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 I

ADMINISTRATION OF DEVELOPMENT DOCUMENTATION OVER WWW I

DIPLOMOVÁ PRÁCE
MASTER`S THESIS

AUTOR PRÁCE — BC. MIROSLAV KOLÁŘ
AUTHOR

VEDOUCÍ PRÁCE — ING. MIROSLAV BALÍK, PH.D.
SUPERVISOR

BRNO 2008

Anotace

Diplomová práce vychází ze zadání společnosti Honeywell na vytvoření jednotného systému pro ukládání a správu vývojové dokumentace. Navržený systém zavede jednotné úložiště spolu s přehledným zobrazováním dokumentů. Konkrétně se jedná o požadavky zákazníků, produkty, testy a běhy testů. Řešení je rozděleno na dvě části: A – Požadavky a Produkty a B – Testy, Běhy testů a Uživatelé, přičemž tato diplomová práce řeší část B. Vytvářejícími prostředky jsou jazyky HTML, PHP, JavaScript a databázový systém MySQL. V první fázi byla navržena struktury databáze přidělených částí, jednotlivých databázových tabulek a vazeb mezi nimi. Dále byla navržena struktura pro autorizační algoritmus a vytvořen jednotný koncept celé aplikace. Druhá fáze zahrnuje samotnou realizaci. Nejprve jsou vytvořeny podpůrné algoritmy pro obsluhu databáze pomocí vytvořených funkcí, jednotné zobrazování výsledků a autorizace přístupu uživatelů do aplikace, které je přikládán velký důraz. Dále je řešena knihovna pro jednotné zobrazování formulářových prvků vytvořená s použitím šablonovacího systému Smarty a dalších algoritmů, např. přikládání souborů k záznamům v SQL tabulkách. Důležitou součástí je realizace návrhu databáze. V rámci obou oddělení, Testy a Běhy testů, je vytvořeno uživatelské rozhraní, jednotlivé vazby mezi nimi, spolu s vazbami na ostatní struktury. Dle zadání jsou vytvořeny všechny základní programy spolu s podpůrnými prostředky. Jednotlivé zdrojové soubory jsou uloženy v přehledné struktuře, kterou je možné aplikovat na jakémkoli stroji. Výsledkem diplomové práce je tedy ucelená vývojová zpráva spolu s funkčními zdrojovými kódy.

Klíčová slova

Správa dokumentace
Datový sklad
PHP
MySQL
Autorizace
Formuláře

Abstract

Master's thesis is based on the assignment of company Honeywell to create integral system for saving and management of development documentation. Designed system will introduce data warehouse with transparent display of documents being connected with development of customer's requirements, products and tests, test runs. Solution consists of two parts: A – Requirements and Products and B – Tests, Test runs and Person, whereas Master's thesis deals with the part B. For system creation utilities of following programming languages were used HTML, PHP, JavaScript and database system MySQL. In the first phase was designed database structure (tables and relations between them). Except that was designed structure for authorization algorithm and integral concept of whole application was created. The second phase is realization. At first are created supporting algorithms. It is servicing database by functions, uniform displaying results from database and authorized access of users to the application. Further data library for integral display data forms, based on used template system Smarty and other libraries, f.e. attribution of the files to records in SQL tables. Important part was implementation of designed database system and related programs. In Tests and Test runs there is user interface established and individual relations between them. According to assignment all fundamental programs were designed together with support instruments. Individual source files are saved in transparent structure of folders, so this structure is possible to apply on any computer, where are installed programs to web hosting. The result of Master's thesis is then integrated progress report together with functional source codes.

Keywords

Management documentation
Data warehouse
PHP
MySQL
Authorization
Data forms

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/paní

Jméno a příjmení: Bc. Miroslav Kolář

Bytem: Obůrka 74, 678 01, Blansko - Obůrka

Narozen/a (datum a místo): 13.7.1984, Brno

(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 I

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 2

* 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: 26.5.2008

…… ……

Nabyvatel Autor

Obsah

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM KÓDŮ

1 ÚVOD

Následující text diplomové práce se bude zabývat realizací návrhu na řešení části aplikace pro správu vývojové dokumentace, která bude využívána pracovníky firmy Honeywell. Návrh byl obsažen v předcházejícím semestrálním projektu.

Dosud tito pracovníci užívají ke správě vývojové dokumentace komerčních programů, především Microsoft Word a Excel. Cílem této práce je tedy vytvořit jednotný sofistikovaný systém použitelný jak pro zaznamenávání požadavků zákazníků, tak pro následné zpracování vývojovým a testovacím oddělení firmy.

V diplomové práci jsem se zaměřil na realizaci struktury databáze pro správu vývojových testů a jejich běhů, naprogramování obslužných skriptů a propojení s druhou částí řešenou kolegou. Dále se budu zabývat knihovnou pro autorizovaný přístup do aplikace, knihovnou pro jednotné zobrazení formulářových prvků pro účel zadávání dat a rozhraním na pokročilé zobrazování výsledků dotazů do databáze, spolu s dalšími podpůrnými algoritmy. Praktické naprogramování aplikace bude vytvořeno za použití programovacích jazyků HTML, PHP, JavaScript a databáze MySQL.

Výsledkem pak bude funkční systém, který je schopen nasazení do užívání ihned po předání zadavateli.

2 FUNKCE APLIKACE

Tato kapitola popisuje funkci aplikace jako celku. Systém je rozdělen na dvě části. První, Requirement, obsluhuje požadavky od zákazníků i zaměstnanců firmy Honeywell. V podstatě se jedná o zadání kritérií pro vývoj určitého produktu. Druhá část, Test, do níž mají přístup pouze zaměstnanci, se zabývá vytvořením a následným provedením testu, který prakticky ověří funkci nově vyvinutého produktu a splnění požadavků na něj.

Zákazník figuruje jako zadavatel požadavku na vývoj produktu, přičemž má možnost vytvořit požadavek nový, nebo upravit kritéria u již existujícího. Dojde tím ke klonu, u kterého je povolena následná editace. Další zásahy do systému umožněny zákazníkovi nejsou. Ve většině případů budou požadavky ovšem zadávány zaměstnanci na základě písemných podkladů od zákazníků.

Zaměstnanci mají v aplikaci různá práva, tzn. mohou vytvářet, kopírovat a editovat jen určité oblasti, vzhledem k jejich pracovnímu zařazení. Nejvyšší práva v části Requirement má uživatel Design Analysis (DA), jehož úloha spočívá hlavně v tom, že schvaluje, nebo také přímo zadává, nové požadavky a rozhoduje o tom, zda budou dále zpracovány, nebo se stanou neaktivními. Nově vložený požadavek je nejprve neaktivní, DA je upozorněn na jeho existenci a provede případnou aktivaci. To znamená, že je odstartován proces vývoje. Dalším uživatelem je Design Engineer (DE), který je oprávněn požadavky pouze zadávat. Funkce DA a DE mohou být fyzicky přiděleny jedné osobě. Založit a schválit nové testy ke konkrétním požadavkům smí pouze uživatel Test Analysis (TA), jehož úloha je v rámci Test prakticky stejná, jako DA. Zadávat a editovat nové testy je oprávněno také uživateli Test Engineer (TE). Vytvářet testy mají právo také role DA a DE. Posledním krokem je samotná realizace vytvořeného testu a zhodnocení jeho výsledků. Vytvořit testovací proceduru může DA, DE, TA i TE. Schválení náleží pouze TE. K provedení procedury je oprávněna především uživatel Test Technician (TT). Jak již bylo řečeno výše, jednotliví uživatelé mohou být sloučeni do jedné osoby,což v praxi většinou bude. Jeden člověk bude schopen vytvořit a schválit požadavek, další potom v návaznosti vytvořit a provést jeho test.

Zásadním požadavkem od dohlížejících pracovníků firmy Honeywell byla možnost archivace všech kroků a pohybů v databázi. Tím bude docíleno naprosté

kontroly nad všemi zaměstnanci, kteří budou mít oprávnění pracovat se systémem. V praxi to znamená, že bude možné zpětně dohledat zadávání požadavků, jejich změn a především to, kdo tyto změny provedl. Stejně bude aplikace fungovat v části testů.

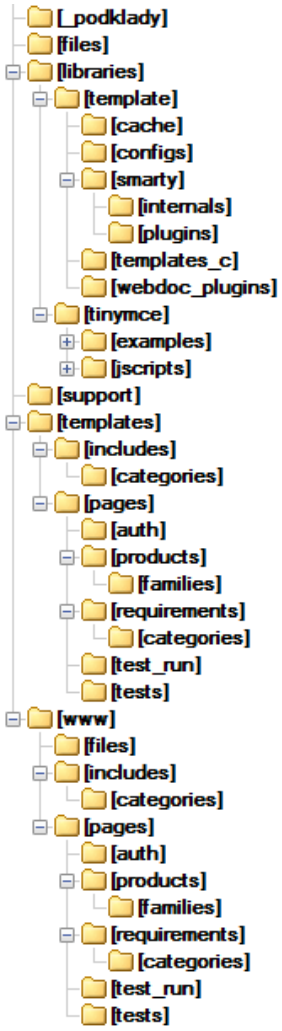
Výstupem ze systému budou tiskové sestavy v různých formátech – DOC, XLS a PDF, které budou podávat ucelené informace o struktuře požadavku, testu a jeho provedení. Sestavy budou přímo sloužit jako výsledné zprávy.

3 TECHNICKÉ PROSTŘEDKY

V rámci Semestrálního projektu jsem popisoval jako technické prostředky zejména prostředí, na kterém aplikace běží a programovací jazyky, pomocí kterých je vytvořena. V diplomové práci se soustředím na popis struktury uložení jednotlivých zdrojových a dalších souborů, a také nosného šablonovacího systému Smarty.

3.1 Adresářová struktura

Soubory, ze kterých je aplikace vytvořena, jsou koncepčně uspořádány do adresářů tak, aby při přenosu na jakýkoli jiný webový server zůstala funkčnost aplikace nezměněna. Při přenosu se v podstatě jedná o nakopírování souborů na danou stanici, import databázové struktury a nastavení údajů pro přihlášení k databázi v příslušném zdrojovém souboru.



Obr. 3.1: *Adresářová struktura aplikace*

Adresář `_podklady` obsahuje zmíněný export databázové struktury, který je potřeba importovat. To je možné na unixových či linuxových operačních systémech (nebo pomocí vzdálené správy) přes příkazovou řádku, na všech platformách pak databázovými manažery. Pro databázi MySQL je to např. PhpMyAdmin, pro PostgreSQL PgAdmin.

Adresář `files` je určen pro ukládání souborů, které je možné připojit k záznamům v tabulkách Requirement, Product, Test či Test Run. Má systémová práva pro zápis, aby bylo možno potřebné soubory na toto místo ukládat.

Jak již název napovídá, `libraries` obsahuje základní knihovny aplikace. Je zde uložen především šablonovací systém Smarty, který popíšu dále, zdrojové kódy použitého WYSIWYG formuláře a dále vytvořené knihovny pro potřeby aplikace.

V Smarty bych chtěl zmínit adresář `webdoc_plugins`, který obsahuje knihovny volané v rámci šablon, dodatečně vytvořené pro splnění zadání práce. Popsány jsou v kapitole 4.

Adresář `support` obsahuje pomocné programy, např. vytváření systémových hlášek, pouřitých pro usnadnění práce či provedení následných změn a nových funkcí v aplikaci. Případný další editor programů aplikace zde najde kompletní popis k funkcím, použitých v šablonách.

Následují dva nejvíce důležité adresáře. Prvním z nich je `templates`, který obsahuje právě šablony systému Smarty, jehož struktura dále pokračuje. A to `pages`, kde je přesně vytvořená struktura odkazů na šablony dle umístění uživatele v rámci aplikace. Směrování probíhá na šablony v adresáři `includes`, které zobrazují potřebné výstupy z PHP skriptů.

Druhým neopomenutelným adresářem je `www`, který obsahuje právě PHP skripty, které zajišťují funkci aplikace. Šablony jsou tedy zobrazovacím a PHP skripty funkčním elementem v rámci aplikace. Takovéto rozdělení vyplývá ze zadání práce – oddělení HTML kódu užitého v šablonách od vlastního programu, vytvořeného v jazyku PHP. Struktura adresáře je tedy stejná, jako v případě `templates`. Obsahuje nejprve odkazy na PHP skripty dle požadavků (kliknutí v rámci aplikace) uživatele, poté jsou v adresáři `includes` umístěny vlastní programy.

3.2 Šablonovací systém Smarty

Tuto část řeší ve své diplomové práci kolega, který provedl potřebná nastavení pro zabezpečení funkčnosti. Jelikož se ale jedná o kostru celé aplikace a kromě prvotního uvedení do provozu jsem pro svoje potřeby šablony konfiguroval a vytvářel sám, chtěl bych na tomto místě uvést co to vlastně Smarty je a jak se používá.

Smarty znamená chytrý, elegantní, pohotový. A přesně takový je i stejnojmenný systém šablon pro PHP, který umožňuje vkládat do HTML kódu speciální znaky a příkazy a oddělit tak aplikační logiku od prezentace dat. Viz. [4]. Je to jeden z nejlepších šablonovacích systémů v současné době, ovšem není univerzálně nejlepší pro všechna řešení. Například rychlost není vlastností, kterou by vynikal. Je ale velmi dobrý ve funkčnosti, rozšiřitelnosti, podpoře a dokumentaci (i v českém jazyce). Vyniká způsobem, jakým jsou šablony zpracovávány. Při prvním volání, tzn. spuštění skriptu, který pro svůj výstup šablonu využívá, jsou data převedena do podoby PHP skriptu,

který je následně spuštěn na serveru, výsledek je odeslán internetovému prohlížeči. Při dalším volání je poté spuštěna pouze převedená verze šablony, čímž se urychluje odezva serveru při opakovaném volání skriptů, což je vlastnost vytvářené aplikace.

Distribuce Smarty obsahuje kromě základních tříd také mnoho předdefinovaných uživatelských funkcí, které jsem využil sice velmi málo, ovšem co jsem využil ve značné míře, byla možnost si podobné funkce vytvořit.

Hlavní výhodou Smarty je tedy jednoduché oddělení aplikační a prezentační logiky. Pod aplikační logikou si představme programový kód, který se například stará o získání dat z databáze, výpočty, vytváření PHP session atd.. Z názvu je patrné, že tento kód by neměl mít nic společného s prezentací dat. Naopak prezentační logika se týká jen a pouze prezentování dat. Jedná se například o zobrazení záznamů z SQL tabulky. Nyní si položme otázku proč, pokud máme k dispozici šablony, prezentační logiku vyčleňovat z kódu HTML stránky, když je můžeme stejně efektivně a přehledně začlenit do šablony stránek. Smarty umožňuje zařadit do HTML kódu stránky příkazy a řídící struktury známé z PHP (`if`, `foreach`, `else`, atd.), které se o prezentační logiku postarají.

Instalací a zavedení řeší ve své diplomové práci kolega. Moje diplomová práce na jeho řešení navazuje, a proto se dále věnuji používání šablon. Základem je znalost syntaxe šablon, které jsou v zásadě standardní HTML dokumenty, obsahující základní HTML tagy. Pro použití Smarty slouží oddělovače, jejichž obsah je při zpracování nahrazen PHP kódem. V základu jsou pro tento účel používány složené závorky, které jsou definovány v zakládající třídě. Je důležité říci, že se tyto oddělovače nesmí v samotném HTML kódu vyskytovat, protože by byla snaha je interpretovat. V některých případech je ovšem nutné složené závorky použít, např. v JavaScriptu, v tomto případě lze použít příkaz `{literal}`…`text`…`{/literal}`, který překladači říká, že text takto označený nemá interpretovat. Proměnné jsou v šablonách označovány stejně jako v PHP skriptech, čili jestliže je třeba vložit obsah proměnné, vloží se do šablony příkaz `{$promenna}`. Stejně jednoduše lze vkládat i hodnoty polí `{$pole.prvek}` a objektů `{$objekt->vlastnost}`. Dále je definována proměnná `$smarty`, která je v podstatě asociativním polem, obsahující několik důležitých hodnot, např. proměnné předané skriptu v globálních polích (`$_POST`, `$_SESSION`, `$_REQUEST` atd.), přičemž přístup je stejný, jako v případě obyčejného pole. Klíčovou metodou objektu `$smarty` je předávání hodnoty proměnné použité

v PHP skriptu šabloně - `$smarty->assign( "pole", array( "první", "druhá", "třetí" ) );` . Mimo standardních HTML lze vkládat komentáře ve tvaru `{* komentar *}`. [5]

Kód 3.1: Ukázková Smarty šablona [6]

```
<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<title>Vítejte!</title>
</head>
<body>
 <p>Vítáme uživatele {$jmeno}</p>
</body>
</html>
```

Kód 3.2: Ukázkový PHP skript k šabloně [6]

```
<?php
require_once 'moje-smarty.php';
$smarty = new MojeSmarty;

$jmeno = array_key_exists('jmeno',$_COOKIE)? $_COOKIE['jmeno']:
'návštěvník';

$smarty->assign('jmeno',$jmeno);
$smarty->display('index.tpl');
?>
```

Kód 3.3: Přeložená verze Smarty [6]

```
<?php /* Smarty version 2.6.14, created on 2007-05-07 22:36:36

         compiled from index.tpl */ ?>

<html>
<head>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<title>Vítejte!</title>
</head>
<body>
 <p>Vítáme uživatele <?php echo $this->_tpl_vars['jmeno']; ?>
</p>
</body>
</html>
```

V následující kapitole je popsáno vytváření Smarty funkcí a jejich používání v rámci šablon.

4 REALIZACE ALGORITMŮ

Před začátkem budování databázových tabulek a vazeb mezi nimi jsem vytvořil algoritmy, které usnadňují další práci. Jedná se v podstatě o funkce, které jsou naprogramovány v rámci knihovny obecnými parametry, následně potom volány v jednotlivých skriptech s parametry konkrétními. Jde o obsluhu databáze, autorizaci uživatelů, jednotné zobrazování formulářových prvků pro zadávání a editaci dat a přehledné zobrazení výsledků v celé aplikaci. Kromě těchto základních algoritmů, které jsou součástí zadání práce, uvádím na konci kapitoly další funkce, které používám v jiných úlohách.

4.1 Obsluha databáze

Provádí se vkládáním SQL příkazů do zdrojového kódu jazyka PHP.

V prvé řadě je nutné se k databázi připojit, protože běží jako systémový proces. Je nutné znát název hostitele, může se zadávat buď jménem (např. `mysql.domena.cz`), nebo IP adresou. V našem případě databáze běží na stejném stroji, jako jsou umístěny skripty, proto lze použít `localhost` či `127.0.0.1`. Dalším atributem je port, na kterém databáze naslouchá. Standardně to je port 3306, takže pokud není v konfiguraci řečeno jinak, může se při přihlašování vynechat, je doplněn automaticky. Důležitými atributy, bez nichž se připojení nezdaří a které jsou často zadány nesprávně, jsou název databáze, uživatelské jméno a heslo. Na serveru existuje většinou několik vzájemně nesouvisejících databází, každá obsahuje buď žádnou, nebo více tabulek. Z hlediska připojení je přístupná vždy právě jedna databáze. Uživatelské jméno může být stejné, jako název databáze. Toto jméno však nijak nesouvisí s přihlašovacím jménem k serveru. Heslo plní základní bezpečnostní pravidla. V PHP se ke zvolené databázi MySQL připojuje pomocí funkce `mysql_connect`. Jejími parametry jsou název hostitele, uživatelské jméno a heslo. Po připojení je třeba ještě vybrat databázi, což se děje příkazem `mysql_select_db`. Pokud je přihlášení úspěšné, budou se všechny SQL příkazy v rámci PHP provádět na tomto spojení, které zůstane otevřené až do doby, než je zavolána funkce `mysql_close`, nebo do ukončení běhu skriptu. [3]

Připojení k MySQL databázi:

```
mysql_connect("localhost", "login", "password");
mysql_select_db("my_db");
```

Kromě krátkodobého připojení, jehož funkce byla vysvětlena výše, lze vytvořit trvalé spojení s databází. K tomu slouží příkaz `mysql_pconnect`. Spojení tedy není uzavřeno při ukončení skriptu, ale zůstává otevřené. V prostředí vícevláknových serverů je ale čas potřebný pro vykonání funkce `mysql_connect` zanedbatelný, proto i já budu používat vytvoření nového spojení pro každý skript.

Bylo by velmi neefektivní psát výše uvedené příkazy pro spojení s databází do každého skriptu zvlášť. Při jakékoli změně údajů by bylo potřeba editovat všechny skripty, což by při rozsahu tohoto projektu bylo velmi časově náročné. V uvedených příkazech také nejsou ošetřeny chybové hlášky. Dle mých zkušeností je nejvhodnější vytvořit knihovnu, která obsluhuje všechny skripty. Po dohodě s kolegou jsme vytvořili knihovnu `database-mysql.php`, která obsahuje všechny funkce týkají se databáze. Výpis mnou vytvořených funkcí je v Příloze 1.

Kód 4.1: Připojení a odpojení od MySQL databáze, zdrojové kódy funkcí v knihovně `database-mysql.php`

```
function database_connect()
{
  if (!mysql_connect("localhost","honeywell","password"))
  return false;
  if (!mysql_select_db("honeywell")) return false;
  return true;
}

function database_close()
{
  return mysql_close();
}
```

Uvedené funkce jsou volány ze souboru `index.php`, tzn. z kořenového skriptu aplikace. To zaručuje, že budou spuštěny s každou knihovnou či vnořeným skriptem. Pokud se připojení k databázi nezdaří, je vše zastaveno a je vypsána chybová hláška „`Sorry, fail connect to database.`“.

Tímto je dokončeno spojení s databází. Dalšími procedurami, které je nutné definovat pro následnou práci definovat v knihovně, je výběr (SELECT), vkládání (INSERT), editace (UPDATE) a mazání (DELETE) záznamů v tabulkách. [2] Zde se

omezuji pouze na obecné definice uvedených příkazů. Funkce použité pro jednotlivé skripty jsou uvedeny vždy při popisu daného problému. Je třeba zdůraznit, že stejně jako SQL příkaz `mysql_connect`, je níže uvedené struktury nutné uvodit PHP funkcí `mysql_query`.

Kód 4.2: Funkce pro vykonání SQL příkazu (zdrojový kódy z knihovny `database-mysql.php`)

```
function database_query($query)
{
  return mysql_query($query);
}
```

Struktura příkazu pro vložení nového záznamu do tabulky:

```
INSERT INTO [název_ tabulky] (sloupec, ...) VALUES (hodnota, ...)
```

Příklad vložení nového záznamu do tabulky:

```
INSERT INTO pokus (pole1, pole3) VALUES (“hodnota”, 1)
```

Struktura příkazu pro editaci záznamu v tabulce:

```
UPDATE [název_tabulky] SET sloupec=hodnota [,jiný sloupec=hodnota...]
WHERE [podmínka]
```

Příklad editace záznamu v tabulce:

```
UPDATE pokus SET pole1 = ”hodnota2”, pole3 = 0
```

Struktura příkazu pro mazání záznamu v tabulce:

```
DELETE FROM [název_tabulky] WHERE [podmínka]
```

Příklad smazání záznamu v tabulce:

```
DELETE FROM  pokus WHERE  id = 1
```

Struktura příkazu pro výběr záznamů z tabulky:

```
SELECT [seznam polí] FROM [název_tabulky]
WHERE [podmínka] GROUP BY [seskupení] ORDER BY [seřazení]
```

Příklad výběru záznamů z tabulky:

```
SELECT * FROM pokus WHERE pole1=hodnota ORDER BY id
```

Nelze opomenout vlastní založení tabulky, což není součástí žádné funkce ani knihovny, protože probíhá jednorázově. Pokud by nebyla vytvořena daná tabulka, není možno vykonávat výše uvedené příkazy.

Tabulku si lze představit jako list aplikace Microsoft Excel, nebo OpenOffice OOCalc. Každá tabulka má určitý počet sloupců a řádků. Každý řádek obsahuje určitou skupinu dat. Pokud dochází k přidávání řádků, znamená to, že se mění data v tabulce, jsou-li přidávány sloupce, znamená to, se mění její struktura. Do polí tabulky se dají ukládat řetězce (jedno či vícepísmenné), logické hodnoty, datumy a další různě přesná čísla. Pokud se stane, že je do pole snaha zapsat hodnotu jiného datového typu, většinou databáze skončí chybou. Je důležité říci, že ne všechna pole v tabulce musí obsahovat nějaká data. Pokud není nic zapsáno, databáze automaticky dolní hodnotu NULL, která není rovna ničemu - nule, prázdnému řetězci ani logické hodnotě `false`. Při zakládání tabulky lze ovšem striktně nařídit, že dané pole nesmí být nulové (děje se přiřazením atributu NOT NULL). Potom při nezadání hodnoty, vypíše operace chybu. Před samotnou definicí nové tabulky je třeba vědět, kolik bude mít sloupců, co bude v jednotlivých polích uloženo, jakého mají být datového typu a jak se mají jmenovat. Používáme datové typy INTEGER a TINYINT (číselný datový typ), VARCHAR (textový datový typ s omezením délky znaků, která je uváděna v kulaté závorce), DATETIME (formát představující datum a čas), BOOL (nabývá hodnot 0, nebo 1, používán k určení programového `True` a `False`) a TEXT (textový datový typ). Pojmem „primární klíč“ se rozumí hodnota, která jednoznačně identifikuje každý záznam v databázové tabulce. Žádná hodnota, která je součástí primárního klíče, nesmí obsahovat hodnotu NULL. Každá tabulka musí mít definovaný právě jeden primární klíč. [2]

Kód 4.3: Příklad vytvoření tabulky

```
CREATE TABLE pokus (
id INTEGER NOT NULL AUTO_INCREMENT , // AUTO_INCREMENT automaticky
zvysuje hodnotu id o 1 pri vkladani noveho zaznamu
pole1 VARCHAR(255 ) NOT NULL ,
pole2 VARCHAR(2) ,
pole3 BOOL NOT NULL,
PRIMARY KEY (id)
)
```

4.2 Přehledné zobrazení výsledků

V předchozí kapitole bylo vysvětleno, jak pracovat s databází. Tato kapitola se bude zabývat tím, jak pomocí jazyka PHP vytvořit SQL příkazy a jak přehledně zobrazovat data získaná z databáze pomocí knihovny k tomuto určené.

Jak bylo uvedeno, k vnoření výše uvedených SQL příkazů do zdrojového kódu jazyka PHP slouží příkaz `mysql_query`. Při akci SELECT ukládám výsledek do proměnné `$result`, která je datového typu `array` (pole) a dále s ním pracuji. Pro získání konkrétních dat z `$result` a jejich zapsání do pole, ze kterého lze pohodlně číst, slouží příkaz `mysql_fetch_array`. Jednotlivé SQL dotazy do databáze, určené pro jeden či více skriptů, jsou uloženy v knihovně `database-mysql.php` formou funkcí, které jsou volány právě z PHP programů. Komentáře k nim uvádím u konkrétních případů jejich použití.

Kód 4.4: Funkce pro vytvoření pole výsledků dotazu do databáze (zdrojový kódy z knihovny `database-mysql.php`)

```
function database_fetch_array($result)
{
  return mysql_fetch_array($result);
}
```

Výsledky ukládám do proměnné `$list`, která je datového typu pole. Konkrétní informaci lze potom získat např. z proměnné `$list[“informace”]`. Před zahájením práce s touto proměnnou je dobré otestovat, zda obsahuje vůbec nějaká data, tzn. SQL dotaz vrátil nějaké informace. Pokud by nebyla žádná data k dispozici, je vypsána chybová hláška `„List is empty.“`.

Jakmile je k dispozici proměnná `$list`, je třeba data obsažená v ní přehledně zobrazit, aby byla bez problémů čitelná uživateli aplikace. Za účelem snadného zapracování uživatelských požadavků na grafické uspořádání zobrazení jsem navrhl univerzální funkci, která po převzetí parametrů vytvoří požadovaný výstup. Forma výstupu je HTML tabulka, která má svoje záhlaví, což jsou nadpisy jednotlivých sloupců a vlastní tělo tabulky. Tuto funkci jsem pojmenoval `view_result` a je možné ji zavolat ve Smarty šabloně následujícím způsobem.

Struktura funkce pro zobrazení výsledku dotazu do databáze:

```
{view_result title=[zahlavi_tabulky] list=[sql_dotaz]
values=[prvky_dotazu] actions=[akce_s_prvky] parametr=[odeslat_s_akci]
role=[role] color=[barva_aktualniho] terminate=[terminate]}
```

Význam použitých proměnných:

[zahlavi_tabulky] - pole názvů jednotlivých sloupců v záhlaví vykreslované HTML tabulky

[sql_dotaz] - pole obsahující výsledek SQL dotazu, jedná se o proměnnou $list

[prvky_dotazu] - které prvky z dotazu (proměnné $list) se mají zobrazit, tzn. které sloupce z SQL tabulky

[akce_s_prvky] - jedná se o vykresleni názvu akce s daty (např. Edit), zobrazující se na konci řádku tabulky, společně s vytvořením odkazu pod tímto názvem
- je třeba vytvořit dvojrozměrné pole, kde je přiřazeno názvu akce jméno skriptu, který ji vykoná

[odeslat_s_akci] - jedna se o pole názvů sloupců z tabulky, jejichž hodnota má být poslána s odkazem pro vykonání akce
- hodnota bude poslána pod proměnnou, která je názvem příslušného sloupce z SQL tabulky

[role] - pole uživatelských rolí, podle kterého jsou zobrazovány akce se záznamy
- jedná se o nepovinný parametr, pokud není uveden, jsou zobrazeny všechny akce pro všechny uživatele
- musí mít stejný rozměr jako pole [akce_s_prvky]

[barva_aktualniho] - název sloupce v databázi, jehož hodnota má byt porovnávána s hodnotou aktuálně zobrazovaného záznamu

- pokud se shoduje, provede podbarveni záznamu, tzn. ukazuje aktuální záznam

[terminate] - týká se části Test a Test Run, při zadání tohoto přepínače kontroluje, zda není záznam ve stavu terminated (odstraněn), pokud je, přeškrtne tento záznam při zobrazení

Je zřejmé, že před zavoláním funkce `view_result` je třeba v obslužném PHP skriptu vytvořit požadovaná pole, která funkci naplní.

Kód 4.5: Příklad vytvoření polí pro naplnění funkce pro zobrazení výsledků dotazu do databáze

```
$list=database_get_list($rows); //vrati seznam pozadavku
if (count($list)==0) // jsou nejake hodnoty k zobrazeni
$webdoc["message"][]=database_get_message("list_empty"); // funkce,
ktera provede vypsani prislusne hlasky

// prirazovani jednotlivých promennych do sablony
$template->template_assign("title",array("Name","Description",
"Actions"));
$template->template_assign("values",array("name","description"));
$template->template_assign("actions",array("View"=>"view",
"Edit"=>"edit","Delete"=>"delete","History"=>"history","Copy"=>"copy"));
$template->template_assign("role",array("5","1","1","5","5"));
$template->template_assign("parametr", array("sequence_id"));
$template->template_assign("list",$list);
```

Příklad volání funkce `view_result`:

```
{view_result title=$title list=$list values=$values actions=$actions
parametr=$parametr role=$role color="sequence_id"}
```

Samotná knihovna `view_result` je naprogramována v jazyku PHP a funguje tak, že nejprve načte vstupy, zadané ze Smarty šablony. Poté zkontroluje, zda jsou zadány požadované povinné parametry, což jsou `title`, `list` a `values`. Pokud ne, je vyvolána Smarty chybová hláška. Jestliže je vše v pořádku, je vytvořena výstupní proměnná, do které je zapsán HTML příkaz pro vytvoření tabulky. Následně je vytvořen první řádek v tabulce, což je záhlaví, do kterého je vypsáno příslušné pole, formou přidávání HTML tagů a proměnných do výstupu. Dále jsou vytvářeny další řádky tak, jak jsou data uložena v proměnné `$list`. Pokud je zadán parametr `actions`, je na konec řádku přidána další buňka tabulky, do které je vepsán obsah pole `$actions` a jsou vytvořeny příslušné odkazy. V případě požadavku je před

zobrazením odkazu kontrolována role uživatele, zda má potřebná práva. V případě zadání parametru je obsah odkazů doplněn o proměnné, které se s ním odešlou. Závěrem je do výstupní proměnné přidáno ukončení HTML tabulky. Jelikož je zdrojový kód knihovny příliš dlouhý, je vypsán v Příloze 1.

Knihovna `view_result` vytváří univerzální nástroj. V případě požadavku na změnu grafického uspořádání, nebo jiného doplnění výsledku zobrazení, je tato změna provedena pouze v rámci knihovny, v ostatních skriptech není nutný žádný zásah či úprava.

4.3 Autorizace přístupu uživatelů do aplikace

Celá aplikace je přístupná pouze pro určitý okruh uživatelů, žádnou její část nelze navštívit neautorizovaně, proto přikládám výběru správného algoritmu, spolu s jeho vytvořením, velký význam. Tato kapitola se nejprve věnuje možnostem, jak autorizaci řešit. V závěru je popsáno řešení, které jsem si vybral, spolu s vyhotovením a implementací do aplikace.

4.3.1 Ověřování pomocí WWW-Authenticate

Principem tohoto algoritmu je přinutit server, aby požádal návštěvníka stránky o autorizaci. To lze provést předáním jedné z hlaviček protokolu HTTP.

Příklad předání hlavičky protokolu HTTP:

```
Header("WWW-Authenticate: Basic realm=\"oblast_hesel\"");
Header("HTTP/1.0 401 Unauthorized");
```

Jakmile dorazí hlavička na server, neodešle klientskému prohlížeči stránku, ale právě žádost o autorizaci – objeví se klasické okno pro zadání jména a hesla. Ty jsou po zadání odeslány zpět na server. Pro další použití jsou uloženy v asociativním poli $_SERVER. Ověření je ovšem nutné naprogramovat, a to porovnáním se správnými údaji (např. daty v databázi).

Výhodou je, že po úspěšném ověření jsou přihlašovací informace k dispozici i dalším skriptům, a to až do odhlášení nebo uzavření okna prohlížeče. U dalších skriptů je ale nutné ošetřit stav, kdyby nebyla autorizace ještě provedena, což je možné realizovat testem, zda existuje neprázdné pole $_SERVER.

Podstatnou nevýhodou tohoto algoritmu, a také důvodem, proč není použit v mojí práci, je fakt, že kromě přihlašovacího jména a hesla jím nelze elegantně přenášet jiné informace, jako např. identifikátor uživatele, čas přihlášení apod. Přenos dalších proměnných lze realizovat pouze přidáním jejich názvů a hodnot do URL stránky a neustále je mezi sebou posílat. Tím hrozí neoprávněný přístup, když klient jednoduše změní hodnoty v URL. Další nevýhodou je velká náročnost při psaní kódu, nehodí se pro příliš dlouhá data. [3] Autorizační údaje jsou odesílány při každé žádosti prohlížeče o stránku. Při nešifrovaném spojením mohou být tedy kdykoli odposlechnuta.

4.3.2 Ověřování pomocí Cookies

Hlavní vlastností `cookies` je, že jsou díky nim ukládány libovolné informace na počítači klienta. Pomocí PHP je prohlížeči předána zpráva, aby se stránkou odeslal také jednu, nebo více `cookies`, které jsou zasílány v hlavičce. Když se potom klient vrátí na stejnou stránku, budou mu opět zaslány. Platnost jde časově omezit, což má výhodu v případě ukládání přihlašovacích údajů. Protože jsou ukládány na volně přístupné místo na disku, předejde se tím jejich zneužití druhou osobou, např. při nekorektním odhlášení.

PHP příkaz pro zaslání cookies s časovým omezením (10 min):

```
SetCookie ("nazev_cookie", "hodnota", time()+10*60);
```

Čtení `cookies` je velmi jednoduché, v PHP jsou k dispozici pomocí asociativního pole `$_COOKIE`. Jsou tedy přístupné pro všechny skripty.

PHP příkaz pro čtení cookies, jejich uložení do proměnné:

```
$promenna = $_COOKIE[“nazev_cookie”];
```

Jakmile je hodnota uložena v proměnné, je připravena k porovnání v případě, že v ní budou umístěny přihlašovací údaje.

Přesto, že je použití `cookies` mnohem snazší, než předávání parametru pomocí URL, má jejich nasazení jisté problémy. Největším z nich je, že uživatel může `cookies` ručně smazat a změnit. Může je také číst, takže informace mohou být lehce zjištěny. Ukládají se do obyčejných textových souborů a na serveru neexistuje kontrola,

zda přijal stejná data, jako vyslal. Dalším úskalím je, že uživatel může jejich ukládání zakázat. Lze sice vytvořit test, který upozorní klienta, že je třeba `cookies` zapnout, pokud to ovšem neudělá, aplikace mu zůstane nepřístupná. Z výše uvedených důvodů nebudu `cookies` ve své práci používat. [1]

4.3.3 Ověřování pomocí Sessions

Z předchozího textu je patrné, že nejdůležitější v oblasti autorizace uživatelů není, jakým způsobem provést porovnání zadání správných údajů, ale jak předávat tyto údaje v rámci celé aplikace. Při použití `sessions` se jedná o způsob přenosu informací mezi stránkami, který má z uvedených řešení nejméně omezení. Hlavní předností je, že zatímco `cookies` jsou generovány na serveru a posílány prohlížeči, `sessions` fungují přímo na serveru, bez posílání dat sem a tam. Data jsou uložena mimo kořenový adresář serveru. Celý mechanismus funguje tak, že jakmile PHP obdrží příkaz k započetí `sessions`, zajistí nejprve, zda již neběží. Pokud ne, vytvoří ji, pokud ano, tak se k ní připojí. PHP přiděluje `sessions` identifikátor a vyhradí někde místo pro ukládání jejích proměnných. Od tohoto místa lze u jakékoli proměnné zvolit, že bude `session` a tím bude zpřístupněna všem skriptům. Lze ji kdykoli ukončit, a pokud není ukončena v rámci programu, stane se tak při uzavření prohlížeče. Je třeba zmínit, jak server jednotlivé prohlížeče (klienty) odlišuje. Použije buď `cookies`, nebo URL, kde posílá identifikátor `sessions`. Zneužití identifikátoru nehrozí, protože i při jeho získání druhou osobou nelze zjistit obsah na serveru. Registrace `sessions` i jejích uložení do proměnné funguje podobně jako v případě `cookies`, pomocí globálního asociativního pole `$_SESSION`. [1]

Příklad zápisu informace do session a následný výpis do proměnné:

```
session_start();
session_register("promenna");
$_SESSION["promenna"]="hodnota";
$vypis = $_SESSION["promenna"];
```

4.3.4 Vlastní realizace ověřování

Domnívám se, že právě `sessions` představují nejlepší způsob, jak předávat informace mezi skripty, proto je používám ve své práci. Abych dosáhl co nejvyššího stupně zabezpečení aplikace, použil jsem některé další prvky.

Je třeba dbát na bezpečnost přenosu dat mezi uživatelem (prohlížečem) a serverem. Jakmile budou data na serveru, jsou zde v bezpečí. Proto jsou všechny skripty přístupné pouze přes protokol HTTPS, který je nadstavbou protokolu HTTP. Poskytuje ochranu před odposloucháváním a podvržením dat, protože nejsou posílána v běžném textu, ale šifrována pomocí SSL. Ustavení spojení SSL funguje na principu asymetrické šifry, kdy každá z komunikujících stran má dvojici šifrovacích klíčů. Veřejný klíč je možné vystavit, a pokud tímto klíčem kdokoliv zašifruje data, je zajištěno, že ji bude moci rozšifrovat jen majitel použitého veřejného klíče svým soukromým klíčem. Protokol HTTPS není implementován přímo v aplikaci, ale na serveru, na kterém je aplikace hostována. Při přenosu je třeba vytvořit toto bezpečné spojení správnou konfigurací webového serveru Apache.

Dalším bezpečnostním prvkem, který jsem zahrnul, je přihlašovací heslo ne ve formě obyčejného textu, ale zašifrovaně. Jako šifrovací algoritmus lze použít buď `MD5`, nebo `SHA1`. `MD5` vytváří kontrolní součet o velikosti 128 bitů s otiskem, tzv. `hash`. Prosadil se do mnoha aplikací, především pro ukládání hesel, a také pro kontrolu integrity souborů.

Implementace `MD5` do jazyka PHP:

```
$pokus_md5 = md5(pokus);
- tím se provede konverze slova pokus, výsledkem je tento hash:
fde27fdce626407e9ad040b7bb017882
```

Jelikož tento algoritmus vznikl již v roce 1991, byly v posledních letech nalezeny chyby, které jej poslaly do kategorie nebezpečných algoritmů. Především proto, že byl objeven zpětný algoritmus, kterým je možné vysledovat přibližnou množinu zpětných znaků, což má za následek malou časovou náročnost prolomení i při použití dnešních klasických PC. Největší je tedy riziko při použití jednoduchých hesel. Samozřejmě je velmi jednoduchá obrana – použít dostatečně dlouhé a složité heslo. Ovšem s prudkým růstem výkonu dnešní techniky je pravděpodobné, že s postupem času by i tato hesla mohla být prolomena. Proto ve své práci použiji algoritmus `SHA1`, který funguje podobným způsobem, jako `MD5`. Nebyl dosud prolomen, neexistuje žádný zpětný algoritmus. Podpora v PHP také nechybí, zápis je identický, jako v případě `MD5`.

Implementace SHA1 do jazyka PHP:

```
$pokus_sha1 = sha1(pokus);
- tím se provede konverze slova pokus, výsledkem je tento hash:
f14213784ef7468abe3a4325b24ea8ead61d6351
```

Dále popisuji funkci samotné autorizace - porovnání zadaných přihlašovacích údajů.

Úvodní strana aplikace načte skript `login.php`, který obsahuje formulář, požadující po klientovi zadání jeho přihlašovacího jména a hesla. Jak je popsáno v kapitole 5, přihlašovací údaje jsou umístěny v SQL tabulce, jejíž název je `Person`. Po stisknutí tlačítka pro odeslání údajů je opětovně spuštěn skript `login.php`, který díky testu na stisknutí odesílacího tlačítka začne nejprve kontrolovat, zda byly zadány vůbec nějaké přihlašovací údaje (jestli nebyl formulář odeslán prázdný, nebo s jedním nevyplněným polem). Pokud je registrován tento nedostatek, je vyvolána chybová hláška „`Field Login must be filled`“ v případě, že není zadáno přihlašovací jméno. Analogicky je vyvolána hláška v případě hesla. Jestliže jsou data ve formuláři korektní, je spuštěna funkce `database_get_auth` z knihovny `database-mysql.php`.

Ta nejprve převezme data z formulářů, vytvoří ze zadaného hesla `SHA1` hash a zašle do databáze dotaz, zda existuje k zadanému přihlašovacímu jménu vytvořený hash ze zadaného hesla. Dále také kontroluje, zda je přihlašovaný uživatel aktivní, tzn. databázové pole `active` není rovno 0. Pokud je výsledek tohoto dotazu roven jedné, tzn. existuje právě jeden takový uživatel, jsou vytvořeny PHP `session auth` a `role_id`, které reprezentují identifikátor uživatele a jeho práva. Následně je otevřena hlavní strana aplikace. Jestliže je výsledek dotazu jiný, je zobrazena chybová hláška „`Sorry, your login wasn`t successful. Bad Name or Password. Or you are not active.`“ a funkce vrátí hodnotu `false`. Dokud není uživatel autorizován, není mu zobrazeno cokoli jiného, než-li přihlašovací skript. Pokud ručně zadá URL jakéhokoli skriptu v aplikaci, je automaticky opět načtena stránka s formulářem pro přihlášení. [7]

Kód 4.6: Funkce `database_get_auth`

```
function database_get_auth($name,$pass)
{
  global $template, $message, $webdoc;
  $pass_sha1 = sha1($pass); // prevedeni zadaneho hesla na hash,
```

```
algoritmus SHA1
  $result=database_query("SELECT * FROM person WHERE name LIKE '$name'
  AND pass LIKE '$pass_sha1' AND active NOT LIKE '0'");
  if (database_num_rows($result)==1) //zaznam existuje v DB
  {
    $result_db = database_fetch_array($result);

    // vytvoreni PHP session
    $_SESSION["$webdoc[main_table]_auth"]=$result_db["person_id"];
    $_SESSION["$webdoc[main_table]_role_id"]=$result_db["role_id"];
    $url="./"; // presmerovani na main-page
    header("location: $url");
  }
  else // porovnani se nezdarilo - vypsani chybove hlasky
  {
    $webdoc["message"][]=database_get_message("bad_auth");
    return false;
  }
}
```

Na tomto místě je uživatel ověřen a korektně přihlášen do systému. Může se volně pohybovat po aplikaci, samozřejmě v rámci práv, která mu byla přidělena.

Každý skript je opatřen funkcí, která bude kontrolovat, zda je spouštěn autorizovaným uživatelem, a také zda má tento uživatel k zobrazení práva. Tuto funkci jsem nazval `auth` a je spouštěna na začátku všech programů. Má jediný vstupní parametr `person_role`, který musí být v rámci každého skriptu ručně nastaven. Jedná se o práva uživatele. Samotná funkce je velmi jednoduchá, pouze kontroluje, zda je nastavena příslušná globální proměnná, která reprezentuje PHP `session` s identifikátorem uživatele. Pokud ano, je ještě poslán dotaz do databáze za účelem získání práv a informace o aktivnosti kvůli zobrazení. Následný obsah stránky je potom zpřístupněn pouze osobě, která má práva vyšší, nebo rovna zadané hodnotě a je zároveň aktivní. Pokud není nastavena PHP `session`, je zobrazen přihlašovací formulář. Jestliže nemá uživatel práva k zobrazení, je vypsána chybová hláška „`Sorry, but you have not access permission or you are not active.`“. Těchto dvou chybových stavů je dosaženo pouze v případě, že je zadána přímá URL ke skriptům. Navigace aplikace je vytvořena tak, aby se nezobrazovaly odkazy na stránky, které nejsou určeny pro konkrétní osobu. Šablona Smarty pro vykreslování menu totiž přejímá parametr `person_role` a dle něj zobrazuje odkazy, funguje stejně jako `auth`.

Kód 4.7: Funkce `database_get_auth`

```
function auth($person_role)
{
  global $webdoc, $template, $message;
  if (!isset($webdoc["auth"])) // kontrola, zda je uzivatel prihlasen
  {
    $url="./?subpage1=auth&subpage2=login"; // neni, presmerovani na
    login page
    header("location: $url");
    $display_page=false; // nezobraz stranku
  }
  else
  {
    $person_id = $webdoc["auth"]; // ulozeni informaci do globalniho
    pole
    $role_id = $webdoc["role_id"];
    $template->template_assign("no_login",1);
    $result=database_query("SELECT * FROM person WHERE person_id LIKE
    '$person_id'"); // ziskani promenne role_id
    $result_db = database_fetch_array($result);
    $active=$result_db["active"]; // je uzivatel aktivni?

    if ($role_id<=$person_role && $active) $display_page=true; //
    testuje, jestli je uzivatel na urovni, nebo nad urovni zadanych
    prav, dale testuje, zda je uzivatel aktivni, pokud ano, zobrazi
    stranku
    else $display_page=false;

  }
  return $display_page;
}
```

Z výše uvedeného kódu je zřejmé, že funkce `auth` vrací parametr `display_page`. Jedná se o proměnnou, která je určena pro šablonovací systém Smarty. Na začátku každé šablony je testována tato proměnná, pokud není `false`, dojde k zobrazení. V opačném případě není zobrazeno nic a je vypsána pouze chybová hláška. Neautorizovaný, nebo neaktivní uživatel tedy nevidí obsah příslušné stránky.

V rámci této části aplikace jsem vytvořil skript umožňující roli DA vkládání nových uživatelů, jejich přehledný výpis s možností editace a mazání – `adduser`. Jestliže dojde k jeho zavolání přímo z menu, je zobrazen prázdný formulář se všemi potřebnými prvky ke kompletnímu naplnění databáze. Po odeslání je provedeno několik kontrol. Zda jsou všechna pole vyplněna, porovnávají se zadaná hesla pro případ překlepu. Jelikož není přípustné, aby v systému byly dva uživatelé se stejným login, je kontrolováno pomocí funkce `database_get_auth_select`, zda zadávané přihlašovací jméno již není jednou v databázi. Jako poslední je spuštěna funkce `is_right_email`, která ověří formát zadaného e-mailu. Musí obsahovat typické

znaky pro e-mailovou adresu, tzn. zavináč a tečku. Pokud není vyhověno některé z podmínek, je opět zobrazena příslušná chybová hláška. Naopak v případě jejich splnění dochází k volání funkce `database_save_record_auth`. Ta nejprve převede vložené heslo na hash algoritmu `SHA1`, a poté vykoná SQL příkaz, kdy jsou všechna data z formuláře vložena do databáze. Následně otevře stránku se seznamem uživatelů – `list_auth`, ve které je vypsána hláška „`New user was successfuly added.`“. V jejím rámci lze dále zadané uživatele editovat či mazat.

Editace funguje podobně jako vkládání. Je opět zobrazen skript `adduser`, ovšem s parametrem `edit`, díky kterému na základě zaslaného identifikátoru uživatele (`person_id`) ze skriptu `list_auth` vytvoří funkce `database_get_auth_select` pole, které je postupně zobrazeno ve všech formulářových polích. Pole určené pro zadávání hesla není z bezpečnostních důvodů vyplněno. V tuto chvíli může oprávněná osoba provádět změny v údajích konkrétního uživatele systému. Při stisku tlačítka pro vykonání editace jsou opět prováděny uvedené kontroly jako v případě nového založení. Jediná odlišnost je v případě hesla. Pokud je při editaci ponecháno pole pro vyplnění hesla prázdné, je to skriptem chápáno tak, že editor změnu hesla u uživatele nepožaduje a stejně k tomu přistupuje i funkce `database_save_record_auth`, která provádí kromě vkládání také editaci a to na základě parametru, který je zasílán při jejím volání. Pokud je heslo zadáno, provede se kromě editace samotné také převedení nového hesla na hash algoritmu `SHA1` a jeho nahrazení v databázi. Samozřejmostí jsou chybové hlášky zobrazované v případě chybně vykonaného SQL příkazu. Jestliže je editace úspěšná, je opět otevřen skript `list_auth`, ve kterém jsou viditelné provedené změny a je zobrazena potvrzovací hláška „`User was successfuly edited.`“.

Kód 4.8: Funkce `database_save_record_auth`

```
function database_save_record_auth($record,$what)
{
  global $webdoc, $template, $message;

  switch ($what) // testuje jaka akce ma byt provedena
  {
  case "1": // ulozeni noveho uzivatele
    $pass=sha1($record["pass"]); // prevod hesla
    if (database_query("INSERT INTO person (login, pass, role_id,
    name, surname, email, register_id, register_time, active) VALUES
    ('$record[login]', '$pass', '$record[role_id]', '$record[name]',
    '$record[surname]', '$record[email]', '$record[register_id]',
    '$record[register_time]', '$record[active]')"))
```

```
    {
      $_SESSION["$webdoc[main_table]_new_auth"]=1;
      $url="./?subpage1=$_GET[subpage1]&subpage2=list_auth";
      header("location: $url");
    }
    else
    {
      $webdoc["message"][]=database_get_message("error_saving");
      return false;
    }
  break;

  case "2": // update informaci o uzivateli
    if (database_query("UPDATE person SET login = '$record[login]',
    role_id = '$record[role_id]', name = '$record[name]', surname =
    '$record[surname]', email = '$record[email]', active =
    '$record[active]' WHERE person_id = $record[person_id]"))
    {
      $_SESSION["$webdoc[main_table]_edit_auth"]=1;
      $url="./?subpage1=$_GET[subpage1]&subpage2=list_auth";
      header("location: $url");
    }
    else
    {
      $webdoc["message"][]=database_get_message("error_edit");
      return false;
    }
  break;

  case "3": // zmena uzivatelskeho hesla
    $pass=sha1($record["pass"]);
    if (database_query("UPDATE person SET login = '$record[login]',
    pass = '$pass', role_id = '$record[role_id]', name =
    '$record[name]', surname = '$record[surname]', email =
    '$record[email]', active = '$record[active]' WHERE person_id =
    $record[person_id]"))
    {
      $_SESSION["$webdoc[main_table]_edit_auth"]=1;
      $url="./?subpage1=$_GET[subpage1]&subpage2=list_auth";
      header("location: $url");
    }
    else
    {
      $webdoc["message"][]=database_get_message("error_edit");
      return false;
    }
  break;

  }
}
```

V případě, že je třeba odstranit uživatele ze systému, provede osoba s příslušnými právy kliknutí na odkaz „`Delete`“ v rámci skriptu `list_auth`. Následně je požadováno potvrzení, zda má být konkrétní uživatel opravdu smazán. Po potvrzení je spuštěna funkce `database_delete_record_auth`, která provede

výmaz příslušného řádku v databázi. Poté je opětovně načtena stránka `list_auth` spolu s hláškou „`User was successfuly deleted.`“.

Kód 4.9: Funkce `database_delete_record_auth`

```
function database_delete_record_auth($record)
{
  global $webdoc, $template, $message;
  if (database_query("DELETE FROM person WHERE person_id =
  '$record'")) // vymazani uzivatele
  {
    $_SESSION["$webdoc[main_table]_delete_auth"]=1;
    $url="./?subpage1=$_GET[subpage1]&subpage2=list_auth";
    header("location: $url");
  }
  else
  {
    $webdoc["message"][]=database_get_message("error_delete");
    return false;
  }
}
```

Uživatelům, kteří nemají práva k celkové editaci, je umožněna změna jejich vlastního hesla pomocí skriptu `change_pass`. Opět zde probíhá kontrola správnosti zadání hesla. Pokud proběhne v pořádku, je zavolána funkce `database_change_userpass_auth`, která provede změnu v databázi. Současně s heslem je funkci zasíláno také `person_id`, které jednoznačně identifikuje uživatele. Tím docíleno toho, že lze změnit pouze heslo, které se váže ke konkrétnímu účtu.

Kód 4.10: Funkce `database_change_userpass_auth`

```
function database_change_userpass_auth($record)
{
  global $webdoc, $template, $message;

  $pass=sha1($record["pass"]);
  if (database_query("UPDATE person SET pass = '$pass' WHERE person_id
  = $record[person_id]")) // změna hesla uzivatele (pouze jeho
  vlastniho)
  {
    $_SESSION["$webdoc[main_table]_edit_auth"]=1;
    $url="./?subpage1=$_GET[subpage1]&subpage2=change_pass";
    header("location: $url");
  }
  else
  {
    $webdoc["message"][]=database_get_message("error_edit");
    return false;
  }
}
```

4.4 Jednotné zobrazování formulářových prvků

Formuláře slouží k zadávání uživatelských dat do hlavního programu, což je použito k naplnění databáze, nebo systémových proměnných. Na základě těchto vstupů mohou být vykonávány určité funkce, např. vytvoření tiskové sestavy či založení nového uživatele. V aplikaci budou použity dva typy formulářových prvků. První je klasický, vytvořený pomocí jazyka HTML a stylů CSS, druhý je v podstatě vnořeným programem, nadstavbou. Jedná se o tzv. WYSIWYG (What you see is what you get - co vidíš, to dostaneš) formulář. Aby byl oddělen HTML kód od vlastního zpracování pomocí jazyka PHP, vytvořil jsem funkci, která je spouštěna v rámci šablony systému Smarty a generuje právě příkazy jazyka HTML.

4.4.1 Klasické formulářové prvky

Nejprve bych chtěl nastínit, jak se formuláře vytvářejí. Pro vložení formuláře do HTML kódu slouží element FORM, který má atributy ACTION, což je URL skriptu, který je použit pro zpracování zadaných dat. V mém případě bude tento atribut prázdný, protože se spouštěný skript obsluhuje vždy sám podle toho, zda bylo zmáčknuto odesílací tlačítko. Dále pak atribut METHOD, určující způsob, jakým budou data předána na server. Používám metodu POST, která je určena pro odesílání větších formulářů, přičemž data jsou přenášena v těle HTTP požadavku. Konkrétní pole lze potom zavolat pomocí PHP proměnné `$_POST[“form_field”]`.

Nejčastěji používaným elementem je INPUT. Může vystupovat v mnoha různých podobách v závislosti na atributu TYPE. Ten určuje použití jednoho ze základních prvků formulářů. Při programování aplikace je použito sedmi prvků. Jedná se o TEXT, který slouží k zadání krátkého textu, např. jméno uživatele, příbuzný je PASSWORD, který je stejný jako TEXT, ovšem místo psaného textu jsou zobrazovány hvězdičky. Dále potom CHECKBOX, což je zatrhovací pole sloužící pro vstup logických hodnot. Zde je nutné navíc nastavit atribut VALUE, který určuje hodnotu, která se má při zaškrtnutí odeslat. Užil jsem také typ RADIO. Má stejný význam jako předešlý typ. Rozdíl je v tom, že RADIO zaručuje výběr právě jedné varianty, CHECKBOX žádné, nebo všech. Velmi důležitý je typ SUBMIT. Jedná se o tlačítko sloužící k odeslání formuláře. Pomocí atributu VALUE je nastavován text, který má být na tlačítku uveden, NAME potom jeho jméno. Neméně významným je HIDDEN. Toto pole se ve formuláři vůbec nezobrazí. Slouží k uchování stavové informace, která je

odeslána s vyplněným formulářem zpět skriptu. Posledním typem je potom FILE, který zobrazí vstupní pole a tlačítko otevírající dialog s možností výběru souboru, který má být uploadován na server. Při používání zadávacích polí se nesmí zapomenout na atribut NAME. Pod tímto jménem je obsah prvku dostupný v obslužném skriptu. [3]

Ukázka vytvoření formuláře:

```
<form action="index.php" method="post">
<input type="text" name="pokus">
<input type="submit" value="ODESLAT">
</form>
```

Výše uvedené HTML příkazy zobrazují formulář klasického vzhledu. Pro změnu je nutné definovat podobu pomocí CSS (Cascading Style Sheets – kaskádové styly).

Ukázka kódu stylu pro element INPUT, typ TEXT

```
.text { margin: 2px; color: #333333; border: 1px solid #5A7D37;
background-color: #C2D7AE; width : 100px; height: 18px; font-family :
Arial, sans-serif; font-size: 10pt; padding: 0px;}
```

4.4.2 Knihovna pro vytváření klasických formulářových prvků

Vytvoření knihovny pro vykreslování formulářů je jedním z bodů zadání diplomové práce, proto ji popíši detailně. Jelikož je její programový kód poměrně dlouhý, je vypsán v Příloze 1. V úvodu vysvětlím, jak se používá, dále se věnuji popisu zadání parametrů a na závěr popisuji postup vypracování.

Základem je propojení s užitým šablonovací systémem Smarty. Knihovna tedy není volána z obslužných PHP skriptů, ale přímo v šabloně, která obsahuje pouze HTML tagy. Šablonovací systém Smarty obsahuje funkce pro vykreslení formulářů typu RADIO a CHECKBOX. Moje funkce je tedy na zbývající používané typy, její název je `insert_form`.

Struktura funkce pro vykreslení formulářů:

```
{insert_form type="[typ_formulare]" name="[jmeno_pole]"
size="[velikost_pole]" value="[hodnota_pole]" rows="[radky]"
cols="[sloupce]" checked="[zvoleno]"}
```

Význam použitých proměnných:

[typ_formulare] - podporované typy jsou TEXT, PASSWORD, HIDDEN, TEXTAREA, SUBMIT, CHECKBOX (ale pouze pro jedno zaškrtávací pole)
- v případě nevyplnění je nastavena defaultní hodnota "text"
- při nesprávném vyplnění je vyvolána Smarty chybová hláška

[jmeno_pole] - při nevyplnění je vyvolána Smarty chybová hláška
- v případě typu SUBMIT a nevyplnění je nastavena defaultní hodnota "button"

[velikost_pole] - určuje velikost pole u typu TEXT a PASSWORD
- v případě nevyplnění je defaultně nastavena hodnota "20"

[hodnota_pole] - nastavuje přednastavenou hodnotu pole
- je nepovinným parametrem u typu TEXT, PASSWORD, FILE a TEXTAREA
- při nevyplnění u ostatních vyvolá Smarty chybovou hlášku

[radky] - určuje počet řádků u typu TEXTAREA
- v případě nevyplnění je nastavena defaultní hodnota "20"

[sloupce] - určuje počet sloupců u typu TEXTAREA
- v případě nevyplnění je nastavena defaultní hodnota "60"

[zvoleno] - určuje, zda má být pole typu CHECBOX zaškrtnuto - stačí vložit jakoukoli nenulovou hodnotu
- v případě nevyplnění je nastavena defaultní hodnota "0", tzn. pole nebude zaškrtnuto

Příklad volání funkce insert_form pro některé typy formulářů:

```
{insert_form type="text" name="login" value=$record.login} – pro TEXT
{insert_form type="checkbox" name="active" value="1"
checked=$record.active} – pro CHECKBOX s jedním zatrhávacím elementem
{insert_form type="hidden" name="person_id" value=$record.person_id} –
pro HIDDEN
```

`{insert_form type="submit" value="Edit"}` **– pro SUBMIT**

Stejně jako v případě `view_result`, je i `insert_form` vytvořena v jazyku PHP. Nejprve je testováno, zda jsou zadány všechny povinné parametry. Pokud ne, je zobrazena Smarty chybová hláška. Pokud ano, jsou testovány uvedené výjimky a podle toho jsou nastaveny proměnné. Na základě zadaného typu formulářového pole, získaného z parametru type, je do výstupní proměnné z knihovny vložen příslušný HTML tag formulářového pole.

Jak jsem již zmínil, Smarty obsahuje funkce pro vkládání formulářových polí typu RADIO a CHECKBOX, tedy `html_radios` a `html_checkboxes`. Rozdílem oproti mé funkci je ten, že je třeba před zavoláním poslat Smarty šabloně pomocí skriptu PHP pole, která budou obsahovat hodnoty, jež mají být uloženy při zvolení. Jména, která mají být před vybírací ikonou, ukazatel na prvek v poli, který má být při načtení stránky zvolen a separátor, který odděluje jednotlivě zobrazené elementy. Jako příklad uvádím zobrazení typu RADIO ve skriptu pro zadávání nového uživatele do aplikace. Nejprve jsou pomocí vytvořené funkce `database_get_person_role` načtena data z tabulky `Person_role` a vytvořena požadovaná pole. Tím je docíleno univerzálnosti, protože při přidání nové role do SQL tabulky je automaticky doplněna do zadávacího (editačního) formuláře a může být přidělena uživateli. Vytvořená pole jsou poté odeslána šabloně, odkud je zavolána požadovaná funkce.

Kód 4.11: Vytvoření polí potřebných k vykreslení formuláře typu CHECKBOX ve skriptu `adduser`

```
$result = database_get_person_role(); // nacteni dat z databaze
person_role
 while ($record_person_role = database_fetch_array($result)) //
 vytvoreni podkladu pro funkci na vytvoreni formulare type radio
  {
  $values[]=$record_person_role["role_id"];
  $names[]=$record_person_role["name"];
  }
  $template->template_assign("values",$values);
  $template->template_assign("names",$names);
  $template->template_assign("id",$role_id);
```

Příklad volání funkce html radios, která vytvoří formulářový typ RADIO ve skriptu adduser:

```
{html_radios name="role_id" values=$values output=$names selected=$id
separator=" "}
```

4.4.3 WYSIWYG formulář

Jedná se o program, který umožňuje vkládání a editaci textu ve formulářovém poli, kdy je verze psaná do tohoto pole vzhledově totožná s výslednou verzí dokumentu. Princip spočívá v tom, že je automaticky vytvářen HTML kód dle potřeb uživatele. Výsledná verze je potom uložena do databázového pole. Prostředí se podobá známým textovým editorům. V aplikaci je použita především při vyplňování komentářů, popisů a výsledků testovacích procedur.

Jelikož je v současné době k dispozici celá řada WYSIWYG formulářů, rozhodl jsem se vybrat jeden hotový produkt, který je volně šířitelný a ten včlenit do aplikace. Vybíral jsem z několika hledisek: modulárnost, možnosti nastavení, použitý programovací jazyk, kompatibilita s nejpoužívanějšími internetovými prohlížeči, zabezpečení proti úmyslnému vkládání HTML kódu mimo program formuláře, redundance výsledného kódu. Oproti návrhu, který byl obsažen v Semestrálním projektu, jsem se při realizaci práce rozhodl pro `TinyMCE`.

Ten je založen na programovacím jazyce JavaScript a jeho hlavní předností je, že automaticky vytváří formulářová pole z typu TEXTAREA. Díky tomu jsem ho jednoduše začlenil do knihovny pro vytváření formulářů `insert_form`. Volá se standardním příkazem, jehož popis je uveden výše. Pro jeho zavedení jsem musel nejprve do hlavního souboru šablony `index.tpl` a do hlavního souboru PHP skriptů `index.php` doplnit nastavení určující cestu ke zdrojovým souborům, vzhled a hlavně jaké akce s textem jsou povoleny.

Kód 4.12: Zavedení WYSIWYG formuláře `TinyMCE` v souboru `index.tpl`

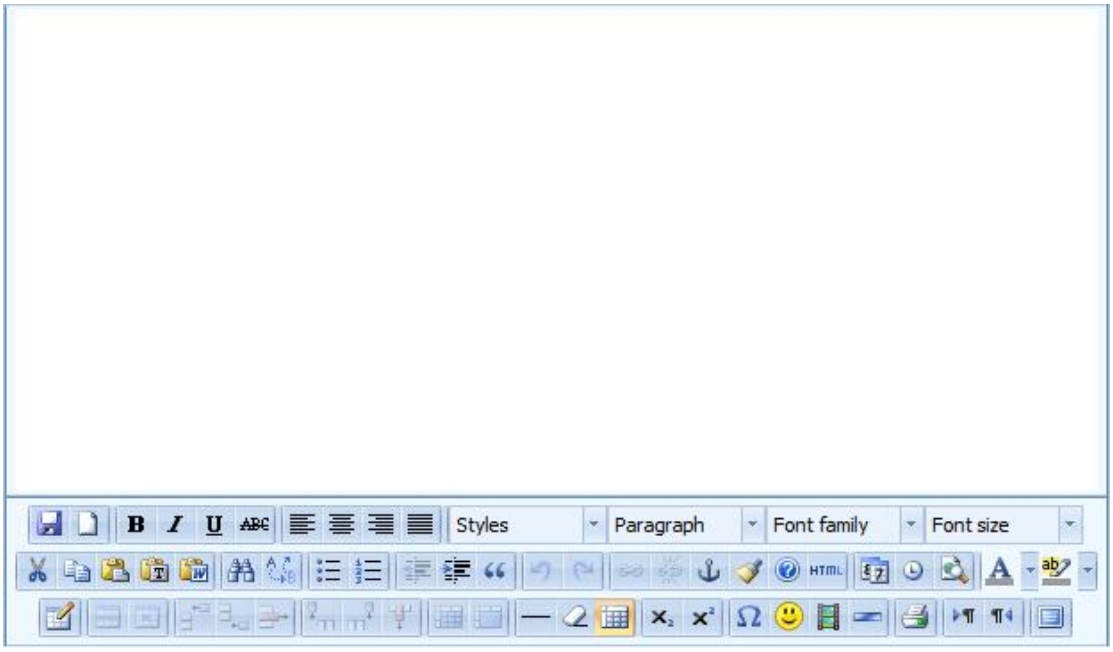
```
<script language="javascript" type="text/javascript"
src="../libraries/tinymce/jscripts/tiny_mce/tiny_mce.js"></script>

<script language="javascript" type="text/javascript">
{$webdoc.tinyMCE_init}
</script>
```

Kód 4.13: Zavedení WYSIWYG formuláře `TinyMCE` v souboru `index.php` (inicializace do globálního pole `webdoc`)

```
"tinyMCE_init" => "tinyMCE.init({
mode : 'textareas',
theme: 'advanced',
languages: 'en',
disk_cache: 'true',
debug: 'false',
plugins : 'safari,pagebreak,style,layer,table,save,advhr,advimage,
```

```
advlink,emotions,iespell,insertdatetime,preview,media,searchreplace,
print,contextmenu,paste,directionality,fullscreen,noneditable,
visualchars,nonbreaking,xhtmlxtras,template,inlinepopups',
theme_advanced_buttons1 : 'save,newdocument,|,bold,italic,underline,
strikethrough,|,justifyleft,justifycenter,justifyright,justifyfull,|,
styleselect,formatselect,fontselect,fontsizeselect',
theme_advanced_buttons2 : 'cut,copy,paste,pastetext,pasteword,|,
search,replace,|,bullist,numlist,|,outdent,indent,blockquote,|,
undo,redo,|,link,unlink,anchor,cleanup,help,code,|,insertdate,
inserttime,preview,|,forecolor,backcolor',
theme_advanced_buttons3 : 'tablecontrols,|,hr,removeformat,
visualaid,|,sub,sup,|,charmap,emotions,iespell,advhr,|,print,|,
ltr,rtl,|,fullscreen',
button_tile_map: 'true',
skin : 'o2k7'
})",
```

Předností tohoto formuláře je také to, že při poslání obyčejného výsledku dotazu z databáze, který obsahuje HTML tagy, např. při editaci, dojde k vykreslení dat do formuláře přesně tak, jak byla prvotně zadávána. [9]



Obr. 4.1: *Vzhled WYSIWYG formuláře* `TinyMCE`

4.5 Ostatní algoritmy

4.5.1 Přikládání souborů k záznamům v SQL tabulkách

Struktura `File`, reprezentována tabulkou `File_list`, vznikla kvůli splnění zadání, aby bylo možné k jednotlivým požadavkům, produktům, testům a jejich běhům

přikládat soubory. V okamžiku, kdy uživatel požaduje vložení souboru ke konkrétnímu záznamu, klikne na odkaz v `Files` v příslušné nabídce. Nejprve jsou zobrazeny již vložené soubory, které lze otevřít a prohlížet. Dále je možno soubory mazat. Zadávající požadoval, aby se soubory pouze označovali jako smazané, přitom ale na severu fyzicky zůstaly. Jména těchto souborů jsou pří výpisu přeškrtnuta. Důležitá je vlastnost algoritmu, že daný soubor může označit jako vymazaný pouze ten, kdo provedl vložení. Dále je ve stránce nabídnut dialog, po jehož vyplnění je daný soubor uploadován na server. V tabulce `File_list` je vytvořen nový záznam, přičemž nejdůležitějšími poli jsou `table_id` a `record_id`, která určují k jaké tabulce a jakému záznamu se soubor vztahuje.

Základem funkčnosti je Smarty funkce `files`, která je vložena do šablony `files.tpl`, jejímž úkolem je korektně v rámci šablony zobrazovat připojené soubory a příslušné akce (vymazání a vložení souboru). Kompletní kód této knihovny je uveden v Příloze 1.

Struktura funkce pro akce se soubory:

```
{files table_id="[nazev_tabulky]" record_id="[identifikator_zaznamu]“}
```

Význam použitých proměnných:

[nazev_tabulky] - přiřazuje záznamu o vložení souboru jeho konexi na určitou tabulku (requirement, product, test, test_run)

[identifikator_zaznamu] - určuje identifikační číslo záznamu, který se váže k uvedené tabulce

Příklad volání funkce files ve zdrojovém souboru files.tpl:

{files table_id=$webdoc.main_table record_id=$webdoc.sequence_id}

Druhým zdrojovým souborem, zajišťující funkci vkládání souborů je files.php, který obsahuje úlohy pro vkládání a mazání souborů.

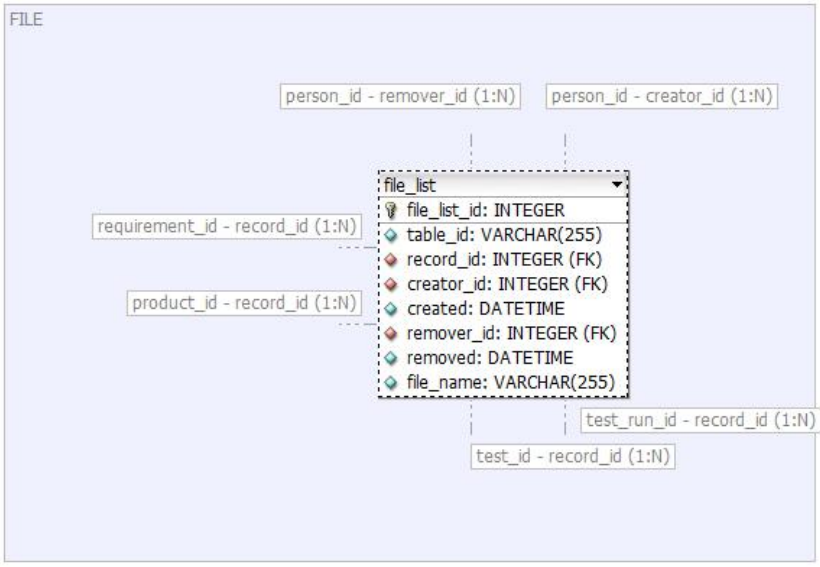
Kód 4.14: Vložení, přiřazení, souboru ke konkrétní tabulce a jejímu záznamu

```
if (isset($_POST["save_file"]))
{
```

```
  if ($_FILES["soubor"]["name"]<>"")
  {
      $soubor_name = $webdoc["auth"] . "_" .
      $_FILES["soubor"]["name"]; // priprava jmena souboru
      $result=database_query("SELECT * FROM file_list WHERE file_name
      LIKE '$soubor_name'"); // existuje takovy soubor v db?
      if (database_num_rows($result)==0)
      {
        if (!move_uploaded_file($_FILES["soubor"]["tmp_name"],
        "../files/$soubor_name")) // upload souboru s puvodnim nazvem
        {
          $webdoc["message"][]=database_get_message("no_upload");
          return false;
        }
      }
      else
      {
        $soubor_name = rand(0,999) . "_" . $soubor_name; // doplneni
        nazvu o vygenerovane cislo - predejiti prepsani
        if (!move_uploaded_file($_FILES["soubor"]["tmp_name"],
        "../files/$soubor_name"))
        {
          $webdoc["message"][]=database_get_message("no_upload");
          return false;
        }
      }
      if (!database_query("INSERT INTO file_list (table_id, record_id,
      creator_id, created, file_name) VALUES ('$webdoc[main_table]',
      '$webdoc[sequence_id]', '$webdoc[auth]', now(),
      '$soubor_name')")) // vlozeni zaznamu do db
      {
        $webdoc["message"][]=database_get_message("error_saving");
        return false;
      }
  }

  else
  {

$webdoc["message"]["form_error"][]=database_get_message("no_file");
  }
}
```

Kód 4.15: Vymazání souboru (jeho de facto označení za smazaný)

```
if (isset($_POST["delete_file"])) // test zmacknuti tlacitka
{
  if (!database_query("UPDATE file_list SET
  remover_id='$webdoc[auth]', removed=now() WHERE
  file_list_id=$_POST[file_list_id]")) // oznaceni souboru za smazany
  {
    $webdoc["message"][]=database_get_message("error_saving");
    return false;
  }
}
```



Obr. 4.2: *Schéma struktury* `File`

Tab. 4.1: Struktura tabulky `File_list`

Název pole	**Datový typ pole**	**Komentář**
File_list_id (PK)	INTEGER	Identifikační číslo souboru.
Table_id	VARCHAR(255)	Název tabulky.
Record_id	VARCHAR(255) (FK)	Identifikátor záznamu.
Creator_id	INTEGER (FK)	Identifikační číslo vkládajícího uživatele.
Created	DATETIME	Čas vložení.
Remover_id	INTEGER (FK)	Identifikační číslo mazajícího uživatele.
Removed	DATETIME	Čas vymazání.
File_name	VARCHAR(255)	Název souboru.

4.5.2 Vypisování hlášek

Kromě algoritmů obsažených v zadání práce bylo nutné pro univerzální chod aplikace vytvořit některé další funkce.

V rámci aplikace jsme se s kolegou snažili ošetřit všechny stavy, které kdyby nastaly, nezaručí plynulý chod systému. Jedná se například o doručení prázdného, nebo

chybně vykonaného dotazu do databáze, nevyplnění povinného či nesprávně vyplněného formulářového pole atd. Tyto stavy vyvolají vypsání chybové hlášky, zobrazené ve Smarty šabloně, tzn. ve výstupu, který je určený pro uživatele aplikace. Aby byly všechny hlášky, vážící se k jednomu problému, stejně napsané a nemuseli jsme se jejich psaním zabývat při programování, vytvořil jsem funkci `database_get_message`, která po zavolání vrátí požadovanou hlášku. Jednotlivé hlášky jsou uloženy v SQL tabulce text a jsou vyhledávány podle jejich jména. Prakticky to funguje tak, že je v PHP skriptu zavolána funkce s příslušným názvem hlášky. Výsledek je uložen do globálního pole a vykreslen v šabloně.

Kód 4.16: Zavolání funkce `database_get_message` spolu s jejím kódem z knihovny `database-mysql.php`

```
$webdoc["message"][]=database_get_message("error_delete");

function database_get_message($name)
{
  $result=database_query("SELECT * FROM text WHERE name LIKE
  '$name'"); // ziskani chybove hlasky na zaklade jejiho pojmenovani
  if (database_num_rows($result)==1) //zaznam existuje v db
  {
    $result_db = database_fetch_array($result);
    return $result_db["text"];
  }
  else //porovnani se nezdarilo, dana hlaska neni v db, zobraz
  univerzalni
  {
    $no_message="There was an error.";
    return $no_message;
  }
}
```

Obr. 4.3: *Schéma struktury* `Text`

Tab. 4.2: Struktura tabulky `Text`

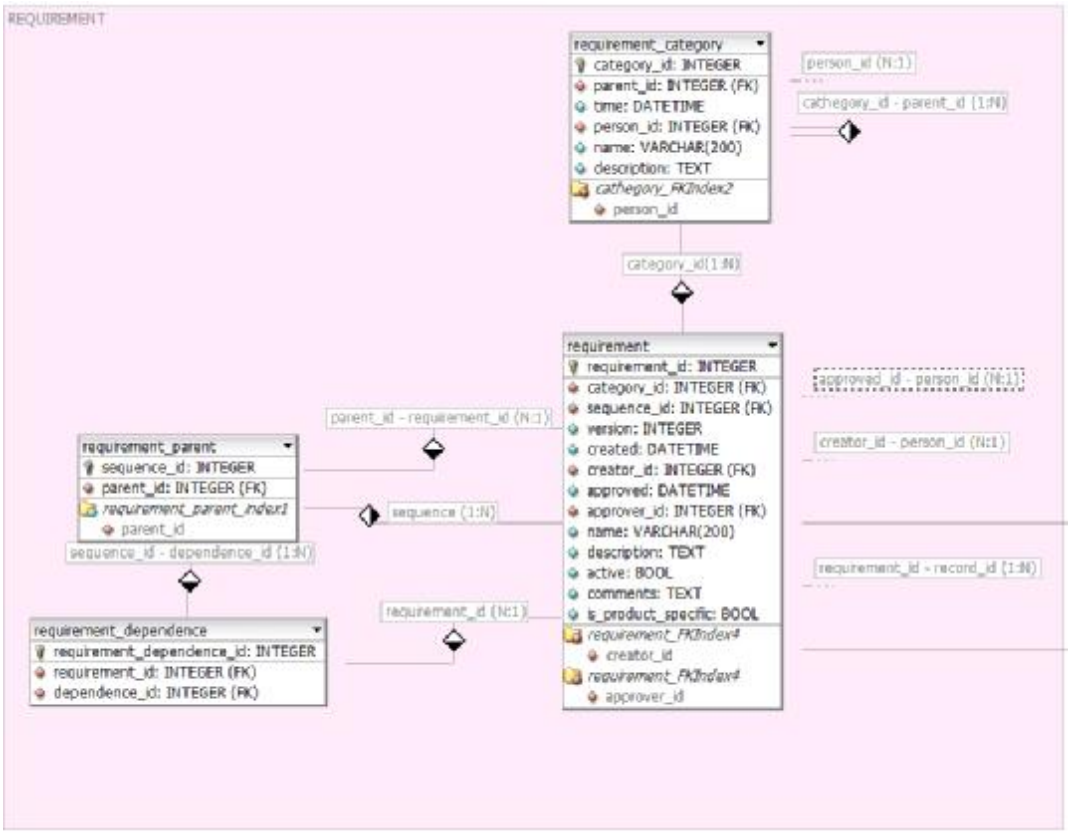
Název pole	**Datový typ pole**	**Komentář**
Text_id (PK)	INTEGER	Identifikační číslo textového záznamu.
Name	VARCHAR(255)	Název záznamu.
Text	TEXT	Text záznamu.

5 REALIZACE DATABÁZE

Tvorba databázové části aplikace je rozděleno na dvě části – Requirement (požadavky) Products (produkty), které řeší kolega Bc. Ondřej Gregárek, Person (autorizace uživatelů), Test (testovací procedury) a Test Run (běhy testů), které řeším v diplomové práci já. Některé databázové struktury, File a Text, jsou řešeny v předchozí kapitole, neboť se jedná o jednodušší struktury a stěžejní je algoritmus, který je obsluhuje.

5.1 Část Requirement

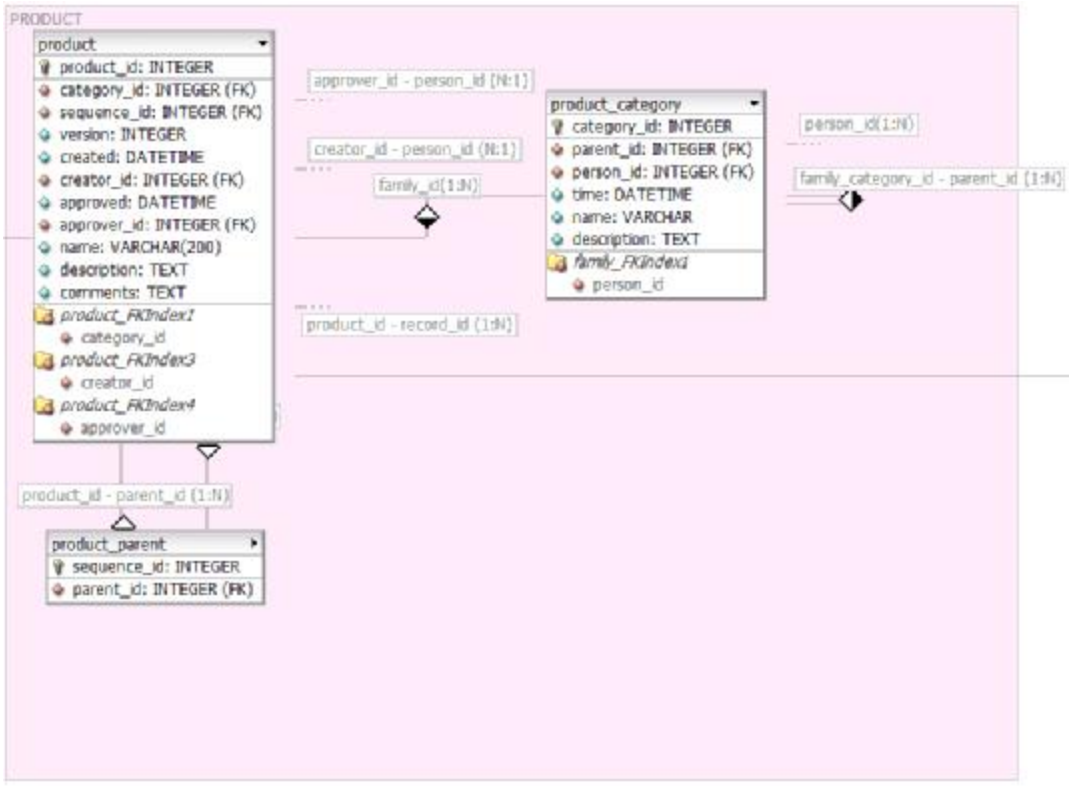
Nejprve bych chtěl uvést blokové schéma struktur části Requirement, kterou jsem převzal od kolegy. Uvádím je pouze pro lepší pochopení celé problematiky včetně krátkých komentářů.



Obr. 5.1: *Blokové schéma struktury Requirement*

Soustava tabulek `Requirement`, `Requirement_cathegory`, `Requirement_parent` a `Requirement_dependence` řeší zadávání požadavků

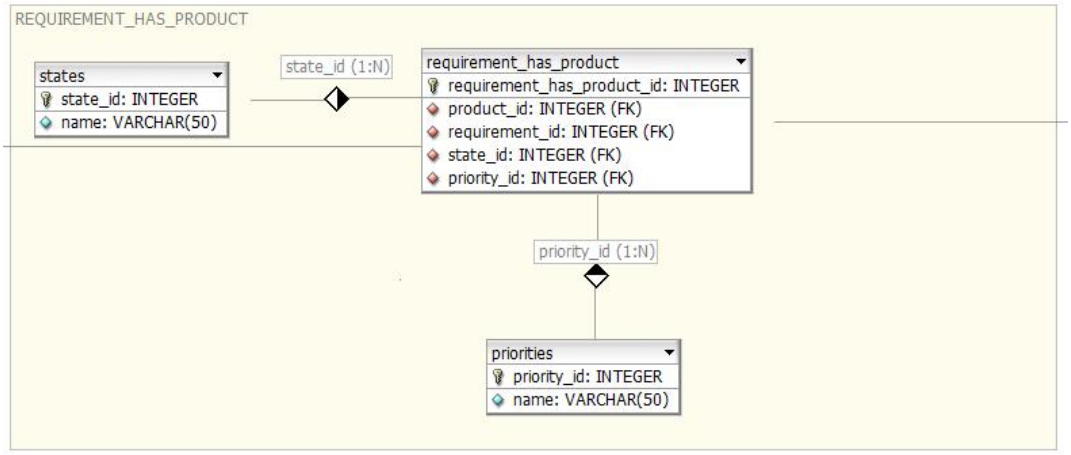
do systému a práci s nimi, viz. Obr. 5.1. Nejprve dojde k vytvoření požadavku, přičemž jsou uloženy potřebné informace spolu s identifikačním číslem zakládající osoby přes vazbu s tabulkou `Person`. V tuto chvíli lze záznam zpětně editovat a upřesňovat. V okamžiku schválení se požadavek stává uzavřeným pro editaci a zároveň aktivním pro jeho následné vypracování. Pokud je třeba změny, nebo se při realizaci zjistí, že dle zadaných kritérií nelze s požadavkem pracovat, lze vytvořit požadavek nový, který se stává kopií starého, včetně všech vazeb. Vytvořit novou větev lze i z historických záznamů, stejně tak se na jakýkoli záznam může dotazovat nově vytvořený test či produkt. Tabulka `Requirement` je vazbou N:N propojena se strukturou `Product` a stejnou vazbou s částí `Test`.



Obr. 5.2: *Blokové schéma struktury Product*

`Product` je v podstatě podčástí Requirement a je praktickou realizací konkrétního požadavku, viz. Obr. 5.2. Jelikož se nepředpokládá, že by prvně vytvořený produkt vždy splňoval očekávání je umožněno opět větvení. Dále je umožněno dělení produktů do různých kategorií, a také vytvoření více produktů z jednoho požadavku a naopak. Vazbu N:N s tabulkou `Requirement` znázorňuje následující obrázek, další

vazby jsou opět s Person, kvůli identifikaci uživatele, a s části Test, protože i vytvořený produkt musí jít otestovat.



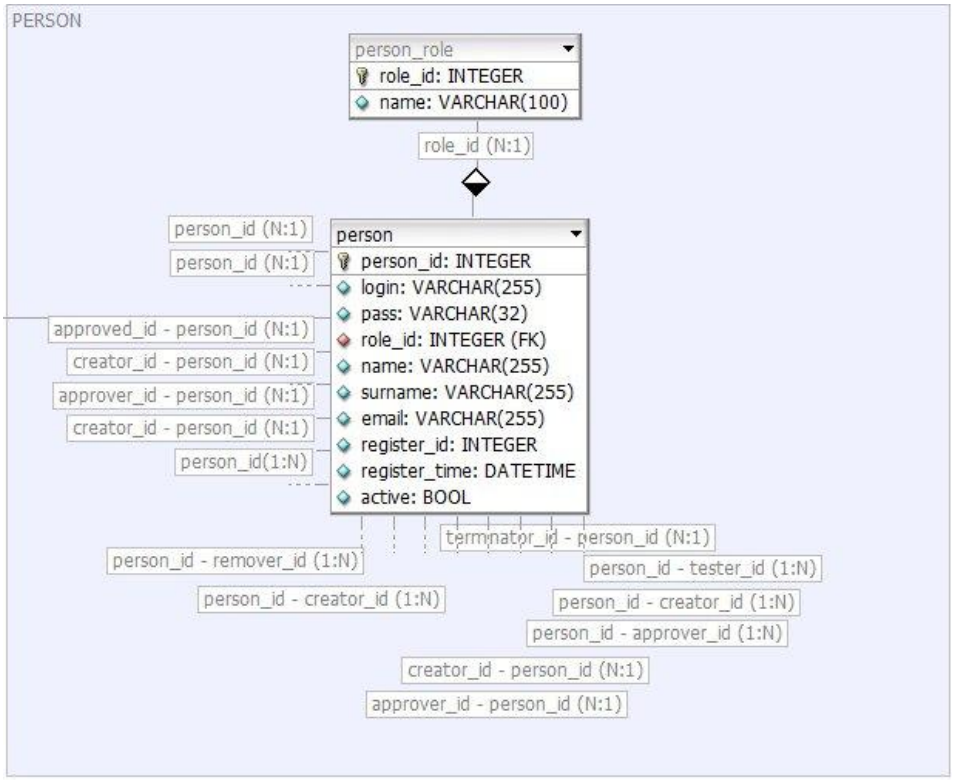
Obr. 5.3: *Vazba N:N mezi Requirement a Product*

5.2 Část Person

Je velmi důležitá, protože obsahuje seznam uživatelů systému a především jejich přiřazení k určitým rolím. Je provázána se všemi strukturami, kromě `Text`. Jedná se o předávání identifikačního čísla uživatele, což umožní, že lze vždy zjistit, která osoba vytvořila, editovala, zaktivnila, atd. požadavek, produkt či test. Vazby jsou tedy typu 1:N – jedno identifikační číslo uživatele může být přiřazeno k N záznamům v provázaných tabulkách. Také je vždy kontrolováno, zda má daná osoba právo ke konkrétním úkonům. To je zajištěno přidělením oprávnění jednotlivým uživatelům.
Struktura se skládá ze dvou tabulek. `Person` obsahuje všechna požadovaná data o uživateli, `Person_role` je pomocnou tabulkou, která přiřazuje identifikátoru `role_id` jeho textové vyjádření (DA, DE, TA, TE, TT).

Přidání nového uživatele vede přes formulářové pole, přístupné pouze uživatelům DA a TA.

Kostrou této části je autorizační algoritmus, který byl popsán v předchozí kapitole.



Obr. 5.4: *Blokové schéma struktury* `Person`

Tab. 5.1: Struktura tabulky `Person`

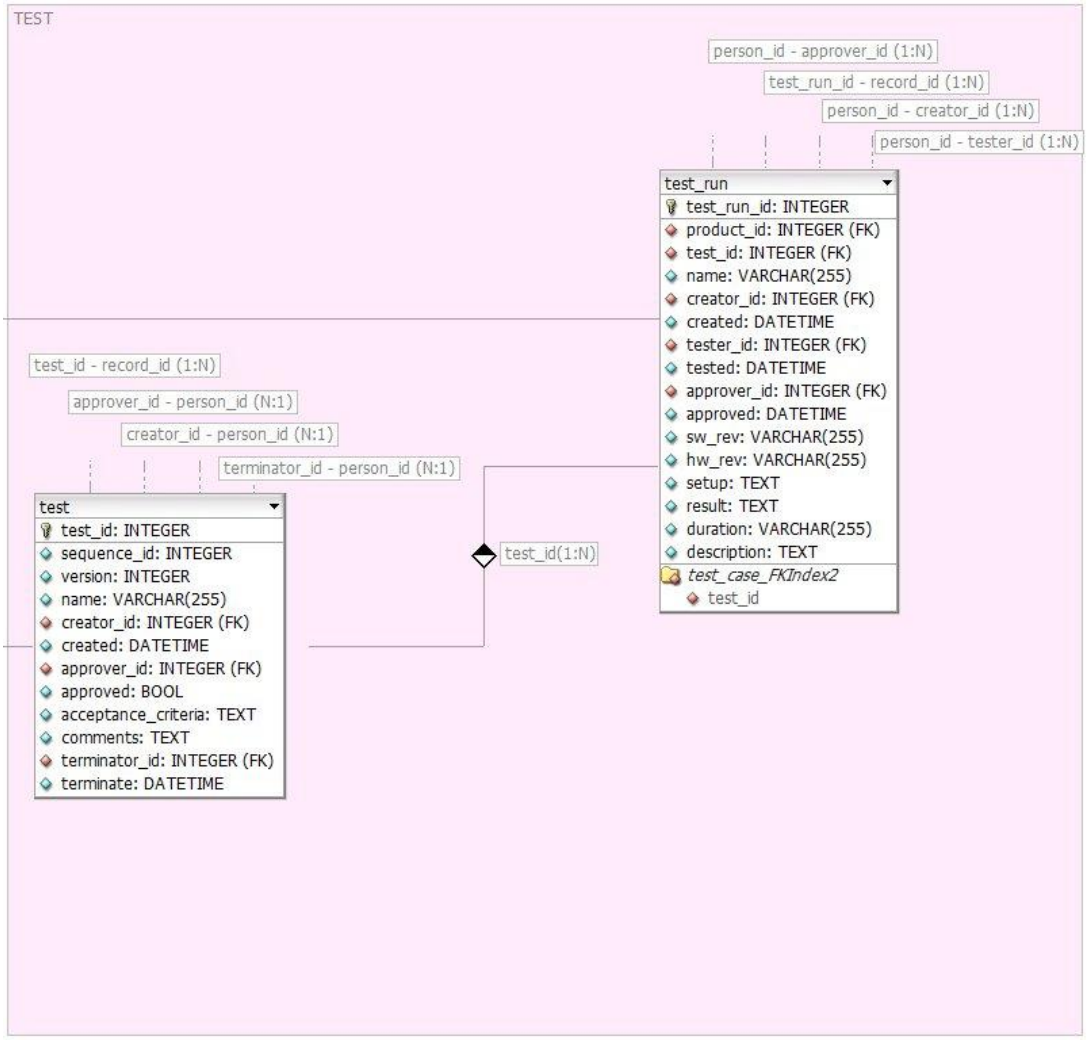
Název pole	**Datový typ pole**	**Komentář**
Person_id (PK)	INTEGER	Identifikační číslo uživatele.
Login	VARCHAR(255)	Přihlašovací jméno.
Pass	VARCHAR(32)	Heslo uložené v sha1.
Role_id	INTEGER (FK)	Identifikátor role uživatele
Name	VARCHAR(255)	Křestní jméno uživatele
Surname	VARCHAR(255)	Příjmení uživatele.
Email	VARCHAR(255)	E-mail uživatele.
Register_id	INTEGER	Identifikační číslo registrátora.
Register_time	DATETIME	Čas registrace.
Active	BOOL	Je uživatel aktivní?

Tab. 5.2: Struktura tabulky `Person_role`

Název pole	Datový typ pole	Komentář
Role_id (PK)	INTEGER	Identifikační číslo role.
Name	VARCHAR(100)	Textové vyjádření role (DA, DE..).

5.3 Část Test

Na základě požadavku, nebo již hotového produktu, je nutné vytvořit testovací proceduru. Nejprve je vytvořena definice testu, a poté je na jejím základě test prakticky proveden, proto byla část `Test` rozdělena na dvě podčásti - `Test` a `Test Run`.



***Obr. 5.5:** Blokové schéma struktury `Test`*

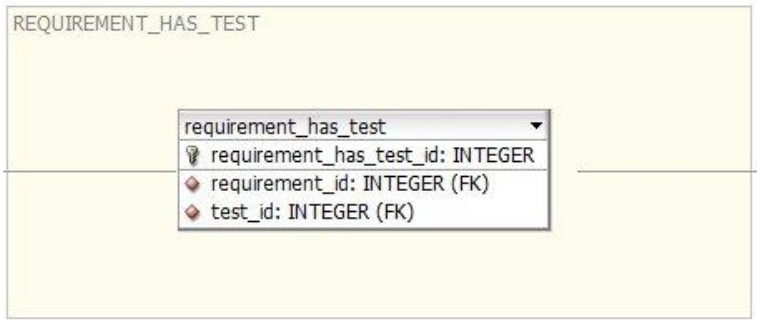
5.4.1 Struktura Test

K vytvoření zadání testu je určena tabulka `Test`. Protože i zde vyvstala nutnost větvení testů, aby byla zachována historie, tzn. veškeré editace, neobsahuje tabulka pouze sloupce pro identifikaci a definici testu, ale také informace o jeho sekvenční linii a verzi. Záznam nelze v tabulce smazat, lze jej pouze označit jako ukončený.

Tab. 5.3: Struktura tabulky `Test`

Název pole	**Datový typ pole**	**Komentář**
Test_id (PK)	INTEGER	Identifikační číslo testu.
Sequence_id	INTEGER	Identifikační číslo sekvenční linie
Version	INTEGER	Číslo verze testu v rámci sekvenční linie.
Name	VARCHAR(255)	Jméno testu.
Creator_id	INTEGER (FK)	Person_id osoby, která test vytvořila.
Created	DATETIME	Datum a čas vytvoření testu.
Approver_id	INTEGER (FK)	Person_id osoby, která schválila test.
Approved	DATETIME	Datum a čas schválení testu.
Acceptance_criteria	TEXT	Kritéria pro přijetí testu.
Comments	TEXT	Komentář k testu.
Terminator_id	INTEGER (FK)	Person_id osoby, která test označila jako ukončený
Terminate	DATETIME	Datum a čas ukončení testu.

Podmínkou zadávajícího byla možnost přiřazení více požadavků (Requirement) k jednomu zadání testu a naopak. Tím byla definována vazba N:N mezi tabulkami `Requirement` a `Test`, kterou představuje tabulka `Requirement_has_test`.



Obr. 5.6: *Blokové schéma tabulky* `Requirement_has_test`

Tab. 5.4: Struktura tabulky `Requirement_has_test`

Název pole	**Datový typ pole**	**Komentář**
Requirement_has_test_id (PK)	INTEGER	Identifikační číslo záznamu.
Requirement_id	INTEGER (FK)	Převzato z tabulky Requirement.
Test_id	INTEGER (FK)	Převzato z tabulky Test.

V dalším textu se zaměřím na popis funkčnosti struktury `Test`.

Po kliknutí na odkaz `Tests` v hlavním menu se uživatel dostane do stejnojmenné části. Jako první je mu zobrazen výpis již existujících záznamů.

Výpis vytvořených testů

Nejprve je vykonán dotaz do databáze PHP funkcí `database_get_list`, výsledek je zobrazen pomocí Smarty funkce `view_result`, jejíž popis je uveden v předchozí kapitole. Vypsáno je jméno testu spolu s časem jeho vytvoření. Pokud je záznam označen jako ukončený, jsou vypsané údaje přeškrtnuty. Dle uživatelského oprávnění je povolena následná práce, a to výpis celého záznamu, jeho editace a historie. Jestliže je tabulka `Test` prázdná, je zobrazena hláška „`List is emty`“.

Zadání nového testu

Kromě výpisu je při spuštění této části aktivní odkaz pro zadání nového testu. Po jeho kliknutí je uživatel nejprve přesměrován do části Requirement, kde si vybere, z jakého požadavku bude test vytvořen. Zpět je zasláno identifikační číslo zvoleného

požadavku, `requirement_id`. Je zobrazen formulář pro zadání potřebných dat k vytvoření nového testu, přičemž povinnými poli jsou `Name`, `Acceptance Criteria` a `Comments`. Na jejich nezadání je upozorněno chybovými hláškami (např. „`Field Name must be filled.`“).

Jestliže jsou potřebná data korektně zadána, je spuštěna PHP funkce `database_save_record_test`, která provede zápis do příslušných tabulek. V prvé řadě je nastavena hodnota `version` (číslo verze testu v jedné větvi) na hodnotu 0, neboť se jedná o zakladatele nové větve. Poté je proveden dotaz do tabulky `Test` pro získání naposledy použitého identifikátoru větve, `sequence_id`, který je zvýšen o 1. Je proveden zápis do tabulky `Test`. Dále je nutné vytvořit vazbu N:N mezi `Test` a `Requirement` pomocí tabulky `Requirement_has_test`. Je zjištěna hodnota identifikátoru nově založeného testu, `test_id`, který je spolu s `requirement_id` zapsán do uvedené tabulky. Tím je vytvořena potřebná vazba. Nový test je založen. Funkce otevře opět úvodní stránku s výpisem spolu s hláškou informující o úspěšném uložení záznamu. Ošetřil jsem stav chybného zápisu zobrazením informující hlášky.

Kód 5.1: Výpis části funkce `database_save_record_test` realizující zápis nového záznamu

```
$record["version"]=0;

  $result=database_query("SELECT MAX(sequence_id) AS sequence_id FROM
  test"); //zjisteni posledního sequence_id
  $result_db = database_fetch_array($result);
  $record["sequence_id"]=($result_db["sequence_id"])+1; // vytvoreni
  noveho sequence_id

  if (!database_query("INSERT INTO test (sequence_id, version, name,
  creator_id, created, approver_id, approved, acceptance_criteria,
  comments, terminator_id, terminate) VALUES ('$record[sequence_id]',
  '$record[version]', '$record[name]', '$record[creator_id]',
  '$record[created]',
  ".(isset($record["approver_id"])?"'$record[approver_id]'":"NULL").",
  ".(isset($record["approved"])?"'$record[approved]'":"0").",
  '$record[acceptance_criteria]', '$record[comments]',
  ".(isset($record["terminator_id"])?"'$record[terminator_id]'":
  "NULL").",".(isset($record["terminate"])?"'$record[terminate]'":"0")
  .")")) // zapis informaci do databaze
  {
    $webdoc["message"][]=database_get_message("error_saving");
    return false;
  }

  $result=database_query("SELECT MAX(test_id) AS test_id FROM test");
  //zjisteni posledniho test_id
  $result_db = database_fetch_array($result);
```

```
$record["test_id"]=($result_db["test_id"]); // vytvoreni noveho
test_id

if (database_query("INSERT INTO requirement_has_test
(requirement_id, test_id) VALUES ('$record[requirement_id]',
'$record[test_id]')")) // zapis do databáze
{
    $_SESSION["$webdoc[main_table]_record_saved"]=true; // hlaska o
    ulozeni
    $url="./?subpage1=tests&subpage2=list";
    header("location: $url"); // presmerovani na vypis
}
else
{
    $webdoc["message"][]=database_get_message("error_saving");
    return false;
}
```

Zobrazení celého testu

Na úvodní straně jsou zobrazeny pouze informace identifikující daný test. Pro zobrazení kompletního výpisu je třeba kliknout na odkaz `View`, který je uveden pro každý záznam zvlášť. Skriptu je zaslán identifikátor testu, `test_id`, pomocí kterého je proveden dotaz do databáze k získání příslušných dat. Tato data jsou následně uspořádána do pole, které je předáno k vykreslení Smarty šabloně.

Editace vytvořeného testu

Kvůli požadavku na archivaci všech změn nedochází k editaci záznamu, ale vytváření nového záznamu, ovšem se změnami, jaké si uživatel žádá v rámci editace. Po kliknutí na odkaz `Edit`, který je uveden u každého záznamu zvlášť, je otevřen stejný formulář, jako v případě založení testu nového, ovšem s vyplněnými hodnotami dle příslušnosti. Na tomto místě je tedy možné libovolně měnit data. Dále je uživateli umožněno připojit k editovanému testu další požadavky (Requirement). Počet takto připojených požadavků není omezen. Jsou uloženy v poli, které je předáno funkci při provedení editace. V jejím rámci je možné tyto požadavky odpojovat, a to zaškrtnutím příslušné hodnoty a stiskem tlačítka.

Při stisku tlačítka editace je zkontrolováno, zda jsou zadány všechny povinné hodnoty a je opět spuštěna PHP funkce `database_save_record_test`. Ta na základě zaslaného parametru `$record["edit"]` pozná, že se jedná o editaci. Stejně jako u založení nového, je při editaci třeba zjistit hodnoty verze a sekvence. Proto se provede dotaz do databáze, kde se vyhledají pod původním `test_id` tyto hodnoty. Sekvence zůstává stejná i pro editovaný záznam, je tím zachována příslušnost

k sekvenční linii. Číslo verze je zvýšeno o 1, čímž je vytvořena hierarchie. Dále je proveden zápis do tabulky `Requirement_has_test` tak, aby vznikly vazby. Do této tabulky je postupně zapisováno podle toho, kolik prvků má pole připojených požadavků. Jestliže je editace úspěšná, je uživatel přesměrován na výpis všech testů, v opačném případě je zobrazena chybová hláška „`Get an error during saving.`“. Pokud je test označen jako ukončený, není možné jej editovat. Při kliknutí na takový test není otevřen editační formulář, ale je vypsána hláška „`Impossible to edit this Test. Record is terminated.`“. Editace je umožněna uživateli s právy DA, DE, TA a TE.

Kód 5.2: Výpis části funkce `database_save_record_test` realizující editaci záznamu

```
if ($record["edit"]) // jedna se o editaci?
  {
    $result=database_query("SELECT * FROM test WHERE test_id LIKE
    '$record[test_id]'");
    $result_db=database_fetch_array($result); // zjisteni sekvence a
    verze
    $record["version"]=($result_db["version"])+1;
    $record["sequence_id"]=$result_db["sequence_id"];
    if (!database_query("INSERT INTO test (sequence_id, version, name,
    creator_id, created, approver_id, approved, acceptance_criteria,
    comments, terminator_id, terminate) VALUES
    ('$record[sequence_id]',
    '$record[version]', '$record[name]', '$record[creator_id]',
    '$record[created]',
    ".(isset($record["approver_id"])?"'$record[approver_id]'":
    "NULL").",
    ".(isset($record["approved"])?"'$record[approved]'":"0").",
    '$record[acceptance_criteria]', '$record[comments]',
    ".(isset($record["terminator_id"])?"'$record[terminator_id]'":
    "NULL").",
    ".(isset($record["terminate"])?"'$record[terminate]'":"0").")"))
    {
      $webdoc["message"][]=database_get_message("error_saving");
      return false;
    }
    $result=database_query("SELECT MAX(test_id) AS test_id FROM
    test"); //zjisteni maximalniho test_id
    $result_db = database_fetch_array($result);
    $new_test_id=$result_db["test_id"];
    $result=database_query("SELECT * FROM requirement_has_test WHERE
    test_id LIKE '$record[test_id]'");
    if (database_num_rows($result)==0) return false;
    else
    {
     while ($zaznam = database_fetch_array($result)) // zapis pole
     pozadavku pro vytvoreni vazby N:N
      {
        if (!database_query("INSERT INTO requirement_has_test
       (requirement_id, test_id) VALUES ('$zaznam[requirement_id]',
```

```
        '$new_test_id')"))
        {
          $webdoc["message"][]=database_get_message("error_saving");
          return false;
        }
      }
    }
    $_SESSION["$webdoc[main_table]_record_saved"]=true;
    $url="./?subpage1=tests&subpage2=list";
    header("location: $url");
  }
```

Zobrazení historie testu

Jak bylo řečeno výše, editace vytvoří nový záznam dle požadovaných úprav. Skript zobrazující historii spustí PHP funkci `database_get_history`, která vytvoří pole výsledků dotazu do databáze. Obsahuje záznamy se stejným sekvenčním číslem, `sequence_id`, jako záznam, v rámci kterého byl požadavek na historii zaznamenán. Je tedy zobrazena hierarchie testů, přičemž nejnovější je záznam s nejvyšší verzí, `version`, a je zobrazen nahoře. Zobrazení je opět pomocí Smarty funkce `view_result`, se záznamy lze manipulovat stejně, jako v případě výpisu na úvodní straně.

Schválení testu

Je úlohou pro DA, DE či TA. Po vytvoření testu je třeba jej schválit, aby bylo možno založit jeho běh, Test Run. Schválený test je možné editovat, protože je vytvořen test nový, který při založení schválen není.

Po kliknutí na odkaz `Approving` je nejprve zobrazena hláška informující uživatele, zda test je, nebo není schválen. Pokud dosud nebyl, je zobrazeno tlačítko, jehož odesláním je zavolána PHP funkce `database_approve_record`, která jej schválí. Ve zbytku stránky je zobrazen úplný výpis záznamu, aby uživatel věděl přesně, co schvaluje.

Kód 5.3: Výpis funkce `database_approve_record` schvalující záznam

```
function database_approve_record($record_id)
{
  global $webdoc;
  return database_query("UPDATE $webdoc[main_table] SET
  approved=now(), approver_id='$_SESSION[_auth]' WHERE
  $webdoc[main_table]_id=$record_id");
}
```

Ukončení testu

Aby byla zachována archivace všech vytvořených záznamů a přitom mohly být některé záznamy označeny jako ukončené, vytvořil jsem skript, který toto umožňuje. Funguje obdobně jako schvalování. Důsledky ukončení testu v sekcích výpisu a editace jsou uvedeny výše.

Vytvoření běhu testu (Test Run)

Předpokladem pro test je následné vytvoření jeho běhu, Test Run. Struktura vytváření je podstatou stejná jako schvalování a ukončování testu. Jestliže test není schválen, nebo je ukončen, není umožněno vytvoření běhu testu a je zobrazena hláška „Impossible to make a Test Run. Record is not approved.“, respektive „Impossible to make a Test Run. Record is terminated.“. V opačném případě je opět zobrazeno tlačítko, konkrétně Make Test Run. Po kliknutí na toto tlačítko je uživatel s oprávněním DA, DE, TA či TE přesměrován do struktury Test Run. Je zobrazen formulář pro vyplnění dat běhu testu.

5.4.2 Struktura Test Run

Je-li vytvořen a schválen test, přichází na řadu jeho vlastní realizace, což představuje struktura Test Run. Nejprve musí dojít k jeho vytvoření, což je umožněno uživatelům s právy DA, DE, TA, TE a následně ke schválení TA. K samotnému provedení testu je určen především uživatel s právy TT.

Tab. 5.5: Struktura tabulky Test_run

Název pole	**Datový typ pole**	**Komentář**
Test_run_id (PK)	INTEGER	Identifikační číslo běhu testu.
Product_id	INTEGER	Identifikační číslo produktu, ke kterému se běh testu váže.
Test_id	INTEGER	Identifikační číslo testu, ke kterému se běh testu váže.
Creator_id	INTEGER	Person_id osoby, která běh testu vytvořila.
Created	DATETIME	Datum a čas vytvoření běhu testu.

Tester_id	INTEGER	Person_id osoby, která provedla běh testu.
Tested	DATETIME	Datum a čas provedení běhu testu.
Approver_id	INTEGER	Person_id osoby, která schválila běh testu.
Approved	DATETIME	Datum a čas schválení běhu testu.
Sw_rev	VARCHAR(255)	Softwarová revize.
Hw_rev	VARCHAR(255)	Hardwarová revize.
Setup	VARCHAR(255)	Nastavení provedeného testu.
Result	TEXT	Výsledek běhu testu.
Duration	INTEGER	Doba trvání běhu testu.
Description	TEXT	Další popis k běhu testu.

Stejně jako v případě Testu je při vstupu do části Test Run zobrazen výpis již existujících běhů.

Výpis vytvořených běhů testu

Probíhá stejným způsobem jako zobrazení vytvořených testů, které jsem popsal v kapitole 5.3.1, pomocí Smarty funkce `view_result`. Jsou tedy zobrazeny informace o jménu běhu testu společně s datem a časem jeho vytvoření. S každým záznamem je možno dále pracovat, jedná se o úlohy kompletního zobrazení, editace a kopírování.

Zadání nového běhu testu

Jak bylo uvedeno v předchozí kapitole, založení nového běhu testu je možné z části Test či Product, jestliže byl daný záznam schválen a není označen jako ukončený. Jestliže uživatel systému klikne na odkaz pro založení běhu testu ve stejnojmenné části, je přesměrován do části Test. Test Run je propojen s Test a Product vazbou 1:N, tzn. z jednoho testu či produktu lze vytvořit N běhů.

Po výběru záznamu, ze kterého má vzniknout běh je zobrazen formulář pro zadání požadovaných dat. Povinnými položkami jsou `Name` a `Setup`. Jestliže není jedno z těchto polí vyplněno, je zobrazena chybová hláška a je požadováno doplnění.

Pokud je splněna podmínka vyplnění všech požadovaných údajů, je zavolána funkce `database_save_record_test`, která provede zápis do databáze. Poté je uživatel přesměrován na výpis existujících záznamů a je zobrazena hláška informující o úspěšném uložení. Pokud nastala při ukládání chyba, je o tom uživatel informován opět chybovou hláškou.

Kód 5.4: Výpis části funkce `database_save_record_test`, která provádí uložení běhu testu

```
if (!database_query("INSERT INTO test_run (product_id, test_id, name,
creator_id, created, tester_id, tested, approver_id, approved, sw_rev,
hw_rev, setup, result, duration, description) VALUES
('$record[product_id]', '$record[test_id]', '$record[name]',
'$webdoc[auth]', now(),
".(isset($record["tester_id"])?"'$record[tester_id]'":"NULL").",
".(isset($record["tested"])?"'$record[tested]'":"0").",
".(isset($record["approver_id"])?"'$record[approver_id]'":"NULL").",
".(isset($record["approved"])?"'$record[approved]'":"0").",
'$record[sw_rev]', '$record[hw_rev]', '$record[setup]',
'$record[result]', '$record[duration]', '$record[description]')"))
      {
        $webdoc["message"][]=database_get_message("error_saving");
        return false;
      }
      $_SESSION["$webdoc[main_table]_record_saved"]=true;
      $url="./?subpage1=test_run&subpage2=list";
      header("location: $url");
```

Zobrazení celého běhu testu

Na úvodní straně jsou zobrazeny pouze informace identifikující daný běh testu. Pro zobrazení kompletního výpisu je třeba kliknout na odkaz `View`, který je uveden pro každý záznam zvlášť. Skriptu je zaslán identifikátor běhu testu, `test_run_id`, pomocí kterého je proveden dotaz do databáze k získání příslušných dat. Tato data jsou následně uspořádána do pole, které je předáno k vykreslení Smarty šabloně.

Editace vytvořeného běhu testu

Oproti části Test není v Test Run třeba archivovat změny. Proto se na tomto místě jedná o skutečnou editaci záznamu v databázi. Může ji provádět uživatel s jakýmkoli oprávněním. Po kliknutí na odkaz editace u příslušného záznamu je zobrazeno formulářové pole s přednastavenými hodnotami dle daného výběru. Uživatel provede potřebné úpravy. Po kliknutí na tlačítko editace je opět kontrolováno, zda jsou vyplněny povinné položky `Name` a `Setup`. Jestliže ano, je zavolána funkce

`database_save_record_test` s parametrem `$zaznam["edit"]`. Tato funkce provádí prostou aktualizaci záznamu s příslušným identifikačním číslem běhu testu, `test_run_id`. Poté dojde k přesměrování na výpis existujících běhů testu spolu s hláškou o úspěšném uložení změn. Jestliže se během této operace vyskytla chyba, je vyvolána příslušná chybová hláška.

Kód 5.5: Výpis části funkce `database_save_record_test`, která provádí editaci běhu testu

```
if (isset ($record["edit"])) // jedna se o editaci?
    {
      if (!database_query("UPDATE $webdoc[main_table] SET
      name='$record[name]', sw_rev='$record[sw_rev]',
      hw_rev='$record[hw_rev]', setup='$record[setup]',
      result='$record[result]', duration='$record[duration]',
      description='$record[description]' WHERE
      test_run_id=$record[test_run_id]")) // provedeni update
      {
        $webdoc["message"][]=database_get_message("error_saving");
        return false;
      }
      $_SESSION["$webdoc[main_table]_record_saved"]=true;
      $url="./?subpage1=test_run&subpage2=list";
      header("location: $url");
    }
```

V rámci editace existují dva další stavy. První nastane v případě, kdy je běh testu schválen. Je počítáno s tím, že jsou položky jako jméno běhu, jeho nastavení atd. dostatečně optimalizované a tím pádem probíhá praktická realizace. Je zobrazena informační hláška „`Test Run was approved. Is possible to edit Result, Duration and Description.`“ a je tedy umožněno editovat pouze uvedené položky. Jedná se o data informující o výsledku běhu testu a o další připomínky. Druhý nastane v případě, kdy je Test Run schválen a proběhlo jeho otestování, přičemž výsledky jsou zaznamenány a již se nebudou dále měnit. Uživatel označí běh testu za ukončený (otestovaný). Tento záznam již proto nelze dále editovat, uživatel je o tom informován hláškou „`Impossible to edit Test Run. Testing was ended.`“.

Kopírování vytvořeného testu

V rámci stanovení nastavení běhu testu nemusí vždy dojít k uspokojivým výsledkům. Proto jsem vytvořil možnost Test Run okopírovat, tzn. vytvořit nový záznam, který bude v prvopočátku obsahovat stejná data, nebude ovšem schválen ani

označen za ukončený, takže půjdou tato data libovolně editovat. Daný běh testu se ovšem bude nadále týkat příslušného testu či produktu. Při kliknutí na odkaz `Copy` je tedy zobrazen úplný výpis daného běhu testu, který je po stisknutí tlačítka `Copy Test Run` vložen do databáze pod novým identifikačním číslem, přičemž jsou hodnoty pro schválení a otestování vynulovány. K uložení je opět zavolána PHP funkce `database_save_record_test`.

Schválení běhu testu

Je úlohou pro DA, DE či TA. Po vytvoření Test Run je třeba jej schválit, aby bylo možno započít jeho praktickou realizaci a zapsat výsledky. Schválený běh testu je možné omezeně editovat, viz. odstavec o editaci.

Po kliknutí na odkaz `Approving` je nejprve zobrazena hláška informující uživatele, zda je, nebo není Test Run schválen. Pokud dosud nebyl, je zobrazeno tlačítko, jehož odesláním je zavolána PHP funkce `database_approve_record`, která jej schválí. Ve zbytku stránky je zobrazen úplný výpis záznamu, aby uživatel věděl přesně, co schvaluje.

Označení ukončeno

V okamžiku, kdy tester ukončí praktickou realizaci běhu testu a zapíše výsledky, může označit záznam jako ukončený. Musí být ovšem předtím schválen. Toto právo má každý uživatel aplikace. S takto označeným záznamem není možné dále manipulovat. Lze jej pouze prohlížet a kopírovat.

Stejně jako v případě schválení je po kliknutí na odkaz `Set Tested` zobrazena hláška informující uživatele o stavu záznamu, případně zobrazeno tlačítko umožňující označení běhu testu jako ukončený. Je zavolána PHP funkce `database_tested_record`, která provede u příslušného záznamu update v oblasti databázových sloupců `tester_id` a `tested`.

Kód 5.6: Výpis funkce `database_tested_record` označující záznam jako ukončený

```
function database_tested_record($record_id)
{
  global $webdoc;
  return database_query("UPDATE $webdoc[main_table] SET tested=now(),
  tester_id='$_SESSION[_auth]' WHERE
  $webdoc[main_table]_id=$record_id");
} // update db za ucelem oznaceni jako ukonceny (otestovany)
```

6 ZÁVĚR

V diplomové práci jsem se zabýval praktickým řešením části aplikace pro správu vývojové dokumentace, určenou pro firmu Honeywell. Řešení navazuje a vychází z předchozího teoretického návrhu, zpracovaného v semestrálním projektu.

Základní funkcí aplikace je vložení nového, nebo modifikace již existujícího požadavku na vývoj určitého produktu. Následuje vytvoření tohoto produktu. Proces je ukončen sestavením a realizací testu, který ověří funkčnost a správnost návrhu.

Mým cílem bylo nejprve osobními konzultacemi se zadavatelem a spolupracujícím kolegou vytvořit harmonogram práce a zjistit technické požadavky. Celý systém je založen na šablonovací systému Smarty, který jsem se naučil obsluhovat a vytvářet příslušné šablony pro vznikající skripty. Tímto byl splněn základní požadavek na oddělení programového kódu jazyka PHP od statických HTML dokumentů.

Při zpracování diplomové práce jsem nejprve vytvořil hlavní obslužné algoritmy pro připojení k databázi. Dále jsem se popsal samotnou práci s databázovým systémem, kde jsem vysvětlil, jak se k databázi připojit, spolu se základními příkazy pro vložení, editaci, mazání a výpis záznamů. V návaznosti jsem řešil, jak přehledně zobrazovat výsledky dotazů do databáze. Vytvořil jsem knihovnu pro šablonovací systém Smarty `view_result`, která po zadání určitých parametrů provede výpis dat spolu s požadovanými odkazy pro následnou práci s daty. Velký prostor jsem věnoval možnostem autorizace uživatelů do aplikace, protože to považuji za důležité. Popsal jsem známé postupy a vytvořil vlastní, který je jejich kombinací. V rámci aplikace jsem založil oddělenou strukturu, která se zabývá výpisem, editací a vkládáním nových uživatelů, spolu s možností změny vlastního hesla. Dále jsem se soustředil na zobrazování formulářových prvků. Řešením je opět Smarty funkce, protože formuláře jsou vkládány ve statických dokumentech, které jsou generovány právě šablonovacím systémem. Výsledkem je knihovna `insert_form`, která dle zadaných vstupů vygeneruje příslušný formulář. Do aplikace jsem zavedl WYSIWYG formulář pro snadnou úpravu při vkládání rozsáhlejších textů. Posledním významným algoritmem bylo vkládání souborů k záznamům v SQL tabulkách.

Vytvořené knihovny jsem využil v databázové části aplikace. Nejprve jsem popsal samotnou strukturu databáze, kterou řeší kolega, potom realizoval návrh na části

Test, Test Run a Person. Výsledkem je funkční prostředí, plnící všechny požadované úkoly, spolupracující s ostatními částmi.

Jelikož byla realizace aplikace rozdělena na dvě části, včetně podpůrných algoritmů, nelze provést striktní rozdělení práce. Kolega využíval ve svých databázových strukturách mých knihoven a naopak. Vyzkoušel jsem si tím práci v týmu, která se mi osvědčila.

Výsledkem této diplomové práce je kompletní realizace zadání, splnění všech cílů při vytváření algoritmů i databázových struktur. Současně s textem odevzdávám na přiloženém CD všechny zdrojové kódy se zálohou databáze. Díky tomu lze velmi jednoduše spustit tuto aplikaci na libovolném webovém serveru. Po určitou dobu je také spuštěno demo na adrese `http://honeywell.macocha.net/www`, kde je možnost na výsledek práce nahlédnout.

POUŽITÁ LITERATURA

[1] Jesus Castagnetto, Harish Rawat, Sascha Schumann, Chris Scyllo, Deepak Veliath, Programujeme PHP - profesionálně, Computer Press, Brno, 2004, ISBN 80-7226-310-2

[2] Milan Šimůnek, SQL – kompletní kapesní průvodce, Grada, Praha, 1999, ISBN 80-7169-692-7

[3] Jiří Kosek, PHP – Tvorba interaktivních internetových aplikací, Grada, Praha, 1998, ISBN 80-7169-373-1

[4] Interval.cz - Internetový magazín o webdesignu, www.interval.cz [online] 2008 [cit. 2008-05-07], ISSN: 1212-8651

[5] Smarty - documentation, www.smarty.net [online] 2008 [cit. 2008-05-07]

[6] Smarty, http://smarty.ronnieweb.net [online] 2008 [cit. 2008-05-07]

[7] Jiří Bráza, PHP 4 – Praktické příklady, Grada, Praha, 2003, ISBN 80-247-0441-2

[8] Frank Boumphrey, Cassandra Greer, Dave Raggett, Jenny Raggett, Sebastian Schnitzenbaumer, Ted Wugofski, XHTML – Průvodce vývojáře, Wrox Press, Praha, 2002, ISBN 80-86593-14-2

[9] TinyMCE – documentation, http://wiki.moxiecode.com/index.php/TinyMCE:Index [online] 2008 [cit. 2008-04-12]

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma Správa vývojové dokumentace přes WWW I 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 26.05.2008 .....

podpis autora

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce z firmy Honeywell Ing. Radomíru Svobodovi, Ph.D. za cenné rady plynoucí z jeho zkušeností ze zpracování podobných projektů.

V Brně dne 26.05.2008 .....

podpis autora

PŘÍLOHY

PŘÍLOHA 1: výpisy programových kódů knihoven

PŘÍLOHA 2: blokové schéma struktury databáze