplikace pro drobné zemědělce. Cílem je vytvořit uživatelsky přívětivý informační systém vyhovující požadavkům zadavatele. V závěru této práce bude zhodnoceno otestování a přínos tohoto informačního systému pro OSVČ Karla Procházky.

2 Informační systémy

Práce je zaměřena na vývoj informačního systému, tato kapitola se věnuje obecnému popisu informačních systémů. Informační technologie se neustále modernizují a stávají se součástí běžného života každého člověka. Proto se informační systémy dostávají do velkého okruhu firem. Množství informací a dat, které musí firmy zpracovávat neustále roste. Informační systémy pomáhají efektivně zpracovat a uchovat velké množství dat. Firmy využívají informační systémy pro podporu strategického řízení (Vymětal, 2009).

Informační systém je specifickým typem obecného systému. Existuje celá řada definic systémů. Například systém je množina komponent (prvků), která interaguje, aby splnila nějaký cíl. Nebo systém je pravidelně se ovlivňující nebo vzájemně závislá skupina položek, která je chápána jako celek. Z těchto definic vyplývá, že systém se skládá z více prvků, mezi kterými probíhá interakce. Na základě vstupních dat informační systém s daty pracuje a provádí jejich transformaci a zpracovaná data poskytuje jako výstup takového systému.(Bruckner, 2012)

2.1 Informace, data, znalost

Informační systémy jsou zaměřeny na zpracování a transformaci dat. Vedení každé firmy neustále činí důležitá rozhodnutí, které ovlivňují budoucnost firmy. Pro své rozhodování používají podporu právě informačních systémů. Ty pomáhají vedení zpracovávat velké množství informací. Z tohoto důvodu je nutné obecně definovat co informace je. Informace vznikají, když se určitým datům přiřadí určitý význam. Data obecně vypovídají o reálném světě. Jsou získávány z reálného světa pomocí měření nebo pozorování. Na základě kombinace informací, předpokladů a zkušeností vznikají znalosti, které jsou využity vedením pro určitá rozhodnutí.

2.2 Přístupy k vývoji informačních systémů

Každý informační systém prochází určitým životním cyklem. Tento cyklus začíná fází plánování a předběžnou analýzou. Následuje specifikace požadavků na konkrétní systém a dále návrh. Dalším krokem je implementace informačního systému. Po implementaci následuje část testování a nasazení do zkušebního provozu. Tím tento cyklus nekončí, ale pokračuje udržováním informačního systému v provozu. Základem návrhu informačního systému je důkladná analýza požadavků na vyvíjený systém. Před samotným začátkem implementace informačního systému je nutné tento systém navrhnout a důkladně konzultovat se zadavatelem. Během návrhu informačního systému, jsou sestavovány různé modely, které odpovídají určitým pohledům na vyvíjený systém. Požadavky zadavatele se zpravidla v průběhu vývoje informačního systému mění a je nutné na tyto změny adekvátně reagovat. Agilní metody vývoje softwaru jsou postaveny

právě na schopnosti rychle reagovat na změny požadavků. Princip je zaměřen na rychle vyvinutých částech systému, které jsou postupně konzultovány se zadavatelem a je tak možné v krátkém čase reagovat na případné změny v jednotlivých požadavcích. Je kladen důraz na častou zpětnou vazbu od zadavatele, který nejen že reaguje na vyvinutý software, ale má neustálý přehled o stavu projektu (Buchalceková, 2005).

3 Rostlinná výroba v ČR

Základním odvětvím zemědělské výroby je právě rostlinná výroba. Dle českého statistického úřadu česká republika využívá k zemědělské činnosti 54% rozlohy půdy. Tato plocha tvoří zemědělský půdní fond cca 4,3 mil. ha z něhož je 38% půda orná. 3,7 mil. ha zemědělské půdy vlastní fyzické osoby, ale 70% půdy obhospodařují podnikatelé, družstva a další společnosti. Jen cca 30% zemědělské půdy, patřící fyzickým osobám obhospodařují drobní zemědělci. Velikost drobných zemědělců se liší na základě rozlohy obhospodařované půdy. Dle českého statistického úřadu je v ČR 48 119 (údaj k 31. 12. 2013) zemědělských podniků. Zemědělců obhospodařujících 0 – 4,99 ha je 24 405, dále pak 5 446 podniků obhospodařujících 5 – 9,99 ha zemědělské půdy (Český statistický úřad, 2015).

Rostlinná výroba se dělí do tří skupin potravinových kultur a to na obiloviny, okopaniny a luštěniny. V České republice největší část zastupuje skupina obilovin. Jedná se o nejdůležitější část rostlinné výroby. Produkty rostlinné výroby slouží k výživě lidí, hospodářských zvířat a jako průmyslové suroviny.

3.1 Osevní postupy

Cílem osevního postupu je nezvyšovat náklady na výrobu, ale optimálním využitím přírodních podmínek při snižování negativních vlivů zemědělské činnosti dosáhnout zvyšování produkce. Jedná se o postupné střídání plodin na jednotlivých pozemcích podle nároků plodin a záměrů produkce. Základem rotace plodin je norfolkský osevní postup, při němž se střídá jetelovina, ozim, okopanina organicky hnojená a jařina. Jedním z úkolů osevního postupu je zajistit optimální využití půdního fondu obhospodařovaného zemědělcem. Důvody pro střídání plodin na půdě vyplývají z celé řady komplexně působících činitelů. Jde zejména o vztah plodin k vodě, živinám, vlivu na půdní strukturu, způsob zakořeňování plodin, vztah plodin k plevelům, chorobám a škůdcům, obohacování půdy hmotou zbytků rostlin, reakce plodin na organické hnojení, délku meziporostního období a projevy únavy půdy.

Při plánování střídání plodin je možné dodržovat například tyto postupy střídání. Klasickým postupem střídání plodin je jednoduché střídání plodin vycházející z výše uvedeného norfolkského osevního postupu. Dvojitě střídání plodin podle Kónneckeého využívá stejného zastoupení plodin, avšak využívá vždy dvě listnaté a dvě stébelnaté plodiny. Jedná se o málo používaný způsob střídání plodin z důvodu velkého procenta pěstovaných obilnin, ale hodí se pro kukuřičnou výrobu. Střídání plodin v rámci tzv. skupinových sledů se využívá častěji, jedná se o střídání širokolisté plodiny a dvou úzkolistých. Nicméně na základě pokusů prováděných v Žabčicích se jako nejlepší jeví norfolkský systém. Osevní postupy se dále z hlediska dodržování střídání plodin dělí na pevné, uvolněné a volné osevní postupy. Pevné osevní postupy přesně dodržují navrženou strukturu střídání plodin. Uvolněné neboli pružné osevní postupy dodržují hlavní zásady střídání plodin, nicméně se v některých letech odchylují, popřípadě mění sled článků

osevního postupu. Dodržují však kostru osevního postupu, zařazování plodin reprodukcí půdní úrodnosti. Volné osevní postupy nedodržují strukturu střídání plodin. V závislosti na potřebě, nebo poptávce se mění zařazování plodin, posuzuje se pouze, jak předplodina vyhovuje následující plodině (Kostelanský, 2004).

3.2 Příprava půdy

Příprava půdy patří mezi jednu z hlavních činností zemědělce. Zemědělci užívají určité půdní bloky, jsou to pozemky uvedené ve veřejném registru půdy LPIS. Tyto půdní bloky potom mohou jednotliví zemědělci dále dělit na tzv. půdní hony, které dále využívají pro tvorbu osevních plánů. Velká část nákladů na jednotlivé půdní hony je spojena právě s přípravou půdy. Hodnota nákladů se odvíjí od různých aspektů, jako je druh prováděné práce, svažitost terénu, velikost obhospodařovaného půdního bloku atd.

Zpracování půdy se dělí do třech základních částí. Jedná se o základní zpracování půdy, přípravu půdy pro setí a sázení plodin a zpracování půdy během vegetace. První část má za úkol připravit fyzikální, chemické a biologické vlastnosti půdy tak, aby byly zajištěny dobré podmínky pro růst a vývoj plodin. Pro základní zpracování půdy se využívá například podmítka tj. mělké zpracování půdy. Podmítka má všestranný význam, ale jako nejdůležitější se považuje zlepšení hospodaření s půdní vodou a boj proti plevelům. Její další význam spočívá v odplevelení půdy, regulaci škůdců a původců chorob, podpory biologické činnosti půdy a zlepšení pro další zpracování půdy. Dalším krokem základního zpracování půdy je orba. Hlavním úkolem orby je vytvořit v ornici kyprou, drobtovitou vrstvu s příznivými hydrofyzikálními a biologickými poměry. Orbou se do půdy zapravují různé složky, jako jsou posklizňové zbytky, organická a minerální hnojiva, nebo plevelé vzešlé na podmítce. Část pro přípravu půdy pro setí se nazývá předseťová příprava. Jedná se o obdělávací zásahy do malé hloubky půdy, jejichž cílem je připravit půdu pro nové osivo. Předseťová příprava tak zajišťuje, že nové osivo je vyseto v požadované hloubce, má tak dostatek živin a vrchní zkyplená půda poskytuje dostatek vzduchu pro růst a usnadňuje pronikání rostlin na povrch. Pro předseťovou přípravu se využívá smykávání, vláčení, kypření a válení půdy. Do poslední části tradičního zpracování půdy patří zpracování v průběhu vegetace. Jedná se o takové zásahy, které nepoškodí kořenovou část rostlin. Jsou určeny k umožnění přívodu vzduchu a výměně plynů v rhizosféře. Rhizosférou se rozumí oblast povrchu a okolí kořenů rostlin. Dále se tyto úkony využívají k obnově strukturního stavu v povrchu ornice, ničení škraloupu po deštích, mechanické omezování zaplevelenosti porostů, zlepšení vsakování srážek a naopak omezení evaporace z půdy. Pro zpracování půdy se také můžou využívat tzv. zjednodušený systém zpracování půdy. Existuje velké množství způsobů zjednodušeného zpracování půdy, ty jsou založené na vylučování některých operací, spojování více zákroků do menšího počtu operací, mělké nebo speciální zpracování půdy nebo setí plodin do nezpracované půdy. Systémy setí se používají v různých variantách

v závislosti na půdních a klimatických podmínkách způsobu hospodaření na půdě a také na základě vybavení mechanickými prostředky. Například lze využít sloučení přípravy půdy a setí, sloučení orby, přípravy půdy a setí nebo snižování hloubky orby (Kostelanský, 2004).

3.3 Přípravky na ochranu rostlin

Nedílnou součástí hospodářského roku je činnost zemědělce spojená s péčí o vyseté plodiny. Ty mohou být napadeny různými druhy škůdců, nemocí, nebo ohrožovány nežádoucím plevelem. V průběhu let se vypracovalo mnoho způsobů a chemických látek, které usnadňovali činnost spojenou s ochranou rostlin. Velká část přípravků na ochranu rostlin je toxická. Proto osoby pracující s těmito přípravky musí mít školení odborné způsobilosti pro používání přípravků na ochranu rostlin. Toto nařízení vešlo v platnost v novele zákona od roku 2013 č.199/ 2012 Sb. (původní znění §86 zákona č. 326 / 2004 Sb. o rostlinolékařské péči). Odborná způsobilost pro zacházení s přípravky na ochranu rostlin se dělí do tří stupňů. První stupeň musí absolvovat všichni, kteří manipulují s přípravky na ochranu rostlin. Pro řízení činnosti spojené s ochranou rostlin a dohledem nad osobami s prvním stupněm způsobilosti je nutný druhý stupeň odborné způsobilosti. Třetí stupeň je potom určen pro distributory a poradce (Kostelanský, 2004).

Zemědělci mají k dispozici soubor metod ochrany rostlin. Na základě monitoringu škodlivých organismů, určení škodlivého organismu a výběru vhodného souboru metod ochrany mohou zvolit pěstitelské a šlechtitelské metody, které se často kombinují s chemickými metodami. Dále také biologické metody ochrany, ty využívají viry, bakterie, houby a například parazitoidy. V praxi se biologické metody využívají převážně proti živočišným škůdcům. Preventivně se mohou využívat fyzikální a mechanické metody ochrany rostlin, jako například čištění osiva. V § 23 zákona č. 78/2004 Sb., o nakládání s geneticky modifikovanými organismy a genetickými produkty, v prováděcí vyhlášce 2 č. 209/2004 Sb. k tomuto zákonu a v Pravidlech koexistence v rostlinné produkci Ministerstva zemědělství ČR jsou uvedeny povinnosti spojené s biotechnologickými metodami. Ty využívají geneticky modifikované organismy. Využívají se také chemické metody, které mají rychlé působení. Nevýhody však spočívají v důsledcích těchto metod na životní prostředí. Přípravky na ochranu rostlin lze také rozdělit podle jejich praktického využití. Fungicidy se využívají proti původcům houbových chorob. Herbicidy používané proti plevelům, mohou být buď selektivní, nebo neselektivní. Selektivní působí na určitý druh plevelu a neselektivní působí na veškerou rostlinnou vegetaci. Pro hubení hmyzu se například využívají insekticidy. Do kategorie přípravků na ochranu plodin také patří regulátory růstu, které lze využít například pro regulaci plodnosti, urychlení dozrávání plodů atd.

Každý, kdo využívá přípravky na ochranu rostlin je dle § 49, odst. 3 zákona č. 326/2004 Sb. povinen vést evidenci použití přípravků na ochranu rostlin.

Evidence musí obsahovat číslo půdního bloku, kde byl přípravek použit, název plodiny, výměru oseté plochy, výměru ošetřené plochy, datum použití přípravku, úplný název přípravku, způsob aplikace, celkové množství, dávku a účel použití přípravku. Vedení této evidence je mimo jiné nutnou podmínkou některých státní finančních podpor (Česká společnost rostlinolékařská, 2010).

3.4 Nitrátová směrnice

Hnojiva jsou nedílnou součástí zemědělství. Jsou rozdělena do dvou základních skupin, organická a minerální hnojiva. Využívají se například pro zvyšování půdní úrodnosti, jsou jimi dodávány živiny do půdy a jsou zdrojem mikroorganismů, stimulačních a růstových látek. Hnojení však způsobuje přidávání dusičnanů do půdy a tím i do podzemní vody. Velká koncentrace dusičnanů v půdě způsobuje v blízkosti zdrojů pitné vody znečištění. Dusičnany jsou v pitné vodě závadné obzvláště pro kojence, ale i pro dospělé. Z důvodu snižování dusičnanů v půdě vstoupil v platnost evropský předpis tzv. nitrátová směrnice. Vodní zákon č. 254/2001 Sb., uplatňuje tento předpis. Nitrátová směrnice určuje zranitelné oblasti. Zranitelné oblasti jsou ty, kde dochází k znečištění vod dusičnany ze zemědělských zdrojů. Nitrátová směrnice uplatňuje akční program, který udává správný způsob hospodaření ve zranitelných oblastech. Pokyny nitrátové směrnice jsou pro všechny zemědělce doporučující, avšak jejich dodržování ve zranitelných oblastech je povinné. Jedním z požadavků akčního programu je například zákaz hnojení v zimním období.

Stejně tak jako při využívání prostředků na ochranu rostlin jsou zemědělci dle zákona č. 156/1998 Sb., o hnojivech (§ 9, odst.5), povinni vést evidenci hnojiv. Tato vyhláška poskytuje vzor pro vedení potřebné evidence. Evidence hnojiv slouží jako podklad pro nařízené kontroly, ale také jako zdroj informací pro samotné zemědělce. Na základě evidence můžou napláňovat použití vhodného hnojiva a kontrolovat své hospodaření s živinami v půdě. Informace, které musí evidence obsahovat, jsou uvedeny ve vyhlášce č. 377/2013 Sb., o skladování a používání hnojiv (§ 9). Jsou to například číslo půdního bloku, typ hnojení, množství, plodina atd (Klír, Kozlovská, 2013).

3.5 Osevní plán pro hospodářský rok

Na rozdíl od osevního postupu, který se zaměřuje na dlouhodobé plánování střídání plodin, jak bylo popsáno výše, se osevní plán zabývá jednotlivými činnostmi v jednom hospodářském roce. Hospodářský rok začíná přípravou půdy a končí sklizní jednotlivých plodin. Cílem osevního plánu je napláňovat všechny operace, které jsou nutné provést na jednotlivých půdních honech. Na základě dobře zpracovaného osevního plánu tak zemědělec může zjistit svoje náklady potřebné na splnění osevního plánu na všech svých půdních honech, pro které osevní plán zpracuje. Do osevního plánu spadá obdělávací činnost přípravy půdy, setí, aplikace přípravků na ochranu rostlin, hnojiv a následná sklizeň. Na základě

dávkování jednotlivých osiv, hnojiv a přípravků na ochranu rostlin zemědělec určuje potřebné množství na splnění osevního plánu. Díky informacím o hodnotách látek v půdních blocích se určí jaké množství živin a látek je nutné dodat zpět do půdy formou hnojiv. Stejně tak poskytuje osevní plán informace o potřebném množství přípravků na ochranu rostlin. Díky údajům o vybraném dávkování a naplánování jednotlivých činností osevní plán poskytuje komplexní informace o nákladech a vytiženosti jednotlivých strojů. Součástí osevního plánu jsou i informace o sklizni, které slouží pro vyhodnocení výnosu půdních honů (Kostelanský, 2004).

3.6 Současný stav IS pro drobné zemědělce v ČR

Na českém trhu je k dispozici několik řešení pro podporu zemědělské výroby. Tyto systémy nabízí uživatelům velké množství funkcí. Z velké části slouží tyto aplikace pro vedení legislativně povinné evidence. Hlavní rozdíl mezi jednotlivými systémy je v množině nabízených funkcí, složitosti uživatelského rozhraní a nákladech na provoz daného informačního systému. Velká část těchto systémů nabízí velké množství propracovaných evidencí, které může uživatel využívat. Například Soft bit nabízí rozšíření účetního informačního systému SQL Ekonom o moduly zaměřené na živočišnou a rostlinnou výrobu. Klient tak musí vlastnit licenci pro účetní systém a přikoupit požadované moduly. Stejným způsobem je možné rozšířit informační systém poskytovaný firmou Helios. Tyto systémy se zaměřují jak na rostlinnou výrobu tak živočišnou a slouží pro evidenci velkého množství informací. Takto rozsáhlé evidence vyžadují propracované uživatelské rozhraní s velkým počtem ovládacích prvků, tím se pro uživatele stává náročné tyto aplikace efektivně využívat. Pro větší podniky jsou na trhu k dispozici řešení, které nabízí různou formu podpory, jako jsou školení, linka technické podpory atd. tyto systémy jsou zpravidla velice rozsáhlé a určené pro potřeby těchto velkých firem. Možnostem a službám, které tyto systémy nabízejí, také odpovídají náklady na pořízení a provoz takového systému.

Pro evidenci své činnosti mají drobní zemědělci také možnost využít nekomerční aplikace, které jsou dostupné například na portálu farmáře. Ty nabízejí možnosti základní evidence potřebných údajů. Aplikace nabízí celou řadu funkcí a tomu odpovídá poměrně složitě uživatelské rozhraní. Pro práci s tímto druhem aplikací je nutné, aby uživatel prostudoval velké množství materiálu pro efektivní využití aplikace. Ve většině případů je evidence zaměřena na uchování informací o použitých materiálech a nenabízí efektivní vyhodnocení nákladů. Velký počet drobných zemědělců zaměřených na rostlinnou výrobu proto pro evidenci povinných údajů ve velké míře využívá tabulkových editorů. A to zejména z důvodu snadného a rychlého použití. Systémy, které jsou pro podporu zemědělství k dispozici, vyžadují větší míru investovaného času a nákladů pro efektivní využití nabízených funkcí. Typ uživatelů, jako jsou zejména drobní zemědělci, zpravidla nejsou ochotni investovat tolik času, aby ve větší míře využili možnosti dostupného programu. V případě sofistikovanějších systémů je pro

drobné zemědělce hlavním ukazatelem pro výběr softwaru velikost nákladů spojených s nasazením a provozem. Z tohoto důvodu jsou nabízené systémy využívány hlavně většími podniky.

4 Metodika práce

Pro úspěšné splnění cílů této práce je nejdříve nutné zvolit vhodnou metodu návrhu informačního systému. Tento IS bude navrhován na základě konzultací se zadavatelem, který bude výsledný systém testovat. Zvolená metoda návrhu tak musí být přijatelná jak pro zadavatele, tak pro vývojáře. Zvolené metody pro návrh IS a technologie pro implementaci jsou v této kapitole níže popsány.

Na základě konzultací se zadavatelem musejí být sestaveny jednotlivé požadavky na informační systém, na základě kterých bude dále proveden samotný návrh informačního systému. Následně bude proveden návrh funkcionality informačního systému pomocí UML, kterému se věnuje kapitola 4.1.

Po konzultaci a schválení návrhu funkcionality IS bude přistoupeno k návrhu databáze, která bude základem tohoto IS. V této fázi návrhu IS je třeba provádět důkladné konzultace, aby navržený model databáze odpovídal potřebám vyvíjeného informačního systému. Pro modelování databáze bude použit entitně relační diagram, který bude obsahovat všechny objekty a jejich vlastnosti.

Tento informační systém bude vyvíjen na základě vytvořeného návrhu, který odpovídá potřebám drobných zemědělců. IS bude vyvíjen jako webová aplikace pomocí technologie ASP .Net. Při implementaci této aplikace bude vycházeno z vícevrstvé architektury software, jejíž princip je popsán v kapitole 4.2. Datová vrstva bude implementována dle návrhového vzoru repository pattern, který určuje jakým způsobem aplikace přistupuje k datům uloženým v relační databázi. Pro manipulaci s daty bude využito objektově relační mapování pomocí micro-ORM Dapper. Klientská aplikace bude implementována pomocí javascriptového frameworku AngularJS. Jedná se o cliet-site Framework, který je blíže popsán v kapitole 4.4.4. Výsledný informační systém bude předán zadavateli k testování implementovaných funkcí a na základě jeho připomínek bude následně upravován.

4.1 UML

Pro návrh funkcionality informačního systému byl zvolen jazyk UML, který disponuje dostatečným počtem srozumitelných diagramů, pomocí kterých lze sestavit model informačního systému. Jazyk Unified Modeling Language se využívá pro objektově orientovanou analýzu a návrh nejen informačních systémů, ale je využíván pro vývoj celé řady projektů. Jedná se o standard v oblasti návrhu systémů. Jeho udržování a rozvoj má za úkol skupina Object Management Group. Počátek tohoto jazyka spadá do poloviny 90.tých let, bylo nutné sjednotit principy návrhu systémů.

Modelování systému je užitečnou součástí při komunikaci mezi vývojářem a klientem. Pomocí různých modelů je vývojář schopen navrhnout co má systém dělat. Modely informačního systému můžeme vyjádřit pomocí diagramů. Diagramy jsou grafické pohledy na části systému. Celý informační systém obvykle bývá navržen pomocí více diagramů. Jazyk UML nabízí použití třinácti různých diagramů pro modelování různých pohledů na systém. Diagramy, které jsou v UML

k dispozici, lze rozdělit na dvě základní skupiny. Na diagramy chování, ty vyjadřují chování systému a diagramy struktury, které vyjadřují elementy nezávislé na čase (Zelinka, 2008).

4.1.1 Use case diagram

Diagramy případu užití se využívají pro popis chování systému. Jsou nedílnou součástí návrhu informačního systému a velice usnadňují proces návrhu během komunikace se zadavatelem. Diagramy případu užití se sestavují pomocí use case, aktorů a vazeb mezi nimi. Element případu užití se znázorňuje pomocí elipsy a vyjadřuje konkrétní funkcionalitu systému. Okolí systému se modeluje pomocí elementu aktor, který má symbol osoby. Pomocí tohoto elementu je tak možno vyjádřit komunikaci systému se svým okolím. Interakce aktorů a use case se znázorňují pomocí jednotlivých vazeb (Zelinka, 2008).

4.2 Vícevrstvá architektura software

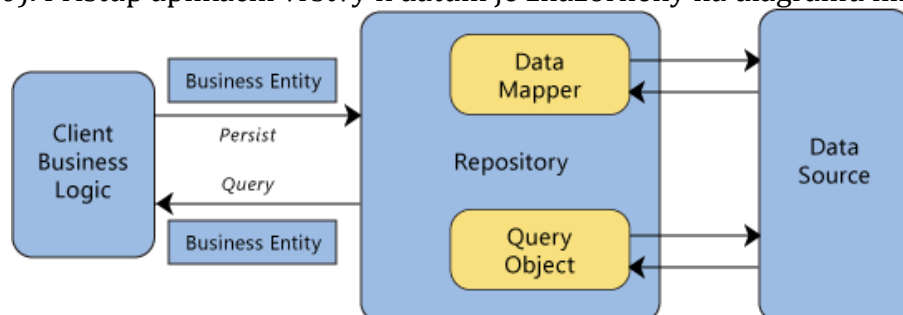
Existuje celá řada architektonických vzorů, které lze použít při vývoji aplikací. Ty určují základní strukturu softwarového systému. Jednotlivé vzory mají své výhody a nevýhody. Pro vývoj software můžeme zvážit například microservices patter nebo monolith pattern. Microservices pattern je náročnější na nasazování jednotlivých servis a může nastat problém se synchronizací. Oproti tomu monolith pattern poskytuje relativně vyšší rzchlost, ale není tak škálovatelný a testovatelný jako vícevrstvá architektura. (Microservice, 2014)

Pro webovou aplikaci tedy použijeme vícevrstvou architekturu pro její jednoduchost, možnost škálovatelnosti a testovatelnosti. Na základě této architektury je aplikace rozdělena do samostatných vrstev. Každá z vrstev má jasně danou úlohu a s ostatními vrstvami komunikuje pomocí implementovaného rozhraní. V našem případě bude aplikace sestavena ze čtyř vrstev. Tímto způsobem bude implementován navržený informační systém. Jedná se o prezentační vrstvu, aplikační, datovou a relační databázi. Jednotlivé vrstvy je tak možné snadno testovat a díky rozdělení rolí vrstev je v případě zachování rozhraní možné provádět změny v aplikační logice, popřípadě databázi aniž by to omezilo činnost uživatele. Každé vrstvě je v části Realizace věnována samostatná kapitola.

4.3 Repository pattern a Dapper

Pro manipulaci s daty bude použit návrhový vzor repository pattern. Hlavní myšlenkou návrhového vzoru repository pattern je zpřehlednit psaní a používání kódu, který manipuluje s daty. Jeho hlavní výhodou je vyšší testovatelnost oproti srovnatelným řešením, přehlednost, jelikož jsou všechny databázové dotazy na „jednom místě“ a možnost použití Dependency Injection. Další se zvažovaných řešení bylo použití CQRS nebo Metadata Mapping. Datová vrstva spolu s repository pattern využívá micro-ORM systém Dapper, který slouží pro mapování záznamů databáze na příslušné modely aplikace a zpět. Při dodržení tohoto návrhového

vzoru tak lze výsledný kód, který s daty manipuluje snadno testovat (Mackey, 2010). Přístup aplikační vrstvy k datům je znázorněný na diagramu níže.



Obrázek 1: Použití návrhového vzoru Repository pattern

Zdroj: (The Repository Pattern. Microsoft API and reference catalog [online]. [cit. 2015-05-13].)

V datové vrstvě je implementováno několik tříd, které slouží jako modely pro entity relační databáze. Záznamy jsou v relační databázi uloženy jako řádky tabulek představujících entity, zatím co v datové vrstvě aplikace jsou záznamy instance objektů. O automatickou konverzi mezi daty v relační databázi a daty v objektově orientovaném jazyku se stará programovací technika zvaná object-relational mapping. Jako ORM v této aplikaci byl použit produkt platformy Microsoft .Net Dapper. Jedná se o micro ORM, dapper nenabízí tak velkou funkcionalitu jako například EntityFramework, ten umí například generovat SQL dotazy na základě příkazů. To může vést k chybám během implementace a v případě složitých SQL dotazů může EntityFramework vygenerovat mnohořádkové dotazy, jejichž zpracování zabere čas. Dapper podporu generování SQL dotazů nemá, ale je zaměřen na jednoduchost a rychlost. Slouží tedy pouze k mapování objektů na jednotlivé třídy (Miller, 2015).

4.3.1 Extenzní metody

Extenzní metody jsou speciálním druhem statických metod. Umožňují vybranou třídu rozšířit o dané metody, bez nutnosti implementovat speciální třídu, která by požadovanou třídu dědila. Rozšiřující metody, jsou implementovány ve vlastních statických třídách, které rozšiřují základní třídu BaseRepository. Metody, které vykonávají jiné než CRUD operace se díky použitím extenzních metod volají stejně, jako by byly implementovány přímo v třídě, kterou rozšiřují. Příklad implementace extenzních metod je uveden v následující kapitole (Rahman, 2013).

4.4 Technologie použité pro implementaci GUI

Tento systém, bude implementován jako webová aplikace. Uživatelské rozhraní je implementováno za použití javascriptového frameworku AngularJS. Jedná se o framework zaměřený na vývoj single page aplikací (SPA). Jak z názvu vypovídá je postaven na jazyku javascript. To znamená, že jeho kód se neprovádí na straně serveru, ale na straně klienta. Oproti klasické MVC aplikaci má SPA výhodu, že se

webová stránka nemusí při každém požadavku znovu načítat a pouze se dynamicky mění její obsah tudíž je stránka uživatelsky přívětivější a práce s ní plynulejší. Nevýhodou je např. nutnost synchronizace dat na pozadí nebo správa webové historie. Pro SPA existuje ještě několik dalších frameworků jako Ember, Meteor nebo Backbone. Angular byl zvolen pro mojí osobní znalost toho frameworku a spokojenost s jeho použitím na jiných projektech.

4.4.1 Html

Webové stránky jsou tvořeny pomocí hypertextového značkovacího jazyka Html. Jedná se o hypertextový značkovací jazyk HyperText Markup Language, využívaný pro tvorbu dokumentů, obsahující hypertextové odkazy. Počátek tohoto jazyka se datuje do roku 1992, kdy ho vytvořil Tim Berners-Lee.

Html poskytuje celou řadu elementů, které ovlivňují vzhled a chování webové stránky. Pomocí tohoto jazyka je formátován text takového dokumentu. Html 5 již dovoluje programátorovy využívat celou řadu elementů, tak aby vytvořil stránku podle jeho představ. Webové aplikace využívají tento jazyk pro vytvoření uživatelského rozhraní. Html dokument je uložen na serveru a klient jeho obsah zobrazuje pomocí internetového prohlížeče.

Pro vytvoření „single page“ webové aplikace se využívá jednoho souboru zvaného layout, ten obsahuje definici html dokumentu hlavičku, tělo a patičku stránky, který je společný pro celou aplikaci. Obsah stránky se poté dynamicky vkládá do těla layoutu (Schafer, 2009).

Nová verze jazyka přináší nové možnosti html, ale také rozšiřuje vlastnosti stávajících tagů. Novým elementem je například element `hgroup` ten lze využít pro seskupování nadpisů. Dalším z rozšířených elementů je tag `input`, ten dostal velké množství nových atributů, které rozšiřují možnosti jeho použití. Více informací je dostupných v literatuře (Smith, 2012).

4.4.2 Css

Struktura webových aplikací je tvořena pomocí jazyka html, pro definování vzhledu jednotlivých elementů se využívá CSS. Zkratka css znamená Cascading Style Sheets, tedy kaskádové styly. Tento jazyk byl vytvořen standardizační organizací W3C a jeho autorem je Håkon Wium Lie. Dříve strukturu a vzhled elementů webové stránky určoval samotný jazyk html. Cílem bylo oddělit vzhled stránky od samotného obsahu. V dnešní době je k dispozici již třetí verze jazyka CSS.

Kaskádové styly využívají systému jedinečných identifikátorů a tříd, pomocí kterých se css vlastnosti aplikují na konkrétní elementy stránky. Díky tomu je tak možné definované styly aplikovat na různé webové stránky a velice usnadňují změnu stylu zobrazení elementů prohlížečem. Každý element html stránky podporuje atribut *id* a *class*. Jak již bylo zmíněno, tyto atributy mimo jiné slouží pro svázání CSS s konkrétním elementem na stránce. Níže uvedený příklad ukazuje

definici vzhledu jednoduchého tlačítka, které bude patřit do třídy `buttonExample`. Při psaní CSS můžeme využít buď odkazování se přímo na názvy jednotlivých tagů s možností využití jejich stromového řazení, nebo se pomocí křížku odkazovat na id elementu a pomocí tečky na název třídy.

```
.buttonExample {
  border-radius: 28px;
  box-shadow: 0px 1px 3px grey;
  font-family: Courier New;
  color: white;
  font-size: 20px;
  background: darkblue;
  padding: 10px 20px 10px 20px;
}
.buttonExample:hover {
  background: lightblue;
}
```

Výše uvedená ukázka kódu představuje nastavení vzhledu tlačítka. V css je nejdříve uveden selektor, který definuje, k jakému prvku stránky se definovaný vzhled bude vztahovat. V tomto případě se jedná o elementy s třídou *buttonExample*. Následně jsou ve složených závorkách nastaveny hodnoty požadovaných vlastností ve formátu „*atribut*“ : „*hodnota*“; . Tlačítko s třídou *buttonExample* bude mít zaoblené rohy díky atributu `border-radius`, dále je definovaný vržený stín, font, barva textu, velikost textu, pozadí tlačítka a vnitřní okraje elementu. Css podporuje tzv. pseudotřídy, díky kterým můžeme definovat změnu vzhledu v závislosti na akcích uživatele. Příklad uvádí změnu pozadí tlačítka, v případě kdy na tlačítko najede uživatel myší. Další pseudotřídy jsou například *:link* nebo *:active*. Pseudotřída *link* je automaticky přidána k odkazům, které ještě nebyly navštíveny a *active* se přidá elementu, na který uživatel klikl (Procházka, 2011).

4.4.3 JavaScript

Pomocí výše zmiňovaných jazyků je možné napsat velké množství webových stránek. Nicméně tyto stránky budou pouze statické. Pro vytvoření interaktivní webové stránky i webové aplikace je jednou z možností využít právě jazyka javascript. Tento jazyk umožňuje programátorovi vytvořit interaktivní ovládací prvky GUI.

Jedná se o jazyk, který vytvořil Brendan Eich z firmy Netscape a byl v roce 1997 standardizován asociací ECMA. Nejdříve byl uveden pod názvem Mocha, následně pod názvem LiveScript a nakonec z marketingových důvodů se použil název JavaScript i když s jazykem Java má společnou pouze podobnost v syntaxi. Jak bylo zmíněno, jazyk je určen pro psaní interaktivních částí webové stránky. Kód tohoto jazyka je ze serveru stažen a spuštěn až na straně klienta.

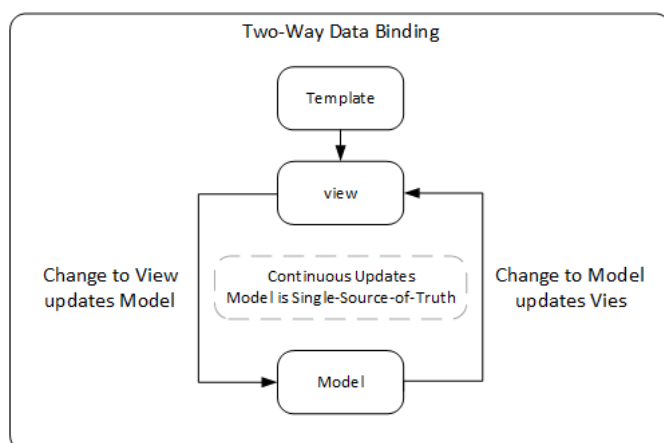
4.4.4 AngularJS

Moderní webové aplikace využívají propracované uživatelské rozhraní. Vytvoření takového rozhraní vyžaduje velké množství javascriptového kódu. Právě nutnost psaní velkého množství kódu vedla k vytvoření různých javascriptových frameworků. Jedním z nich je právě angularJs, který byl využit pro vytvoření tohoto IS. Tento Framework vytvořila společnost Google a dále pokračuje v jeho vývoji.

Jedná se o MVC javascriptový framework, který je určený pro vývoj uživatelského rozhraní webových aplikací. Použitá architektura model-view controller odděluje jednotlivé vrstvy klientské aplikace. Aplikace je tak rozdělena do logických celků a její implementace se stává přehlednější. Aplikace obsahuje základní layout, do kterého jsou dle potřeby dynamicky vkládány šablony, které slouží jako view vrstva frameworku. AngularJS doplňuje jazyk html o speciální příkazy tzv. direktivy. Existuje celá řada direktiv a vybrané budou níže popsány. Jednotlivé direktivy se zapisují jako atributy vybraných html elementů a rozšiřují tak jejich funkcionalitu. AngularJS zavádí dvě základní direktivy `ng-app` a `ng-controller`. Direktiva `ng-app="Nazev_aplikace"` se uvádí jako atribut počátečního tagu párového html elementu, který obsahuje vyvíjenou aplikaci. V tomto případě se jedná o základní tag html stránky, který uvozuje celý html dokument. Kód aplikace, který je implementován uvnitř takto deklarovaného elementu tak zpracovává přímo Framework a jsou zde k dispozici další direktivy.

Vrstva zvaná model, je ve frameworku reprezentována speciálními objekty označovanými `$rootScope` a `$scope`. Oba objekty slouží jako vnitřní paměť a využívají se k manipulaci s daty. Objekt `rootScope` je základní model, který prostupuje celou aplikací a zavádí ho právě direktiva `ng-app`. Druhá zmíněná direktiva `ng-controller` slouží k definování menších bloků aplikace a umožňuje tak implementovaný kód rozdělovat do přehledných bloků. Tato direktiva se používá obdobně jako direktiva `ng-app` vepsáním o párového tagu zpravidla elementu `div`. O zpracování části aplikace, která je implementována uvnitř takového tagu se stará příslušný controller, který má k dispozici vlastní model scope. Ten slouží pro výměnu dat mezi vrstvou view a controller (Williamson, 2015).

Aplikace vytvořená pomocí tohoto frameworku je tak přehledně rozdělena na jednotlivé bloky. AngularJS podporuje dependency injection. Jedná se o techniku, která zajišťuje propojení jednotlivých modulů aplikace. Šablony jednotlivých view mají k dispozici direktivu `ng-model="proměnná"`. V uvozovkách je uveden název proměnné, jejíž hodnota je uložena ve stejnojmenné proměnné objektu scope.



Obrázek 2: Diagram Two-way data binding

Zdroj: AngularJS: Data Binding. 2015, vlastní úprava

AngularJS pro synchronizaci dat využívá two-way databinding. Jak je patrné z výše uvedeného diagramu, tento princip zajišťuje automatickou synchronizaci v případě, že je hodnota změněna uživatelem na straně view, nebo ji změní příslušný controller. V modelu, který je reprezentovaný objektem scope, je tak vždy k dispozici aktuální hodnota uložené proměnné (AngularJS, 2015).

5 Návrh IS pro drobné zemědělce

5.1 Specifikace požadavků informačního systému

Informační systém je vyvíjen na základě požadavků zadavatele. Cílem této práce je vytvořit pohodlnou aplikaci, která bude splňovat očekávání zadavatele a usnadňovat jeho zemědělskou činnost. Tento systém musí rychle a přehledně poskytovat informace, které jsou potřebné v zemědělské činnosti. Hlavním účelem systému je zadavateli poskytovat informace o nákladech, výnosnosti a ziscích jednotlivých půdních bloků během jejich užívání. S touto činností následně souvisejí další potřebné vlastnosti systému, jako je tvorba osevních plánů, vedení skladu a ukládání informací o fakturách.

5.1.1 Funkční požadavky systému

- Registrace pro přístup do aplikace - Přístup do informačního systému je podmíněn registrací, při které zadavatel vyplní potřebné informace. Ty bude systém dále využívat při doplňování informací při evidenci faktur a exportu potřebných dat.
- Správa půdních bloků – IS bude poskytovat snadnou možnost manipulace s půdními bloky. Při vytvoření nového půdního bloku si systém vyžádá potřebné informace vztahující se k půdnímu bloku a umožní jednotlivé půdní bloky rozdělit na půdní hony. Dále umožní jejich opětovnou editaci, nebo smazání v případě, že se na půdní blok nevztahuje žádný osevní plán.
- Správa skladu – Skladová část systému bude přehledně rozdělena dle požadovaných kategorií. Jednotlivé části umožní vkládání nových položek do skladu a manipulaci s nimi. Při přidávání položek do skladu, nebo jejich odebírání, systém uloží záznam o pohybu na skladě, na jehož základě bude moci uživatel vytvořit fakturu.
- Informace o uskladněném materiálu - Skladová část systému musí zadavateli přehledně zobrazovat informace o množství a hodnotě uskladněného materiálu. Zároveň musí poskytovat možnost exportu dat pro případ kontroly.
- Kategorie plodin – Zadavatel musí mít možnost sestavit seznam pěstovaných plodin. Tento seznam bude využívat při evidenci přípravků na ochranu rostlin a také během tvorby osevního plánu.
- Evidence techniky a práce – Systém musí poskytovat možnost evidence informací o vlastněné technice a možnosti vykonávané práce na obhospodařované půdě. Zadavatel tak definuje svoje náklady vyjádřené na hektar pozemku dle jednotlivých prací a předpokládanou spotřebu jednotlivých strojů při provádění dané práce.
- Sestavení osevního plánu pro jednotlivé půdní hony – Systém musí jednoduše umožnit vytvoření nového osevního plánu. Během tvorby osevního plánu zadavatel zvolí pěstovanou plodinu. Dále umožní do

osevního plánu zařadit potřebné práce. Systém poskytne možnost volby osiva pro danou plodinu. Na základě dávkování systém zobrazí potřebné množství osiva a jeho cenu. Stejně tak poskytne možnost naplánování použití přípravků na ochranu rostlin nebo hnojení. Systém na základě údajů o dávkování jednotlivých materiálů a výměře půdního honu bude kalkulovat předpokládané náklady na splnění osevního plánu během hospodářského roku.

- Skladový alarm – po vytvoření jednotlivých osevních plánů systém varuje uživatele v případě, že na skladě nemá dostatek materiálu pro splnění osevních plánů.
- Kontrola splatnosti faktur – systém musí přehledně zobrazovat faktury u kterých se blíží datum splatnosti a dosud nejsou uhrazeny.
- Kalendář plánu práce – systém uživateli nabídne přehled naplánovaných operací v dané časové době.
- Evidence hnojiv a přípravků na ochranu rostlin – Během hospodářského roku bude zadavatel určovat splnění jednotlivých bodů osevního plánu. Systém pak musí poskytovat možnost exportu potřebných dat pro nařízené kontroly.
- Evidence výnosů – po sklizni musí mít zadavatel možnost přidat do skladu informace o sklizených plodinách. Systém pak musí poskytnout informace o výnosech jednotlivých půdních honů.
- Prodej materiálu – systém musí umožnit prodávat jednotlivé plodiny popřípadě ostatní uskladněný materiál a k nim vystavovat potřebné faktury.

5.1.2 Nefunkční požadavky

- Přehlednost a snadné užití – Navrhovaný systém musí být dostatečně přehledný a srozumitelný. Manipulace s tímto systémem musí být co možná nejvíce intuitivní.
- Dostupnost dokumentace – Aplikace musí poskytovat rychle dostupný popis pro použití jednotlivých částí.
- Podpora webových prohlížečů – systém by měl být podporovaný nejrozšířenějšími internetovými prohlížeči.
- Zabezpečení dat – přístup do systému musí podléhat autorizaci.

5.2 Návrh IS

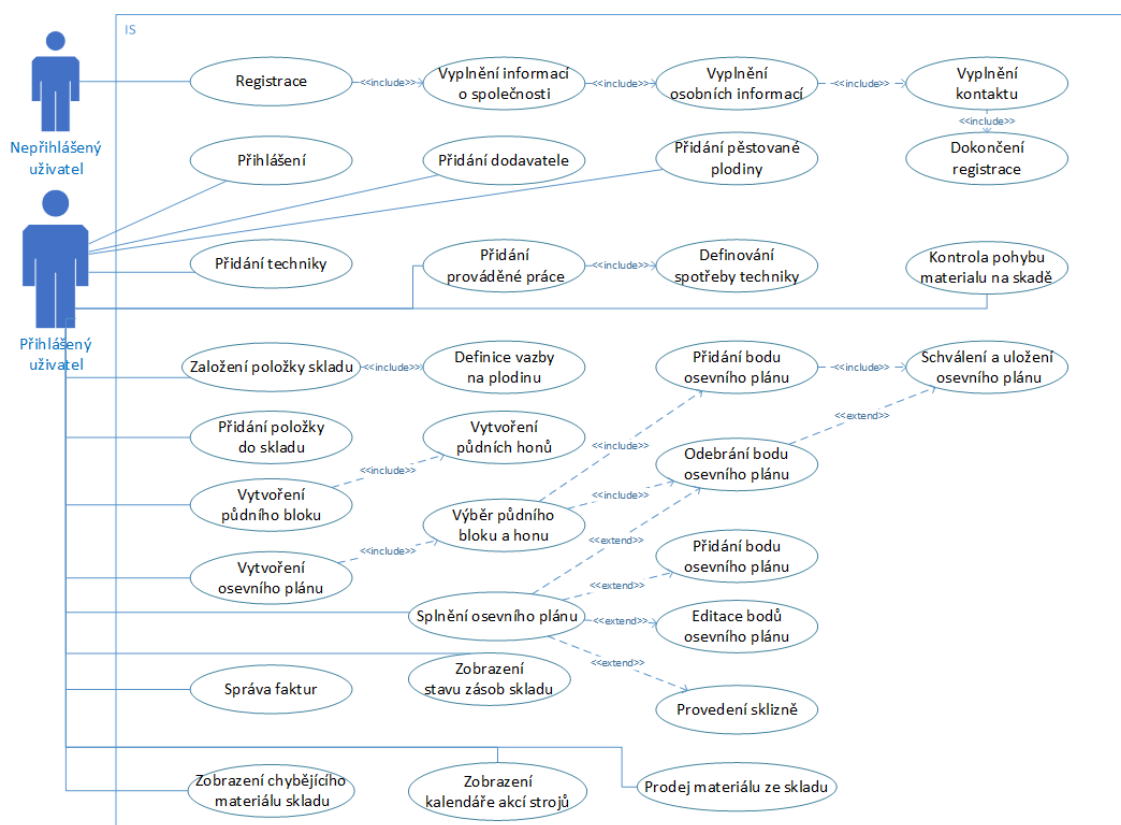
Na základě funkčních a nefunkčních požadavků bylo přistoupeno k samotnému návrhu informačního systému. Po konzultacích se zadavatelem byly sestaveny use case diagramy, které byly také pravidelně konzultovány, aby navržený informační systém co nejlépe splňoval požadavky zadavatele.

Hlavním cílem tohoto systému je usnadnit činnost drobných zemědělců. Jedním z hlavních požadavků je rychle a přehledně poskytovat informace spojené s náklady na jednotlivé půdní hony a následným ziskem po sklizni. S tímto požadavkem je spojena možnost rychle a přehledně vytvářet osevní plány pro jednotlivé půdní bloky. Systém také musí poskytovat možnost evidovat informace o uskladněných materiálech, které jsou následně využity pro tvorbu osevních plánů. Evidence skladu tak bude poskytovat informace o hodnotě a množství uskladněných materiálů. Dále systém musí poskytovat informace o fakturách spojených s pohybem materiálu ve skladu. Zemědělec tak bude mít k dispozici příjmové i výdejové faktury a databázi svých dodavatelů. Je třeba, aby IS poskytoval data spojená jak s uskladněným materiálem, tak s používáním jednotlivých přípravků pro potřeby nařízených kontrol.

Výsledná aplikace musí být uživatelsky přívětivá, dostatečně přehledná a intuitivní. Data, které IS poskytuje, musí být dostupná na různých zařízeních. Z toho důvodu je informační systém vytvořen jako webová aplikace. Hlavní funkční požadavky tedy jsou vytvoření půdního bloku s možností jeho rozdělení na půdní hony, evidence osiv, hnojiv a přípravků na ochranu rostlin, vytvoření osevního plánu pro jednotlivé půdní hony s využitím konkrétních materiálů ze skladu a kalkulací předpokládaných nákladů. Součástí vytvoření osevních plánů je plánování obhospodařovacích prací na půdních blocích a tím vytvoření kalendáře nasazení potřebné techniky. Hospodářský rok zakončený sklizní uzavírá osevní plán. Systém tak musí poskytovat možnost zanesení informací spojených se sklizní do skladu a následné vyčíslení skutečných nákladů a výnosů jednotlivých půdních honů. Dále IS poskytuje údaje, které evidují použití přípravků na ochranu rostlin a také evidenci hnojiv. Skladová část systému, do které jsou ukládány informace o množství sklizených plodin, také musí umožnit prodej jednotlivých plodin nebo ostatního uloženého materiálu. Ve výsledku tak systém vyhodnotí i ziskovost jednotlivých půdních honů.

5.2.1 Návrh funkcionality informačního systému

Na základě analýzy byl proveden návrh informačního systému, který byl namodelován pomocí use case diagramu. Návrh byl následně konzultován a vyvíjen. Výsledný use case diagram uvedený níže popisuje funkce informačního systému.



Obrázek 3: Use case diagram akcí uživatele

Informační systém musí uživateli poskytovat akce navržené pomocí use case diagramu. Na základě navrženého use case diagramu bylo přistoupeno k návrhu databáze a uživatelského rozhraní, ERD databáze je uveden níže.

5.2.1.1 Popis základního use case informačního systému

- **Registrace**

Základní akcí systému bude registrace nového uživatele. Systém uživateli poskytne možnost vyplnění údajů o vlastní společnosti, osobních informací a kontaktu. Následně systém poskytne uživateli kontrolu zadaných údajů a možnost dokončení registrace. Součástí registrace uživatele je vytvoření záznamu v tabulce dodavatelů. Tento záznam nazvaný „Vlastní výroba“ dále v systému slouží pro monitorování položek skladu, které se vážou přímo na vlastní společnost. V příloze A je uveden diagram aktivit, který popisuje proces registrace.

- **Přihlášení do systému**

Ostatní funkce informačního systému jsou dostupné pouze pro autorizované uživatele. Z tohoto důvodu je jediným vstupním bodem aplikace akce přihlášení uživatele. Diagram aktivit popisující postup přihlášení do systému je uveden v příloze B.

- **Přidání dodavatele**

IS ukládá dodavatele, se kterými uživatel spolupracuje. Uživatel tak musí mít k dispozici možnost přidávat, odebírat a editovat jednotlivé dodavatele. Možnost akcí souvisejících s manipulací s dodavateli je k dispozici v části systému pro ně určené. Dále také musí mít uživatel možnost přidat nového dodavatele v případě, kdy vybírá vazbu na konkrétního dodavatele. Např. při vkládání položky do skladu, vybere dodavatele ze seznamu, nebo vytvoří nového.

- **Přidání pěstované plodiny**

Každý uživatel může být zaměřen na pěstování různých plodin. IS musí uživateli poskytnout možnost vytvořit si vlastní seznam pěstovaných plodin, se kterými uživatel v systému dále pracuje. Diagram aktivit popisující činnost systému během této akce se nachází v příloze D.

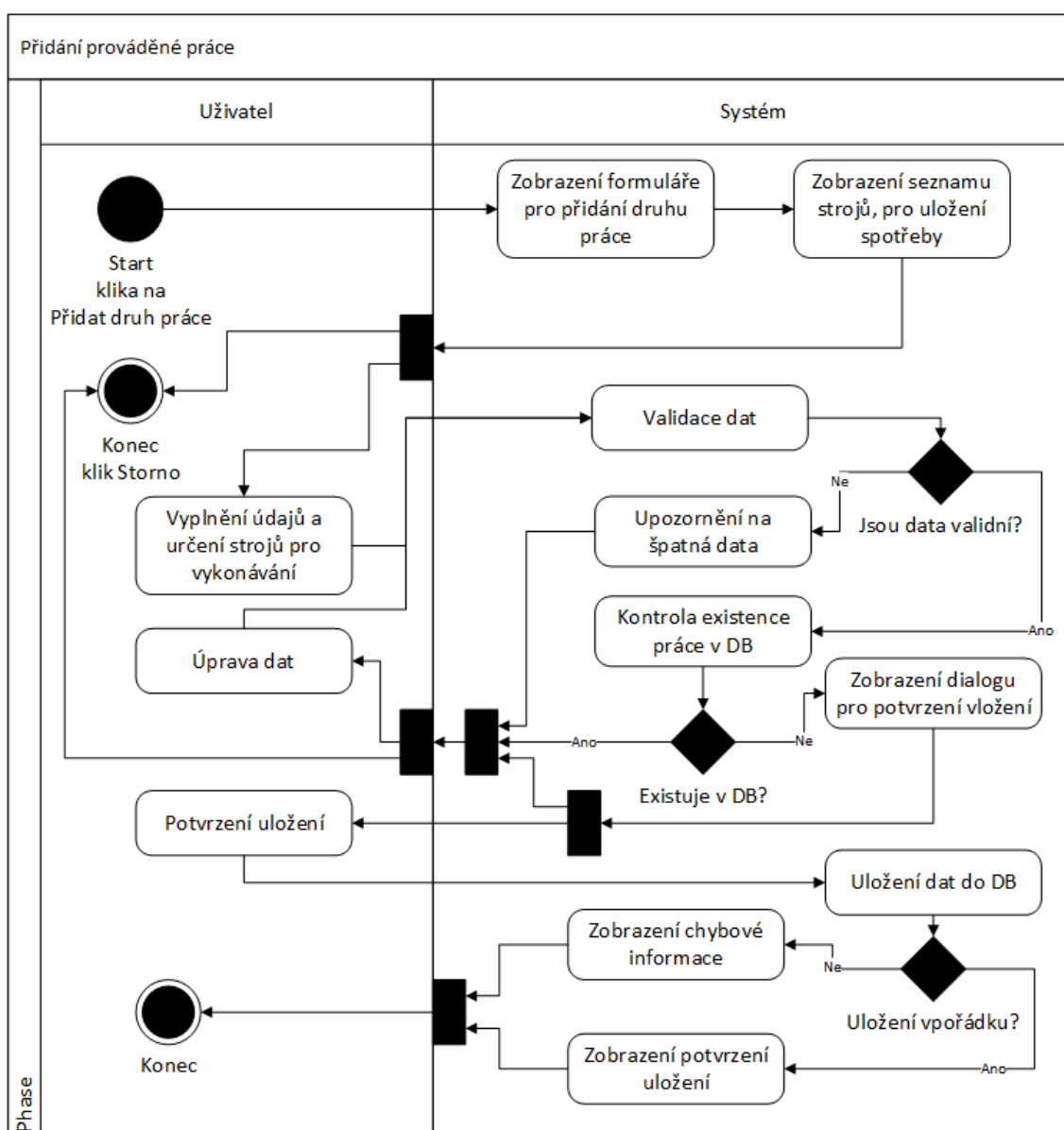
- **Přidání techniky**

Pro možnost plánování osevního plánu je nutné, aby systém evidoval jednotlivé stroje, které má uživatel k dispozici. Klient tak bude mít k dispozici kalendář naplánovaných akcí pro jednotlivé stroje, data pro vytvoření statistik ohledně spotřeby a délky plnění jednotlivých úkonů. Systém také bude evidovat údržbu strojů. Klient tak bude mít k dispozici přehled výdajů na údržbu jednotlivých strojů. Diagram aktivit je uveden v příloze C.

- **Přidání prováděné práce**

Každá akce, kterou klient plánuje v osevním plánu, je vázaná na určitou práci. Klient musí mít možnost nadefinovat si vlastní seznam prováděných prací, určit náklady v závislosti na hektar a také definovat průměrnou spotřebu jednotlivých strojů při provádění dané práce. Níže uvedený diagram aktivit uvádí činnost systému v průběhu akce přidání nové práce do systému.

Pomocí diagramu aktivit lze vyjádřit, v jakých stavech se systém bude nacházet. Počáteční bod diagramu aktivit je zobrazován jako černý kruh. Definuje, počáteční stav systému, za jakých podmínek se modelovaná činnost spustí. Koncoví stav se poté zobrazuje jako prázdný a plný kruh. Další entitou diagramu aktivit jsou jednotlivé činnosti zobrazované jako ovál s popisem konkrétní aktivity. Pomocí podmínek guard condition můžeme znázornit větvení jednotlivých aktivit. Pomocí speciálních elementů decision branch a decision merge můžeme znázornit rozdělení nebo spojení větvení diagramu (Zelinka. 2009).



Obrázek 4: Diagram aktivit přidání prováděné práce

- **Vytvoření půdního bloku**

Každý uživatel spravuje své vlastní půdní bloky. Musí tak mít k dispozici možnost přidávat jednotlivé půdní bloky s příslušnými informacemi. Jednotlivé půdní bloky se dále dělí na půdní hony. Uživatel může půdní blok rozdělit na libovolný počet půdních honů. Uživatel také musí mít možnost upravovat, nebo mazat jednotlivé půdní bloky a půdní hony, pokud se na ně neváže vytvořený osevní plán. Diagram aktivit pro vytvoření půdního bloku je uveden v příloze E.

- **Založení položky skladu**

IS jednotlivým uživatelům musí poskytovat správu skladu. Sklad se dělí do šesti základních kategorií osiva, hnojiva, přípravky na ochranu rostlin, pohonné hmoty, sklizeň a ostatní položky. Uživatel musí mít možnost vkládat položky do jednotlivých kategorií, kde definuje jejich použití pro jednotlivé pěstované plodiny, doporučené dávkování a v případě hnojiv množství látek dodaných do půdy. Systém v závislosti na vložené nové položce vytvoří záznam o pohybu na skladě. Diagram aktivit popisující činnost během vytváření nové položky skladu je uveden v příloze F.

- **Přidání položky do skladu**

Uživatel si pomocí zakládání položek skladu vytvoří vlastní strukturu skladu. Dále potom má možnost přidávat přikoupený materiál na sklad. Informační systém pak ve skladové části uchovává reálné množství materiálu na skladě. Stejně jako při vytvoření nové položky se uloží záznam o pohybu na skladě.

- **Zobrazení stavu zásob skladu**

Nezbytnou součástí skladovací části systému je poskytování přehledných dat o aktuálním stavu zásob. Uživatel tak má k dispozici informace o aktuálním množství materiálu a jeho hodnotě.

- **Vytvoření osevního plánu**

Jednou z hlavních funkcí systému je vytvoření osevního plánu pro jednotlivé půdní hony. Půdní hony jsou vytvořené pro každý hospodářský rok a vázané na jednotlivé půdní bloky. Vytvoření osevního plánu je podrobněji navrženo ve vlastním use case uvedeném níže.

- **Zobrazení chybějícího materiálu skladu**

Informační systém je určený k usnadnění práce během plánování osevního plánu. Uživatel v závislosti na vytvořených osevních plánech musí mít k dispozici informace o potřebném množství materiálu, které je třeba pro splnění jednotlivých položek osevního plánu. Uživatel má tak k dispozici informace o předpokládaných nákladech.

- **Splnění osevního plánu**

Uživatel musí mít přehledně k dispozici přehled svých nadefinovaných osevních plánů. V závislosti na této akci musí mít k dispozici možnost splnit jednotlivé body osevního plánu. To bude mít za následek uložení reálných nákladů na splnění jednotlivých položek osevního plánu. V průběhu plnění musí mít uživatel možnost editovat data v jednotlivých položkách, odebírat je z osevního plánu, nebo přidávat nové položky.

- **Provedení sklizně**

Hospodářský rok je ukončený sklizní. Uživatel tak po splnění všech položek osevního plánu má k dispozici možnost uložení dat týkajících se sklizně jednotlivých plodin. Po zadání výnosu je vyjádřena hodnota vypěstovaného kilogramu plodiny a uživatel tak má k dispozici během prodeje informace o nákladech na vypěstování dané plodiny.

- **Zobrazení kalendáře strojů**

IS na základě sestavení osevních plánu uživatele informuje o naplánované činnosti jednotlivých strojů.

- **Prodej materiálu skladu**

Uživatel musí mít k dispozici možnost prodávat vypěstované a uskladněné plodiny.

- **Kontrola pohybu materiálu na skladě**

Uživatel musí mít k dispozici přehled pohybu materiálu na skladě. Záznamy pro kontrolu této činnosti systém ukládá automaticky při vkládání položek na sklad, jejich aplikaci při splnění osevního plánu nebo při případném prodeji materiálu.

- **Správa faktur**

Systém uživateli poskytuje možnost evidence faktur pro jednotlivé pohyby materiálu na skladě. Uživatel vytváří jednotlivé faktury pro konkrétní dodavatele. Obsah faktury může uživatel zadat ručně, nebo sestavit fakturu pomocí záznamů o pohybu na skladě. Během vytváření faktury uživatel vybere druh platby, faktury a datum splatnosti. Faktura také nese informaci o tom, zda byla uhrazena.

5.2.1.2 Popis use case Vytvoření osevního plánu

Následující diagram popisuje návrh funkcionality, která je nutná pro správné sestavení osevního plánu. Pro každý hospodářský rok musí zemědělec vytvořit osevní plány jednotlivých půdních honů. Během této činnosti se tak plánuje jaký materiál a jaké práce na půdních honěch budou prováděny během hospodářského roku. Jedním z hlavních cílů informačního systému je tuto činnost zemědělci usnadnit a přehlednou formou poskytovat informace o potřebném množství materiálu a předpokládaných nákladech na daný osevní plán. Systém uchovává údaje o již provedených osevních plánech a uživatel má tak k dispozici data o skutečných nákladech a množství potřebného materiálu na již splněné položky osevních plánů.



Obrázek 5: Use case vytvoření osevního plánu

- **Výběr hospodářského roku, půdního bloku a honu**

Uživatel může tvořit osevní plány pro jednotlivé půdní hony. Pro rychlejší a přehlednější orientaci během vytváření osevních plánů uživatel nejprve vybere půdní blok, ze kterého následně vybere konkrétní půdní hon. Zobrazené půdní hony také odpovídají zvolenému hospodářskému roku. Následně definuje uživatel název osevního plánu, pod kterým bude plán v systému dále vystupovat.

- **Přidání/odebrání položky osevního plánu**

Osevní plán se skládá z jednotlivých položek osevního plánu. Položky osevního plánu jsou rozděleny do čtyř základních skupin osiva, hnojiva, přípravky na ochranu rostlin a samostatné práce. Pomocí těchto položek uživatel sestavuje jednotlivé osevní plány. Má také k dispozici možnost přidání položky odebrat a tím manipulovat s osevním plánem.

- **Výběr materiálu, práce a techniky**

Po přidání některé z položek osevního plánu musí mít uživatel k dispozici výběr materiálu ze skladu, který chce použít. Systém následně na základě zvoleného dávkování vypočítá množství materiálu, které je potřebné pro splnění položky. Uživatel je také informován o množství uskladněného materiálu a předpokládaných nákladech na materiál. Součástí položky osevního plánu je také definování práce spojené s vykonáním daného bodu např. orba. Následně má uživatel k dispozici informaci o nákladech na provedení práce, týkající se naplánovaného bodu. V závislosti na potřebné práci uživatel zvolí stroj, který bude danou práci vykonávat. Poté je vypočítána předpokládaná spotřeba stroje a na základě zvolených

pohonných hmot jsou vyčísleny předběžné náklady. Díky zvoleným hodnotám tak bude mít uživatel k dispozici informace o předpokládaných nákladech a množství potřebného materiálu jak pro splnění jednotlivých bodů osevního plánu tak náklady na celý hospodářský rok.

- **Kontrola sestaveného osevního plánu**

Před potvrzením a uložením do systému musí systém uživateli přehledně zobrazit navržený osevní plán. Uživatel dále tento osevní plán uloží do systému, nebo se může vrátit k editaci jednotlivých položek osevního plánu.

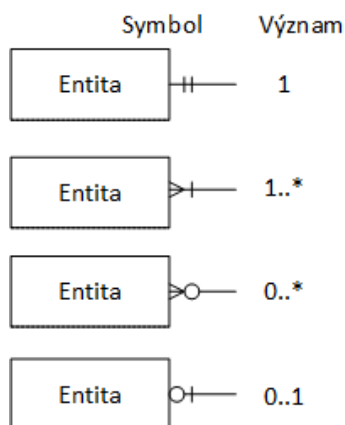
5.3 Návrh datové vrstvy

Modelování datových struktur lze rozlišit na základě úrovně popisu struktury modelu. V případě, že je model zaměřen pouze na popis obsahu dat v systému, nezávisle na vlastním implementačním a technologickém prostředí, jedná se o konceptuální model datové vrstvy. V případě, že entity modelu obsahují také klíče (primární a cizí), které realizují vazby mezi entitami, jedná se o logický model. Fyzický model je navíc obohacen o typy jednotlivých atributů, jedná se o popis vlastní realizace databáze v konkrétním implementačním prostředí. (Vymětal, 2009)

5.3.1 Konceptuální model datové vrstvy

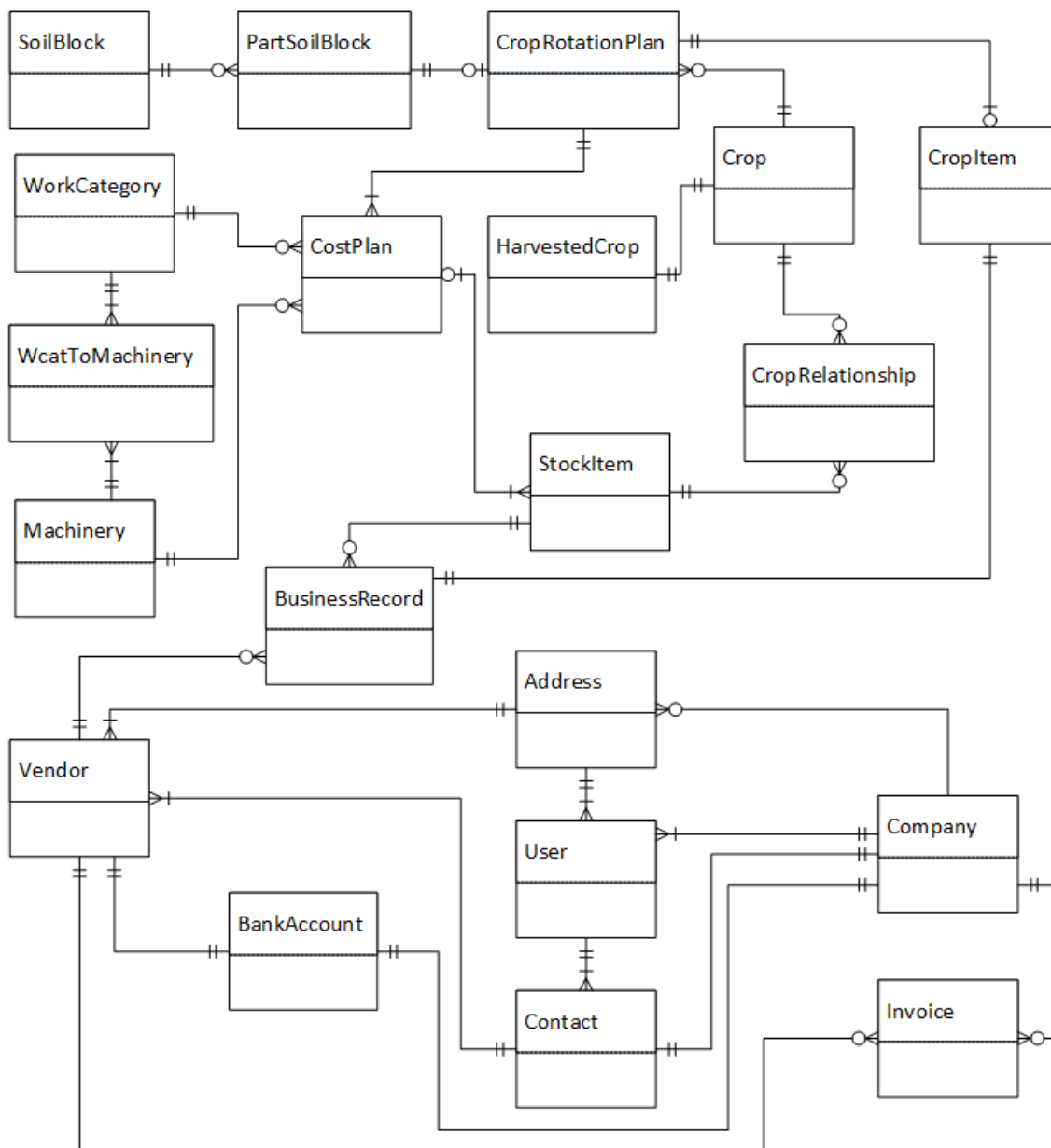
Jak bylo uvedeno výše, konceptuální model datové vrstvy se zaměřuje na popis dat, se kterými systém pracuje. Pro zobrazení konceptuálního modelu se používá entitně relační diagram. Pomocí ERD jsme schopni vyjádřit jednotlivé entity, které odpovídají datovým objektům vztahy mezi nimi a jejich podstatné atributy. Níže uvedený zjednodušený konceptuální model znázorňuje pouze názvy entity a jejich vazby, vlastnosti budou podrobně popsány níže. Podrobné ERD je k dispozici na příloženém CD.

Pro vyjádření násobnosti vztahů je použita Crow's foot notace pro entitně relační diagramy. Níže uvedený obrázek vysvětluje význam jednotlivých symbolů.



Obrázek 6: Crow's foot notace

Zdroj: (Shelly, Rosenblatt, 2012), vlastní úprava



Obrázek 7: Zjednodušený konceptuální model datové vrstvy

5.3.1.1 Popis vlastností entit konceptuálního modelu

Dále bylo přistoupeno k návrhu databáze. Následující část je věnována detailnímu popisu entit výše uvedeného konceptuálního modelu, který jsem vypracova na základě zpracovaných požadavků. Na základě tohoto konceptuálního modelu byl navrhnut fyzický model, který lze nalézt v příloze. Entity konceptuálního modelu odpovídají entitám fyzického modelu, z tohoto důvodu budou popsány i atributy

reprezentující relace mezi entitami. Všechny entity mají atribut Id, jenž slouží jako primární klíč a atribut CreatedAt, který ukládá datum vzniku záznamu v databázi.

- **User** – Entita reprezentující tabulku databáze, do které se ukládají jednotliví uživatelé. Tato entita má atributy Name, LastName, Role, ContactId, AddressId, Password, CompanyId, UserName a Counter. Atribut Role slouží pro identifikaci práv uživatele. CompanyId slouží k přiřazení uživatelů do konkrétní společnosti a atribut UserName slouží jako přihlašovací jméno do informačního systému.
- **Address, Contact** – Jedná se o entitu uchovávající informace o adresách a kontaktech uživatelů, dodavatelů a jednotlivých společností.
- **Vendor** – Slouží pro ukládání jednotlivých dodavatelů, se kterými uživatel spolupracuje. Entita obsahuje cizí klíče definující vazbu s entitou Address, Contact, Company a BankAccount. Dále také obsahuje atributy pro uložení čísla ičo, dič, název společnosti a atribut Production, který slouží pro vytvoření dodavatele „Vlastní výroba“. Dodavatel nazvaný vlastní výroba je v systému vytvořen automaticky během registrace a slouží pro evidenci materiálu vlastní produkce.
- **SoilBlock** – entita slouží k uchování informací o jednotlivých půdních blocích. Má atributy pro název půdního bloku, jeho výměru, identifikační číslo a číslo čtverce v registru půdy. Dále má také atributy N,S,P a K, které slouží k uložení obsahu živin v půdě.
- **PartSoilBlock** – entita reprezentuje rozdělení půdních bloků na jednotlivé půdní hony. Mezi entitou SoilBlock a PartSoilBlock existuje relace. Půdní blok může být rozdělen na libovolný počet půdních honů. PartSoilBlock má dále atributy marketingYear, který uchovává příslušnost půdního honu k danému hospodářskému roku. Další atribut je Area, který uchovává informaci o výměře jednotlivých půdních honů. A atribut description pro popis jednotlivých půdních honů.
- **Machinery** – Entita reprezentuje jednotlivé stroje, které uživatel využívá pro práci zadanou v osevním plánu. Uchovává informace o názvu stroje, spz, najeté motohodiny a popis stroje.
- **WorkCategory** – entita odpovídá druhům práce, které si klient v systému definuje. Tyto práce jsou nezbytnou součástí pro definování položek osevního plánu. Entita má atributy pro název práce, libovolný popis a atribut Price sloužící pro uchování informace o nákladech spojených s prací definovanou v Kč na hektar.
- **WcatToMachine** – tato entita slouží k uchování informací o spotřebě strojů v závislosti na prováděné práci.
- **Crop** – entita uchovává informace o pěstovaných plodinách jednotlivých uživatelů. Také má atributy N,S,P a K, které uchovávají informace o množství látek odebraných z půdy po sklizení jedné tuny této plodiny.
- **StockItem** – entita odpovídá jednotlivým položkám uskladněných na skladě. Pro potřeby skladu entita obsahuje atributy reprezentující hodnotu a množství uskladněného materiálu. Dále také název a popis položky, také

atributy N,S,P a K, které definují množství látek přidané do půdy použitím daného materiálu v osevním plánu. Atributy CategoryId a MeasureId slouží pro rozdělení skladu do jednotlivých celků a to osiva, hnojiva, přípravky na ochranu rostlin, pohonné hmoty a ostatní položky. Sklad nabízí ještě kategorii sklizeň, položky této kategorie reprezentuje entita HarvestedCrop. MeasureId definuje měrnou jednotku jednotlivých položek. V systému mohou být položky uchovány v kg, l nebo balících. Posledním atributem je atribut Dosage, sloužící pro definování doporučeného dávkování materiálu.

- **CropRelationship** – systém umožňuje uživatelům definovat u jednotlivých položek skladu pro kterou plodinu má být materiál použit. Tato entita slouží pro uchování daných vztahů.
- **CropRotationPlan** – jedná se o stěžejní entitu informačního systému. Reprezentuje jednotlivé osevní plány, které uživatel sestavuje. Obsahuje atribut PartSoilBlockId, který slouží jako cizí klíč pro relaci definující příslušnost osevního plánu k jednotlivému půdnímu honu. Další atributy slouží pro popis osevního plánu, uchování předpokládaných nákladů, reálných nákladů, uchování výnosu osevního plánu. Také cizí klíč CropId pro relaci definující pěstovanou plodinu a atribut uchovávající datum poslední změny.
- **CostPlan** – plán nákladů reprezentuje jednotlivé body osevního plánu. Uchovává informace o předpokládaných nákladech na splnění položky, materiál použitý v naplánovaném bodu osevního plánu, také práci, kterou je třeba vykonat a stroj sloužící pro splnění položky osevního plánu. Entita také obsahuje cizí klíč pro uchování požitých pohonných hmot. Atribut uchovávající informaci o naplánovaném datu aplikace, potřebné množství a zvolené dávkování vybraného materiálu. Také obsahují atributy N,S,P a K uchovávající informace o množství přidaných látek do půdy po splnění položky osevního plánu. Atribut status uchovává informaci o tom, zda byl bod osevního plánu splněn a atribut realCost sloužící k uchování reálných nákladu na položku osevního plánu. Reálné náklady nemusí odpovídat předpokládaným nákladům např. z důvodu měnící se spotřeby stroje.
- **CropItem** – entita uchovává informace o množství a nákladech na sklizené plodiny z jednotlivých půdních honů.
- **HarvestedCrop** – entita reprezentuje jednotlivé vypěstované plodiny, které zemědělec uskládňuje na skladě. Obsahuje atributy pro uchování informace o množství, hodnotě, data změny a cizí klíč pro relaci s jednotlivými plodinami.
- **BusinessRecord** – tato entita slouží pro uchování informací o pohybu na skladě. Obsahuje atributy reprezentující cizí klíče pro relace mezi dodavateli, jednotlivými položkami osevního plánu. Atribut Income uchovává informaci, zda se jedná o příjem nebo výdej skladu. Dále obsahuje atributy uchovávající cenu a množství materiálu. Dále také atribut zařazující položky do skladových kategorií a měrné jednotce.

6 Realizace IS

V této části práce bude čtenář seznámen s postupem implementace informačního systému na základě výše uvedeného návrhu. Tento informační systém pro drobné zemědělce je vytvořen jako webová aplikace. Výsledná aplikace je umístěna na cloudové službě Azure. Aplikace je vyvinuta pomocí IDE Microsoft Visual studio 2013. Technologie použité pro vytvoření této aplikace jsou popsány níže a v kapitole 4.

6.1 Databáze aplikace

Na základě ERD bylo třeba vytvořit databázi, do které bude webová aplikace ukládat potřebná data. Jako databázový systém byl zvolen SQL Server od společnosti Microsoft. Pro implementaci a modifikaci databáze byl vytvořen databázový projekt, který usnadňuje práci s návrhem a nasazením samotné databáze. Visual studio umožňuje vytváření jednotlivých tabulek pomocí grafického rozhraní, nebo lze sepsat přímo kód jednotlivých tabulek databáze. Do databázového projektu byly postupně vytvořeny všechny potřebné tabulky databáze. Níže je uvedena ukázka kódu, pomocí kterého je vytvořena tabulka ukládající informace o půdních honech.

```
CREATE TABLE [dbo].[PartSoilBlock]
(
    [Id] INT NOT NULL PRIMARY KEY IDENTITY(1, 1),
    [SoilBlockId] INT NULL,
    [Description] NVARCHAR(50) NULL,
    [CreatedAt] DATETIMEOFFSET NULL,
    [Area] FLOAT NULL,
    [MarketingYear] INT NULL,
    CONSTRAINT [FK_PartSoilBlock_SoilBlock] FOREIGN KEY
    ([SoilBlockId]) REFERENCES [SoilBlock] ([Id])
)
```

Jedná se o kód proprietárního rozšíření jazyka SQL společností Microsoft, jazyk je nazván Transact-SQL. Microsoft tento jazyk využívá ve svých produktech pro práci s Microsoft SQL Serverem. Tabulky navržené relační databáze jsou vytvořeny pomocí výše uvedeného kódu. Ten je složen s příkazu pro vytvoření nové tabulky s atributy, které jsou uzavřeny v kulatých závorkách. Každá tabulka relační databáze obsahuje atribut `Id`, jenž slouží jako primární klíč dané tabulky a atribut `CreatedAt`, který ukládá informaci o vzniku záznamu v databázi. Kód každému atributu definuje jeho název, datový typ a podmínku, zda může atribut nabývat hodnoty `null`. Primární klíče záznamů v jednotlivých tabulkách musí být unikátní, to je zajištěno pomocí příkazu `IDENTITY(1,1)`, který automaticky generuje čísla vyjadřující primární klíče. Jako parametry má tento příkaz počáteční hodnotu a hodnotu skoku při generování. Dále lze v kódu pomocí struktury `CONSTRAINT [FK_PartSoilBlock_SoilBlock] FOREIGN KEY ([SoilBlockId]) REFERENCES [SoilBlock] ([Id])` vytvořit potřebné cizí klíče pro relační databázi. Příkaz uvádí,

že se jedná o cizí klíč mezi tabulkou `partSoilBlock` a `Soillock`, u kterých existuje relace pomocí atributů `SoilBlockId` a `SoilBlock.Id`.

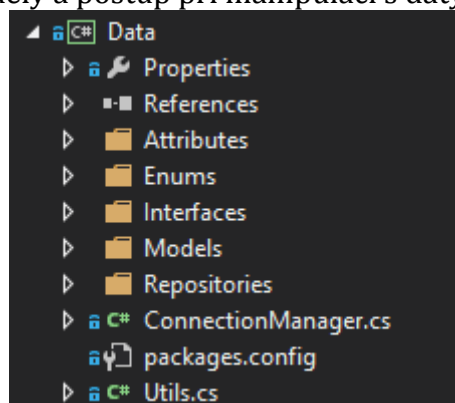
Databázový projekt, ve kterém je relační databáze implementována tak slouží pro jednoduchou správu celé databáze. Visual studio nabízí funkci `publish`, pomocí které je databáze snadno nasazena na databázový server. Celý kód databázového projektu lze nalézt na přiloženém CD.

6.2 Datová vrstva

Úkolem datové vrstvy je přímá manipulace s daty. Struktura projektu s datovou vrstvou je poměrně jednoduchá. Obsahuje implementace tříd, které tvoří datový model a následně třídy sloužící pro práci s jednotlivými entitami relační databáze. Všechny entity relační databáze musí mít v datové vrstvě implementovanou třídu, která slouží jako model příslušné entity. Třída modelu je obecně velice jednoduchá, název každého modelu odbovídá názvu entitě v relační databázi a obsahuje příslušné atributy. Níže je uveden příklad implementace jednoho z modelů. Každý model dědí ze základního modelu, který obsahuje atribut `Id` a `CreatedAt`. Tyto atributy jsou povinné pro všechny modely.

```
namespace Data.Models
{
    public class PartSoilBlock : BaseModel
    {
        public long SoilBlockId { get; set; }
        public string Description { get; set; }
        public float Area { get; set; }
        public int MarketingYear { get; set; }
    }
}
```

Příklad komunikace datové vrstvy a vrstvy aplikační je v práci uveden níže. Tato část se věnuje popisu datového modelu, principu mapování dat na jednotlivé modely a postup při manipulaci s daty.



Obrázek 8: Struktura datové vrstvy aplikace

Struktura datové vrstvy je rozdělena jak znázorňuje výše uvedený obrázek. Základní třídou tohoto projektu je třída `ConnectionManager`, která slouží pro

vytvoření spojení s databází a tím umožní práci s daty. Jedná se o public třídu obsahující jednu statickou metodu `getConnection()`. Tato metoda vytváří novou instanci třídy `System.Data.SqlClient`, která slouží pro připojení ke konkrétní databázi definovanou parametrem této metody. Složka `Interfaces` slouží k uložení rozhraní, které datová vrstva využívá. Projekt definuje rozhraní `IRepository`, které implementuje třída sloužící pro práci s konkrétními daty. Detailní popis rozhraní je uveden níže. Projekt datové vrstvy dále obsahuje složky `Enums` a `Attributes` obsahující pomocné třídy a množiny, které využívají jednotlivé modely. Jedná se například o množiny sloužící k vyjádření měrné jednotky, nebo pro práci s jednotlivými kategoriemi skladu. Statická třída `Utils` obsahuje pomocné statické metody například pro šifrování hesla. Dále můžeme vidět složky `Models` a `Repositories`. Jedná se o hlavní části datové vrstvy. Složka `Models` obsahuje třídy reprezentující entity relační databáze a složka `Repositories` třídy určené pro práci s daty. Jejich podrobnější popis je uveden v samostatných podkapitolách.

6.2.1 Doménový model

Základní částí datové vrstvy je doménový model. Ten obsahuje třídy reprezentující modely, pro jednotlivé entity relační databáze. Struktura ERD je popsána výše a podrobný diagram používané databáze lze najít v příloze. Jak již bylo zmíněno, třídy reprezentující jednotlivé modely jsou uloženy ve složce `Models` projektu datové vrstvy. Názvy jednotlivých tříd, odpovídají názvům tabulek v relační databázi. Instance těchto tříd slouží pro vyjádření konkrétních dat uložených v databázi. Jak bylo zmíněno v části návrhu, všechny tabulky relační databáze obsahují atribut `Id`, jenž slouží jako primární klíč a atribut `CreatedAt`, pro uchování data vzniku záznamu. Aplikace slouží pro uchování dat jednotlivých uživatelů, kteří patří do konkrétní registrované společnosti v této aplikaci. Z tohoto důvodu mají určité entity databáze atribut `CompanyId`.

Jako základní třída pro modely je implementována abstraktní třída `BaseModel`, obsahující právě atributy `Id` a `CreatedAt`. Z této třídy dále dědí všechny potřebné modely. Pro vyjádření relace některých entit s entitou `Company` je implementována třída dědící `BaseModel` s názvem `CompanyRelatedModel` s atributem `CompanyId`, ze které dědí potřebné modely. Na základě faktu, že všechny modely dědí z třídy `BaseModel` byla implementována generická třída pro práci s daty, ta bude popsána níže. Všechny třídy sloužící jako modely jsou implementovány jako `public class Název modelu : třída ze které model dědí` a v složených závorkách uvedeny datové typy a názvy atributů modelované entity. Ukázka kódu implementující základní třídy a konkrétní modely je uveden v příloze (H).

6.2.2 Manipulace s daty

Datová vrstva aplikace jako jediná přímo manipuluje s daty. Třídy určené pro manipulaci s daty využívají instance tříd uložených ve složce `models`, které reprezentují modely entit databáze. Tyto třídy jsou uloženy ve složce `Repositories`

datového projektu a implementovány pomocí návrhového vzoru repository pattern. Datová vrstva této aplikace pro práci s databází využívá základní třídu `BaseRepository`, která je generická a implementuje základní CRUD metody. Pro správnou funkcionalitu je nutné u některých modelů implementovat metody, které nad databází provádějí složitější operace. Toho je docíleno za použití extenzních metod.

6.2.2.1 Použití návrhového vzoru RepositoryPattern

Jak již bylo zmíněno, datová vrstva aplikace definuje rozhraní `IRpositories`. Toto rozhraní musí implementovat třída, která slouží k manipulaci s daty. Rozhraní vynucuje implementaci základních CRUD operací (jedná se o základní operace nad záznamy uloženými v relační databázi), nutných pro manipulaci s daty. `IRpositories` implementuje generická třída `BaseRepositories`, která zajišťuje provádění CRUD operací nad všemi entitami relační databáze. Genericita umožňuje implementování metod nad neznámými typy, díky tomu třída `BaseRepositories` implementuje metody pro práci nad všemi objekty, které dědí ze základní třídy `BaseModel`. Tato třída tak tvoří jediné místo v aplikaci, kde dochází k přímé manipulaci s daty uloženými v relační databázi. Jednotlivé metody této třídy tvoří rozhraní datové vrstvy, které využívá aplikační vrstva pro práci s daty. Jednotlivé metody využívají pro práci s databází třídu `SqlConnection` a její metodu `Open()` zajišťující navázání spojení. Manipulace s jednotlivými záznamy je zajištěna pomocí dotazovacího jazyka SQL. ORM `Dapper` následně zajistí automatickou konverzi záznamů v relační databázi na instance objektů, které je reprezentují. Níže je popsána implementace vybrané metody třídy `BaseRepository`. Kompletní kód třídy je uveden na příloženém CD. Tato metoda slouží pro získání všech záznamů určité entity uložené v relační databázi.

```
public async virtual Task<IEnumerable<T>> GetAll(long CompanyId) {
    using (SqlConnection connection =
        ConnectionManager.GetConnection()) {
        connection.Open();
        var properties = typeof(T).GetProperties();
        string command = "";
        if (Array.Exists(properties,
            col => col.Name == „CompanyId”))
        {
            command = String.Format(„SELECT * FROM [{0}]
                WHERE CompanyId = @CompanyId”, typeof(T).Name);
            return await Task.FromResult<IEnumerable<T>>
                (connection.Query<T>(@command,
                    new { CompanyId = CompanyId }));
        }
        else {
            command = String.Format(„SELECT * FROM [{0}]”,
                typeof(T).Name);
            return await Task.FromResult<IEnumerable<T>>
                (connection.Query<T>(@command));
        }
    }
}
```

Jednotliví uživatelé aplikace jsou jednoznačně rozlišeni pomocí atributu `CompanyId`. Databáze tak může spravovat velké množství klientů. Metody implementované v třídě `BaseRepository` jsou používány pro manipulaci s daty nad všemi entitami relační databáze. Z tohoto důvodu je nutné, aby metode `GetAll`, která slouží pro vrácení všech dat z konkrétní tabulky databáze, identifikovala zda model obsahuje atribut `CompanyId`. Pokud ano, omezí se vyhledávání podle id konkrétní společnosti. Metoda po navázání spojení s databází do proměnné `properties` uloží atributy objektu, který metodu vyvolal. A pokud objekt obsahuje atribut `CompanyId` je proveden SQL dotaz, který vrátí z databáze všechny záznamy s vazbou na příslušnou společnost. Metodě je v parametru předáno id společnosti přihlášeného uživatele. Dotazovaná tabulka je zadána pomocí parametru v závislosti na typu objektu, který metodu vyvolal. Jak již bylo zmíněno ORM `Dapper` zajistí automatickou konverzi a metoda vrací kolekci záznamů příslušného typu.

Pomocí výše uvedené třídy jsou zajištěny základní operace nad všemi tabulkami relační databáze. Jejich konkrétní používání bude popsáno níže v části věnující se aplikační vrstvě. Provádění složitějších dotazů nad jednotlivými tabulkami je zajištěno implementací konkrétních repositářů. Jedná se o statické třídy, které využívají extenzní metody pro rozšíření definovaného rozhraní. Implementace vybraných repositářů je popsána níže.

Třída `SoilBlockRepository` implementuje metodu, která slouží pro zjištění počtu půdních honů jednotlivých půdních bloků.

```
public static class SoilBlockRepository{
    public static async Task <Dictionary<long, int>>
        GetBlocksPartCount(
            this IRepository<SoilBlock> baseRepo, IEnumerable<long> ids)
    {
        using (SqlConnection connection =
            ConnectionManager.GetConnection())
        {
            connection.Open();
            string command = „SELECT SoilBlockId, count(*) As Count
            FROM [PartSoilBlock] WHERE SoilBlockId In @ids GROUP BY
            SoilBlockId“;
            return await Task.FromResult<Dictionary<long, int>>
                (connection.Query(@command, new { ids = ids}))
                .ToDictionary(x => (long)x.SoilBlockId, y => (int)y.Count));
        }
    }
}
```

Za názvem metody je v kulatých závorkách uvedeno `this IRepository<SoilBlock>`. To definuje, že se jedná o extenzní metodu rozšiřující dané rozhraní. Dále parametr, který přijímá kolekci identifikačních čísel konkrétních půdních bloků. Po vytvoření spojení je sestaven SQL dotaz, který je po svém provedení automaticky namapován do datového typu `Dictionary`, který jako

klíč uchovává identifikační číslo půdního bloku a hodnotu počet patřičných půdních bloků.

Tímto způsobem jsou ve složce `Repositories` implementovány statické třídy, které slouží pro pokročilejší práci s entitami relační databáze. Příklady implementace dalších metod, které rozšiřují rozhraní `IRepositories` jsou uvedeny v dalších kapitolách této práce za popisem metod, které volají tyto metody.

6.3 Aplikační vrstva

Tato vrstva zajišťuje komunikaci mezi prezenční a datovou vrstvou této aplikace a provádění výpočetních akcí. V této vrstvě jsou implementovány třídy tzv. serisy, které zprostředkovávají vykonávání konkrétních metod repositářů datové vrstvy.

Základní třídou aplikační vrstvy je generická třída `BaseService`, která implementuje rozhraní `IService`. Toto rozhraní zajišťuje implementaci CRUD metod. Tím je zajištěno, že prezentační vrstva má přístupné základní metody repositářů. Popis komunikace prezenční a aplikační vrstvy je uveden v samostatné kapitole. Jak bylo zmíněno, třída `BaseService` zajišťuje vykonání základních metod pro práci s daty. Níže uvedený příklad ukazuje, že třída `BaseServices` má základní atribut `Repository`, který je typu `IRepository`. Dále obsahuje konstruktor, který vytvoří instanci třídy `BaseRepository` konkrétního modelu určité entity. To zajistí manipulaci pro všechny implementované modely relační databáze.

```
public class BaseService<T> : IService<T> where T : BaseModel
{
    public IRepository<T> Repository { get; set; }
    public BaseService()
    {
        Repository = new BaseRepository<T>();
    }
    public async virtual Task<IEnumerable<T>>
    GetAll(long CompanyId)
    {
        return await Repository.GetAll(CompanyId);
    }
    ...
}
```

V předchozí ukázce kódu je uvedena implementace metody `GetAll`, která volá metodu datové vrstvy uvedenou v kapitole 5.3.2.2. Obdobně jsou implementovány ostatní CRUD metody.

Projekt aplikační vrstvy dále obsahuje implementace tříd, které slouží pro vykonávání operací nad jednotlivými modely. Pro každý model, který je implementovaný v datové vrstvě je v aplikační vrstvě implementována třída, která má složený název z názvu modelu a slova `Service`. Tyto třídy dědí ze základní třídy `BaseService` a předávají do konstruktoru této třídy konkrétní model. Tímto způsobem je zajištěna komunikace mezi aplikační a datovou vrstvou.

Třídy zajišťující práci nad jednotlivými modely implementují metody, které slouží jako rozhraní pro prezenční vrstvu. Tyto metody zajišťují vykonání potřebných operací a obstarávají volání metod konkrétních tříd datové vrstvy určených pro manipulaci s daty v relační databázi. Jako příklad je zde uveden

zdrojový kód třídy `SoilBlockServices`, kde bude vysvětlen princip implementace tříd aplikační vrstvy.

```
public class SoilBlockService : BaseService<SoilBlock>{
```

Jak bylo zmíněno, třída dědí základní třídu `BaseService` a jako typ určuje třídu reprezentující model odpovídající entitě v databázi pro půdní blok. Následuje implementace metody, která slouží pro vytvoření záznamu o novém půdním bloku. Půdní blok je rozdělen na počet půdních honů, který definuje uživatel, každý půdní blok však obsahuje minimálně jeden půdní hon. Záznamy, reprezentující půdní hony jsou vždy vytvořeny automaticky souběžně s vytvořením půdního bloku.

```
    public async Task<IEnumerable<PartSoilBlock>>
        PutAndGetParts(SoilBlock model, int partsCount)
    {
```

Metoda v parametru obdrží objekt, jenž odpovídá instanci třídy `SoilBlock`. Hodnoty této instance jsou následně pomocí konkrétního repositáře uloženy do databáze. Druhým parametrem je počet příslušných půdních honů, které je nutné vytvořit. Metoda po svém vykonání vrazí kolekci objektů, které obsahují data s nově vytvořenými půdními hony.

```
        var partSoilBlockService = new PartSoilBlockService();
        var idNewSoilBlock = base.Put(model).Result;
```

Aby měla metoda přístup k metodám manipulujícím s entitou půdních honů, je vytvořena instance odpovídající servisy. Následně je provedena metoda rodičovské třídy, která pomocí repositáře zajistí vytvoření záznamu v tabulce půdních bloků a vrátí id nově vytvořeného záznamu.

```
        for (int i = 0; i < partsCount; i++)
        {
```

```
            await partSoilBlockService.Put(
```

Tento forCyklus postupně provádí volání metody servisy, která zajišťuje práci s modely pro půdní hony a vytváří potřebný počet záznamů půdních honů.

```
            new PartSoilBlock
            {
                CreatedAt = DateTime.UtcNow,
                MarketingYear = DateTime.UtcNow.Year,
                SoilBlockId = idNewSoilBlock,
                Description = „Nový půdní hon“
            });
        }
```

```
        return await partSoilBlockService.
            GetBySoilBlock(idNewSoilBlock);
    }
```

```
    ...
```

Nakonec metoda vrací všechny záznamy půdních honů, které mají vazbu na nově vytvořený půdní blok pro další zpracování. Třída `SoilBlockService` dále implementuje metody, které slouží pro výpočet počtu půdních honů jednotlivých půdních bloků a metodu zajišťující mazání půdních honů v případě smazání půdního bloku.

6.4 Prezentační vrstva

Úlohou této vrstvy je zajištění interakce aplikace s uživatelem a zobrazování dat. Tato vrstva obsahuje implementované uživatelské rozhraní a prostředky, které slouží pro komunikaci s vrstvou aplikační. Prezentační vrstva pro komunikaci s vrstvou aplikační využívá třídy, zvané *controllery*. Popis implementace jednotlivých *controllerů* a komunikace mezi uživatelským rozhraním a metodami aplikační vrstvy, je v práci uveden níže.

6.4.1 Struktura prezentační vrstvy

Projekt prezentační vrstvy je na straně serveru vytvořen jako projekt ASP .Net Web Api. Na straně klienta je implementováno jako single page application pomocí frameworku AngularJS. Běžné internetové stránky se skládají z různého množství html dokumentů, které klient stahuje ze serveru. Princip single page aplikací spočívá v tom, že klient načte šablonu html stránky a obsah takto vytvořené aplikace je do této šablony dynamicky doplňován (Wojcieszyn, 2014).

6.4.1.1 Owin

Serverová část prezentační vrstvy je tvořena pomocí standardu owin. OWIN definuje standart rozhraní mezi .NET webovým serverem a webovou aplikací. Cílem tohoto standartu je oddělit server a aplikaci, takovým způsobem, že lze jednoduše zaměňovat běhové prostředí a zjednodušit vývoj aplikace. V prostředí .NET je využívána implementace OWIN rozhraní nazvaná Katana. Jde o sadu komponent umožňujících vytvoření webové aplikace odpovídající rozhraní OWIN (Ugurlu, 2013).

6.4.1.2 Owin middleware

Jednou ze základních složek OWIN standartu je existence tzv. *middlewareu*, které slouží ke komunikaci mezi aplikací a běhovým prostředím. *Middlewarey* musí být zadány v přesném pořadí, protože vytváří tzv. *pipeline*, kdy požadavek na server postupně prochází všemi *middlewarey* až k poslednímu a poté stejnou cestou zpět ke klientovy. Pomocí *middlewareů* lze nastavit základní chování aplikace jako autorizaci, routování atd. (Ugurlu, 2013).

Aplikace implementuje dva *middlewarey*. První, *ConfigureAuth*, zajišťuje autorizaci uživatele. Definuje přístupový bod pro přihlašování, získání tokenu a ohodnocení přístupových práv. Druhý, *ConfigWebApi* definuje základní nastavení API pro routování pomocí *AttributeRoute*, nastavení JSON serializátoru / deserializátoru pro výměnu dat mezi klientem a serverem a nastavení IoC kontejneru *Lightinject*, který je využit při vytváření instancí kontrolerů (Ugurlu, 2013).

6.4.1.3 WebApi controllers

Serverová část prezentační vrstvy pro komunikaci s aplikační vrstvou využívá třídy tzv. *controllery*. Klientská část aplikace pak s metodami jednotlivých *controlleru* pracuje pomocí architektury REST. Architektura REST rozhraní slouží pro zjednodušení práce s daty pomocí http požadavků a implementuje základní CRUD metody.

Jak bylo zmíněno, *controllery* slouží jako rozhraní pro aplikační vrstvu. Pro práci s jednotlivými modely entit je tak implementován *controller*, který zajišťuje zpracování dat a vyvolání potřebných metod aplikační vrstvy. Všechny implementované *controllery* dědí ze základního generického *controlleru* *baseController*. Jedná se o abstraktní třídu, která implementuje metody zajišťující CRUD operace a dědí z třídy *ApiController* (Kurtz, 2013).

```
public abstract class BaseController<TModel, TViewModel> :
    ApiController
    where TModel : BaseModel
    where TViewModel : IViewModel
    {
        private IService<TModel> _service { get; set; }

        public BaseController(IService<TModel> service)
        {
            this._service = service;
        }
    }
```

Třída *baseController* vyžaduje informace o typech, se kterými bude pracovat. Během implementace třídy, která dědí tuto základní třídu, je nutné předat informace o datových typech entit, které budou využívat implementované metody této třídy. *BaseController* obsahuje privátní proměnnou *_service*, do které je uložena instance konkrétní třídy, jejíž metody budou volány. Jak bylo zmíněno, třída implementuje metody pro základní operace. Součástí této třídy jsou také konverzní metody.

```
protected virtual IEnumerable <TViewModel> DataToModels
    (IEnumerable <TModel> models)
    {
        return models.Select(m => DataToModel(m)).ToList();
    }

protected abstract TModel ModelToData(TViewModel model);
protected abstract TViewModel DataToModel(TModel data);
```

Jak je vidět na příkladu třída ve své implementaci vyžaduje dva typy, se kterými pracuje. Prezentační vrstva aplikace nepracuje přímo s instancemi modelů, ale využívá k tomu speciální třídy zvané *viewModelely*. Tyto třídy dědí z jednotlivých modelů a navíc obsahují doplňující atributy, které ukládají data pro uživatelské rozhraní. *ViewModelely* mohou obsahovat například instanci jiného modelu. Tento systém je využit pro zpracování vazeb mezi entitami.


```

public class BusinessRecordViewModel :
    BusinessRecord, IViewModel
{
    public Vendor vendorInfo { get; set; }
    public string categoryName { get; set; }
    public string measureName { get; set; }
}

```

Příklad uvádí implementaci jednoho z `viewModelů`. Tato třída obsahuje všechny atributy základního modelu a navíc atribut, který je typu `Vendor`. Konverzní funkce `controlleru`, který zpracovává obchodní záznamy, pak do tohoto atributu uloží data odpovídajícího požadovanému dodavateli. Dále obsahuje atributy, které dle příslušného id kategorie a měrné jednotky z množin uloží konkrétní název. Výše uvedená metoda `DataToModels` slouží pro konverzi kolekce záznamů, které jsou z databáze vráceny. Projde jednotlivé záznamy a provede nad nimi metodu `DataToModel`, která slouží ke konkrétní konverzi.

```

protected override BusinessRecordViewModel DataToModel
    (BusinessRecord data) {
    var vendor = new VendorService();
    return new BusinessRecordViewModel {
        CategoryId = data.CategoryId,
        ...
        vendorInfo = vendor.GetById(data.VendorId).Result,
        categoryName=Enum.GetName(typeof(CategoryType), data.CategoryId),

```

Příklad uvádí část kódu konkrétní konverzní metody. Tato metoda slouží ke konverzi dat vrácených z aplikační vrstvy do konkrétního `viewModelu`. Metoda vytvoří instanci třídy, která zpracovává informace o dodavatelích. Následně metoda vrací instanci `viewModelu`, do které jsou uloženy data příslušného datového modelu. Pro získání informací je využita metoda `GetById`, kterou implementuje základní třída aplikační vrstvy.

Základní třída `base model` implementuje metody pro CRUD operace. Díky konverzním metodám, které implementují všechny konkrétní `controllery` je tak možné využít implementaci CRUD metod v `baseControlleru`. Jako příklad je zde uvedena implementace metody, která zpracuje požadavek o vrácení konkrétního záznamu z databáze.

```

[Route("{id:long}"), HttpGet]
public virtual async Task<TViewModel> Get(long id)
{
    var model = DataToModel(_service.GetById(id).Result);
    return await Task.FromResult<TViewModel>(model);
}

```

Tato metoda dodržuje routování dle architektury REST. Jako parametr vyžaduje id konkrétního záznamu. Její návratový typ je konkrétní `viewModel` a využívá metody `GetById`, která je implementována v aplikační vrstvě. Data, které jsou vráceny touto metodou, jsou předány jako parametr konverzní metodě, a následně jsou vráceny jako návratový typ metody `get`.

Pro práci s jednotlivými daty, v relační databázi je pro každý model implementován vlastní controller. Tyto třídy dědí `baseController` a tím je zajištěna základní manipulace se všemi daty. Controllery implementují metody pro konverzi typů a vlastní metody, které obstarávají vykonání jiných než CRUD operací. Jako příklad je zde uvedena část implementace controlleru, který slouží pro práci s entitou `BusinessRecord`.

```
[Authorize]
[RoutePrefix("businessRecord")]
public class BusinessRecordController
: BaseController<BusinessRecord, BusinessRecordViewModel>{
public BusinessRecordService Service { get; set; }
public BusinessRecordController(BusinessRecordService service)
: base(service)
{Service = service;}
[Route("record/{type}"), HttpGet]
public async Task<IEnumerable< BusinessRecordViewModel >>
GetAllByType(bool type)
{return await DataToModels.Service.GetAllByType(type,
IdentityHelper.CurrentUser.CompanyId);}
```

Atribut `Authorize` zpřístupňuje metody tohoto controlleru pouze ověřenému uživateli. Následuje nastavení routingu, pomocí kterého jdou jednotlivé metody volané. Třída dědí z `baseControlleru`, kterému předává typ entity a typ `viewModelu`, se kterým bude pracovat. Všechny controllery obsahují proměnnou `Service`, která je typu příslušné třídy, jenž zprostředkovává rozhraní s aplikační vrstvou. Konstruktory controlleru v parametru tuto proměnnou předává rodiči a tím zajistí dostupnost konkrétních CRUD operací nad danou entitou. Instance dané služby je do proměnné doplněna pomocí `lightInject`.

V ukázce je také uvedena implementace metody, která slouží pro získání záznamů o všech pohybech na skladu v závislosti na typu. Záznamy jsou rozděleny na příjem a výdej ze skladu. Metoda v parametrech očekává hodnoty typu požadované položky, a identifikační číslo uživatele, který tento výpis žádá. V aplikaci je implementovaná třída `IdentityHelper`. Tato třída poskytuje informace o právě přihlášeném uživateli.

```
public static class IdentityHelper
{
public static User CurrentUser
{get{return GetCurrentUser();}}
}
private static User GetCurrentUser()
{
var user = new User();
var identity =
Thread.CurrentPrincipal.Identity as ClaimsIdentity;
var claims =
identity.Claims.ToDictionary(c => c.Type, c => c.Value);
user.CompanyId = long.Parse(claims["companyId"]);
user.Id = long.Parse(claims["userId"]);
user.Role = (UserRole)int.Parse(claims["role"]);
return user;
}
```

Jedná se o statickou třídu, která v případě vyžádání načte informace o uživateli uložené v tzv. `claimu` této aplikace. Informace a přihlášeném uživateli se do `claimu` ukládají pomocí třídy `OAuthProvider`. Princip autorizace uživatelů je v práci popsán níže. Statická třída `IdentityHelper` v implementované metodě `GetCurrentUser()` vytváří instanci `claimu`, jednotlivé informace jsou pomocí datové struktury `Dictionary` uloženy pro další zpracování. Metoda následně vytvoří instanci třídy `User` pro uložení potřebných dat o aktuálním uživateli. Tímto způsobem statická třída `IdentityHelper` poskytuje jednotlivým controllerům informace o přihlášeném uživateli.

Na výše uvedené metodě `GetAllByType(bool type)` bude popsán postup, jakým způsobem v aplikaci probíhá zpracování požadavku klienta. Implementaci uživatelského rozhraní se věnuje samostatná kapitola. Uživatelské rozhraní je implementováno pomocí frameworku AngularJS. Jednotlivé metody serverových controllerů prezentační vrstvy jsou spouštěny pomocí modulu `RestAngular`, který zajišťuje korektní dodržení architektury REST. V uživatelském rozhraní jsou implementovány servery pro práci s jednotlivými controllery. Následující ukázka představuje implementaci funkce, která je určena pro spuštění metody web api controlleru. Detailní popis uživatelského rozhraní je popsán níže.

```
this.getAllByType = function (type, success, error) {
    Restangular.all('businessRecord')
        .one("record", type)
        .get(type).then(success, error);
};
```

Jedná se o javascriptovou funkci uživatelského rozhraní. Modul `RestAngular` sestaví požadované URI, které je odesláno na server. V tomto případě bude mít podobu `/businessRecord/record/type`. Na základě takto vyvolaného požadavku se spustí metoda web api controlleru, která v parametru získá hodnotu požadovaného typu. V proměnné `Service` je vytvořena instance příslušné třídy aplikační vrstvy, v tomto případě třídy `BusinessRecordService`. Metoda controlleru vrací výsledek metody `Service.GetAllByType`. Této metodě je předán v parametru `typ` požadovaných záznamů a pomocí zmíněné třídy `IdentityHelper` id společnosti přihlášeného uživatele. Jak bylo zmíněno výše, třída `BusinessRecordService` dědí ze základní třídy `BaseService` a předává jí typ modelu, se kterým třída pracuje. Tato třída implementuje požadovanou metodu `GetAllByType`.

```
public async Task<IEnumerable<BusinessRecord>>
    GetAllByType(bool type, long companyId)
    {
        return await Repository.GetAllByType(type, companyId);
    }
```

Tato metoda vrací kolekci instancí modelu `BusinessRecord`. Metoda aplikační vrstvy tyto instance obdrží jako výsledek extenzní metody, kterou implementuje příslušný repositář datové vrstvy. Metoda repositáře zajistí operaci nad daty uloženými v databázi pomocí SQL dotazu `"SELECT * FROM [BusinessRecord] WHERE Income = @type AND CompanyId = @companyId"`. Princip zpracování tohoto SQL dotazu a mapování jeho výsledků na jednotlivé instance modelů je

popsán v kapitole 5.3.2.3. Metoda controlleru touto cestou obdrží záznamy z databáze v podobě kolekce instancí příslušného modelu, nad nimiž zavolá konverzní metodu, která data převede do požadovaného `viewModelu`. Tyto data jsou následně vráceny javascriptové funkci, která provedení metody controlleru vyžádala a jsou dále zpracovány v uživatelském rozhraní. Tímto způsobem probíhá výměna a manipulace s daty mezi jednotlivými vrstvami aplikace. Díky vícevrstvé architektuře a dodržení implementovaného rozhraní je tak možné měnit jednotlivé metody, které s daty pracují.

6.4.1.4 Autentizace

Aplikace pro autentizaci uživatelů využívá protokol OAuth2, který poskytuje možnost autorizace a získávání informací o uživateli pomocí claimů. Uživatel je v aplikaci autentizován pomocí přihlašovacího jména a hesla a poté je pro uživatele vygenerován unikátní token, mající podobu hashe. Do tokenu je možné zakódovat libovolné informace o uživateli, včetně rolí nebo práv. Protokol OAuth2 také umožňuje přihlašování pomocí aplikací třetích stran, takže je možné aplikaci rozšířit a pro přihlašování využít služby jako je Facebook nebo Google. Každý token má také nastavenou dobu platnosti, ale je možné na základě již zastaralého tokenu získat token nový. Přístupová práva jsou řešena na úrovni OWIN middlewaru k tomu určenému a jsou kontrolována v kontrolerech web api pomocí atributu `Authorize`.

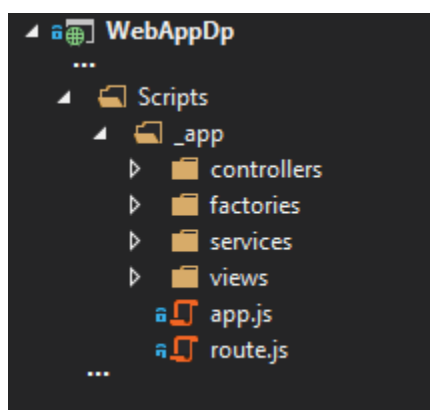
6.4.1.5 Spuštění aplikace

Při spuštění aplikace dojde k nastartování Web API a klienta prezentační vrstvy. Spuštění klienta je řešeno pomocí .NET MVC 5, který obsahuje pouze základní stránku s vloženými kaskádovými styly a javascriptem. Poté už veškerou komunikaci mezi uživatelem a serverem přebírá javascriptový client, který komunikuje se serverem pomocí Web API a další interakce s MVC již není potřeba.

6.4.2 Implementace uživatelského rozhraní

Použitý Framework AngularJS rozděluje kód aplikace do přehledných celků. Tato část práce je zaměřena na popis implementace jednotlivých prvků uživatelského rozhraní, které je zpracováváno pomocí jednotlivých modulů angularu. Výměnu dat se serverem a autentizaci uživatelů zajišťují další prvky prezentační vrstvy, se kterými bude čtenář v práci dále seznámen.

Následující obrázek zjednodušeně zobrazuje strukturu webové aplikace, jenž je implementována frameworkem AngularJS. V práci bude struktura dále vysvětlena.



Obrázek 9: Souborová struktura části prezentační vrstvy

Prezentační vrstva této aplikace obsahuje složku scripts, která je určena pro ukládání javascriptových souborů. Klientská část tohoto systému je následně uložena ve vlastní složce _app, která je dále rozdělena dle jednotlivých modulů, které AngularJS využívá. Soubor app.js je zaváděcím souborem javascriptové aplikace. Složka controllers slouží pro ukládání zdrojového kódu jednotlivých controllerů. Dále jsou moduly aplikace rozděleny na factories a services, jednotlivé service zajišťují komunikaci s controllery prezentační vrstvy, které slouží pro výměnu dat a vyvolání metod aplikační vrstvy. Ke komunikaci je použit restAngular. Jedná se o službu, která umožňuje jednoduše využívat architekturu REST pro komunikaci s controllery web api. Jednotlivé šablony, které jsou dynamicky doplňovány do webové stránky, jsou uloženy ve složce views. Příklady implementace vybraných view, service a controllerů budou v této práci uvedeny níže. Hlavní složka _app dále obsahuje dva soubory, ve kterých je uloženo základní nastavení klientské aplikace. Jak bylo uvedeno, AngularJS rozděluje aplikaci na velký počet modulů, jejichž využívání je zajištěno pomocí dependency injection. Pro spuštění aplikace slouží direktiva ng-app, kde je v uvozovkách uveden název aplikace. V souboru app.js je zavedena proměnná, která slouží k propojení modulů celé aplikace.

```
var dpApp = angular.module('dpApp', [
    'ngRoute',
    'ngSanitize',
    'LocalStorageModule',
    'ngCookies',
    'ui.bootstrap',
    'ui.bootstrap.tpls',
    'restangular',
    'ngSlider',
    'smart-table',
    'ui.sortable',
    'angular-svg-round-progress'
]);
```

Uvedený kód představuje základní nastavení aplikace, které zavádí jednotlivé moduly. Modul ngRoute umožňuje dynamické vkládání jednotlivých šablon na základě url. Tento modul se dále nastavuje v posledním souboru uloženém ve

složce `_app`. Konkrétní nastavení jednotlivých route bude popsáno níže. Následují moduly, které v aplikaci dovolí využívat určité úložiště. Další moduly, které aplikace využívá, slouží zavedení speciálních direktiv, které AngularJS podporuje. Jejich použití bude popsáno níže. V tomto souboru je také implementován modul, který spustí definované funkce po startu aplikace. Jedná se například o ověření identity návštěvníka.

```
$rootScope.$on('$locationChangeStart',
function (event, next, current) {
    var isInApp = next.indexOf(ngAuthSettings.baseUrl+'/#/app')==0
    $rootScope.isInApp = isInApp;

    if (!authService.authentication.isAuth && isInApp) {
        $location.path('/unauthorized');
        event.preventDefault();
    }
});
```

Tato funkce je registrována do globálního modelu `rootScope` a je automaticky spuštěna po startu aplikace. Aplikace je rozdělena na dvě části, veřejnou a soukromou. Url adresa soukromé části vždy začíná prefixem `#!/app`. Funkce nejprve zjistí na jakou url se chce uživatel dostat, následně výsledek uloží do proměnné `isInApp` globálního modelu `rootScope`. Dále se provede podmínka, která pomocí autorizační služby vyhodnotí, zda se jedná o přihlášeného uživatele. V případě, že ne, je pomocí modulu `$location` a jeho funkce `path` přesměrován na stránku, která ho informuje, že pro tuto část aplikace je třeba být přihlášen.

Uživatelské rozhraní je sestaveno z jednotlivých view, které jsou na základě url adresy dynamicky vkládány do základního layoutu. Nastavení vkládání jednotlivých šablon je implementováno v souboru `route.js`.

```
dpApp.config(function ($routeProvider) {
    $routeProvider.when("/unauthorized", {
        templateUrl: "/Scripts/_app/views/unauthorized.html"
    });
    $routeProvider.when("/", {
        controller: "initController",
        templateUrl: "/Scripts/_app/views/Home.html"
    });
    $routeProvider.when("/registration", {
        templateUrl: "/Scripts/_app/views/Registration.html"
    });
});
```

Nastavení zobrazení konkrétních šablon je implementováno pomocí modulu `routeProvider`, který na základě url adresy vkládá konkrétní šablonu.

Základní stránka uživatelského rozhraní nabízí základní informace o aplikaci, možnost registrace nového klienta a přístupový bod aplikace přihlašovacím formulářem. Registrace obsahuje formuláře, které slouží uživateli pro vyplnění informací o společnosti, osobní informace a kontaktu.

Obrázek 10: Ukázka formuláře pro registraci

Šablona pro registrační formulář je implementována pomocí elementu tabset, která rozděluje jednotlivé formuláře do vlastních záložek. Uživatel tak vidí celý průběh registrace. Před dokončením registrace se uživateli zobrazí zadaná data pro kontrolu. Zpracování registrace je implementováno v modulu `registrationController`. Komunikaci mezi view a controllerem aktivuje direktiva `ng-controller` s názvem příslušného controlleru uvedená v hlavním elementu `div` této šablony.

```
<div ng-controller="registrationController">
  <tabset justified="true" ng-hide="registrationLoad">
    <tab active="currentPage == 1"
        disabled="currentPage != 1" heading="Společnost"
        class="text-center">

      <div class="row">
        <div class="panel-body col-md-4 col-md-offset-4">
          <form role="form" name="formCompany">
            <div class="form-group">
              <label for="spolName">Název</label>
              <input id="spolName" name="nameCompany" type="text"
                class="form-control" placeholder="Název společnosti"
                required ng-model="Company.Name">

              <div class="error-container"
                ng-show="formCompany.nameCompany.$dirty &&
                  formCompany.nameCompany.$invalid">
                <small class="errorMessage"
                  ng-show="formCompany.nameCompany.$error.required">
                  Zadejte prosím název společnosti</small>
              </div>
            </div>
          </form>
        </div>
      </div>
    </tab>
  </tabset>
</div>
```

Průběh registrace uživatel řídí pomocí tlačítek ve spodní části obrazovky. Uvedená ukázka kódu zobrazuje implementaci šablony pro registraci nového uživatele. Jak

bylo zmíněno, element `tabset` je použit pro rozdělení registrace do jednotlivých kroků. Použitá direktiva `ng-hide` slouží ke skrytí bloku registrace v průběhu ukládání nového uživatele do databáze. Ve chvíli, kdy má proměnná uvedená v apostrofech hodnotu `true` je element, který jako atribut obsahuje tuto direktivu skryt. Element `tabset` obsahuje jednotlivé záložky, které jsou implementovány pomocí tagu `tab`. Atributy tohoto tagu využívají proměnnou `currentPage`, která určuje, ve kterém kroku registrace se uživatel nachází. Uživatelské rozhraní této aplikace je implementováno pomocí knihovny `bootstrap`, která slouží pro vytvoření responsivního designu aplikace. Dále je v ukázce možno vidět využití direktivy `ng-model`. Ta obsahuje proměnnou, do které bude uložen vstup zadaný uživatelem. Formulář také obsahuje element, který slouží pro validaci uživatelských vstupů. V případě, že uživatel zadá neplatný vstup, se pomocí direktivy `ng-show` zobrazí chybové hlášení. Tato direktiva pracuje obdobně jako direktiva `ng-hide`. Validace uživatelských vstupů je v `angularu` velice jednoduchá, jednotlivé vstupy mají vlastní proměnné, které ukládají informaci o validitě použitých formulářů. Tento způsob validace je použit pro všechny vstupy uživatelského rozhraní.

Informace, které uživatel zadává v průběhu registrace, jsou uloženy do příslušných proměnných pomocí direktivy `ng-model`. Níže je uvedena ukázka implementace `controlleru`, který řídí registraci.

```
dpApp.controller('registrationController',
function ($scope, $rootScope, $location, $timeout, regService) {
    $scope.registrationLoad = false;
    $scope.registrationOk = false;

    $scope.totalItems = 4;
    $scope.currentPage = 1;

    $scope.registrationFunction = function () {
        $scope.registrationLoad = true;
        regService.registrationComplete($scope.Company,
                                        $scope.User,
                                        $scope.Contact,
                                        $scope.Address,

        function () {
            $scope.registrationOk = true;
            $timeout(function () {
                $location.path('/login');
                $scope.registrationLoad = false;
            }, 1500);
        });
    };
});
```

Jak je patrné z ukázky, do aplikace `dpApp` je přidán `controller` s názvem `registrationController`. Kód tohoto `controlleru` je spuštěn ve chvíli, kdy je načtena šablona obsahující direktivu `ng-controller` s jeho názvem. `Controllery` v `Angularu` jsou funkce, které v parametru očekávají názvy modulů, které jsou pomocí `dependency injection` zavedeny. Následně tento `controller` inicializuje proměnné, které jsou použity pro řízení vizualizace částí uživatelského rozhraní. Šablona tyto první dvě proměnné využívá jako hodnoty pro direktivy `ng-show`

a `ng-hide`. Proměnné `totalItems` a `currentPage` ukládají hodnoty, které jsou použity pro ovládání průběhu registrace. V tomto controlleru je implementována funkce, která slouží pro uložení zadaných dat do databáze. Aby mohla být tato funkce aktivována v šabloně, je pomocí tečkové notace přidána do modelu scope. Takto definované funkce controllerů mohou být následně používány v jednotlivých šablonách. Vyvolání funkce je zajištěno pomocí direktivy `ng-click`, která je zavedena jako atribut html elementu. Pokud html element obsahuje atribut `ng-click`, po kliknutí na tento element je vyvolána funkce uvedená v této direktivě.

```
<button
  class="btn btn-lg btn-primary btn-block"
  ng-click="registrationFunction()"
  ng-disabled=
    "formCompany.$invalid||
    formUserReg.$invalid||formContactReg.$invalid">
  Dokončit registraci
</button>
```

Příklad uvádí implementaci tlačítka pro dokončení registrace, které pomocí direktivy `ng-click` vyvolává funkci, která je implementována v `registrationController`. Element `button` také využívá direktivu `ng-disabled`, pomocí které je klik na tlačítko zakázán v případě, že je některý z registračních formulářů špatně vyplněn.

Funkce `registrationFunction` předává data servise, která pomocí modulu `RestAngular` vyvolává metodu příslušného web api controlleru. Díky použitému principu two-way data binding, jsou data zadaná uživatelem v objektech modelu scope automaticky. Princip jakým funkce servisy vykoná uložení nového uživatele do databáze je popsán v kapitole 55.1.4. Všechny funkce jednotlivých service v parametru očekávají další dvě funkce. Jedná se o tzv. callback funkce. První se vykoná v případě úspěšného provedení požadavku a slouží pro další zpracování dat. Druhá funkce je vyvolána v případě, že se vyskytne nějaká neočekávaná chyba. Po správném vykonání funkce v příkladu je aktivována první funkce, která využívá modul `$timeout` po uplynutí 1500ms přesměruje uživatele na přihlašovací stránku aplikace.

Uživatelské rozhraní v autorizované části aplikace se dělí na dvě hlavní části. V levé části obrazovky se nachází hlavní menu aplikace. Pomocí tohoto menu se uživatel pohybuje mezi hlavními částmi informačního systému. Jedná se o správu půdních bloků, osevních plánů, skladu, techniky a financí. Úvodní obrazovka tohoto systému uživateli zobrazuje nejdůležitější informace, které informační systém uchovává. Uživateli se zobrazují informace o stavu zásob na skladě, výměře obhospodařované půdy a statistické údaje o pěstovaných plodinách. K dispozici má také uživatel seznam aktivních osevních plánů. Osevní plány uživatel vytváří pro jednotlivé půdní hony. Uživatel má k dispozici možnost zvolit hospodářský rok, pro který chce informace zobrazit. Součástí hlavního menu je i možnost upravovat osobní informace v profilu uživatele a zobrazit nastavení.

Celkový přehled pro hospodářský rok 2016/2017

Plán práce
03.01. 2016 - 03.02. 2016
Tento měsíc je v plánu následující činnost

Datum	Název bloku	Název honu	Výměra	Stroj	Práce
03.01. 2016	U elektrárny	U elektrárny	26 ha	John Deere 8430	Orba
02.02. 2016	U elektrárny	U elektrárny	26 ha	John Deere 8430	Setí
19.01. 2016	U elektrárny	U elektrárny	26 ha	John Deere 8430	Hnojení

Splatnost faktur
03.01. 2016 - 03.02. 2016
Blíží se splatnost následujících faktur

Typ	Číslo faktury	Datum splatnosti	Firma 1	Firma 2	Celková cena
+	1	20.01. 2016	Lovochemie, a. s.	Vlastní výroba	12.800,00 Kč

Základní informace
Obhospodařovaná výměra: 99 ha
Počet pudních bloků: 3
Počet pudních honů: 4

Pěstované plodiny
Kukuřice setá: 33 ha
Pšenice ozimá: 26 ha

Skladový alarm

- GARDOPRIM PLUS GOLD 500 SC: Aktuálně je naskladněno: 30 L. Pro splnění všech osevních plánů chybí: 29 L.
- Nurelle D: 29 L
- Nafta: 2179 L

© 2016 - Diplomová práce

Obrázek 11: Hlavní obrazovka aplikace

Část systému, která je dostupná tlačítkem „Technika“ slouží pro spravování strojů, které má uživatel k dispozici. Uživatel zde má přehled jednotlivých strojů a kalendář naplánované práce pro jednotlivé stroje. Každý stroj má také k dispozici knihu údržby, pomocí které může uživatel evidovat například záznam o výměně oleje popřípadě jiné závady.

S technickou částí systému také souvisí evidence prováděné práce. Uživatel zde definuje úkoly, které jsou nutné pro plnění uživatelských osevních plánů. Práce definované uživatelem ukládají informace o nákladech prováděné práce na hektar a pro výpočet předpokládané spotřeby strojů uživatel zadává předpokládanou spotřebu stroje při provádění této práce.

Jako jeden ze způsobů interakce s uživatelem aplikace využívá systému modálních dialogů. Tyto dialogy jsou použity pro vkládání záznamů s omezeným počtem hodnot. Modální dialogy v AngularJS jsou implementovány pomocí komponenty modal a ke své činnosti používají direktivu ng-controller. Díky tomu tak mají dialogy vlastní paměť scope a s daty mohou pracovat nezávisle. Pro každý dialog je implementován vlastní controller, který řídí chování jednotlivých oken. Šablony těchto dialogů jsou implementovány v samostatném souboru a odděleny tak od zbytku aplikace. Tyto šablony jsou samostatné html soubory, které jsou

implementovány jazykem html a využívají direktivy frameworku angularJS. Šablony pro modální dialogy mají danou strukturu. Jsou rozděleny pomocí elementu div na tři bloky, pro hlavičku, tělo a patičku dialogu. V příloze (I.) je uvedena ukázka implementace šablony dialogu. Implementace controlleru pro dialogy je totožná jako implementace aplikačních controllerů, stejně tak jako práce s daty a vnitřními funkcemi. Zobrazení modálního dialogu je podmíněno akcí uživatele. Funkce, které reagují na jednotlivé akce, jsou implementovány v příslušných controllerech s využitím modulu \$modal.

```
$scope.showNewWorkCategoryModal = function () {
    var modalInstance = $modal.open({
        templateUrl:
            '/Scripts/_app/views/modals/newWorkCategoryModal.html',
        controller: 'newWorkModalController',
        size: 'sm'
    });
    modalInstance.result.then(function () {
        $scope.loadWork();
    });
}
```

Výše uvedená ukázka kódu představuje implementaci funkce controlleru, která slouží k otevření modálního dialogu. Jak bylo zmíněno, pomocí dependency injection je připojen modul \$modal. Funkce zavádí proměnnou, do které je vytvořena instance modálního dialogu. Pomocí metody open je tento modální dialog otevřen. Metoda v parametrech očekává nastavení dialogového okna. První atribut určí, která šablona se pro tento konkrétní dialog použije. Dále je nastaven controller, ve kterém jsou implementovány patřičné funkce pro data modálního dialogu. Atribut size určuje velikost. Může být také využito atributu resolve, pomocí kterého můžou být controlleru dialogu předána určitá data. V případě potřeby je možné implementovat callback funkce, které se vyvolají po ukončení dialogu. V tomto případě se po uložení nové práce do databáze vyvolá funkce pro nové načtení seznamu prací.

Obrázek 12: Příklad dialogového okna

Na obrázku 12 lze vidět příklad vykresleného dialogu, jehož implementace je uvedena v příloze (I). Mimo jiné je pro implementaci tohoto dialogu využita i direktiva `ng-repeat`. Tato direktiva slouží pro automatické opakování elementů html stránky. Funguje podobně jako programová struktura `forEach`.

```
<tr ng-repeat="item in machineryList">
```

Postupně vykresluje element řádku tabulky pro každý prvek v poli objektů `machineryList`. Data týkající se jednotlivých strojů jsou pak v šabloně uložena v proměnné `item`. Dále se s daty pracuje pomocí tečkové notace a díky two-way data bindingu může být měněna hodnota přímo dat v počátečním poli objektů.

Skladová část informačního systému slouží uživateli pro správu reálného množství materiálu, které uchovává na svém skladě. Položky skladu jsou rozděleny do několika sekcí dle požadavku zadavatele. V této části má uživatel možnost přehledně procházet jednotlivé části skladu, které jsou implementovány pomocí elementu `tbody` do jednotlivých záložek. V této části systému uživatel eviduje seznam plodin, které využívá pro svoji rostlinnou výrobu. Také má možnost založení nové skladové položky, přikoupení stávajícího materiálu, nebo také uskladněný materiál prodat, včetně sklizených plodin. Ve všech případech se do databáze ukládá příslušný záznam o pohybu na skladě. Při zakládání nové skladové položky uživatel určuje, pro kterou plodinu je konkrétní přípravek určen. To je dále využito při tvorbě osevního plánu. Informační systém informuje uživatele, o celkovém množství a hodnotě materiálu dle jednotlivých kategorií. V případě prodeje materiálu je uživatel informován o hodnotě jednotky uskladněného materiálu. Této funkcionality využije uživatel hlavně v případě prodeje vypěstované plodiny, kdy jsou náklady na jednotky počítány s ohledem na náklady vynaložené na provedení osevního plánu. Jednotlivé záznamy o pohybu na skladě jsou dostupné v části Finance informačního systému. Zde uživatel eviduje dodavatele, se kterými spolupracuje a na základě pohybů na skladě může vystavovat faktury, které systém dále uchovává.

Pro správu půdních bloků uživatele je v systému určena samostatná část. V této části uživatel spravuje svoje půdní bloky. Jak bylo zmíněno při návrhu, uživatel může půdní blok rozdělit na libovolný počet půdních honů. Půdní hony uživatel zakládá pro každý hospodářský rok, který chce evidovat v informačním systému. Pro tyto půdní hony uživatel vytváří jednotlivé osevní plány. Informační systém uživatele v části systému pro správu půdních bloků informuje o aktuálním stavu osevních plánů zvoleného půdního bloku.

Nejdůležitější částí systému je samotná správa osevních plánů. Na základě vybraného osevního plánu je zde uživateli zobrazen seznam vytvořených osevních plánů a důležité informace, které se jich týkají. V informačním systému je implementován systém automatického varování nedostatečného množství materiálu na skladě. Systém automaticky vyhodnocuje předpokládanou spotřebu materiálu pro jednotlivé osevní plány a v případě nedostatku materiálu na skladě uživatele varuje.

Při zakládání nového osevního plánu uživatel postupně vybírá půdní blok, půdní hon a plodinu, kterou chce na daném půdním honu pěstovat. Osevní plán pak uživatel dále sestavuje přidáváním jednotlivých položek, které představují jednotlivé operace na půdním honu. Seznam přidávaných položek je implementován pomocí elementu `accordion`.

Obrázek 13: Tvorba osevního plánu

Tato část systému uživateli poskytuje informace o předpokládaných nákladech na provedení vytvořeného osevního plánu. Jednotlivé položky elementu `accordion` jsou vytvářeny dynamicky pomocí direktivy `ng-repeat`. V šabloně pro vytváření osevních plánů je implementována jedna položka, která je pomocí této direktivy opakována v závislosti na počtu přidávaných prvků do interního pole. Toto pole je uloženo v modelu `controlleru`, který implementuje všechny potřebné funkce pro vytváření osevních plánů. Controller po svém startu nejdříve pomocí `servisy` načte seznam půdních bloků, který je uložen do proměnné `selectSoilBlock` jako pole objektů. Následně je v šablona pomocí této proměnné naplní rozbalovací seznam pro výběr půdního bloku.

```
<select class="form-control"
  ng-model="selectSoilBlockCRP"
  ng-change="changeSoilBlock(selectSoilBlockCRP.id)"
  ng-options="obj as (obj.name) for obj in selectSoilBlock"
  popover="text" popover-trigger="mouseenter">
  <option disabled value="">Půdní bloky</option>
</select>
```

Jednotlivé položky seznamu jsou pomocí direktivy `ng-options` vygenerovány dle obsahu proměnné `selectSoilBlock`, aktuální hodnota vybrané položky je uložena v proměnné direktivy `ng-model`. Stejným způsobem jsou implementovány všechny entity `select` této aplikace.

Controller pro vytváření nových osevních plánů definuje objekt `CrpItem`, který slouží pro uložení důležitých informací pro jednotlivé položky osevního plánu. Jsou to pole pro výběr materiálu, práce, techniky a pohonných hmot, které se načtou po přidání položky do osevního plánu. Dále pak objekt uchovává informace o zvolených hodnotách, na základě kterých se provádí výpočet předpokládaných nákladů.

Obrázek 14: Editace položky osevního plánu

V průběhu vytváření osevního plánu controller ukládá objekty `CrpItem` do řídicího pole, pomocí kterého jsou generovány položky seznamu `accordion` představujícího osevní plán. Uživatel pak nastavuje hodnoty položky, kterou vložil do osevního plánu a systém na základě rozlohy půdního honu vypočítává předpokládané náklady na provedení dané položky. Uživatel tak hned vidí, kolik bude potřebovat materiálu, předpokládanou spotřebu stroje a stav naskladněného materiálu. V případě nedostatku je uživatel varován. Pokud se jedná o položku, která po použití mění obsah látek v půdě je v závislosti na zvolené dávce zobrazena i kalkulace těchto látek. Direktiva `ng-repeat` pomocí filtru uvedeného za svislou čarou přidávané položky v seznamu `accordion` automaticky řadí.

```
<accordion-group is-open="item.openStatus"
  ng-repeat="item in cropRotationItem| orderBy:'date'">
```

Po sestavení osevního plánu je uživateli pro kontrolu vytvořený plán zobrazen v přehledné tabulce. Tabulka zobrazuje přehled veškerých naplánovaných operací a předpokládaných nákladů. Po uložení osevního plánu má uživatel k dispozici kalendář naplánovaných prací pro jednotlivé stroje a úkony spojené s jednotlivými půdními hony.

Po otevření detailu osevního plánu má uživatel možnost označovat jednotlivé body osevního plánu za splněné. Ve chvíli, kdy uživatel označuje bod osevního plánu za splněný, se otevře dialogové okno, které slouží pro uložení skutečných nákladů na provedení dané položky. V tomto okně jsou uživateli zobrazeny všechny informace týkající se dané položky. Uživatel zde zadá hodnoty, které odpovídají reálné spotřebě materiálu, nafty a také skutečné ploše na kterou byla práce nebo materiál použit. Tuto možnost systém nabízí pro případ, že z nějakého důvodu nemohla být práce provedena na celou plochu půdního honu. Tyto informace jsou následně uloženy do systému, je uložen příslušný záznam o pohybu na skladě a opotřebení stroje.

Po splnění všech položek osevního plánu uživatel zadává informace o množství sklizené plodiny. Na základě reálných nákladů na osevní plán jsou vypočítány náklady na kg sklizené plodiny. Jednotlivé plodiny mají definováno, kolik živin z půdy odebere tuna sklizené plodiny. Půdní bloky ukládají informace o množství živin v půdě. Uživatel tak během tvorby osevního plánu plánuje kolik živin je za hospodářský rok do půdy přidáno prostřednictvím použitých hnojiv. Po uložení sklizně, je vypočítané množství z půdy zase odebráno. Množství sklizené plodiny se automaticky uloží do kategorie „Sklizeň“ ve skladu systému a uživatel zde může řídit prodej plodin, kde jsou k dispozici informace o uskladnění množství plodiny a nákladech na její vypěstování.

7 Diskuze

Informační systém byl navržen tak, aby poskytoval uživateli dostatečnou funkcionalitu pro usnadnění činnosti drobných zemědělců věnujících se rostlinné výrobě. Systém nabízí evidenci půdních bloků, pěstovaných plodin, využívané techniky, skladové činnosti a osevních plánů.

Registrovaný uživatel po přihlášení do systému zaeviduje základní informace, jako je seznam pěstovaných plodin, využívaná technika, prováděná polní činnost a průměrná spotřeba strojů při vykonávání této činnosti. Skladová část informačního systému umožňuje ukládat informace o naskladněném materiálu rozděleném do příslušných kategorií. Skladové položky ukládají údaje, které byly požadovány zadavatelem. Dále informační systém dovoluje uživateli spravovat jednotlivé půdní bloky, které mohou být rozděleny na libovolný počet půdních honů. Informační systém podporuje sestavování osevních plánů pro jednotlivé půdní hony. V této části systému uživatel jednoduše vytvoří seznam položek, ze kterých je sestaven osevní plán. Položky se dělí do čtyř typů osiva, hnojiva, přípravky na ochranu rostlin a samostatná práce. Po přidání některé z položek uživatel vybere materiál, typ práce a techniku, která bude použita. Systém informuje o předpokládaných nákladech a množství potřebného materiálu. Na základě těchto informací má uživatel přehled výdajů. Systém automaticky ukládá informace o pohybu na skladě při vložení položek, jejich prodeji, nebo při splnění jednotlivých bodů osevního plánu. Dále systém informuje uživatele v případě nedostatku materiálu pro splnění osevního plánu. Uživatel po splnění položek osevního plánu ukládá informace o množství sklizené plodiny. Tyto informace jsou uloženy ve skladové části systému, zde je možné provádět prodej. Na základě informací, které systém eviduje, má uživatel přehled o výnosech jednotlivých půdních honů a nákladech na vypěstované plodiny.

Implementovaný informační systém byl předán k testování zadavateli, který ověřuje funkcionalitu jednotlivých částí tohoto systému. Zadavatel je s dodaným informačním systémem spokojený. Na základě testování lze zhodnotit, že implementovaný informační systém splnil očekávání zadavatele. Systém poskytuje možnost uchovávání potřebných dat pro činnost drobného zemědělce. Možným rozšířením tohoto systému je například správa údržby jednotlivých strojů, popřípadě rozšířit kategorie skladu dle potřeb jednotlivých uživatelů. Dalším vhodným rozšířením by bylo propojit systém s registrem půdy a následně implementovat automatickou správu půdních bloků.

8 Závěr

Cílem diplomové práce bylo vytvořit informační systém pro podporu rostlinné výroby na základě požadavků zadavatele. Tento systém je určen pro usnadnění vytváření osevních plánů, evidenci skladu a evidenci polní činnosti. Po prvotních konzultacích se zadavatelem byl vytvořen seznam požadavků na funkcionalitu tohoto systému, uvedených v kapitole 5.1. V průběhu analýzy byl vytvořen model informačního systému, který byl pravidelně konzultován. Tento informační systém byl navržen jako webová aplikace, která je dostupná na adrese zemedelec.azurewebsites.net. Pro přístup do systému je se přihlásit, byl vytvořen testovací účet Demo s heslem demodemo.

V průběhu zpracování diplomové práce byl kladen důraz na důkladné zpracování požadavků zadavatele, aby navrhovaný informační systém vyhovoval širokému spektru drobných zemědělců a poskytoval potřebnou funkcionalitu pro podporu jejich činnosti. Informační systém je navržený s ohledem na skupinu cílových uživatelů tak, aby jeho využívání bylo co možná nejvíce intuitivní.

Pro implementaci informačního systému bylo nutné nastudovat použití moderních technologií určených pro vývoj webových aplikací. Pro implementaci uživatelského rozhraní byl zvolen framework AngularJS, který se ukázal jako velice užitečný pro implementaci klientských aplikací. Zvolená vícevrstvá architektura softwaru umožňuje tento informační systém dále jednoduše rozvíjet na základě připomínek, které vzejdou v průběhu testování a nasazení tohoto systému do ostrého provozu.

Stávající informační systémy, zaměřené na zemědělskou výrobu jsou převážně určeny pro využití velkými společnostmi a jejich použití je tak pro drobné zemědělce, věnující se rostlinné výrobě poměrně složité. Jak bylo zmíněno, byl návrh a vývoj informačního systému zaměřen právě na potřeby drobných zemědělců. Podařilo se implementovat systém s potřebnou funkcionalitou a přehledným uživatelským rozhraním tak, aby pro cílovou skupinu uživatelů byl při jejich práci přínosem.

9 Literatura

AngularJS: Data Binding. 2015. Angularjs.org [online]. [cit. 2015-05-14]. Dostupné z: <https://docs.angularjs.org/guide/databinding>

BRUCKNER, Tomáš. Tvorba informačních systémů: principy, metodiky, architektury. 1. vyd. Praha: Grada, 2012, 357 s. Management v informační společnosti. ISBN 978-80-247-4153-6.

BUCHALCEVOVÁ, Alena. Metodiky vývoje a údržby informačních systémů: kategorizace, agilní metodiky, vzory pro návrh metodiky. 1. vyd. Praha: Grada, 2005, 163 s. ISBN 80-247-1075-7.

Česká společnost rostlinolékařská, PŘÍRUČKA ODBORNÉ ZPŮSOBILOSTI PRO ZACHÁZENÍ S PŘÍPRAVKY NA OCHRANU ROSTLIN. In: Praha, 2010. Dostupné z: <http://www.vinarskaasociace.cz/fotky22580/pruka-SRS.pdf>

Český statistický úřad. Czso.cz [online]. 2015 [cit. 2015-05-19]. Dostupné z: www.czso.cz/csu/czso/zemedelstvi_zem

DAYLEY, Brad. Learning AngularJS. 1: Addison-Wesley Professional, 2014. ISBN 0134034619.

KLÍŘ, Jan a Lada KOZLOVSKÁ. Agromanual.cz, Hnojení podle nitrátové směrnice. [online]. 2013 [cit. 2015-05-19]. Dostupné z: <http://www.agromanual.cz/cz/clanky/vyziva-a-stimulace/legislativa-67/hnojeni-podle-nitratove-smernice.html>

KOSTELANSKÝ, František. Obecná produkce rostlinná. 2., nezměn. vyd. V Brně: Mendelova zemědělská a lesnická univerzita, 2004, 212 s. ISBN 978-80-7157-765-2.

KURTZ, Jamie. 2013. ASP.NET MVC 4 and the Web API: building a REST service from start to finish. New York: Apress, xvii, 131 s. Expert's voice in ASP.NET. ISBN 1430249781.

Microservice architecture: Pattern: Monolithic Architecture. 2014. <http://microservices.io> [online]. [cit. 2015-12-20]. Dostupné z: <http://microservices.io/patterns/monolithic.html>

MILLETT, Scott. 2015. Professional-domain driven design patterns. Indianapolis: John Wiley & Sons, pages cm. ISBN 11-187-1470-9.

PROCHÁZKA, David. CSS a XHTML: tvorba dokonalých WWW stránek krok za krokem - 2., aktualizované vydání. Praha: Grada Publishing a.s., 2011. ISBN 9788024771274.

RAHMAN, Mohammad. 2013. Expert C# 5.0 with .NET 4.5 framework. New York: Distributed to the book trade worldwide by Springer Science Business Media, xiv, 596 s. Expert's voice in .NET. ISBN 14-302-4860-2.

SHELLY, Gary B a Harry J ROSENBLATT. Systems analysis and design. 9th ed. Boston: Course Technology Cengage Learning, 2012, xx, 737 s. ISBN 11-332-7405-6.

SCHAFER, Steven M. HTML, XHTML a CSS: bible [pro tvorbu WWW stránek] : 4. vydání. 1. vyd. Praha: Grada, 2009, 647 s. ISBN 978-80-247-2850-6.

SMITH, Michael. HTML: The Markup Language. W3C [online]. 2012 [cit. 2015-02-20]. Dostupné z: <http://w3c.github.io/html-reference/spec.html>

Strukturální šetření v zemědělství 2013: analytické vyhodnocení. Praha: Český statistický úřad, 2015, 205 stran. Zemědělství. ISBN 978-80-250-2620-5.

ŠÍMA, František a David VILÍMEK. Microsoft Visual Studio .NET: praktické programování krok za krokem. 1. vyd. Praha: Grada, 2006, 254 s. Průvodce (Grada). ISBN 80-247-1418-3.

TARASIEWICZ, Philipp a Robin BÖHM. AngularJS. 1: Brainy Software Inc, 2014. ISBN 1771970014.

The Repository Pattern. Microsoft API and reference catalog [online]. [cit. 2015-05-13]. Dostupné z: <https://msdn.microsoft.com/en-us/library/ff649690.aspx>

UGURLU, Tugberk, Alexander ZEITLER a Ali KHEYROLLAHI. 2013. Pro ASP.NET web API: HTTP web services in ASP.NET. Berkeley, CA: Apress, xxv, 587 s. Expert's voice in .NET. ISBN 14-302-4725-8.

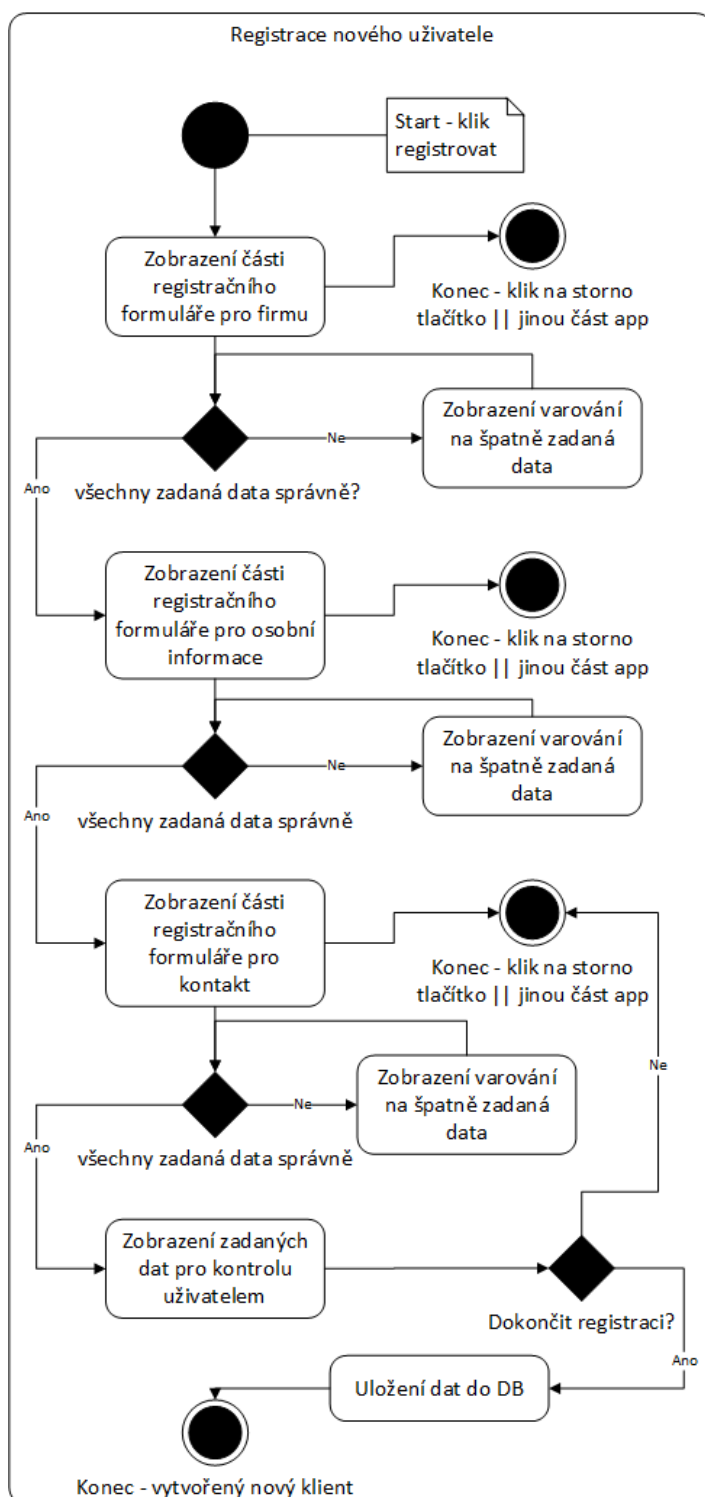
VYMĚTAL, Dominik. Informační systémy v podnicích: teorie a praxe projektování. 1. vyd. Praha: Grada, 2009, 142 s. Průvodce (Grada). ISBN 978-80-247-3046-2.

WOJCIESZYN, Filip. ASP.NET Web API 2 Recipes: A Problem-Solution Approach. California: Apress, 2014. ISBN 1430259817.

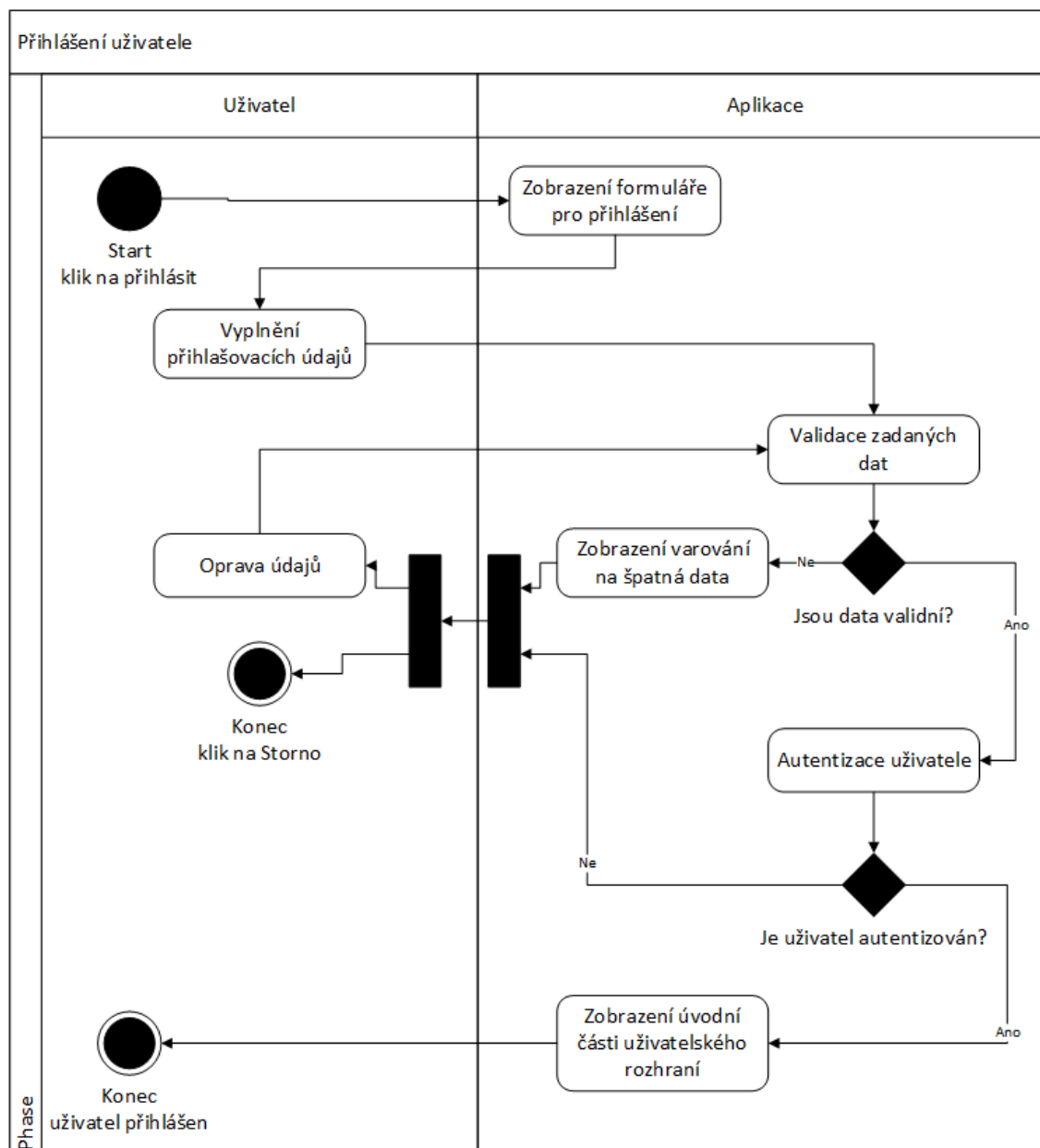
ZELINKA, Tomáš a Miroslav SVÍTEK. Telekomunikační řešení pro informační systémy síťových odvětví. 1. vyd. Praha: Grada, 2009, 218 s. ISBN 978-80-247-3232-9.

10 Přílohy

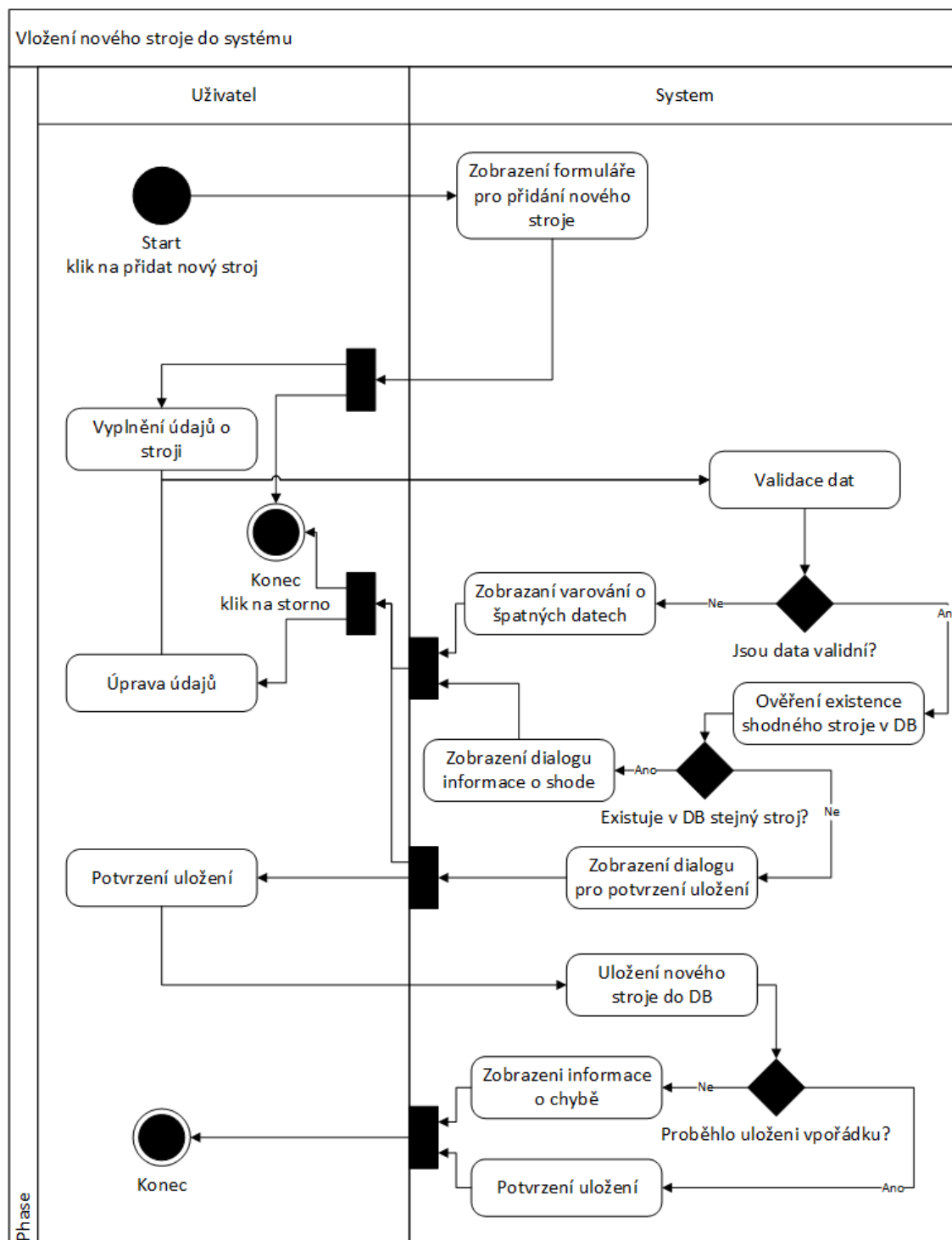
A. Diagram aktivit Registrace klienta



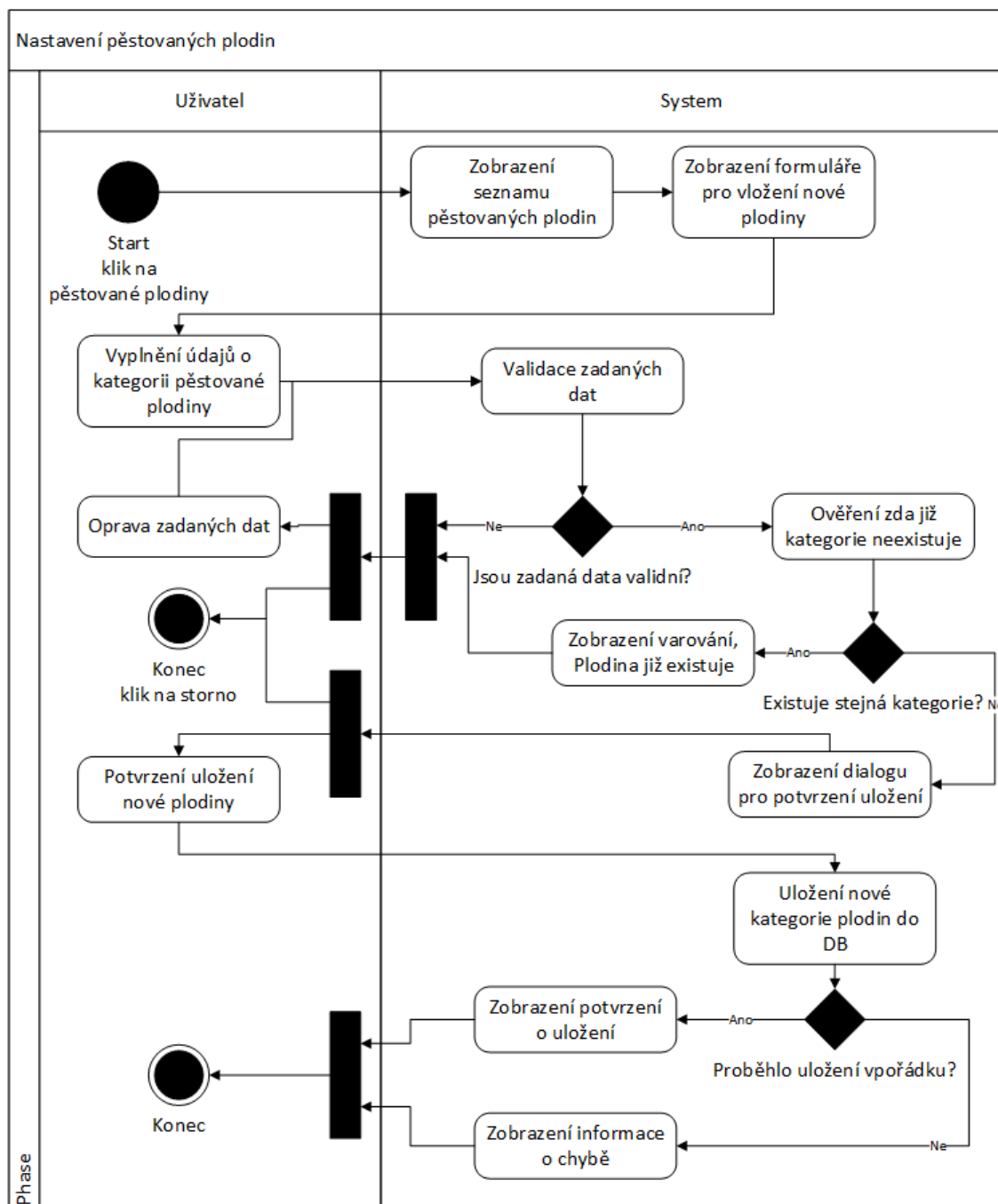
B. Diagram aktivit přihlášení do systému



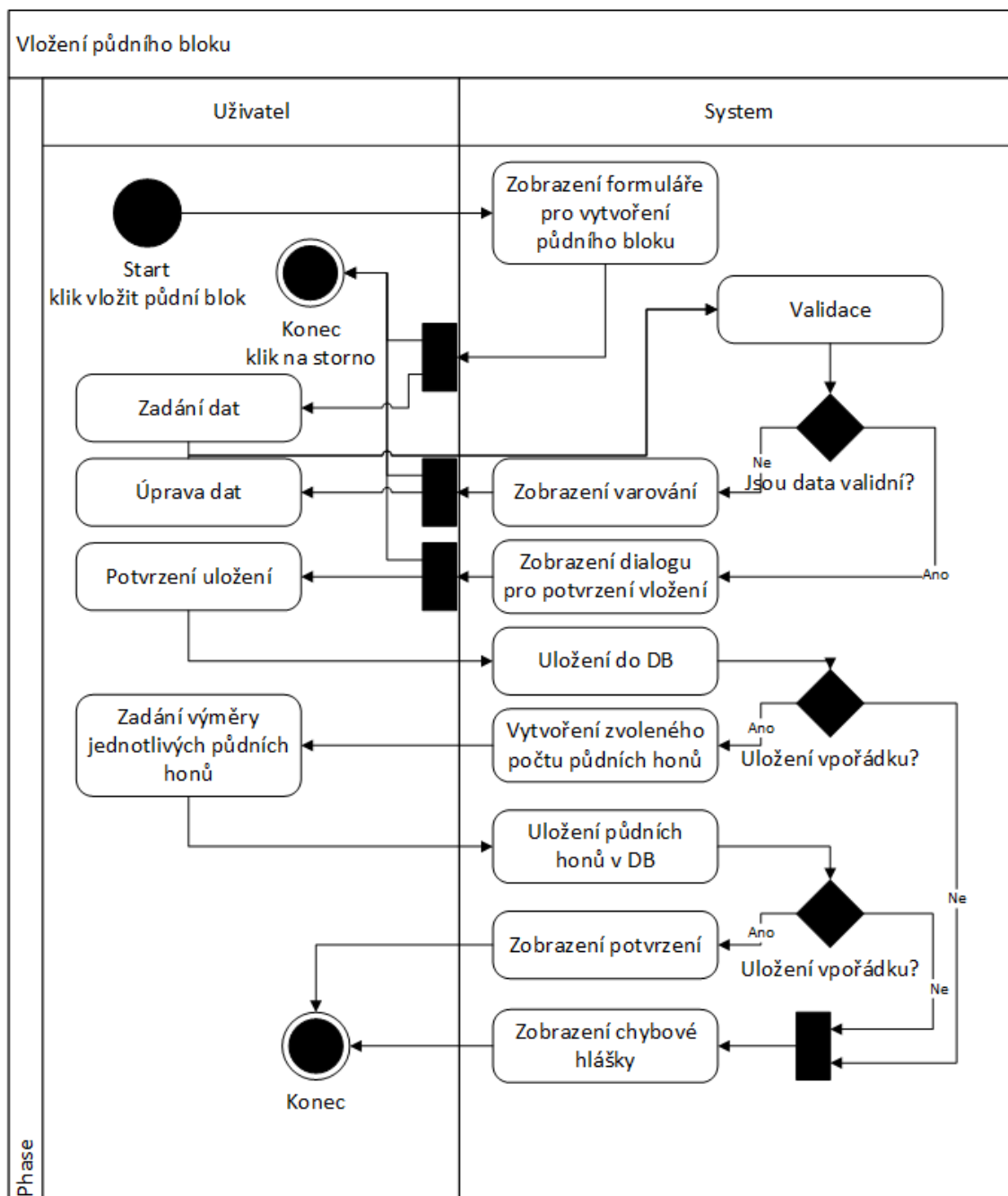
C. Diagram přidání nového stroje do IS



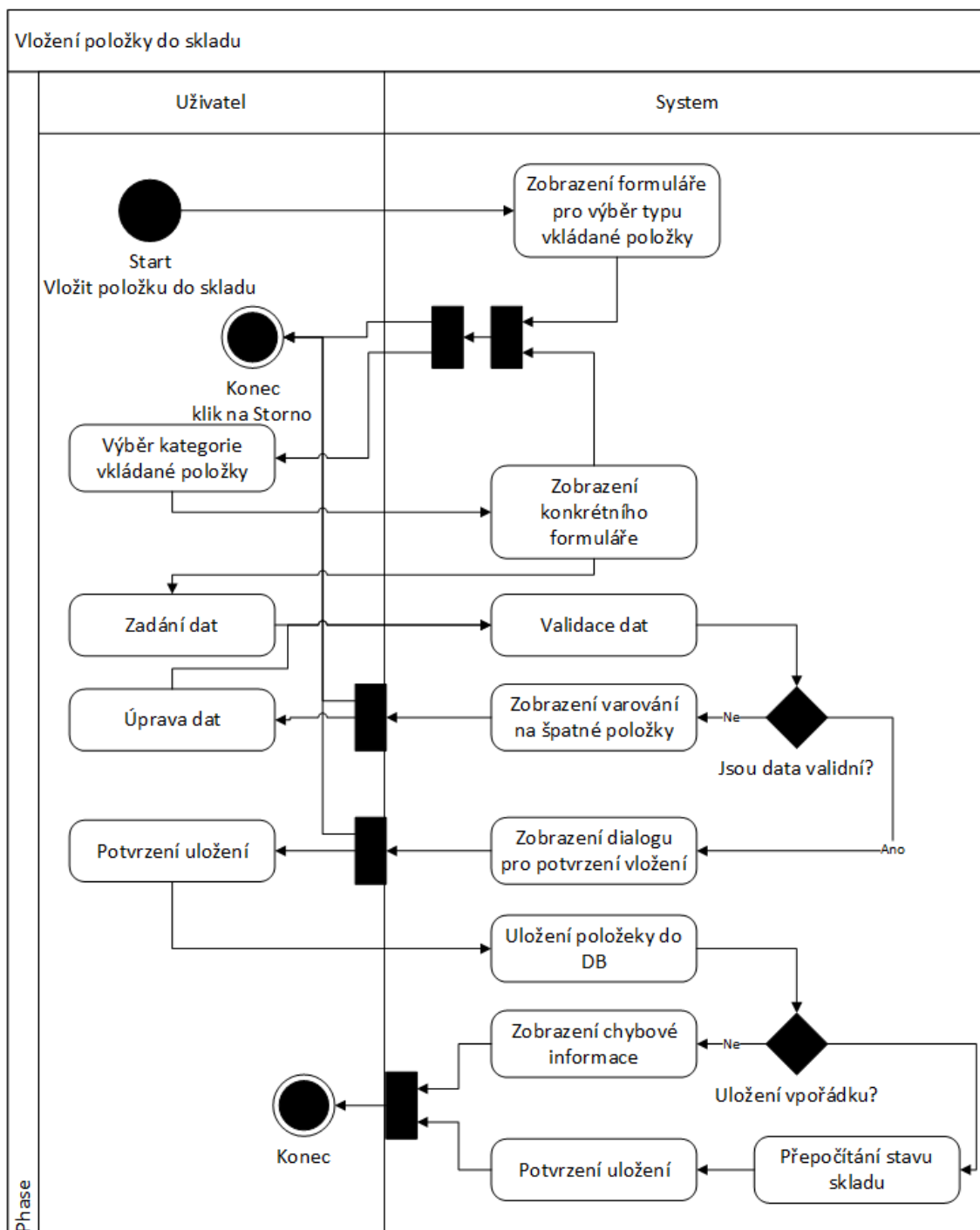
D. Diagram vytvoření kategorie plodin



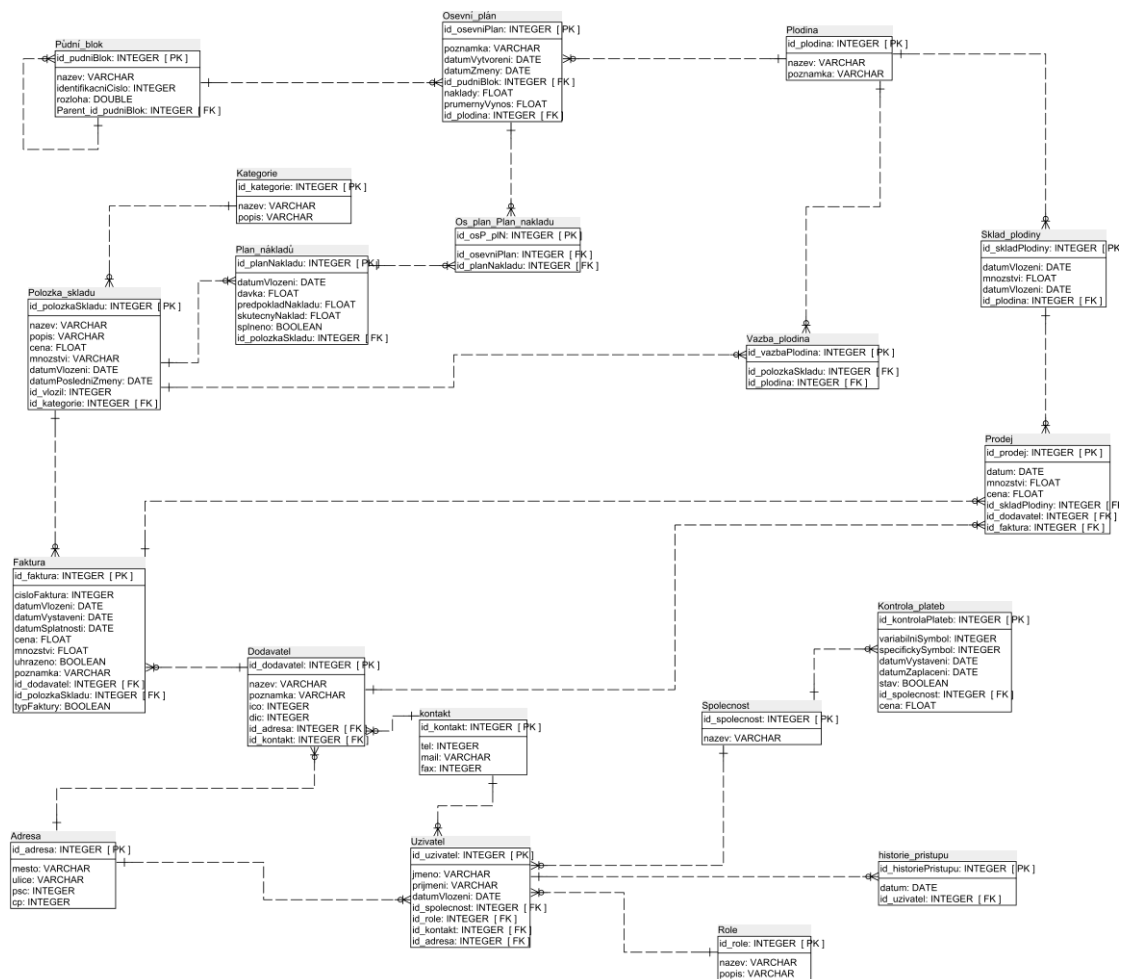
E. Diagram aktivit vytvoření půdního bloku



F. Diagram aktivit vložení nové položky skladu



G. ERD databáze



H. Ukázka implementace modelů entit

V příloze je uvedena ukázka kódu použitá pro implementaci modelů datové vrstvy.

```
namespace Data.Models
{
    public abstract class BaseModel
    {
        public long Id { get; set; }
        public DateTimeOffset CreatedAt { get; set; }
    }
}

namespace Data.Models
{
    public class CompanyRelatedModel : BaseModel
    {
        public long CompanyId { get; set; }
    }
}

namespace Data.Models
{
    public class SoilBlock : CompanyRelatedModel
    {
        public int IdentificationNumber { get; set; }
        public string Name { get; set; }
        public float Area { get; set; }
        public float N { get; set; }
        public float S { get; set; }
        public float K { get; set; }
        public float P { get; set; }
    }
}
```

I. Ukázka implementace šablony modálního dialogu

```

<div ng-show="newWorkModal">
  <div class="modal-header">
    <!--Nadpis dialogu-->
  </div>
  <div class="modal-body">
    <!--Jednotlivé vstupní pole dialogu-->
    <div class="col-md-5 well">
      <p>Spotřeba strojů při práci:</p>
      <table st-table="rowCollection" class="table table-striped">
        <tr ng-repeat="item in machineryList">
          <td>
            <label class="control-label"
              for="machineryCon">{{item.machineryName}}</label>
            <input type="number" class="form-control input-sm"
              id="machineryCon"
              ng-model="item.averageConsumption" />
          </td>
          <td><br><br>L/ha</td>
        </tr>
      </table>
    </div>
  </div>
  <div class="modal-footer">
    <!--Patička dialogu obvykle s ovládacími prvky-->
  </div>
</div>

```

J. Ukázka rozhraní pro tvorbu osevního plánu

Jak bylo v práci uvedeno osevní plán uživatel sestavuje z jednotlivých položek, které přidává během tvorby osevního plánu.

Domu
O aplikaci
Odhlásit se

Lukáš Dohnálek

Hospodářský rok

2016/2017

Vytvoření nového Osevního plánu pro hospodářský rok 2016/2017

Náklady: 88868 Kč

Půdní bloky

Osevní plány

Sklad

Technika

Finance

Nastavení

Výběr půdního bloku:

3 - U elektrárny / 26 ha

Výběr půdního honu:

U elektrárny

Výběr pěstované plodiny:

Pšenice ozimá

Název osevního plánu:

test

Tuna sklizené plodiny z půdy odebere:

25 Kg Dusíku, 4 Kg Síry
5 Kg Fosforu, 20 Kg Draslíku

Výměra půdního honu: 26 ha

Obsah živin v půdě:
0 Kg Dusíku, 0 Kg Síry
0 Kg Fosforu, 0 Kg Draslíku

Pomocí následujících tlačítek sestavte osevní plán.

+ Osivo

+ Hnojivo

+ PoR

+ Ostatní práce

Ulož

Zpět

03.01. 2016 Ostatní práce - Orba

03.01. 2016 Hnojivo - Hnojení

Hnojivo

LOVOFERT LAD 27

Dávkování(Kg/ha)

100

Práce

Hnojení

Technika

John Deere 8430

Popis práce Güstrower GDK 7000

Cena práce 550 Kč/ha

Popis stroje 340 koní

Spotřeba stroje 4 L/ha

Plánovaná aplikace

01/03/2016

Popis:

27 % N.

Cena za jednotku: 8.00 Kč

1 Kg přidá

N 27 kg/ha, S 0 kg/ha

P 0 kg/ha, K 0 kg/ha

Dodane latky do pudy

2700 Kg Dusíku, 0 Kg Síry

0 Kg Fosforu, 0 Kg Draslíku

Nafta

Na skladu: 1,000.00 L, hodnota 1L: 31.00 Kč

Material

Množství na skladu 180000 Kg

Potřebné množství 2800.00 Kg

Naklady material 20,800.00 Kč

Prace

Naklady práce 14,300.00 Kč

Odhadovaná spotřeba 104.00 L

Naklady pohonné hmoty: 2,418.00 Kč

Celkové naklady 37,518.00 Kč

© 2016 - Diplomová práce

K. Ukázka rozhraní pro kontrolu osevního plánu před uložením

Domu
O aplikaci
Odhlásit se

Lukáš Dohnálek

Hospodářský rok

2016/2017

Půdní bloky

Osevní plány

Sklad

Technika

Finance

Nastavení

Kontrola osevního plánu

Informace o půdním bloku

Identifikační číslo: U elektrárny

Název: U elektrárny

Celková výměra: 26 ha

Informace o půdním honu

Název: U elektrárny

Výměra: 26 ha

Celkové náklady na osevní plán: 202044

Informace o plodině

Název: Pšenice ozimá

Popis:

Obsah látek v půdě

N: 0 Kg/ha, S: 0 Kg/ha

P: 0 Kg/ha, K: 0 Kg/ha

Po splnění osevního plánu bude do půdy přidáno

N: 2700 Kg/ha, S: 0 Kg/ha

P: 0 Kg/ha, K: 0 Kg/ha

Tuna sklizené plodiny odebere z půdy

N: 25 Kg/ha, S: 4 Kg/ha

P: 5 Kg/ha, K: 20 Kg/ha

Osevní plán

Datum aplikace	Práce Stroj	Náklady práce Spotřeba	Prostředek Název	Dávka	Množství	Náklady	Celkové náklady
03.01. 2016	Orba John Deere 8430	31200.00 Kč 650.00 L	Ostatní práce	X	X	X	51350.00 Kč
19.01. 2016	Hnojení John Deere 8430	14300.00 Kč 104.00 L	Hnojivo LOVOFERT LAD 27	100 Kg/ha	2600.00 Kg	20800.00 Kč	37518.00 Kč
02.02. 2016	Setí John Deere 8430	31200.00 Kč 260.00 L	Osivo Osivo pšenice ozimé	0.1 Balení/ha	3.00 Balení	15000.00 Kč	54280.00 Kč
17.02. 2016	Postřik Fendt 716 Vario	7800.00 Kč 26.00 L	Přípravky na ochranu rostlin Nurelle D	1 L/ha	26.00 L	23920.00 Kč	32526.00 Kč
30.03. 2016	sklizen John Deere 8430	14300.00 Kč 390.00 L	Ostatní práce	X	X	X	28390.00 Kč

+ Uložit
↶ Zpět

© 2016 - Diplomova práce

L. Ukázka rozhraní detailu osevního plánu

Domu
O aplikaci
Odhlásit se

Lukáš Dohnálek

Hospodářský rok
2015/2016

Půdní bloky

Osevní plány

Sklad

Technika

Finance

Nastavení

Detail osevního plánu

Zpět

Informace o půdním bloku
Identifikační číslo: 1
Název: Pod silnicí
Celková výměra: 33 ha

Informace o půdním honu
Název: Pod silnicí
Výměra: 33 ha
Plánované náklady: **344629 Kč**
Skutečné náklady: 0 Kč

Informace o plodině
Název: Kukuřice setá
Popis:

Obsah látek v půdě
N: 0 Kg/ha, S: 0 Kg/ha, P: 0 Kg/ha, K: 0 Kg/ha

Napoveda:
☒ Bod osevního plánu není splněn
☒ Bod osevního plánu byl splněn

Provést sklizeň

Přidat položku

Body osevního plánu

15.04.2016 - , Předpokládané náklady: 21219 Kč

16.04.2016 - , Předpokládané náklady: 33000 Kč

02.05.2016 Osiva - Osivo kukuřice Pioneer , Předpokládané náklady: 52084 Kč

05.05.2016 PoR - GARDOPRIM PLUS GOLD 500 SC , Předpokládané náklady: 67023 Kč

15.08.2016 PoR - Nurelle D , Předpokládané náklady: 41283 Kč

25.10.2016 - , Předpokládané náklady: 64845 Kč

02.11.2016 - , Předpokládané náklady: 65175 Kč

© 2016 - Diplomova prace

M. Ukázka rozhraní pro editaci dodavatelů

Domu

O aplikaci

Odhlásit se

Lukáš Dohnálek

Hospodářský rok

2016/2017

Půdní bloky

Osevní plány

Sklad

Technika

Finance

Nastavení

Administrativa

Přidat dodavatele

Přidat fakturu

Dodavatelé

Přijem/Výdej

Faktury

Seznam dovatelů

Datum	Název	Ičo	Dič	
29.12.2015	Vlastní výroba	45678954	6543215CZ	
29.12.2015	Pioneer	63078295	CZ63078295	
29.12.2015	OSEVA,AGRO Bmo, spol. s r.o.	60707925	CZ60607925	
29.12.2015	Lovochemie, a. s.	49100262	CZ49100262	
29.12.2015	Agro - Artikel, s.r.o.	25589032	CZ25589032	
02.01.2016	AGRO 2000 s.r.o.	25586521	CZ25586521	
03.01.2016	Selgen, a. s.	47116099	CZ47116099	

Informace

Informace o dodavatelích, pohybu na skladě a fakturách.

© 2016 - Diplomova prace

O. Ukázka rozhraní pro správu faktur

Domu
O aplikaci
Odhlásit se

Lukáš Donálek

Hospodářský rok

2016/2017

Půdní bloky

Osevní plány

Sklad

Technika

Finance

Nastavení

Administrativa

Přidat dodavatele

Přidat fakturu

Informace

Detail faktury číslo: 1

Výdej materiálu

Firma 1

Název firmy: Lovochemie, a. s.

IČO: 49100282

DIČ: CZ49100282

Sídlo:

Město: Lovosice

Ulice: Terežínská 57

PSČ: 41002

Firma 2

Název firmy: Vlastní výroba

IČO: 45878954

DIČ: 6543215CZ

Sídlo:

Město: Moutnice

Ulice: Mou 57

PSČ: 66455

Typ platby: Převodem

Účet příjemce: 321 - 78452103 / 7690

Konstantní sym.: 500

Variabilní sym.: 126456

Specifický sym.: 968545

Položky faktury:

Popis	Cena
1. "Osivo kukuřice Pioneer (Odrůda PR37N01) Množství v jednotkách: Balení" Počet: "33"	"4300" Kč
2. "GARDOPRIM PLUS GOLD 500 SC (Herbicid na jednoděložné v kukuřici) Množství v "8500" jednotkách: L" Počet: "20"	Kč

Datum splatnosti: 20.01. 2016

Celková cena: 12800 Kč

Tisk

Zavřít detail

Dodavatelé

Příjem/Výdej

Faktury

Seznam faktur

Typ	Číslo faktury	Datum vystavení	Datum splatnosti	Firma 1	Firma 2	Počet položek	Celková cena	Stav
+	1	31.12. 2015	20.01. 2016	Lovochemie, a. s.	Vlastní výroba	2	12.800,00 Kč	Uhrazeno

Neuhrazeno

© 2016 - Diplomová práce