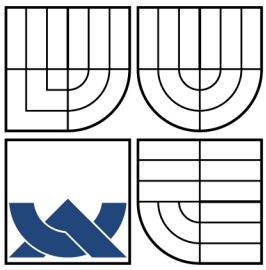
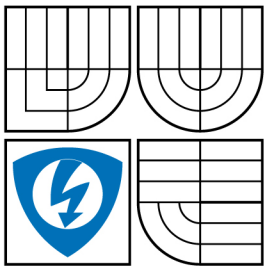


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

SPRÁVA VÝVOJOVÉ DOKUMENTACE PŘES WWW II

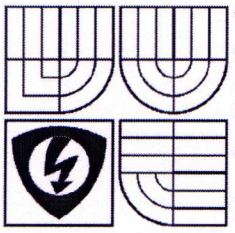
ADMINISTRATION OF DEVELOPMENT DOCUMETATION OVER WWW II

DIPLOMOVÁ PRÁCE
MASTER´S THESIS

AUTOR PRÁCE — Bc. ONDŘEJ GREGÁREK
AUTHOR

VEDOUCÍ PRÁCE — Ing. MIROSLAV BALÍK, Ph.D.
SUPERVISOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Gregárek Ondřej, Bc. ***ID:*** 47879
Ročník: 2 ***Akademický rok:*** 2007/08

NÁZEV TÉMATU:

Správa vývojové dokumentace přes WWW II

POKYNY PRO VYPRACOVÁNÍ:

Práce je součástí rozsáhlejšího projektu Aplikace pro správu vývojové dokumentace. Navrhněte strukturu tabulek pro správu požadavků na vývoj výrobku (Requirements) jako součásti celkové databáze vývojové dokumentace. Za použití HTML, PHP a MySQL (případně také dalších technologií jako JavaScript) navrhněte, vytvořte, odlaďte a otestujte knihovny pro generování HTML kódu pomocí šablon a pro jednotnou navigaci a univerzální rozhraní zobrazování a editaci záznamů v tabulkách. Zaměřte se na návrh tabulek, výběr konceptu a návrh rozhraní jednotlivých knihoven. Práci řešte ve spolupráci s ostatními členy vývojového týmu pod vedením pracovníků firmy Honeywell.

DOPORUČENÁ LITERATURA:

[1] Hlavenka,J.: Vytváříme WWW stránky a spravujeme moderní web site. Computer Press, Brno, 2005, ISBN 80-251-0801-5.
[2] Lacko, Ľ. : PHP a MySQL - hotová řešení, CP Books, 2005.
[2] Huseby,S.H.: Zranitelný kód. Computer Press, Brno, 2006, ISBN 80-251-1180-6.

Termín zadání: 11.2.2008 ***Termín odevzdání:*** 28.5.2008

Vedoucí projektu: Ing. Miroslav Balík, Ph.D.

prof. Ing. Kamil Vrba, CSc.
předseda oborové rady

UPOZORNĚNÍ:

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Bc. Ondřej Gregárek

Bytem: Březinova 98, 586 01, Jihlava

Narozen/a (datum a místo): 8.11.1982, Klatovy

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií

se sídlem Údolní 244/53, 602 00, Brno

jejímž jménem jedná na základě písemného pověření děkanem fakulty:

prof. Ing. Kamil Vrba, CSc.

(dále jen „nabyvatel“)

Čl. 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
   - □ disertační práce
   - ☒ diplomová práce
   - □ bakalářská práce
   - □ jiná práce, jejíž druh je specifikován jako .....

   (dále jen VŠKP nebo dílo)

Název VŠKP: Správa vývojové dokumentace přes WWW II

Vedoucí/ školitel VŠKP: Ing. Miroslav Balík, Ph.D.

Ústav: Ústav telekomunikací

Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v*:

- ☒ tištěné formě – počet exemplářů 2
- ☒ elektronické formě – počet exemplářů 1

* hodící se zaškrtněte

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: …….

…….. ……

Nabyvatel Autor

Abstrakt

Document server je webová aplikace ovladatelná přes internetový prohlížeč, která má sloužit pro správu vývojových dokumentů. Aplikace je rozdělena do čtyř základních částí: Requirements, Products, Tests a Test Run. Část Requirements slouží pro vkládání požadavků na výrobky. Na základě těchto požadavků se vyrobí produkt, který je evidován v části Products. V části Tests jsou vytvářena zadání testů dle požadavků z části Requirements. Testy jsou pak realizovány s jednotlivými produkty. Výsledky těchto testů eviduje část Test Run. Součástí aplikace je správa uživatelů, připojování příloh k záznamům, tisk a exportování dat do různých formátů, ukládání historie záznamů, filtrování a řazení záznamů, atd.

Veškerá data jsou uložena v databázi MySQL, aplikace je napsána ve skriptovacím programovacím jazyce PHP. Data jsou prezentována pomocí šablonovacího systému Smarty, výstup je v jazyce XHTML a k formátování je využito kaskádových stylů CSS.

Tato práce popisuje vývoj aplikace. Začátek je věnován návrhu databáze, propojení a vazby jednotlivých tabulek. Přitom je podrobně vysvětlena funkce programu, což je pro správný návrh databáze podstatné. Na databázi je postavena aplikace. Ukázána je zvolená struktura souborů, vztahy skriptů a knihoven funkcí. Vysvětlen je šablonovací systém a rozhraní pro přístup programu do databáze. Nejvíce je v práci věnováno popisu řešení důležitých funkcí aplikace, tj. výpisu záznamů, jejich stránkování, filtrování a řazení, operace se záznamy: ukládání, prohlížení, kopírování, schvalování a práce s historií, kategorie a problémy při udržování konzistentního stromu, export dat do různých formátů. Vždy je nastíněn problém, myšlenka řešení a popis příslušných skriptů. Pro lepší pochopení komplikovaných algoritmů jsou také uváděny ukázky zdrojového kódu.

Klíčová slova

Document server, vývojová dokumentace, požadavek, produkt, test, historie, výpis, filtrování, stránkování, řazení, tisk, export, www, databáze MySQL, skriptovací jazyk PHP, šablonovací systém Smarty, jazyk XHTML, kaskádový styl CSS, tabulka, vazba, program, kód, aplikace, knihovna, skript, internetový prohlížeč.

Abstract

Document server is a web application controllable by way of web browser. It is meant to serve for the management of development documentation. The application is divided to the four basic parts: Requirements, Products, Tests and Test Run. The section Requirements serves for inserting requirements for products. Product is produced on the basis of these needs and registered in part Products. Test setting is created in the part Tests according to requirements from the part Requirements. Particular products are then tested. The part Test Run registers records of these tests. These are parts of the application: management of users, connecting supplements to records, printing and exportation of data to different formats, saving history of records, filtration and sorting of entries, etc.

All the data is saved in the database MySQL. The application is written in scripting language PHP. Data is presented by template system Smarty. The output is in language XHTML. Cascading style CSS is used to formatting.

This work describes development of the application. First it is dealing with the proposal of the database, connection and structure of particular tables. The function of the programme is explained in detail at the same time, which is essential for the correct proposal of the database. The application is based on the database. The selected structure of files and relations of scripts to library functions are shown. The template system and the interface for access of the programme to the database are explained. The most attention is paid to the description of solving important functions of the application, i.e. listing of records, their pagination, filtration, sorting and operation with them: saving, browsing, copying, confirmation and work with history, category and problems by upkeep of consistent tree and export of data to various formats. It is always outlined the problem, the idea of solving and the description of appropriate scripts. Samples of source code are also included for better understanding of complicated algorithms.

Keywords

Document server, development documentation, requirement, product, test, history, list, filtration, pagination, sorting, print, export, www, database MySQL, scripting language PHP, template system Smarty, language XHTML, cascading style CSS, table, relation, program, code, application, library, script, web browser.

Obsah

Seznam obrázků

Úvod

Každá větší organizace, která vyrábí velké množství produktů, se zřejmě setkala s problémem organizace a uspořádání těchto produktů, požadavků a testů na ně, jejich vývojové dokumentace. Vzniká tak potřeba zavést systém, který by tento problém řešil, tj. přehledně uchovával veškeré produkty, požadavky, testy a vývojovou dokumentaci. Nabízí se řešení v podobě aplikace ovladatelné pomocí internetového prohlížeče, což je také téma této práce.

Aplikace byla vyvinuta ve spolupráci s Bc. Miroslavem Kolářem. Celý projekt byl rozdělen do několika částí, které byly přibližně rovnoměrně přiděleny oběma autorům. Jednotlivé části však nebylo možné rozdělit zcela jednoznačně, takže se mnohdy překrývaly, což vyžadovalo těsnou spolupráci. Oba autoři se ve svých diplomových pracích věnovali pouze svým částem aplikace, proto v této práci budou popsány pouze ty části aplikace, které zpracoval autor této diplomové práce. Výjimkou je pouze popis některých tabulek databáze (v kapitolách 2.1 a 2.3), které sice vytvořil Bc. Miroslav Kolář, avšak následně k nim autor této práce přistupoval prostřednictvím svých skriptů. Pro pochopení těchto skriptů i principu celé aplikace byl popis těchto tabulek nezbytný. Oba autoři také navzájem využívali funkce toho druhého. V takovém případě bude pouze odkázáno na diplomovou práci druhého autora [1], kde daná funkce bude popsána.

Text je napsán tak, jakoby čtenář aplikaci vytvářel také. To umožní snadnější pochopení popisovaných jevů. Postupný vývoj aplikace určil sled kapitol. Nejprve stanovíme, co má aplikace umět. Podle toho pak navrhneme databázi. Na hotové databázi začneme stavět aplikaci, jejímuž popisu bude věnováno nejvíce prostoru. Definujeme si strukturu skriptů, zavedeme šablonovací systém a zavedeme jednotný způsob přístupu do databáze. Pak již budeme moci programovat jednotlivé stránky pro operace se záznamem a pro výpis záznamů.

I podkapitoly jsou psány v podobném duchu. Nejprve je nastíněn řešený problém. V prvních podkapitolách se pustíme do nejdůležitějších skriptů a funkcí. S jejich využitím dokončíme zbylé skripty a funkce. Jednotlivé kapitoly na sebe navazují, proto je vhodné číst práci postupně. Kapitola vytržená z kontextu nebude sama o sobě dávat smysl. Její obsah vždy navazuje na údaje řečené v předchozím textu. Ne vždy však bylo možné postupovat takto strukturovaně, protože jednotlivé skripty jsou vzájemně propojené a funkce často využívané na více místech. Jejich význam přitom bývá stejně velký a v reálné situaci je programu-

jeme často najednou, proto je obtížné začít s popisem jedné a další popisovat až po dokončení popisu předchozí. Přesto však byla snaha práci tímto způsobem psát.

Tam, kde se toto nepodařilo dodržet, jsou zmíněny funkce nebo skripty, které budou popsány v pozdějších kapitolách. V takovém případě bude vždy uvedeno číslo dané kapitoly. Není však nutné na tyto kapitoly hned přecházet. Většinou se jedná o popis funkcí, které nemají zásadní význam pro popisované skripty v dané kapitole. Proto se pro danou chvíli spokojíme s tím, že budeme vědět, že funkce existuje, jaké parametry vyžaduje, co vrací a že bude popsána později společně s jinými skripty, pro které má větší význam.

Doporučeno je také číst práci společně se zdrojovými kódy aplikace, které jsou mnohdy rozsáhlé a složité. Jedině tak bude možné pochopit zdrojové kódy popisované v tomto textu. Tam, kde to bude vhodné, bude uvedena podrobně popsaná ukázka části zdrojového kódu. Většinou se bude jednat o složitě sestavované dotazy do databáze, které jsou ve zdrojovém kódu zapsány na jediném, ale velmi dlouhém řádku.

1. Požadavky na aplikaci

Nejprve je nudné definovat vztahy mezi jednotlivými částmi aplikace. Části Requirements a Products musí být navrženy tak, aby bylo možné zadat více požadavků na jeden produkt a naopak – jeden požadavek se může vztahovat k více produktům. Stejná vazba bude i mezi částmi Requirements a Tests. Jeden test může být napsán pro více požadavků a zároveň může být více testů na jeden požadavek. Část Test run musí být propojitelná s částmi Test a Products, aby bylo zjistitelné, jaký test se prováděl a s kterým produktem. Požadavky by měly být pro přehlednost řazeny do kategorií, produkty do rodin produktů. Souvisejícím požadavkům a produktům by mělo jít přiřadit prioritu a stav zpracování.

Zásadním požadavkem je uchování historie změn pro jejich zpětné dohledání, takže u všech požadavků, produktů a testů musí být zachované všechny jejich modifikace a také změny vazeb mezi nimi. V části Test run se historie ukládat nebude, neboť výsledek testu se jednou uloží a nebude potřeba jej dále modifikovat. Aplikace bude provádět určité operace s daty, jakými budou např. ukládání, schvalování, zrušení (v části Test run) a testování (v části Tests). U každé operace musí být zaznamenán čas a osoba, které operaci provedla, což vede k nutnosti vedení správy uživatelů. Uživatelé budou rozděleni do několika skupin podle toho, čemu se fakticky věnují a ke které části aplikace mají mít přístup. Vzniká tak nárok na vhodné zabezpečení aplikace, aby neoprávněná osoba neměla přístup mimo své vymezené pole působnosti.

Potřebné bude ukládat informaci i o tom, že nový požadavek vznikl z jiného zkopírováním. Jinak řečeno, je potřeba zpětně dohledávat jednotlivé větve historie úprav. Aplikace by také měla umožňovat nastavovat závislost požadavků na jiných požadavcích, aby je bylo možné použít pouze s jinými požadavky tak, aby požadavek dával smysl. Možné bude vytvářet požadavky šité na míru určitým produktům.

Zadání a popisy jednotlivých požadavků, produktů a testů se neobejdou bez přiložených souborů. Proto bude možné přikládat různé typy souborů. Důležité je umožnit k jednomu požadavku (resp. produktu, testu) přiřadit libovolný počet souborů a zaznamenávat údaje o času vložení a osobě.

S daty z databáze pak bude pracovat aplikace. Důraz je kladen na to, aby veškerá data byla oddělena od programu. Několik požadavků se vztahuje i ke vnitřní podobě programu,

k jeho skriptům a struktuře. Aplikace má být rozdělena do knihoven funkcí podle několika oblastí, které je potřeba do aplikace zahrnout:

- knihovna pro přístup do databáze,
- knihovna pro dynamické generování obsahu stránek pomocí šablon,
- knihovna pro generování HTML kódu pomocí šablon a jednotnou navigaci,
- univerzální rozhraní zobrazování a editaci záznamů v tabulkách, včetně filtrování, stránkování a výběru dle kritérií,
- knihovna pro tiskový výstup a pro export do formátu XML, PDF, MS Excel a MS Word,
- knihovna pro autorizovaný přístup k aplikaci,
- jednotné univerzální formulářové rozhraní k databázi, včetně pokročilých polí, jako např. WYSIWYG textová pole a pole pro nahrávání souborů,
- univerzální rozhraní pro pokročilé prohledávání záznamů v tabulkách a předání výsledků k zobrazení.

Technologie, které mají být pro aplikace použity, jsou následující: databáze bude pracovat na MySQL, aplikace pak na skriptovacím jazyce PHP. Data budou prezentována ve formě HTML. Pro usnadnění práce s aplikací může být využito také JavaScriptu.

2. Návrh databáze

Prvním krokem při tvorbě webové aplikace je obvykle návrh databáze. Jedná se o návrh struktury tabulek a vazeb mezi nimi tak, aby nadřazená aplikace mohla snadno a rychle s touto strukturou a daty pracovat.

Základem dobrého návrhu databáze je podrobné prodiskutování požadavků se zadavatelem projektu. Musíme vědět, jaká data chceme ukládat a co s nimi chceme dělat a až poté je rozdělíme do tabulek a sloupců. K tabulkám přiřadíme primární klíče, které slouží pro jednoznačnou identifikaci jednotlivých záznamů a definujeme vazby mezi tabulkami podle toho, jak spolu data z různých tabulek souvisí. Zároveň dbáme na normalizační pravidla, která nám pomáhají se správným strukturováním databáze. Více se na téma návrhu databáze dočteme v [3] a [4].

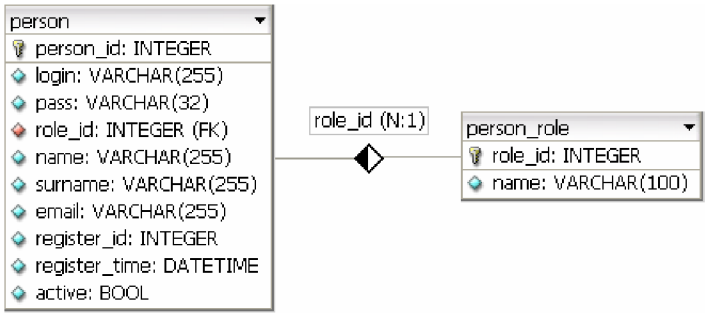
Při návrhu databáze si také můžeme definovat sql dotazy, které by měly s databází pracovat. Většinou to jsou dotazy na výpis řady záznamů, nebo výpis jednoho záznamu včetně vazeb na záznamy v jiných tabulkách. Můžeme tím odstranit řadu problémů, kdy např. zjistíme, že dotaz na námi vytvořenou strukturu tabulek by byl příliš složitý. Strukturu proto upravíme tak, aby výsledný dotaz byl jednoduchý. Při tvorbě dotazů nám výrazně pomůže [2], dobrou pomůckou je i [7]. O elegantní řešení různých specifických problémů se dočteme v [6]. Podrobný popis celé MySQL včetně řešených příkladů nalezneme v [5].

Způsob popisu databáze byl zvolen tak, jak skutečně databáze vznikala, tedy od tabulky uživatelů, přes návrh základních tabulek jednotlivých částí, až po implementaci historie. Konečnou podobu celé databáze si můžeme prohlédnout v příloze. V tomto textu nebudou blíže popsány tabulky pro připojování souborů k záznamům (file_list) a pro ukládání textů (text), protože nebyly součástí úkolů autora tohoto textu. Proto pouze odkážeme na [1], kde jsou popsány více.

2.1 Uživatelé

Téměř při každé operaci bude potřeba zaznamenávat uživatele, který operaci provedl. Definujme si nejprve tabulku person, která bude sloužit jako evidence uživatelů. Každý uživatel bude jednoznačně identifikován primárním klíčem person_id. Tato hodnota se pak bude ukládat k záznamům v ostatních tabulkách a bude tak možné identifikovat uživatele, který záznam vytvořil nebo upravil.

Uživatelé budou mít své jedinečné uživatelské jméno, které bude uložené v poli login a heslo uložené v poli pass ve formě haše. Jméno a příjmení bude uloženo v polích name a surname. V poli email bude uložena emailová adresa uživatele. Dále zavedeme pole register_time a register_id, jež budou sloužit pro zaznamenání času vytvoření uživatelského účtu a uživatele, který nového uživatele vytvořil. Nakonec zavedeme pole active, které bude určovat, zda je uživatelský účet aktivní či nikoliv.



Obr. 2.1: Tabulky person a person_role

Abychom mohli uživatele rozdělovat do skupin podle částí aplikace, ve které mají pracovat a do níž mají mít přístup, zavedeme tabulku person_role, která bude udržovat seznam všech skupin (rolí). Mezi oběma tabulkami vytvoříme vazbu pomocí pole role_id, které bude definovat skupinu, do niž je uživatel zařazen.

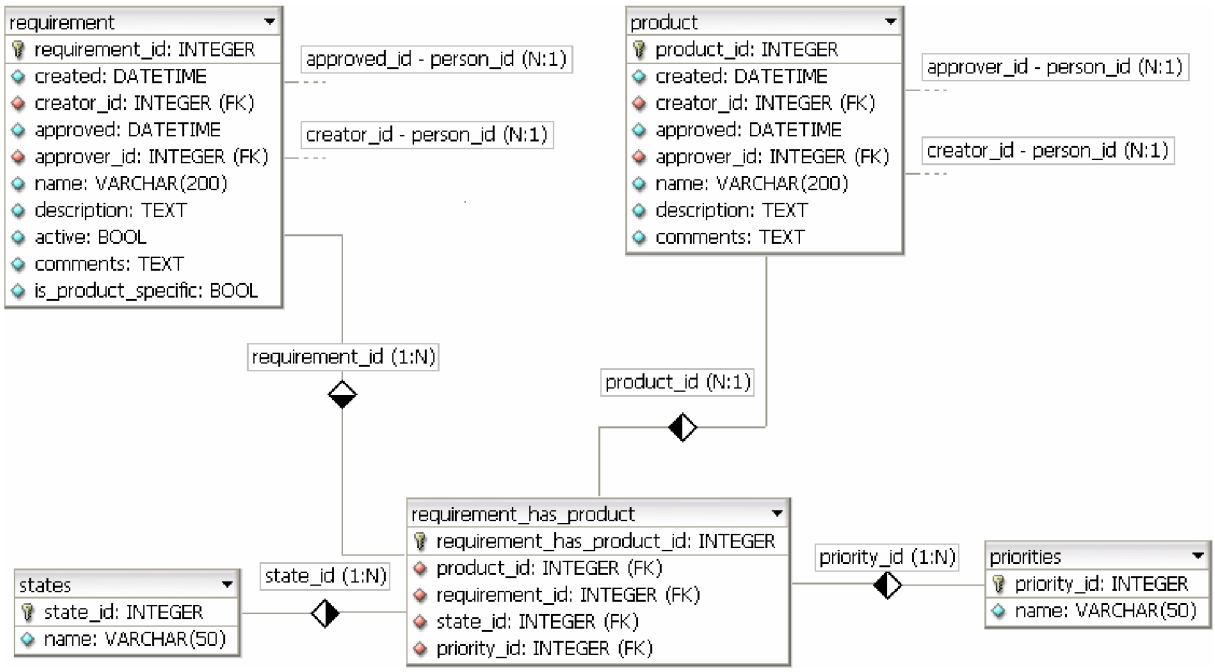
2.2 Požadavky a produkty

Základem částí Requirements a Products bude tabulka requirement a její vazba na tabulku product. Tabulka requirement bude sloužit pro ukládání požadavků. Každý požadavek bude mít svůj vlastní záznam identifikovaný primárním klíčem requirement_id. Bude mít své jméno a popis (pole name a description) a čas vytvoření a schválení (pole created a approved). Uživatel, který záznam vytvořil nebo schválil bude zaznamenán v polích creator_id a approver_id. Vazbou s tabulkou person umožníme jeho identifikaci. Pole comment poslouží pro uložení interních poznámek souvisejících s požadavkem. Požadavky, které budou ušity na míru produktům, bude určovat pole is_product_specific. Pole active určí, zda bude požadavek aktivní či nikoliv. Tabulka product, sloužící pro ukládání produktů, bude mít strukturu podobnou tabulce requirement s tím rozdílem, že zde nebudou pole active a is_product_specific.

Máme-li obě tabulky vytvořené, můžeme přistoupit k definici vazeb mezi nimi. Jedním z požadavků bylo, aby k jednomu produktu bylo možné definovat více požadavků a aby se jeden požadavek mohl vztahovat k více produktům. To nám dává relaci typu N:N, kterou realizujeme pomocí další propojovací tabulky requirement_has_product. Ta bude mít dva cizí

klíče requirement_id a product_id, sloužící pro vytvoření dvou relací typu 1:N s tabulkami requirement a product. Primárním klíčem bude pole requirement_has_product_id.

Každý záznam v tabulce requirement_has_product může nabývat jeden z několika stavů a priorit. Pro jednotlivé hodnoty stavů a priorit zavedeme tabulky states a priorities. Pro uložení hodnoty použijeme v obou tabulkách pole name, pole state_id a priority_id budou primární klíče, které budou zároveň cizími klíči v tabulce requirement_has_product a vytváří tak relaci typu 1:N. Hodnoty stavů v tabulce states budou: uloženo, schváleno, implementováno, testováno a smazáno. Hodnoty priorit v tabulce priorities budou: nízká, střední a vysoká. Celá struktura částí Requirements a Products je naznačena na obr. 2.2.



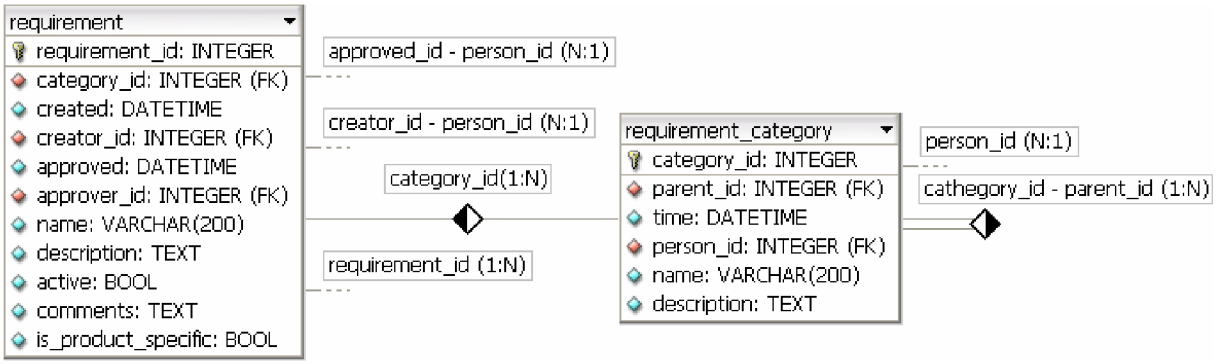
Obr. 2.2: Struktura tabulek pro požadavky a produkty

Abychom neztráceli ve velkém množství požadavků a produktů přehled, rozdělíme je do kategorií. Budou k tomu sloužit dvě nové tabulky, requirement_category pro kategorie v části Requirements a product_category pro rodiny produktů.

Vytvoříme tedy tabulku requirement_category, jejíž primární klíč category_id bude udávat danou kategorii. Každý záznam se bude vyznačovat časem vytvoření (time) a uživatelem, který záznam vytvořil (person_id), jménem (name) a popisem (description). Členění podkategorií do stromové struktury umožníme polem parent_id. Pokud bude v záznamu tato hodnota vyplněna, znamená to, že se jedná o podkategorii, jejíž nadřazená kategorie bude mít category_id rovno parent_id tohoto záznamu. V opačném případě se jedná o kategorii na nejvyšší úrovni. Do tabulky requirement přidáme pole category_id a vytvoříme tak vazbu mezi

oběma tabulkami. Category_id bude udávat příslušnost záznamu k dané kategorii. Pokud bude nulové, požadavek není do žádné kategorie zařazen.

Budeme-li chtít vypisovat kategorie seřazené do stromové struktury, dostáváme se do problému, neboť na úrovni databázového sql dotazu nejsme schopni data takto získat. Řešení spočívá v naprogramování rekurzivní funkce nebo cyklu v nadřazené aplikaci, která bude postupně opakovanými dotazy data získávat ze všech podkategorií. Více v [8].



Obr. 2.3: Tabulky requirement a requirement_category

Stejným způsobem, jako tabulku requirement_category, vytvoříme tabulku product_category a její vazbu na tabulku product.

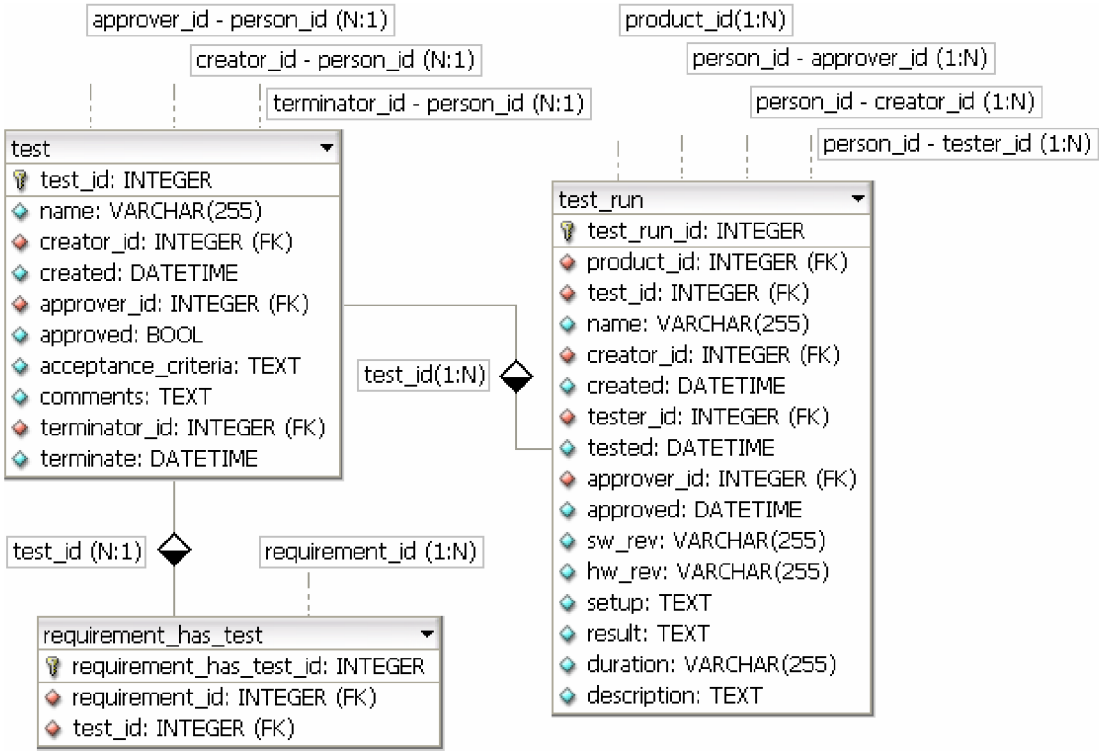
2.3 Testy a výsledky testů

Druhou částí celé databáze budou sestavy testů na základě požadavků a výsledky testů produktů. Testy budeme ukládat do tabulky test, jejíž struktura se bude podobat tabulkám requirement a product, takže zde opět použijeme pole name, created, creator_id, approved, approver_id a comments. Primárním klíčem bude pole test_id. Dále zavedeme pole acceptance_criteria, kde budou definována kritéria a popis testů. Přidáme také sloupce terminate a terminator_id pro ukládání času ukončení a uživatele, který test ukončil. Zajímavostí je název pole terminate: původně měl být název v souladu s ostatními sloupci tohoto typu, tj. terminated. V MySQL je však tento název vyhrazen pro jiné účely, takže nemohl být použit.

Uživatelé budou testy vytvářet podle požadavků na produkty. Proto vytvoříme vazbu tabulky test na tabulku requirement. Tato vazba bude typu N:N, protože jeden test může být vázán na více požadavků a na jeden požadavek může mít více testů. Vytvoříme vazební tabulku requirement_has_test, která bude mít cizí klíče requirement_id a test_id ukazující na tabulky requirement a test. Primárním klíčem bude pole requirement_has_test_id.

Výsledky provedených testů budou uloženy v tabulce test_run. Jeden konkrétní test můžeme provést vícekrát pro různé produkty, stejně tak pro jeden produkt můžeme provést

více rozdílných testů. Z toho plyne, že tato tabulka bude vazební mezi tabulkami test a product s relací typu N:N. Tabulka test_run bude obsahovat cizí klíče test_id a product_id. Primárním klíčem bude pole test_run_id. Opět zde budou pole created, creator_id, approved a approver_id.



Obr. 2.4: Tabulky test, test_run a requirement_has_test

Pole tested a tester_id budou udávat čas provedení testu a uživatele, který test provedl. Dále přidáme pole hw_rev pro hardwarové revize desky, sw_rev pro revizi softwaru, pole setup pro krátký popis uspořádání testu a pole duration pro udání délky trvání testu. Výsledek testu bude uveden v poli result.

Struktura částí Tests a Test Run je zobrazena na obr. 2.4. Tímto máme hotovu základní strukturu databáze, kterou tvoří tabulky requirement, product, test a test_run a vazby mezi nimi.

2.4 Historie změn

Zásadním požadavkem, který významně ovlivní celou databázi, je ukládání historie všech změn a jejich verzí. Máme-li již provázaný požadavek s produkty nebo testy a chceme-li provést jeho změnu, musíme zachovat původní verzi produktu pro požadavky a testy, se kterými byl již provázán a vytvořit novou kopii záznamu. Pro lepší orientaci budeme jednotlivé verze číslovat a také budeme ukládat návaznost záznamů a jejich modifikací na jiný zá-

znam v případě, že jsme vytvořili již trochu odlišný požadavek z jiného. Vznikne nám tím stromová struktura – větve této struktury tvoří záznamy a jejich modifikace. Každá taková větev bude číslována – opět pro snadnější orientaci. Taktéž budeme ukládat závislost záznamů na jiných záznamech v případech, že požadavek bez jiného požadavku nemá opodstatnění nebo nelze použít.

Způsobů, jak tento požadavek vyřešit, je několik. Popíšeme krátce dva základní zvažované způsoby, zhodnotíme jejich klady a zápory a poté jeden z nich vybereme. Obě řešení popíšeme na tabulce requirement.

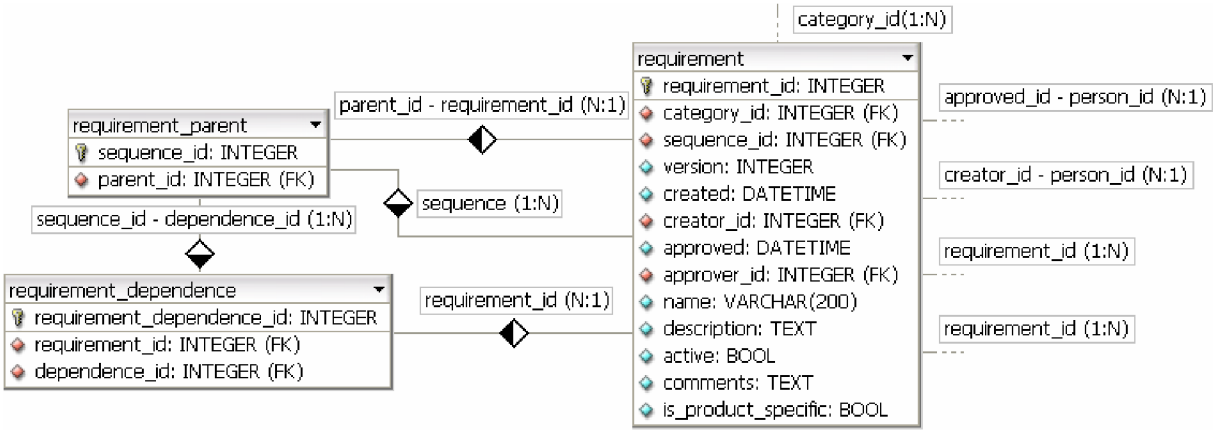
První možností je rozdělení tabulky requirement na dvě tabulky. V první tabulce by byly uloženy pouze aktuální záznamy, druhá tabulka by sloužila pro neaktuální záznamy. Výhodou je to, že v první tabulce by bylo pořadí (pole sequence_id) přímo primárním klíčem. Ve druhé tabulce by sequence_id bylo cizím klíčem pro vazbu mezi tabulkami. Jednotlivé verze záznamů v rámci jedné historie by byly rozlišovány polem version. Pro výpis seznamu aktuálních požadavků i jedné větve historie by nám stačil jednoduchý sql dotaz. Nevýhodou je však nutnost na úrovni aplikace rozlišovat, zda je záznam aktuální či nikoliv, podle toho použít vhodný dotaz a hledat ve správné tabulce. Dalším problémem je nutnost záznamy při aktualizaci přesouvat z jedné tabulky do druhé. Máme-li již vazby mezi záznamy v různých tabulkách, musíme i po aktualizaci záznamu zachovat vazby na původní záznam. Tím, že záznam přesuneme z jedné tabulky do druhé, musíme nějakým způsobem upravit i vazby, což je značně pracné.

Z předchozího tedy vyplývá, že je vhodné ukládat aktuální i neaktuální záznamy do jedné tabulky. Význam polí sequence_id a version by byl stejný, museli bychom však zajistit získávání hodnot sequence_id pomocí nadřazené aplikace, nebo ze speciální tabulky, ve které by sequence_id bylo primárním klíčem s automatickou inkrementací. Výhodou zde je, že většina dotazů je jednoduchá, pouze pro vypisování seznamu aktuálních záznamů bychom potřebovali poddotaz, ovšem nikterak složitý. Taktéž vytváření a aktualizace záznamů nepředstavuje žádný problém. Protože toto řešení má více kladů než první řešení, budeme se dále zabývat jen tímto řešením.

Zavedeme tedy pole sequence_id a version. Pole sequence_id bude určovat jednu větev modifikací požadavků, pro kterou bude mít hodnotu vždy stejnou. Nově také převezme funkci číslování požadavků, kde číslo požadavku je stejné i pro jeho modifikace. Pro tuto funkci již nemůžeme použít pole requirement_id, protože toto pole má pro všechny záznamy a modifikace jedinečné číslo. Nyní můžeme k požadavkům přistupovat díky znalosti jejich pořadí (sequence_id), případně verzí (version), která jsou číslována po sobě bez mezer, což již

requirement_id nesplňuje. Zde je také důležité zmínit, že pod pojmem historie záznamu budou myšleny starší varianty aktuálního záznamu (tj. se stejným sequence_id).

Návaznost větví požadavků na jiné požadavky bude řešit tabulka requirement_parent, ze které též budeme brát hodnoty sequence_id pro nové požadavky v tabulce requirement. Tím, že sequence_id zde bude primárním klíčem s automatickou inkrementací, dosáhneme toho, že nebudeme muset před vložením nového požadavku zjišťovat maximální hodnotu sequence_id. Pole parent_id bude ukazovat na requirement_id v tabulce requirement. Tato vazba tedy ukazuje na rodičovský požadavek. Pokud vytváříme zcela nový požadavek, který rodičovský požadavek nemá, bude parent_id nabývat hodnoty NULL.



Obr. 2.5: Řešení historie, návazností a závislostí záznamů v části Requirements

Pro nastavení závislosti požadavků na jiných požadavcích použijeme tabulku requirement_dependence. Zde bude hodnota requirement_id cizím klíčem z tabulky requirement. Bude označovat závislý záznam. Díky vazbě na requirement_id můžeme i tyto závislosti měnit a uchovávat v historii. Dependence_id je pak odkaz na požadavek (sequence_id - bráno jako celá větev), na kterém je požadavek závislý. Pro vazbu dependence_id – sequence_id s výhodou použijeme tabulku requirement_parent, ve které je pole sequence_id jedinečné. Tabulka bude realizovat vazbu N:N, bude tak možné mít požadavek závislý na více požadavcích a více požadavků může být závislých na jednom požadavku.

Tímto máme hotové ukládání historie, pořadí a verzí, rodičovských požadavků a závislostí pro část Requirements. Celá struktura je naznačena na obr. 2.5. Stejný princip uplatníme i pro tabulku části Products a Tests s tím rozdílem, že se zde nebudeme ukládat a závislosti. U části Tests nebudeme ukládat ani rodičovské záznamy. Tabulky product_dependence, test_parent a test_dependence zde tedy nebudou. V části Test Run nebude nutné historii zavádět, protože se zde budou ukládat výsledky testů bez nutnosti jejich další modifikace.

Pro práci nad strukturou tabulek v části Requirements si definujeme několik dotazů:

- Výpis seznamu aktuálních požadavků:

```
SELECT * FROM requirement
WHERE requirement_id
IN (
  SELECT MAX (requirement_id)
  FROM requirement
  GROUP BY sequence_id
)
```

Poddotazem si zjistíme maximální hodnoty requirement_id záznamů se stejnou hodnotou sequence_id. V hlavním dotazu pak omezíme výpis sloupce requirement_id na tyto hodnoty a získáme tím výpis pouze aktuální záznamů.

- Výpis aktuálního požadavku ze znalosti jeho hodnoty sequence_id:

```
SELECT * FROM requirement
WHERE sequence_id = 2
ORDER BY version DESC
LIMIT 1
```

Záznamy sestupně seřadíme podle pole version (řadit lze též podle pole requirement_id) a omezíme podle požadované hodnoty sequence_id, takže na prvním místě bude aktuální požadavek. Parametrem limit omezíme výpis na tento jeden aktuální záznam. Budeme-li chtít vypsat i rodičovský požadavek, dotaz rozšíříme takto:

```
SELECT r.*, r2.requirement_id AS parent_id, r2.name AS parent_name
FROM requirement r
LEFT JOIN requirement_parent rp USING (sequence_id)
LEFT JOIN requirement r2 ON rp.parent_id = r2.requirement_id
WHERE r.sequence_id = 4
ORDER BY r.version DESC
LIMIT 1
```

- Výpis historie jednoho požadavku:

```
SELECT * FROM requirement
WHERE sequence_id = 2
ORDER BY version DESC
```

- Výpis požadavku z historie při znalosti polí sequence_id a version:

```
SELECT * FROM requirement WHERE sequence_id = 1 AND version = 3
```

3. Struktura aplikace

Máme-li dokončenou a odladěnou databázi, můžeme přistoupit k návrhu samotné aplikace, která má být naprogramována ve skriptovacím programovacím jazyce PHP. Pro nauku programování v PHP nám dobře poslouží [11], určitě se neobejdeme bez [10], kde zjistíme podrobné popisy všech funkcí a příkazů. Základy o správném formátování HTML výstupu se dočteme v [12]. Stejně jako u MySQL se v [6] dozvíme praktické rady při programování v PHP.

Protože na aplikaci spolupracuje více lidí současně, je potřeba si zvolit vhodnou souborovou strukturu tak, aby každá podstránka nebo knihovna funkcí měla svůj vlastní soubor. V takovém případě pak každý člen týmu může pracovat na své části aplikace a neovlivní kolegovu práci. Vhodná struktura je také důležitá pro pozdější úpravy aplikace. Pokud například budeme mít veškeré operace s databází na jednom místě, jistě budou úpravy snadnější, ale také bude snadnější přejít na jinou databázi.

Nejprve se zaměříme na **skripty**. Pro snadné programování a budoucí úpravy aplikace bude výhodné, pokud bude veškeré části aplikace obsluhovat jeden hlavní skript, např. index.php, který umístíme do složky www. Tento skript pak podle parametru v adrese bude volat jednotlivé dílčí skripty, které se budou specializovat na určitý problém. O bezpečném vkládání skriptů se více dočteme v [9] (první kapitola).

Vytvoříme složku www/pages, kde budou umístěny soubory z první úrovně aplikace. Složky a soubory v této složce (a v podsložkách) budou odpovídat struktuře webových stránek, takže máme-li části Requirements, Products, Tests a Test Run, zavedeme pro každou z nich vlastní obslužný skript. Jedná se o soubory requirements.php, products.php, tests.php a test_run.php. Tyto soubory kopírují členění databáze do čtyř hlavních oblastí. K nim přibudou soubory main-page.php pro úvodní stranu a auth.php pro správu uživatelů.

Do složky www/pages také vytvoříme stejnojmenné složky (bez koncovky .php, tedy requirements, products, apod.), které budou obsahovat druhou úroveň aplikace. Zde již budou skripty, které budou vykonávat konkrétnější úlohy. Většinou to bude výpis záznamů, zobrazení záznamu, editace, zobrazení historie a jiné. Stejným způsobem můžeme pokračovat i třetí úrovní, kterou využijeme pro správu kategorií v částech Requirements a Products.

Zcela jistě se bude stávat, že některé skripty v různých částech budou vykonávat stejnou, nebo velmi podobnou funkci, ale s různými daty. Proto vytvoříme složku www/includes,

která bude obsahovat skripty sdílené více stránkami. Např. skript pro výpis záznamů můžeme použít pro všechny části aplikace, protože ve výpisu záznamů se budou lišit minimálně. Podobné budou i některé skripty v rámci jedné části aplikace: vytváření nového záznamu, editace záznamu a kopírování záznamu jsou natolik podobné operace, že bude jistě výhodnější je obsluhovat jedním skriptem umístěným ve složce www/includes. Uživatel však uvidí tři různé stránky (New, Edit a Copy).

Soubory ve složce www jsou určeny pouze provádění operací s daty, jedná se o tzv. aplikační část. Prezentační část, tj. zobrazení informace ve webovém prohlížeči uživatele, budou mít na starosti **šablony**, pro které vytvoříme složku templates. Struktura složky templates bude kopírovat strukturu ve složce www. Každý php skript ve složce www bude mít svou šablonu ve složce templates, které předá proměnné, se kterými má šablona pracovat. Šablony budou obsahovat html kód a také příkazy šablonovacího systému, o kterém pojednává kapitola 3.2.

Knihovny funkcí a celé začleněné aplikace bude obsahovat složka libraries. Rozdělíme je podle několika druhů problémů, které potřebujeme řešit:

- database-mysql.php pro přístup do databáze, kterou je MySQL,
- template-smarty.php pro šablonovací systém (Smarty) a navigaci.
- kernel.php pro všeobecné funkce, které jsou klíčové pro celkový chod aplikace,
- export.php pro export dat do různých formátů.

Začleněné aplikace (WYSIWYG editor, šablonovací systém Smarty, externí knihovny pro export) budou mít vlastní složku. Díky tomu, že máme aplikaci rozdělenou do knihoven podle problémů, které řeší, můžeme aplikaci v budoucnu snadněji upravit pro práci s novými technologiemi. Například knihovnu pro práci s databází, která bude napsána pro MySQL, můžeme upravit pro práci s jiným druhem databáze. Zbytek aplikace měnit nemusíme. Úpravy se tak dotknou jediného souboru. Totéž bude platit pro šablonovací systém a export. Postupně také zjistíme, že některé požadované knihovny (viz. kap. 1) bude lepší řešit přes šablonovací systém (např. navigaci, formulářové prvky a výpis záznamů).

3.1 Načítání skriptů

Vraťme se nyní k **hlavnímu skriptu** index.php a ke způsobu načítání jednotlivých skriptů. Zavedeme zde pole $webdoc, kde budou nastavena všechna důležitá nastavení aplikace. Jedná se např. o cesty ke složkám libraries a templates, názvy skriptů pro přístup do databáze, pro šablonovací systém a pro obecné funkce, dále identifikaci aktuálního záznamu,

přihlášeného uživatele, aj. Pokud máme správně nastavené tyto proměnné, můžeme příkazem require snadno načíst knihovny funkcí ze složky libraries.

Protože celá aplikace pracuje s databází, je vhodné se k ní pomocí funkce database_connect připojit již zde. Výstup funkce si uložíme do proměnné $webdoc["database_connection"], abychom později mohli detekovat, zda jsme k databázi připojeni. Dále je potřeba inicializovat šablonovací systém vytvořením objektu $template. Do něj pak budeme přiřazovat jednotlivé proměnné, se kterými mají šablony pracovat. Do proměnných $menu1_url a $menu1_titles vložíme url adresy a popisky jednotlivých položek první úrovně menu. Tyto proměnné pak přiřadíme příkazem template_assign do šablonovacího systému. Zajistíme tak, že obě proměnné budou přístupné v šabloně.

Jednotlivé položky menu budou obsahovat odkazy na skripty první úrovně, tj. skripty jednotlivých částí aplikace. Tyto skripty určí parametr subpage1. Pouze u hlavní stránky, tj. Main page, parametr subpage1 nebude. Hlavní stránka dané úrovně je totiž určena neexistencí parametru subpage dané úrovně. Chceme-li zajistit, aby se místo odkazu aktuální stránky zobrazil pouze text jejího názvu, nastavíme příslušný prvek v poli $menu1_url na prázdný řetězec. Stejným způsobem, jak bylo právě popsáno, budeme zobrazovat i menu v dalších úrovních, jejichž vypsání zajistí skript nižší úrovně.

Nyní již máme vše potřebné k tomu, abychom mohli načíst skript první úrovně. K tomu bude vhodné vytvořit zvláštní funkci, která bude sloužit k načítání skriptů všech úrovní. Uložíme ji do souboru kernel.php a nazveme ji kernel_require_script. Tato funkce podle parametrů v adrese stránky nastaví příslušné globální proměnné a načte další skript. Tímto skriptem je jeden ze souborů, které přísluší jednotlivým částem aplikace (Requirements, Products apod.). Úloha těchto skriptů bude jednoduchá – jejich funkce bude spočívat pouze v nastavení globálních proměnných charakteristických pro každou část aplikace, nastaví položky menu druhé úrovně a opět pomocí funkce kernel_require_script načtou další podřízené skripty.

Po provedení všech dílčích skriptů se běh programu vrátí na konec skriptu index.php, za funkci kernel_require_script. Do šablonovacího systému přiřadíme globální proměnnou $webdoc. Metoda template_display zařídí zpracování šablon, proměnných přiřazených do šablonovacího systému a vygenerování html výstupu. Po jeho vygenerování ještě nesmíme zapomenout ukončit spojení s databází. To provede funkce database_close. Připojení k databázi ukončíme až po vygenerování obsahu záměrně, aby bylo možné do databáze přistupovat i prostřednictvím šablon, např. pro načtení chybové zprávy apod.

Jak už bylo řečeno, pro **načítání skriptů** v různých úrovních aplikace je vhodné použít funkci, kterou nazveme kernel_require_script a kterou umístíme do soboru libraries/kernel.php. Úkol celé funkce spočívá ve zpracování parametrů v url, podle kterých funkce pozná, který skript má načíst. Parametry nazveme subpage1, subpage2 a subpage3. Parametr subpage1 bude obsahovat jméno skriptu první úrovně, parametr subpage2 bude obsahovat jméno skriptu druhé úrovně, atd.

Funkce bude volána s dvěma parametry $sub a $main. Parametr $sub bude sloužit pro určení hloubky vnoření. Funkce tak pozná, ve které úrovni je volána a z kterého parametru subpage má získat jméno načítaného skriptu. Pokud bude parametr $sub roven 3, bude název načítaného skriptu v parametru subpage3. Zároveň budou parametry nadřazených úrovní, tj. subpage1 a subpage2, udávat cestu k tomuto skriptu.

Vzhledem k tomu, že při přechodu z nadřazené do podřízené úrovně nebudeme znát jméno hlavního skriptu podřízené úrovně a tudíž není ani příslušný parametr subpage v url, využijeme druhý parametr funkce $name pro určení hlavního skriptu dané úrovně. Budeme-li tedy stále uvažovat parametr $sub roven 3, bude název skriptu hledán v parametru subpage3. Pokud však v url tento parametr nebude existovat, automaticky načteme hlavní skript daný parametrem $name. Hlavní skript dané úrovně je tedy určen neexistencí příslušného parametru subpage v url. Pokud by však tento parametr existoval a zároveň by obsahoval jméno hlavního skriptu, měli bychom pro jedno zobrazení dvě různé adresy, což není žádoucí. Provedeme tedy přesměrování na nové url, které již tento parametr nebude obsahovat.

Nyní přistoupíme k samotnému kódu funkce. Deklarujeme globální proměnné $webdoc a $template za klíčovým slovem global, abychom mohli pracovat s proměnnými aplikace i v načítaných skriptech a také abychom v těchto skriptech mohli přiřazovat proměnné. Zkontrolujeme, zda je funkční připojení k databázi. Při nefunkčním připojení k databázi se funkce bez načtení dalšího skriptu ukončí. Protože je celá aplikace bez výjimky závislá na funkční databázi, nemělo by smysl načítat další skript.

Dále si připravíme proměnné $url a $dir. Do proměnné $url budeme v průběhu funkce ukládat novou adresu url pro případné přesměrování. Její výchozí hodnota bude "./". Proměnná $dir bude obsahovat cestu k načítanému skriptu. Obě proměnné naplníme v cyklu for, který bude probíhat od jedné do $sub-1. Při každém průchodu cyklem zkontrolujeme, zda existuje příslušná hodnota subpage. Pokud existuje, připojíme subpage na konec proměnných $url a $dir. Pokud hodnota subpage neexistuje, cyklus se ukončí. Při správném použití funkce by však k tomuto nemělo docházet, protože pokud některý z parametrů neexistuje, neměla by

být vůbec stránka, která funkci volá, načtena. Po ukončení cyklu přidáme na začátek proměnné $dir řetězec "pages/" a budeme tak mít v této proměnné kompletní cestu ke skriptu.

Nyní již zbývá načíst parametr úrovně $sub. Otestujeme, zda tento parametr existuje. Pokud neexistuje, načteme hlavní skript určený parametrem funkce $main. Před načtením ještě provedeme kontrolu existence skriptu a v případě jeho neexistence načteme skript notfound.php, který informuje o neexistenci stránky. Pokud parametr subpage úrovně $sub existuje, otestujeme, zda není shodný s parametrem funkce $main. V takovém případě by se totiž jednalo o hlavní skript a jak bylo napsáno výše, neošetřením tohoto stavu bychom měli dvě různé adresy pro jedno zobrazení. Provedeme tedy přesměrování na novou adresu danou proměnnou $url, která již tento parametr neobsahuje. Neshoduje-li se parametr subpage úrovně $sub s parametrem funkce $main, nejedná se o načítání hlavní stránky, ale jakékoliv jiné. V takovém případě opět otestujeme existenci skriptu a skript načteme.

Většině skriptů v aplikaci bude třeba předávat určité informace, bez kterých by jejich funkce nebyla možná. Takovou informací jsou například složky, ve kterých se nacházejí knihovny funkcí, nadpis stránky, id hodnoty záznamů, se kterými skripty mají pracovat a jiné. Nejvíce se nabízí tyto informace předávat v **globální proměnné**, resp. v jednom asociativním poli, které jsme pojmenovali $webdoc a již jsme ho použili v souboru index.php. Již ve skriptu index.php jsme toto pole naplnili několika hodnotami, zejména cestami a názvy jednotlivých knihoven funkcí a nadpisem stránky. Pro předávání různých, často chybových zpráv uživateli na výstup budeme používat pole $webdoc["message"]. Šablonovací systém pak zajistí automatické vypsání všech zpráv v tomto poli.

Tyto hodnoty jsou pro skripty v celé aplikaci stejné, avšak na nižších úrovních budeme zavádět další hodnoty, které se již mezi úrovněmi budou lišit, případně mohou úplně chybět. Například bude výhodné nastavovat v každé části aplikace jméno hlavní tabulky do proměnné $webdoc[main_table]. Pro část Requirements nastavíme tuto proměnnou na "requirement" (v souboru requirements.php), pro část Products na "product" (v souboru products.php). Dále budeme v každém skriptu, který aplikace načte, přidávat dílčí nadpis dané úrovně do proměnné $webdoc["page_title"]. Šablona pak tuto proměnnou pouze zobrazí, a to jak v titulu okna prohlížeče, tak i na stránce.

Dále budeme potřebovat do pole $webdoc ukládat hodnoty, jejichž získání bude mírně složitější, avšak postup by byl ve všech skriptech stejný. Bude to např. proměnná pro uložení id hodnoty aktuálního záznamu. Jméno proměnné tohoto záznamu může odlišné pro různé části aplikace, avšak postup při získávání této hodnoty z pole $_GET nebo $_SESSION bude

stejný. K tomuto si vytvoříme funkci kernel_set_variables, kterou budou všechny tyto skripty volat. Funkce pak tyto hodnoty do pole $webdoc nastaví.

V příkaze switch nastavíme proměnnou $variable dle obsahu proměnné $webdoc[main_table]. Proměnná $variable bude obsahovat jméno sloupce s id hodnotou hlavní tabulky. Pro tabulky requirement, product a test je to sloupec sequence_id, pro tabulky requirement_category a product_category pak sloupec category_id. Tuto proměnnou budeme hledat nejprve v poli $_GET a pokud tam není, tak v poli $_SESSION. Pokud proměnnou nalezneme, uložíme ji současně do polí $webdoc, $_SESSION a také ji přiřadíme do šablony. V opačném případě ji do pole $webdoc nevložíme, na což musí být skripty připraveny.

Pokud touto proměnnou je sequence_id, uložíme do pole $webdoc i s proměnnou "version" (existuje-li), která se s proměnnou "sequence_id" přímo pojí. Existuje-li proměnná $webdoc["version"], znamená to, že pracujeme se záznamem v historii.

3.2 Šablonovací systém

Šablonovací systém [15] je nástroj, který nám efektivně umožní oddělit aplikační a prezentační část aplikace. Princip je takový, že nejprve vykonáme všechny potřebné operace v php skriptech, ovšem bez výstupu textu nebo html kódu. Výstup zajistí speciální šablony, do kterých ve skriptech přiřazujeme proměnné, která má šablona zobrazit. Šablona je v podstatě soubor s html (a případně css, javascript) kódem, který je doplněný o speciální značky šablonovacího systému umožňující větší možnosti pro generování výstupu, než poskytuje samotný jazyk html.

Pomocí těchto značek můžeme např. snadněji generovat prvky formulářů. Podmínkami můžeme ovlivnit, zda se určitá část kódu vypíše, pomocí cyklů můžeme snadno vypsat celou tabulku. Některé šablonovací systémy umožňují programovat vlastní funkce. Důležitá je také možnost načítání dalších šablon. Jak bude ukázáno dále, všechny tyto možnosti budou hojně využívány.

Ve chvíli, kdy je ve skriptu zavolán příkaz pro zobrazení šablony, přebírá řízení šablonovací systém. Ten šablonu načte a dle vlastní syntaxe převede kód šablony na html výstup. Každý šablonovací systém má svou vlastní syntaxi, kterou je třeba dodržovat. V opačném případě může parsování šablony skončit chybou. Dobré šablonovací systémy také umožňují kompilace šablon do cache paměti, která výrazně celý systém zrychluje. Kompilace spočívá v převedení šablony na speciální php skript obsahující zobrazovací logiku. Při příštím zavolání šablony se již nepřevádí značky šablony, pouze se vykoná tento skript.

Podívejme se na některé **známé šablonovací systémy**. Pro php existuje několik šablonovacích systémů, z nichž jmenujme nejznámější HTMLTmpl, Teng a Smarty. HTMLTmpl [23] je jednoduchý šablonovací systém, který dle jeho tvůrců umožňuje pouze to, co by šablonovací systém umožňovat měl, nic víc. V podstatě dovoluje pouze zobrazovat proměnné, cykly, podmínky a načítat další šablony. Nevýhodou v jeho použití je nepřehledná a možná i nekompletní dokumentace.

Teng [17], [19] má možností více. Nabízí navíc funkce pro práci s řetězci a se slovníky. V podmínkách nabízí více operátorů. Syntaxe je přehledná a jednoduchá. Kromě PHP podporuje Teng i jazyky Python a C++. Zásadní rozdíl oproti jiným šablonovacím systémům je ten, že Teng je zkompilovaný modul pro php přímo na serveru. Díky tomu je mnohem rychlejší. Jeho použití je tak výhodné zejména u velkých projektů s velkým množstvím komplikovaných šablon. Podoba zkompilovaného modulu však může být také nevýhodnou, protože ne vždy máme možnost moduly do php přidávat, zejména to platí u hostingových služeb. To je také důvod, proč Teng nebyl použit v tomto projektu. Více v [16] a [18].

Populární Smarty [22] se možnostmi Tengu docela podobá. Má zřejmě vůbec nejvíce možností, což z něj dělá robustní nástroj. Zejména pro jednoduchou syntaxi a výbornou dokumentaci použijeme Smarty v tomto projektu. Kritici sice říkají, že Smarty je příliš silná zbraň v rukou grafika [15], avšak v této práci je grafická část natolik jednoduchá, že tento argument zde nemá opodstatnění. Z možností, které Smarty umožňuje navíc, můžeme jmenovat např. generování formulářových polí, jednoduché matematické výrazy přímo v šablonách, modifikátory, tvorba vlastních pluginů a velké množství dalších možností [20], [21].

Přistoupíme tedy k **integrovaní šablonovacího systému** Smarty do naší aplikace. Ve složce libraries vytvoříme složku template, kde se budou nacházet soubory šablonovacího systému. V ní si vytvoříme dalších 5 složek:

- smarty, kam nahrajeme všechny soubory Smarty,
- templates_c pro zkompilované šablony,
- configs pro konfigurační soubory,
- cache pro cachování a
- webdoc_plugins pro vlastní funkce.

Pro ukládání vlastních šablon použijeme složku templates, kterou jsme již vytvořili v kořeni aplikace. Ve složce libraries vytvoříme soubor template_smarty.php, který bude zajišťovat inicializaci a zároveň zabalení Smarty příkazů do vlastního objektu. Umožní to případnou snadnější výměnu šablonovacího systému za jiný, kdy by se pouze vyměnil obsah

složky libraries/template, změnil by se soubor template_smarty.php a změnily by se příkazy v šablonách. Příkazy v php skriptech by se však měnit nemusely.

V souboru template_smarty.php musíme spustit inicializační soubor Smarty.class.php, díky čemuž budeme moci pracovat s šablonovacím systémem a využívat jeho metod. Vytvoříme vlastní třídu template, která zdědí všechny metody z třídy Smarty. V této třídě vytvoříme konstruktor, v němž nastavíme cesty k jednotlivým složkám, které jsme vytvořili výše. K určení cesty použijeme globální proměnnou $webdoc["libraries_dir"], kterou jsme definovali v souboru index.php.

Dále si vytvoříme funkce šablonovacího systému. Smarty jich sice má celou řadu, vystačíme si však pouze se třemi. Budou to funkce:

- template_assign pro přiřazení proměnné do šablonovacího systému. Šablona pak bude moci s touto proměnnou pracovat. Parametry funkce jsou jméno proměnné v šablonovacím systému a její hodnota.
- template_clear_assign pro zrušení proměnné z šablonovacího systému. Parametrem je jméno proměnné.
- template_display pro výstup šablony. Jediným parametrem je jméno a případná cesta k souboru šablony.

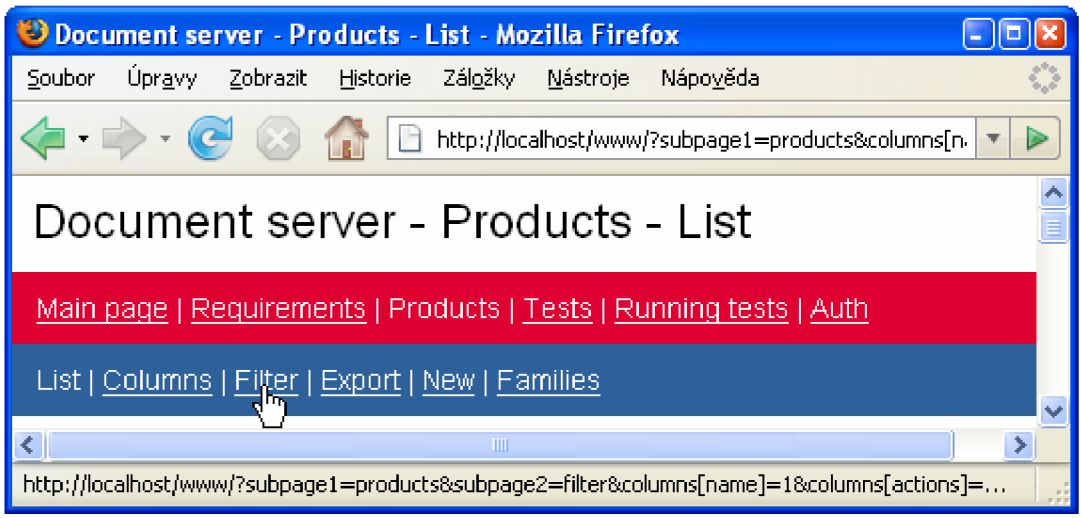
Tím máme šablonovací systém připraven a můžeme začít vytvářet šablony ve složce templates. Tato složka bude mít stejnou strukturu jako složka www. Každému skriptu ve složce pages bude odpovídat šablona ve složce templates, včetně podsložek, takže skript index.php ve složce www bude mít svou šablonu index.tpl ve složce templates. Šablona index.tpl bude také první šablona, kterou vytvoříme. Vypisovat bude pouze část kódu, který je společný pro celou aplikaci. Patří sem:

- doctype dokumentu (XHTML 1.0 Strict dle W3C),
- tagy <html></html>, <head></head> a <body></ body>,
- tagy <meta> informující o použitém kódování (utf-8) a jazyku stránky (en),
- tag <link> pro přidružení kaskádových stylů,
- zobrazení nadpisu v hlavičce stránky (tag <title>),
- zobrazení hlavního nadpisu (tag <h1>) a
- zobrazení první úrovně menu, je-li uživatel přihlášen.

Pro vykreslení menu si napíšeme první Smarty funkci, kterou nazveme menu. Skript uložíme do souboru function.menu.php do složky libraries/template/webdoc_plugins, kterou jsme si pro vlastní Smarty funkce již připravili. Dle pravidel Smarty se musí název souboru

každé funkce skládat z řetězce "function.", názvu funkce a přípony .php. Podobně název funkce ve skriptu musí začínat řetězcem "smarty_function_", takže zde to bude funkce smarty_function_menu. Šablona funkci předá parametr $level pro určení úrovně menu, pole $url s url adresami položek menu a pole $titles s názvy položek menu. Funkce tyto proměnné získá extrahováním z pole $params. Zkontroluje existenci všech proměnných a v cyklu foreach vypíše jejich obsah společně s html kódem menu do proměnné $output. Tuto proměnnou pak funkce vrátí a šablonovací systém ji vypíše.

Menu zobrazíme prostřednictvím seznamu <ul> a jeho položek <li>. V souboru www/style.css vytvoříme třídu menu, která nám zajistí zobrazení menu přes celou stránku, definuje vhodné rozměry okrajů, barvy písma a pozadí. Tag <li> nastavíme tak, aby nezobrazoval odrážky a aby se zobrazoval jako řádkový element, tj. bez řádkového zlomu na konci. Tím dosáhneme, že jednotlivé položky menu se budou zobrazovat vedle sebe, nikoliv pod sebou. Můžeme si také vytvořit další třídy sub1 a sub2, které nám vykreslí pozadí dalších úrovní menu jinou barvou.



Obr. 3.1: Nadpis a menu aplikace v okně prohlížeče

Poslední akcí, kterou je potřeba na konci šablony udělat, je načtení další šablony na nižší úrovni, pokud nižší úroveň existuje. V případě souboru index.tpl nižší úrovně jistě budou existovat. Abychom ošetřili případnou neexistenci šablony a abychom mohli bez znalosti parametru subpage v url načítat výchozí šablony, bude vhodné pro načítání šablon napsat vlastní funkci, která oproti vestavěné funkci include, bude tato ošetření zahrnovat.

3.3 Načítání šablon

Funkci nazveme include_template. Bude provádět totéž, co funkce kernel_require_script, avšak načítat bude šablony. Otestujeme, zda jsme úspěšně připojeni k databázi a pokud nejsme připojeni, načteme šablonu message.tpl, která vypíše chybovou zprávu, funkci ukončíme a načtení se neprovede. Jsme-li připojeni k databázi, může funkce pokračovat dál testem proměnných $sub a $main, které mají stejný význam, jako parametry funkce kernel_require_script, tj. $sub určuje úroveň, na které se nachází šablona a $main určuje jméno hlavní skriptu dané úrovně. Tyto proměnné získáme importem z pole $params. Zavedeme proměnnou $dir, která má opět stejný význam jako ve funkci kernel_require_script, tj. určuje cestu k php skriptu (v době provádění funkce již provedenému), který odpovídá načítané šabloně. V cyklu for naplníme proměnnou $dir obsahem parametrů subpage z url. Podmínka v cyklu omezí parametry pouze na nadřazené úrovně.

Před vlastním načtením je nutné otestovat existenci parametru subpage dané úrovně v url a existenci odpovídajícího php skriptu. Podle výsledku těchto testů pak načteme vlastní šablonu určenou parametrem subpage dané úrovně, nebo hlavní šablonu danou parametrem funkce $main. Pokud skript odpovídající načítané šabloně neexistuje, načteme šablonu notfound.tpl. Je nutné, aby se funkce chovala stejně jako funkce kernel_require_script. Když se nám toto podaří splnit, zajistíme, že obě funkce načtou odpovídající si php skript a šablonu.

Vlastní načtení šablony (ať už jakékoliv) budeme provádět s pomocí Smarty funkce _smarty_include. Tato funkce vyžaduje parametr $params, ve kterém bude proměnná $smarty_include_tpl_file obsahovat jméno a cestu k načítané šabloně. Z tohoto důvodu provedeme v tělech podmínek přiřazení cesty do této proměnné a vlastní načtení provedeme až za těmito podmínkami. Provedeme test existence samotné šablony a v případě existence šablonu načteme. Jestliže šablona neexistuje, načteme z databáze chybovou zprávu o neexistenci šablony, uložíme ji do pole $webdoc["message"], pole $webdoc přiřadíme do Smarty a načteme šablonu message.tpl, kde tuto zprávu zobrazíme.

3.4 Práce s databází

Pro práci s databází, kterou jsme navrhli v první kapitole, zavedeme knihovnu funkcí, díky které budeme k databázi přistupovat. Uložíme ji do souboru database-mysql.php ve složce libraries. Názvy všech funkcí budou začínat na slovem database, aby byla příslušnost funkce v aplikaci ihned zřejmá a aby nedocházelo ke kolizím kvůli stejným názvům funkcí

z jiných částí aplikace. Načtení knihovny funkcí již máme zajištěno přímo v souboru index.php.

Připravíme si několik základních funkcí pracujících s databází. Často budou tyto základní funkce pouze zabalovat do sebe php funkce pro práci s MySQL. Je to z důvodu případného snadnějšího přechodu na jinou databázi. Bude pak stačit pouze vytvořit novou knihovnu funkcí, ale funkce použité ve skriptech nebude třeba měnit. Složitější funkce, které budou vykonávat určité specifické operace se záznamy, již samozřejmě obdobu v zabudovaných php funkcích mít nebudou. Tyto funkce budou popsány samostatně v kapitolách, které se těchto funkcí týkají.

Nejprve se musíme k databázi připojit. Vytvoříme si funkci database_connect, která pomocí příslušných přístupových údajů připojíme funkcí mysql_connect k databázi. Současně zde můžeme vybrat naší databázi pomocí funkce mysql_select_db. Pokud jsou obě operace úspěšné, vrátí funkce true. V opačném případě false.

Podobně vytvoříme další funkce, které budou volat již zabudované funkce pro práci s MySQL v PHP, pouze v názvu bude na místo mysql slovo database. Budou to funkce:

- database_close pro odpojení od databáze,
- database_query pro zadání sql dotazu,
- database_insert_id pro získání id hodnoty posledního příkazu insert,
- database_fetch_array pro načtení jednoho záznamu výsledku dotazu do pole a
- database_num_rows pro zjištění počtu záznamů výsledku dotazu.

Vytvoříme si také funkci database_fetch_result, která bude vracet všechny záznamy výsledku dotazu v poli. Nebude tak nutné v každém výpisu záznamů provádět cyklus nutný pro získání všech záznamů, postačí použít tuto funkci. V prvním argumentu ($result) bude výsledek dotazu a druhý argument ($count) bude předaný odkazem. Uložíme do něj celkový počet záznamů, což nám zjistí funkce database_num_rows. V cyklu for budeme do pole $list ukládat jednotlivé záznamy vrácené funkcí database_fetch_array. Výsledný obsah pole $list funkce vrátí.

4. Operace s jednotlivými záznamy

V této kapitole popíšeme operace, které provádějí přistupují k záznamům jednotlivě. Jedná se o tyto operace:

- vytvoření nového záznamu,
- úprava existujícího záznamu,
- prohlížení záznamu,
- kopírování záznamu,
- schvalování záznamu,
- aktivování záznamu.

Každá u těchto operací bude mít vlastní stránku, např. New, Edit, View, apod. Protože ve všech částech aplikace budou tyto operace stejné nebo podobné, bude výhodné je provádět stejnými skripty pro všechny části. Tyto skripty budou ve složce www/includes. Ve složkách skriptů jednotlivých částí aplikace pak bude pouze malý skript, který si načte další skript ve složce www/includes. Totéž bude platit pro šablony.

Vytvoření nového záznamu a úprava existujícího záznamu je vlastně stejná operace: jedním z požadavků na aplikaci je uchovávání starých verzí záznamů, tj. historie záznamů, takže úprava existujícího záznamu ve skutečnosti znamená vytvoření záznamu nového a zanechání původního záznamu beze změny. Z tohoto důvodu můžeme obě operace sloučit do jedné, i když uživateli se budou jevit odděleně. Přidat sem můžeme ještě další operaci, kterou je kopírování. Opět se jedná pouze o editaci původního záznamu, avšak se vznikem nové historické větve s novým sequence_id.

Obsluhovat je bude skript edit.php ve složce www/includes a šablona edit.tpl ve složce templates/includes. Ve složce www/pages/requirements (stejně tak ve složkách ostatních částí) bude skript edit.php, jehož funkce bude spočívat pouze v načtení skriptu edit.php ve složce templates/includes. Dále ve složce www/pages/requirements zavedeme skripty new.php a copy.php, které si načtou skript edit.php ve stejné složce. Tím zajistíme sloučení obou operací do jedné. Výše uvedené bude stejně platit i pro šablony.

Do menu druhé úrovně zařadíme první položky, které nazveme New, Edit a Copy, jejichž první parametr sequence1 bude udávat část aplikace, ve které se nacházíme a druhý parametr sequence2 bude udávat stránku New, Edit nebo Copy. Způsob zpracování těchto pa-

rametrů a načtení příslušných skriptů byl popsán v kapitole 3.1. U stránky Edit a Copy budeme navíc potřebovat vědět, se kterým záznamem má pracovat. To bude udávat parametr sequence_id, jenž označuje větev celé historie záznamů, z nichž pro editaci načteme ten aktuální.

4.1 Šablona pro uložení záznamu

Nejprve si připravíme formulářové prvky v šabloně edit.tpl ve složce templates/includes. Ještě před zobrazením formuláře však bude dobré zobrazovat zprávy o úspěšnosti ukládání, k čemuž slouží šablona message.tpl ve složce templates. Ošetříme také chybové stavy, při kterých by se formulář neměl zobrazit. Jedním z těchto stavů je odepření přístupu, kdy uživatel nemá práva pro editaci. V takovém případě bude nastavena proměnná $no_display. Ve druhém případě je možné, že dojde ke snaze editovat neexistující záznam, což poznáme podle nastavené proměnné $no_display_form. Pokud jedna z těchto proměnných bude nastavena na true, podmínky if nám zajistí, že se formulář nezobrazí.

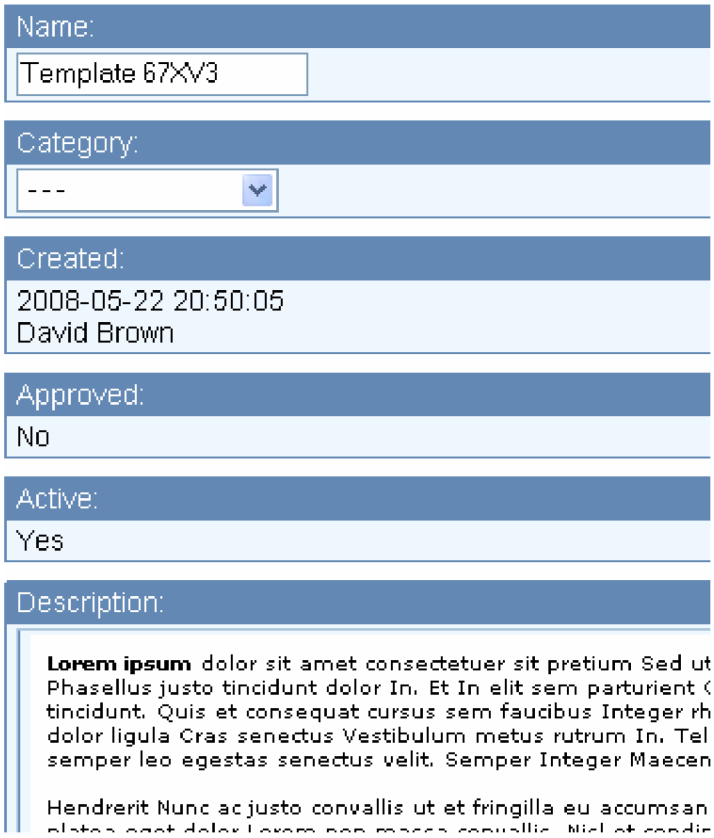
Po ošetření chybových stavů můžeme přejít k samotnému formuláři. Odesílat jej budeme metodou post. Atribut action zanecháme prázdný, což umožní odesílat formulář na stejnou stránku, tedy index.php ve složce www. Jednotlivá formulářová pole začleníme do tabulky třídy record. V kaskádových stylech pak upravíme vzhled záhlaví a buněk tabulky na odstíny modré, podle celkového vzhledu aplikace.

Před každým formulářovým polem zobrazíme nejprve buňku se záhlavím a názvem tohoto pole. Následovat bude buňka tabulky se samotným polem a poté prázdný řádek tabulky. Opticky se tak budou formulářové prvky se záhlavím jevit jako samostatné kolonky, avšak všechny se stejnou šířkou.

Obyčejná textová pole se budou zobrazovat pro vkládání **kratších textových hodnot** do sloupců tabulek, které jsou typu varchar. HTML kód pole (tag <input>) vložíme pomocí funkce insert_form naprogramované ve Smarty [1]. Parametr type="text" zajistí zobrazení právě tohoto pole. Název pole se bude shodovat s názvem sloupce v tabulce databáze. Výchozí hodnota (pro zobrazení dat z databáze, nebo z odeslaného formuláře) bude udávat příslušná proměnná v poli $record. Název této proměnné se bude shodovat s názvem pole. Kód pro formulářový prvek pro vložení jména záznamu bude vypadat takto:

```
<tr><th>Name:</th></tr>
<tr><td>{insert_form type="text" name="name" value=$record.name}</td></tr>
<tr class="space"><td></td></tr>
```

Rozsáhlá textová pole budou sloužit pro vkládání **větších textových hodnot** do sloupců tabulek, které jsou typu text. Takové pole použijeme např. pro uložení popisu požadavku (pole description). Opět použijeme funkci insert_form, kde parametr type nastavíme na hodnotu textarea. Tím nám funkce vloží formulářový prvek textarea. Aplikace TinyMCE, kterou jsme integrovali do naší aplikace, nám tato pole přemění na plnohodnotný WYSIWYG editor s možnostmi formátování [1].



Obr. 4.1: Část formuláře pro editaci záznamu

Dále budeme ukládat informace, ze kterých si uživatel bude moci **vybrat ze skupiny předem daných hodnot**. V databázi tak budeme nastavovat všechny sloupce, které jsou typu int. Tím je např. sloupec active v tabulce requirement, kde je možné nastavit pouze 1, nebo 0. Uživatel však uvidí hodnoty Yes, nebo No. Stejným způsobem budeme ukládat vazby na záznamy v jiných tabulkách. V tomto případě načteme seznam hodnot z příslušné vazební tabulky, které uživateli dáme na výběr. Příkladem je nastavení kategorie, do které záznam náleží. Uživatel sice uvidí jména kategorií, ve skutečnosti se však uloží pouze id hodnota vybrané kategorie.

Možné hodnoty na výběr bude možné nastavit ve výběrovém poli <select>, které bude obsahovat jednotlivé hodnoty (tag <option>). Pro vypsání seznamu použijeme funkci

html_options, která je součástí Smarty. Této funkci předáme název pole a asociativní pole, které bude obsahovat jednotlivé hodnoty. Klíč k hodnotám v tomto poli se bude zobrazovat v atributu value tagu <option>. V případě nastavování vazeb na záznamy v jiných tabulkách tak bude výhodné nastavovat jako klíč id hodnoty těchto záznamů.

Speciálním případem při nastavování vazeb bude možnost **výběru libovolného počtu záznamů** na provázání. Jedná se o nastavování vazeb typu N:N pomocí vazební tabulky v databázi. V části Requirements je to nastavování závislostí, kdy si uživatel bude moci vybrat libovolných počet záznamů, na kterých bude ukládaný záznam závislý.

Hodnoty bude možné vybírat opět pomocí výběrového seznamu, avšak navíc bude potřeba dynamicky zobrazovat tolik seznamů, kolik bude potřeba vazeb. Přidávání a odebírání seznamů umožníme tlačítky s názvy Add a Cancel. K tomu si naprogramujeme vlastní Smarty funkci s názvem insert_id_selections, která bude generovat potřebný kód. Na konci formuláře zobrazíme tlačítko Save sloužící pro odeslání a uložení hodnot z formuláře.

Poslední skupinou formulářových polí budou pole pro **hodnoty, které nebude možné editovat**. Jsou to hodnoty, které vzniknou buď automaticky, nebo na jiném místě aplikace. V podstatě se o formulářové pole nejedná, pouze zobrazíme příslušnou proměnnou v poli $record. Typickým příkladem je zobrazení času vytvoření záznamu nebo zobrazení informace o schválení záznamu, které se provádí na jiném místě aplikace.

Ve formuláři se budou nacházet určité formulářové prvky, které se mají zobrazit pouze v některých částech aplikace a v jiných se zobrazit nedají. Tento problém vyřešíme pomocí podmínky if, ve které se budeme dotazovat na příslušnou hlavní tabulku. Podle ní poznáme, zda daný formulářový prvek zobrazit či nezobrazit.

4.2 Generování výběrových seznamů

Pro dynamické zobrazování výběrových seznamů si naprogramujeme vlastní Smarty funkci insert_id_selections. Této funkci budeme předávat následující parametry:

- $select_name pro nastavení atributu name vygenerovaných tagů <select>,
- $buttons_name pro nastavení atributu name u tlačítek.
- pole $select, které bude obsahovat jména záznamů z propojované tabulky. Klíčem budou id hodnoty příslušných záznamů. Klíče se budou vypisovat v atributu value tagu <option>. Všechny výběrové seznamy budou stejné.
- pole $array_id, které bude obsahovat id hodnoty vybraných záznamů. Toto pole bude podle své délky určovat, kolik seznamů se zobrazí a zároveň které hodnoty v těchto seznamech budou přednastaveny.

Načteme Smarty funkce insert_form a html_options, které použijeme pro vypsání jednotlivých výběrových polí a tlačítek. Zkontrolujeme existenci proměnných $select_name, $buttons_name a $select, bez kterých funkce nemůže pracovat. Existence proměnné $array_id není nutná, pokud proměnná neexistuje, nevypíše se žádný seznam, pouze tlačítko Add.

Výstupní řetězec budeme ukládat do proměnné $output. Do pole $params_form budeme nastavovat hodnoty potřebné pro zobrazení tlačítek funkcí insert_form. Jelikož všechna tlačítka budou mít atribut type nastaven na submit, můžeme již nyní nastavit tuto proměnnou v poli $params_form.

Obr. 4.2: Výběrové seznamy pro nastavení závislostí požadavku

Pokud existuje proměnná $array_id, víme, že zobrazíme alespoň jeden výběrový seznam. Zavedeme pole $params_options, které bude obsahovat parametry pro zobrazení výběrového seznamu funkcí html_options. Všechny seznamy budou stejné, proto hned můžeme do proměnné $params_options["options"] uložit proměnnou $select s hodnotami výběrového seznamu. Do proměnné $params_form["value"] si uložíme hodnotu "Cancel" pro zobrazení tlačítek Cancel za každým seznamem.

V cyklu foreach projdeme všechny hodnoty pole $array_id. Do proměnné $params_options["name"] uložíme jméno příslušného výběrového seznamu. To je určeno proměnnou $select_name a v hranatých závorkách také aktuální klíčem právě procházeného pole $array_id. Všechny zobrazené seznamy tak budou mít stejný název doplněný číslem v hranaté závorce. Tím zajistíme, že po odeslání formuláře získáme vybrané id hodnoty v jednom poli, jehož název byl dán proměnnou $select_name.

Abychom správně zobrazili již dříve nastavené hodnoty v seznamech, uložíme do proměnné $params_options["selected"] obsah proměnné $id, která obsahuje id hodnotu prvku z procházeného pole $array_id. Nyní již máme nastavené všechny parametry pro Smarty funkci html_options, která nám vrátí HTML kód pro zobrazení jednoho výběrového seznamu. Tento kód přidáme do proměnné $output

Do proměnné $params_form["name"] uložíme jméno tlačítka Cancel, které se bude vázat k zobrazenému seznamu. Toto jméno bude složeno z řetězce "button_cancel_", proměnné $buttons_name a proměnnou $key v hranatých závorkách. Při kliknutí na některé

z tlačítek Cancel tak podle klíče v proměnné dané předchozími řetězci poznáme, které z tlačítek bylo stisknuto. Pomocí Smarty funkce insert_form pak získáme HTML kód tohoto tlačítka, který přidáme do proměnné $output.

Po ukončení cyklu foreach máme v proměnné $output všechny výběrové seznamy a k nim tlačítka Cancel, která se mají zobrazit. Pokud se nemá zobrazit žádný seznam, je proměnná $output prázdná. Zbývá přidat tlačítko Add, které bude sloužit pro přidání dalšího seznamu. Do proměnné $params_form["value"] uložíme řetězec "Add" a do proměnné $params_form["name"] uložíme název tlačítka složený z řetězce "button_add_" a proměnné $buttons_name. U názvu tlačítka Add již není potřeba přidávat klíč, protože zde již bude tlačítko pouze jedno. Pomocí funkce insert_form získáme HTML kód tlačítka, čímž máme kompletní HTML kód všech seznamů, který funkce vrátí příkazem return.

4.3 Skript pro uložení záznamu

Skript pro ukládání záznamů uložíme do souboru edit.php ve složce www/includes. Jeho funkce spočívá v načtení dat z databáze, nebo z odeslaného formuláře a zpracování těchto dat tak, aby je šablona mohla zobrazit. Druhou, důležitější funkcí je ukládání dat do databáze.

Stejný skript bude využíván nejen pro editaci záznamu, ale i pro vytvoření nového záznamu nebo pro kopírování záznamu. Proto hned na začátku musíme podle parametru z url adresy subpage2 rozlišit nadpis stránky (Edit, New nebo Copy). Poté zkontrolujeme práva přihlášeného uživatele, zda může provádět editaci záznamu. Pokud tato práva nemá, vypíšeme chybovou zprávu, do šablony přiřadíme proměnnou $no_display a skript ukončíme.

V další podmínce zkontrolujeme existenci proměnné pro **identifikaci úspěšného uložení** v globálním poli $_SESSION. Skript se totiž při ukládání nového záznamu (stránka New) nebo při kopírování záznamu (stránka Copy) bude chovat tak, že ihned po úspěšném uložení přesměruje na stránku Edit, kde bude uživateli umožněno vytvořený nebo zkopírovaný záznam dál editovat. Aby aplikace mohla zobrazit zprávu o úspěšném uložení, musí si tuto informaci pamatovat i po přesměrování. K tomu využijeme již zmíněné globální pole $_SESSION.

Název proměnné v poli $_SESSION bude složen ze jména hlavní tabulky a řetězce "_record_saved". Pokud tato proměnná existuje, znamená to, že byl úspěšně uložen záznam a došlo k přesměrování. Do pole $webdoc["message"] přidáme zprávu o úspěšném uložení, která se společně s ostatními zprávami zobrazí uživateli. Současně z pole $_SESSION zrušíme proměnnou identifikující uložení, aby se zpráva nezobrazovala opakovaně.

Zavedeme proměnnou $button, kde bude uložena informace o tom, zda bylo stisknuto tlačítko a byl tak odeslán formulář. Zjistíme to prostým porovnáním existence všech možných tlačítek formuláře v logickém součtu. Proměnná $button nám bude říkat, zda máme očekávat hodnoty odeslané z formuláře, zpracovat je a znovu je ve formuláři zobrazit, nebo zda máme načíst data z databáze.

Pokud byl formulář odeslán, převedeme všechny hodnoty z formulářů z globálního pole $_POST do pole $record. Poté budeme testovat, zda nebylo stisknuto tlačítko Cancel u dynamicky generovaných výběrových seznamů. Bylo-li na takové tlačítko kliknuto, bude v globálním poli $_POST existovat příslušná proměnná tlačítka. Tato proměnná je typu array a teoreticky by mohla obsahovat více prvků. Proto v cyklu foreach projdeme toto pole a z pole id hodnot v proměnné $record příslušnou hodnotu odstraníme. Klíče v poli tlačítek jsou současně klíče k hodnotám v poli id hodnot, které odstraňujeme.

Stejným způsobem budeme odstraňovat i id hodnoty v poli $record["requirements_id"] v části Products, ke kterému se však vážou pole $record["states_id"] a $record["priorities_id"]. Souvislost hodnot těchto tří polí je zajištěna pomocí stejných hodnot klíčů, takže prvky ze všech tří polí můžeme odstranit stejným způsobem.

Opačnou operací je přidávání výběrového seznamu, kliknul-li uživatel na tlačítko Add. Pokud proměnná příslušného tlačítka existuje, přidáme na konec pole id hodnot další prvek. Kromě pole $record["requirements_id"] v části Products přidáme nové prvky se stejným klíčem i do polí $record["states_id"] a $record["priorities_id"].

Pro editaci nebo kopírování budeme potřebovat před samotnými úpravami **načíst hodnoty záznamu** z databáze. Přidáme podmínku, ve které budeme parametr subpage2 v url testovat na hodnoty "edit" nebo "copy". Otestujeme také, zda existuje proměnná $webdoc["sequence_id"], která určuje příslušný záznam. Pokud neexistuje, skript ukončíme, neboť není co upravovat (kopírovat). V případě existence této proměnné načteme původní záznam pomocí funkce database_get_record (viz. kapitola 4.5).

Protože zatím nevíme, zda byl odeslán formulář a zda dojde k uložení nové verze záznamu do databáze, uložíme záznam do dočasné proměnné $tmp. Tuto operaci provedeme v rámci podmínky if. V případě neúspěšného načtení nebo neexistence záznamu načteme chybovou zprávu a zakážeme zobrazení formuláře nastavením proměnné $no_display_form ve Smarty.

Proběhlo-li načtení záznamu v pořádku a současně nedošlo ke kliknutí na žádné z tlačítek, uložíme celý záznam z dočasné proměnné $tmp do proměnné $record. Tím zajistíme zobrazení všech hodnot záznamu ve formuláři. Pokud však bylo kliknuto na některé z tlačítek,

kromě tlačítka Save, máme již v poli $record hodnoty převzaté z globálního pole $post a nebudeme tedy záznam načítat z databáze. Výjimkou jsou hodnoty, které ve formuláři nelze editovat a které se neodesílají, pouze se v něm zobrazují. Jsou to údaje o času uložení, schválení a o uživatelích, kteří záznam uložili a schválili. Tyto hodnoty převezmeme z dočasné proměnné $tmp. V této chvíli již pole $record máme naplněno a můžeme jej přiřadit do Smarty.

V další části skriptu načteme hodnoty pro **výběrové seznamy**. První načteme seznam kategorií. Vhodné bude zobrazovat kategorie ve stromové struktuře, pro jejíž získání je potřeba složitější algoritmus. K tomu si využijeme samostatnou funkci, které se věnuje kapitola 6.1. Získané pole kategorií uložíme do Smarty. Jednodušší bude vytvoření výběrového seznamu pro nastavení polí Active a Is product specific. Tento seznam bude mít pouze dvě položky: Yes a No, jejichž přesné hodnoty získáme funkcí database_get_message.

Pro vytvoření seznamu závislostí použijeme funkci database_get_list, která bude primárně určena pro získání výpisu záznamů ve skriptu list.php (viz. kapitola 5.5). Získané záznamy uložíme do proměnné $list. Vytvoříme pole $dependences, jehož první hodnota bude řetězec "- - -" s klíčem NULL. V cyklu foreach překopírujeme záznamy z pole $list do pole $dependences tím způsobem, že hodnoty pole $dependences budou tvořit jména záznamů. Klíče budou jejich id hodnoty. Vynecháme pouze právě modifikovaný záznam, tedy záznam o hodnotě $webdoc["sequence_id"]. Po zkopírování pole $dependences přiřadíme do Smarty. Celý proces vytváření výběrového seznamu ukazuje následující kód:

```
$list=database_get_list($rows,$total); //získání záznamů
$dependences=Array(NULL=>"- - -"); //výchozí hodnota výběrového seznamu
foreach ($list as $key => $row) //průchod polem $list
{
  //vynechání modifikovaného záznamu
  if ($row["sequence_id"]!=$webdoc["sequence_id"])
  {
    //převod jména a id hodnoty záznamu do asociativního pole
    $dependences[$row["sequence_id"]]=$row["name"];
  }
}
```

Podobným způsobem vytvoříme pole $requirements pro zobrazení výběrového seznamu vázaných požadavků v části Products. Zde však nemůžeme použít funkci database_get_list, která automaticky vrací záznamy z tabulky dle parametru subpage2 v url. V tomto případě by vracela produkty, my však potřebujeme vracet požadavky. Dalším problémem je nutnost zobrazovat ve výběrovém seznamu i požadavky, které již nejsou aktuální, ale byly dříve vybrané. Jinak řečeno, je to případ, kdy jsme v minulosti udělali vazbu mezi pro-

duktem a tehdy aktuálními požadavky. Požadavky však později mohly být upraveny, tím pádem nám zůstala vazba na neaktuální požadavky, kterou je potřeba zachovat. Proto je nutné tyto neaktuální požadavky do výběrového seznamu přidat.

Tyto problémy vyřešíme napsáním nové funkce, kterou pojmenujeme database_get_select. Prvním parametrem bude název tabulky, ze které záznamy získáváme (v tomto případě requirement), druhý parametr bude uchovávat id hodnoty požadavků, které již byly dříve provázány s aktuálním záznamem tabulky product a které již nemusí být aktuální. V tomto případě jsou pod pojmem id hodnoty myšleny hodnoty requirement_id, nikoliv sequence_id, jinak by nebylo možné znovu ukládat vazby na neaktuální záznamy.

V poli $records máme id hodnoty již uložených záznamů. Zkontrolujeme, zda je proměnná typu array a odstraníme duplicitní prvky pole. Poté ho projdeme cyklem foreach a zkontrolujeme, zda je každá id hodnota pole větší nebo rovna nule. Pokud tuto podmínku splňuje, vytvoříme v části WHERE novou podmínku dotazu, kterou přidáme do proměnné $where:

```
$where.="OR m.$table"."_id=$id ";
```

$table je v tomto případě název tabulky, na kterou provádíme dotaz. Uvedená podmínka nám zajistí, že dotaz vrátí dříve uložené záznamy i v případě, že již nejsou aktuální. Příkazem switch ($table) provedeme rozlišení dle prohledávané tabulky, protože tato funkce bude použita i pro výpis stavů a priorit, pro něž bude dotaz jednoduchý, avšak dotaz pro vrácení požadavků bude složitější:

```
"SELECT m.name, m.$table"."_id, m.sequence_id, (
  (
    SELECT MAX(version) FROM $table
    WHERE sequence_id=m.sequence_id
  )>m.version
) AS old
FROM $table m WHERE m.$table"."_id
IN (
  SELECT MAX($table"."_id) FROM $table GROUP BY sequence_id
)
$where ORDER BY sequence_id, requirement_id"
```

První poddotaz zjistí maximální hodnotu sloupce version ze všech záznamů, které mají stejnou hodnotu sequence_id jako má vrácený záznam. Maximální hodnota se porovná s hodnotou version vráceného záznamu. Výsledek porovnání se uloží do proměnné old. Pokud vrácený záznam není aktuální, bude jeho hodnota version menší než maximální hodnota ze všech sloupců version a proměnná old bude obsahovat hodnotu 1. V opačném případě bude obsahovat hodnotu 0. Druhý poddotaz zajistí vypsání pouze aktuálních záznamů, ale násle-

dující podmínka z proměnné $where tuto podmínku obejde a umožní vypsání záznamů, které již byly dříve ve vazbě s editovaným záznamem tabulky product.

Dotaz vykoná funkce database_fetch_result. Jeho výsledek bude uložen do proměnné $list ve skriptu edit.php, ve kterém budeme pokračovat dále. Do pole $requirements uložíme obsah pole $list upravený pro zobrazení ve výběrovém seznamu. K názvům záznamů, které mají proměnnou $old rovnu 1, budeme připojovat řetězec " --[OLDER VERSION]--", aby bylo možné rozlišit novou a starou verzi záznamu. Pomocí funkce database_get_select a cyklu foreach naplníme pole pro zobrazení výběrových seznamů pro výběr stavu a priority.

V další části skriptu provedeme **kontrolu formulářových polí**, pokud bylo kliknuto na tlačítko Save. Bude-li mít některé z těchto polí neplatnou hodnotu, načte se z databáze chybová zpráva, která se uloží do pole $webdoc["message"]["form_error"]. Obsah tohoto pole pak zobrazí šablona. Nejprve otestujeme obsah proměnných $record["name"] a $record["description"], které nesmí být prázdné. Poté otestujeme existenci zvolené kategorie. K tomu vytvoříme jednoduchou funkci database_test_existing_category, která toto ověří v databázi. Pokud dotaz vrátí jeden záznam, vrátí funkce hodnotu true. V opačném případě vrátí false a načte se chybová zpráva.

Dále budeme testovat existenci záznamů v tabulce requirement, jejichž id hodnoty jsou ve smyslu závislostí uloženy v poli $record["dependences_id"]. K tomuto účelu vytvoříme funkci database_test_existing_records, jejíž první parametr $records_id bude obsahovat id hodnoty záznamů pro testování. V druhém parametru uvedeme jméno tabulky, ve které testujeme existenci záznamu. V tomto případě to bude tabulka requirement. Pokud nastavíme třetí parametr $self na true, bude funkce testovat také přítomnost hodnoty z proměnné $webdoc["sequence_id"] v poli $records_id. V takovém případě by to znamenalo, že se snažíme uložit závislost sama na sobě a funkce proto vrátí false.

Podle jména tabulky nastavíme proměnnou $name_id, do které vložíme název sloupce, jehož id hodnoty jsou v poli $records_id. Zkontrolujeme, zda proměnná $records_id má nějaké prvky. Pokud by pole bylo prázdné, nebyla zřejmě žádná závislost zadána a funkce vrátí true. V dalším testu zkontrolujeme, zda pole $records_id je skutečně pole. Kdyby nebylo, vrátili bychom false. Dále z pole $records_id odstraníme duplicitní prvky. Je-li proměnná $self nastavena na true, otestujeme přítomnost prvku $webdoc["sequence_id"] v poli $records_id.

Hodnoty pole $records_id projdeme v cyklu foreach. U neplatných hodnot vrátí funkce false. V opačném případě přidáme id hodnotu do řetězce $where, který bude obsahovat část dotazu za klauzulí WHERE. Vykonáme dotaz a pokud bude počet vrácených zá-

znamů stejný jako počet prvků pole $records_id, vrátíme true. Bude-li však vrácených záznamů méně, znamená to, že záznamy s některou z hodnot z pole $records_id se v tabulce zřejmě nenacházejí. Funkce pak vrátí false. Stejnou funkci použijeme i pro testování existence záznamů v polích $record["requirements_id"], $record["states_id"] a $record["priorities_id"].

Opět se vrátíme do skriptu edit.php, kde již máme provedeny všechny možné testy formulářových polí. Pokud při některém testu došlo k chybě, poznáme to podle existujícího pole $webdoc["message"] a provádění skriptu ukončíme příkazem return. K uložení záznamu nedojde a šablona zobrazí chybové zprávy z tohoto pole.

Nedošlo-li k žádné chybě, pokračuje skript samotným **uložením záznamu**. Záznam v poli $record_id uložíme pomocí funkce database_save_record. Funkci zavoláme v podmínce if, abychom mohli testovat výsledek funkce, který udává úspěšnost ukládání. Funkce navíc nastaví parametry předané odkazem, tj.

- $record_id, obsahující id hodnotu nově uloženého záznamu (např. requirement_id),
- $sequence_id je sequence_id historické větve záznamu a
- $version, udávající verzi záznamu.

Pokud bylo uložení záznamu úspěšné, měla by již být známa hodnota $webdoc["sequence_id"] uloženého záznamu. Do proměnné $tmp znovu načteme uložený záznam. Uděláme to proto, abychom mohli zjistit čas uložení a případného schválení a uživatele, kteří záznam uložili a schválili. Tyto hodnoty doplníme do pole $record a znovu je přiřadíme do šablony tak, aby se mohly zobrazit ve znovu načteném formuláři. Jestliže se uložení nezdařilo, načteme chybovou zprávu, kterou pak zobrazí šablona. Jak bude ukázáno na konci následující kapitoly, může funkce database_save_record provést přesměrování a k těmto operacím již nemusí dojít.

4.4 Uložení záznamu do databáze

Pro ukládání záznamů si vytvoříme funkci s názvem database_save_record, které předáme parametr $record se záznamem, který chceme uložit. Funkce bude ukládat záznamy jak ze stránky Edit, tak i ze stránek New a Copy. To povede k určitým odlišnostem, které bude třeba respektovat. Při operaci kopírovaní bude potřeba zjistit id hodnotu (např. requirement_id) kopírovaného záznamu. Pokud je tedy v url parametr subpage2 roven "copy", provedeme dotaz, který načte záznam s hodnotou $webdoc["sequence_id"], která ještě v této chvíli obsahuje hodnotu sequence_id kopírovaného záznamu. Kopírujeme-li starší záznam z historie, bude existovat proměnná $webdoc["version"], která dotaz omezí na jediný záznam.

Pokud kopírujeme aktuální záznam, proměnná $webdoc["version"] existovat nebude a v takovém případě vrátí dotaz aktuální záznam.

V každém případě musí dotaz vrátit jeden záznam. Pokud se tak nestane, vrátí funkce false a kopírování se nezdaří. Záznam načteme do pole $parent a otestujeme, zda je v něm jeho id hodnota. Tuto id hodnotu pak uložíme v následujícím dotazu. Tento se bude provádět pouze při kopírování nebo vytváření nového záznamu. V tabulce requirement_parent (resp. product_parent) vytvoří nový záznam. Založíme tak novou historickou větev s novou hodnotou sequence_id, kterou získáme funkcí database_insert_id a uložíme do proměnné $sequence_id. Tato proměnná je parametr funkce volaný odkazem, takže její hodnotu získá i skript, který funkci volal. Pokud byl ukládaný záznam zkopírován, uložíme id hodnotu zkopírovaného záznamu do sloupce parent_id. V opačném případě uložíme hodnotu NULL. Proměnnou $version, která bude udávat verzi ukládaného záznamu (v hlavní tabulce) nastavíme na nulu, jelikož se jedná o novou historickou větev.

Při editaci tento dotaz provádět nebudeme, protože hodnotu sequence_id již v příslušně tabulce uloženu máme. Naopak provedeme dotaz do hlavní tabulky (např. requirement), který vrátí maximální hodnotu sloupce version u záznamů s hodnotou $webdoc["sequence_id"]. Dotaz je nastaven tak, aby při použití funkcí extract a database_fetch_array byla tato hodnota uložena do proměnné $version.

Dotaz pro uložení samotných dat bude sestaven v několika krocích, proto jej uložíme do proměnné $query. V prvním kroku nastavíme sloupce společné pro tabulky requirement a product. Ve druhém kroku přidáme sloupce, které obsahuje pouze tabulka requirement. Totéž uděláme pro vložení samotných hodnot z pole $record do dotazu. Do dotazu vložíme proměnné $sequence_id a $version, které jsme získali v předchozích částech funkce a také hodnotu person_id uživatele, který provedl uložení. Čas uložení zjistíme pomocí databázové funkce now. U některých hodnot, které nemusely být uživatelem zadány, ošetříme vloženou podmínkou, zda tyto hodnoty existují. Pokud neexistují, vložíme do dotazu hodnotu NULL. Dotaz provedeme a pokud byl výsledek úspěšný, zjistíme id hodnotu nově uloženého záznamu.

Pro speciální vazby typu N:N na nově uložený záznam musíme vytvořit zvláštní dotaz. Postup pro vytváření dotazu je velice podobný postupu při sestavení dotazu pro testování existence záznamů ve funkci database_test_existing_records. Zde však sestavujeme dotaz pro vložení záznamů do databáze, kdežto v uvedené funkci byl sestavován dotaz pro čtení záznamů z databáze. Tímto způsobem uložíme závislosti na jiných záznamech v části Requirements. Vazbu na požadavky v části Products uložíme obdobně, navíc budeme ukládat i stavy

a priority z polí $record["states_id"] a $record["priorities_id"]. Souvislost mezi hodnotami těchto polí a pole $record["requirements_id"] zajišťují stejné klíče v těchto polích.

Nyní již máme uložena všechna data i vazby na jiné tabulky. Pokud jsme prováděli modifikaci záznamu na stránce Edit, funkce vrátí true. Pokud jsme vytvářeli nový záznam na stránce New nebo jsme kopírovali původní záznam na stránce Copy, umožníme uživateli další modifikaci záznamu přesměrováním na stránku Edit.

Zavedeme proměnnou $url, do které vložíme adresu pro přesměrování, jež se od současné adresy bude lišit v parametru subpage2, který nastavíme na "edit". Do pole $_SESSION uložíme aktuální hodnotu z proměnné $sequence_id, což využijeme při přecházení mezi částmi aplikace. Ta si bude i nadále pamatovat hodnotu sequence_id záznamu, se kterým jsme naposledy pracovali a v menu zobrazí odkazy na operace s tímto záznamem.

Do pole $_SESSION také uložíme proměnnou, která ponese informaci o úspěšném uložení záznamu. Při příštím načtení stránky bude díky této proměnné zobrazena zpráva o úspěšném uložení. Název této proměnné bude sestaven z názvu tabulky, do které byl záznam ukládán a z řetězce "_record_saved". V každé části aplikace se proměnná bude jmenovat jinak, čímž zamezíme zobrazení zprávy v jiných částech aplikace, než se zobrazit má.

Funkce header zajistí přesměrování prohlížeče na novou stránku, jejíž adresa je dána proměnnou $url. Pokud prohlížeč nepodporuje přesměrování, zobrazíme zprávu o úspěšném uložení (na jinak prázdné stránce) a nabídneme odkaz pro přechod na stránku, na kterou jsme chtěli přesměrovat. Skript pak ukončíme příkazem exit.

4.5 Zobrazení záznamu

V tomto okamžiku již aplikace zvládá vytváření záznamů. Aby si mohli uživatelé **záznamy prohlížet** bez nutnosti editace (nebo nemají-li pro editaci práva), vytvoříme stránku View, kde záznam zobrazíme. Obslužný skript view.php ve složce www/includes bude jednoduchý. Jestliže bude existovat proměnná $webdoc["sequence_id"], načte záznam pomocí funkce database_get_record do proměnné $record a přiřadí jej do Smarty. Tato funkce musí kromě načtení dat z hlavní tabulky načíst také data z provázaných tabulek. Dotaz sestavíme takto:

```
"SELECT r.*, CONCAT(pc.name,' ',pc.surname) AS creator,
CONCAT(pa.name,' ',pa.surname) AS approver, c.name AS category, (
  (
    SELECT MAX(version) FROM $webdoc[main_table]
    WHERE sequence_id=$webdoc[sequence_id]
  )=r.version
) AS current,
```

```
rp.parent_id, rpr.name AS parent
FROM $webdoc[main_table] r
LEFT JOIN person pc ON r.creator_id=pc.person_id
LEFT JOIN person pa ON r.approver_id=pa.person_id
LEFT JOIN $webdoc[main_table]_category c ON r.category_id=c.category_id
LEFT JOIN $webdoc[main_table]_parent rp ON r.sequence_id=rp.sequence_id
LEFT JOIN $webdoc[main_table] rpr ON p.parent_id=rpr.$webdoc[main_table]_id
WHERE r.sequence_id=$webdoc[sequence_id]".(isset($webdoc["version"])?"
AND r.version=$webdoc[version]":"").
ORDER BY r.version DESC LIMIT 1"
```

Hlavní tabulku propojíme s tabulkou person a to hned dvakrát. Jednou pro zjištění uživatele, který záznam uložil a podruhé pro zjištění uživatele, který záznam schválil. Jméno a příjmení uživatele sloučíme do jedné proměnné pomocí funkce CONCAT. Pro zjištění jména kategorie provedeme propojení s tabulkou requirement_category (resp. product_category). Propojením s tabulkou requirement_parent (resp. product_parent) a následně opět s hlavní tabulkou zjistíme rodičovský záznam, byl-li vypisovaný záznam kopírován.

Načítáme-li záznam z historie, bude existovat proměnná $webdoc["version"], která rozlišuje mezi verzemi záznamů se stejným sequence_id. Tato proměnná omezí výpis na tento záznam. V opačném případě výpis omezíme podle proměnné $webdoc["sequence_id"]. Vrácení aktuálního záznamu dosáhneme sestupným seřazením podle sloupce version (klauzule ORDER BY) a omezením výpisu záznamů na první z nich (klauzule LIMIT). Poddotaz v části SELECT nám umožní zjistit, zda je záznam aktuální, nebo je z historie.

Pro získání záznamů z vazebních tabulek, které jsou vázány vazbou typu N:N, musíme použít samostatný dotaz. Dotaz pro načtení závislostí v části Requirements bude mít následující podobu:

```
"SELECT *, r.name AS dependence
FROM requirement_dependence rd
LEFT JOIN requirement r ON rd.dependence_id=r.sequence_id
WHERE r.requirement_id
IN (
  SELECT MAX(requirement_id)
  FROM requirement GROUP BY sequence_id
)
AND rd.requirement_id='$record[requirement_id]'"
```

Tabulku requirement_dependence propojíme s tabulkou requirement. Výpis omezíme na hodnotu rd.requirement_id zobrazovaného záznamu. Tyto záznamy jsou vázány přes sloupec rd.dependences_id, což je hodnota sequence_id historické větve záznamů, ze které potřebujeme získat aktuální záznam. Musíme zadat další omezující podmínku, jinak by nám dotaz vypsal záznamy z celé historické větve. To uděláme pomocí poddotazu v části WHERE, který

nám vrátí hodnotu requirement_id aktuálního záznamu. Výpis pak omezíme pouze na tuto hodnotu.

Dotaz provedeme a hodnoty dependence_id vrácených záznamů naplníme do pole $record["dependences_id"]. Toto pole bude využito pro editaci závislostí, která je popisována v kapitole 4.3. Do pole $record["dependences"] naplníme jména vrácených záznamů, které zobrazíme na stránce View nebo ve výběrových seznamech při editaci.

Obr. 4.3: Část stránky pro zobrazení záznamu

Podobně načteme i vázané požadavky na produkt. Dotaz bude jednodušší, protože zde načítáme hodnoty requirement_id záznamů v tabulce requirement_has_product, omezené podle hodnoty product_id zobrazovaného záznamu. Hodnotu product_id již máme načtenu z prvního dotazu funkce. V následujícím cyklu for uložíme id hodnoty a názvy požadavků, stavů a priorit do příslušných buněk v poli $record.

Hodnoty tohoto pole zobrazíme v šablonou view.tpl ve složce templates/includes. Pokud proměnná $record.current nemá hodnotu 1, zobrazíme upozornění, že prohlížíme záznam z historie. K zobrazení dat použijeme tabulku třídy record, kterou jsme použili pro zobrazení

editačního formuláře. Většina buněk tabulky bude stejná, pouze místo formulářového pole hodnotu přímo vypíšeme.

Tam, kde se zobrazoval výběrový seznam, zobrazíme nastavené hodnoty z tohoto seznamu. Jméno kategorie máme uloženo v proměnné $record.category, jméno rodičovského záznamu máme v proměnné $record.parent. Názvy požadavků vázaných k produktu máme uloženy v poli $record.requirements. Toto pole vypíšeme v cyklu foreach, jednotlivé požadavky oddělíme zalomením. Stejným způsobem vypíšeme pole závislostí v části Requirements.

V závěru této kapitoly popíšeme dvě stránky, jejichž součástí bude také šablona view.tpl. Na těchto stránkách bude možné provádět operace **aktivování a schválení záznamu**. Ve složce templates/includes vytvoříme dvě nové šablony: activation.tpl a approving.tpl. Tyto šablony budou obsahovat jednoduchý formulář s tlačítkem pro aktivování, nebo deaktivování, resp. schválení záznamu. Pak jen načtou šablonu view.tpl, která záznam zobrazí, aby bylo jasné, který záznam je aktivován, resp. schvalován.

Ve složce www/includes vytvoříme skripty activation.php a approving.php. Oba skripty nejprve načtou záznam do pole $record. Poté zkontrolují, zda bylo kliknuto na tlačítko pro aktivování, resp. schválení záznamu. V takovém případě zavolají funkci database_active_record, resp. database_approve_record, které záznam aktivují nebo schválí. Funkce database_active_record nastaví hodnotu ve sloupci na 1, funkce database_approve_record nastaví čas schválení a hodnotu person_id uživatele, který záznam schválil. Pokud byla operace úspěšná, načte se příslušná zpráva a záznam se znovu načte, abychom mohli zobrazit správně upravené údaje.

Ve skriptu activation.php navíc testujeme kliknutí na tlačítko pro deaktivování. V takovém případě se deaktivování provede opět funkcí database_active_record, ovšem bez druhého parametru. Podle něj funkce rozpozná, zda se jedná o aktivování, nebo deaktivování. Opačná operace pro schválení není možná, proto již skript approving.php nic neprovádí. Na konci obou skriptů se přiřadí pole $record do šablony.

5. Výpis záznamů

Zatímco dosud jsme se zabývali operacemi s jednotlivými záznamy, v této kapitole uživatelům umožníme záznamy vypisovat do přehledných tabulek. Z těchto výpisů budou uživatelé moci přistupovat k jednotlivým stránkám a na nich provádět operace, které již máme hotové. To umožní sloupec Actions, ve kterém budou odkazy na tyto operace.

Pouze pro výpis záznamů by nám stačilo několik řádků kódu. Zde však budeme pracovat s větším množstvím dat, proto by aplikace měla nabízet další možnosti práce s výpisem. Abychom uživateli co nejvíce zpříjemnili práci se záznamy, zavedeme 4 nástroje, kterými bude možné výpis záznamů ovlivnit. Budou to:

- **stránkování** pro rozdělení delšího výpisu záznamů na stránky. Možné bude zvolit číslo stránky a počet záznamů na stránce,
- **řazení záznamů** vzestupně i sestupně podle různých sloupců,
- **volba zobrazovaných sloupců**: ve výpisech bude možné zobrazovat velké množství různých informací ve sloupcích. Zobrazování všech sloupců by bylo nepřehledné, uživatel by byl nucen horizontálně rolovat. Proto umožníme výběr pouze těch sloupců, které budou uživatele zajímat,
- **filtrování** je mocný nástroj, který významně přispívá k rychlému vyhledání požadovaného záznamu dle zadaných kritérií.

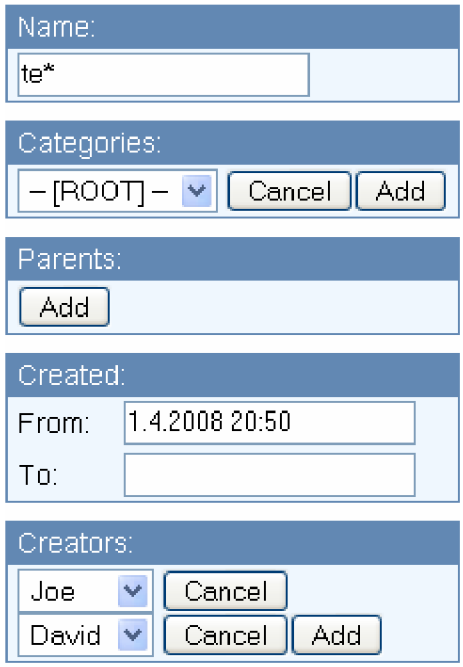
Je zřejmé, že všechny tyto nástroje se budou ovlivňovat a ovlivní také výsledný dotaz do databáze. Pro výpis záznamů bude sloužit stránka List, která také bude výchozí stránkou všech částí aplikace. Na této stránce bude možné nastavit stránkování a řazení. Nastavení filtru a sloupců umožníme na samostatných stránkách Filter a Columns. Jejich nastavení budeme přenášet v parametrech url adresy. Je však nereálné tyto parametry přenášet ve všech částech aplikace, proto jej budeme přenášet pouze mezi zmíněnými stránkami. Budeme je ukládat také do cookies, takže aplikace si je bude pamatovat i po odhlášení. Funkce pro načtení, nebo uložení parametrů z/do url budeme programovat v souboru kernel.php ve složce libraries.

5.1 Šablony pro nastavení filtru a sloupců

Stránka Filter bude obsahovat formulář, kde bude možné nastavit parametry filtru. Protože budeme nastavení všech nástrojů přenášet v url, bude výhodné formulář odesílat me-

todou get. To bude vyžadovat vygenerování parametrů ostatních nástrojů (stránkování, řazení a sloupců) do sady formulářových polí typu hidden, aby se tato nastavení v url zachovala.

To nám pro každý nástroj zajistí speciální minišablona umístěná ve složce templates. Jedná se o šablony pages_hidden.tpl, sort_hidden.tpl a columns_hidden.tpl. Samozřejmě i pro filtr bude nutné vypsání jeho nastavení do formulářů na jiných stránkách, proto i on bude mít svou šablonu filter_hidden.tpl. Každá z těchto minišablon bude obsahovat pouze kód pro vypsání skrytých formulářových polí. Způsob ukládání parametrů některých nástrojů v polích umožňuje tato pole vypsat v cyklu foreach.



Obr. 5.1: Část stránky pro nastavení filtru

Názvy formulářových polí filtru budou dány názvy sloupců v tabulkách, kterým daný parametr filtru omezuje vrácené záznamy. Výchozí hodnoty těchto polí budou předány v poli $filter, jenž naplní obslužný skript filter.php. Parametry bychom mohli rozdělit do několika skupin podle typu sloupce v tabulce, kterého se parametr týká.

První skupinou budou **textové parametry**, které budou omezovat výpis podle sloupců typu varchar a text, nad kterými je fulltextový index. Zařadit bychom sem mohli např. pole Name pro filtrování podle názvu záznamu nebo pole Description pro filtrování podle popisu záznamu. Nastavovat je bude možné pomocí formulářových polí typu text. Zadáním libovolného řetězce do tohoto formulářového pole se výpis omezí na záznamy, které v příslušném sloupci obsahují zadaný řetězec. Fulltextové vyhledávání v MySQL však podporuje i některé operátory [14], kterými je možné ovlivňovat váhu jednotlivých slov zadaných ve formulářo-

vém poli, vypisování záznamů, které zadaný výraz neobsahuji, a jiné. Podporovány jsou též zástupné znaky, jež mají význam libovolného znaku nebo libovolného počtu znaků.

Textová formulářová pole budou umožňovat nastavování také **časových parametrů** filtru. Ty budou omezovat výpis podle sloupců typu datetime, ve kterých se ukládá čas vytvoření záznamu, čas schválení záznamu a jiné. Pro nastavení jednoho takového parametru použijeme dvě textová formulářová pole, aby bylo možné zadat začátek a konec intervalu, do kterého mají spadat časové údaje vypisovaných záznamů. Bude zde platit jediná výjimka v názvech formulářových polí: aby bylo možné jednotlivé dvojice polí od sebe odlišit, bude jejich název doplněn o řetězec "_from", nebo "_for".

Významnou skupinou filtru budou dynamicky generované výběrové seznamy pro nastavení **parametrů z vazebních tabulek**. Pomocí výběrových seznamů umožníme filtrovat záznamy, které jsou svázané se záznamy z vazebních tabulek. Jména záznamů z vazební tabulky nabídneme ve výběrových seznamech. Pomocí tlačítek Add a Cancel umožníme tyto záznamy přidávat a odebírat, takže bude možné filtrovat podle více vázaných záznamů.

Způsob přidávání a odebírání výběrových seznamu bude stejný jako u nastavování závislostí při editaci požadavku, včetně vlastní Smarty funkce insert_id_selections pro generování HTML kódu všech výběrových seznamů, kterou jsme popsali v kapitole 4.2. Tímto způsobem bude možné nastavovat filtrování podle kategorií, rodičovských záznamů, uživatelů (kteří záznamy vytvořili, schválili, apod.), závislostí, atd.

Poslední skupinou bude nastavování **parametrů typu pravda/nepravda**. Tyto parametry budou filtrovat sloupce, ve kterých je možné nastavit pouze jednu z těchto dvou hodnot, jakou jsou například sloupce active nebo is_product_specific v tabulce requirement. K nastavování těchto parametrů postačí jeden výběrový seznam o třech položkách: první položka umožní daný parametr filtru nepoužít, další dvě položky umožní filtrovat záznamy pouze s jednou, nebo druhou hodnotou.

Rozdílné sloupce v jednotlivých hlavních tabulkách opět povedou k tomu, že budeme muset omezovat zobrazení některých parametrů filtru podle proměnné $webdoc.main_table. Jinak bude stránka Filter (stejně jako Columns a List) společná pro všechny 4 části aplikace. Na konci formuláře zobrazíme tlačítko Filtrate, kterým filtr odešleme. Po zpracování parametrů filtru a přesměrování na stránku List se zobrazí výpis záznamů, na který již bude nastavený filtr aplikován. Pro zrušení nastaveného filtru zobrazíme tlačítko Clear. Po jeho použití se všechna formulářová pole vyprázdní a ve výpisu se zobrazí všechny záznamy.

Šablona pro **nastavení sloupců** bude výrazně jednodušší. Stejně jako o filtru bude potřeba vypsat sadu skrytých formulářových polí s nastavením ostatních nástrojů a parametry

subpage. Celý formulář bude obsahovat zaškrtávací pole pro výběr jednotlivých sloupců. Vypsání zaškrtávacích polí zajistí funkce html_checkboxes2, což je mírně upravená Smarty funkce html_checkboxes. Rozdíl je pouze v tom, že funkce html_checkboxes2 pracuje s asociativním polem, které je šabloně předáno a také podporuje atribut disabled pro znemožnění ovládání zaškrtávacího pole.

Obr. 5.2: Nastavení sloupců

Funkci předáme pole $checkboxes, což je asociativní pole s popisky sloupců. Klíče k hodnotám jsou jména sloupců z tabulek databáze. V parametru selected předáme pole sloupců, které mají být zaškrtnuté. Parametr disabled určuje sloupce, které nebude možné měnit (sloupec Name) a separator je oddělovací řetězec zaškrtávacích polí. Po vypsání všech sloupců zobrazíme tlačítko Select, které formulář odešle.

5.2 Zpracování parametrů pro výpis záznamů

V úvodní části všech skriptů, které se nějakým způsobem zabývají výpisem záznamů (např. list.php, columns.php, filter.php a další), musíme provést načtení parametrů všech čtyř nástrojů pro ovlivnění výpisu. Tyto parametry pak bude potřeba zkontrolovat a uložit do příslušných polí, nazvaných podle nástrojů. Celý proces bude složitější operace, proto si napíšeme funkci, která ji provede. Funkci nazveme kernel_get_url.

Pole, kam funkce bude ukládat nastavení filtru, funkci předáme odkazem, takže skript, který volá tuto funkci, bude mít po jejím dokončení proměnné k dispozici již bez případných chyb převzatých z url. Pokud v url nebudou hodnoty některého z nástrojů, pokusí se funkce

tyto hodnoty načíst z cookies, kde mohou být uložené. Pokud ani v cookies nebudou, vytvoří funkce novou url s výchozím nastavením nástrojů, na kterou přesměruje.

Některé nástroje budou vždy vyžadovat některé parametry v url, jinak by nemohly správně fungovat. Příkladem je stránkování, kde bude vždy potřeba znát číslo stránky a počet záznamů zobrazovaných na stránce. Jestliže tyto údaje nebudou známy, automaticky nastavíme stránku č. 1 a počet příspěvků na stránku nastavíme třeba na 10.

Všechny nástroje použijeme pro výpis záznamů z hlavních tabulek jednotlivých částí aplikace. Některé nástroje však použijeme i pro výpis historie záznamu, kategorií a pro export. Pro každý nástroj zvlášť tak musíme kontrolovat, na které stránce se nacházíme, protože by jinak mohlo docházet ke zbytečnému přesměrovávání. Url musíme zpracovávat i na stránkách Filter a Columns, aby bylo možné přecházet mezi těmito a výše uvedenými stránkami bez přesměrování, což budeme mít význam, když např. uživatelův prohlížeč nebude umožňovat ukládání cookies.

Nastavení **filtru** budeme zpracovávat pouze na stránkách List, Filter, Columns a Export. Provedeme tedy příslušná omezení podmínkou if. Podle filtrované tabulky musíme rozlišit, které sloupce se budou filtrovat. Pole $filter_arrays bude obsahovat názvy parametrů filtru pro filtrování podle vázaných záznamů z jiných tabulek. Proměnné těchto parametrů budou vždy typu array a budou obsahovat id hodnoty vázaných záznamů. V poli $filter_names budou názvy všech ostatních parametrů filtru.

Zkontrolujeme, zda bylo kliknuto na tlačítko Clear, které ruší nastavení filtru. K tomu by mohlo dojít na stránce Filter a také na stránce List, kam toto tlačítko pro rychlé zrušení filtru také umístíme. Nebylo-li na toto tlačítko kliknuto, zpracujeme parametry filtru z url.

V cyklu foreach projdeme všechny prvky pole $filter_names. V těle cyklu testujeme, zda existuje parametr filtru s názvem obsaženým v poli $filter_names. Pokud existuje, převedeme hodnotu z url do pole $filter. Současně nastavíme proměnnou $cookie_filter na false. Je-li tato proměnná false, signalizuje, že alespoň jeden parametr filtru se nachází v url a není tak potřeba zjišťovat nastavení filtru z cookies. Před cyklem jsme tuto proměnnou nastavili na true, takže pokud není žádný z parametrů filtru v url, hodnota proměnné zůstane nezměněna a později dojde ke čtení z cookies.

V dalším cyklu foreach projdeme prvky pole $filter_arrays, které obsahuje možné názvy vazebních parametrů filtru. Zkontrolujeme, zda parametr v url existuje a zda se jedná o pole hodnot. Z pole odstraníme duplicitní prvky a ve vnořeném cyklu foreach pole projdeme. V tomto cyklu již procházíme jednotlivé id hodnoty vázaných záznamů, na které má být výpis omezen. Pokud je id hodnota platná, uložíme ji do pole v poli $filter. Název pole je dán prv-

kem pole $filter_arrays, které bylo procházené v nadřazeném cyklu foreach. Proměnnou $cookie_filter opět nastavíme na false.

Pokud na tlačítko Clear bylo kliknuto a zároveň bylo v cookies uloženo nastavení filtru, cookies filtru zrušíme. Nastavení filtru se do cookies ukládá se stejnou strukturou, jako má pole $filter, avšak název těchto cookies je dán také jménem tabulky, kterou filtrujeme. Pro každou tabulku totiž budeme ukládat vlastní filtr.

Pole uložené v cookies nelze zrušit celé najednou, jak je možné u běžných polí. Jednotlivé prvky pole jsou samostatná cookies, které musíme zrušit samostatně. Proto v cyklu foreach projdeme jednotlivé proměnné filtru uloženého v cookies a danou cookie vymažeme. Je-li však tato proměnná parametr pro filtrování vazebních záznamů, jedná se o další pole id hodnot, jehož prvky jsou jednotlivá cookies a musíme je zrušit opět samostatně v dalším cyklu foreach.

Jestliže se nastavení filtru v url nenacházelo, bude mít proměnná $cookie_filter hodnotu true. Podíváme se tedy do cookies, zda není nastavení filtru uloženo tam. Pro průchod polem $_COOKIE použijeme stejnou strukturu cyklů foreach. Tentokrát však nebudeme parametry filtru ukládat do pole $filter, ale vytvoříme proměnnou $url_filter, což bude řetězec s url parametry filtru. Později se tento řetězec stane součástí celkové adresy url, na kterou se přesměruje. V url prohlížené stránky tak vždy budou všechny parametry všech nástrojů, aby je bylo možné např. uložit nebo se k nim později vrátit.

Oproti filtru bude možné nastavovat **sloupce** také ve výpisu historie záznamu, proto bude podmínka if doplněna navíc o parametr subpage2 této stránky. Opět si vytvoříme pole obsahující možné názvy sloupců pro danou tabulku. Nazveme jej $columns_array. V cyklu foreach budeme hledat nastavení sloupců v url. Podle toho, zda je v url přítomen sloupec Name, který by tam měl být vždy, budeme nastavení sloupců převádět buď do pole $columns, nebo do proměnné $url_columns ve formě url parametrů. Po dokončení cyklu přidáme do proměnné $url_columns sloupec Name, není-li tam.

Bude-li proměnná $cookie_columns nastavena na true, znamená to, že v url nebyl žádný sloupec. Stejným způsobem tedy projdeme pole $_COOKIE["columns"] a hledáme sloupec tam. Nastavení sloupců budou sdílet všechny části aplikace, proto nebudeme pro přístup do pole $_COOKIE používat název hlavní tabulky, jak tomu bylo u filtru. Případné nalezené sloupce ukládáme do proměnné $url_columns, kde budou url parametry sloupců pro přesměrování. Pokud ani v cookies sloupec nebudou, nastavíme výchozí sloupce Name a Actions do parametrů url.

Pro nastavení **stránkování** postačí přenášet přes url nebo cookies pouze dva parametry: $page pro nastavení čísla stránky a $records pro nastavení počtu příspěvků na stránce. Ostatní parametry později vypočítáme z těchto dvou. Podíváme se postupně do url a pak do cookies, zda-li tam jsou tyto dva parametry. Pokud je nalezneme, uložíme je do pole $pages. Pokud v url tyto parametry nebudou, vytvoříme řetězec $url_pages, kam tyto parametry pro url uložíme. Vypočítáme parametr $from, kam uložíme pořadí záznamu, od kterého se mají záznamy vypisovat. Do cookies budeme přistupovat přes jméno hlavní tabulky a řetězec "_pages", abychom mohli ukládat stránkování zvlášť pro každou část aplikace.

Řazení umožníme nastavovat i pro výpis historie a kategorií. U kategorií se bude hlavní tabulka jmenovat requirement_category nebo product_category. Podle hlavních tabulek nastavíme sloupce, podle kterých bude možné řadit. Tyto sloupce uložíme do pole $sort_array. Parametry řazení uložíme do pole $sort, jehož prvky budou názvy sloupců, podle kterých se má řadit. Pro sestupné řazení bude název sloupce doplněn řetězcem " desc". Klíče prvků pole budou čísla, která určí prioritu řazení. Bude-li mít pole $sort např. dva prvky o hodnotách $sort[1]="name" a $sort[2]="created desc", znamená to, že se záznamy seřadí nejdříve vzestupně podle jejich jména. Vyskytnou-li se některé záznamy se stejným jménem, seřadíme je sestupně podle času vytvoření.

V cyklu foreach projdeme parametry řazení obsažené v url a převedeme je do pole $sort. Pokud parametry v url nejsou, budeme je hledat v cookies. V cookies budou parametry řazení společné pro všechny hlavní tabulky, avšak pro výpis historie záznamu a pro výpis kategorií budeme řazení ukládat zvlášť. Proto bude ke jménu cookie přidána proměnná $tmp, ve které bude nastaven název stránky. V případě řazení záznamů z hlavních tabulek bude tato proměnná prázdná. Parametry z cookies převedeme do url parametrů řazení v proměnné $url_sort. Pokud ani v cookies parametry nebyly, do url nastavíme výchozí parametry řazení, tj. sestupně podle data vytvoření záznamu.

Nyní máme zpracovány všechny nástroje pro výpis záznamů. Pokud byly parametry příslušného nástroje v url, byly převedeny do odpovídajícího pole. Pokud parametry v url nebyly, byla vytvořena proměnná s odpovídajícími parametry nástroje pro přesměrování na novou url. Tato proměnná vznikla buď čtením parametrů z cookies, nebo nastavením výchozích hodnot parametrů, pokud v cookies parametry nebyly.

Jestliže alespoň jedna z těchto proměnných existuje, znamená to, že příslušný nástroj nemá své parametry v url, takže bude potřeba přesměrovat na url novou, jejíž součástí budou parametry v uložené proměnné. Zavedeme proměnnou $url, kam uložíme kompletní adresu.

Nejprve do ní uložíme parametry subpage1 a subpage2 a id hodnoty primárních sloupců tabulek, pokud existují.

Poté připojíme samotné url parametry jednotlivých nástrojů a přesměrujeme na novou adresu. Pokud proměnná pro url parametry daného nástroje nebyla vytvořena, měl nástroj své parametry v url. V takovém případě bylo vytvořeno pole, kam tyto parametry byly převedeny. Aby mohla být url adresa kompletní, musíme tato pole také převést do url. Pokud byly parametry všech nástrojů v url, nebyla vytvořena žádná proměnná s url parametry. K přesměrování nedojde a skript bude pokračovat po dokončení funkce s naplněnými poli parametrů nástrojů.

5.3 Sestavení url pro výpis záznamů

Často bude potřeba generoval url s parametry jednotlivých nástrojů, které máme uložené v proměnných nástrojů. Tuto url použijeme buď pro přesměrování, když původní url parametry neobsahuje, nebo pro nastavení odkazů v některých položkách menu, aby nebylo nutné při přechodu na jinou stránku neustále přesměrovávat.

Pro generování url si vytvoříme sadu čtyř funkcí, pro každý nástroj jednu. Argumentem každé funkce bude pole parametrů nástroje, jehož url generuje. Tyto funkce krom výše uvedeného budou také ukládat parametry nástrojů do cookies, odkud mohou být v případě neexistence parametrů v url přečteny.

Důvod, proč použijeme pro každý nástroj zvláštní funkci je ten, že občas bude v url chybět pouze jeden z nástrojů, takže příslušné funkce použijeme pouze u těch nástrojů, jejichž parametry budou v url a budou v pořádku. Druhý důvod je použití jen některých nástrojů na některých stránkách. Skripty těchto stránek pak budou volat pouze ty funkce, které přísluší k využívaným nástrojům. Někdy bude potřeba uložit několik různých url pro různé odkazy, takže bude záležet i na pořadí, v jakém budou funkce volány. Jistě budeme používat jiné odkazy pro nastavení řazení a jiné odkazy pro stránkování, než budou odkazy ve zbytku aplikace.

Funkci pro generování url parametrů **filtru** nazveme kernel_create_filter_url. Mohli bychom ji rozdělit na dvě části: v první zpracujeme všechny jednoduché parametry a ve druhé zpracujeme parametry nastavující filtrování podle záznamů z vazebních tabulek. Do pole $filter_names uložíme názvy sloupců, podle kterých půjde filtrovat. V první části to tedy budou textové, časové a hodnoty typu bool.

Ještě před tím, než parametry filtru uložíme do cookies, musíme původní parametry z cookies vymazat. To se provede stejně jako mazání cookies po kliknutí na tlačítko Clear,

což bylo popsáno v předchozí kapitole. Po vymazání projdeme pole s názvy možných parametrů filtru a budeme zjišťovat, zda jsou tyto parametry obsaženy v poli $filter. Nalezený parametr pak přidáme do proměnné $url, která bude tvořit část url s parametry filtru. Současně uložíme nalezený parametr také do cookies.

V druhé části funkce použijeme dva cykly foreach. Jeden pro průchod názvů parametrů filtru v poli $filter_names a druhý, vnořený, pro průchod přes jednotlivé id hodnoty parametru filtru. Každou id hodnotu zvlášť přidáme do proměnné $url a do cookies.

Generování url parametrů **sloupců** zajistí funkce kernel_create_columns_url. V poli $columns_names si opět nastavíme názvy sloupců. Tyto názvy projdeme v cyklu foreach a při existenci sloupce v poli $columns přidáme parametr sloupce do url a do cookies. Není-li daný sloupec v poli $columns, danou cookie vymažeme.

Nejjednodušší je nastavování url a cookies u **stránkování** (funkce kernel_create_pages_url), které má dva pevně dané parametry. Oba parametry uložíme do cookies a url parametry vypíšeme přímo v příkaze return. U výpisu historie nebudeme do cookies ukládat číslo stránky, které bude lepší vždy nastavit na první stránku.

Ve funkci kernel_create_sort_url, která nastavuje url a cookies pro **řazení**, musíme rozlišovat řazení hlavních tabulek, řazení výpisu historie záznamu a kategorií. Podle toho pojmenujeme cookies. V cyklu foreach projdeme jednotlivá cookies a všechna je vymažeme. V druhém cyklu foreach projdeme pole $sort a uložíme aktuální parametry řazení do cookies. Uložíme přitom pouze prvních pět parametrů řazení. Další již nemá smysl ukládat, protože by měly tak malou váhu, že na výsledném výpisu by se stejně neprojevily.

5.4 Skript pro výpis záznamů

Jak už bylo uvedeno, záznamy bude vypisovat stránka List pomocí skriptu list.php. Skript nejprve zavolá funkci kernel_get_url, které předá prázdná pole pro nastavení parametrů nástrojů. Tato funkce přečte parametry z url a pole jimi naplní, případně přesměruje na novou url, pokud původní url parametry neobsahuje.

Zavedeme proměnnou $webdoc["url"], která bude obsahovat aktuální url parametry nástrojů. Využití bude mít především pro odkazy na stránky Columns, Filter, Export a pro případné přesměrování. Nejprve proměnnou naplníme parametry filtru a sloupců. Vytvoříme pomocné proměnné $url_pages a $url_sort a naplníme je parametry stránek a řazení. V poli $pages zavedeme novou hodnotu s klíčem "url", kam uložíme parametry všech nástrojů kromě stránkování. Význam to má pro odkazy ve stránkování, které se ve svých parametrech budou lišit. Parametry ostatních nástrojů však budou mít stejné. Totéž uděláme s proměnnou

$sort2["url"] pro odkazy na řazení. Teprve po těchto přiřazeních sloučíme parametry všech nástrojů do proměnné $webdoc["url"].

Již na tomto místě skriptu přiřadíme pole $filter do Smarty. Důvod je ten, že v poli $filter vytvoříme ještě několik proměnných, které budou sloužit jako výstup **kontroly časových údajů** a bylo by dokonce nechtěné, aby tyto proměnné byly také přiřazeny do Smarty. Došlo by tak k výpisu těchto proměnných do skrytých formulářových polí a k následnému odeslání s ostatními parametry filtru. Zbytečně, protože by kontrolními funkcemi nebyly vůbec přečteny.

Tyto proměnné však budou mít význam při pozdějším čtení dat záznamů z databáze. Pokud uživatel zadá některý časový údaj filtru i mírně odlišně, než je formát sloupce DATETIME v databázi, došlo by k chybě nebo k výpisu nechtěných dat. Vytvoříme funkci, která se tato data pokusí upravit do formátu vhodného pro databázi.

Nazveme ji kernel_date_control. V parametru $date bude čas zadaný uživatelem. Výstupem funkce bude stejný nebo poupravený řetězec, který však bude snadno převoditelný do formátu vhodném pro databázi. Funkce může vrátit i hodnotu false, nepodaří-li se čas rozpoznat. Celé rozpoznávání času bude založeno na funkci date_parse, která je součástí php. Funkce dokáže rozpoznat čas v různých formátech, avšak nedělá to příliš dokonale. Proto před funkcí provedeme předzpracování zadaného data tak, aby funkce date_parse čas úspěšně rozpoznala.

Zadaný čas si rozdělíme pomocí funkce split do pole $date_split podle různých znaků, u kterých se předpokládá, že budou oddělovat jednotlivé časové jednotky. Nejprve ošetříme stav, kdy uživatel zadá pouze jedno číslo. Je-li jeho hodnota čtyřciferná, jedná se o rok. Funkce date_parse však v tomto čísle chybně rozpozná minuty a vteřiny. Proto k proměnné $date připojíme nulu oddělenou pomlčkou. Je-li číslo v rozsahu 1 až 31, jedná se o den v měsíci, ale funkce date_parse by vůbec nerozpoznala časový údaj. Do proměnné $date proto doplníme aktuální měsíc a rok. Ošetříme také stav, kdy uživatel zadá datum a hodinu. Funkce správně rozpozná datum, ale hodinu už ignoruje. K řetězci tedy doplníme nultou minutu.

Teprve po některé z těchto úprav proměnné $date zavoláme funkci date_parse. Ta převede časový údaj do asociativního pole, jehož prvky budou obsahovat detailní informace o čase. Jeden z těchto prvků je pole, ve kterém jsou soustředěné chybové zprávy z převodu. Nedošlo-li k žádné chybě a zároveň je nastaven údaj o hodině v tomto poli, můžeme upravený řetězec považovat za rozpoznatelný časový údaj a vrátíme jej nadřazenému skriptu. V opačném případě naše funkce vrátí false.

Touto funkcí tedy zkontrolujeme všechny časové údaje filtru. Protože je funkce může vrátit pozměněné, uložíme si je do nových proměnných, jejichž názvy budou stejné s původními proměnnými, ale doplněné o dvojku. Původní proměnné ponecháme záměrně nezměněné, aby se uživateli zobrazovalo stále stejné nastavení filtru přesně tak, jak jej zadal.

Vrátí-li funkce pro některý časový údaj false, přesměrujeme na stránku Filter, kde bude vypsána chybová zpráva o špatně zadaném časovém údaji filtru. K přesměrování použijeme proměnnou $webdoc["url"], kam jsme na začátku skriptu uložili všechny parametry nástrojů, takže i původní parametry filtru budou zobrazeny ve formuláři stránky Filter. Proměnnou $webdoc["url"] pouze doplníme o parametry subpage1 podle hlavní tabulky a subpage2 nastavíme na "filter".

Proběhne-li kontrola časových údajů bez chyby, můžeme načíst záznamy z databáze. Vytvoříme k tomu funkci database_get_list, které předáme všechna pole nástrojů. Funkci se budeme věnovat v následující kapitole. Funkce nám kromě záznamů nastaví i proměnnou $rows s počtem vrácených záznamů a nový parametr stránkování $pages["records_total"] s celkovým počtem záznamů v tabulce.

Tuto proměnnou budeme potřebovat pro nastavení **odkazů pro stránkování**. Pro pohodlné procházení mezi stránkami musíme zobrazit odkazy několika předchozích a následujících stránek a první stránku. Zobrazíme také sadu odkazů pro nastavení počtu příspěvků na stránce. Pro všechny tyto odkazy bude potřeba vypočítat příslušná url, k čemuž si napíšeme funkci kernel_create_pages, které předáme pole se stránkováním včetně celkového počtu záznamů.

Tato funkce zavede do pole $pages další parametry, z nichž první bude celkový počet stránek. Určíme tři stránky před a tři stránky za aktuální stránkou pro zobrazení seznamu odkazů. Zkontrolujeme, zda krajní hodnoty nepřesáhly počet stránek nebo nešly do mínusu. Do pole $pages["list_pages"] naplníme čísla stránek a chybí-li číslo aktuální stránky, číslo první stránky nebo číslo poslední stránky, přidáme je.

Průchodem pole $pages["list_pages"] v cyklu foreach naplníme další pole $pages["list_pages_url"] se stejnými klíči, které bude obsahovat adresy odkazů na jednotlivé stránky. Zde využijeme proměnnou $pages["url"], kterou jsme naplnili na začátku skriptu list.php url parametry všech nástrojů kromě stránek. K němu doplníme parametr čísla stránky, počtu příspěvků na stránku a parametry subpage1 a subpage2 (pokud existuje). Vytvoříme také parametry $pages["url_page_prev"] a $pages["url_page_next"], které budou obsahovat url pro nastavení předchozí a následující stránky. Tyto odkazy budou zobrazeny šipkami tam a zpět.

Podobně jako pole $pages["list_pages"] vytvoříme i pole $pages["list_records"] pro počty příspěvků na stránku. Tomuto poli nastavíme pevné hodnoty. Přidáme pouze aktuální počet příspěvků na stránku, pokud v tomto poli není uveden, a pole seřadíme. V cyklu foreach k tomuto poli vytvoříme pole $pages["list_records"], kam uložíme odkazy pro změnu počtu příspěvků na stránku.

Pro sestavení **odkazů pro řazení** také vytvoříme funkci: kernel_create_sort. Funkci předáme pole parametrů řazení $sort a také pole předané odkazem, $sort2. Toto pole bude naplněno odkazy pro změnu řazení. Vytvoříme v něm dva prvky: oba budou další asociativní pole s klíči podle názvů sloupců tabulek. Pole $sort2["asc"] bude obsahovat odkazy na změnu řazení vzestupně a pole $sort2["desc"] bude obsahovat odkazy na změnu řazení sestupně.

Třetím parametrem funkce je pole sloupců $columns, které se zobrazí ve výpisu a pro které bude potřeba vygenerovat odkazy na změnu řazení. Pokud se bude jednat o výpis kategorií, nastavíme tomuto poli sloupce pevně, protože na stránce kategorií nebude možné měnit vypisované sloupce.

V cyklu foreach projdeme pole sloupců a pro každý sloupec budeme nastavovat vzestupné i sestupné adresy pro změnu řazení. Vynecháme pouze sloupec Actions, jenž má význam pouze pro zobrazení odkazů na operace se záznamy. Každá adresa bude složena z parametru subpage1, z proměnné $sort2["url"], která obsahuje parametry ostatních nástrojů a z prvního prvku pole $sort, jehož hodnota bude název procházeného sloupce v cyklu foreach. V druhém cyklu foreach projdeme pole $sort a do adres nastavíme původní parametry řazení (pouze první 4). Funkce tedy zopakuje parametry řazení v odkazech, ale před ně vždy vloží parametr daného sloupce.

Vrátíme se do skriptu list.php. Řazení a stránkování máme připravené, takže si můžeme připravit sadu **proměnných pro zobrazení seznamu** Smarty funkcí view_result. Připravíme si pole $columns_array, které naplníme názvy všech možných sloupců vypisované tabulky. V cyklu foreach projdeme pole $columns, které obsahuje názvy sloupců, které se mají zobrazit. V těle cyklu naplníme pole $title názvy sloupců, které se zobrazí uživateli (s velkými písmeny a bez podtržítek) a pole $values názvy sloupců dle tabulky z databáze. K přidání nových prvků do těchto polí dojde, pouze když sloupec z pole $columns je obsažen v poli $columns_array. Budeme tak mít jistotu, že se budou zobrazovat sloupce, které skutečně existují v databázi.

Pole $title a $values přiřadíme do šablony. Přiřadíme do ní i pole $actions, $role a $parametr. Všechna tři pole mají význam pro zobrazení odkazů ve sloupci Actions. První pole definuje názvy a popisky akcí. Druhé pole určuje práva uživatelů, které mohou tyto akce pro-

vádět. Podle toho se pak jednotlivé odkazy zobrazí, nebo skryjí. Třetí pole udává seznam parametrů z url, které se mají převést do adres odkazů ve sloupci Actions. Další proměnné, které přiřadíme do Smarty, budou pole $pages, $sort, $sort2 a $columns s parametry jednotlivých nástrojů (pole $filter jsme přiřadili dříve). Dále přiřadíme délku pole $title (tj. počet zobrazených sloupců), proměnnou $webdoc["url"] s url parametry nástrojů a také pole $list se samotnými záznamy.

Přímo na stránce List budeme zobrazovat **textový popis filtru**, aby uživatel nemusel přecházet na stránku Filter jen pro zjištění jeho nastavení. K tomu, abychom mohli zobrazit nastavení filtru, potřebuje v databázi zjistit jména záznamů ve vazebních tabulkách, jelikož známe jen jejich id hodnoty.

Vytvoříme funkci database_get_filter, které předáme pole $filter. Ve funkci zavedeme pole $filter_text, do kterého budeme vkládat textové popisy jednotlivých parametrů filtru, které se budou skládat z názvu filtrovaného sloupce a ze zadané hodnoty (nebo skupiny hodnot) na stránce Filter. Funkce bude volána ve všech částech aplikace, takže podle toho musíme rozlišovat, které parametry filtru můžeme použít. Je-li některý parametr filtru použitelný pouze v některých částech aplikace, omezíme přiřazení do pole $filter_text v příslušné podmínce if.

Nejprve naplníme textové parametry filtru. To je jednoduché, protože jsou obsaženy přímo v poli $filter, proto je jen převedeme do pole $filter_text a přidáme název sloupce. Podobně to uděláme s časovými parametry. Zde však musíme správně kombinovat začátek a konec časového intervalu a podle toho přidat řetězec "from", nebo "to", nebo oba.

U polí id hodnot musíme provést dotaz do databáze pro zjištění názvů záznamů, podle kterých se filtruje. Pro tyto dotazy vytvoříme funkci get_array, jejíž vrácený textový popis filtru připojíme do pole $filter_text. Funkci předáme pole $filter a klíč k id hodnotám v tomto poli, z nichž má zjistit textový popis. V cyklu foreach projdeme jednotlivé id hodnoty a postupně sestavíme část dotazu za klauzulí WHERE (proměnná $where), kterou omezíme výpis na záznamy, podle nichž se filtruje.

Hodnotu -1 nebudeme do dotazu přidávat. Jedná se o hodnotu, která říká, že uživatel chce zobrazit záznamy, které nemají vazbu do daných tabulek. Např. nejsou v žádné kategorii, nemají žádného rodiče, žádnou závislost, apod. V příkaze switch rozlišíme, které id hodnoty se vypisují a naplníme proměnnou $text_root příslušným popisem parametru filtru bez vazeb.

Běžné hodnoty, které jsou větší než -1, přidáme do dotazu. Zde musíme správně vybrat sloupec, ze kterého id hodnoty pocházejí. U některých polí id hodnot to jsou stejné sloupce. Např. pro filtrování podle závislostí a podle rodičů je to sloupec sequence_id. Pro

filtrování podle uživatelů, kteří záznam vytvořili, schválili, testovali nebo zrušili, je to vždy sloupec person_id v tabulce person.

Po sestavení proměnné $where můžeme sestavit celý dotaz. Ten bude podle různých id hodnot směrován do různých tabulek, což rozlišíme v příkaze switch. U kategorií budeme vypisovat tabulku requirement_category nebo product_category, u rodičů a závislostí to bude dotaz do tabulky requirement (product) a u uživatelů to bude tabulka person. Vrácené záznamy projdeme v cyklu while, kde naplníme proměnnou $text jmény záznamů. Na začátek připojíme proměnnou $text_root, tento řetěz vrátíme a připojíme do prvku pole $filter_text v nadřazené funkci.

5.5 Načtení záznamů z databáze

Pro skript list.php je nejdůležitější funkce database_get_list, která načte záznamy z vypisované tabulky. Parametr $count, předaný odkazem, funkce naplní počtem vrácených záznamů a parametr $total, taktéž předaný odkazem (důležitý pro stránkování), funkce naplní počtem záznamů, které by funkce vrátila bez použití stránkování. V dalších parametrech $pages, $sort, $columns a $filter funkci předáme pole nástrojů, podle nichž funkce upraví výpis záznamů.

Velkou část funkce zabere zpracování parametrů filtru. Zavedeme pole $filters, do kterého budeme vkládat omezující podmínky za klauzulí WHERE výsledného dotazu. Pro každý parametr filtru budeme vždy kontrolovat, jestli je obsažen v poli $filter (tj. byl uživatelem zadán) a zda-li jde použít pro vypisovanou tabulku.

Textové parametry filtru, jenž nesou vyhledávaný řetězec, lze v MySQL zpracovat několika způsoby [13]. Na výběr máme několik možností:

- operátor LIKE omezuje výběr podle zadaného řetězce. Lze použít zástupné znaky pro libovolný počet znaků a pro libovolný jeden znak,
- fulltextové vyhledávání v přirozeném jazyce dovede vyhledávat podle několika zadaných slov. Slova, která se vyskytují ve více než padesáti procentech záznamů, ignoruje,
- fulltextové vyhledávání v boolean módu tento padesátiprocentní práh nemá. Dovoluje navíc použít operátory pro zpřesnění výběru. Např. lze některým slovům dát větší váhu než jiným nebo lze vypisovat záznamy, které dané slovo neobsahují.

Zadá-li uživatel často se vyskytující řetězec, vždy musí být vypsány záznamy, které tento řetězec obsahují, i když se vyskytuje ve většině záznamů. Proto je pro nás fulltextové

vyhledávání v přirozeném jazyce nepoužitelné. Použijeme boolean vyhledávání, které oproti ostatním variantám nabízí více možností, jak upřesnit vrácené záznamy.

Ve výchozím nastavení MySQL obě varianty fulltextového vyhledávání ignorují slova, která jsou kratší než 4 znaky a také některá běžná anglická slova. Tyto vlastnosti však lze zrušit nastavením konfiguračních proměnných ft_stopword_file na 1 a ft_min_word_len na prázdný řetězec. Teprve po tomto nastavení se filtr bude chovat zcela správně.

Část dotazu pro filtrování sloupce Name sestavíme takto:

```
$filters[]="MATCH(m.name) AGAINST ('$filter[name]' IN BOOLEAN MODE)";
```

Ve funkci MATCH uvedeme filtrovaný sloupec a do funkce AGAINST umístíme hledaný řetězec z proměnné $filter[name]. Přidáme řetězec "IN BOOLEAN MODE", který nastaví boolean mód hledání.

Pro omezení výpisu podle časových hodnot lze jednoduše použít operátory větší nebo rovno a menší nebo rovno. Do dotazu však musíme vložit časový údaj v sql formátu. V nadřazeném skriptu jsme si již pomocí funkce kernel_date_control upravili časy tak, že v nových proměnných pole $filter, jejichž klíč je zakončen číslem 2, máme časy, které spolehlivě rozpozná funkce date_parse.

Vytvoříme si malou funkci sql_date, které předáme poupravený časový údaj. Funkcí date_parse jej převedeme do asociativního pole $date_parse, do jehož jednotlivých prvků vloží detailní údaje o převedeném čase (rozděleném na hodiny, minuty apod.). Z tohoto pole pak vytvoříme řetězec s časem v sql formátu, který funkce vrátí.

Pro zpracování vazebních id hodnot filtru budeme potřebovat znát názvy sloupců včetně tabulky, resp. její zkratky zadané v části SELECT, do které sloupec patří. Všechny možné značky a sloupce vložíme do pole $table_columns, které se bude pro každou tabulku lišit. Do pole $names zadáme možné klíče k parametrům id hodnot filtru. Jednotlivé prvky obou polí si musí odpovídat, abychom mohli nastavit správné id hodnoty pro správnou tabulku.

V cyklu foreach projdeme pole klíčů k parametrům id hodnot a budeme testovat, zda tyto parametry v poli $filter existují. V dalším cyklu foreach projdeme jednotlivé id hodnoty daného parametru a budeme je přidávat do části dotazu v proměnné $values. Jednotlivá omezení budeme oddělovat operátorem OR, aby se vypsaly všechny záznamy, které obsahují jednu z těchto id hodnot. Pokud je id hodnota rovna -1, nebudeme ji zadávat do dotazu, ale místo ní zadáme operátor IS NULL. Výsledný řetězec v proměnné $values přidáme do pole $filters jako sadu id hodnot jednoho parametru filtru, podle kterého omezujeme výpis. Pro-

měnnou $values následně zrušíme, aby její obsah nevložil do dalších parametrů filtru při dalších průchodech cyklem foreach.

Část dotazu pro parametry filtru, u kterých bylo možné nastavit pouze hodnoty pravda/nepravda, vznikne zadáním názvu sloupce a připojením jedné ze dvou pevných hodnot, které se ve sloupci můžou vyskytovat. Daný parametr filtr může obsahovat hodnotu 2 pro pravdu, nebo hodnotu 1 pro nepravdu.

Jsou přitom sloupce, jejichž hodnoty mohou být typu NULL a sloupce, které být typu NULL nemohou. Výpis u sloupců, které mohou být typu NULL, budeme omezovat hodnotami IS NULL a IS NOT NULL. Platí to pro sloupec is_product_specific v tabulce requirement. U sloupců, které nemohou být typu NULL, což je např. sloupec active v tabulce requirement, budeme výpis omezovat hodnotami 0 a 1.

Po naplnění pole $filters toto pole projdeme cyklem foreach a do proměnné $filter_all zadáme jednotlivé omezující podmínky. Oddělíme je operátorem AND, aby se vypsaly jen ty záznamy, které všechny podmínky splňují.

Zavedeme proměnnou $query_part, do které budeme ukládat část dotazu za klauzulí FROM. Za touto klauzulí budeme uvádět jednotlivé tabulky (včetně vazebních tabulek), jejich zkratky a propojovací sloupce. Vliv na to, které tabulky budou připojeny, bude mít hlavně nastavení sloupců, které chce uživatel zobrazit. Vliv bude mít také nastavení filtru a řazení. Pokud chceme filtrovat nebo řadit podle hodnot z vazební tabulky, musíme tuto tabulku připojit. Můžeme i filtrovat a řadit podle sloupce, který nebude ve výpisu zobrazen. I v tomto případě musíme tabulku připojit. Rozhodnutí o tom, zda se daná tabulka připojí, nebo nepřipojí, bude provedeno v podmínkách if.

Po připojení tabulek zadáme do proměnné $query_part také část dotazu za klauzulí WHERE. Všechny omezující podmínky budeme oddělovat operátorem AND, bereme-li id hodnoty jednoho parametru filtru, které jsme oddělili operátorem OR, jako jednu omezující podmínku. Jelikož dopředu nevíme, která z omezujících podmínek bude do dotazu vložena, zavedeme jako první omezující podmínku hodnotu 1, která bude pro všechny záznamy vyhodnocena kladně. Potom budeme moci přidávat další omezující podmínky uvozené operátorem AND.

Pro výpis na stránce List musíme zavést omezující podmínku, protože chceme vypsat pouze aktuální záznamy. Použijeme k tomu poddotaz, v němž agregační funkce MAX vrátí maximální id hodnoty (např. requirement_id) seskupených záznamů podle sloupce sequence_id. Nadřazený dotaz omezí výpis záznamů na vrácené id hodnoty, takže získáme pouze aktuální záznamy. Stejná technika již byla ukázána v kapitole 4.5 pro načtení vazebních

záznamů. Vypisujeme-li historii záznamu, do podmínky uvedeme pouze sequence_id historické větve záznamu.

Na stránce List a Export do proměnné $query_part připojíme omezující podmínky filtru, které máme v proměnné $filter_all. Pokud však exportujeme historii záznamů, pole $filter bude funkci předáno prázdné, takže i proměnná $filter_all bude prázdná a k filtrování historie docházet nebude.

Za klauzulí SELECT definujeme všechny sloupce, které chceme vypsat. Uložíme je do proměnné $query_main, která bude obsahovat celkový dotaz pro výpis záznamů. Vždy budeme vypisovat jméno záznamu a id hodnotu záznamu (name, requirement_id, product_id). Kromě tabulky test_run budeme vypisovat také sequence_id záznamu. Pro výpis historie uvedeme sloupec version a zavedeme proměnnou current, která bude označovat aktuální záznam. Proměnnou nastavíme pomocí poddotazu, jenž byl vysvětlen také v kapitole 4.5.

Jméno kategorie a jméno rodiče, které jsou ve sloupcích nazvaných stejně jako je nazván sloupec se jménem vypisovaného záznamu, přiřadíme do nových proměnných category a parent. Jméno a příjmení uživatele, který záznam vytvořil, schválil, testoval nebo zrušil, sloučíme funkcí CONCAT do jedné proměnné. Budou to proměnné creator, approver, tester a terminator. Pomocí funkce IF zjistíme stav sloupců, ve kterých lze nastavit pouze hodnoty pravda, nebo nepravda. Výsledek funkce uložíme do nových proměnných, např. active a is_product_specific.

Po naplnění proměnné $query_main částí SELECT připojíme i část dotazu z klauzulemi FROM a WHERE, které jsme již měli v proměnné $query_part. Přidáme klauzuli ORDER BY, za kterou vypíšeme sloupce, podle nichž se má výpis seřadit. V cyklu foreach projdeme pole $sort a naplníme proměnnou $order názvy sloupců. Tuto proměnnou pak spolu s klauzulí připojíme k řetězci $query_main. Nakonec připojíme klauzuli LIMIT s nastaveným pořadím záznamu, od kterého se má začít vypisovat a počtem záznamů ve výpisu. Tyto informace máme uložené v poli $pages.

Proměnnou $query_part, ve které máme uloženu část dotazu s klauzulemi FROM a WHERE, využijeme i pro druhý dotaz na počet záznamů bez použití stránkování. Součástí dotazu bude nastavení filtru, ale vynecháme nastavení sloupců a řazení, zde by to nemělo smysl. Proměnnou $query_part vložíme do poddotazu za část SELECT, kde bude pouze hodnota 1. To způsobí, že poddotaz nevypíše žádné sloupce, pouze jeden sloupec s jedničkami.

Takto nám to stačí, protože nadřazený dotaz potřebuje spočítat pouze počet záznamů vracených poddotazem. Pro spočítání počtu záznamů použijeme agregační funkci COUNT, jejíž výsledek přiřadíme do proměnné total. Celý dotaz vrátí pouze jeden řádek s počtem zá-

znamů. Funkcí extract výsledek převedeme do proměnné $total, což je proměnná předaná funkci odkazem, takže nadřazený skript bude znát celkový počet záznamů a bude moci správně nastavit stránkování.

Na konci funkce provedeme hlavní dotaz uložený v proměnné $query_main. Funkce database_fetch_result nastaví proměnnou count předanou odkazem, takže nadřazený skript bude znát počet vrácených záznamů. Výpis záznamů vrátíme příkazem return.

Protože celý dotaz a jeho sestavení je poměrně složité, uvedeme příklad dotazu s nastaveným filtrem

```
SELECT m.name, m.requirement_id, m.sequence_id, c.name AS category,
m.created, CONCAT( pc.name, ' ', pc.surname ) AS creator
FROM requirement m
LEFT JOIN requirement_category c ON m.category_id = c.category_id
LEFT JOIN person pc ON m.creator_id = pc.person_id
WHERE 1
AND m.requirement_id IN (
  SELECT MAX( requirement_id )
  FROM requirement GROUP BY sequence_id
)
AND (
  MATCH (m.name) AGAINST ('te*' IN BOOLEAN MODE)
  AND m.created >= '2008-4-1 20:50:0'
  AND (pc.person_id = '1' OR pc.person_id = '3')
  AND (c.category_id IS NULL)
)
GROUP BY m.requirement_id
ORDER BY created DESC
LIMIT 10, 5
```

V části SELECT, která byla prvním obsahem v proměnné $query_main, vždy vložíme sloupce name, requirement_id a sequence_id. Dále jsme uvedli sloupce, které jsme chtěli zobrazit ve výpisu a podle kterých jsme filtrovali. Přidali jsme proto sloupce name z tabulky categories a protože sloupec name je i v tabulce requirement, dostal alias category. Sloupec created je přímo součástí tabulky requirement, takže k němu jsme alias přidávat nemuseli. Sloupec creator vznikl sloučením sloupců name a surname z tabulky person.

V části FROM jsme uvedli tabulku, ze které jsme chtěli vypisovat záznamy, v tomto příkladě requirement. Připojili jsme tabulky requirement_category a person, protože jsme některé jejich sloupce chtěli vypsat a filtrovat. K nim jsme přidali aliasy používané u názvu sloupců ve zbytku dotazu.

Do části WHERE jsme nejprve přidali poddotaz pro omezení výpisu na aktuální záznamy. Pak jsme přidali parametry filtru oddělené operátorem AND. První parametr provedl filtrování ve sloupci name. Byl to textový filtr, takže jsme použili boolean vyhledávání.

Druhý parametr omezil vypsané záznamy pouze od zadaného data. Ve třetím parametru jsme chtěli zobrazit pouze příspěvky od uživatelů s hodnotami person_id 1 a 3. Čtvrtý parametr nám omezil výpis na záznamy, které nebyly zařazeny do žádné kategorie. Části WHERE a FROM jsme ukládali do proměnné $query_part, která byla použita i pro dotaz na zjištění celkového počtu záznamů.

Za klauzulí GROUP BY jsme seskupili záznamy se stejným requirement_id. V části ORDER BY jsme uvedli sloupce, podle kterých jsme chtěli řadit, což určovalo pole $sort. v tomto případě jsme řadili podle sloupce created. Parametrem DESC jsme uvedli, že jsme chtěli řadit sestupně. Poslední klauzule byla LIMIT. Zde jsme zadali pořadí záznamu, od kterého jsme chtěli vypisovat (10) a počet záznamů na stránce (5), tj. údaje z pole $pages. Znamená to tedy, že jsme chtěli vypsat stránku č. 3.

Dotaz pro zjištění celkového počtu záznamů s použitým filtrem by byl téměř stejný. Předchozí dotaz by byl použit jako poddotaz tohoto dotazu. Část SELECT by byla nahrazena jedničkou a části LIMIT a ORDER BY by byly vypuštěny. Jak by mohla vypadat stránka, která zobrazuje záznamy vrácené tímto dotazem, ukazuje obr. 5.3.

5.6 Šablona pro výpis záznamů

V tomto okamžiku již skript list.php zpracoval všechny nástroje, získal výpis záznamů a do Smarty přiřadil všechny potřebné proměnné. Byla zavolána šablona list.tpl, na kterou se teď podíváme. Funkce samotné šablony je jednoduchá: veškeré údaje za ni vypíšou jiné šablony nebo funkce. Šablona vypíše jen úvodní tagy tabulky a prázdné řádky, mezi ně vloží další šablony a funkce, které vypíšou své řádky. Tato kapitola tedy bude více o popisu těchto šablon.

Nejprve potřebujeme vypsat **stránkování**. Protože ho budeme vypisovat na začátku i na konci stránky, bude vytvoření malé šablony pro tento účel výhodné. Vytvoříme ji ve složce templates a soubor pojmenujeme pages.tpl. Šablona vypíše dva řádky. První bude záhlaví stránkování s nadpisem Pages. Ve druhém řádku zobrazíme samotné ovládání stránkování. Součástí bude možnost zadávání čísla stránek a počtu příspěvků na stránku ve formulářových polích, proto vypíšeme jednoduchý formulář. Ten se bude odesílat metodou get, takže musíme zavolat další šablony pro výpis skrytých formulářových polí ostatních nástrojů. V horní části řádku zobrazíme odkazy pro nastavení čísla stránky. Následující komentovaná ukázka zobrazuje tento úsek kódu rozepsaný do více řádků:

```
Page number:
{if $pages.page>1}<a href="{$pages.url_page_prev}">{/if}
```

```
<<                                                  {*předchozí stránka*}
{if $pages.page>1}</a>{/if}
 |                                                  {*oddělovač*}
{foreach from=$pages.list_pages key=key item=page} {*průchod polem stran*}
  {if $page!=$pages.page}<a href="{$pages.list_pages_url.$key}">{/if}
  {$page}                                           {*číslo stránky*}
  {if $page!=$pages.page}</a>{/if}
  |                                                 {*oddělovač*}
{/foreach}
{if $pages.page<$pages.pages_total}<a href="{$pages.url_page_next}">{/if}
>>                                                  {*další stránka*}
{if $pages.page<$pages.pages_total}</a>{/if}
                            {*formulářové pole pro vložení čísla stránky*}
{insert_form type="text" size="4" name="page" value=$pages.page}
<br />
```

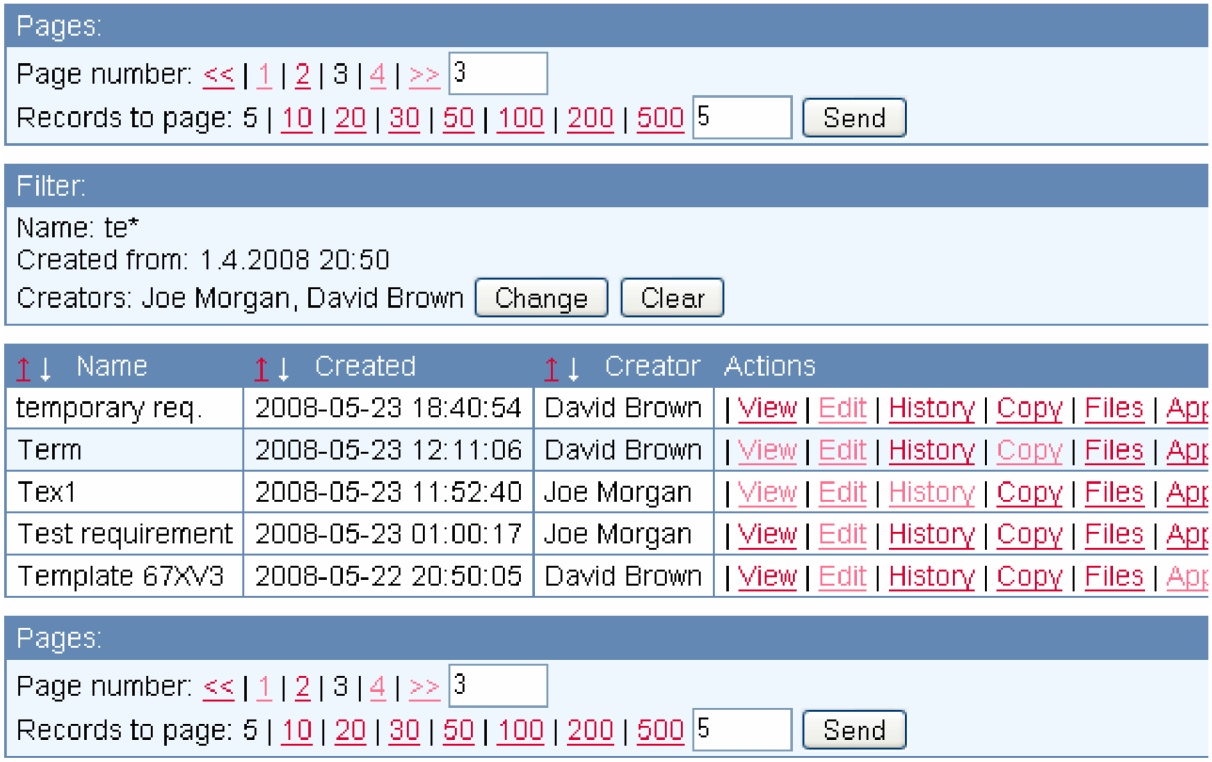
Nejdříve budeme odkazovat pomocí dvou šipek vlevo na předchozí stránku. Odkaz na předchozí stránku je v proměnné $pages.url_page_prev. Pokud se uživatel bude nacházet na první stránce, což poznáme podle proměnné $pages.page, zobrazíme pouze šipky bez odkazu.

Za odkazem na předchozí stránku zobrazíme řadu odkazů na další stránky. Čísla stránek jsou uložena v poli $pages.list_pages a jejich adresy jsou v poli $pages.list_pages_url. Klíče obou polí si odpovídají. Pole $pages.list_page projdeme v cyklu foreach a budeme zobrazovat odkazy z pole $pages.list_pages_url na stránky v poli $pages.list_page. Pouze pokud vypisovaná stránka je stránkou aktuálně zobrazenou, zobrazíme jen číslo stránky bez odkazu. Po dokončení cyklu foreach zobrazíme odkaz na následující stránku. Je-li aktuálně zobrazená stránka poslední, zobrazí se pouze dvě černé šipky vpravo bez odkazu. Za odkazy ještě vložíme formulářové pole, aby uživatel mohl vložit číslo stránky ručně.

V druhé části řádku zobrazíme odkazy na změnu počtu příspěvků na stránku. Provedeme to stejným způsobem. Poli $pages.list_records máme čísla s počty příspěvků na stránky a v poli $pages.list_records_url máme odkazy na nastavení počtu příspěvku na stránku o těchto číslech. V cyklu foreach odkazy zobrazíme, pouze aktuální počet příspěvků na stránku vypíšeme bez odkazu. Za odkazy vložíme formulářové pole pro možnost ručního nastavení počtu příspěvků na stránku. Zobrazíme také tlačítko Send, které obě formulářové pole po kliknutí odešle.

Druhá šablona, filter.tpl ve složce templates, bude zobrazovat textový popis nastaveného **filtru** a tlačítka pro přenastavení filtru a smazání filtru, je-li nějaký parametr filtru nastaven. Opět vytvoříme formulář, do kterého vypíšeme skrytá formulářová pole s nastavením ostatních nástrojů. Pak vypíšeme textové popisy jednotlivých parametrů filtru, které máme uložené v poli $filter_text. Pokud žádný parametr filtru nastaven není, je pole prázdné a pro-

vede se část foreachelse, ve které vypíšeme No. Za popisem posledního parametru filtru zobrazíme zmíněná tlačítka pro změnu a smazání filtru.



Obr. 5.3: Výpis záznamů

Máme tedy vypsáno stránkování a filtr, takže můžeme **vypsat záznamy**, je-li k dispozici alespoň jeden záznam. Výpis záznamů zajistí Smarty funkce view_result [1], které předáme pole záznamů $list, titulky sloupců, které má funkce zobrazit, v poli $title, názvy sloupců tak, jak jsou v tabulkách, v poli $values, asociativní pole $actions s definovanými odkazy na další operace se záznamy, pole $parametr, kde jsou definovány parametry, které mají být přidány do odkazů a pole $role, které určuje práva, které musí mít uživatelé, aby se jim odkazy ve sloupci Actions mohly zobrazit.

Po výpisu záznamů ještě jednou zobrazíme stránkování. Pokud není v poli $list žádný záznam, tak se výpis záznamů a stránkování na konci stránky nezobrazí. Obr. 5.3 ukazuje, jak by mohla vypadat stránka pro výpis záznamů, včetně nastaveného stránkování a filtru.

6. Kategorie

V částech Requirements a Products umožníme členit jednotlivé záznamy do kategorií. Nejprve však musíme umět kategorie vytvářet a teprve pak je budeme moci použít. S ohledem na firemní zvyklosti budou kategorie v části Requirements pojmenovány jako Category a v části Products jako Family, ale ve skutečnosti půjde o totéž. Ve zdrojovém kódu budeme proměnné obou částí nazývat $category, $categories, apod., pouze v nadpisech budeme uživateli zobrazovat někdy Category někdy Family. I v tomto textu bude pojem „kategorie“ myšlen obecně pro obě části.

Přístup do kategorií bude umožněn v druhé úrovni menu v částech Requirements a Products pod odkazem Categories. Každá operace s kategorií bude mít vlastní stránku:

- New pro vytvoření nové kategorie,
- Edit pro editaci kategorie,
- View pro prohlédnutí kategorie,
- Delete pro smazání kategorie a
- List pro zobrazení seznamu kategorií.

To si vyžádá třetí úroveň v menu a také v celé struktuře aplikace. Ve složkách www/pages/requirements, www/pages/products a www/includes vytvoříme složky s názvem categories. Stejnou strukturu zavedeme i v šablonách ve složce templates. Všechny obslužné skripty budou ve složce includes, ve složkách requirements a products budou jen jednořádkové skripty, které budou načítat skripty ve složce includes, čili stejný způsob, jaký byl používán o úroveň výš.

Možné bude členit kategorie do úrovní, kdy jedna kategorie bude moci být součástí jiné kategorie. Vznikne tím stromová struktura. Nadřazená kategorie, tj. kategorie na vyšší úrovni, bude rodič. Podřízená kategorie, tj. kategorie na nižší úrovni, bude potomkem rodiče. Kategorie, které nemají rodiče, budou na nejvyšší úrovni stromu, nebo-li budou tvořit kořen stromu. V tabulkách kategorií tyto vazby uskutečňuje sloupec parent_id. Hodnota v tomto sloupci ukazuje na rodičovskou kategorii. Pokud tato hodnota není nastavena, je kategorie v kořeni stromu. Stromová struktura kategorií však přináší určité problémy. Musíme dávat pozor, aby tato struktura byla skutečně stromovou, tj. aby nevznikaly slepé větve, smyčky a jiné nekonzistence.

6.1 Udržování konzistentního stromu kategorií

Vnořování a možnost vytváření stromové struktury kategorií vyžaduje určitou kontrolu konzistence stromu. Nesmíme dopustit, aby tato konzistence byla porušena. Máme-li dvě kategorie, z nichž každá uvádí ve sloupci parent_id hodnotu category_id druhé kategorie, dojde k porušení konzistence. Totéž nastane, bude-li kategorie odkazovat sama na sebe nebo na neexistující kategorii. Dostáváme tedy dvě podmínky, které musíme za všech okolností dodržovat:

- rodičem se může stát pouze jiná a existují kategorie, která však není potomkem editované kategorie, ani přes více úrovní stromu.
- nesmíme smazat kategorii, která je rodičem alespoň jedné kategorie.

Při editování kategorie budeme ve výběrovém seznamu Parents nabízet pouze kategorie, které odpovídají první podmínce. Kontrolu této podmínky provedeme i při ukládání kategorie. Druhou podmínku dodržíme tak, že odmítneme smazat kategorii, která má potomky.

Pro **vytvoření výběrového seznamu** kategorií použijeme funkci database_get_categories_tree, která vrátí pole kategorií seřazené do stromové struktury. Hodnoty pole budou tvořit názvy kategorií odsazené podle hloubky vnoření. Potomky editované kategorie funkce nevrátí, aby nemohlo dojít k porušení první podmínky.

Funkci předáme odkazem proměnnou $count, kterou funkce naplní počtem kategorií. Ve druhém parametru bude název tabulky kategorií, tj. requirement_category nebo product_category. Třetí parametr bude category_id hodnota kategorie, která nemá být zařazena do výpisu. Většinou půjde o kategorii, pro kterou tento strom sestavujeme. Výchozí hodnota bude NULL. Čtvrtý parametr je pole, ve kterém můžeme definovat prvky, které mají být vloženy na začátek stromové struktury. Výchozí hodnota bude jednoprvkové pole s hodnotou NULL, která bude sloužit pro výběr bez nadřazené kategorie. Třetí a čtvrtý parametr nebude nutné zadávat.

Běžným dotazem načteme neseřazené pole a uložíme jej do proměnné $list. Pokud byla nastavena proměnná $without_id, bude v poli $list chybět kategorie s touto category_id hodnotou. V cyklu do..while vyhledáme kategorie, které jsou mimo strom a z pole $list je odstraníme. Cyklus bude probíhat tak dlouho, dokud bude docházet k odstraňování záznamů. Je to z toho důvodu, že např. kategorie A může odkazovat na kategorii B, která ale odkazuje na neplatnou kategorii. Kategorie A je zpočátku platná a nebudeme ji mazat. Po smazání kategorie B se však i kategorie A stane neplatnou, takže ji v některém z dalších prů-

chodů cyklu do..while z pole odstraníme. Smazání kategorie bude identifikovat proměnná $deleted.

V prvním cyklu foreach projdeme pole $list a budeme prohledávat hodnoty parent_id. Nebude-li hodnota parent_id nastavena, přejdeme hned na další prvek pole, protože se jedná o kategorii v kořeni stromu a kategorie v kořeni stromu chybné být nemůžou. Je-li hodnota parent_id nastavena, pokusíme se najít kategorii s hodnotou category_id rovné této hodnotě. To provede druhý cyklus foreach. Po nalezení přejdeme příkazem continue 2 na další iteraci nadřazeného cyklu, takže se oba nedokončí.

Nebude-li rodičovská kategorie nalezena, vnitřní cyklus foreach se dokončí a kategorii z pole $list odstraníme. Zároveň snížíme celkový počet kategorií a proměnnou $deleted nastavíme na true, což oddálí ukončení cyklu do..while. V jednom průchodu tohoto cyklu může být odstraněno i více kategorií. Uvedený algoritmus také zajistí, že z pole $list2 budou odstraněni všichni potomci (ze všech úrovní) kategorie s category_id hodnotou v proměnné $without_id. Tím dosáhneme toho, že uživateli při editaci kategorie nabídneme pouze platné rodičovské kategorie.

Zavedeme prázdné asociativní pole $list2, do kterého budeme v následujícím cyklu while vkládat kategorie z pole $list. Klíče k tomuto poli budou desetinná čísla, ale v datovém typu string. Setřídíme-li výsledné pole $list2 podle klíčů, získáme stromovou strukturu kategorií. Pro hodnoty klíčů budou platit následující pravidla: kategorie z první úrovně stromu, které nemají rodiče, získají celočíselnou hodnotu klíče. Každá následující kategorie bez rodiče bude mít hodnotu klíče zvýšenou o 1.

Určení klíče vnořené kategorie, která ještě nebyla vložena do pole $list2, bude složitější. Označme si vnořenou kategorii, pro kterou potřebujeme vypočítat klíč, jako X. Rodičovskou kategorii označíme jako R a N bude kategorie, která z hlediska pole $list2 následuje bezprostředně za rodičem. Kategorie N tedy může být přímým potomkem kategorie R, ale může být i na stejné nebo vyšší úrovni, nemá-li kategorie R v poli $list2 žádného potomka. Situaci znázorňuje obr. 6.1 a), odmyslíme-li si kategorie X1 a X2, které v daném okamžiku ještě nemohou být zařazené v poli $list2. Klíč získáme následujícím výpočtem:

$$k(X) = \frac{k(N) - k(R)}{2} + k(R),$$

kde k je klíč dané kategorie. Klíč kategorie X je tedy číslo, které leží mezi klíči kategorií N a R. Pokud kategorii vložíme do pole $list2 s takto vypočítaným klíčem a pole podle klíčů setřídíme, bude pozice klíče hned za kategorií R.

V cyklu foreach budeme procházet pole $list. Kategorie, která nemá nastavenu hodnotu $list["parent_id"], je kategorie z kořene stromu. Vložíme ji na konec pole $list2 a její klíč bude celé číslo dané proměnnou $i. Tuto proměnnou pak zvýšíme o 1, prvek z původního pole vymažeme a nastavíme proměnnou $list2["$i"]["level"] na 0, což označuje nultou úroveň dané kategorie, čili kořen stromu.

Pro kategorii (X) s nastavenou hodnotou $list["parent_id"] musíme nalézt v poli $list2 rodičovskou kategorii (R). Ta v něm ještě nemusí být, takže v takovém případě danou kategorii nezařadíme a počkáme na zařazení rodičovské kategorie. Nalezneme-li rodičovskou kategorii (R), musíme pro nezatříděnou kategorii (X) vypočítat klíč. Pro hledání rodiče a výpočet klíče si vytvoříme funkci put_row, které odkazem předáme pole $list2 a proměnnou $row, která obsahuje nezařazenou kategorii (X).

Před vyhledáním seřadíme pole $list2 pomocí funkce ksort, která řadí podle klíčů. Projdeme pole $list2 a vyhledáme nadřazenou kategorii (R). Do proměnné $parent_key si uložíme její klíč a do proměnné $parent_level uložíme její úroveň. V následující iteraci cyklu foreach detekujeme, že byla nalezena rodičovská kategorie. Do proměnné $next_key si uložíme klíč z aktuální kategorie (N), čili kategorie, která v poli $list2 následovala po rodičovské kategorii (R). Cyklus předčasně ukončíme a provedeme výpočet klíče nezařazené kategorie (X). Může se stát, že rodičovská kategorie (R) bude na poslední pozici v poli $list2. Klíč následující kategorie (N) pak nemůže být detekován, protože kategorie (N) nebude existovat. V takovém případě pevně nastavíme proměnnou $next_key o 1 vyšší než je klíč rodičovské kategorie (R).

Podle výše uvedeného vzorce vypočítáme klíč nezařazené kategorie (X), jehož hodnotu uložíme do proměnné $new_key. Úroveň nezařazené kategorie (X) nastavíme o 1 vyšší, než je úroveň rodičovské kategorie (R) v proměnné $parent_level. V cyklu for, jehož počet průchodů nastavíme na číslo úrovně nezařazené kategorie (X), budeme vkládat před název kategorie prázdné mezery. Tím docílíme toho, že uživatel ve výpisu kategorií snadno rozpozná stromovou strukturu. Nakonec nezařazenou kategorii s vypočítaným klíčem $new_key vložíme do pole $list2 a vrátíme true, což bude znamenat úspěšně zařazenou kategorii do pole $list2.

Jestliže v předchozím cyklu foreach rodičovská kategorie (R) nalezena nebyla, ke vložení nezařazené kategorie (X) do pole $list2 nedojde a funkce put_row vrátí false. To bude také detekováno v nadřazené funkci, která v takovém případě nezařazenou kategorii (X) z pole $list neodstraní a jak už bylo řečeno, udělá to až po zařazení rodičovské kategorie (R).

K opakovanému hledání rodičovské kategorie slouží nadřazený cyklus while, který probíhá tak dlouho, dokud se pole $list nevyprázdní a současně pole $list2 nenaplní všemi prvky.

Zbývá již pouze vložení předefinovaných hodnot z pole $start, které ve výchozím stavu obsahuje kořen stromu, což uživateli umožní nezadat rodičovskou kategorii. Vytvoříme nové pole $tree zkopírováním pole $start. V cyklu foreach projdeme pole $list2 a všechny kategorie převedeme do pole $tree. Toto pole pak vrátíme skriptu, který funkci volal. Možné výsledky vidíme na obr. 6.1. Na levém obrázku (při vytváření nové kategorie) nebyla nastavena proměnná $without_id, funkce proto vrátila kompletní strom. Na pravém obrázku byla prováděna editace kategorie X. Její hodnota category_id byla nastavena v proměnné $without_id, takže funkce vrátila strom bez této a všech podřízených kategorií.

Uživatel si tedy vybral rodičovskou kategorii a formulář odeslal. V obslužném skriptu pomocí funkce database_test_category **otestujeme platnost rodičovské kategorie**, kterou jsme funkci předali v proměnné $category_id. Pokud hodnota $category_id neexistuje, znamená to, že uživatel nevybral žádnou rodičovskou kategorii a ukládaná kategorie má být v kořeni stromu. V takové případě to není chyba a funkce vrátí true. Pokud ale ukládaná kategorie ukazuje sama na sebe, čili hodnota $category_id se rovná hodnotě $webdoc["category_id"], je to chyba a vrátíme false.

V další kontrole se postupně pokusíme dobrat ke kořeni stromu přes rodičovský záznam v proměnné $category_id. Pokud se přes hodnoty parent_id nadřazených záznamů dostaneme ke kořeni, je hodnota parent_id ukládaného záznamu správná. Proměnnou $found nastavíme na false a až po nalezení kořene ji nastavíme na true. Cyklus do..while bude probíhat tak dlouho, dokud nebude nalezen kořen stromu, nebo dokud nebude rozpoznán nějaký chybový stav.

V cyklu foreach budeme procházet jednotlivé kategorie načtené funkcí database_get_categories_list. Bude-li některá z hodnot category_id kategorie z pole $list rovna proměnné $category_id, našli jsme rodičovskou kategorii. Zkontrolujeme, zda-li rodičovská kategorie z pole $list neukazuje sama na sebe. V takovém případě bychom vrátili false, avšak teoreticky by k tomuto stavu nikdy nemělo dojít. Pokud rodičovská kategorie nemá nastavenu hodnotu parent_id, je tato kategorie v kořeni stromu, takže vrátíme true. Ukazuje-li rodičovská kategorie na ukládanou kategorii, jejíž hodnota category_id je v proměnné $webdoc["category_id"], vrátí funkce false. Rodičovská kategorie by byla zároveň potomkem. Došlo by tím ke smyčce, což nesmí nastat.

Nebyla-li splněna žádná z předchozích podmínek, uložíme do proměnné $category_id hodnotu category_id rodičovské kategorie. Přesuneme se tak o úroveň výš a pro tuto kategorii

budeme opět hledat rodiče. To zajistíme nastavením proměnné $found na true, takže celý cyklus foreach se bude provádět minimálně ještě jednou. Tímto způsobem budeme po jednotlivých kategoriích směřovat ke kořeni. Až jej dosáhneme, vrátí funkce true.

V případě, že v průběhu celého cyklu foreach nebude nalezena rodičovská kategorie, nebude nastavena proměnná $found na true. Oba cykly se ukončí a funkce vrátí false, jinak by mohla vzniknout slepá větev stromu.

Před smazáním určité kategorie provedeme **test existence potomků** mazané kategorie. Kategorii, která má potomky, nesmíme smazat. Potomci by pak ukazovali na neexistující kategorii, což by porušilo konzistenci stromu. Test provede funkcí database_has_category_children, která vrátí true, nebo false podle toho, zda má aktuální kategorie potomky. Funkce také bude testovat, zda je do aktuální kategorie zařazen některý záznam z tabulek requirement nebo product. Takový záznam by se také dal považovat za potomka a funkce by tedy vrátila true.

Nejprve zjistíme, jestli existují kategorie s hodnotou parent_id rovnající se aktuální hodnotě $webdoc[category_id]. Dotaz by měl vrátit všechny kategorie z první podřízené úrovně. Je-li vrácena alespoň jedna kategorie, má aktuální kategorie potomky a funkce vrátí true. V opačné případě aktuální kategorie nemá žádné podřízené kategorie a pokračovat budeme kontrolou sloupců category_id v tabulkách requirement nebo product. Bude-li vrácen některý záznam s hodnotou category_id rovnající se proměnné $webdoc["category_id"], vrátí funkce true a kategorii nepůjde smazat. Taková kategorie bude v podstatě nesmazatelná, protože ani záznamy v tabulkách requirement a product nelze mazat. Nebudou-li vráceny žádné záznamy, nemá aktuální kategorie žádné potomky a bude možno ji smazat. Funkce vrátí false.

6.2 Úpravy a zobrazování

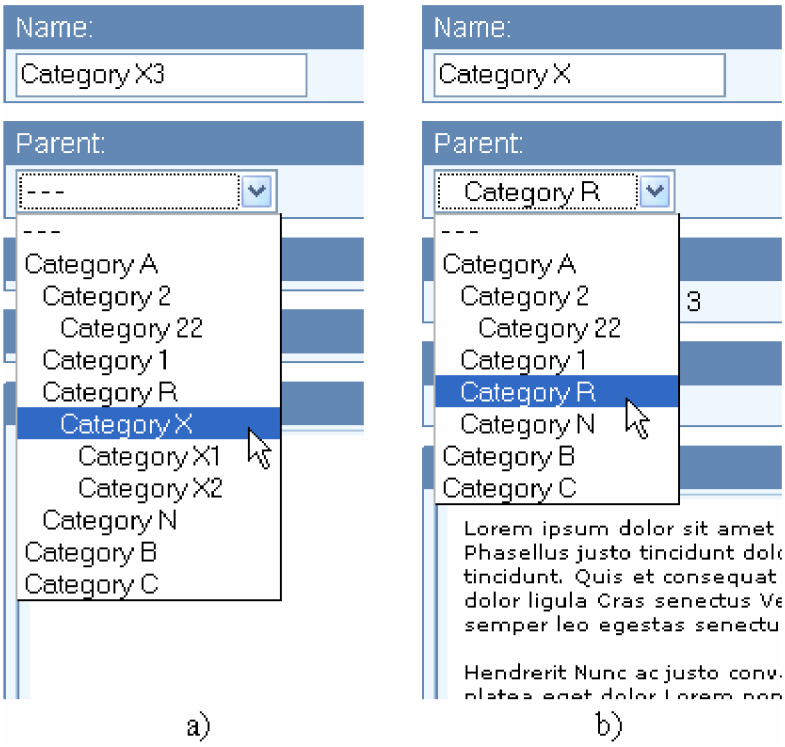
Editace kategorií sice neznamená vytvoření nového záznamu v tabulce databáze, protože zde neukládáme historii záznamů, přesto bude výhodné tuto operaci společně s **vytvářením nových kategorií** obsluhovat jedním skriptem. Rozdíl bude pouze v použitém dotazu. Stejně tak použijeme jednu šablonu (edit.tpl, obr. 6.1), kterou si teď popíšeme. Obsahovat bude formulář s následujícími formulářovými poli:

- Name pro zadání názvu kategorie (textové formulářové pole),
- Parent pro výběr nadřazené kategorie. Výběr bude uskutečněn pomocí výběrového seznamu, v němž budou nabídnuty již existující kategorie,
- Time pro zobrazení času vytvoření nebo modifikace kategorie,

- Person pro zobrazení uživatele, který kategorii naposledy modifikoval,
- Description pro zadání popisu kategorie prostřednictvím WYSIWYG editoru.

Pole Time a Person pouze vypíšeme, nebude možné je editovat. Na konec formuláře doplníme tlačítko Save pro uložení dat.

Obslužný skript edit.php bude podobný skriptu edit.php v nadřazené úrovni: v nadpisu rozlišíme, zda se jedná o editaci nebo vytvoření nové kategorie, ošetříme přístupová práva, zobrazíme případnou zprávu o úspěšném vytvoření nové kategorie po přesměrování a převedeme data z odeslaného formuláře do pole $record.



Obr. 6.1: Výběr rodičovské kategorie při
a) vytváření nové kategorie, b) editaci kategorie

Do pole $tree načteme existující kategorie ve stromové struktuře a přiřadíme jej do Smarty, kde bude použito pro výběr nadřazené kategorie ve výběrovém seznamu. Pokud editujeme kategorii, bude potřeba ji načíst nejprve do proměnné $tmp. Současně zkontrolujeme, zda záznam s danou hodnotou category_id existuje. K načtení kategorie použijeme funkci database_get_category, kterou popíšeme vzápětí. Zkontrolujeme, zda-li bylo kliknuto na tlačítko Save. Pokud na tlačítko kliknuto nebylo, převedeme záznam z proměnné $tmp do proměnné $record. Záznam pak přiřadíme do Smarty.

Další část skriptu provedeme pouze, pokud byl odeslán formulář. Zkontrolujeme formulářová pole Name a Description, zda byla vyplněna. U proměnné $record["parent_id"]

ověříme, zda kategorie, jejíž hodnotu category_id proměnná obsahuje, může být pro ukládanou kategorii rodičovská. Nesmíme uložit jako rodiče kategorii, která má sama jako svou rodičovskou kategorii nastavenu právě ukládanou kategorii.

Po testech zkontrolujeme, zda bylo naplněno pole $webdoc["message"]. V takovém případě skript ukončíme bez uložení. Nebyla-li zjištěna žádná chyba, uložíme kategorii pomocí funkce database_save_category. Při editaci kategorii znovu načteme, abychom zjistili čas vytvoření a editora kategorie. Při vytvoření kategorie k tomuto nedojde, neboť funkce database_save_category přesměruje na stránku Edit a skript ukončí.

Tato funkce musí rozlišovat, zda-li jde o vytváření nové kategorie nebo o editaci kategorie. Pozná to podle url parametru subpage3, který bude roven buď "new" nebo "edit". Při vytváření kategorie použijeme dotaz s příkazem INSERT. Po úspěšném vytvoření zjistíme hodnotu category_id, kterou funkce nastaví do proměnné předané odkazem. U editace použijeme příkaz UPDATE. V části WHERE omezíme na category_id upravovaného záznamu.

Po vytvoření nové kategorie přesměrujeme na stránku Edit. Ještě před tím nastavíme proměnnou v poli $_SESSION, která zajistí, že se po přesměrování zobrazí zpráva o úspěšném uložení kategorie. Pak již provedeme samotné přesměrování a skript ukončíme.

Velice jednoduché bude **zobrazení jedné kategorie**. Ve skriptu view.php kromě kontroly práv zkontrolujeme, zda existuje proměnná $webdoc["category_id"] zobrazované kategorie. Kategorii pak pomocí funkce database_get_category načteme do pole $record a přiřadíme do Smarty. Dotaz ve funkci provede propojení tabulky kategorií na tutéž tabulku, kde parent_id první tabulky bude propojeno s category_id druhé tabulky. Získá tak jméno rodičovské kategorie, které vrátí v proměnné parent. Dále provede propojení s tabulkou person pro získání jména a příjmení uživatele, který kategorii naposledy upravil. Vrátí jej v proměnné person. Získané pole $record zobrazíme v šabloně view.tpl. Tato šablona se bude velmi podobat šabloně edit.tpl. Rozdíl bude pouze v absenci formulářových polí

Šablonu view.tpl použijeme také na stránce pro **mazání kategorií**. Samotná šablona delete.tpl před smazáním kategorie načte šablonu view.tpl, která zobrazí prostřednictvím šablony message.tpl dotaz na smazání a samotnou kategorii. Po smazání kategorie už šablonu view.tpl nenačteme, pouze zobrazíme zprávu o úspěšném smazání. Zda byla kategorie smazána, určí proměnná $deleted přiřazená do Smarty ze skriptu delete.php. Tento skript načte kategorii do pole $record a zjistí, zda má kategorie potomky – podřízené kategorie nebo záznamy z tabulky requirement nebo product. Jestliže kategorie má potomky, nemůžeme ji smazat. Zobrazíme pouze zprávu o tom, že kategorie má potomky.

V další podmínce budeme ověřovat, zda url parametr confirm má hodnotu "yes". Před smazáním dotazu se uživateli zobrazí dotaz, zda chce kategorii skutečně smazat. Za dotazem se zobrazí dva odkazy – Yes a No. Odkaz Yes má parametr confirm nastavený na "yes" a odkaz No má tento parametr nastavený na "no". Až potom, co uživatel klikne na odkaz Yes, je možné kategorii smazat. V případě, že na odkaz Yes kliknuto nebylo, otestujeme, zda byl odeslán parametr confirm. Pokud nebyl odeslán, zobrazíme dotaz na smazání kategorie.

Teprve až uživatel klikne na odkaz Yes, zavoláme funkci database_delete_category, která jednoduchým dotazem vymaže záznam kategorie z databáze. Pak odstraníme proměnnou $category_id z globálního pole $webdoc i ze sessions. Do pole $_SESSION uložíme proměnnou, jejíž název sestavíme z tabulky kategorií a řetězce "_record_deleted". Vytvoříme řetězec $url, který bude obsahovat novou url bez hodnoty category_id původní kategorie. Na tuto adresu pak přesměrujeme. Proměnná z pole $_SESSION, která nese informaci o smazání, bude po přesměrování na začátku skriptu detekována, takže se zobrazí zpráva o smazání kategorie. Po načtení zprávy proměnnou smažeme.

Pro **výpis kategorií** převezmeme skript pro výpis záznamů, jen jej mírně zjednodušíme. Jméno souboru bude taktéž list.php. Budeme zde používat stránkování a řazení, ne však filtr a sloupce. Funkci kernel_get_url tedy použijeme pro vygenerování polí $pages a $sort. Pole $filter a $columns funkce nenaplní. Ponecháme stejný způsob naplnění proměnných $url, $pages["url"] a $sort2["url"], budou však chybět parametry filtru a sloupců.

Vytvoříme funkci database_get_categories_list, která nám načte kategorie. Tato funkce bude také vycházet z funkce pro načtení záznamů - database_get_list. První a druhý parametr funkce naplní počtem vrácených záznamů a počtem záznamů bez uvažování stránkování. V druhém a třetím parametru funkci předáme parametry stránkování a řazení. Nejprve vytvoříme dotaz, který zjistí počet záznamů bez uvažování stránkování. Výslednou hodnotu uložíme do proměnné $total.

Druhým dotazem načteme samotné záznamy. Provedeme v něm propojení se stejnou tabulkou, kterou vypisujeme, abychom zjistili jméno nadřazené kategorie. Další propojení bude s tabulkou person pro zjištění uživatele, který záznam modifikoval. Pak zpracujeme řazení a stránkování, stejně jako tomu bylo ve funkci database_get_list. Výsledný dotaz provedeme a vrátíme seznam kategorií.

Funkce kernel_create_pages a kernel_create_sort dopočítají další parametry pro stránkování a řazení, např. seznam stránek, počty příspěvků na stránku a příslušné odkazy. Pak jen přiřadíme příslušné proměnné do Smarty. Sloupce nastavíme napevno. Šablona list.tpl pomocí funkce view_result zobrazí seznam kategorií.

7. Export a tisk

V poslední kapitole ukážeme, jak uživatelům umožnit tisk a export záznamů do souborů různých formátů, což bude jistě vítanou službou. Můžeme exportovat výpis záznamů, historii záznamu nebo detail jednoho záznamu. Podle toho nabídneme některé běžné formáty:

- sešit aplikace Microsoft Excel (xls) pro výpis záznamů a historie záznamu,
- Portable Document Format (pdf) pro export jednoho záznamu a
- dokument aplikace Microsoft Word (doc) pro export jednoho záznamu.

Pro výběr exportovaných dat a formátu zavedeme v každé části aplikace stránku Export, kde toto půjde nastavit. Určitým usnadněním také bude možnost exportovat data buď tak, jak by se zobrazila na stránce List nebo History, tj. s nastavenými nástroji, nebo exportovat všechny záznamy bez ohledu na nastavené nástroje.

Stránky budeme tisknout pomocí standardních tiskových dialogů, které běžné webové prohlížeče nabízejí. V CSS umíme vytvořit speciální kaskádový styl, který zajistí, že stránka, kterou uživatel vytiskne, bude vypadat na papíře jinak než na monitoru. Tiskovým stylem můžeme vytištěnou stránku zcela libovolně naformátovat, aniž by se to dotklo formátování stránky v prohlížeči. Nebude nutné vytvářet speciální stránky pro tisk. Tiskový styl nastavíme tak, aby vytištěná stránka odpovídala zvyklostem formátovaného textu na papíře.

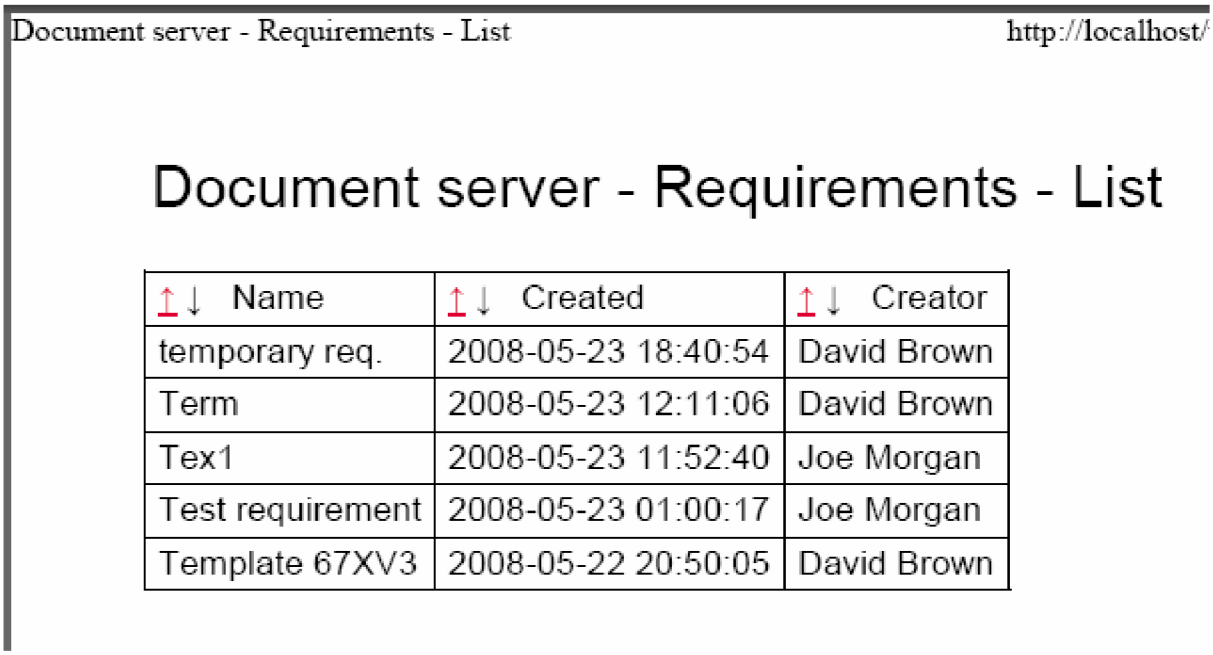
7.1 Tisk

Úkolem tiskového stylu bude změnit všechny barvy na černou, aby jsme stránku lépe přizpůsobili černobílé tiskárně, případně zbytečně neplýtvali barvou tiskárny barevné. Ze stejného důvodu zrušíme barevné pozadí, což často dělají prohlížeče automaticky. Důležitým úkolem bude také skrytí všech nepotřebných částí stránky, např. menu.

Tiskový styl uložíme do souboru print.css ve složce www. Mezi tagy `<head>` v souboru index.php přidáme tag `<link>`, který ke stránce tiskový styl připojí. Přidáme atribut `media`, který nastavíme na hodnotu `"print"`. Tím dosáhneme toho, že tiskový styl se použije pouze pro tisk stránky. Půjde vlastně jen o přestylování prvků stránky, které mají na papíře vypadat jinak, případně tam vůbec nemají být.

První tedy schováme menu, které na papíře potřebovat nebudeme. U třídy `menu` nastavíme vlastnost `display` na `none`. Pro prvky, které nejsou adresovatelné žádnou třídou nebo identifikátorem, jako jsou třeba jednotlivé řádky tabulky, zavedeme novou třídu `noprint`, které

taky nastavíme vlastnost display na hodnotu none. Projdeme všechny šablony a vyhledáme všechny prvky, které nechceme tisknout na papír. Těmto prvkům nastavíme třídu noprint. Ve výpisu záznamů tuto třídu nastavíme řádkům tabulky, které obsahují stránkování a textový popis filtru. Skryjeme také prázdné řádky, které jsme vkládali před záhlaví v tabulkách. Buňkám tabulek schováme pozadí, barva písma a ohraničení bude černá.



Obr. 7.1: Vytištěné záznamy

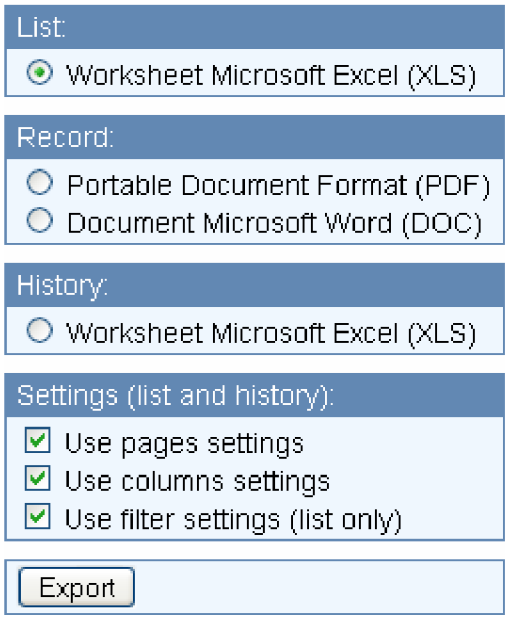
7.2 Šablona pro export

Pro nastavení výběru exportovaných dat a formátu, do kterého se má exportovat zavedeme novou stránku Export. Odkaz na tuto stránku bude standardně v druhé úrovni menu ve všech částech aplikace. Dosud jsme měli jednotlivé stránky rozdělené podle toho, zda se týkaly výpisu záznamů, nebo operace s jedním záznamem (včetně výpisu historie záznamu). Stránka Export však bude jediná, která se bude týkat obou těchto částí.

Připravíme si nejprve šablonu export.tpl. Nastavení exportu budeme provádět pomocí zaškrtávacích polí a přepínačů, takže použijeme formulář. Odesílat jej budeme metodou get, abychom mohli použít všechny funkce pro práci s nástroji, které jsou na této metodě postavené. Kvůli exportu jednoho záznamu budeme ve skrytém formulářovém poli uvádět i id hodnotu tohoto záznamu. Dále do skrytých polí dáme parametry subpage1, subpage2 a parametry všech nástrojů, které vloží příslušné šablony.

Ve formuláři zobrazíme přepínače, které umožní nastavit exportovaná data a formát. Rozdělíme je do tří částí: export výpisu záznamů, jednoho záznamu a historie záznamu. Výpis

záznamů a historii záznamu budeme exportovat do formátu xls. Detail jednoho záznamu pak do formátů pdf a doc. Všechny přepínače budou mít jméno "export" a hodnotu podle toho, která data do kterého formátu exportují, např. "list_to_xls" nebo "record_to_pdf". Přepínače pro export jednoho záznamu a historie záznamu se budou zobrazovat pouze bude-li známa id hodnota záznamu.



Obr. 7.2: Nastavení exportu

Uživateli dáme na výběr, zda chce exportovat data podle nastaveného stránkování, filtru a sloupců, nebo zda chce exportovat data bez ohledu na nastavení těchto nástrojů. Pomocí třech zaškrtávacích polí umožníme vybrat každý nástroj zvlášť. Pokud bude pole zaškrtnuto, nástroj bude použit. Na konci formuláře zobrazíme tlačítko Export.

7.3 Exportování dat

Obslužný skript export.php ve složce www/includes bude mít dva hlavní úlohy: přípravu stránky a posílání exportovaných souborů. Nejprve připravíme proměnné pro šablonu, aby mohla vygenerovat skrytá formulářová pole s parametry nástrojů. Z url nebo z cookies přečteme proměnnou $export, podle které se přednastaví druh a formát exportu. Všechny proměnné pak přiřadíme do Smarty.

Uživatel, který klikne na tlačítko Export, bude očekávat, že mu bude nabídnut soubor ke stažení. Skript, který exportuje data, musí poslat pouze soubor s těmito daty včetně hlaviček, aby prohlížeč správně soubor zpracoval. Kromě tohoto souboru nesmíme poslat žádná jiná data, takže ani html kód stránky. Odeslaný formulář tedy zpracujeme hned na začátku

skriptu, tj. před zpracováním proměnných pro stránku. Pokud zpracování formuláře proběhne bez chyb, musíme skript ukončit, aby se k poslanému souboru nepřipojil html kód stránky.

Podle existence parametru export v url poznáme, že byl odeslán formulář. Otestujeme správné hodnoty v této proměnné a také ověříme existenci a platnost id hodnoty, pokud se má exportovat jeden záznam. Pak uložíme do cookies nastavení nástrojů. V případě, že nedošlo k chybě, uložíme do cookies i druh exportu a načteme knihovnu s funkcemi pro export.

U exportu jednoho záznamu musíme nastavit položky, které mají být exportovány. Podle tabulky, ze které záznam pochází, nastavíme pole $titles, které bude obsahovat popisky položek, které mají být zobrazeny. Pole $items bude obsahovat názvy sloupců, jejichž obsah bude vypsán v položkách. Do pole $items2 vložíme názvy sloupců, jejichž obsah se má vypsat na druhý řádek v položkách. Bude to využito např. pro dvojice sloupců created a creator. Do proměnné $header uložíme název části aplikace, ze které byl záznam vypsán.

Záznam načteme do pole $record. Provedeme úpravy proměnných, které nemají nastavenu hodnotu, aby se v příslušné položce zobrazilo No nebo pomlčka. Podle druhu exportu zavoláme funkci, která vygeneruje soubor.

Exportujeme-li výpis záznamů nebo historii jednoho záznamu, musíme zpracovat nastavení nástrojů. Z pole $columns odstraníme sloupec Actions, který v exportovaném souboru nebude mít význam. Proměnné nástrojů, které nebyly zaškrtnuté, zrušíme. Pokud nemá být použito nastavení sloupců, musíme sestavit nové pole $columns, které bude obsahovat seznam všech sloupců u dané tabulky. V cyklu foreach toto pole projdeme a do hodnot uložíme popisky sloupců. Pak již můžeme pomocí funkce database_get_list načíst záznamy. Funkce použije pouze nástroje, které byly povolené v nastavení exportu. Po načtení záznamů zavoláme funkci export_list_to_xls, která vygeneruje soubor formátu xls.

Jestliže dosud nedošlo k chybě, byl exportovaný soubor odeslán úspěšně. Skript již nesmí nic dalšího generovat, proto jej ukončíme. U uživatele se to projeví tak, že se mu zobrazí dialogové okno pro stažení souboru. K načtení nové stránky nedojde a ta původní zůstane zachována. Skript nepřerušíme pouze v případě, že k chybě dojde. K odeslání souboru ani hlaviček nedošlo, takže můžeme stránku znovu načíst a zobrazit chybovou.

Pro export záznamů zřídíme novou knihovnu funkcí, kterou uložíme do složky libraries a pojmenujeme export.php. Do tohoto souboru budeme postupně přidávat funkce, kde každá bude mít na starost export do jednoho formátu.

Export výpisu záznamů do formátu xls bude provádět funkce export_list_to_xls. V proměnné $list funkci předáme výpis záznamů a v proměnné $columns předáme sloupce, které se mají v dokumentu zobrazit. Funkce bude exportovat výpis záznamů a historii zá-

znamu. K samotnému exportu dat použijeme volně dostupnou knihovnu php_writeexcel [24], která umožňuje výsledný dokument téměř libovolně formátovat, vkládat více listů a ukotvovat příčky. S přehledem předčí jiná, většinou polovičatá řešení [26], [27] a [28].

Ve složce libraries vytvoříme složku export, kam budeme vkládat externí knihovny pro export dat. Pro knihovnu php_writeexcel vytvoříme složku se stejným názvem, kam nahrajeme její soubory. Ve funkci export_list_to_xls pak příkazem require_once knihovnu načteme. Knihovna vytvoří dočasný soubor, který bude odeslán uživateli. Proměnná $fname bude na tento soubor ukazovat. Vytvoříme jej funkcí tempnam, která ho uloží do složky dočasných souborů. Do proměnné $workbook uložíme ukazatel na nový objekt sešitu a do něj pak příkazem addworksheet přidáme nový list, který pojmenujeme 'List'.

Příkazem freeze_panes ukotvíme příčky tak, aby se první řádek, který bude obsahovat názvy sloupců, neschovával, bude-li uživatel rolovat dolů. Formátování nastavíme příkazem addformat. Názvy sloupců v prvním řádku budeme zobrazovat tučně. Výchozí vybranou buňku nastavíme příkazem set_selection. Nastavíme ji na první buňku s daty.

	A	B	C	D	E
1	**Name**	**Created**	**Creator**		
2	temporary req.	2008-05-23 18:40:54	David Brown		
3	Term	2008-05-23 12:11:06	David Brown		
4	Tex1	2008-05-23 11:52:40	Joe Morgan		
5	Test requireme	2008-05-23 01:00:17	Joe Morgan		
6	Template 67XV	2008-05-22 20:50:05	David Brown		
7					

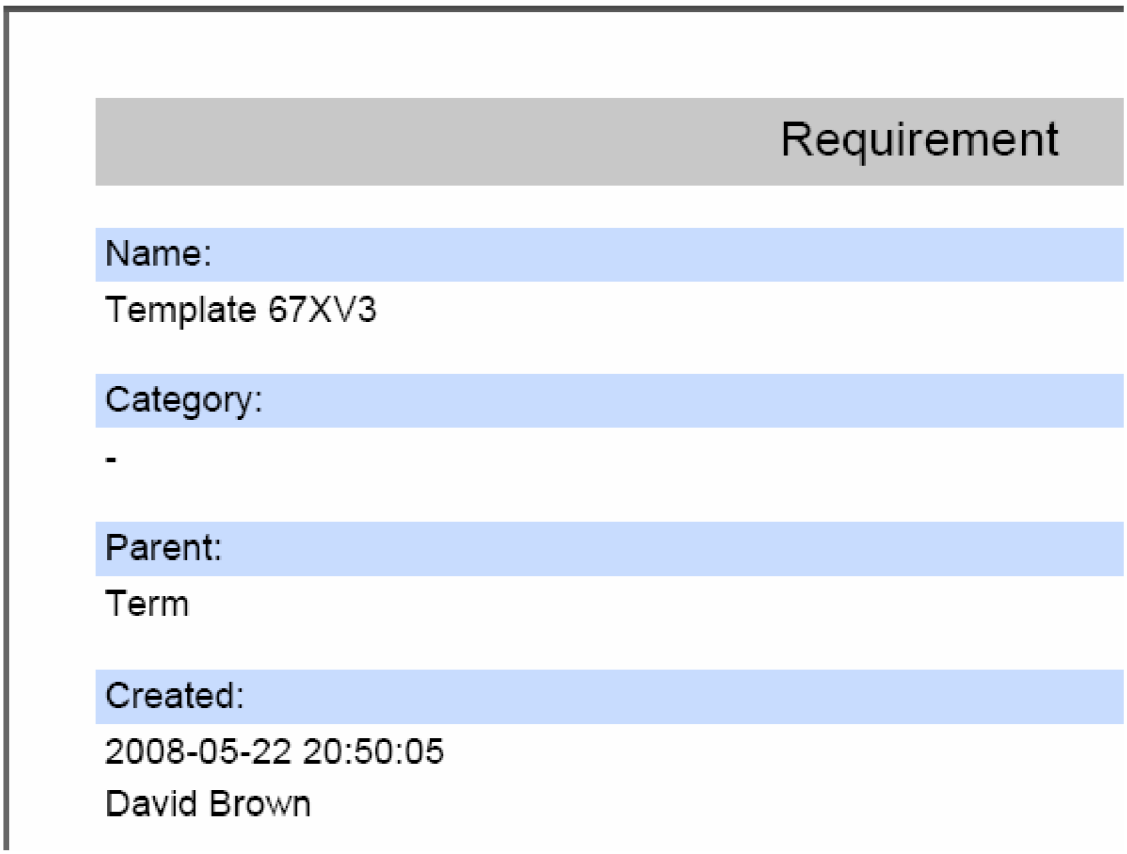
Obr. 7.3: Záznamy ve formátu xls

Sešit včetně listu máme připraven, takže jej můžeme naplnit daty. Do prvního řádku listu vložíme názvy sloupců. Cyklem foreach projdeme pole $columns a příkazem write budeme zapisovat názvy sloupců do buněk listu. Číslo řádku bude vždy 0, číslo sloupce bude určeno proměnnou $i, která bude v každém průchodu cyklem inkrementována o 1. Šířku sloupce nastavíme přibližně tak, aby odpovídala délce dat ve sloupci. V dalších dvou cyklech foreach zapíšeme samotné záznamy. V prvním cyklu projdeme pole $list po záznamech a v druhém cyklu projdeme sloupce. Konkrétní sloupec pak určí, která položka ze záznamu se vypíše. Proměnné $i a $j určí číslo řádku a sloupce, tj. buňku, do které bude položka zapsána.

Po zapsání všech dat sešit uzavřeme příkazem close. Pomocí funkce header pošleme hlavičky Content-Type a Content-Disposition [29], aby prohlížeč správně rozpoznal soubor ke stažení. Název souboru sestavíme z řetězce "Export ", části aplikace, ze které jsou data

exportována a přípony xls. Soubor otevřeme a funkcí fpassthru vypíšeme, čímž dojde k odeslání obsahu souboru. Dočasný soubor po vypsání odstraníme.

Pro **export jednoho záznamu do formátu pdf** vytvoříme funkci export_record_to_pdf. Záznam funkci předáme v proměnné $record, v proměnné $header předáme jméno části aplikace, ze které je záznam exportován. Pole $titles, $items a $items2 budou obsahovat popisky a názvy položek (sloupců), které budou vepsány do dokumentu. K exportu do pdf použijeme knihovnu FPDF [30], [32], která má velké možnosti formátování a dovoluje použít několik druhů písem. Velmi podobná (možnostmi i kódem) je knihovna Cookdojo Home [31].

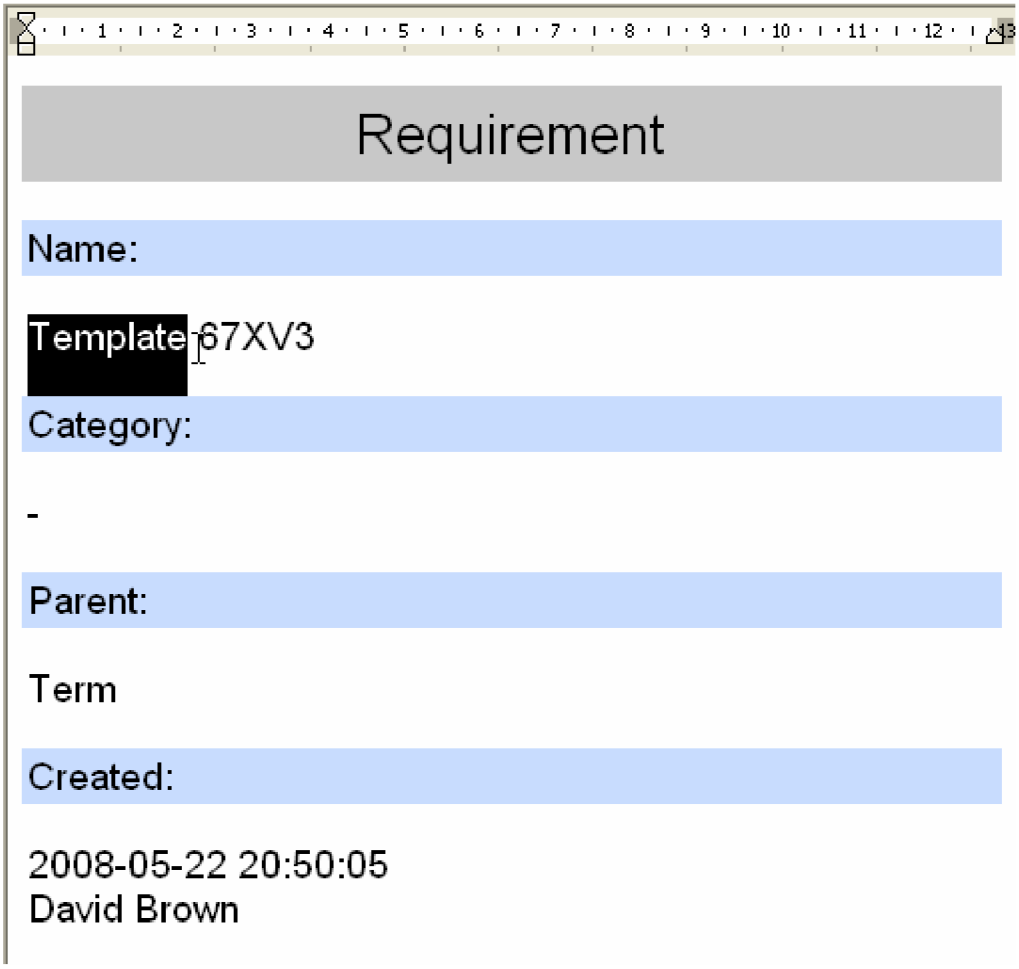


Obr. 7.4: Záznam ve formátu pdf

Knihovnu FPDF rozbalíme do složky libraries/export/fpdf. Soubor fpdf.php pak načteme příkazem require. Vytvoříme třídu PDF, která zdědí metody původní třídy FPDF. V této třídě zavedeme funkci item, která zapíše jednu položku záznamu do dokumentu. K této funkci se za chvíli vrátíme. Exportovaný dokument budeme vytvářet pomocí objektu $pdf. Příkazy SetTitle a SetAuthor zadáme titulek dokumentu (podle části aplikace, ze které exportujeme a jména záznamu) a jméno uživatele, který záznam vytvořil. Příkazem AddPage vytvoříme stránku pdf dokumentu a příkazem SetFont nastavíme písmo a jeho velikost.

Nyní můžeme do dokumentu zapisovat. Nejprve zapíšeme jméno části aplikace, ze které záznam exportujeme. Příkazem SetFillColor nastavíme pozadí textu. Příkazem Cell zapíšeme text. Pro zalomení použijeme příkaz Ln. V cyklu foreach budeme procházet jednotlivé položky záznamu. Zapisovat je budeme pomocí funkce item, kterou teď naprogramujeme.

Nejprve nastavíme barvu pozadí tak, aby bylo možné rozpoznat popisek položky. Tento popisek pak zapíšeme do dokumentu. Ověříme, zda je položka typu array. Položky typu array zapíšeme v cyklu foreach. Pokud položka není typu array, rovnou ji vypíšeme. Pokud existuje příslušný prvek pole $item2, vypíšeme jej také. Po zapsání všech položek použijeme příkaz Output pro odeslání pdf souboru.



Obr. 7.5: Záznam ve formátu doc

Funkce pro **exportování jednoho záznamu do formátu doc** bude fungovat velmi podobně. Nepoužijeme však externí knihovnu, protože žádná k dispozici není. Využijeme však vlastnosti aplikace MS Word, která dovede otevřít html soubor s příponou doc [25]. Export do formátu doc tedy bude spočívat ve vytvoření html souboru.

Výstup souboru budeme ukládat do proměnné $output. Nejprve vložíme nastylovaný tag <h1> s nadpisem části aplikace, ze které se záznam exportuje. Pak přidáme tag <body>, kde nastavíme písmo v dokumentu. Opět použijeme funkci item, která zapíše jednu položku záznamu. Kód funkce bude analogický s funkcí item pro export položky do formátu pdf. Do proměnné $output vložíme nastylovaný tag <h2> a název položky. Obsahem tabu <p> bude položka sama. Opět budeme rozlišovat, zda-li je položka typu array. V případě, že je položka typu array, použijeme pro její výpis cyklus foreach.

Po vložení celého záznamu pošleme příslušné hlavičky a obsah proměnné $output. Uživatel tak získá soubor se záznamem ve formátu doc, který aplikace MS Word bude schopna otevřít. Po uložení aplikací MS Word již bude soubor v běžném binárním formátu.

Závěr

Cílem této práce bylo popsat aplikaci pro správu vývojové dokumentace, její vývoj, funkce databáze a skriptů. Nejprve byla navržena databáze. Té byla věnována značná pozornost, neboť jedině na dobře navržené databázi bylo možné vytvořit fungující aplikaci. Důležité bylo pochopení vztahů mezi jednotlivými částmi aplikace. Díky tomu bylo možné definovat vhodné vazby mezi tabulkami. Započalo se návrhem hlavních tabulek, ke kterým byly postupně připojovány další tabulky pro doplňkové funkce. Zásadní byla implementace historie, což ovlivnilo celý následný vývoj. Již při návrhu databáze byly vyzkoušeny dotazy pro předpokládané operace s daty.

Po navržení databáze byl zahájen vývoj aplikace. Byla zavedena struktura skriptů a knihoven funkcí, které byly rozděleny podle řešených problémů a použitých technologií. Důraz byl kladen na možné budoucí úpravy aplikace, aby byla možná případná výměna některé technologie za jinou (např. databáze). Zaveden byl šablonovací systém, který striktně oddělil aplikační a prezentační část aplikace.

Největší prostor v práci byl věnován popisu řešení jednotlivých problémů, jakými byly např. operace se záznamy, jejich vytváření, upravování, kopírování a schvalování. Byly zavedeny kategorie a popsán byl problém udržení konzistentního stromu kategorií. Pro inteligentní výpisy záznamů bylo zprovozněno stránkování, řazení, filtr záznamů a možnost nastavování zobrazovaných sloupců. Tyto nástroje byly použity také při exportu dat do různých formátů. Exportovat bylo možné jak samostatný záznam, tak i výpis záznamů.

Aplikace nebyla dostatečně odladěna, protože nebyla dodána reálná data, která by aplikace měla spravovat. Testování bylo provedeno pouze s pseudodaty, které si autor sám vymyslel. Vyzkoušeno bylo velké množství postupů a operací, aby bylo možno odhalit chybové stavy. Ty také byly odhalovány a následně opravovány. Po těchto úpravách by již měly být chyby minimalizovány. Pravidelné testy byly prováděny také během vývoje. Nasazení do ostrého provozu by tak mělo být bezproblémové.

Nabízí se několik možných budoucích vylepšení, která sice nebudou mít bezprostřední vliv na fungování aplikace, ale určitě přispějí k pohodlí uživatelů. Všechna tato vylepšení již byla nad rámec této práce a proto nemohla být realizována. Výhodné by bylo použít souboru .htaccess [33] a [34], který dovoluje částečnou konfiguraci serveru Apache včetně módu rewrite. Tento mód dovede přepisovat url adresu stránky do přijatelnější podoby pro uživa-

tele. Současně by přepisování adresy umožnilo využít parametr path v cookies [35]. Tím by se dosáhlo, že jednotlivá cookies by se posílala jen těm stránkám, pro které by byla určena, což by nepatrně odlehčilo přenos dat.

Využití by měl i JavaScript, např. pro kontroly před smazáním. Nyní je aplikace plně funkční bez JavaScriptu, avšak aby mohl být zobrazen kontrolní dotaz před smazáním, musí být znovu načtena stránka. K tomu by při použití JavaScriptu nedošlo. Totéž platí pro kontrolu formulářových polí. Aplikace je také snadno rozšiřitelná o jazykové verze. V tabulce text by stačilo přidat sloupec s novou jazykovou verzí a upravit funkci, která texty načítá.

Export databáze a veškeré zdrojové kódy jsou uloženy na přiloženém cd.

Literatura

[1] KOLÁŘ, M.: Správa vývojové dokumentace přes WWW I. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 67 s. Vedoucí diplomové práce Ing. Miroslav Balík, Ph.D.

[2] MySQL AB: *MySQL Documentation* [online]. c1995-2008 [cit. 2007-10-11]. Dostupný z WWW: <http://dev.mysql.com/doc/>.

[3] ULLMAN, L.: *PHP a MySQL - Názorný průvodce tvorbou dynamických WWW stránek.* 1. vyd. Brno : Computer Press, 2004. 534 s. ISBN 80-251-0063-4.

[4] Microsoft Office Online: *Základy návrhu databáze* [online]. 2007 [cit. 2007-10-11]. Dostupný z WWW: <http://office.microsoft.com/cs-cz/access/HA012242471029.aspx>

[5] ZAJÍC, P.: *MySQL (1) - pestrý svět databází.* [online]. 2005 [cit. 2007-10-11]. Dostupný z WWW: <http://www.linuxsoft.cz/article.php?id_article=731>. ISSN 1801-3805.

[6] VRÁNA, J.: *PHP Triky : Weblog o elegantním programování v PHP pro mírně pokročilé* [online]. 2005-2008 [cit. 2007-10-23]. Dostupný z WWW: <http://php.vrana.cz/>.

[7] JUN, A.: *MySQL manuál* [online]. c2003-2005 [cit. 2007-10-11]. Dostupný z WWW: <http://mm.gene.cz/>.

[8] ZELENKA, P.: *Metody ukládání stromových dat v relačních databázích* [online]. Vydáno: 2.3.2005 [cit. 2007-12-16]. Dostupný z WWW: <http://interval.cz/clanky/metody-ukladani-stromovych-dat-v-relacnich-databazich/> ISSN 1212-8651

[9] RŮŽIČKA, P.: *Bezpečnost především* [online]. 2004-2007. [cit. 2007-11-24]. Dostupný z WWW: <http://interval.cz/serialy/bezpecnost-predevsim/>. ISSN 1212-8651

[10] PHP Documentation Group: *PHP : Manuál PHP* [online]. Vydáno: 2007-03-24 [cit. 2007-11-24]. Dostupný z WWW: <http://cz.php.net/manual/cs/>.

[11] KOSEK, J.: *PHP - tvorba interaktivních internetových aplikací.* 1. vyd. Praha : Grada Publishing, 1998. 492 s. ISBN 80-7169-373-1.

[12] JANOVSKÝ, D.: *Jak psát web, návod na html stránky* [online]. c1998-2008 [cit. 2007-12-16]. Dostupný z WWW: <http://www.jakpsatweb.cz>. ISSN 1801-045.

[13] VRÁNA, J.: *PHP Triky - Fulltextové vyhledávání v MySQL* [online]. Vydáno: 2006-09-13 [cit. 2008-03-15]. Dostupný z WWW: <http://php.vrana.cz/fulltextove-vyhledavani-v-mysql.php>.

[14] MySQL AB: MySQL 3.23, 4.0, 4.1 Reference Manual. *11.8.2 Boolean Full-Text Searches* [online]. c1995-2008 [cit. 2008-03-15]. Dostupný z WWW: <http://dev.mysql.com/doc/refman/4.1/en/fulltext-boolean.html>.

[15] VRÁNA, J.: *PHP Triky - Šablony* [online]. Vydáno: 2005-05-18 [cit. 2008-03-02]. Dostupný z WWW: <http://php.vrana.cz/sablony.php>.

[16] KRYL, M.: *Template Engine - teng* [online]. Vydáno: 2004-10-20 [cit. 2008-03-02]. Dostupný z WWW: <http://kryl.info/clanek/170-template-engine-teng>.

[17] Seznam.cz, a.s.: *Teng: Template Engine for PHP, C++ and Python* [online]. c1996-2008 [cit. 2008-03-02]. Dostupný z WWW: <http://teng.sourceforge.net/?page=home>.

[18] PHILIPP, Z.: *Python na webu 3.díl (Teng Template Engine)* [online]. Vydáno: 2008-02-09 [cit. 2008-03-02]. Dostupný z WWW: <http://www.karotka.cz/clanek/16-python-na-webu-3.dil-teng-template-engine.html>.

[19] Seznam.cz, a.s.: *Teng Manual* [online]. c2003-2006 [cit. 2008-03-02]. Dostupný z WWW: <http://teng.olmik.net/>.

[20] KOUBA, Š.: *SMARTY - šablonovací systém pro PHP* [online]. c2004-2008 [cit. 2008-03-05]. Dostupný z WWW: <http://interval.cz/serialy/smarty-sablonovaci-system-pro-php/> ISSN 1212-8651.

[21] MROZEK, J.: *Smarty v PHP 5* [online]. c2006 [cit. 2008-03-05]. Dostupný z WWW: <http://smarty.ronnieweb.net/>.

[22] New Digital Group, Inc: *Smarty: Template Engine* [online]. c2002-2008 [cit. 2008-03-05]. Dostupný z WWW: <http://www.smarty.net/>.

[23] STÝBLO, T., VRÁNA, J.: *htmltmpl: templating engine for separation of code and HTML* [online]. c2001-2007 [cit. 2008-03-02]. Dostupný z WWW: <http://htmltmpl.sourceforge.net/>.

[24] HANNE, J.: *php_writeexcel* [online]. c2002-2005 [cit. 2008-05-11]. Dostupný z WWW: <http://www.bettina-attack.de/jonny/view.php/projects/php_writeexcel/>.

[25] ŠEDO, J.: *Soubory MS Excel a MS Word v PHP, ASP či Notepadu* [online]. Vydáno: 2002-10-06 [cit. 2008-05-11]. Dostupný z WWW: <http://interval.cz/clanky/soubory-ms-excel-a-ms-word-v-php-asp-ci-notepadu/>.

[26] AppServNetwork: *Easy way to create XLS file from PHP* [online]. Vydáno: 2006-10-11 [cit. 2008-05-11]. Dostupný z WWW: <http://www.appservnetwork.com/modules.php?name=News&file=article&sid=8>.

[27] Chumby.net: *PHP Excel Export class* [online]. Vydáno: 2007-03-27 [cit. 2008-05-11]. Dostupný z WWW: <http://chumby.net/2007/03/27/php-excel-export-class/>.

[28] ERH-WEN, K.: *excelgen.class.php* [online]. Vydáno: 2002-07-18 [cit. 2008-05-11]. Dostupný z WWW: <http://www.koders.com/php/fid2163EF6648BF0A5A9D58837899B249F420EF45A5.aspx?s=excel#L8>.

[29] Advameg, Inc: *RFC 1945 - Hypertext Transfer Protocol -- HTTP/1.0* [online]. c2004 [cit. 2008-05-11]. Dostupný z WWW: <http://www.faqs.org/rfcs/rfc1945.html>.

[30] PLATHEY, O.: *FPDF Library – PDF generator* [online]. c2002-2006 [cit. 2008-05-11]. Dostupný z WWW: <http://www.fpdf.org/>.

[31] RingsWorld.com: *Cookdojo Home* [online]. c2005 [cit. 2008-05-11]. Dostupný z WWW: <http://scripts.ringsworld.com/organizers/cookdojo-home/>.

[32] SOUKUP, L.: *Generování dokumentů PDF pomocí PHP a fpdf.php* [online]. Vydáno: 2002-08-27 [cit. 2008-05-10]. Dostupný z WWW: <http://www.webguru.cz/clanky/view.php?id=91>.

[33] The Apache Software Foundation: *.htaccess files* [online]. c2008 [cit. 2008-05-20]. Dostupný z WWW: <http://httpd.apache.org/docs/1.3/howto/htaccess.html>.

[34] JANOVSKÝ, D.: *Soubor .htaccess* [online]. Vydáno: 2005-11-17 [cit. 2008-05-20]. Dostupný z WWW: <http://www.jakpsatweb.cz/server/htaccess.html>.

[35] JANOVSKÝ, D.: *Cookies* [online]. c2002-2007 [cit. 2008-05-20]. Dostupný z WWW: <http://www.jakpsatweb.cz/enc/cookies.html>.

Přílohy

1. Struktura databáze
2. Zdrojové kódy aplikace a export databáze (na přiloženém CD)

Prohlášení

Prohlašuji, že svou diplomovou práci na téma "Správa vývojové dokumentace přes WWW II" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.“

V Brně dne …… ……..

(podpis autora)

Poděkování

Děkuji vedoucímu diplomové práce z firmy Honeywell Ing. Radomíru Svobodovi Ph.D., za velmi užitečnou metodickou pomoc a cenné rady při zpracování diplomové práce.

V Brně dne …… ……..

(podpis autora)