UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

OSOBNÉ ÚČTOVNÍCTVO NA WEBE

2012 Pavel Melo

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

OSOBNÉ ÚČTOVNÍCTVO NA WEBE

Bakalárska práca

BRATISLAVA 2012

Pavel Melo

Študijný program:	Aplikovaná informatika
Študijný odbor:	1.1.1 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Mgr. Pavel Petrovič, PhD.
Konzultant:	—
Evidenčné číslo:	be495a24-3eaa-4e14-972e-db5458c32c73

Čestne vyhlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

Bratislava, Máj 2012

.....

Na tomto mieste sa chcem poďakovať vedúcemu mojej práce, Mgr. Pavlovi Petrovičovi, PhD., za jeho ochotu, rady i jeho trpezlivosť.

Obsah

Kapitola 1

Úvod

1.1 Osobné účtovníctvo na webe

V posledných rokoch sa vo vyspelom svete život veľmi zrýchlil. Každý deň nás obklopuje množstvo informácií, využívame veľa služieb, kupujeme si oveľa viac produktov, ako pred rokmi. Táto zmena v našom životnom štýle má okrem svojich výhod aj svoj negatívny dopad na naše financie. V množstve informácií sa orientujeme čoraz horšie, postupne strácame prehľad o našich financiách, o príjmoch, ale hlavne o výdavkoch. Ako riešenie na tento problém existujú rôzne aplikácie ponúkajúce osobné účtovníctvo. V mojej práci sa budem snažiť takúto aplikáciu navrhnúť a implementovať, dbajúc o prehľadnosť, jednoduchosť ovládania a bezpečnosť.

Kapitola 2

Východiská

V tejto časti sa budem venovať cieľom práce, požiadavkam na výslednú aplikáciu, ktoré som bral do úvahy pri plánovaní, návrhu aj implementácii aplikácie. Ďalej spomeniem existujúce aplikácie, desktopové aj webové, ktoré ponúkajú používateľom osobné účtovníctvo. Neskôr popíšem architektúru MVC, moderný a v súčasnosti veľmi rozšírený spôsob stavby aplikácií. Predstavím návrhové vzory, bez ktorých sa pri implementácii nezaobídem, taktiež technológie, ktoré pri nej využijem. Ukážem aj, ako je možné vyhnúť sa jednej z najrozšírenejších zraniteľností webových aplikácií napísaných nielen v php, a to SQL injection.

2.1 Ciele práce

Cieľom mojej práce je pomocou moderných webových technológii navrhnúť a implementovať internetovú aplikáciu pre správu osobného účtovníctva.

2.2 Špecifikácia požiadaviek

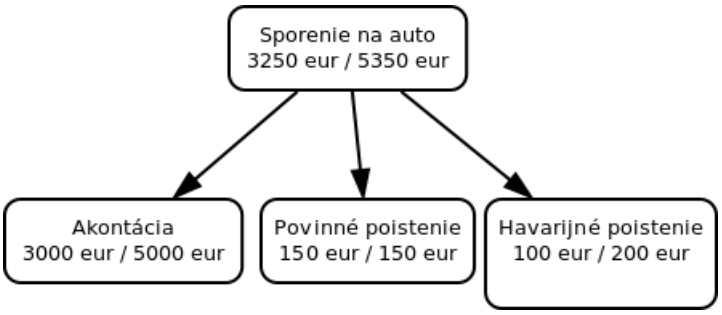
2.2.1 Hlavné účty

Hlavné účty reprezentujú skutočné účty používateľa. Používateľ si môže vytvoriť akékoľvek množstvo hlavných účtov. Tieto účty obsahujú informáciu o aktuálnom množstve peňazí na nich. Predstavujú jedinú kategóriu účtov v aplikácii, ktoré obsahujú údaje o peniazoch. Z nich sú peniaze prerozdeľované do virtuálnych účtov.

2.2.2 Virtuálne účty (sporenia)

Používateľ si môže v aplikácii vytvoriť tzv. virtuálne účty. Nie sú to skutočné účty, nie je možné im zmeniť aktuálny zostatok. Je možné ale na tieto virtuálne účty z hlavných účtov rezervovať peniaze a nastaviť cieľovú sumu, ktorú si chce používateľ nasporiť. Ak má používateľ cieľ, na ktorý si chce nasporiť istú sumu peňazí, dobrým riešením je vytvorenie virtuálneho účtu. Nastaví cieľovú sumu, ktorú potrebuje nasporiť, a postupne si môže preúčtovať peniaze zo svojich hlavných účtov. Kedykoľvek je možné skontrolovať, koľko peňazí už je rezervovaných pre dané sporenie.

Virtuálne účty sú organizované do stromovej štruktúry, vnútorné vrcholy predstavujú kategórie účtov, listy sú účty samotné. Ako pre účtysamotné, aj pre každú kategóriu je možné zistiť množstvo už nasporených peňazí.



Obr. 2.1: Stromová štruktúra virtuálnych účtov a ich aktuálne a cieľlové zostatky

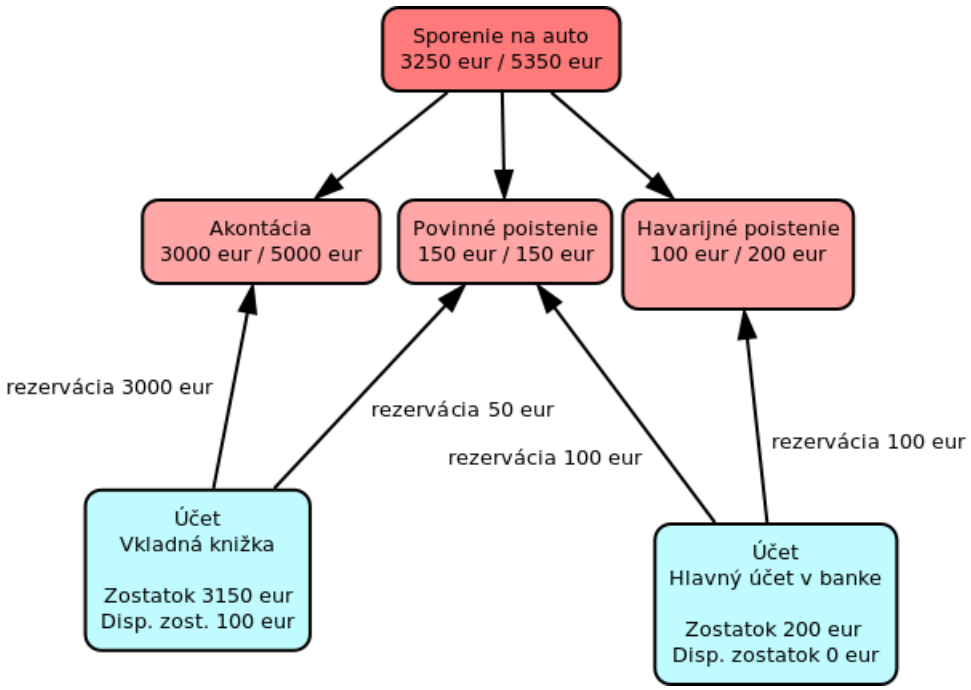
2.2.3 Príjmy a výdavky

Hlavné a virtuálne účty postačujú na reprezentáciu rozloženie peňazí používateľa. Nepostačujú však na reprezentáciu peňažných tokov od neho (výdavky) alebo k nemu (príjmy). Tomuto slúžia príjmové a výdavkové účty. Neobsahujú žiadne informácie o množstve peňazí, slúžia výhradne ako zdroj peňazí alebo cieľ transakcií.

Obr. 2.2: Príjmy a výdavky vo vzťahu k hlavnému účtu

2.2.4 Rezervácie

Rezervácie sú záznamy, ktoré tvoria vklady na virtuálne účty. Každá rezervácia spája jeden hlavný a jeden virtuálny účet. Obsahuje v sebe sumu peňazí, ktorá je z daného havného účtu rezervovaná pre virtuálny účet. Ak chceme zistiť aktuálny stav na účte (či už hlavnom, alebo virtuálnom), je nutné zosumovať rezervácie patriace k danému účtu a túto sumu pripočítať, resp. odpočítať od stavu účtu.



Obr. 2.3: Príklad rezervácií, teda vzťahov medzi hlavnými a virtuálnymi učtami

2.2.5 Príkazy

Presne ako v banke, aj v aplikácii je možné na útoch nastaviť jednorazové aj trvalé príkazy. Peniaze je možné preúčtovať nielen medzi jednotlivými účtami (či už hlavnými, alebo virtuálnymi), ale určiť aj príjmy a výdavky na hlavných účtoch. Znamená to, že používateľ si môže v aplikácii nastaviť svoj pravidelný príjem a výdavky, ale systém umožňuje aj jednorázové prevody. Takisto sa dajú vytvoriť jednorázové, aj trvalé prevody medzi účtami (napr. sporenie na auto reprezentované virtuálnym účtom).

2.3 Existujúce riešenia

Pred začatím práce na projekte som preskúmal rôzne aplikácie ponúkajúce osobné účtovníctvo, webové aj desktopové.

Desktopové aplikácie Mojím prvým krokom bolo preskúmanie alternatív medzi desktopovými aplikáciami. Bohužiaľ, na žiadnu zadarmo použiteľnú aplikáciu som nenarazil. Spoplatnené aplikácie v slovenskom jayzku, ktoré som našiel, sú

- Domáce účtovníctvo FILIP 2009 - podľa popisu aplikácia ponúka domáce účtovníctvo, evidenciu hotovosti, bankových účtov, majetku aj sadzby energií, podporuje zobrazenie štatistík za určité časové obdobia v podobe grafov.
- RUDo - Rodinné a osobné Účtovníctvo na Doma - podporuje evidenciu príjmov, výdavkov, pohľadávok, štatistiky vo forme grafov, import a export údajov.

Webové aplikácie

mint.com Mint je zadarmo dostupný webový účtovný softvér. Avšak v čase môjho prieskumu mal ako podmienku použitia aj zadanie prihlasovacích údajov do internetbankingu používateľa. Aj keď tieto prístupové údaje používa len na získanie údajov o účtoch a transakciách používateľa, táto skutočnosť pravdepodobne väčšinu používateľov od registrácie odradí. Navyše mnou preverované slovenské banky v čase prieskumu túto službu nepodporovali.

Výhody

- je zadarmo
- automaticky spracúva a kategorizuje transakcie používateľa
- podporuje emailovú aj sms notifikáciu o pohybe na účtoch

Nevýhody

- požaduje prístup k internetbakingu používateľa
- je dostupný len v anglickom jazyku

- nemá podporu slovenských bánk, bez ktorej je nepoužiteľný

money.strands.com Podobne ako mint.com, aj táto webová aplikácia je použiteľná bez poplatku. Na rozdiel od predchádzajúcej aplikácie však nepožaduje pri registrácii bankové údaje používateľa. Podporuje tvorbu virtuálnych aj hlavných účtov. Pomocou sprievodcu dokáže používateľ vo viacerých krokoch generovať štatistiky pre rôzne časové obdobia a kategórie.

Za zlú vlastnosť aplikácie považujem neprehľadnú tvorbu jednorazových aj trvalých príkazov. Pri prevode peňazí totiž nie sú výdavky a príjmy jasne odlíšené, peniaze sa naúčet pripíšu alebo z neho odčitajú podľa toho, či druhý účet patrí do kategórie príjmov ale výdavkov.

Výhody

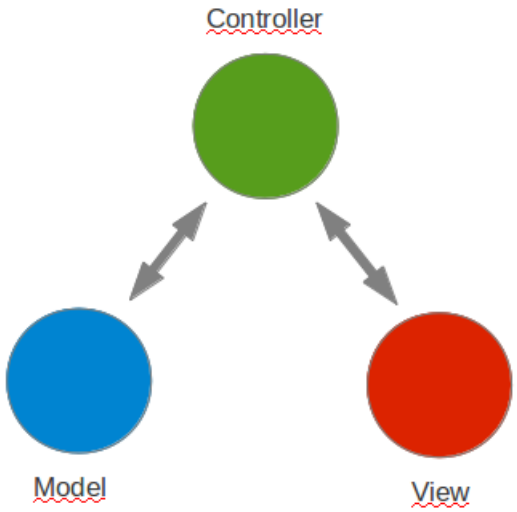
- je zadarmo
- virtuálne účty
- tvorba štatistík podľa rôznych nastavení

Nevýhody

- neprehľadné zadávanie príkazov
- hierarchia virtuálnych účtov je obmedzená na dve úrovne

2.4 Architektúra MVC

Model-View-Controller architektúra je moderný spôsob stavby aplikácií, v ktorom sa oddelujú časti zodpovedné za prácu s údajmi (model), logiku (controller) a za zobrazenie údajov (view). MVC je známe hlavne vďaka frameworkom pre webovské aplikácie.



Obr. 2.4: Vzťah medzi časťami MVC - modelom, view a controllerom

Princíp fungovania MVC Po spustení aplikáce s MVC arhitektúrou sa na základe vstupných parametrov (tými môžu byť parametre získané z príkazového riadku, v našom prípade to bude URL stránky) vyberie správny controller, v rámci neho správna akcia. Akcia získa potrebné údaje z databázy prostredníctvom modelu, spracuje ich (prípadne zmení). Údaje, ktoré sa majú zobraziť, sa priradia šablóne (view). Po zbehnutí akcie view spracuje jemu zadané údaje a zobrazí výstup.

Beh aplikácie s MVC architektúrou sa teda podobá behu akejkoľvek inej aplikácie. Hlavný rozdiel medzi nimi je, že kým v aplikáciách nepostavených na MVC frameworku môžeme miešať v jednom zdrojovom súbore aplikačnú logiku, prácu s databázou aj generovanie výstupu, MVC frameworky nám umožňujú oveľa pohodlnejšie rozdelenie aplikácie na tieto tri časti (v niektorých prípadoch až toto rozdelenie nanútia) .

Výhody MVC architektúry

- MVC frameworky nútia programátora rozdeliť aplikáciu na na model, view a controller, výslekdom jeho práce je teda zvyčajne kvalitnejší a prehľadnejší kód.
- Oddelenie view od ostatných častí umožňuje na danej časti apikácie programátorovi aj grafikovi súčasne. Kým programátor sa stará o aplikačnú logiku, grafik môže navrhnúť a realizovať rozhranie pre používateľa.
- Model zjednodušuje komunikáciu s databázovým systémom a zvyčajne ošetruje vstupy, čo slúži ako prevencia pred útokmi typu SQL injection.
- MVC frameworky napísané v php sú väčšinou optimalizované pre programovanie webových aplikácií, podporujú spracovanie zadaných url, generovanie url, presmerovania používateľa na iné stránky. Často majú v sebe zabudovanú podporu pre ajax alebo webservisy.

Existujúce MVC frameworky Existuje viacero frameworkov pre jazyk php, ktoré ponúkajú jednoduchú prácu s MVC architektúrou. Najznámejšie z nich sú:

- CakePHP - jeden z najznámejších php MVC frameworkov. Používa objektovo orientovaný prístup, ponúka veľké množstvo funkcií. View sa píšu v php, nepoužíva vlastný šablónovací jazyk. Výhodou je veľká komita, ktorá tento framework podporuje, nevýhodou sú striktné pravidlá, podľa ktorých treba pomenovať triedy modelov i controllerov.
- Zend Framework - php framework od spoločnosti stojacej v centre diania okolo vývoja php. Ponúka veľmi širokú funkcionalitu, podobne, ako CakePHP, je podporovaný veľkou komunitou. Jeho nevýhodou je neobjektový prístup k programovaniu v jeho prvej verzii.
- Nette Framework - na Slovensku a v Česku pomerne rozšírený php framework českého pôvodu. Je jednoduchý na použitie, využíva metódy objektovo orientovaného programovania. Má v sebe integrovaný vlastný šablónovací systém Latte, ktorý vĎaka svojej pokročilej funkcionalite umožňuje jednoduché a rýchle písanie šablón pre view. Integrovanej podpore ajaxových dopytov zjednodušuje aj tvorbu pokročilých funkcií a efektov používateľského prostredia aplikácie. Nevýhodou tohto frameworku je malá rozšírenosť a podpora v celosvetovom meradle, menšia komunita vývojárov aj slabá dokumentácia frameworku.

2.4.1 Model

Model zodpovedá za prácu s údajmi v databáze.

Obyčajne každá tabuľka v databáze reprezentuje objekty jedného typu (napr. používatelia, produkty). Ak chceme pridať do tabuľky nový záznam, musíme databázovému servru odoslať dopyt tvaru:

`INSERT INTO tabulka (stlpec1, stlpec2, ..., stlpecN) VALUES (hodnota1, hodota2, ..., hodnotaN)`,

podobne pri získavaní údajov z databázy a ich zmene.

Používanie priameho prístupu k SQL databáze môže byť výhodné v prípade, ak dopytov nie je veľa. Ak je ale nutné do databázy pristupovať často a rôznymi spôdobmi, je výhodnejšie vytvoriť vrstvu, ktorá ponúka jednoduchší prístup k objektom v databáze. Práve model predstavuje túto vrstvu. Každú tabuľku obaľuje jedna trieda, jedna inštancia z tejto triedy reprezentuje riadok danej tabuľky. Nastavovaním vlastností týchto objektov sa menia údaje v príslušnom riadku tabuľky, vďaka modelu teda nie je nutné pre prácu s databázou písať zložité SQL dopyty.

Nespornou výhodou použitia modelu aj jednoduchá zmena štruktúry tabuliek. V prípade, že do tabuľky pridáme stĺpec, alebo ho naopak odoberieme alebo premenujeme, stačí v modeli ten stĺpec takisto pridať, odstrániť alebo premenovať. V prípade ručne písaných SQL dopytov by bolo potrebné v každom dopyte zmeniť zoznam stĺpcov, s ktorými sa pracuje.

Nevýhodou jednoduchého modelu je, že väčšinou podporuje len obyčajné dopyty na databázu bez agregácie alebo iných, pokročilých funkcií. V prípade, že programátor potrebuje využívať agregačné funkcie, alebo join tabuliek, často musí siahnuť po ručne napísaných SQL dopytoch.

Niektoré databázové modely podporujú aj pokročilé funkcie (agregácia, join tabuliek, rekurzívne dopyty), ich nevýhoda však bežne spočíva v náročnosti ich použitia. Pri výbere modelu pre aplikáciu je preto vhodné prijať kompromis jednoduchosťou a šírkou funkcionality modelu .

2.4.2 View

View je časť aplikácie zodpovedná za zobrazovanie údajov. Zvyčajne ide o šablónu napísanú v jazyku php, alebo v inom, vlastnom šablónovacom jazyku. Takýmito jazykmi sú:

- Smarty - samostatná šablónovacia knižnica používajúca šablónovací jazyk trochu pripomínajúci php.
- Latte - šablónovací jazyk MVC frameworku Nette.
- Fluid - šablónovací jazyk frameworku FLOW3, je používaný aj frameworkom extbase, ktorý je backportom frameworku FLOW3 vytvoreným pre CMS systém TYPO3.

Výhodou týchto jazykov je pomerne jednoduchá syntax. Sú špecializované na zobrazovanie údajov. Nepodporujú teda nastavovanie premenných ani zmenu stavu objektov, pri vytváraní aplikácie sme teda nútení akékoľvek zmeny stavu vykonávať v kontroleri a šablónu sutočne využívať iba na zobrazovanie údajov. Na druhej strane jednoduchosť o obmedzená funkcionalita týchto jazykov sa môže prejaviť ako nevýhoda, keď nepodporujú programátorom požadované funkcie. Väčšinou ale šablónovacie systémy ponúkajú aj mechanizmy na ich rozšírenie v prípade potreby.

Spoločným znakom týchto šablónovacích systémov je, že šablóny prekladajú do jazyka php, keďže ich spracovanie pri každom zobrazení aplikácie by bolo výpočtovo náročné (staršie verzie šablónovacieho systému Fluid v CMS TYPO3 spracovávali šablóny pri každom zobrazení stránky, no od verzie 4.6 sa už aj tieto šablóny prekladajú do php).

2.4.3 Controller

Controller je časť systému, ktorá sa stará o aplikačnú logiku. Jeho úlohou je spracovať vstupy, pomocou modelu načítať, prípadne zmeniť a uložiť údaje do databázy a šablóne nastaviť údaje, ktoré sa majú zobraziť.

Controller je zvyčajne trieda odvodená od abstraktnej triedy Controller. Controllery sú v podstate sady akcií (funkcií), ktoré môže aplikácia vykonať. Tieto akcie môžu byť zaradené do controllerov podľa rôznych kritérií, napríklad akcie starajúce sa o správu účtov môžu patriť jednému controlleru. Toto rozdelenie by malo byť pre programátora prehľadné, aby sa lepšie orietoval v zdrojových kódoch.

Akcia je reprezentovaná funkciou controllera. Názov funkcie zvyčajne musí zodpovedať vzoru nazovAkcieAction(). Ku každej akcii patrí aspoň jedna šablóna, do ktorej akcia naviaže údaje, ktoré sa majú zobraziť. Niektoré akcie nepotrebujú komunikovať s databázou, teda môžu, ale nemusia pracovať s modelom.

2.5 Návrhové vzory

Programátori sa počas návrhu a vývoja aplikácií často stretávajú s rovnakými alebo aspoň podobnými problémami. V prípade objektovo orientovaného programovania často ide o problémy so zdieľaním údajov medzi jednotlivými časťami aplikácie, využívaním tých istých funkcií na podobné úlohy alebo skrývaním premenných a funkcií pred niektorými časťami kódu.

Pre veľké množstvo problémov a úloh existujú štandardné riešenia, ktoré sa dajú pri návrhu aplikácie využiť. Takýmito riešeniami sú aj rôzne návrhové vzory, ktoré sa používajú v objektovo orientovanom programovaní. Je nutné pripomenúť, že pre riešenie každého problému treba vybrať vhodný návrhový vzor, ktorý daný problém rieši. Tiež je dobré uvedomiť si, že návrhové vzory nie sú riešením na všetky problémy, ktoré sa pri programovaní vyskytnú, teda niekedy je vhodnejšie siahnuť po vlastnom riešení danej úlohy.

Pri návrhu a implementácii sa nezaobídem bez návrhových vzorov jedináčik (singleton), továreň (factory) a prepravka (crate).

2.5.1 Jedináčik

Jedináčik špecifikuje, ako vytvoriť triedu, ktorá bude mať najviac jednu inštanciu. Táto inštancia nemusí byť inštanciou danej triedy. [1]

Jedináčika je dobré použiť, ak je výhodné alebo nutné zabezpečiť, aby z triedy bolo možné vytvoriť najviac jednu inštanciu. Dôvodom môže byť pamäťová alebo časová náročnosť vytvorenia novej inštancie, ale aj potreba zdieľať danú inštanciu triedy medzi rôznymi časťami aplikácie. Ako dobrý príklad môže poslúžiť spojenie s databázou. Pripojiť sa k databáze je počas behu aplikácie odporúčane iba raz. Potom je ale potrebné toto spojenie sprístupniť vo všetkých častiach aplikácie. Návrhový vzor jedináčik ponúka na tieto požiadavky jednoduché riešenie.

Implementácia Vytvorenie triedy jedináčika je jednoduché, trieda musí disponovať nasledujúcimi súčasťami:

- Súkromný konštruktor, aby sme zvonku nevedeli vytvárať nové inštancie triedy
- Súkromá statická premenná - inštancia triedy
- Verejná funkcia instance(), ktorá vytvorí inštanciu triedy (ak ešte nebola vytvorená), uloží ju do statickej premennej a túto inštanciu aj vráti.

```
class Singleton {
  protected static $instance = null;
  protected function __construct() ...

  public static function instance() {
    if (is_null(self::$instance)) {
      self::$instance = new self();
    }
    return self::$instance;
  }

}
```

2.5.2 Továreň (factory)

Deklaruje rozhranie s metódou pre získanie objektu. Rozhodnutie o konkrétnom type vráteného objektu však ponecháva na svojich potomkov, tj. na prekrývajúcich verziách deklarovanej metódy. [1]

Výhodou návrhového vzoru továreň je, že pri vytváraní inštancie triedy nemusíme vedieť, konkrétne o akú inštanciu ide. O výber správnej inštancie sa stará práve továrenská trieda, ktorá na základe parametrov, aktuálneho stavu systému alebo nastavení vyberie, prípadne vytvorí vhodný objekt (inštanciu) a vráti ju. Ďalšou výhodou továrne je, že umožňuje kontrolovať počet vytvorených inštancií, v prípade potreby si inštacie si môže ukladať do pamäte ale cache.

```
class Factory {
  protected static $instance = null;

  public static getObject($param) {
    if ($param == 0) {
      $instance = new Class1();
    } else {
      $instance = new Class2();
    }
    return $instance;
  }

}
```

2.5.3 Prepravka

Objekty triedy implementovanej podľa návrhového vzoru prepravka umožňujú uskladnenie údajov a ich podávanie ďalej vo forme jediného parametra funkcie. Takýto prístup k predávaniu parametrov má viacero výhod oproti iným metódam (asociatívne pole, viac parametrov funkcie):

- parametre je možné funkcii predávať vo forme jediného objektu, prípadne viacerých objektov (ak predstavujú parametre rôzne logické celky), vďaka tomu je táto metóda prehľadnejšia ako predávanie každého parametra zvlášť
- parametre je možné ochrániť pred zmenou, ak sú uložené v chránených alebo súkromných premenných prepravky (na rozdiel od poľa, v ktorom údaje nie sú nijakým spôsobom chránené)

```
class Person {
  protected $firstName;
  protected $lastName;

  public function __construct($firstName, $lastName) {
    $this->firstName = $firstName;
```

```
        $this->lastName = $lastName;
    }

    public function getFirstName() {
        return $this->firstName;
    }

    public function getLastName() {
        return $this->lastName;
    }

}
```

2.6 Použité technológie

Rozhodol som sa, že aplikáciu naprogramujem v skriptovacom jazyku php. Php je jazyk vytvorený a vyvýjaný primárne pre programovanie webových aplikácií. Je najviac rozšírený v oblasti webových aplikácií a zároveň ide o jeden z najpopulárnejších programovacích jazykov vôbec. Jeho nevýhodou je, že jeho slabá a dynamická typovosť často zvádza neskúsených programátorov k nesprávnym programátorským zvykom a postupom.

Pre aplikáciu som si vybral php verzie 5.3, ktorá už podporuje menné priestory.

Pre ukladanie údajov som si vybral relačnú databázu MySQL, menšími webovými aplikáciami najčasťejšie používaný databázový systém.

Používateľské rozhranie je napísané v jazyku html5, s použitím CSS a javascriptu. V aplikácii som sa rozhodol použiť sadu css a javascriptových funkcií Twitter Bootstrap. Tento balíček uľahčuje tvorbu používateľského rozhrania tým, že implementuje rôzne komponenty použiteľné na webových stránkach (tlačidlá, zoznamy, rozbaľovacie ponuky, formuláre).

Pre aplikáciu som sa rozhodol vytvoriť vlastný MVC framework.

2.7 Bezpečnosť

Pri vývoji aplikácií, najmä verejne prístupných webových aplikácií, by mala mať bezpečnosť veľmi vysokú prioritu. Znamená to, že návrh, ale aj implementácia by mali zabezpečiť, aby používatelia nemohli v aplikácii vykonávať žiadne akcie, pre ktoré nemajú oprávnenie, či už nevedomky, alebo zámerne.

Existuje viacero metód, pomocou ktorých môžeme útočníkovi zabrániť v prístupe k údajom, alebo vo vykonávaní činností, na ktoré nemá oprávnenie. Niektoré metódy spočívajú v správnych programátorských postupoch, niektoré v správnej architektúre aplikácie, niektoré v dôslednej kontrole vstupov od používateľa.

2.7.1 SQL Injection

SQL injection je metóda, pomocou ktorej útočník dokáže manipulovať SQL dopyty odosielané databázovému systému, a tým neoprávnene získať prístup k údajom v databáze alebo manipulovať ich. Aplikácia môže byť zraniteľná voči tejto metóde, ak vstupy od používateľa bez vhodného spracovania alebo filtrácie vkladá do SQL dopytu. Tejto zraniteľnosti sa dá veľmi jednoducho zabrániť, no napriek tomu je jednou z najrozšírenejších zraniteľností webových aplikácií vôbec.

2.7.2 Príklad zneužitia zraniteľnosti voči SQL injection

Nasledujúci príklad demonštruje, ako funguje metóda SQL injection.

```
$username = $_GET['username'];
$password = $_GET['password'];
$sql = 'SELECT␣*␣FROM␣users
␣␣WHERE␣username␣=␣"' . $username . '"
␣␣and␣password␣=␣"' . $password . '"';
$result = mysql_query($sql);
```

Je vidno, že vstupy, teda parametre username a pasword sa do SQL príkazu vkladajú bez akéhokoľvek spracovania. Útočník môže túto skutočnosť zneužit a do parametra name vložiť reťazec so špeciálnymi znakmi:

```
admin"␣%
```

Ak tento parameter vložíme do SQL príkazu, výsledný príkaz bude

```
SELECT * FROM users WHERE
  username = "admin" %  and password = ""
```

Keďže znak ükončí reťazec a % označuje v SQL komentár, skontroluje sa len prvá podmienka a druhá sa ignoruje. Týmto spôsobom sa útočník môže do systému prihlásiť do akéhokoľvek účtu. Ak navyše databáza podporuje vykonávanie viacerých príkazov SQL v jednom dopyte, namiesto znaku % môže útočník použiť ; a zadať ďalší SQL príkaz. Takýmto spôsobom môže vykonať akúkoľvek operáciu, ktorá je v danej databáze povolená.

Vo väčšine prípadov má používateľ databázy povolené všetky operácie, teda útočník môže zmeniť, prípadne aj vymazať všetky údaje v databáze.

2.7.3 Filtrácia vstupu

Jedným z riešení, ako predísť útoku pomocu SQL injection, je vhodné spracovanie vstupov pred vytvorením SQL dopytov. Na takéto spracovanie vstupu slúži v php funkcia mysql_real_escape_string($str), ktorá vstupný reťazec upraví tak, aby všetky zanky špeciálne v jazyku SQL boli neutralizované, teda po vložení do SQL dopytu sa tieto znaky budú správať ako súčasť reťazca, nie ako špeciálne znaky.

Nevýhodou tohoto prístupu je, že programátor môže zabudnúť na predspracovanie vstupu a tým umožniť útočníkovi SQL injection.

```
$parameter = $_GET['param'];
$filteredParameter = mysql_real_escape_string($parameter);
$sql = 'SELECT * FROM table WHERE column = ' . $parameter;
$result = mysql_query($sql);
```

2.7.4 Viazanie premenných do predpripravených dopytov

Najodporúčanejšou metódou na vkladanie parametrov do SQL dopytov je používanie viazania (binding) premenných do predpripravených dopytov (prepared query). V tomto prípade sa vytvorí SQL dopyt, ktorý namiesto parametrov obsahuje ozázniky. Tento dopyt sa odošle serveru, ktorý ho spracuje. Do tohto spracovaného dopytu sa potom naviažu želané parametre.

```
$parameter = $_GET['param'];
$sql = 'SELECT * FROM table WHERE column = ?';
$statement = $mysqli->prepare($sql);
$statement->bind_param('s', $parameter);
$statement->execute();
$result = mysql_query($sql);
```

Aj keď je kód používajúci viazanie premenných dlhší v porovnaní s ostanými metódami, neumožňuje programátorovi vložiť do SQL príkazu neupravený vstup a tým tento príkaz upraviť.

Kapitola 3

Návrh

Pri návrhu systému som sa inšpiroval architektúrou CMS systému TYPO3. TYPO3 je pokročilý systém optimalizovaný na tvorbu a správu pokročilých webových stránok a webových aplikácií. Ponúka obrovské množstvo funkcií, taktiež možnosť rozšírenia. Od verzie 4.3 obsahuje rozšírenie Extbase, ktoré pridáva podporu pre rozšírenia a pluginy s architektúrou MVC. Keďže ale ide o starší projekt, jeho jadro je zastaralé, jeho kód je neprehľadný a nepoužíva žiadne návrhové vzory ani riešenia zjednodušujúce úpravy samotného jadra systému.

Výhody CMS TYPO3

- Možnosť použitia rôznych šablón pre rôzne stránky
- Pokročilý a konfigurovateľný backend pre administráciu systému s možnosťou vytvárať používateľov rôznych úrovní, pre ktoré je možné nastaviť rôzne práva na administráciu
- Nastavenie prístupu na stránky rôznym úrovniam používateľov
- Podpora MVC architektúry s použitím rozšírenia extbase
- Podpora rozšírení

Nevýhody CMS TYPO3

- Kód jadra ze zastaralý, neprehľadný, týmpádom obsahuej chyby a ťažko sa upravuje

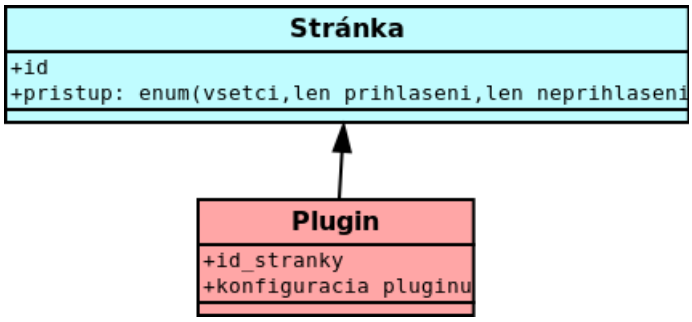
- Vysoká náročnosť na výpočtový výkon, v prípade väčšej návštevnosti webových stránok je nutné použiť cache

3.1 Architektúra systému

3.2 Dátový model

Údaje v databáze môžeme rozdeliť do troch hlavných skupín:

- údaje potrebné k behu aplikácie - konfigurácia, zoznam pluginov na zobrazenie



Obr. 3.1: Vzťah stránok a pluginov, ktoré sa na nich budú zobrazovať

- údaje o aktuálnom stave aplikácie - zoznam používateľov, hlavných aj virtuálnych účtov, príkazy (jednorazové aj trvalé)
- údaje o histórii - akcie vykonané používateľom, uskutočnené príkazy, pravidelne ukladané záznamy o stavoch účtov

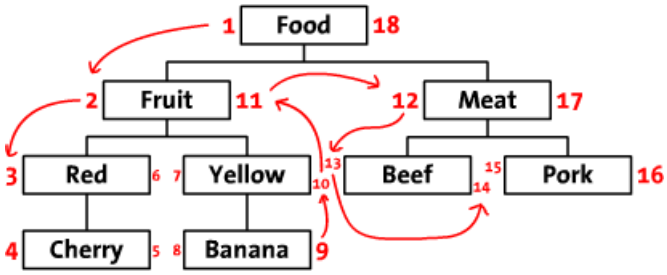
Obr. 3.2: Vzťah používateľov, účtov a histórie účtov

3.2.1 Reprezentácia hierarchických dát v relačnej databáze a algoritmus MPPT

Relačné databázy pracujú s údajmi ako s riadkami v tabuľke. Je to vhodná reprezentácia, ak sú údaje len zoznamom atribútov nejakých objektov. Niektoré údaje v projekte však budú

vytvárať stromovú štruktúru. Reprezentácia takýchto dát je možná použitím kľúča na predchodcu daného záznamu v strome. Ako reprezentácia to postačuje, avšak pohyb po tomto strome, vyhľadávanie predkov, nasledovníkov nie sú efektívne operácie, keďže požadujú rekurzívne prehľadávanie stromu. Takýto prístup ale má za následok množstvo zbytočných SQL dopytov, ktoré môžu v prípade vyššej návštevnosti stránok preťažiť databázový systém. Na tento problém ponúka riešenie algoritmus Modified Preorder Tree Traversal [6] (ďalej už len MPTT), ktorý doplnením záznamov v tabuľkách o ďalšie informácie umožní rýchlejšie a efektívnejšie prehľadávanie hierarchických štruktúr v relačných databázach.

Podmienky pre použitie MPTT Algoritmus MPTT pracuje so stromovými štruktúrami reprezentovanými v tabuľkách. Predpokladá, že riadky v tabuľkách obsahujú celočíselné hodnoty left a right. Tieto hodnoty sa nastavujú pri každom pridaní prvku do stromu, taktiež pri odobraní alebo presunutí prvkov. Hodnoty sa nastavujú pri preorderovskom prehľadávaní stromu, hodnoty left sa nastavujú pri vnáraní do stromu, kým hodnoty right sa nastavujú vždy pri vynáraní zo stromu. Po každom nastavení bude ďalšia hodnota o jednu väčšia.



Obr. 3.3: Príklad hodnôt generovaných algoritmom MPTT. Zdroj: http://www.sitepoint.com

Vyhľadávanie predkov a potomkov v strome pomocou MPTT Ak sú v tabuľke správne nastavené hodnoty left a right, je možné jediným SQL dopytom získať z databázy všetkých predkov, ale aj potomkov daného prvku.

Ak chceme získať z databázy záznamy predkov daného prvku, použijeme SQL dopyt:

```
SELECT * FROM tabulka WHERE left < left_prvku AND right > right_prvku}
```

Ak chceme získať z databázy záznamy potomkov daného prvku, použijeme SQL dopyt:

```
SELECT * FROM tabulka WHERE left > left_prvku AND right < right_prvku}
```

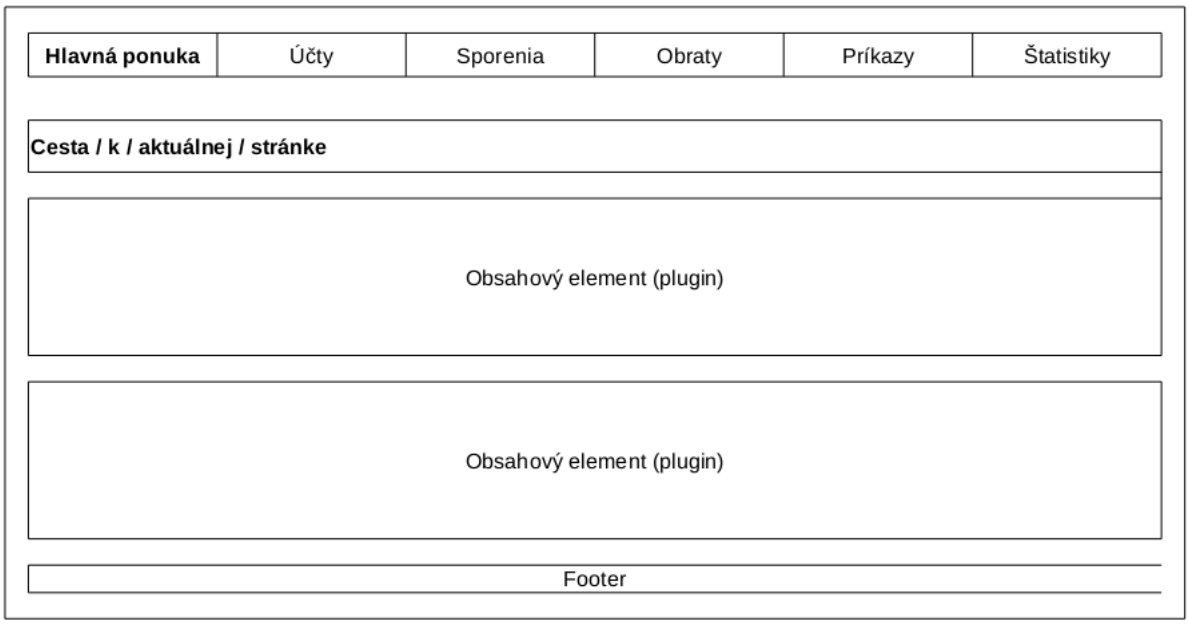
Ak chceme zístiť, či prvok patrí do daného podstromu, skontrolujeme, či platí
`left_korena_stromu > left_prvku AND right_korena_stromu < right_prvku}`

Hodnota level v tabuľke V prípade, že máme záujem o získanie predkov alebo potomkov záznamu v konkrétnej hĺbke stromu, je vhodným riešením do tabuľky pridať stĺpec level, ktorý označuje hĺbku záznamu v strome. Táto hodnota sa nastavuje spolu s hodnotami left a right pri preorderovskom prehľadávaní stromu.

Výhoda a nevýhoda MPTT Výhodou použitia MPTT v relačných databázach je rýchle prehľadávanie stromu, obzvlášť pri veľkých hĺbkach, pri ktorých by klasické prehľadávanie vyžadovalo veľké množstvo SQL dopytov. Nevýhodou tohto prístupu je ale, že je nutné prepočítavanie hodnôt left, right (prípadne level) pri každom pridaním, mazaní alebo presunutí prvku v strome. Táto cena je zanedbateľná, ak počet prehľadávaní v databáze mnohonásobne prevyšuje počet zmien stromu.

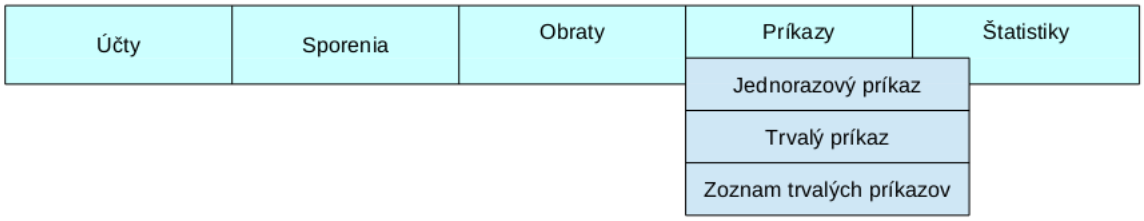
3.3 Návrh používateľského rozhrania

Pri návrhu používateľského prostredia som kládol dôraz na prehľadné a jednoduché ovládanie. Na vrchu stránky je hlavná ponuka obsahujúca najdôležitejšie podstránky. Pod hlavnou ponukou sa nachádza navigácia breadcrumbs - zoznam stránok v stromovej štruktúre od hlavnej stránky po aktuálne prezeranú stránku, ktorá uľahčuje orientáciu na stránke. Pod touto navigáciou sa už nachádza obsah stránky.



Obr. 3.4: Štruktúra stránky aplikácie

Štruktúra hlavnej ponuky Hlavná ponuka obsahuje odkazy na najdôležitejšie podstránky aplikácie.



Obr. 3.5: Štruktúra hlavnej ponuky stránky

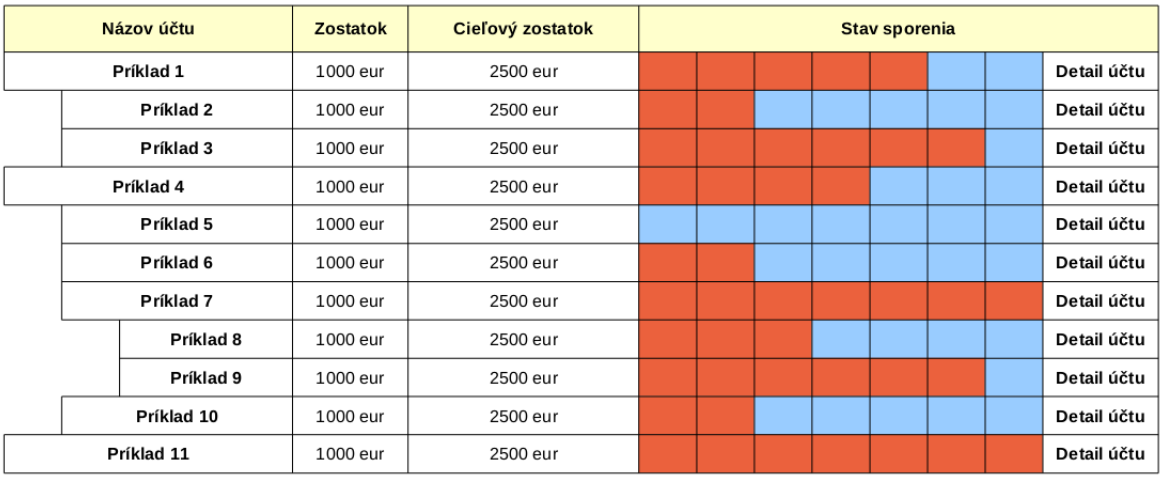
3.4 Zoznam účtov

Stránka zobrazujúca skutočné bankové účty používateľa. Je na nej možné vytvoriť, zmeniť aj zmazať tieto účty. Taktiež umožňuje jednoduchú úpravu zostatku na účte formou jednorazového vkladu alebo výberu. Zobrazuje aj disponibilný zostatok, ktorý je rozdielom sumy peňazí na účte a rezervácií na virtuálne účty.

Obr. 3.6: Zoznam účtov používateľa

3.5 Zoznam virtuálnych účtov

Stránka zobrazuje virtuálne účty používateľa. Povoľuje pridávať, meniť, aj rušiť virtuálne účty. Dôležitou vlastnosťou tohto zobrazenia je, že zvýrazňuje stromovú štruktúru virtuálnych účtov, vďaka čomu je zoznam prehľadnejší.



Obr. 3.7: Zoznam virtuálnych účtov používateľa

3.6 Príkazy

Zoznam trvalých príkazov používateľa, obsahuje informácie o tom, kedy (v akých časových intervaloch), odkiaľ, kam a koľko peňazí sa má preúčtovať. Umožňuje pridať, upraviť, pozastaviť aj zrušiť trvalý príkaz.

Popis	Zdrojový účet	Cieľový účet	Suma	Interval	Posledná realizácia	Najbližšia realizácia	Operácie		
Splátka	Hlavný účet	Hypotéka na dom	400	15. deň v mesiaci	15.5.2012	15.6.2012	Upraviť	Pozastaviť	Zrušiť
Splátka	Hlavný účet	Hypotéka na dom	400	15. deň v mesiaci	15.5.2012	15.6.2012	Upraviť	Pozastaviť	Zrušiť
Splátka	Hlavný účet	Hypotéka na dom	400	15. deň v mesiaci	15.5.2012	15.6.2012	Upraviť	Pozastaviť	Zrušiť
Splátka	Hlavný účet	Hypotéka na dom	400	15. deň v mesiaci	15.5.2012	15.6.2012	Upraviť	Pozastaviť	Zrušiť
Splátka	Hlavný účet	Hypotéka na dom	400	15. deň v mesiaci	15.5.2012	15.6.2012	Upraviť	Pozastaviť	Zrušiť

Obr. 3.8: Zoznam príkazov používateľa

3.7 Triedny diagram

V tejto časti predstavím hlavné triedy MVC frameworku a vzťahy medzi nimi.

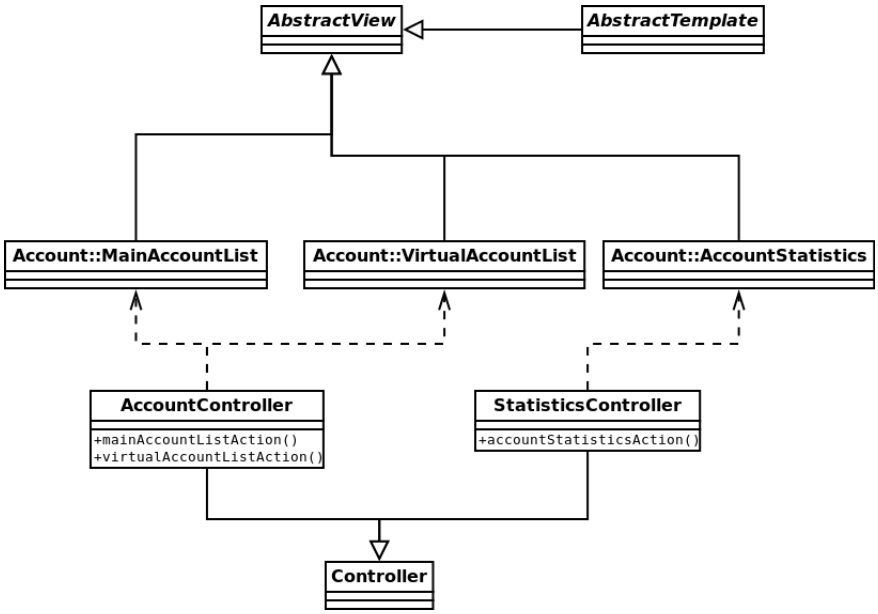
3.7.1 View a controller

View a controller sú časti MVC aplikácie, ktoré spolu najviac súvisia. Controller zabezpečuje aplikačnú logiku, kým view zobrazí výsledok činnosti controlleru. Znamená to, že sa bez seba navzájom nezaobídu.

Podobne ako v iných MVC frameworkoch, aj v mojom riešení existuje abstraktná trieda Controller, ktorá zabezpečuje funkcionalitu spoločnú pre všetky od nej odvodené controller triedy.

Každý controller implementuje jednu alebo viac akcií. Akcie sú reprezentované metódami controllerov. Ku každej akcii patrí jedna view trieda odvodená od abstraktného predka AbstractView. Každá trieda odvodená od AbstractView sa stará o zobrazenie výstupu jednej akcie.

Špeciálnym potomkom view triedy je trieda AbstractTemplate, ktorej potomkovia predstavujú šablóny celej webovej aplikácie.

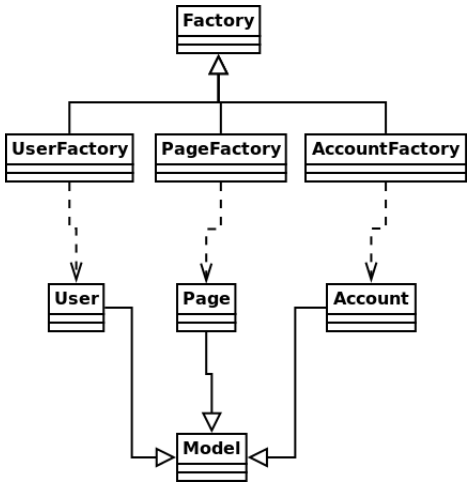


Obr. 3.9: Vzťahy view a controller tried

3.7.2 Model

Napriek tomu, že model je veľmi dôležitou súčasťou aplikácií, aj tých, ktoré nie sú postavené na MVC architektúre, nie je so žiadnou súčasťou aplikácie v takom úzkom vzťahu, ako sú model a view.

Model sa skladá z dvoch kategórií tried. Prvú kategóriu predstavuje model samotný, teda triedy, ktorých inštancie reprezentujú záznamy v databáze. Druhou kategóriou sú factory triedy, ktoré sa starajú o načítanie údajov z databázy a z týchto údajov vytvárajú inštancie modelových tried.



Obr. 3.10: Vzťah medzi modelových a factory triedami

Kapitola 4

Implementácia

Počas implementácie projektu som sa snažil čo najviac využiť svoje vedomosti nadobudnuté v oblasti programovania a tvorby webových dokumentov. Mojími hlavnými cieľmifa

- práca s programovacím jazykom PHP
- návrh a implementácia projektu pomocou modernej a pokročilej architektúry a s využitím OOP návrhových vzorov
- práca s MySQL databázou prostredníctvom rozšírenia mysqli kvôli objektovo orientovanému prístupu, pre rýchlejšie osluhovanie SQL dopytov a elimináciu zraniteľnosti voči SQL injection.

4.1 MVC framework

V mojom projekte som sa rozhodol vytvoriť si vlastný, v porovnaní s existujúcimi na použitie aj implementáciu jednoduchší framework. Pri implementácii som využil dynamické vlastnosti jazyka php (získanie názvu triedy, dynamické názvy volaných metód alebo inštanciovaných tried) aj podporu menných priestorov.

4.1.1 Model

Pre modely objektov som sa do projektu rozhodol zaviesť mechanizmus, vďaka ktorému je možné definovať model pre každú tabuľku rýchlo a jednoducho. Pre základnú funkcionalitu

(jednoduché selecty, update, mazanie) stačí vytvoriť modelovú a factory triedu pre daný model.

Modelová trieda tabuľky V triede odvodenej od triedy \Core\Model treba určiť názov tabuľky a stĺpcov. Factory trieda odvodená od \Core\

```
class Content extends \Db\Item {
  protected static $tableName = 'contents';
  protected static $columns = array(
    'page' => 'fk_page_id',
    'position' => 'position',
    'extensionName' => 'extension',
    'pluginName' => 'plugin',
    'configuration' => 'configuration',
  );
}
```

Factory trieda tabuľky Factory trieda pre danú tabuľku je ešte jednoduchšia na implementáciu, jediným členom triedy musí byť premenná instance, v ktorej je uložená inštancia danej factory.

```
class ContentFactory extends  \Db\Factory {
  protected static $instance;
}
```

Vytvorením týchto dvoch tried umožníme jednoduché načítavanie, zmenu aj mazanie údajov v danej tabuľke.

Implementácia abstraktnej modelovej triedy Účelom abstraktnej triedy modelu je uchovávať dáta z jedného riadku tabuľky a umožniť tieto dáta meniť a mazať. Jedným z mojich cieľov bolo umožniť jednoduchú a rýchlu implementáciu modelov pre tabuľky v databáze, popritom ale dbať o správne programátorské postupy a bezpečnosť. Takýto prístup okrem iného vyžaduje, aby vlastnosti a premenné modelu neboli verejné, teda aby k nim programátor nemohol pristupovať priamo, ale cez setter a getter funkcie. Keďže je ale časovo náročné vytvárať tieto funkcie pre každú hodnotu modelu zvlášť, rozhodol som sa využiť v php magickú funkciu ___call, ktorá mi umožní volať aj v triede neexistujúce metódy.

Ak programátor zavolá v modeli funkciu s menom zodpovedajúcim vzoru getPremenná alebo setPremenná a funkcia s takýmto názvom neexistuje, php automaticky zavolá mag-

ickú metódu ___call(). Táto metóda sa podľa názvu volanej funkcie rozhodne, či vráti hodnotu premennej modelu alebo hodnotu premennej nastaví.

```
public function __call($functionName, $values) {
  if (substr($functionName, 0, 3) == 'get') {
    $key = substr($functionName, 3, strlen($functionName)-3);
  if (isset(static::$columns[strtolower($key)])) {
    return $this->dataArray[static::$columns[strtolower($key)]];
  } else {
    throw new \Exception('Unknown property ' . $key, $code);
  } elseif (substr($functionName, 0, 3) == 'set') {
  $key = lcfirst(substr($functionName, 3, strlen($functionName)-3));

    if (isset(static::$columns[$key])) {
    $this->dataArray[static::$columns[$key]] = $values[0];
    } else {
      throw new \Exception('Unknown property ' . $key, 12);
    }
  }
}
```

Výsledkom tohto prístupu je, že model mi umožní volať metódy getVlastnost() a setVlastnost(novaHodnota) bez nutnosti ich v skutočnosti implementovať.

V prípade ukladania údajov do databázy musí model z definovaných názvov stĺpcov tabuľky a z hodnôt priradených modelu vytvoriť SQL príkaz, ktorý následne vykoná. O túto činnosť sa stará metóda save() modelu, ktorá podľa potreby do databázy pridá nový záznam, alebo upraví už existujúci. Rozhoduje sa podľa toho, či už má daná inštancia modelu databázou automaticky priradené id číslo. Ak nie, je nutné vytvoriť nový záznam.

Ako príklad fungovanie ukladania dát uvediem vytváranie nového záznamu v tabuľke.

V prvom kroku je potrebné vytvoriť SQL príkaz, ktorý neskôr databáza spracuje. Do tohto príkazu sa nevkladajú žiadne vstupy od používateľa, na miesto parametrov sa vkaldajú otázniky.

```
$sql = 'INSERT INTO ' . self::getTableName() . '
    SET ' . implode(', ', array_map(function($column) {
        return '`' . $column . '`=?';
    }, self::getColumns()));
```

SQL príkaz posunieme na spracovanie.

```
$statement = \Db\Connection::getInstance()->prepare($sql);
if (!$statement) {
  throw new \Exception(\Db\Connection::getInstance()->error);
}
```

Do vytvoreného SQL príkazu musíme naviazať hodnoty, ktoré chceme zapísať do databázy. Problémom ale je, že viazať musíme referencie na premenné s týmito hodnotami, nie hodnoty samotné. Keďže nepoznáme počet týchto hodnôt, musíme volať funkciu na viazanie premenných dynamicky, ktorá ale priamo nepodporuje predávanie referencií na premenné. Preto model najprv vytvorí pole s referenciami na hodnoty inštancie, ktoré už môže naviazať do SQL príkazu.

```
$values = $this->dataArray;

foreach ($values as $key => $value) {
  $valueReferences[$key] = &$values[$key];
}

$valueTypes = implode('', array_map(function() {return 's';}, self::getColumns()));

$params = array_merge(array($valueTypes), $valueReferences);
```

Po vytvorení referencií sa môže pripraviť a vykonať SQL dopyt. V prípade, že dopyt sa nevykoná, funkcia vyhodí výnimku.

```
call_user_func_array(array($statement, "bind_param"), $params);

if (!$statement->execute()) {
  throw new \Exception(\Db\Connection::getInstance()->error);
}
```

4.1.2 View

View sú triedy odvodené od predka AbstractView. Musia implementovať funkciu render(), ktorá zobrazí výstup na základe údajov, ktoré boli triede poskytnuté prostredníctvom funkcie set().

```
abstract class AbstractView {

  public function set($name, $value) {
    $this->$name = $value;
  }

  abstract public function render();

}
```

4.1.3 Controller

Hlavnou funkciu triedy controller je execute(), ktorá na základe parametra $configuration vyberie správnu akciu a vykoná ju:

- od kofigurácie získa názov akcie, ktorá sa má vykonať
- z vlastného mena triedy a názvu akcie vytvorí názov view, ktorý sa má použiť
- vytvorí view a uloží ho
- zavolá akciu, ktorá sa má spustiť

```
public function execute(ControllerConfiguration $configuration) {
  $actionName =  $configuration->getAction();

  $className = get_class($this);
  $classNameParts = explode('\\', $className);

  $extensionName = $classNameParts[0];
  $controllerName = substr($classNameParts[2], 0, strlen($classNameParts[2]) - 10 );

  $viewClassName = $extensionName . '\\View\\' . $controllerName . '\\' . ucfirst($actionName);

  $this->view = new $viewClassName();

  $this->view->set('settings', $configuration->getPluginConfiguration());


  $this->urlMaker = new \Core\UrlMaker();
  $this->urlMaker->setController($this);

  $this->view->set('urlMaker', $this->urlMaker);

  $actionMethodName = $actionName . 'Action';
  $this->$actionMethodName();

}
```

4.1.4 Rozšírenia a pluginy

Aplikáciu som sa snažil navrhnúť tak, aby bola jednoducho a rýchlo rozšíriteľná, ale aby pritom ostala prehľadná. Z tohto dôvodu som sa rozhodol zaviesť systém rozšírení. Rozšírenia v aplikácii sú balíky tried, controllerov, view, funkcií, ktoré so sebou nejakým spôsobom súvisia. Rozšírenia, ktoré som v rámci projektu vytvoril, sú:

Rozšírenia aplikácie

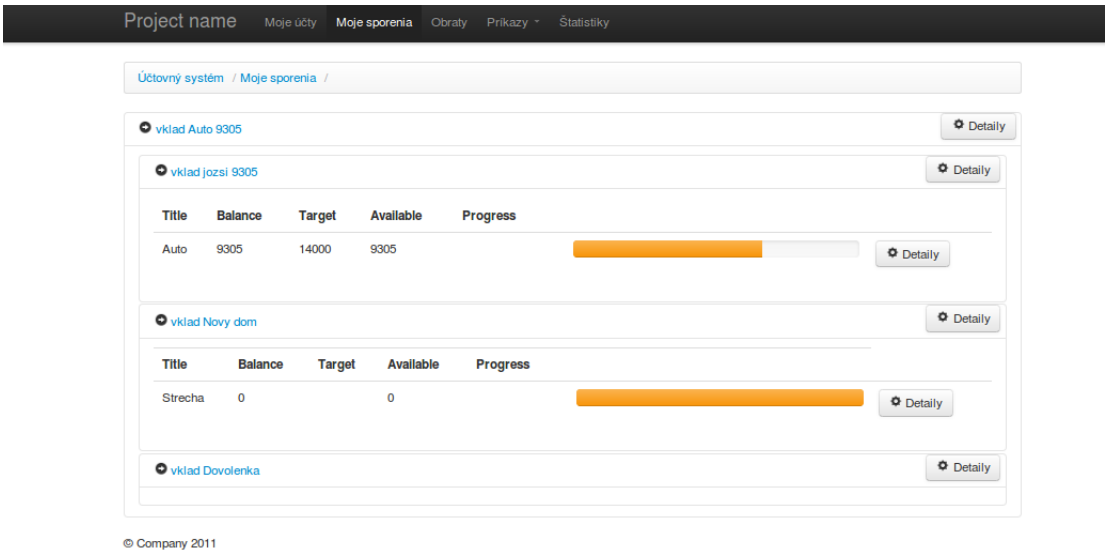
- Db - balík, ktorý obsahuje základné triedy umožňujúce pripojenie na databázu a triedy potrebné na vytvorenie modelu.
- Core - rozšírenie nevyhnutné pre beh aplikácie, obsahuje triedy potrebné pre spracovanie a vytváranie url, základné triedy MVC frameworku (okrem modelu). Poskytuje mechanizmy pre autentifikáciu používateľa.
- Account - rozšírenie, ktoré obsahuje model pre účty používateľa, pre príkazy, prevody, štatistiky. Taktiež poskytuje všetky pluginy, ktoré zobrazujú finančné údaje používateľa.

4.1.5 Pluginy

Všetok obsah na stránke (okrem obsahu daného hlavnou šablónou) je tvorený pluginmi. Každý plugin je obsahový prvok tvorený akciou controlleru a šablónou definovanou vo view. Tabuľka contents v databáze určuje, aké pluginy sa majú na danej stránke zobraziť.

4.2 Používateľské prostredie a Twitter Bootstrap

V projekte som v používateľskom prostredí použil Bootstrap od Twitteru, voľne prístupný balík HTML, CSS a Javascript kódov, ktoré uľahčujú tvorbu webstránok. Tento balík nám ponúka možnosť formátovať obsah webstránky pomocou preddefinovaných štýlov (nadpisy, zoznamy, oznamy), ale aj množstvo ovládacích prvkov (tlačidlá, prvky formulárov, dropdown ponuky). Celé používateľské prostredie som vytvoril pomocou tejto knižnice.



Obr. 4.1: Zoznam virtuálnych účtov vytvorených pomocou knižnice Twitter Boostrap

Kapitola 5

Záver

Cieľom mojej práce bolo vytvoriť webovú aplikáciu umožňujúcu jednoduché a prehľadné osobné účtovníctvo pre používateľov.

5.1 Ďalšie možnosti vývoja

Aplikáciu, hlavne MVC framework, na ktorom aplikácia beží, sa mi podarilo navrhnúť a implementovať tak, že je možné všetky súčasti aplikácie aj samostatne rýchlo a jednoducho rozširovať. Vďaka tomu sa naskytuje viacero možností, ako s projektom pokračovať, pričom tieto možnosti sa navzájom nevylučujú.

5.1.1 Rozšírenie osobného účtovnítva o ďalšie funkcie

Možnou alternatívou ďalšieho vývoja je rozšírenie osobného účtovníctva nielen o funkcie, ktoré sa nepodarilo implementovať v rámci práce, ale aj o mnoho ďalších možností, ktoré by spravili aplikáciu užitočnejšou:

- Pokročilé štatistiky, interaktívne grafy s možnosťou nastavení intervalov, za aký čas sa majú zobrazovať údaje
- Emailové, SMS notifikácie používateľa o pohyboch, o dosiahnutí cieľovej sumy alebo o nedostatku peňazí na účtoch
- Zavedenie úrokov na účtoch
- Poradca pri výbere pôžičky

5.1.2 Rozšírenie funkcionality MVC frameworku

Aj keď mnou implementovaný MVC framework poskytuje základnú funkcionalitu potrebnú pre beh webových aplikácií, jeho zďaleka nepokrýva všetku funkcionalita, ktorú by od neho programátor mohol očakávať. V prípade ďalšieho vývoja by bolo vhodné sa sútrediť na nasledujpce oblasti:

- vytváranie pokročilých URL, automatické spracovávanie parametrov, prípadne kofigurovateľné vytváranie URL
- vytvorenie konfigurovateľnej cache, s granularitou nastaviteľnou ako pre celé stránky, tak pre jednotlivé pluginy
- vytvorenie cache pre model
- rozšírenie modelu o podporu agregačných funkcií, joinov tabuliek

5.1.3 Doplnenie MVC frameworku o administráciu, vytvorenie CMS systému

Keď som navrhoval a implementoval MVC framework pre projekt, snažil som sa o to, aby bolo možné v budúcnosti vytvoriť k nemu administračné rozhranie (backend), ktoré by umožňovalo úpravu údajov v databáze. Takéto rozhranie by si vyžadovalo implementáciu pokročilej správy oprávnení pre používateľov aplikácie. Myslím si však, že moje riešenie by mohlo byť vhodným základom pre takýto projekt.

Literatúra

[1] Rudolf Pecinovský *Návrhové vzory.* Prvé vydanie, Brno : Computer Press, a.s., 2007. 527 s. ISBN 978-80-251-1582-4.

[2] George Schlossnagle *Advanced PHP Programing.* 1st Edition: Pearson Education, Inc, 2004. 685 s. ISBN 0672325616.

[3] Storing Hierarchical Data in a Database Article. [online], [cit. 25.5.2012]. `http://www.sitepoint.com/hierarchical-data-database/`.

[4] Peter Moulding *PHP Black Book.* The Coriolis Group, 2002. 880 s. ISBN 1588800539.

[5] Online dokumentácia jazyka php. [online], [cit. 25.5.2012]. `http://php.net/docs.php`.

[6] Ryan Asleson, Nathaniel T. Schutta *Ajax - Vytváříme vysoce interaktivní aplikace.* Prvé vydanie, Brno : Computer Press, a.s., 2006. ISBN 80-251-1285-3.

[7] Sverre H. Huseby *Zranitelný kód.* Prvé vydanie, Brno : Computer Press, a.s., 2006. ISBN 80-251-1180-6.