VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

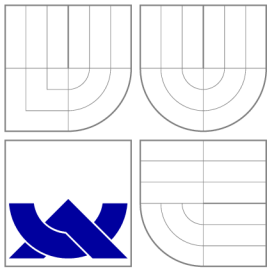
VIRTUÁLNÍ BOTANICKÝ HERBÁŘ S PODPOROU VÝUKY

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

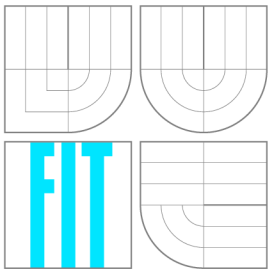
AUTOR PRÁCE
AUTHOR

JIŘÍ SEMMLER

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VIRTUÁLNÍ BOTANICKÝ HERBÁŘ S PODPOROU VÝUKY

VIRTUAL HERBARIUM WITH E-LEARNING FEATURES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JIŘÍ SEMMLER
AUTHOR

VEDOUCÍ PRÁCE — RNDr. MAREK RYCHLÝ, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Tato práce se zabývá analýzou, návrhem a implementací webového informačního systému pro správu virtuálního herbáře v prostřední střední školy. Stávající služby na trhu nenabízí žádnou variantu, kde by mohli studenti spolupracovat s profesory při tvorbě jejich virtuálních herbářů. Tato služba poskytuje platformu pro tvorbu vlastních sbírek fotografií rostlin s velkým důrazem na jejich zařazení do taxonomického systému, důkladný popis, kontrolu plagiátů a vzájemnou komunikaci studentů a učitelů.

Abstract

This thesis is concerned with analysis, design and implementation web information system for management of virtual digital herbarium for high or middle schools. Online available services can not provide any alternative where students can cooperate with teachers in creating own virtual digital herbarium. This service provides a platform for creating own collection of pictures of plants with stress on their categorization to the taxonomy, detailed description, checking plagiarism and communication between students and teachers.

Klíčová slova

herbář, taxonomie, elearning, informační systém, Nette, YetORM, agilní metodologie, Lean Software Development

Keywords

Herbarium, taxonomy, elearning, information system, Nette, YetORM, agile metodology, Lean Software Development

Citace

Jiří Semmler: Virtuální botanický herbář s podporou výuky, bakalářská práce, Brno, FIT VUT v Brně, 2015

Virtuální botanický herbář s podporou výuky

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana RNDr. Marka Rychlého, Ph.D.

.....
Jiří Semmler
29. května 2015

Poděkování

Děkuji svému vedoucímu práce RNDr. Markovi Rychlému, Ph.D. za odborné vedení, cenné rady a správné nasměrování při práci. Dále děkuji Ing. Vítězslavovi Beranovi, Ph.D. za pomoc a konzultaci v oblasti testování systému.

Obsah

Kapitola 1

Úvod

Cílem této práce je navrhnout a implementovat informační systém pro evidenci virtuálních herbářů ve studentském prostředí pro cílovou střední školu. V kapitole 2 se budeme zabývat motivací řešení virtuálních herbářů formou informačního systému a zaměříme se na principy použití ve výuce. Pro implementaci cílového informačního systému vybereme v kapitole 3 metodiku vývoje, kterou se budeme řídit. Následně v kapitole 4 zanalyzujeme stávající dostupná řešení a na základě této analýzy se budeme zabývat definicí požadavků na cílový softwarový produkt a budeme uvažovat, jakou bude mít informační systém kognitivní funkci. Před samotnou implementací porovnáme a vybereme v kapitole 5 technologie, které použijeme pro vývoj, který budeme detailně popisovat v kapitole 6. Výsledný informační systém, který v kapitole 6 naimplementujeme, podrobíme v kapitole 7 důkladnému uživatelskému testování. To nám odhalí množství funkcionalit, které jsme opomněli v základní specifikaci a budou podnětem k dalšímu rozvoji systému.

Virtuální herbář je sbírka záznamů o rostlinách s jejich fotografiemi (záběry všech důležitých částí rostliny jako jsou květ, stonek, list apod.) a všemi důležitými informacemi, tzv. *schedou*. Správné a dostatečně obsáhlé pojednání a shrnutí botanických informací je podkladem pro další studium, určování a zařazování dalších rostlin. Existence virtuálního herbáře je motivována přístupem ke chráněným druhům, možností sdílení mezi širší zájmovou komunitou a prezentací na originálním zdroji, proti barevně nebo tvarově deformované sušené rostlině. Všechny tyto body ústí v tvorbu informačního systému virtuálního herbáře v kapitole 4.

Kapitola 2

Obecný informační systém

Informační systém (dále IS) v dnešní době je velice žádaný a široce rozšířený druh softwaru, který napomáhá například firmám s managementem, školám s organizací studia nebo například dopravnímu podniku řídit dopravní situaci. Definujme si ovšem prvně pojem informačního systému.

2.1 IS jako klient-server aplikace

Dnešní obecný přístup k základním informačním systémům (školní IS, ekonomické IS, tato práce...) se definuje jako princip klient-server aplikace, kde se klientem myslí pouze webový prohlížeč (bez rozdílu na zařízení, ve kterém je spuštěn), který pracuje se vzdáleným webovým serverem (Apache, IIS, ngnix ...). Technologie použitá pro implementaci na serveru se může lišit dle zvoleného řešení (PHP, Python, Java...), ovšem na straně klienta jsme omezeni technologiemi, se kterými si poradí webový prohlížeč. Jedná se o některou z mutací značkovacího jazyka HTML, kaskádových stylů CSS a scriptovacího jazyka JavaScript. V ojedinělých případech se můžeme setkat s technologiemi Flash, Silverlight nebo Java. Výhody řešení klient-server jsou zejména:

- Klient potřebuje pouze webový prohlížeč, tedy jsou požadavky na instalovaný software minimální.
- Řešení je z pohledu uživatele multiplatformní, tedy IS může používat uživatel bez ohledu na operační systém či procesorovou platformu (x86/ARM).
- IS nemusíme ovládat pouze ze stolního počítače nebo z notebooku, ale i z tabletu nebo z chytrého telefonu (pokud je webové prostředí IS uzpůsobeno pro použití z těchto zařízení).
- IS není vázaný na vybranou pracovní stanici (například v zaměstnání), ale je přístupný kdekoliv, kde je internetové připojení a moderní webový prohlížeč.
- Pro IS je možné vytvořit API, které spolupracuje s jiným softwarem nebo technologií - například REST API[1].

2.1.1 Pravidla přístupného webu

Na straně serveru je definice požadavků dána technickými parametry, zatímco na straně webového prohlížeče je situace velice subjektivní - co je správný design, co se dobře a in-

[1] http://www.zdrojak.cz/clanky/rest-architektura-pro-webove-api/

tuitivně používá apod. Při snaze generalizovat požadavky ze strany klientské části dochází k definici tzv. *pravidel přístupného webu*[13]. Jedná se o novelu zákona č. 365/2000 Sb. o informačních systémech veřejné správy, provedenou zákonem č. 81/2006 Sb., která definuje celkově 37 pravidel, kterými se musí řídit webová stránka, aby byla v souladu s touto normou. Jedná se o normu, která definuje jaké prvky má stránka obsahovat, aby bylo dosaženo maximální použitelnosti zejména pro handicapované uživatele. Principy normy jsou dva. Prvním cílem normy je aby handicapovaní uživatelé byli schopni webovou stránku (z našeho pohledu viděno jako IS) používat bez ohledu na jejich postižení, druhý cíl popisuje procesy, kterými se má web řídit a strukturu, kterou má stránka mít pro maximální efektivitu. Jednou z důležitých motivací celé normy je i nástroj pro hodnocení projektů při výběrových řízeních a jiných posudcích ve státní správě. Norma je složena ze šesti kapitol:

- **Kapitola A: Obsah webových stránek je dostupný a čitelný**
  Popisuje fakt, že informace/funkcionalita webové stránky musí být přenositelná na uživatele i za podmínek, že uživatel například není schopen rozeznat barvy, softwarová výbava uživatele nepodporuje některou technologii (Flash apod.).
- **Kapitola B: Práci s webovou stránkou řídí uživatel**
  Hovoří o práci i přístupu k uživateli a o procesech, které by měly při práci se stránkou probíhat. Tedy například formulář musí být vždy odeslán pouze na požadavek uživatele, nebo zvuky delší než tři vteřiny musí být nastavitelné.
- **Kapitola C: Informace jsou srozumitelné a přehledné**
  Definuje strukturu webu a jejich popis.
- **Kapitola D: Ovládání webu je jasné a pochopitelné**
  Hovoří o umístění a komplexnosti navigačních prvků pro práci s webem.
- **Kapitola E: Kód je technicky způsobilý a strukturovaný**
  Stanovuje pokročilá pravidla kódu webové stránky vycházející z jazyka HTML.
- **Kapitola F: Prohlášení o přístupnosti webových stránek**
  Rozděluje pravidla na povinná a nepovinná a stanovuje podmínky, za kterých je možno stránku prohlásit v souladu s normou.

Nutno zmínit, že splněním této normy není zaručeno jakékoliv zvýhodnění (například ve vyhledávání). Stejně tak norma nevychází z vědeckých výzkumů, ale jedná se pouze o best-practises, ke kterým došlo v roce 2004 Ministerstvo informatiky ČR. I kdyby uživatel, který použije náš web či IS, neprovedl konverzní akci[14], má jeho návštěva pozitivní dopad například na hodnocení webu ve vyhledávači.
IS a jeho webová prezentace, která je předmětem této práce, je v souladu se zmíněnou normou Pravidel přístupného webu.

Kapitola 3

Metodika a plán vývoje

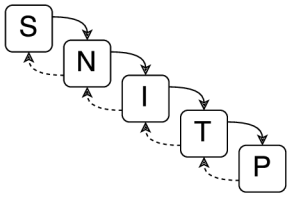
Řešení softwarové krize v 60. a 70. letech vedlo k zavedení tzv. *dobrých programovacích praktik*, tedy postupů, které se osvědčily v programátorské praxi a v následné definování *metodik*, které popisovaly *analýzu* a *návrh* softwaru a později i *procesy* pokrývající celý životní cyklus softwaru[6]. V následující kapitole si probereme základy dvou metodologických přístupů, jejich klady, zápory a zvolíme metodiku pro vývoj IS, který je tématem této práce.

3.1 Obecná metodologie

Každý vývoj softwaru musí projít několika základními kroky. Jedná se o **sběr požadavků**, **návrh**, **implementace**, **testování** a **provoz**[6] softwaru. Komunikací, řízením a spoluprací se zabývá věda zvaná **metodologie**, jejímž výstupem jsou jednotlivé **metodiky** vývoje. Z historického i konceptuálního hlediska dělíme metodiky na klasické rigorózní a agilní.

3.1.1 Klasické rigorózní metodiky a modely

Základem většiny metodik je tzv. *vodopádový model* (obrázek 3.1), ze kterého následně všechny ostatní modely a metodiky dědí. Základ tohoto modelu je, že každým krokem se prochází lineárně a pouze jednou. Tedy například testování může začít až ve chvíli, kdy je implementace finální. Hlavní nevýhody jsou dvě. První nevýhodou fakt, že pokud selže jedna z částí (například změna specifikace), musí se projít celým modelem znova, což může znamenat razantní úpravy. A druhá nevýhoda spočívá ve stavu, kdy výstup vidí zákazník až na konci. Tento model se spojuje s procesy a složitým řízením, které odděluje jednotlivé části od sebe a soustředí se na procesní řízení než na samotný produkt a vývoj. Vodopádový model je ale základem všech ostatních modelů, které ho jen upravují a vzniká například model iterativní, inkrementální nebo spirálový.



Obrázek 3.1: Vodopádový model

3.1.2 Agilní metodiky

Agilní metodiky jsou odpovědí na nespokojenost s vodopádovým modelem, který se používal do druhé poloviny devadesátých let. Na postupné snahy o zeštíhlení používaných rigorózních metodik reagovala skupina softwarových inženýrů v roce 2001 vydáním tzv. *Manifest Agilního vývoje software*[2], které definovalo základy agilních metodik a na základě jejich myšlenek následně vznikají mnohé agilní metodiky, jako je SCRUM[1] nebo XP[2].
Základ agilní metodiky je v iterativním nebo inkrementálním modelu a staví se do opozice proti původním procesně zaměřeným metodikám. Agilní metodiky staví na několika myšlenkách[2]:

- Upřednostnění lidí a jejich spolupráce před procesy a nástroji.
- Fungující software je důležitější než obsáhlá dokumentace.
- Úzká spolupráce se zákazníkem je důležitější než smlouvy.
- Rychlá reakce na změnu má prioritu před smlouvou.

Pro agilní metodiky je typická intenzivní (nejlépe osobní) komunikace, rychlé porady (meetingy), iterace a snaha o co nejjednodušší přístup bez zatížení procesním řízením.

3.1.3 Lean Software Development

Agilní metodologie nemusí být nutně záležitost posledních 15 let, jak bylo zmíněno v předchozím odstavci a zároveň IT není obor, který by nebyl ovlivněn jinými odvětvími průmyslu. V polovině 20. století měla Toyota výrazný problém s konkurencí v podobě amerického General Motors a z důvodu rozdílů mezi trhy nebylo možné jednoduše aplikovat americkou sériovou výrobu v Toyotě. Viceprezident Taiichi Ohno definoval sedm druhů plýtvání a základní myšlenky, které položily základy TPS (Toyota Production System).

Na základě TPS vzniká obecný přístup ke štíhlejším procesům tzv. Lean myšlení a následně aplikací na vývoj softwaru vzniká Lean Software Development [4]. Jedná se agilní metodiku či soubor pravidel (agilní metodologie se vyznačuje odporem k deterministickému předpisu procesů, tedy terminologie není definitivní), které zeštíhluje vývojové procesy a eliminuje plýtvání při vývoji.

Stejně jako u všech agilních metodik, tak i u LeanSD je základním stavebním kamenem iterativní model, tedy celá práce je dekomponovaná do několika částí (iterací). Pro aplikaci LeanSD na vývoj je základem dodržování následujících sedmi klíčových myšlenek [8]. Všechny body mají své metody a modely, které se používají pro analýzu, některé se využívají pro analýzu týmové práce, některé pro analýzu zákazníkovy specifikace.

I **Eliminace plýtvání**
Plýtvání při vývoji softwaru (dále je SW) je stejné jako ve výrobě Toyoty. Následujících sedm principů plýtvání, které vidíme v tabulce 3.1, můžeme následovně transformovat na vývoj softwaru.

[1] https://www.scrum.org/Resources/What-is-Scrum
[2] http://www.extremeprogramming.org/

Výroba	**Vývoj** SW
Nadprodukce	Funkcionalita
Přeprava	Předávky artefaktů mezi rolemi
Zásoby	Neúplná, nedokončená práce
Zpracovávání	Nutnost znovu se učit zapomenuté
Pohyby	Střídání úkolů
Čekání	Prodlevy
Vady	Chyby

Tabulka 3.1: Mapování výrobního plýtvání na softwarový vývoj

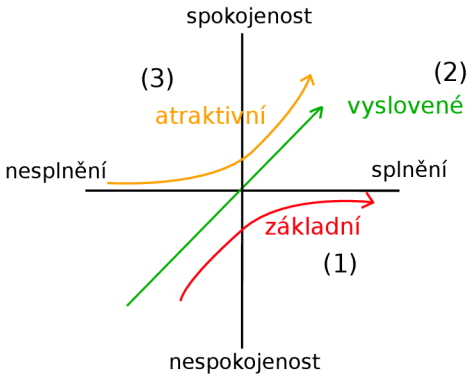
Použití

Pro minimalizaci plýtvání (aby vývojář pracoval co nejefektivněji a nedělaly se redundantní operace) se používá množství nástrojů, mezi nimi například Kanban, ale v této práci využíváme pro jednoduchost a účelnost následující nástroje v různých úrovních:

A - Identifikace hodnoty - Kano model

Pro identifikaci hodnoty při vstupní analýze můžeme využít mnohé nástroje a LeanSD přímo neurčuje, které to musí být. Pro účely této práce byl vybrán tzv. **Kano model**[3].

Kano[3] model je nástroj pro analýzu a mapování vstupních zákaznických požadavků. Jak vidíme na obrázku 3.2, jedná se o rozdělení požadavků do tří kategorií[11]:



Obrázek 3.2: Znázornění závislosti druhů požadavků na spokojenosti zákazníka

1. **Základní požadavky (1)** - Jedná se požadavky, které jsou jistým implicitním pravidlem a o kterých se nepochybuje. Jedná se například o základní zabezpečení, o persistenci dat apod. Tyto požadavky musí být v každém případě naplněny, aby vedly k základní lhostejné spokojenosti. Nerealizace těchto požadavků vede k silné nespokojenosti zákazníka.
2. **Vyslovené požadavky (2)** - Jedná se funkcionalitu/vlastnosti SW, které zákaz-

[3]název dle japonského profesora Noriaki Kana

ník explicitně definuje a naplňují jeho potřeby. Jejich splnění vede ke spokojenosti. Například příjem interní zprávy bude doprovázen notifikací na email adresáta.

3. **Atraktivní požadavky (3)** - Tyto požadavky zákazník přímo nespecifikuje a také je nutně nevyžaduje. Ale jejich realizace vede k nadstandardní spokojenosti zákazníka. Jejich realizace není nutná a zároveň není žádoucí, pokud nejsou splněny první dvě úrovně. Například pro psaní zprávy interní pošty se místo běžného textového pole zobrazí WYSIWYG editor. U těchto požadavků je nutno zvážit poměr cena/-výkon. Tedy kolik zdrojů bude nutno do implementace požadavku investovat a kolik spokojenosti to zákazníkovi přinese.

Požadavky takto dělíme při každé analýze specifikace nebo návrhu, ideálně při každé iteraci. Na základě takto rozdělených požadavků víme, na co se zaměřit, co se dá vynechat, případně kde je prostor pro levnou implementaci atraktivních požadavků. Například implementace již použitého WYSIWYG editoru je výrazně levnější než propojení interní pošty a Facebooku.

B - Eliminace prvků plýtvání

Ke každému prvku potencionálního plýtvání, které jsou zmíněny v kapitole 3.1.3 v tabulce 3.1, máme nástroj pro jeho redukci.

1. **Funkcionalita** - Implementovat potřebnou funkcionalitu až ve chvíli, že bude potřeba. Rozhodnutí o implementaci funkcionality rozhoduje právě Kano modelem I.

   Použití: Před každou iterací definujeme, které funkcionality budeme implementovat dle Kano modelu.

2. **Předávky artefaktů mezi rolemi** - Omezit předávky úkolů mezi členy týmu na minimum, dbát na úzkou komunikaci členů při nutné předávce a na zpětnou vazbu.

   Použití: V této práci je předávání úkolů pouze na úrovni změny vývojových rolí (viz kapitola 3.3).

3. **Neúplná, nedokončená práce** - Nedefinovat požadavky příliš s předstihem, dokud není jasný soupis specifikace, pravidelně testovat, dokumentovat během vývoje, dodávat zákazníkovi co nejčastěji.

   Použití: Iterační vývoj této práce pokrývá tento bod, neboť během implementace dochází k psaní dokumentace kódu, na každém konci iterace je produkt důsledně testován a konzultován každý milník se zákazníkem. Kód je komentován během vývoje technikou JavaDoc[4].

4. **Nutnost znovu se učit zapomenuté** - Dokumentovat co nejblíže k vývoji, ideálně přímo do kódu.

   Použití: Kód je obsáhle komentován.

5. **Střídání úkolů** - Minimalizovat střídání lidských zdrojů na úkolu.

   Použití: Ke střídání lidských zdrojů na práci zde nedochází, poněvadž projekt je realizován pouze jedním autorem.

6. **Prodlevy** - Maximální snaha o přímou komunikaci mezi členy a týmy pro minimalizaci časových prodlev.

   Použití: Zde opět je tento krok minimalizován samostatným řešením práce.

[4] http://www.oracle.com/technetwork/articles/java/index-137868.html

7. **Chyby** - Tvorba testů a opakované testování.
   Použití: Funkcionalita je testována co nejdříve.

II **Včleňovat kvalitu do vývoje**
Základem této druhé myšlenky je, aby byl produkt v každé části svého vývoje v maximální kvalitě a nikdy během vývoje se nesklouzlo k porušení/snížení kvality. Opět LeanSD definuje množství prvků, kterých je vhodné se vyvarovat a k nim definuje nástroje, které negativní elementy eliminují. Z pohledu vývoje této práce byly implementovány pouze tři prvky:

- **Iterativní vývoj se zpětnou vazbou** - Do klasického iterativního modelu se vkládá zpětná vazba, která reportuje výsledky (pozitivní a zejména negativní) pro eliminaci stejných chyb v další iteraci.
- **Testování se zpětnou vazbou** - Stejně jako se předávají best-practices v rovině vývoje či řízení v předcházejícím bodě, v případě testování se předává zpětná vazba stejným způsobem. Tato práce silně závisela na zpětné analýze úspěchů a neúspěchů a na předávání zpětné vazby mezi iteracemi.
- **Principy 5S**[5] - Princip 5S popisuje, jakými kroky se má analyzovat iterace pro další kroky. Pravidlo 5S je složeno z následujících lineárně navazujících kroků:
  1. Vytřídit - refaktorovat kód a odprostit jej od nepotřebných částí.
  2. Zorganizovat - logicky přeorganizovat strukturu projektu a kódu.
  3. Uklidit - vyřešit varovná hlášení ze všech částí projektu.
  4. Standardizovat - nahradit přílišnou složitost, která znepřehledňuje kód a projekt, jednodušším a elegantnějším řešení.
  5. Udržovat - zavést výsledky předchozích kroků do udržitelné podoby.

Všechny tyto body byly realizovány na konci každé iterace vývoje.

III **Vytvářet znalosti**
Cílem projektu a vývoje by mělo být také shromažďovat znalosti a udržovat je v dostupné podobě pro všechny členy týmu, aby následně bylo použití těchto informací efektivnější (nemusela se stejná věc vymýšlet znovu). Dojde-li ke změně zadání, je třeba tuto změnu maximálně popsat a umístit co nejblíže k vývojáři, aby pracoval s nejlepšími daty a nedocházelo k plýtvání. LeanSD opět popisuje množství nástrojů pro tvorbu znalostí, jako je párové programování, interaktivní školení (workshop) nebo sdílené materiály (wiki), ale tato práce využívá pouze JavaDoc dokumentaci.

IV **Odkládat závazky**
Odložení závazného rozhodnutí na co nejpozdější termín umožňuje, aby toto rozhodnutí bylo učiněno na základě aktuálních dat (nejčastěji od zákazníka). Změní-li se totiž data, na kterých stojí závazné rozhodnutí, je potřeba následně dané rozhodnutí přehodnotit, což může opět vést k plýtvání. LeanSD opět popisuje řadu nástrojů pro odkládání závazných rozhodnutí v závislosti aplikace změn s co nejlevnějším dopadem. Základem odkládání závazků je opět analýza mezi iteracemi. Z důvodu nedělení rolí na návrháře, architekty a programátory v této práci (autor vše zastává v jedné osobě) není potřeba se tímto bodem zabývat.

[5] název vznikl z japonských slov Seiri (vytřídit), Seiton (zorganizovat), Seiso (uklidit), Seiketsu (standardizovat), Shitsuke (udržovat)

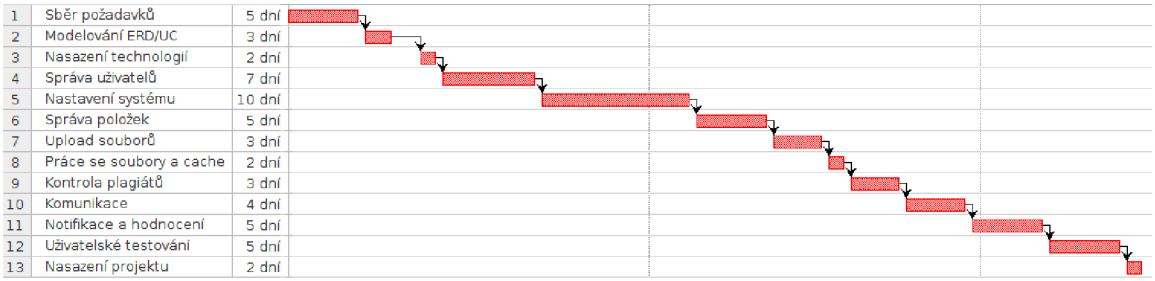
V **Dodávat co nejrychleji**
Co nejvyšší frekvence dodávání výsledků iterací zákazníkovi, kdy zákazník opravdu vidí výsledek a může jej mapovat na své potřeby, předpokládá, že zákazník bude lépe a hlavně co nejdříve upravovat specifikaci pro jeho potřeby. Zákazník nemusí pracovat pouze s domněnkami či modely, ale pracuje s reálným mezivýsledkem. V této práci byla dodávka řešena skrze Skype hovor se zákazníkem po každé iteraci, která se odvíjela z časového harmonogramu 3.2, tedy byla generována hodnota pro zákazníka průměrně jednou týdně. Pokud to iterace dovolovala, byl výstup iterace nasazen na testovací prostředí a zpřístupněn zákazníkovi pro testování.

VI **Důvěra a respekt k lidem podílejících se na vývoji**
Vývoj softwaru je kreativní činnost, kterou je potřeba podpořit důvěrou, respektem a dobrým pracovním prostředím. Vzhledem k jednočlenné realizaci není tento bod implementován do práce.

VII **Optimalizace celku**
Zákazník požaduje funkční celek, ne pouze funkční části. Pokud části nefungují jako celek, pak převážily lokální optimalizace nad celkovými. Vzhledem k jednočlenné realizaci není tento bod implementován do práce.

3.2 Časový harmonogram vývoje

Celý projekt se řídí iterativním modelem a LeanSD. Na začátku, ještě před započetím jakékoliv implementace či analýzy, byl stanoven časový harmonogram, který je znázorněn na Ganttově diagramu v obrázku 3.3. Odhad pracnosti je tedy 56MD[6].



Obrázek 3.3: Ganttův diagram vývoje

Jak vidíme na obrázku, celkově je definováno 13 iterací. Každá iterace se dělí do následujících částí:

- **analýza požadavků** - krze Kano model (I)
- **návrh řešení** - většinou s úpravou schématu databáze, návrh objektové notace v systému apod.
- **implementace** - samotná implementace cílené funkcionality
- **testování** - nutnost testování každé funkcionality

[6]ManDay - 8 pracovních hodin

- **LeanSD režie a konzultace** - Po každé iteraci je nutné provést analýzu dle zmíněných postupů definovaných v LeanSD a konzultovat výsledek iterace se zákazníkem.

3.3 Role během vývoje

Jelikož celá práce je realizována pouze v jedné osobě, nikoliv v týmu, který by obsahoval všechny potřebné role jako analytik, návrhář, programátor a tester, bylo nutné zavést proces pro střídání rolí. Z vlastní zkušenosti vím, že mám-li podat maximální výkon, musím se soustředit pouze na jeden druh činnosti. Je to přínosné z důvodu efektivity, ale zároveň z principu zpětné vazby. Cílem tedy bylo, aby bylo možné měnit svou autorskou roli během vývoje, ale zároveň se vyvarovat plýtvání vzniklé přílišnými změnami rolí a předávkami úkolů. Tohoto cíle bylo dosaženo tím, že jsem si zvolil roli, ve které momentálně vystupuji a dělám danou práci dle maximálních možností bez ohledu na dopady v jiných rolích. Následně jsem zvolil krátkou přestávku a po přestávce jsem začal pracovat na úkolu v podobě jiné role. Lidské myšlení je totiž zkresleno pohledem, ze kterého nahlíží na situaci. Nejsilnější příklad byl v implementaci. Během implementace bylo nutno realizovat například roli tzv. programátora a UX návrháře. Jako programátor jsem všechny elementy, které interagují s uživatelem, vyplnil pouze základními daty (například chybová hláška) a následně při změně role jsem všechna interakční data přepsal bez ohledu na změnu funkčnosti (přepsal jsem chybové hlášky z „*Chybí heslo*“ na ”*Prosím zvolte heslo s minimálně šesti znaky.*”). Je evidentní, že hlášky mají výrazně odlišnou informační hodnotu pro uživatele. A bylo by plýtváním (jelikož výsledek by nebyl tak dobrý a musel by se refaktorovat) snažit se o realizaci obou rolí zároveň.

3.4 Výsledek

Základní výstupy této kapitoly bychom měli vidět v definici vývojových metodik, ve výběru vhodné metodiky pro tento projekt a v časovém odhadu vývoje. Všechna rozhodnutí, která byla nebo budou během vývoje provedena, budou provedena v souladu s definovanými myšlenkami LeanSD.

Kapitola 4

Sběr požadavků a analýza

4.1 Motivace pro řešení tématu

Motivace a zadání projektu vychází z požadavků Mgr. Vladimíra Zatloukala z bruntálského gymnázia (předchozí studium autora) na efektivnější a studentům dostupnou platformu elektronických herbářů, které studenti tvořili. Stávající řešení odevzdávání digitálních herbářů se provádí skrze předání adresářové struktury dle šablony, což je velice nepraktické, chybové a neexistovalo jednoduché řešení, jak studentům tyto materiály zpřístupnit pro studium. Cílem je vytvořit informační systém, který by sdružoval všechny herbáře, umožnil by studentům jejich herbáře do tohoto systému odevzdávat, učitelům hodnotit a obecně tvořil platformu pro studium.

4.2 Cílová skupina služby

Cílovou skupinou jsou studenti základních a středních škol, kteří budou službu používat pro evidenci vlastních botanických herbářů. Tyto sbírky budou v případě školního určení hodnoceny a komentovány učitelem. Ještě před započetím práce bylo domluveno první nasazení na Všeobecném a sportovním gymnáziu Bruntál. Výhledovým cílem autora je prosadit nasazení práce i na dalších školách.

4.3 Výuková hodnota virtuálního herbáře

Přidaná výuková hodnota pro studenty, kteří budou využívat tuto práci, je v blízkém kontaktu s informacemi a prací s taxonomickým systémem. Není cílem práce vytvořit informační platformu, ale samotné přidávání položek, vyplňování informací (scheda) a zařazování rostlin vede k lepším schopnostem a zájmu studentů o botaniku. Z tohoto důvodu, kdy je základ v samotných informacích, které studenti vyhledávají k položkám, aby je mohli přidat, je kladen důraz na maximální upravitelnost systému ze strany učitele. Tento důraz ústí v dynamickou tvorbu zadávacího formuláře, kde si učitel může vytvořit jakoukoliv sadu polí k vyplnění a v dynamicky tvořený a udržovaný taxonomický systém.

4.4 Analýza stávajících řešení

Internetové služby zaměřené na botaniku a výuku již samozřejmě existují a na základě diskuze s učitelem biologie panem Mgr. Vladimírem Zatloukalem, který je v této práci v roli zákazníka, jsme vytvořili následující seznam existujících služeb:

- botany.cz
  - pozitiva
    - Služba má akademický přesah, protože kromě samotných fotografií obsahuje i odborné články.
    - Zařazení položky do kontextu článků.
    - Kvalitní vyhledávání.
  - negativa
    - Špatný vzhled a absence responzivního designu.
    - Nulová možnost zásahu od běžného uživatele.
    - Nelze mít vlastní účet či položky.
- kvetenacr.cz
  - pozitiva
    - Akademický přesah s kapacitami a odborníky.
    - Propracovaný taxonomický systém.
    - Kvalitní vyhledávání již na první stránce.
  - negativa
    - Nelze zobrazit fotografii v maximální velikosti.
    - Opět je uživatel pouze v pozorovací roli bez možnosti zásahu do systému.
    - Vzhled je zastaralý a design není responzivní.
- botanickafotogalerie.cz
  - pozitiva
    - Kvalitní nápověda a statistiky.
    - Široké zastoupení autorů z odborné sféry.
    - Možnost testování s vyhodnocením.
    - Fotografie s dodatečnými informacemi.
  - negativa
    - Opět je uživatel pouze v pozorovací roli bez možnosti zásahu do systému.
    - Vzhled je zastaralý a design není responzivní.
- biolib.cz
  - pozitiva
    - Z odborného hlediska nejdůvěryhodnější služba.
    - Možnost vlastního účtu a podílení se na obsahu.
    - Široké zastoupení autorů z odborné sféry.

    * Velice široká funkcionalita.
  - negativa
    * Absence nápovědy.
    * Nemožnost vlastního vkládání položek.
    * Vzhled je zastaralý a design není responzivní.
    * Nepropracované UX a přívětivost (zadávání ID uživatele jako adresáta zprávy).
    * Nelze vložit celou položku, ale pouze jednu fotografii.

Z výčtu stávajících služeb s jejich pozitivy a negativy vidíme, že žádná služba plně neposkytuje možnosti, které jsou žádané. Pouze jedna služba dovoluje vkládání vlastního obsahu, ale tento obsah se automaticky stává součástí celkového obsahu služby a nelze evidovat svoji vlastní sbírku pro vlastní potřeby. Zároveň všechny služby jsou svým návrhem, grafickým zpracováním a UX designem velice zastaralé a jejich ovládání není jednoduché.

4.5 Edukační cíle práce

Jak je uvedeno v sekci 4.4, neexistuje ve stávající situaci na české scéně řešení, které by pokrývalo potřeby studentských digitálních herbářů, kde by studenti shromažďovali své vlastní digitální botanické sbírky, učitelé by je hodnotili a tvořili by společně platformu pro výuku. **Přidaná edukační hodnota** vzniká v samotné práci studentů s biologickým systémem, taxonomií a vlastnostmi rostlin. Systém nemá ambici stát se informační základnou podobně jako služby zmíněné v 4.4, jelikož pro to nemá data a odborné uživatele, ale stát se platformou, kde studenti budou s rostlinami aktivně pracovat. Tyto požadavky byly shromážděny během dialogu s pedagogy, pro jejichž studenty je práce určena a sepsány do specifikace (příloha A).

4.6 Sběr požadavků a modelování

Sběr základních požadavků je klíčový pro řešení celého projektu. Snaha o realizaci projektu byla již v roce 2008, ale projekt skončil neúspěchem, byl nedokončen zrušen. Základní myšlenky a pochopení potřeb zákazníka ale bylo identické, proto sběr požadavků byl postaven na kritice původního modelu. Samotný sběr probíhal následovně:

- Osobní diskuze nad původním modelem, která vygenerovala seznam funkcionalit.
- Připomínkování funkcionality z řad dalších učitelů a studentů. Výstupem tohoto bodu je základní specifikace v příloze A.
- Modelování z datového pohledu na databázové schéma (DS) (příloha C).

Každá iterace individuálně reviduje specifikaci a tvoří Kano model (příloha B), který identifikuje požadavky. Výsledkem těchto modelů je celková funkční (příloha A) a datová (příloha C) specifikace. Je doporučeno nejdříve prostudovat specifikaci, jelikož v dalších kapitolách se bude pracovat s pojmy a konstrukcemi, které jsou definovány právě specifikací.

Kapitola 5

Výběr technologií pro implementaci

5.1 Klient

Na straně klienta (technologie zpracovávané prohlížečem) můžeme použít mnoho variant technologií. Pro tuto práci byly využity následující technologie:

- CSS3
- HTML5
- Twitter Bootstrap 3
- jQuery 2.1.4 (s dodatečnými knihovnami jako jQuery UI) - viz implementace
- font Awesome 4.3.0
- Google Fonts

Volba zmíněných technologií vycházela z široké oblíbenosti a použitelnosti. Projekt není natolik obsáhlý nebo interaktivní, aby bylo nutné použít například AngularJS nebo Google Closure. Naopak široká nabídka pluginů jQuery je velice žádoucí.

5.2 Databáze

Cílem použité databáze je uchovávat data o všech položkách, uživatelích, akcích apod. Definujeme řadu požadavků na princip uchovávání dat, jako je integrita, rychlost nebo spolehlivost. Proto se v následující kapitole budeme zabývat, jakou databázi použít a jaké mezivrstvy jsou vhodné k aplikaci.

5.2.1 Volba databáze

Technologie úložiště

MySQL, kterou jsme vybrali dříve, nabízí množství tzv. databázových úložišť[1]. Ve stávající situaci se nabízí dvě možnosti - MyISAM, nebo InnoDB. Hlavní výhoda MyISAM

[1] enginů

do posledních let byla vlastní implementaci tzv. fulltext indexu[2]. Bohužel MyISAM není schopna udržovat databázovou integritu skrze cizí klíče a neovládá potřebné transakce[10], které potřebujeme v metodě `persist()`. InnoDB naopak ovládalo transakce a cizí klíče, ale do do příchodu MySQL verze 5.6 nebylo možné použít index vyhledávání textu, který v testech výrazně urychloval vyhledávání. Jelikož tato práce používá novou verzi MySQL 5.6, je logickým krokem použití úložiště InnoDB.

Databázová mezivrstva

Na stranu databáze klademe mnohé požadavky. Jedná se zejména o udržení integrity, o rychlost odpovědí a jednoduchost integrace do projektu a vývoje. Zejména z důvodu posledního bodu požadujeme ORM mezivrstvu, která mapuje databázové řádky na objekty, zjednodušuje vývoj a dovoluje vyšší abstrakci reálného světa na data. Pro zvolenou aplikační vrstvu PHP s Nette se nabízí několik variant:

- **dibi**[3] - Nejedná se o ORM vrstvu, z kandidátů je dokonce na nejnižší úrovni, ale je nejjednodušší pro implementaci a použití.
- **NotORM** (Nette\Database)[4] - Databázová vrstva, kterou momentálně Nette obsahuje v základu. Stejně jako *dibi*, je i *NotORM* nadstavba nad PDO ovladačem, ale jak už sám název napovídá, nejedná se o ORM vrstvu.
- **doctrine**[5] - Obsáhlá databázová vrstva.
- **YetORM**[6] - Lehké a jednoduché ORM řešení.

Z jednotlivých popisů vidíme, že první dvě varianty vyřazujeme, jelikož se fakticky nejedná o ORM. Třetí *doctrine* vyřazujeme také, jelikož je velice obsáhlá na velikost projektu a vývoj v ní není natolik triviální, jak bychom potřebovali. Pro cílovou ORM vrstvu jsme zvolili YetORM, a to z několika důvodů:

- Je velice jednoduché pro implementaci a rychlost vývoje.
- Je to nadstavba nad NotORM (Nette\Database), tedy v případě, kdy by nabídka YetORM nebyla dostačující, můžeme použít obsáhlé možnosti Nette\Database.
- Jedná se o ORM vrstvu.

5.2.2 YetORM

YetORM se skládá ze dvou částí - **entity**, která mapuje řádek na objekt a **repositáře**, které provádí nad entitami operace.

Entita

Základ YetORM je mapování řádku na entitu. Každá entita je objekt, tedy může mít vlastní metody. Například položka je této práci definována jako entita Unit.

[2] index pro vyhledávání textu
[3] http://dibiphp.com/
[4] http://www.notorm.com/
[5] http://www.doctrine-project.org/
[6] https://github.com/uestla/YetORM

Kód 5.1: Příklad entity Unit z práce (/app/model/Entites/Unit.php)

```
1  /**
2   * Class Unit
3   * @package App\Model
4   * @property-read int $id
5   * @property int|NULL $taxonomy -> taxonomy_id
6   * @property int $rating -> rating
7   * @property string $comment -> teacher_comment
8   * @property int $device -> device_id
9   */
10 class Unit extends Entity
11 {
12     public function getUser()
13     {
14         return $this->record->related("unit_directory")->fetch()->
               directory->user_id;
15     }
16 }
```

Jak můžeme vidět v kódu 5.1 (řádek 4-8), každá entita definuje v komentářové anotaci atributy dané entity (fakticky to mapuje sloupce v tabulce) s datovými typy a případnými překlady (například z podtržítkové notace (underscore) do CamelCase). Přítomnost definice datových typů může částečně nabourávat princip dynamicky typovaného jazyka PHP, protože je nutno předkládat data ve správném přetypovaném tvaru. Nad těmito entitami existují repositáře, které s nimi provádějí operace.
A jak můžeme vidět v metodě `getUser()`, můžeme využít zpětné kompatibility s NotORM v použití metody `related()` nebo `fetch()`.

Repositář

S entitami pracují repositáře popsané v kódu 5.2, kterým je opět skrze komentářovou notaci definována entita (řádek 4) a tabulka (řádek 3). Kromě jiných metod obsahuje repositář dvě základní sady metod `findBy*` a `getBy*`[7]. V obou případech se jedná o selektivní metody, které vytahují řádky z databáze (`getBy*` vrací jeden řádek dle zadané podmínky, `findBy*` vrací množinu řádků neboli kolekci entit). Základem logiky těchto metod je možnost zadání kritérií (primárně podmínek klauzule WHERE). Máme tři následující možnosti:

- Použít `getBy([criteria])` - V tomto případě zadáváme podmínky do pole `criteria`. Zadání funguje jako asociativní pole, tedy `[sloupec => hodnota]`. Tuto variantu používáme v případě, že chceme aplikovat vícenásobné či složitější podmínky (řádek 22).
- Použít `getBy<Property>(<value>)` - V tomto přístupu definujeme atribut, dle kterého chceme vyhledávat entity přímo v názvu metody skrze `<Property>` a hodnotu jako argument této metody. Například `getByName($name)`. `<Property>` musí být dle definice atributu entity dle 5.1.
- Zabudované metody - Můžeme použít již předdefinované metody jako `findByAll()` pro výběr všech řádků z tabulky či `getByID()` pro výběr řádku dle ID.

Princip zadávání podmínek metod `findBy*` a `getBy*` je analogický.

[7] sady metod, které úpravami nabývají různých vlastností, viz dále

Kód 5.2: Příklad repositáře z práce (/app/model/Repositories/UnitRepository.php)

```
1  /**
2   * Class UnitRepository
3   * @table unit
4   * @entity Unit
5   */
6  class UnitRepository extends Repository
7  {
8    public function findByDirectory($id){
9      if($id == NULL)
10     {
11       $data = $this->findAll();
12     }
13     else
14     {
15       $data = $this->findBy([':unit_directory.directory_id'=>$id]);
16     }
17     return $data;
18   }
19   public function saveRating($data){
20     $unit = $this->getById($data->unit);
21     $unit->comment = $data->comment;
22     $unit->rating = (int)$data->rating;
23     $this->persist($unit);
24     return true;
25   }
26 }
```

Ukládání

Máme-li entitu, kterou chceme uložit (vytvořená buď skrze metody výše nebo jako nová entita), použijeme metodu `persist(<entity>)` (kód 5.2 řádek 26). Tato metoda se používá pro úpravu i vytvoření záznamu. Jedna z velice přívětivých vlastností YetORM je jeho upravitelnost. Jednotlivé metody lze jednoduše přepsat, ať už v repositáři, chceme-li například provést složitější operaci při zápisu (přepíšeme v daném repositáři metodu `persist(<entity>)`), nebo i v jádru YetORM. Kromě integrity a jednoduchého použití zabezpečuje metoda `persist()` a celé YetORM ještě jednu důležitou databázovou vlastnost - **transakci** [10].

5.3 Aplikační vrstva

Aplikační vrstva poskytuje samostatnou funkční logiku systému - manipuluje s daty, opravňuje uživatele zapisovat a číst nebo automaticky provádí naplánované operace, jako bude třeba kontrola plagiátů.

5.3.1 Volba technologie strany serveru

Stejně jako pro databázi, tak i pro aplikační vrstvu serveru definujeme, jisté požadavky a máme omezené možnosti. Z důvodu volby linuxové platformy se nabízí několik variant jazyků k implementaci Java, PHP, Python nebo například Ruby. Jelikož projekt, který je obsahem této práce, není tak obsáhlý (minimálně v cílové skupině uživatelů) a s ohledem

na rozšířenost jazyků pro dané využití (Python, Perl), bylo zvoleno PHP a pro něj nejpopulárnější [5] webový server Apache 2.4.7.

5.3.2 Volba Nette Framework

Pro PHP opět existuje množství frameworků, které ulehčují práci a zefektivňují vývoj. Pro výsledný výběr byly klíčové dva aspekty.

- dokumentace (pro získání maxima informací bez nutnosti číst kód)
- uživatelská základna a oblíbenost (generující doplňky, manuály a řešené problémy)

Základem rozhodnutí byla statistika na sitepoint.com[8], ve kterém první tři místa obsadily produkty Laravel, Symphony2 a Nette. Vzhledem k českému původu a lokalizaci dokumentace do českého jazyka byl zvolen pro implementaci projektu Nette Framework[9].

5.3.3 Nasazení technologií

Jak již bylo řečeno, celý projekt je webový a založen na PHP s frameworkem Nette. Jako první krok se tedy nabízí instalace webového serveru, ideálně v kombinaci LAMP[10]
Instalace základního Nette byla značně zjednodušena balíčkovacím systémem *composer*[11] a jeho integrací do vývojového prostředí PHPStorm[12]. Již na začátku bylo možné definovat základní čtyři balíčky, které budou pro projekt klíčové.

- `nette/nette` - základ Nette
- `uestla/yetorm` - YetORM vrstva
- `nette/extras` - rozšiřující balíčky Nette
- `instante/bootstrap-3-renderer` - doplněk vykreslující formuláře pomocí Bootstrap knihovny

5.4 Struktura Nette

Framework Nette využívá MVC[13] architekturu[9], která odděluje obsluhu požadavků, aplikační logiku a zobrazovací vrstvu, čímž se dosahuje čistšího návrhu, jednodušší implementace a efektivnější rozšiřitelnosti projektu. Strukturu Nette a jeho řešení MVC si ukažme na schématu 5.4.

Z ze struktury vidíme, že prvky integrovaného YetORMu (5 a 6) se řadí do *modelu*, protože manipulují s daty a kromě tříd **UserManager** a **UploadHandler** nemáme žádné zvláštní další třídy modelu, protože se o celkovou manipulaci s daty stará YetORM.

[8] http://www.sitepoint.com/best-php-framework-2015-sitepoint-survey-results/
[9] http://nette.org/cs/
[10] kombinace Linux, Apache2, MySQL a PHP, většinou jako metabalíček. v GNU/Linuxu
[11] https://getcomposer.org/
[12] https://www.jetbrains.com/phpstorm/ použito ve studentské licenci
[13] Model-View-Controller

```
herbarium/
|-- app/                        (1)  adresář logiky aplikace
|   |-- config/                 (2)  konfigurační soubory
|   |-- forms/                  (3)  továrny pro formuláře
|   |-- model/                  (4)  adresář s logikou modelu
|   |   |-- Entities/           (5)  entity YetORM
|   |   |-- Repositories/       (6)  repositáře YetORM
|   |   |-- UploadHander.php    (7)  model pro upload souborů
|   |   |-- UserManager.php     (8)  model správy uživatelů
|   |-- presenters/             (9)  presentery (controler) a šablony (view)
|   |   |-- templates/          (10) šablony (view)
|   |   |-- BasePresenter.php   (11) jednotlivé presentery
|   |   |-- ErrorPresenter.php
|   |   |-- HomepagePresenter.php
|   |   |-- ImagePresenter.php
|   |   |-- LoginPresenter.php
|   |   |-- PublicPresenter.php
|   |   |-- StudentPresenter.php
|   |   |-- TeaceherPresenter.php
|   |-- router/                 (12) adresář pro konfiruraci URL adres
|   |-- bootstrap.php           (13) soubor se spuštěním aplikace
|-- temp/                       (14) složka s~dočasnými soubory a cache
|-- vendor/                     (15) knihovny: nette, bootstrap-3-render...
|-- www/                        (16) další soubory webu (JS, obrázky, CSS...)
|   |-- index.php               (17) spouštěcí soubor
|-- data/                       (18) adresář pro ukládání dat systému
```

Ukázka 1: Adresářová struktura práce

Každá logicky oddělená sekce má vlastní *presenter* (prakticky *controler*), který však může používat libovolný *model*. Presenter obsahuje v základu několik typů metod, které reprezentují životní cyklus presenteru.

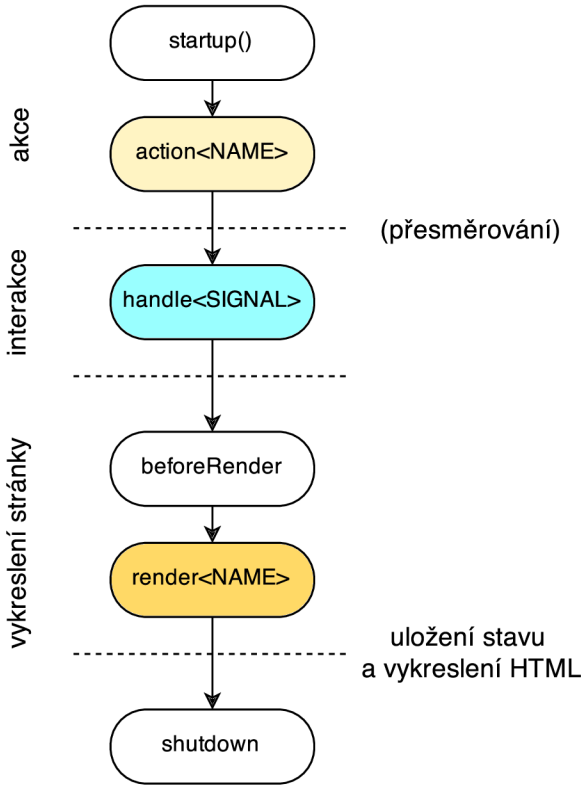
V praxi se využívají termíny jako *handler*, *render* nebo *action*[14] presenteru. Jedná se o metody a prakticky fáze životního cyklu presenteru, který reaguje na uživatelský požadavek. Jak vidíme na obrázku 5.1, volají se v následujícím pořadí s těmito významy:

- `startup()` - Provádí inicializaci nebo základní nastavení.
- `action<NAME>()` - Vykresluje data do šablony. Rozdíl proti *render* je v době volání a cíleném užití. Často se využívá pro kontroly a přesměrování (přihlášení atd.) nebo pro sdílení šablony.
- `handle<SIGNAL>()` - Vpracuje daný `<SIGNAL>`, který je argumentem URL v parametru **do**[15]. Používá se často pro AJAXovou komunikaci nebo jiná API.
- `render<NAME>()` - Obsahuje konečné finální vykreslení stránky s pomocnými metodami `beforeRender()`.

[14] terminologie Nette viz dále

[15] parametr GET metody ve tvaru `?do=<SIGNAL>`

Pro každý *render* nebo *action* musí existovat *latte*[16] šablona se stejným názvem jako `<NAME>` u metody *render* či *action*, pokud není explicitně uvedeno jinak[17].



Obrázek 5.1: Životní cyklus presenteru v Nette

[16]šablonovací systém pro Nette - https://github.com/nette/latte
[17]v action či render můžeme explicitně nastavit, že bude používat jinou šablonu či render

Kapitola 6

Implementace

V kapitole 3.2 jsme definovali iterativní vývoj tohoto projektu a jeho iterace. Celkově je projekt rozdělen do 13 iterací, kde většina z nich bude obsahem této kapitoly. Popsané budou pouze iterace, které jsou klíčové a přínosné. Každá iterace se skládá z následujících čtyř kroků, kterými každá iterace prošla.

1. **Analýza** - Definice, co by tato část systému měla umět. Identifikujeme požadavky dle Kano modelu - příloha B.

2. **Návrh** - Datová abstrakce požadavků. Definujeme na základě databázového schématu (příloha C), Entit, Repositářů, případně další datovou nebo objektovou strukturou.

3. **Implementace** - Detailní popis implementace s použitými metodami se snahou o eliminaci triviálních popisů a zaměření na zajímavé implementační postupy. Často jsou pro lepší demonstraci vloženy sekvenční diagramy, které ideálně znázorňují interakci mezi komunikujícími objekty a zařazují tyto události do časového vývoje[1].

4. **Testování** - Pokud není uvedeno jinak, je testování řešeno uživatelsky autorem. Kromě testování autorem po vývoji byl systém dle scénářů testován i stranou zákazníka - většinou s osobní konzultací, zda výsledek opravdu splňuje dané požadavky. Po celkové implementaci bylo provedeno uživatelské testování celého systému jako celku, které je popsáno v kapitole 7.

Před následujícími implementačními iteracemi proběhly celkově tři přípravné iterace. Jednalo se o celkovou analýzu a sběr požadavků (kapitola 4), modelování 4.6 a nasazení použitých technologií 5.3.3, tak jak můžeme vyčíst z Ganttova diagramu sekce 3.2.

6.1 Správa uživatelů

Jako první krok implementace je práce s uživateli, jejich autentizace, autorizace apod.

Analýza

Pro analýzu využíváme zmíněný Kano model, ve kterém také definujeme, které atraktivní požadavky se budou realizovat (mají dostatečný poměr nároky/výsledky).V případě Kano modelu správy uživatelů B.2 shledáváme většinu požadavků k realizaci, kromě přihlášení třetí stranou (Facebook, LinkedIn, G+ apod.). Dle základního dotazování cílových uživatelů o tuto možnost přihlášení není dostatečný zájem v porovnání s pracností.

Návrh

Datový návrh popisuje databázové schéma (příloha C) v bloku UŽIVATELÉ. Princip zdánlivě zbytečné vazby N:M mezi tabulkami *grade* a *user* je ve faktu, že uživatel může být i učitel, který může mít správu více tříd, což generuje vazbu N:M.
Práce s uživateli je z pohledu **presenteru** členěna do tříd:

- `LoginPresenter` - autentizace, registrace, obnova hesla, uživatelská editace profilu
- `TeacherPresenter` - administrační práce s uživateli a třídami

Správa uživatelů z pohledu **modelu** je velice specifická část systému, tedy nemá pouze repositář na straně modelu jako většina modulů, ale také třídu `UserManager`, která skrývá většinu funkcionality a samostatné repositáře `UserRepository` a `GradeRepository`. Entity tohoto repositáře jsou `User` a `Grade`. Anglický ekvivalent pro třídy/ročníky **Class** byl při volbě jmen zavržen z důvodu možné zaměnitelnosti s klíčovým slovem/pojmem *Class* například v dokumentaci, komentáři nebo specifikaci.

Implementace

Z implementačního hlediska je repositář `UserRepository` specifický zejména tím, že implementuje vlastní autentizační metodu `authenticate(array $credentials)`[1] pro ověření uživatelů. Tato metoda přijímá pole s emailem a heslem uživatele, porovnává je s údaji v databázi a v případě shody vytváří objekt `Identity`, který identifikuje uživatele po celou dobu jeho sezení.

Dále `UserRepository` přepisuje metodu `persist(Entity $user, $grade_id = NULL)`, protože kromě samotného uložení uživatele zařazuje uživatele do tříd (tabulka `grade`), a tedy tvoří vazby v tabulce **grade_user**.

`UserManager` nemanipuluje s daty na úrovni databázové (od toho je repositář), ale generuje množství jiných akcí, proto byla tato třída oddělena z repositáře. Vytváří například registrace, pracuje s profily nebo zajišťuje obsluhu požadavků na resetování hesla. Jelikož se ale jedná o model a v modelu obecně není přístupné prostředí pro generování odkazů, předává se výsledek akce do *presenteru*, kde se odesílají všechny emaily, které jsou v této části důležité (potvrzení registrace, generování nového hesla...).

Správa uživatelů zasahuje do tří částí systému, do tří presenterů s příslušným cílovým uživatelem a jeho oprávněním:

- `LoginPresenter` - veřejnost (nepřihlášen) - registrace, přihlášení a obnovení hesla
- `TeacherPresenter` - učitel (přihlášen) - správa tříd
- `StudentPresenter` - student (přihlášen) - změna profilu. Zobrazení profilu zajišťuje stejný presenter, pouze připouští i nepřihlášený vstup.

6.2 Nastavení systému

Celý informační systém má určitá vlastní nastavení, která jsou klíčová pro jeho chod. Rovinu nastavení systému ale dělíme na dvě části:

[1] implementuje *IAuthenticatior*, z modelu *\Nette\Security\IIdentity*, který zajišťuje autentizaci a sezení

- **technické** - interní nastavení serverů, hesel apod.
- **uživatelské nastavení** - Předchystání systému pro uživatele. Jedná se o nastavení taxonomie, se kterou studenti následně pracují a formulář popisující práci s položkami.

Technické nastavení

Nastavení provozu systému definujeme v konfiguračních souborech Nette, které využívají knihovnu a syntaxi Neon[2]. V konfiguračním souboru **config.neon**, případně v lokální kopii **config.local.neon**, definujeme nastavení připojení k databázi, registraci tříd a další nastavení. Kromě tohoto interního nastavení existují v různých částech systému některé prvky, které se mohou v čase měnit a je tedy vhodné je udržovat v konfiguracích, nikoliv je interně vkládat do kódu. Jedná se například o nastavení serveru pro kontrolu plagiátů, o nápověda apod. Pro tyto účely vzniká modul **Setting** obsahující repositář `SettingRepository` a entitu `Setting`. Repositář `SettingRepository` obsahuje pouze jednu metodu `getValue(key)`, která vrací hodnotu příslušnou ke klíči `key` z tabulky `setting`.

Uživatelské nastavení

Nastavení z pohledu uživatele je následující:

- Nastavení taxonomického (biologického) stromu, do kterého studenti vkládané položky zařazují.
- Nastavení formuláře, jehož prvky definují položku a veškeré informace o ní.

Obě tyto nastavení musí být dynamicky definovatelné dle požadavků Kano modelu B.4.

Analýza

Ačkoliv je tento modul ve dvou rovinách, je stejného charakteru a je sloučen do jedné iterace a jednoho Kano modelu B.4.

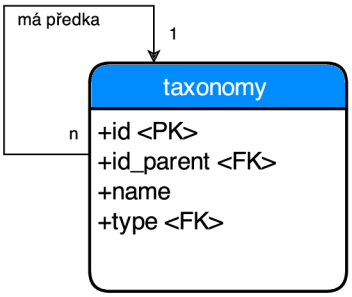
Návrh taxonomie

Taxonomie je realizována dvěma tabulkami - *taxonomy* a *taxonomy_type* (blok TAXONOMIE v DS přílohy C), ale je pro ni definována pouze jedna entita **Taxonomy** s repositářem **TaxonomyRepository**, protože použití tabulky *taxonomy_type* je tak minimální, že není potřeba definovat pro ni vlastní entitu či repositář. Jelikož se snažíme o realizaci n-ární stromové struktury, je nutno vyřešit problém definování stromové hierarchie v databázi. Tento problém řešíme dle nákresu obrázku 6.1. Tedy řešíme strom v podobě jedné tabulky, ve které řádek je uzel a která obsahuje cizí klíč na sebe samotnou. Tedy atribut **id_parent** je ukazatelem na rodiče. Pokud je **id_parent** roven nule, pak se jedná o kořenový uzel. Tento strom je obecný a závislý na nastavení uživatelem, tedy nemůžeme nijak zaručit jeho maximální šířku, výšku nebo vyváženost. Průchod stromem má smysl pouze do hloubky a využívá se metoda *INORDER* (viz implementace).

Návrh formuláře

Návrh formuláře je datově modelován v DS v bloku *FORMULÁŘ*. Tabulka **attribute** je základem formuláře a její záznam definuje jeden prvek formuláře, který může být textové pole (input), textová oblast (textarea), rozbalovací nabídka (selectbox) nebo soubor.

[2] http://ne-on.org/



Obrázek 6.1: Návrh stromové struktury taxonomie

Všechny prvky mohou mít různá nastavení - název, ukázková hodnota (placeholder) apod. Pouze v případě, že se jedná o rozbalovací nabídku, musíme k ní definovat seznam voleb (options). Dochází tak k vazbě 1:N a využíváme tabulku **attribute_option**, která obsahuje pouze zmíněné volby pro rozbalovací nabídku. Prvky formuláře jsou realizovány entitou **Attribute** a pracuje s nimi repositář **AttributeRepository**.

Implementace

Implementace taxonomie

Z pohledu presenteru je celá tato iterace řešena presenterem **TeacherPresenter**, který spojuje akce, které jsou primárně určeny pro učitele.

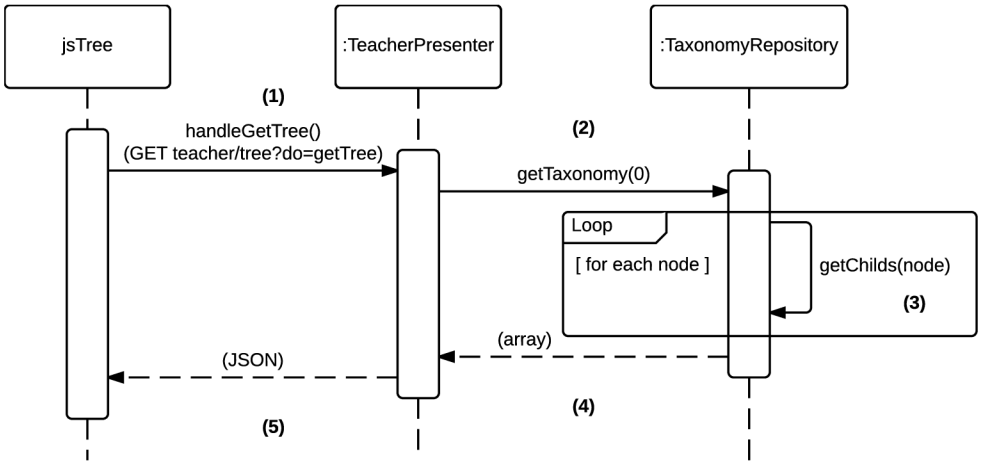
Zobrazení taxonomického stromu a práci s ním popisujeme v diagramu 6.2. Na straně klienta je řešena JavaScriptovou knihovnou jsTree[3]. Ta ve své inicializaci vyžaduje vstupní data ve formátu JSON, která získává skrze GET **(1)** požadavek na `handleGetGtree()` z presenční třídy **TeacherPresenter**. Tento *handler* si pouze požádá o data z repositáře **TaxonomyRepository** (viz dále), vykreslí je v JSON podobě **(5)** skrze `json_encode()`[4] a ukončí skript.

Získání stromu

Získání stromu v **TaxonomyRepository** poskytuje metoda `getTaxonomy($id)` **(2)**, která může vrátit celý strom, zadáme-li počáteční kořen roven nule, nebo část podstromu, pokud je volána s ID daného uzlu, který je v podstromu kořenem. Tato metoda volá metodu `getChilds($parentId,$lvl,$do = NULL)` **(3)**, která se rekurzivně zanořuje do stromu a vrací podstromy v podobě asociativního pole, které reprezentuje strom (a je snadno převeditelné do *JSON* notace) **(4)**. Princip, kterým tato metoda prochází strom, je identický s principem *INORDER* průchodu stromem.

Během průchodu stromem se na vracený uzel mapují data z tabulky **taxonomy_type** skrze vazbu v klíči **type**. Tento typ reprezentuje úroveň zanoření. Například nejnižší biologický taxon je druhový název, druhý nejnižší je rodový název. Postupujeme až k říši, která je nejvyšší.

[3] http://www.jstree.com/
[4] http://php.net/manual/en/function.json-encode.php



Obrázek 6.2: Sekvenční diagram zobrazení taxonomického stromu

Cesta stromem

Na jednotlivé uzly stromu se následně ze studentského hlediska mapují/zařazují položky. Pro výpis zařazení je nutno generovat cestu od zvoleného uzlu až po kořen. Díky faktu, že si uzel uchovává informaci o rodiči, nikoliv rodič o potomkovi, je velice jednoduché generovat cestu. Stačí postupně stoupat po uzlech až ke kořeni a cestou si zaznamenávat cestu. Přesně tak to dělá metoda `getPath($id)` z **TaxonomyRepository**. Jediný nepříjemný bod tohoto postupu je fakt, že cílem je mít cestu od kořene k uzlu, ne od uzlu ke kořenu. Z tohoto důvodu je nutno sesbírané body cesty na konci revertovat. Metoda **getPath()** to řeší ukládáním cesty do pole a následným voláním funkce `array_reverse()`[5].

Implementace formuláře

Nastavení položkového formuláře není náročné implementačně, ale datově. Každý prvek má množství nastavení a tato nastavení se větví dle typu prvku. Problém tedy nastává až na straně klienta, kde je potřeba dynamicky upravovat stav formuláře dle jeho voleb skrze JavaScript.

Stejně jako v případě pořadí typů taxonů, tak i u prvků formuláře záleží na pořadí. Nastavení pořadí položek se řeší na straně klienta principem *táhni a pusť*[6] v tabulce výpisu prvků. O aktivaci *táhni a pusť* se stará metoda `sortable()` z JavaScriptové knihovny **jQuery**. Metodu aktivuje a následné posílá data na server funkce `initSort()`, která inicializuje `sortable()` nad tabulkou, a `sendOrderToServer()`, která serializuje ID jednotlivých řádků tabulky (srkze `jQuery.sortable(serialize)`) a následně je posílá na server, kde je zpracovává *handler*. Navržení tohoto procesu je obecné pro použití řazení ve všech sekcích. Tedy URL, na kterou se zasílají data, je definována přímo v **data-** atributu tabulky.

Následné vykreslení formuláře se provádí stejně jako ve všech jiných případech skrze modelové továrny formulářů (*FormFactory).

[5] http://php.net/manual/en/function.array-reverse.php

[6] Drag and drop

6.3 Správa položek

Analýza

Správa položek je základ projektu. Primární akce studentů souvisí právě s položkou - její vložení, editace, smazání, nastavení viditelnosti a zobrazení (ať už vlastní či cizí). Většina požadavků na správu položek patří do skupiny základní či výslovná, ale i zde byl proveden Kano model B.3.

Při analýze a tvorbě Kano modelu B.3 narážíme na zajímavý jev. Vyzněl na povrch požadavek **možnost diskuze nad položkami**, který patří do skupiny atraktivních, ale zasahuje danou iteraci natolik okrajově, že je vhodné tuto část systému oddělit do zvláštní iterace 6.7. Tedy na základě Kano modelu definujeme novou zvláštní iteraci vývoje.

Návrh

Už v databázovém schématu vidíme, že se správa položek skládá ze dvou částí. Figurují v něm samotné položky a složky (herbáře), do kterých se položky seskupují. Důvod tohoto chování je sepsán ve specifikaci DS (příloha C) v bloku *STUDENTSKÉ POLOŽKY*. Jelikož je vysloveným požadavkem sdílet položku mezi složkami, dochází k vazbě N:M.

Jelikož formulářové prvky, které definují data samotné položky, se tvoří dynamicky a jsou měnitelné v čase, neuchovávají se informace o položce přímo v tabulce **unit**, ale v tabulce **unit_attribute**. Obsahem této tabulky jsou reference na položku a prvek formuláře (tabulka **attribute**) a jako místo pro uložení hodnoty existuje sloupec **value**. Data tohoto sloupce dostávají odlišnou sémantiku podle typu formulářového prvku, kterému náleží. Jedná-li se o typ **soubor** (file), nebo **rozbalovací nabídka** (selectbox), je tato hodnota brána jako odkaz do tabulky **file**, nebo **attribute_option**, na řádek, který zvolená data reprezentuje (odkaz na soubor, nebo volba v rozbalovací nabídce). Vzhledem k použití se jedná o číslo, které odkazuje do jiné tabulky, tedy prakticky o cizí klíč, ale není mu nastavena vazba ani závislost, tedy nedodržuje databázovou integritu. Databázová integrita je ale dodržena databázovým triggerem[7], který kontroluje existenci referencí a simuluje tak vazby cizího klíče. Dalo by se namítnout, že se nejedná o čisté řešení, ale vzhledem k flexibilitě dat, která mohou nabýt hodnoty **value**, by musel být pro každou referenci (ve stávající chvíli pro dvě zmíněné), která by měla odkazovat do cizí tabulky, dle typu formulářového prvku (soubor či rozbalovací nabídka) vytvořen zvláštní sloupec s cizím klíčem. Což při flexibilitě systému a následném možném rozšíření (nové typy hodnot - zaškrtávací políčko *checkbox*, přepínač *radiobutton*, PDF příloha. . .) by to znepřehledňovalo situaci. Položka i herbář mají vlastní entity `Unit` a `Directory` a repositáře `UnitRepository` a `DirectoryRepository`.

Implementace

Entita je v YetORM objekt a jejím hlavním cílem je reprezentovat řádek tabulky jako záznam, což ale neznamená, že bychom nemohli entitě dopsat metody, naopak je to velice vítáno. Vzhledem k faktu, že `Unit` má množství vazeb, agregací a vlastních funkcionalit, obsahuje třída `Unit` množství metod, které tyto funkcionality pokrývají. Například není příliš jednoduché dostat se k vlastníkovi položky, jelikož zmínka o vlastníkovi je pouze ve složce, do které položka spadá, což při N:M vazbě není pohodlné dělat explicitně všude. Proto tvoříme metodu `getUser()` jako metodu entity **Unit** pro snazší práci v dalších částech systému.

[7] triggery **file_Tr** a **attribute_option_Tr**

Zároveň `UnitRepository` je opět další repositář, který si přepisuje vlastní zapisovací metodu `persist(Entity $unit, array $dirIds = NULL)`, poněvadž při tvorbě položky je nutno tvořit i N:M vazbu s **Directory**.

`UnitRepository` nabízí ještě jednu ojedinělou funkčnost - **vyhledávání**. Jediné, co lze v systému vyhledávat, jsou položky, a to pouze nad prvky, které to mají nastaveno (definice při tvorbě formuláře). Není tedy aplikován žádný vyhledávací nástroj jako například elasticsearch[8], ale vyhledává se pouze pomocí MySQL operátorů `MATCH()...AGAINTS`[10] ve sloupcích, které definoval učitel. Pro vyhledávání slouží metoda `search($data,$own)`. Jelikož využíváme InnoDB (viz volba úložiště 5.2.1), můžeme použít textový index a operátory `MATCH(<sloupce>) AGAINST(<hodnoty>)`. Jako `<sloupce>` rozumíme sekvenci čárkou oddělených názvů sloupců tabulky a jako `<hodnoty>` chápeme řetězec s možností použití technik regulárních výrazů[9].

O vkládání nebo editaci položek se stará metoda `manage($data)`, která dle výskytu editovaného **ID** položky vyhodnocuje, zda se jedná o akci vložení nebo editace. Tento přístup je použit v celém systému. Vkládání položky je klíčovým bodem celého systému a pro uživatele se nejedná zrovna o primitivní formulář. Proto byl kladen vysoký důraz na UX formuláře, což se ukázalo klíčové při výsledném uživatelském testování v kapitole 7. Vkládání položky je jednokrokové, i když obsahuje tři části - obsahuje vložení textových dat, nahrání souborů a zařazení položky. Upload souborů je tématem kapitoly 6.4.

6.4 Nahrávání souborů

S vkládáním položek přichází i nahrávání fotografií jako klíčový prvek systému. Proto byl vyvinut silný tlak na uživatelské pohodlí a UX tohoto ovládacího prvku.

Analýza

Při analýze tvoříme opět Kano model B.1, ze kterého je více než jasné, že je kladen důraz na jednoduché ovládání této uživatelsky relativně náročné operace. Zároveň je nutno myslet na fakt, že se budou vkládat velké soubory, většinou nekomprimované fotografie ve vysokém rozlišení a při běžném studijním použití může jeden student vygenerovat až kolem 200 fotografií. Nahráním fotografií správa souborů nekončí, ale může přijít připojování dokumentových příloh, videí apod., které by mělo již být v souladu s tvořeným návrhem, a proto bude nutno vymyslet univerzální řešení. Soubory budou velké, tedy nemůžeme spoléhat na běžný formulářový prvek a na základní nastavení PHP s jeho hodnotou *upload_max_filesize* [12], ale musíme použít jiné řešení, které si bezpečně poradí i se soubory ve velikosti desítek MB[10]. Z tohoto popisu plynou následující prvky:

- spolehlivé nahrání mnoha velkých souborů
- důraz na UX celé komponenty
- dobrý návrh pro další rozšíření

[8] https://www.elastic.co/

[9] operátory +,*,-,~ nebo běžnou konstrukci regulárního výrazu za použití předpony **REGEXP**

[10] megabytů

Návrh

Z důvodu univerzálního návrhu tvoříme tabulku **file**, která bude všechny soubory evidovat, samostatnou entitu `File` a repositář `FileRepository`. Soubory budou ukládány s unikátně generovaným názvem (s respektováním přípony) v adresářích náležících pouze vždy jednomu uživateli, tedy v cestě `/data/<UserID>/<NAME>.přípona`. Tímto způsobem řešíme problém unikátního názvu, limitu souborů v jednom adresáři (dle použitého systému souborů) a logického rozdělení. Tabulka **file** obsahuje atributy názvu souboru a vlastníka, což stačí k definování cesty. Kromě toho obsahuje atribut **checked**, který signalizuje, zda soubor prošel kontrolou plagiátů, která je implementována iterací v sekci 6.6.

Implementace

Nahrávání souborů na straně klienta implementujeme knihovnou FineUploader[11], která má velice dobře řešený šablonovací systém pro nastavení jejího zobrazení, podporuje pohodlné nahrávání souborů skrze princip *táhni a pusť* a má velice dobře zdokumentovanou konfiguraci. Jak můžeme vidět v kódu 6.1, celá inicializace je implementována tak, aby bylo možné použít modul na více místech systému, jako je například nahrání profilového obrázku profilu.

Kód 6.1: Inicializace a nastavení nahrávací knihovny (/www/js/main.js)

```
var url = $(this).data("url");
var delurl = $(this).data("delurl");
$(this).fineUploader({
  debug: false, //for debug
  request: {
      endpoint:  url //where send data to?
  },
  retry: {
      enabled: true
  },
  deleteFile: {
      enabled: true,
      endpoint: delurl //url where to delete the uploaded file
  },
  cancel: {
      enabled: true
  },
  validation: {
      acceptFiles: 'image/*', //set datatypes to enable to upload
      allowedExtensions: ['jpe', 'jpg', 'jpeg', 'png']
  }
})
```

Pro samotný upload souboru strana klienta ale nestačí. Proměnné `url` a `delurl` musí vést na *handler* presenteru, který obslouží přijímaná data na straně serveru. V případě nahrání fotografie položky je použit handler `handleUploadPicture($attId)`, jak můžeme vidět na kódu 6.2.

[11] http://fineuploader.com/

Kód 6.2: Metoda obsluhy nahrávání souboru (/app/presenters/StudentPresenter.php)

```
1  /**
2   * handler to upload img for unit . Using "uploadHandler" from
       model
3   * @param $attId
4   */
5  public function handleUploadPicture($attId) {
6    $uploader = new UploadHandler($this->fileRepository);
7    $uploader->allowedExtensions = array("jpeg","jpe","jpg","png");
8    $result = $uploader->handleUpload(__DIR__ . '/../../data',$this
       ->getUser()->id);
9    if(isset($result["name"]))
10   {
11     $row = $this->fileRepository->insertFile($result["name"],$this
         ->getUser()->id);
12     $result["idFile"] = $row->id;
13     $result["attId"] = $attId;
14   }
15   else
16   {
17     $result["err"] = "err";
18   }
19   $this->sendResponse(new Nette\Application\Responses\JsonResponse
       ($result));
20 }
```

Na kódu 6.4 můžeme vidět použití modelu `UploadHandler`, který zajišťuje finální přenos dat a uložení souboru. Vrací data o souboru, kterým je například název, který je unikátně generován jako kontrolní součet. Na řádku 11 potom vidíme zápis souboru do databáze. Velká výhoda této knihovny je její JSON zpětná vazba zpět klientovi (řádek 19). Skrze toto rozhraní se dá knihovna rozšířit nebo ladit.

6.5 Práce se soubory a cache

Tato iterace není přímo zaměřena na uspokojení uživatelských požadavků, protože s výsledky nepřijde uživatel přímo do interakce, a proto nemá definován Kano model.

Analýza

Cílem této iterace je vyřešení problému, jak můžeme pracovat s obrázky, které jsou nahrány ve vysokém rozlišení, ale pro potřeby aplikace je nutné je zobrazovat v malých náhledech s co nejmenší režií rychlostní i kapacitní.

Návrh

Na zmíněnou otázku nacházíme následující alternativy řešení:

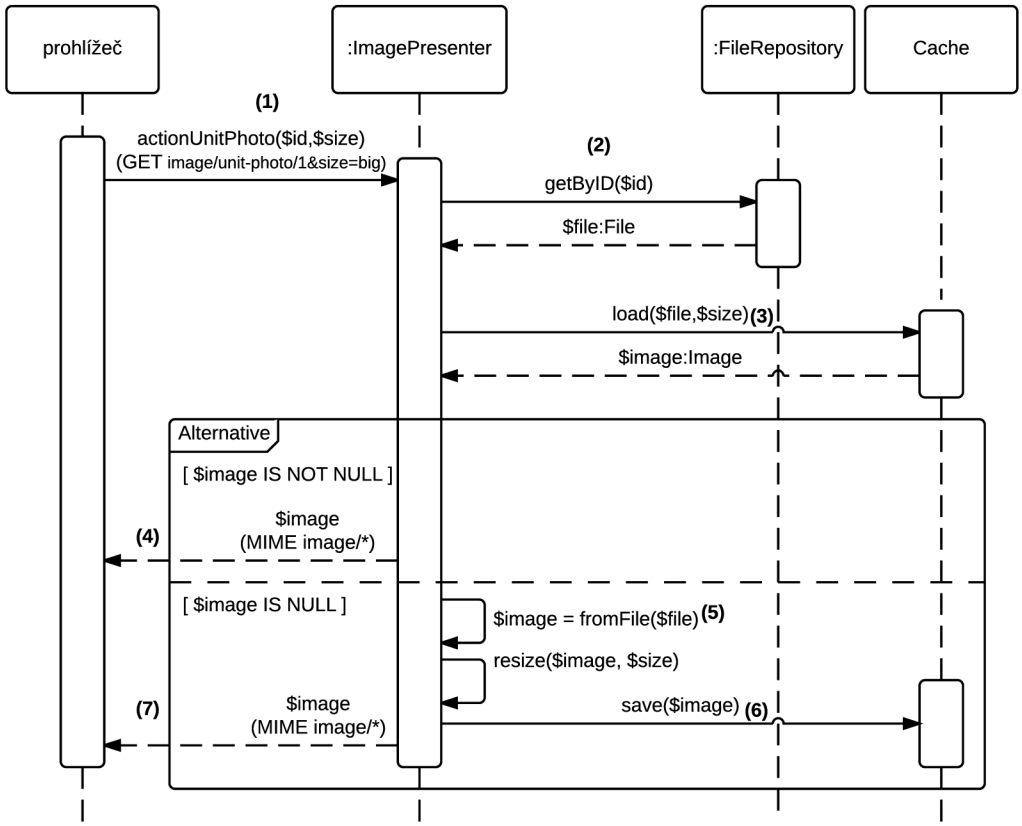
- Při nahrání na server můžeme fotografie zmenšit na požadované varianty rozměrů, ty následně uložit v adresářové struktuře. Jedná se o triviální řešení, které ale zbytečně znásobuje kapacitní potřeby systému.
- Soubory zmenšovat pouze na požádání a neukládat jea nulový kapacitní nárůst dostáváme ale výrazné snížení rychlosti. Zmenšení obrázku je ale velice pomalé a máme-li

například přehled položek, kde každá položka může být uvozena záběrem, pak se jedná o výkonnostní problém.

- Soubory můžeme zmenšovat na požádání, ale využít cache[12]. Toto řešení se nakonec ukázalo jako nejčistší. Využíváme cache, do které ukládáme již zmenšené soubory, ale nové náhledy či miniatury generujeme pouze v případě potřeby.

Implementace

O samotné poskytnutí obrázků se stará samotný presenter `ImagePresenter`. Jak můžeme vidět na sekvenčním diagramu 6.3, instance třídy ImagePresenter dostává skrze metodu `actionUnitPhoto($id,$size)` (1)[13] požadavky na zobrazení obrázků[14]. Tato metoda se dle zadaného ID dotazuje FileRepository na data o obrázku (cestu) (2). Na základě těchto dat dále zjišťuje, zda požadovaný obrázek je již ve správné velikosti v cache (3). Pokud ano, pouze jej načte a zašle do prohlížeče (4). V opačném případě načte originál, upraví jeho velikost (5), uloží do cache (6) a zašle do prohlížeče(7). V obou případech dojde na konci k ukončení skriptu metodou `terminate();`, protože tato *action* nemá poskytovat HTML stránku, ale pouze obrázek.



Obrázek 6.3: Sekvenční diagram získání obrázků

[12] odkládací prostor pro dočasné soubory
[13] metoda zpracovávající GET požadavky tvaru `/image/unit-photo/<ID>&size=<size>`
[14] např. `<img src="/image/unit-photo/2?size=medium">`

6.6 Kontrola plagiátů

Cílovou skupinou této služby jsou studenti, kteří bohužel občas při realizaci svých projektů využívají cizích zdroje, tedy tvoří plagiáty. Atraktivním požadavkem vstupní analýzy byla automatická kontrola plagiátů, která bude obsahem této iterace.

Analýza

Kano model B.5 této iterace vychází spíše z technických požadavků na realizaci. Ve výsledku není uživatel přímo konfrontován s většinou dopadů různého přístupu ke kontrole plagiátů. Cílem je, aby tato prakticky náročná operace nesnižovala přístupové doby pro uživatele a výsledek porovnání byl relevantní. Tedy nemůžeme kontrolovat pouze shody, ale i podobnosti. Použitá technologie by měla odhalovat i zmenšené nebo barevně upravené obrázky. Výstupem porovnání a pozitivního výsledku by měla být notifikace učiteli, který je zodpovědný za dané ročníky.

Návrh

Abychom operací, která může snížit přístupové doby služby, neobtěžovali uživatele, bude kontrola plagiátů řešena dávkově a periodicky v nočních hodinách. Kontrola fotografií, které byly přidány během dne, bude tedy CRON-job. Oddělení již kontrolovaných a nekontrolovaných fotografií řešíme v tabulce **file** sloupcem **checked**. O samotnou kontrolu se stará externí knihovna **libpuzzle**[15], která je k dispozici jako samotný PHP modul, tedy je možno využít jejích funkcí přímo na úrovni PHP. Jelikož je potřeba tento modul explicitně instalovat na straně serveru, což ne každý webhosting dovoluje, bude napsáno API nad touto knihovnou. Toto API bude přijímat požadavek na kontrolu jako POST a bude si udržovat vlastní databázi již porovnaných obrázků (v podobě SQLite). O výsledku porovnání bude informovat v podobě JSON. Toto API zajišťuje škálovatelnost a modulární řešení kontroly. Samotný výpočet nemusí probíhat na stejném serveru, kde běží služba, tedy nebude ohrožen její provoz.
O umístění **externí porovnávací služby** (pracovní název) nese informaci tabulka **setting** v záznamu s klíčem `plagServerPath`. Zde vidíme praktické využití technického nastavení z kapitoly 6.2.

Implementace externí porovnávací služby

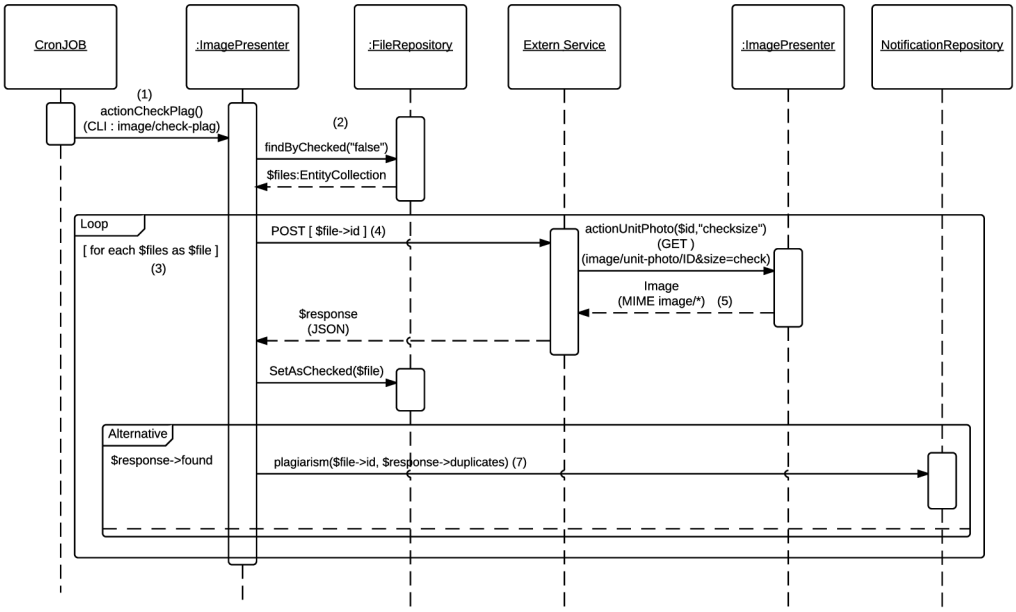
Dle návrhu je potřeba implementovat pro libpuzzle rozhraní, které bude pracovat s databází a komunikovat skrze API. Základ tohoto řešení je již v příkladech knihovny libpuzzle a byla z něj použita práce s databází a volání samotného výpočtu shody. Toto API přijímá POST data s informací o ID souboru, který je porovnáván a URL, na které je tento soubor uložen. Služba si stáhne daný obrázek[16], provede výpočet a porovnání s již uloženými otisky obrázků a uloží si jej do databáze. O výsledku porovnání informuje JSON objektem, který tiskne na výstup. Výpočet otisku zajišťuje interní funkce knihovny libpuzzle `puzzle_fill_cvec_from_file($file)`.

Implementace dávkové kontroly

Implementaci dávkové kontroly vidíme na sekvenčním diagramu 6.4. Dochází k periodickému volání akce presenteru ImagePresenter (1). Tato akce získává od repositáře FileRepository kolekci (kolekci entit, tzv. EntityCollection) doposud nekontrolovaných souborů (2)

[15] http://www.pureftpd.org/project/libpuzzle
[16] funkcí `file_get_contents($url)`

(v tabule **file** mají příznak **checked** nastavený na *false*). Pro každý (3) tento soubor se zasílá POST požadavek s ID a URL (4), na které se nachází porovnávaný obrázek. Tento obrázek poskytuje `ImagePresenter` (5), ale je nutné jej zmenšit na maximální rozměry 3000×3000px. V základním nastavení knihovny libpuzzle je totiž definován maximální rozměr porovnatelného obrázku a pokud si nechceme ručně kompilovat knihovnu, ale chceme ji použít dostupnou z linuxového repositáře, musíme obrázek před odesláním zmenšit. V JSON (6) odpovědi externí porovnávací služby nacházíme v případě shody pole identifikátorů obrázků, se kterými byla nalezena shoda. Pro tyto obrázky generujeme notifikaci (7) pro učitele, který má na starosti třídu studenta, který tyto plagiáty do systému nahrál.



Obrázek 6.4: Sekvenční diagram volání kontroly plagiátů

6.7 Interní komunikace

Z iterace 6.3 vychází požadavek na implementaci interní komunikace mezi uživateli nad položkami. Tento modul má poskytovat párování konverzací k položkám mezi dvěma uživateli.

Analýza

I pro tuto malou iteraci byl vytvořen Kano model B.6. Jak vidíme, nejsou požadavky na modul příliš náročné. Jediný problematický prvek je oddělení zpráv od registrovaných a neregistrovaných uživatelů a udržování návaznosti zpráv. Každé generování zprávy bude doprovázeno emailovou notifikací. Byl definován atraktivní požadavek přímé odpovědi z emailového klienta, ale vzhledem k předpokládanému nulovému využití tohoto požadavku nebude implementován. Tato funkcionalita byla odstraněna z důvodu vysoké náročnosti na implementaci proti nízké přidané hodnotě.

Návrh

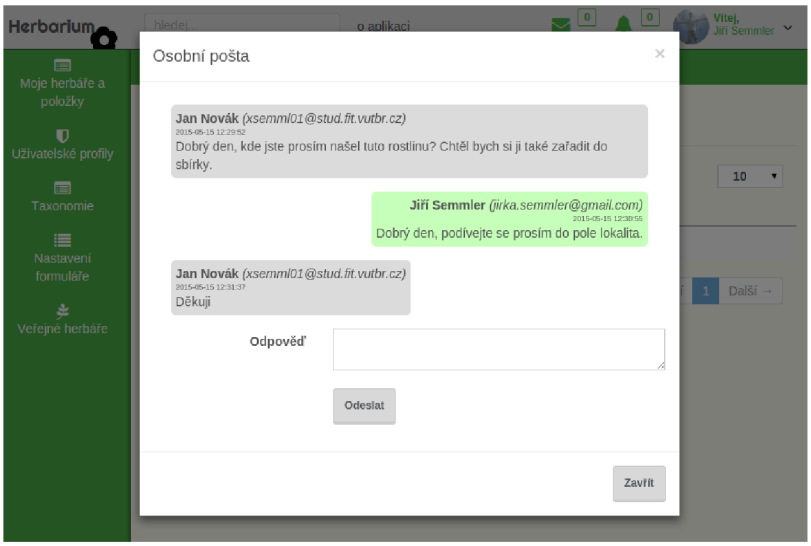
Celá komunikace je řešena v bloku *KOMUNIKACE* tabulkou **thread**, která zapouzdřuje jedno komunikační téma a tabulkou **comment**, která drží samotné zprávy. Návaznost a adresace zpráv je řešena cizími klíči **thread_id** a **parent_id**. Klíč **thread_id** seskupuje zprávy do jedné konverzace a návaznost jednotlivých zpráv na sebe je řešena podobně jako v případě taxonomie skrze klíč **parent_id**, který ukazuje na svoji tabulku. Vazba, kterou má tvořit **parent_id**, je 1:1, tedy ve výsledku se jedná o lineární seznam zpráv.

Modul má podporovat zprávy od neregistrovaných uživatelů, které je ale potřeba identifikovat. Tuto identifikaci udržujeme v tabulce **comment** ve sloupcích **name** a **email**.

Pro modul komunikace slouží na straně modelu repositář `MessageRepository` a entita `Message`. Na straně presenteru se o uživatelské požadavky starají `PublicPresenter` pro zasílání zpráv z položek a `StudentPresenter` pro jejich práci jako registrovaný uživatel.

Implementace

Na straně klienta (z pohledu registrovaného adresáta zprávy) je zobrazen v přehledu pouze seznam konverzací (tabulka **thread**), které poskytuje ze strany modelu metoda `getThreads($user)`. Při otevření jedné z konverzací se zobrazuje modální okno, které zobrazuje jednu konverzaci. Získání konverzace zajišťuje metoda `getTopic($threadId,$me)`, která v cyklu prochází od prvního prvku seznamu (první zpráva konverzace s **parent_id** rovno nule) až po poslední a skrze metodu `appendResult(&$ret,$obj,$me)` připojuje záznamy do pole, které se následně vykresluje ve zmíněném modálním okně. Cílem zobrazení konverzace z hlediska uživatelského rozhraní bylo se co nejvíce blížit navyklému stylu zpráv, který známe například z iMessage od firmy Apple[17].



Obrázek 6.5: Ukázka aplikace - zobrazení komunikace s jiným uživatelem.

[17] https://www.apple.com/ios/messages/

Kapitola 7

Uživatelské testování

7.1 Motivace testování

Během vývoje byl IS testován pouze ze strany autora a zákazníka. Dvoučlenné testování osobami, které jsou autory systému, není dostačující, protože oba trpí autorskou slepotou a nemohou přesně odhadnout uživatelské chování. Proto bylo po implementaci provedeno uživatelské testování, které mělo odhalit chyby a nedostatky systému. Cílem bylo odhalit problémy v následujících rovinách [7]

- **funkční chyby** - Problémy systému, které fakticky nedostačují funkčním požadavkům.
- **nedostatky z hlediska UX** - Elementy, které nebyly schopni studenti použít, protože jim nerozuměli z důvodu jejich špatného návrhu.
- **chybějící prvky** - Funkcionality, které by uživatelé ocenili.

Testovací skupina uživatelů
Testování proběhlo na vzorku devíti studentů 1.-4. ročníku osmiletého gymnázia, které je v této práci definováno jako zákazník. Použitá cílová skupina byla zvolena z následujících důvodů:

- Mají o problematiku zájem, jelikož akce testování proběhla v biologickém kroužek.
- Byli to cíloví uživatelé, kteří fakticky se systémem budou pracovat.
- Byli to technicky nezkušení naivní uživatelé, a proto jsou schopni se chovat zcela bez jiných vlivů.

Učitelská část systému, kde učitel definuje formulářové prvky, taxonomii apod. nebyla podrobena stejnému testování, protože se jedná o administrační úkony, kde je zbytečné investovat do vysoké úrovně uživatelského pohodlí. Naopak je lepší pro složitější operace tuto část osobně proškolit a zapracovat osobní zpětnou vazbu.

Před samotným testováním jsem byl upozorněn na fakt, že přítomnost autority by při testování nepůsobila dobře, tedy bylo nutno odbourat pozici učitele, což vedlo ke zklidnění studentů pracujících se systémem.

7.2 Způsob sběru dat

Sběr dat a zpětné vazby probíhal v následujících krocích:

- Komentář k zadanému úkolu ve scénáři a zápis času, který jim úkol zabral.
- MouseFlow monitoring užívání systému.
- Vlastní poznámky a zvuková nahrávka autora během testování. Důležitými daty byly otázky studentů, nikoliv jejich návrhy řešení. Uživatelé nejsou UX návrháři nebo grafici, ale jejich otázky a nepochopení systému byla klíčová data, která mají vést ke zlepšení služby.
- Technické záznamy v systému.
- GoogleForm dotazník na závěru testování
- Hromadná řízená diskuze.

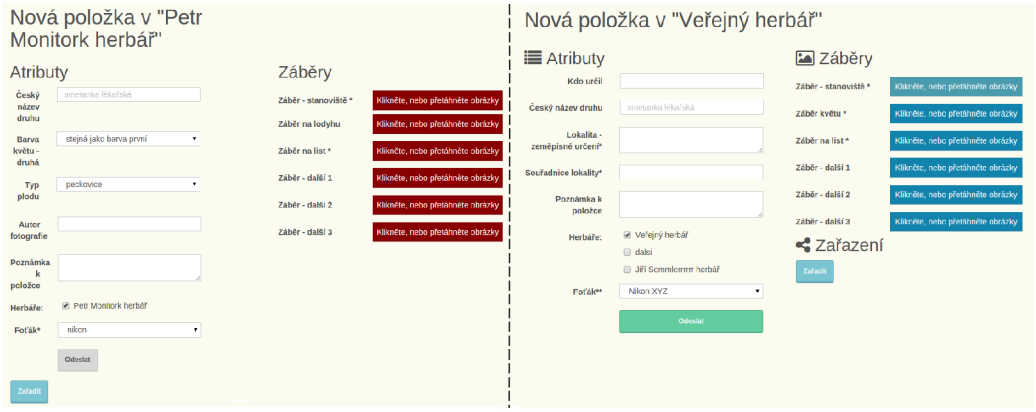
7.3 Průběh testování

Pro studenty byl nachystán scénář - soupis kroků, kterými měli postupně projít. Kroky byly skryté, k dispozici byl vždy pouze jeden řešený krok (řešeno překrytím dalších kroků papírem). Na začátku procesu bylo maximálně vysvětleno ze strany autora, o jaký systém se jedná, co je cílem jejich práce, co se po studentech chce a jak se mají chovat. Byl kladen důraz na maximální upřímnost a otevřenost, kdy jakékoliv pochybení nebo nepochopení úkolu není jejich chybou, ale špatným návrhem autora. Odstranění tohoto psychického bloku se nakonec ukázalo jako velice vhodné, jelikož vedlo k otevřenosti studentů, kteří se neostýchali a ptali se.

V jeden stejný okamžik bylo spuštěno testování, kdy se mohli studenti podívat na první úkol a začít pracovat. Během jejich práce byl autor neustále v kontaktu se studenty a zapisoval si jejich připomínky a otázky, které byly hlavním datovým zdrojem. Pokud někdo narazil na problém ovládání, který neuměl vyřešit, byla snaha o zjištění studentovy představy. Tedy následovaly otázky „Co tě napadlo jako první, když jsi chtěl problém řešit?“, „Kde jsi tuto možnost hledal?“, „Jak bys to chtěl řešit nebo jak jsi zvyklý to řešit?“ Tyto otázky měly ve výsledku velký úspěch, jelikož generovaly množství nápadů na řešení špatného UX.

7.4 Výsledky testování

Výsledná data, která byla sesbírána během celého testování, byla v 70% shodná napříč studenty, neboť většina narazila na stejné problémy. Zbylých 30% byly individuální připomínky. Výsledkem celého testování po analýze byl seznam 14 úprav, které vedly k lepší práci se systémem. Zejména se jednalo o klíčové body, jako je přidávání položek. Na obrázku 7.1 můžeme vidět změnu UX přidávacího formuláře.



Obrázek 7.1: Porovnání rozmístění prvků formuláře. Vlevo před testováním, vpravo po testování.

7.5 Další rozšíření

Služba v první verzi pokrývá požadavky zákazníka, ale až první testovací provoz ukáže nedostatky a chybějící funkcionalitu. Předběžně se uvažuje už o rozšíření, které by komunikovalo s externími službami, jako je FlowerChecker pro detekci rostlin, Moodle pro importování do elearningových kurzů nebo iŠkola pro synchronizaci hodnocení, které studenti obdrží. Jako další krok pro vyšší použitelnost se nabízí implementace mobilní aplikace, která by dovolovala vkládat položky přímo z terénu za předpokladu použití chytrého mobilního telefonu.

Kapitola 8

Závěr

Cílem práce bylo definovat potřeby a navrhnout řešení pro skladování osobních virtuálních herbářů studentů střední školy, které by byly dostupné pro školu, studium a veřejnost s maximálním důrazem na edukační přidanou hodnotu. Po analýze stávajících dostupných řešení jsme došli k závěru, že uspokojivé řešení neexistuje, tedy jsme implementovali vlastní, které bylo uvedeno přímo na míru zadávací škole. Během sběru požadavků jsme došli k závěru, že hlavní přínos pro studium je v dynamickém nastavení a v práci se systémem, nikoliv v informační základně. Kromě uspokojení základních požadavků na systém byly implementovány detekce plagiátů a komunikační rozhraní. Dále byl vývoj po celou dobu řízen agilní metodikou **Lean Software Development**, která zajistila maximální kontakt se zákazníkem, identifikaci požadavků a iterativní vývoj.

Výsledná služba plně poskytuje požadavky zákazníka, které specifikoval na začátku vývoje s mnohými nad rámec. Služba byla důkladně testována v kapitole 7 a ukázala se velká potřeba být v kontaktu s cílovými uživateli během vývoje, jelikož v některých případech ani zákazník neví, jak budou cíloví uživatelé pracovat se službou a co od ní očekávají. Služba byla pojmenována jako Herbarium System a v době dokončení práce byla spuštěna na webové adrese herbarium-system.eu, kde ji aktivně studenti cílové školy začali využívat.

Literatura

[1] Arlow, J.; Neustadt, I.: *UML 2 a unifikovaný proces vývoje aplikací*. Brno: Computer Press, vyd. 1. vydání, 2007, ISBN 978-80-251-1503-9.

[2] Beck, K.: Manifesto for Agile Software Development. online, 2001 [cit. 2015-26-04]. URL http://agilemanifesto.org/iso/cs/

[3] Buriánek, M.: Kano model. online, 2007 [cit. 2015-2-05]. URL http://www.interquality.cz/%C4%8CL%C3%81NKY/tabid/67/ItemId/30/View/Details/AMID/431/Default.aspx

[4] Hefnerová, L.: *Lean Software Development*. bakalářská práce, Vysoká škola ekonomická v Praze, 2012.

[5] Kabir, M.: *Apache server 2*. Brno: Computer Press, vyd. 1. vydání, 2004, ISBN 80-251-0319-6.

[6] Křena, B.; Kočí, R.: Úvod do softwarového inženýrství. online, 2010-12-21 [cit. 2015-26-04]. URL https://wis.fit.vutbr.cz/FIT/st/course-files-st.php/course/IUS-IT/texts/IUS_opora.pdf?cid=8697

[7] Patton, R.: *Testování softwaru*. Praha: Computer Press, vyd. 1. vydání, 2002, ISBN 80-722-6636-5.

[8] Poppendieck, M.; Poppendieck, T.: *Lean software development*. Boston: Addison-Wesley, 10 vydání, 2006, ISBN 03-211-5078-3.

[9] Procházka, F.: MVC aplikace & presentery. online, 2015-04-02 [cit. 2015-08-05]. URL http://doc.nette.org/cs/2.3/presenters

[10] Schwartz, B.; Zaitsev, P.: *MySQL profesionálně*. Brno: Zoner Press, vyd. 1. vydání, 2009, ISBN 978-80-7413-035-9.

[11] Verduyn, D.: Discovering the Kano Model. online, 2013. URL http://www.kanomodel.com/discovering-the-kano-model/

[12] Vrána, J.: *1001 tipů a triků pro PHP*. Brno: Computer Press, vyd. 1. vydání, 2010, ISBN 978-80-251-2940-1.

[13] Špinar, D.: *Tvoříme přístupné webové stránky*. Brno: Zoner Press, vyd. 1. vydání, 2004, ISBN 80-868-1511-0, 301–314 s.

[14] Řezáč, J.: *Web ostrý jako břitva*. Jihlava: BAROQUE PARTNERS, vyd. 1. vydání, 2014, ISBN 978-80-87923-01-6, 14 s.

Příloha A

Specifikace

A.1 Funkční požadavky

Cílem projektu je vytvořit informační systém (IS) pro studenty a profesory střední školy, kteří budou v tomto IS agregovat, hodnotit, diskutovat a studovat digitální botanické herbáře. Funkčnost IS bude členěna do následujících částí

- Správa uživatelů
  - Každý student/profesor musí mít vlastní uživatelský profil, který jej bude identifikovat.
  - Data přístupná veřejnosti (uživatelům bez registrace) musí být explicitně uvedena.
  - Registrace do systému budou vázané na třídy, které definuje učitel v administraci systému. Pro každou třídu budou definovány časové zámky, které povolí registraci do dané třídy.
  - Systém bude obsahovat běžné úkony, jako je emailová notifikace o registraci, možnost resetování hesla nebo editace profilu.
- Správa položek
  - Všichni registrovaní uživatelé budou vkládat své položky a zařazovat je do herbářů, které musí mít nastavitelnou publicitu (veřejný, soukromý, školní).
  - Musí být dynamicky nastavitelné, jaká data budou uživatelé vkládat k položkám.
  - Položky musí být hodnotitelné učitelem.
  - Systém by měl dovolovat interní komunikaci nad položkami.
  - Položky by se měly dát jednoduše zařadit do taxonomického systému.
  - Systém by mohl umět detekovat plagiáty záběrů.
  - V systému by se mělo dát vyhledávat.
- Nastavení systému
  - Musí být nastavitelné, jaká data budou studenti vkládat k položkám.
  - Systém by měl mít nástroj na dodatečnou úpravu taxonomického systému.
  - Učitel by měl mít přehled o profilech v systému.

  - Nastavení taxonomie by mělo být co nejjednodušší.

- Učitelská část
  - Učitel by měl mít možnost hodnotit položky. Toto hodnocení by se mělo skládat z procentuálního hodnocení a komentáře.
  - Každý herbář by měl mít celkové hodnocení počítané jako průměr hodnocení jeho položek.
  - Každé hodnocení by mělo vytvořit notifikaci o hodnocení pro studenta.
  - Učitel by měl mít možnost definovat, kdy se do jaké třídy mohou studenti registrovat.

A.2 Slovník používaných tématických termínů

- **uživatel** - student nebo učitel, který užívá systém
- **třída** - školní ročník
- **položka** - jedna rostlina v herbáři se všemi svými záběry a informacemi
- **herbář** - Položky se seskupují v herbářích. Z technického hlediska se jedná o adresář.
- **formulářový prvek** - jedno formulářové pole, které může nabývat hodnot: textové pole (input), textová oblast(textarea), nabídka (selectbox) nebo soubor (file)
- **záběr** - fotografie rostliny
- **taxonomie** - věda zabývající se klasifikací organismů. V tomto případě se jedná o zařazování položek do biologického stromu.
- **taxon** - například čeleď nebo kmen

Příloha B

Kano modely

Během analýzy každé iterace tvoříme Kano modely dle I pro detailní identifikaci požadavků. Tyto modely jsou součástí každé implementační iterace, ale pro přehlednost je zařazujeme zde.

Sekce : ***Nahrávání souborů***		
Požadavek	**Identifikace**	**K realizaci**
Spolehlivé nahrávání souborů	Základní	ANO
Filtrování cizích souborů dle typu	Vyslovené	ANO
Drag and drop princip vkládání	Základní	ANO
Možnost opakování nahrání souboru	Atraktivní	ANO
Zobrazení průběhu a výsledku nahrání	Atraktivní	ANO

Tabulka B.1: Kano model iterace popsané v sekci 6.4

Sekce : *Správa uživatelů*		
Požadavek	**Identifikace**	**K realizaci**
Registrace uživatelů	Základní	ANO
Bezpečná atentizace a autorizace	Základní	ANO
Obnova hesla	Vyslovené	ANO
Editace profilu	Atraktivní	ANO
Zapínání/vypínání registrace	Vyslovené	ANO
Avatar profilu	Atraktivní	ANO
Autentizace skrze třetí stranu	Atraktivní	NE
Mapování studentů na profil školy	Atraktivní	NE
Zařazení studentů do tříd a práce s nimi	Vyslovené	ANO
Rozlišení registrace pro jednotlivé třídy	Atraktivní	ANO

Tabulka B.2: Kano model iterace popsané v sekci popsané v sekci 6.1

Sekce : *Správa položek*		
Požadavek	**Identifikace**	**K realizaci**
Zařazení položek do složek (herbářů)	Vyslovený	ANO
Možnost sdílení položek mezi herbáři	Atraktivní	ANO
Nastavení práv herbářů	Vyslovený	ANO
Drag and drop nahrávání souborů	Atraktivní	ANO
Jednokroková tvorba položky	Základní	ANO
Splnění všech nastavení formuláře položky	Základní	ANO
Drag and drop přesun položek mezi herbáři	Atraktivní	NE
Možnost diskuze nad položkami	Atraktivní	ANO
Přehrávání galerie záběrů položky	Atraktivní	ANO
Sdílení položek na sociálních sítích	Výslovné	ANO
Editace a mazání položek	Základní	ANO
Jednotné zobrazení položky	Základní	ANO
Hodnocení položek učitelem	Výslovný	ANO

Tabulka B.3: Kano model iterace popsané v sekci 6.3

Sekce : ***Nastavení systému***		
Požadavek	**Identifikace**	**K realizaci**
Dynamické definování taxonomických úrovní	Základní	ANO
Dynamická definice taxonů	Výslovné	ANO
Správa celé taxonomie z jednoho bodu	Atraktivní	NE
Drag and drop[1] přesouvání taxonů ve stromu	Atraktivní	ANO
Definice taxonomie stromovou reprezentací	Základní	ANO
Definice potřebných vlastností atributu formuláře[2]	Základní	ANO
Nastavení atributu jako latinský	Atraktivní	ANO
Nastavení atributu pro vyhledávání	Atraktivní	ANO
Definice pořadí atributů	Základní	ANO
Zobrazování výsledného formuláře v reálném čase	Atraktivní	NE
Nastavení regulárního výrazu pro validaci atributu	Atraktivní	ANO
Možnost selectboxu	Výslovný	ANO
Možnost checkboxů a radiobuttonu	Atraktivní	NE

Tabulka B.4: Kano model iterace popsané v sekci 6.2

Sekce : *Kontrola plagiátů*		
Požadavek	**Identifikace**	**K realizaci**
Porovnání vždy s celou databází	Základní	ANO
Shoda i v upravených obrázcích	Vyslovené	ANO
Modulární řešení - na externí službě	Atraktivní	ANO
Eliminace zatížení serveru	Výslovné	ANO
Automatické spouštění	Atraktivní	ANO
Generování notifikace o pozitivní shodě	Výslovné	ANO

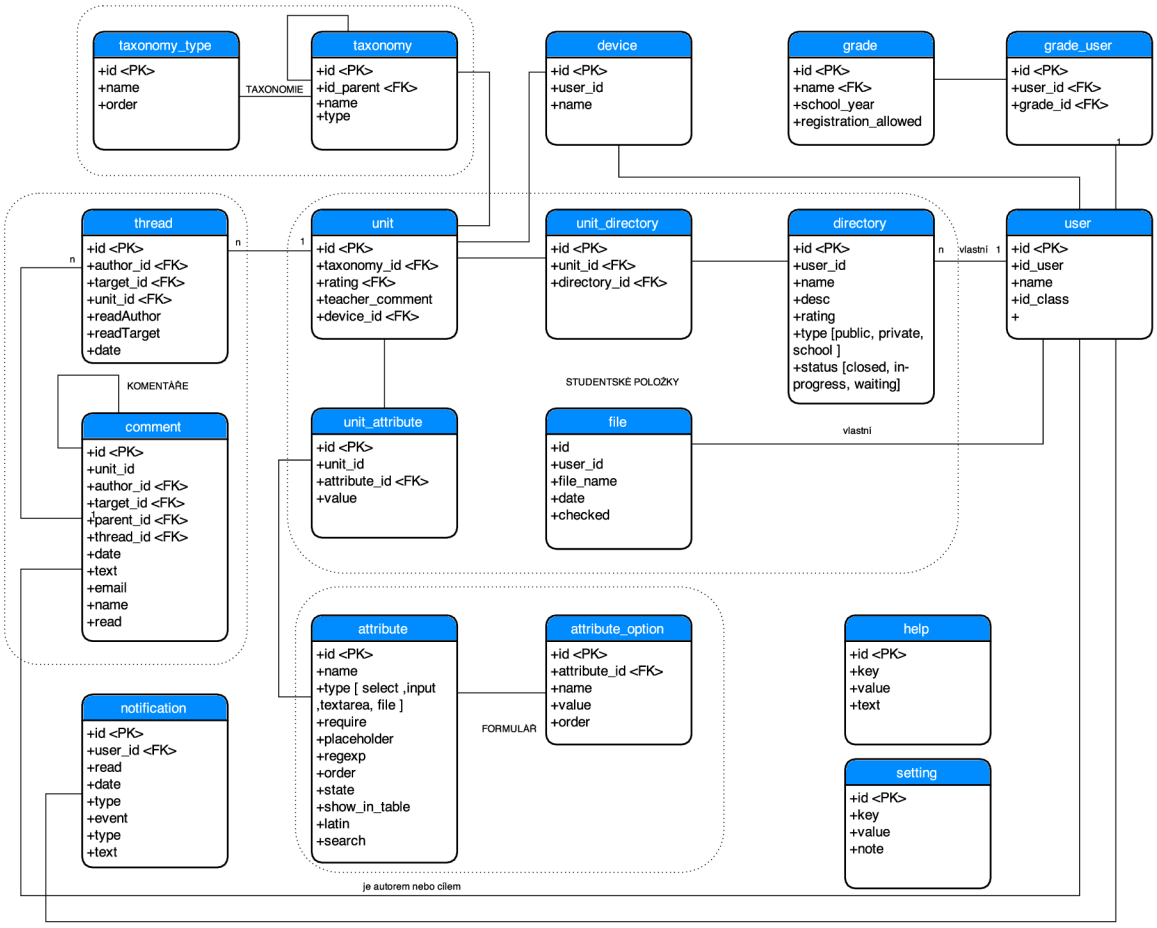
Tabulka B.5: Kano model iterace popsané v sekci 6.6

Sekce : *Interní komunikace*		
Požadavek	**Identifikace**	**K realizaci**
Párování konverzací na položky	Základní	ANO
Emailová notifikace o zprávě	Vyslovené	ANO
Možnost odpovědi na zprávu přímo z emailu	Atraktivní	NE
Vstup do komunikace od neregistrovaných uživatelů	Atraktivní	ANO
Odpovědi v podobě chatu	Výslovné	ANO

Tabulka B.6: Kano model iterace popsané v sekci 6.7

Příloha C

Databázové schéma projektu



Obrázek C.1: Databázové schéma

Příloha D

Obsah CD

Přiložené CD obsahuje následující adresářovou strukturu

- `src/` - zdrojové kódy aplikace
- `sql/` - adresář se schématy databáze
- `doc/` - technická zpráva a její zdrojové kódy
- `data/` - adresář s daty pro instalaci již naplněné databáze
- `readme.pdf` - úvodní dokument popisující strukturu CD a instalaci projektu

Příloha E

Instalace projektu

Nasazení projektu

Tento projekt vyžaduje ke svému běhu tyto zmíněné technologie:

- Apache server 2.4
- PHP 5.6
- MySQL 5.6 (velice důležité pro FULLTEXT INDEX)
- composer

Instalace má následující kroky

1. překopírujte obsah adresáře `/src/herbarium/` do kořenového adresáře webového serveru, který je přístupný přes WWW (typicky na Linuxu `/var/www/projekt`)
2. spusťte příkaz `composer install` pro instalaci závislostí
3. vytvořte databáze ze schématu
   - `/sql/schemaEmpty.sql` - pro instalaci prázdné databáze
   - `/sql/schemaFull.sql` - pro databázi se vzorovými daty
4. v případě volby databáze se vzorovými daty nakopírujte do kořenového adresáře i adresář `/data/`
5. nakonfigurujte databázového připojení v souboru `app/config/config.neon`
6. nastavte hodnoty v tabulce **setting** dle komentáře
7. spusťte CRON úlohu na adresu `/image/check-plag`

Nasazení externí porovnávací služby

Pro spuštění serveru pro kontrolu plagiátů jsou nutné následující technologie:

- Apache server 2.4
- PHP 5.x
- knihovna php5-gd
- knihovna libpuzzle-php

- knihovna php5-sqlite

Instalaci externí porovnávací služby proveďte dle následujících kroků:

1. překopírování obsahu adresáře `/src/plag/` do zvoleného adresáře webového serveru (opět přístupný přes WWW)
2. URL adresu skriptu similar.php, který je obsahem adresáře, překopírujte do nastavení projektu z bodu 5 předchozího postupu