TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

BAKALÁŘSKÁ PRÁCE

Liberec 2012 **Jan Žmolík**

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

WWW aplikace – komunitní portál

Web applications – community portal

Bakalářská práce

Autor: Jan Žmolík
Vedoucí práce: Mgr. Jiří Vraný, Ph.D.

V Liberci 18. května 2012

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2011/2012**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jan Žmolík**
Osobní číslo:	**M08000189**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**WWW aplikace - komunitní portál**
Zadávající katedra:	**Ústav nových technologií a aplikované informatiky**

Zásady pro vypracování:

1. Seznamte se s problematikou návrhových vzorů a zejména s jejich specifiky v prostředí dynamicky typovaných jazyků jako např. PHP.
2. V jazyce PHP vytvořte WWW aplikaci - komunitní portál. Při vývoji aplikace využijte OOP a návrhových vzorů.
3. V závěrečné zprávě uveďte konkrétní příklady použití návrhových vzorů ve vámi vytvořené aplikaci.

Rozsah grafických prací: **dle potřeby**

Rozsah pracovní zprávy: **cca 50 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. Design Patterns : Elements of Reusable Object-Oriented Software. [s.l.] : Addison-Wesley Professional, 1994. 416 s. ISBN 0201633612, ISBN-13: 978-0201633610.

[2] HERRINGTON, Jack D. IBM - United States [online]. 18 Jul 2006 [cit. 2010-10-14]. Five common PHP design patterns. Dostupné z WWW: http://www.ibm.com/developerworks/library/os-php-designptrns/.

Vedoucí bakalářské práce: **Mgr. Jiří Vraný, Ph.D.**
Ústav nových technologií a aplikované informatiky

Datum zadání bakalářské práce: **14. října 2011**

Termín odevzdání bakalářské práce: **18. května 2012**

prof. Ing. Václav Kopecký, CSc.
děkan

L.S.

prof. Dr. Ing. Jiří Maryška, CSc.
vedoucí ústavu

V Liberci dne 14. října 2011

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiju-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce.

Datum 18. května 2012

Podpis

Poděkování

Děkuji Mgr. Jiřímu Vranému, Ph.D. za vedení bakalářské práce a za podnětné konzultace. Dále děkuji všem, kteří mě při tvoření této práce podporovali.

Abstrakt

Tato práce se zabývá návrhovými vzory a jejich využitím v oblasti webových aplikací. Práce je rozdělena do dvou hlavních částí. První část obsahuje nástin problematiky a představení vytvořené aplikace. V druhé části je demonstrováno využití návrhových vzorů na částech webové aplikace. Závěr práce obsahuje shrnutí poznatků z praktické části a uvádí výhody a nevýhody užití návrhový vzorů při tvorbě webových aplikací.

Aplikace, na které je předvedeno použití návrhových vzorů, je obsažena na přiloženém CD.

Abstract

This work deals with design patterns and thein use in Web applications. The work is divided into two main parts. The first part contains an outline of issues presenting created application. In the second part is demonstrated the use of design patterns on parts of the web application. The conclusion summarizes the findings from the practical part and gives the advantages and disadvantages of using design patterns for creating web applications.

Application, which shows using of design patterns, is included on the CD.

Klíčová slova

Návrhové vzory, webové aplikace, PHP, objektově orientované programování

Keywords

Design patterns, web applications, PHP, object-oriented programming

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Pro dnešního internetového uživatele je trendem být na nějaké sociální síti nebo alespoň komunitním portálu. Největšími hráči na poli sociálních sítí jsou Facebook, Twitter a nově také Google+. Tyto velké sociální sítě doplňuje mnoho menších komunitních portálů s užším zaměřením.

Jedním z příkladů těchto menších sítí, zaměřených na jedno téma, je komunitní portál Československá filmová databáze (dále jen csfd), který se zabývá filmy a seriály. Registrovaní uživatelé na něm mohou hodnotit, komentovat a diskutovat o filmových snímcích. Jejich hodnocení se promítá do celkového hodnocení filmu. Tento portál je dnes velmi známý, nejen mezi filmovými nadšenci. Pro mnoho lidí platí dokonce za určující měřítko kvality filmů a seriálů. I když jdete do kina, tak před samotným začátkem filmu se objeví reklama na tento portál s připomínkou, aby diváci nezapomněli ohodnotit tento film na csfd.

Součástí každodenního života mnoha lidí je poslech hudby. Spojení těchto dvou skutečností – hudby a webových stránek s názory samotných uživatelů – dalo vzniknout myšlence vytvořit právě takový webový portál se zaměřením na světovou i domácí hudební scénu. Existuje sice několik světových, obvykle anglických, webů, které jsou právě podobně orientovány, ale mnoho hudebních fanoušků nechce nebo kvůli nedostatečné znalosti anglického jazyka nemůže tyto weby plně využívat. I na českém internetu existuje několik podobných projektů, ale žádný z nich není na podobné úrovni jako csfd a jedná se spíše o neznámé weby mezi českými uživateli internetu.

Dnes se v programování používá mnoho moderních technologií a postupů, mezi nimi i návrhové vzory. Zadání práce specifikuje vytvořit komunitní portál a využít při tom tyto moderních technologie – zvolit framework, použít OOP a vhodné návrhové vzory. Zároveň tak demonstrovat použitelnost, výhody a nevýhody vybraných návrhových vzorů při tvorbě webové aplikace. Jako téma komunitního portálu bylo zvoleno právě zaměření na hudbu.

2 Cíle práce

Cílem práce je seznámení s problematikou návrhových vzorů a jejich specifiky v prostředí slabě typovaných jazyků jako je PHP. Řešitel zvolí vhodné návrhové vzory pro webové aplikace. Ty následně využije při tvorbě vlastní webové aplikace – komunitního portálu.

Implementací těchto vzorů by měl demonstrovat výhody a nevýhody, které použití návrhových vzorů v jazyce PHP přináší.

3 Představení nástrojů

3.1 Návrhový vzor

Odbornou řečí je návrhový vzor obecným předpisem řešení netriviálního a opakujícího se problému. Dle [1] jde o způsob postupu, kterým se budeme při tvorbě programu řídit. Jako příklad uvádí stavbu domu. Můžeme si vybrat mnoho způsobů jak stavět dům. S největší pravděpodobností si vybereme postup, který je léty ověřený a funkční. Podle [1] jsou tedy návrhové vzory právě těmito ověřenými postupy v oblasti programování. Vzory by se daly také přirovnat ke kuchařským receptům.

Vzory mají usnadnit práci už ve fázi navrhování aplikace. Následná implementace podle vybraného vzoru zlepší přehlednost a čitelnost výsledného kódu. [1] uvádí že, využitím vzorů bude kód srozumitelnější pro jiného člověka – bude se v něm lépe orientovat. Dále také uvádí, že je menší pravděpodobnost, vytvoření chyby. Důvodem je skutečnost, že vzory jsou ověřenými postupy. V mnoha případech vzory zajistí snadnou rozšiřitelnost aplikace.

Návrhové vzory jsou úzce spjaty s technikou objektově orientovaného programování, protože popisují řešení aplikace právě pomocí tříd a objektů. Návrhové vzory jsou jazykově nezávislé. Je tedy možné je použít ve všech programovacích jazycích, které jsou objektově orientované.

OOP je metodika vývoje softwaru, která se snaží popsat reálné prvky pomocí objektů a jejich metod, viz kapitola 3.3. V OOP se využívá principů, jako je zapouzdření, dědičnost nebo polymorfismus. Návrhové vzory tyto vlastností využívají.

Vzory jsou obvykle popisovány jazykem UML. UML je jazyk pro popis architektury, která využívá OOP.

3.2 PHP

PHP je skriptovací programovací jazyk. V dnešní době patří k nejpopulárnějším jazykům v oblasti webových aplikací. Má značnou podporu ze strany hostingů a pokročilejší programátoři mohou sáhnout po některém z mnoha PHP frameworků, které si kladou za cíl usnadnit vývoj.

Aktuálně je již ve verzi 5.4 a právě od major verze 5.0 obsahuje PHP vylepšený objektový model. Dá se tedy snáze využít vlastností OOP. Pro podrobnější informace je možné navštívit oficiální stránku, viz [2].

Pro vytvoření vlastní aplikace a následné praktické využití vzorů byl zvolen framework Nette ve verzi 2.0. Jedná se o český framework pro tvorbu webových aplikací v PHP 5.3 a vyšší. Mezi hlavní předností patří dle [3] zaměření na bezpečnost aplikace, využívání technologií AJAX, SEO, implementace architektury MVC nebo vnitřní ladící nástroje. Dále také nabízí funkce pro autentizaci a autorizaci uživatelů.

Velkou výhodou je, že existuje mnoho návodů v češtině, které umožňují rychlé nastudování základních funkcí Nette. Navíc se okolo frameworku vytvořila poměrně početná česká komunita vývojářů. Na oficiálním fóru Nette pomáhají začátečníkům vyřešit problémy, na které mohou při vývoji prvních aplikací v prostředí frameworku narazit.

Více informací lze získat na oficiální stránce, viz [3].

3.3 Objektově orientované programování

Hlavním konceptem OOP je převést reálný svět do podoby objektů a tříd, které budou tento svět reprezentovat. Oproti strukturovanému programování, kde tento svět reprezentuje pouze sada proměnných a funkcí, jsou tyto v OOP nahrazeny třídami a objekty. Každá třída nebo objekt je autonomní samostatnou jednotkou, který o sobě nese informace (uložené v atributech) a nabízí funkce pro práci s nimi (metody) a nejenom s nimi. Všechny objekty a třídy mezi sebou mohou komunikovat a předávat si informace pomocí metod k tomu určených.

Stručný přehled základních pojmů v OOP:

- Třída – je předpis, podle kterého se vytvoří objekt. Jsou v ní definovány atributy a metody.
- Objekt – je instancí třídy. Je určen názvem, hodnotami atributů a metodami, které používá.
- Atribut – je vnitřní proměnná třídy a objektu. Uchovávají informace o stavu objektu.
- Metoda – je vnitřní funkce třídy a objektu. Objektu slouží ke komunikaci (podávání informací o sobě) nebo změně stavu (změna hodnot atributů).
- Rodič – rodičovská třída je taková, od které se odvozují další objekty.
- Potomek – je třída odvozená od nějaké jiné třídy.

Přístupová práva pro atributy a metody (viditelnost):

- Public – jsou přístupné z venku
- Private – jsou přístupné pouze uvnitř třídy
- Protected – jsou přístupné pouze uvnitř třídy a navíc k nim má přístup každá odvozená třída

Mezi vlastnosti objektově orientovaného programování podle [4] patří:

Dědičnost

Dědičnost tříd umožňuje vytvářet nové třídy odvozením od již existující třídy. To znamená, že nová třída získává všechny atributy a metody třídy původní. S atributy lze následně pracovat a měnit jejich hodnoty. Původní metody lze přepisovat, takzvaně překrývat, svou vlastní implementací těla metody v potomkovi. Potomkovi lze také přidat další atributy a nové metody.

Polymorfismus

Jde se o vlastnost, která umožňuje různé chování stejných metod.

Jeden způsob polymorfismu je takzvaně parametrový. Jedná se o situaci, kdy má třída dvě nebo více metod se stejnými názvy, ale liší se počtem nebo typem parametru. Jedná se o přetěžování metod. PHP parametrový polymorfismus neumožňuje.

Druhý způsob souvisí s dědičností. Pokud máme společného předka a od něj odvozeno více potomků, každý může implementovat tělo jedné metody různě. Poté, když je volána tato metoda, záleží, nad jakým objektem je vykonávána.

Zapouzdření

Každý objekt reprezentuje něco z reálného světa. Zapouzdřením je docíleno stavu, kdy každý objekt je samostatná uzavřená jednotka, která uvnitř obsahuje všechny informace o sobě. Objekt má nadefinované metody, pomocí kterých s ním lze komunikovat, zjišťovat stav jeho vnitřních proměnných, případně je i měnit. Podle [5] je zapouzdřením dosaženo skrytí implementace. To znamená, že objekty komunikující mezi sebou si navzájem poskytnou informace o tom, co umí, ale ne o tom, jak to dělají – objekty nedávají ostatním objektům přístup k implementacím svých metod. Zapouzdření je dosaženo pomocí nastavení viditelnosti u jednotlivých atributů a metod, viz výše.

Více o pojmech v OOP v prostředí jazyka PHP lze nalézt v [6], [7] a [8].

4 Problematika návrhových vzorů

Návrhové vzory se opírají o techniku OOP. V mnoha návrhových vzorech je použito dědičnosti a polymorfismu.

Obvykle je pro výklad návrhových vzorů použito programovacího jazyka Java, který má silnou typovou kontrolu. Tato práce je ovšem zaměřena na využití a specifika vzorů v prostředí programovacího jazyka PHP, který je typovaný slabě.

4.1 Slabé typování jazyka PHP

U jazyků se slabou typovou kontrolou je možné ukládat do jedné proměnné jakýkoliv typ dat. Od jednoduchých datových typů jako jsou int, double, bool nebo char až po různé objekty a třídy. Podstatě jazyka to neodporuje a vše je z tohoto pohledu v pořádku. Ovšem v případě, kdy je třeba zajistit, aby se v konkrétní proměnné nacházel určitý typ objektu, je třeba toto opravdu zajistit. Nesmí se totiž stát, že by se najednou v proměnné, která uchovává objekt z některého návrhového vzoru a je kritická pro běh programu, objevil například textový řetězec. Je tedy důležité dát pozor na to, jakým způsobem aplikaci v slabě typovaném jazyce píšeme. Nesmíme totiž dát nikomu možnost zasáhnout do těchto kritických bodů aplikace tím, že by si mohl libovolně upravovat obsah těchto proměnných. Toto je nutné dodržet i u silně typovaných jazyků. U těch je ale menší předpoklad podobné chyby, vzhledem k tomu, že překladač jednoduše nedovolí, například do proměnné typu int, uložit objekt a naopak.

Slabě typovaný jazyk nemusí mít nutně deklarovanou proměnnou jako datový typ třídy předka, aby se do ní dali vložit instance tříd potomků. Je možné, že si mnoho PHP programátorů řekne, proč by tedy v takovém případě měli dodržovat na první pohled složité diagramy se všemi dědičnostmi, když stačí navrhnout tři navzájem podobné třídy a ty následně pouze ukládat do jedné proměnné. Tím by ale bylo popřeno všechno, co návrhové vzory znamenají. Přináší do programování jistou konzistenci, protože pokud všechny tři třídy budeme dědit ze společného předka, bude nás překladač (nebo interpret kódu) nutit, aby všechny tři třídy byly správně implementovány po vzoru svého předka. Pokud by tyto třídy neměly společného předka, což v případě slabě typovaných jazyků nutně nemusí, je velká pravděpodobnost, že během vývoje dojde u těchto tříd k nekonzistenci a zvýší se tak výskyt chyb. Každá potom může mít implementovány různé metody a může se poté stát, že aplikace se zhroutí a nebude fungovat.

4.2 Nestavový HTTP

Při vývoji webových aplikací musíme počítat s tím, že protokol HTTP (hypertext transfer protocol), který zprostředkovává komunikaci mezi klientem a serverem je nestavový. To znamená, že sám o sobě nedokáže mezi jednotlivými požadavky přenášet žádné proměnné nebo objekty. O toto se musí postarat sám programátor. PHP frameworky nabízí prostředky, jak proměnné jednoduše přenášet.

Jak je uvedeno dále v kapitole 6, při použití návrhových vzorů v PHP je nutno tuto skutečnost zohlednit a dbát na správnou implementaci vzorů.

4.3 Znovu použitelnost kódu

Návrhové vzory mají také za účel usnadnit znovu použitelnost již jednou napsaného kódu. Tím, že je kód rozdělen a zapouzdřen do jednotlivých tříd, je snadnější použít kusy kódu na více místech v aplikaci, aniž bychom museli kód opakovat.

Tuto výhodu nejvíce přináší použití architektury MVC nebo MVP.

4.4 Přehlednost kódu

Návrhové vzory jako takové poskytují kromě ucelených řešení daných problému také přehlednost kódu. V případě, že se rozhodneme implementovat celý výkonný kód, jako jeden velký celek s mnoha výhybkami v podobě složité konstrukce ifů, začne být výsledný kód po chvíli značně nepřehledný. Jakékoliv pozdější úpravy budou obtížné a radikálnější zásah do aplikace bude téměř nemožný.

Pokud zvolíme některý z návrhových vzorů, který už svou podstatou rozděluje jednotlivé části kódu na menší segmenty v podobě tříd nebo objektů, bude výsledný kód aplikace přehlednější, čitelnější a budoucí úpravy budou daleko jednodušší.

Navíc v takovéto podobě bude kód aplikace přehlednější a čitelnější pro další programátory, kteří mají v práci na aplikaci pokračovat nebo spolupracovat na vývoji.

4.5 Menší chybovost

Výhoda návrhových vzorů je v tom, jak bylo uvedeno v kapitole 3.1, že řešení které přináší, jsou osvědčená a vyzkoušená. Není sice zaručena stoprocentní bezchybnost kódu, ale riziko výskytů chyb je daleko menší, než v případě, že se budeme snažit najít vlastní zcela nové řešení. Navíc je velmi pravděpodobné, že nakonec se stejně spousta těchto „inovátorů“ dostane k řešení, které je přineseno některým z návrhových vzorů.

5 Komunitní portál

5.1 Rešerše stejně zaměřených webů

Tato část obsahuje shrnutí poznatků o webových portálech, které mají hudební zaměření a nabízejí možnosti hodnocení samotnými uživateli.

Obecně všechny weby obsahují profily interpretů, které nabízí informace v podobě, biografie, seznamu aktuálních a bývalých členů, diskografie, zajímavostí. Dále obsahují profily jednotlivých nahrávek, které taktéž nabízí základní informace, jako seznam skladeb, rok vydání a zajímavosti.

Obvykle weby umožňují hodnotit nahrávky. V některých případech také samotné interprety nebo skladby.

Každý web je označen jménem, internetovou adresou a zkratkou jazykové lokalizace. Následuje výčet kladů a záporů.

Encyclopaedia Metallum – www.metal-archives.com, EN

Klady a zápory:

+ rozsáhlá databáze hudebních kapel a projektů
+ relevantní a aktuální informace
+ přehledné grafické zpracování
+ seznamy interpretů, možnost různého třídění
+ vyhledávání podle zvolených kritérií
+ možnost přidávat nahrávkám jak bodové hodnocení, tak komentáře
- striktní žánrové vymezení (metal a jeho subžánry)

Music Might – www.musicmight.com, EN

Klady a zápory:

+ rozsáhlé biografie u mnoha interpretů
+ široký záběr hudebních žánrů
- absence seznamu interpretů
- nepřehledné grafické zpracování a rozložení ovládacích prvků
- u mnoha interpretů chybí nahrávky
- možnost hodnotit nahrávky pouze bodově
- v současné době již patrně neaktualizovaná

MetalStorm – www.metalstorm.ee, EN

Klady a zápory:

+ webzin, kombinovaný s hudební databází
+ velmi podrobné informace o interpretech
+ možnost hodnocení a komentování nahrávek
+ možnost filtrování nahrávek podle typu
- užší žánrové zaměření (rock a metal)

MusicBrainz – musicbrainz.org, EN

Klady a zápory:

+ poskytuje možnost propojení se softwarem na editaci ID3 tagů hudebních souborů
+ velmi rozsáhlé diskografie, která obsahuje i neoficiální nahrávky
+ možnost hodnotit i interprety a skladby
+ mnoho meta dat k jednotlivým nahrávkám
- nemožnost psát komentáře
- žádné biografie u interpretů

Rate Your Music – rateyourmusic.com, EN

Klady a zápory:

+ kombinace hudební databáze a sociální sítě
+ mnoho doplňkových možností pro registrované uživatele
+ rozsáhlé diskografie jednotlivých interpretů
- nepřehledně řazený seznam hudebníků do jednoho odstavce v profilu interpreta
- i přes anglickou lokalizaci uživatelé vkládají komentáře v jiných jazycích

Ultimate Music Database – umdmusic.com, EN / CZ

Klady a zápory:

+ velké množství interpretů
- nemožnost vkládat hodnocení a komentáře
- pouze profily interpretů, žádné profily nahrávek

Huddba.cz – huddba.cz, CZ

Klady a zápory:

+ široký žánrový záběr
+ možnost hodnotit i komentovat nahrávky
- seznam alb pouze podle roku vydání
- malý počet interpretů
- profil interpreta a jeho diskografie jsou duplicitní
- velké prodlevy v aktualizacích

Muzz.cz - muzz.cz, CZ

Klady a zápory:

+ možnost hodnotit jednotlivé skladby i jednotlivé interprety
- užší žánrové zaměření – především střední proud a alternativní žánry
- celkově nepřehledné rozvržení webu
- hodnocení systémem líbí/nelíbí

5.2 Návrh portálu

Ekvivalentní výrazy:

- Hudebník ~ člen kapely
- Album ~ nahrávka
- Interpret ~ kapela

Portál má obsahovat informace o hudebních interpretech a jejich nahrávkách. Databáze obsahuje interprety a k nim přiřazené nahrávky. Kromě informací nabízí portál uživatelům také možnost diskutovat o jednotlivých nahrávkách. Diskuze se mohou účastnit všichni návštěvníci webu bez ohledu na to, zda jsou registrováni či nikoliv.

Nejstěžejnější částí webu jsou hodnocení a komentáře jednotlivých hudebních nahrávek. Pokud je nahrávka ohodnocena, je u ní zobrazeno procentuální hodnocení, kdy 100% znamená nejlepší a 0% nejhorší. Hodnotit mohou uživatelé webu, kteří provedli jednoduchou a rychlou registraci zdarma. Dostanou tak výhodu oproti neregistrovaným uživatelům, v podobě psaní komentářů a bodového hodnocení nahrávek. Zároveň budou mít svůj profil a v něm seznam všech hodnocení a komentářů, které vložili. Budou mít tak poměrně dobrý přehled o své aktivitě na portálu.

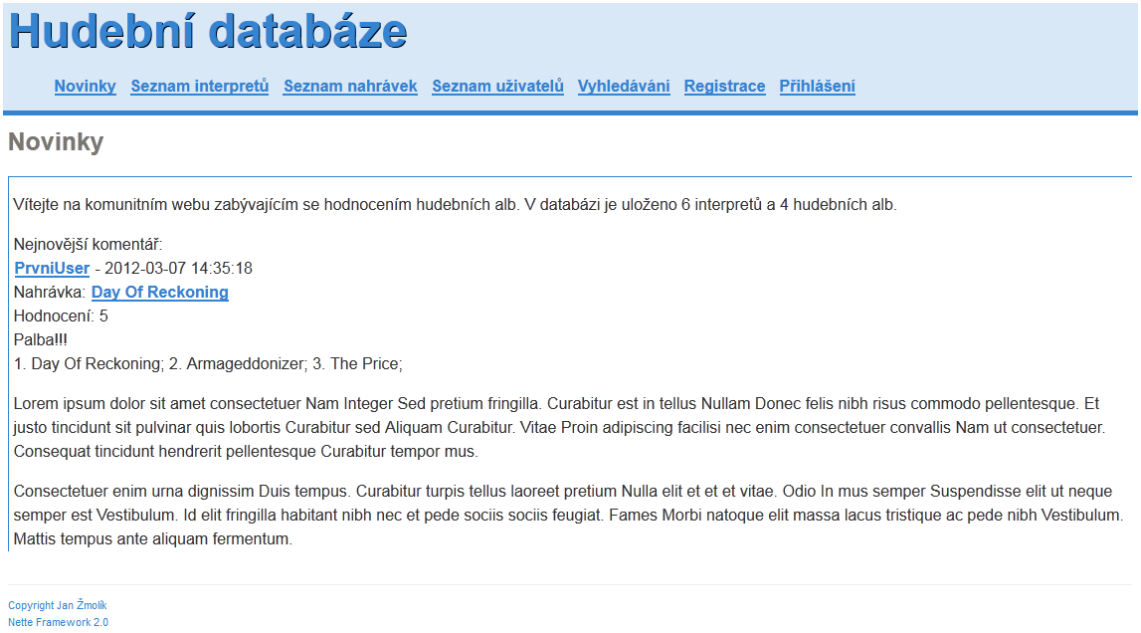
Podmínky hodnocení nahrávek jsou takové, že jeden uživatel může hodnotit jednu nahrávku právě jednou. Poté může kdykoliv svoje hodnocení upravit nebo smazat, ale nemůže se stát, že by mohl některé nahrávce přidat více hodnocení nebo komentářů.

Další podmínkou je, že pokud se uživatel rozhodně nahrávku ohodnotit, nemusí k ní nutně psát textový komentář. Není ale možné psát komentář a nedat nahrávce bodové hodnocení.

Zároveň s hodnocením může uživatel vybrat až tři skladby z alba, které považuje za nejlepší. Tyto skladby budou ohodnoceny podle pořadí body od tří do jednoho. Uživatel může dát jednu skladbu i na více míst (to znamená, že může jedné skladbě dát třeba 4 body – 3 a 1 bod). Toto pořadí bude následně vypsáno u jeho komentáře. Skladby není nutné hodnotit, podobně jako není nutné psát textový komentář.

Na webové adrese http://janzmolik-bp.g6.cz/BPrace/www/ je možné si aplikaci vyzkoušet. Pro přihlášení do administrátorské sekce lze použít přihlašovací jméno „Admin“ a jako heslo „heslo“.

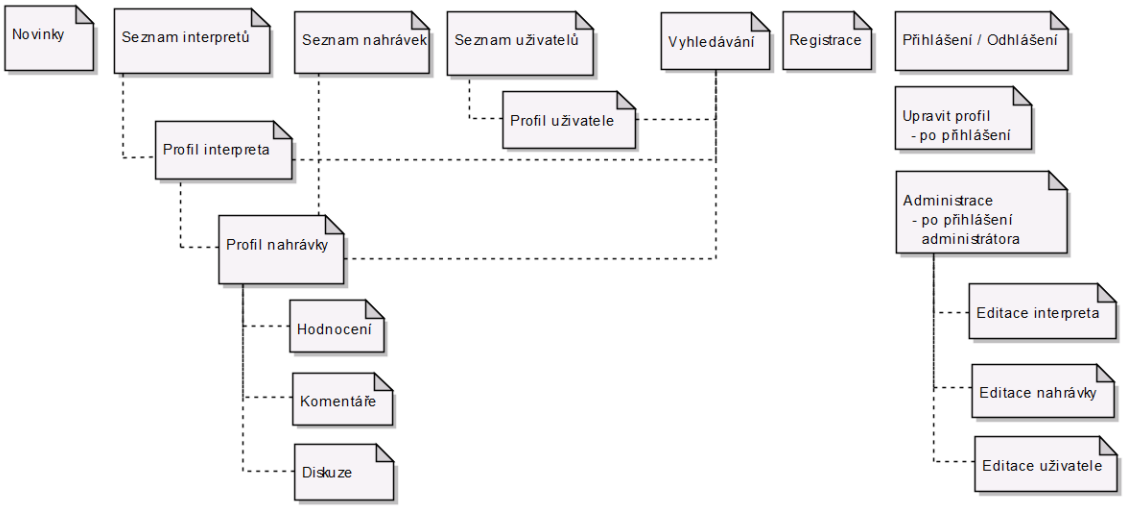
Základní vzhled portálu a jeho části



Obrázek 1 – Vzhled portálu

Obrázek 1 vyobrazuje základní vzhled webu. Hlavička (název a menu) a patička jsou zobrazeny pokaždé. Vlastní obsah webu je dynamicky měněn při procházení webových stránek.

Horizontální menu bylo zvoleno kvůli docílení většího prostoru v obsahové části webu.



Obrázek 2 – Mapa portálu

Obrázek 2 ilustruje mapu webu. Každá z jednotlivých stránek je reprezentována samostatnou webovou stránkou.

Adresářová struktura aplikace rozdělující jednotlivé části dle architektury MVP (viz kapitola 6.1) je následující:

- models – obsahuje všechny modely (*.php soubory)
- presenters – obsahuje všechny presentery (*.php soubory)
- templates – obsahuje soubor @layout.latte, což je šablona pro základní vzhled aplikace. Dále obsahuje složky, které mají stejné pojmenování jako názvy Presenterů – každá složka obsahuje sadu příslušných šablon (*.latte soubry) pro daný Presenter.

Šablonovací systém

Framework Nette používá k vykreslování obsahu webových stránek šablonovací systém Latte. Ten je součástí architektury MVP (viz kapitola 6.1) a šablony představují část view. Jedná se o soubory typu *.latte. Umožňují používat takzvaná latte makra, která obvykle obstarávají šablonovací logiku. Seznam výchozích maker lze nalézt na [9].

Šablona obsahuje standardní html značky, to znamená `<html>`, `<head>` a `<body>` a následně všechny, které jsou zapotřebí k vytvoření požadovaného obsahu. Šablony jsou tedy dokumenty, které obsahují latte makra a html kód. Systém Latte při

vykreslování šablony vykoná potřebná makra a dosadí za ně správné hodnoty a vykreslí webovou stránku podle html kódu.

Složka templates obsahuje základní vzhled webu, který obsahuje rozložení webu, soubor @layout.latte. Uvnitř tagu `<head>` je deklarován typ dokumentu a meta data. Uvnitř tagu `<body>` je blokový element `<div>`, který je nastylován, aby zarovnal celý web na střed obrazovky. Obsahuje název webu (hlavičku) a hlavní navigační menu. Menu je staticky vepsáno v kódu šablony. Díky tomu, že je tato šablona výchozí a všechny ostatní šablony jsou do ní doplňovány, je deklarace pouze na jednom místě a je snadné menu udržovat.

K vytváření odkazů je využito latte makra `{link Presenter:akce $parametr}`. Makro `{link}` vygeneruje odkaz. První parametr uvádí, kam bude odkaz směřovat – na uvedenou akci presenteru. Druhý parametr je předán přímo metodě presenteru. Co je to presenter je podrobněji popsáno v kapitole 6.1.

Po menu následuje makro `{include #content}`. Před název content je vložena mřížka, která označuje vkládání šablony, nikoliv souboru. Makro zajistí, že na tomto místě budou vkládány jednotlivé šablony od konkrétních presenterů.

Každá z šablon, které vykreslují konkrétní obsah webu a starají se o zobrazení dat uživateli (vyobrazení a vykreslení konkrétní stránky), jsou uvozeny makrem `{block content}`. Tímto makrem je blok definován. Makro `{include}` následně pozná, že má vkládat právě takto pojmenované šablony.

Pod vlastním obsahem následuje patička, která obsahuje informace o autorovi a o verzi použitého frameworku.

Seznamy interpretů, nahrávek a uživatelů

Pro lepší orientaci v databázi nabízí portál seznamy interpretů, nahrávek a uživatelů. U nahrávek a interpretů existuje předpoklad, že databáze bude obsahovat větší množství těchto dat (v řádech tisíců, až desetitisíců). Z toho se dá vyvodit, že od každého písmene abecedy bude v databázi minimálně jeden záznam.

Každý seznam obsahuje abecední menu, pro lepší orientaci v záznamech. Toto menu je, z výše uvedeného důvodu, generováno staticky. Není nutné menu generovat dynamicky na základě záznamů uložených v databázi.

I přes to obsahuje šablona pro vykreslení výsledků podmínku, která vypíše informační zprávu, pokud nejsou nalezeny žádné záznamy od zvoleného počátečního písmene. Tato ochrana je implementována především z důvodu, kdy bude portál spuštěn

a prvních pár dní bude probíhat plnění databáze, ale uživatelé již budou moci portál využívat.

Přizpůsobení prostředí frameworku

Framework Nette nabízí funkce pro autentizaci a autorizaci přihlášeného uživatele.

Framework pro tyto účely poskytuje třídy `Identity` a `User` (adresář BPrace\libs\Nette\Security) a třídu `Authenticator` (adresář BPrace\app\models). Třída Identity v sobě uchovává id a přístupová práva uživatele.

Pro potřeby aplikace bylo potřeba uchovávat zároveň i přezdívku přihlášeného uživatele. Je totiž používána na mnoha místech v aplikaci a je tak jednodušší ji uchovat pohromadě s ostatními identifikačními daty právě v objektu `Identity`, místo způsobu kdy by byl pokaždé odeslán dotaz do databáze a přezdívka vrácena základě uloženého id. Aplikace by tak byla zatěžována častými dotazy na databázi. Proto byl zvolen způsob modifikace třídy `Identity`, aby odpovídala požadavkům aplikace.

Třída `Identity` (adresář BPrace\libs\Nette\Security) byla upravena tak, že jí byl přidán privátní atribut `$nick` a zároveň přidány metody `setNick()` a `getNick()`, které objektu umožňují manipulaci s tímto atributem.

Zároveň bylo třeba upravit třídu `Authenticator` (adresář BPrace\app\models) – její metodu `authenticate()` tak, aby se názvy testovaných proměnných shodovaly s názvy sloupců v databázi (v kódu řádek 35) a při vytvoření nového objektu Identity předat jeho konstruktoru zároveň i přezdívku (v kódu řádek 46). U třídy Authenticator bylo třeba upravit i její druhou metodu `calculateHash()`, aby využívala stejný postup generování hashe, který je použit při registraci uživatele, viz 6.1.

Pokud se nyní bude chtít uživatel přihlásit je vytvořen nový objekt typu `User` a zavolána jeho metoda `login()`. Této metodě jsou předány nick a heslo, které uživatel zadal do přihlašovacího formuláře. Tato metoda má implementovanou logiku, která ověří uživatele, pomocí třídy `Authenticator` a nastaví mu identitu.

Ověřovací rutina

Vzhledem k potřebě zajistit, aby přihlášení uživatelé měli umožněn přístup ke komentování a hodnocení nahrávek a zároveň i k editaci jejich profilů (uživatelský frontend), je třeba ověřit, zda jsou skutečně přihlášeni.

Jelikož jedna z těchto podmínek je i v menu (změna Přihlášení/Odhlášení), které se nachází v šabloně layoutu, je třeba tuto rutinu provádět při vykreslování všech šablon.

Je proto vytvořen soubor prihlaseni.php (je uložen ve složce Presenters, ale sám presenterem není) obsahující ověřovací rutinu, zda je uživatel přihlášen. Tento soubor je vkládán PHP funkcí `include()` na začátek každé renderovací metody všech presenterů. V rutině je provedeno nastavení řídících proměnných, které ovlivní vykonávání kódu v metodě.

V šabloně je poté stejného chování docíleno vložením makra `{if $promenna}`, kde proměnná je shodná s proměnnou, která řídí průběh vykonání kódu v renderovací metodě.

Mohli bychom sice po prvním přihlášení nastavit příznak do persistentního parametru BasePresenteru od kterého všechny ostatní presentery dědí, ale vzhledem k povaze těchto dat není nejvhodnější je předávat v parametru v url.

Ověřovací rutina, také testuje, zda má přihlášený uživatel dostatečná práva pro vstup do administrační sekce, kde je možno editovat záznamy v databázi (uživatelský backend).

Modelová vrstva

V architektuře MVP, viz kapitola 6.1, modelová vrstva reprezentuje data a funkce pro práci s daty. Pro účely webové aplikace byl model vytvořen pomocí statických tříd, které nabízí metody k manipulaci s daty uloženými v databázi.

Toto řešení bylo zvoleno především z důvodu nestavového prostředí webových aplikací. Objekty vytvořené namapováním dat z databáze, by měli životnost pouze v rámci jednoho vykonání skriptu. Po dokončení skriptu jsou díky nestavovému prostředí, všechny instance smazány. Aplikace jako taková bude navíc data především zobrazovat. Z těchto důvodů, bylo zvoleno řešení, kdy data nebudou přímo reprezentována objekty – z důvodu menší zátěže serveru a větší rychlosti odpovídání na akce uživatele.

Model proto v této aplikaci slouží především k manipulaci a úpravě dat – ne přímo k jejich samotné reprezentaci.

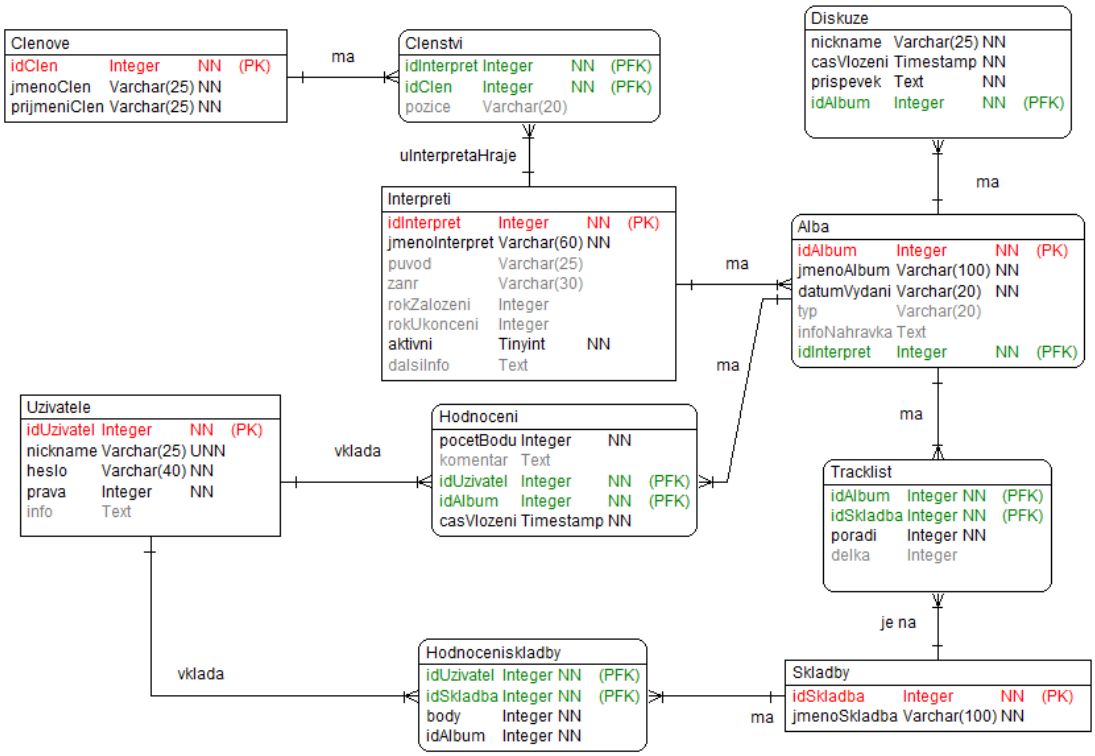
Komunikace s databází

Pro komunikaci s databází je využito knihovny Dibi. V [10] je uvedeno, že po navázání připojení k databázi (to se děje v souboru bootstrap.php – spouštěcí soubor frameworku Nette), je kdekoliv přístupná statická třída dibi a není nutné tak pokaždé získávat nebo si nějakým způsobem předávat instanci této třídy. Stačí uvést název třídy dibi:: a za čtyřtečkou uvést jméno požadované metody. Těmto metodám se v parametru předávají vlastní SQL příkazy, které jsou odeslány databázi.

Více informací lze nalézt na oficiální stránce [11].

5.3 Návrh databáze

Při návrhu databázového úložiště byl zohledněn fakt, že z důvodu větší přehlednosti propojovacích tabulek a jednoduší práce a manipulace s jednotlivými záznamy v rámci vlastní aplikace, budou mít interpreti, nahrávky, skladby, členové kapel a registrovaní uživatelé svoje id číslo. To jednoznačně určuje každý záznam v databázi a je tak primárním klíčem (Obrázek 3 – zobrazeny červeně). Toto řešení navíc umožňuje uložit do databáze například tři interprety stejného jména – pokaždé ale půjde o jiného interpreta. Ti se mohou lišit například původem, hudebním žánrem nebo rokem založení.



Obrázek 3 – Schéma databáze

Vazební tabulky, které obsahují cizí klíče (Obrázek 3 – zobrazeny zeleně), byly vytvořeny, za účelem dosažení vztahů M:N mezi některými tabulkami. Bližší informace jsou uvedeny v následujícím popisu jednotlivých tabulek.

Clenove

Do této tabulky jsou ukládány informace o jednotlivých hudebnících, kteří tvoří členy hudebních kapel (v databázi ukládaných do tabulky Interpreti). Každý hudebník má svoje identifikační číslo v databázi v podobě `idClen`. Dále tabulka obsahuje atributy `jmenoClen` a `prijmeniClen`.

Clenstvi

Tato tabulka zajišťuje vztah M:N mezi tabulkami Clenove a Interpreti. Je tím dosaženo stavu, kdy je kapela tvořena jedním nebo více hudebníky a zároveň jeden hudebník může působit v jedné či více kapelách najednou. Jeden hudebník může být v jedné kapele právě jednou. Tabulka obsahuje identifikační čísla hudebníka a interpreta (`idClen` a `idInterpret`) jako cizí klíče a aktuální pozici hudebníka v dané kapele (např. zpěv, kytara, bicí a další).

Diskuze

V této tabulce jsou uchovávány diskusní příspěvky k jednotlivým nahrávkám. Jelikož je v portálu umožněno vkládat příspěvky do diskuze i neregistrovaným uživatelům, obsahuje tabulka atribut `nickname` a není nijak provázaná s tabulkou Uzivatele, která obsahuje registrované uživatele na portálu.

Dále pak obsahuje atributy `casVlozeni`, `prispevek` a cizí klíč `idAlbum`. Tento atribut udává, ke kterému albu diskusní příspěvky patří, a je tím zajištěn vztah 1:N, kdy jeden příspěvek patří právě k jednomu albu, ovšem album má jeden nebo více diskusních příspěvků

Interpreti

Spolu s tabulkou Alba je toto nejdůležitější tabulka celé databáze. Ukládají se do ní všichni hudební interpreti a informace o nich. Každý interpret je jednoznačně určen svým identifikačním číslem `idInterpret`. Dále má atributy `jmenoInterpret`, `puvod`, `zanr`, `rokZalozeni`, `rokUkonceni`, `aktivni` a `dalsiInfo`.

Aktivita interpreta je rozebrána v kapitole 6.5.

Alba

Nejdůležitější tabulka v databázi. Ukládají se do ní nahrávky, které je možno následně na portálu hodnotit. Každá nahrávka má své identifikační číslo `idAlbum`. Dalšími atributy jsou `jmenoAlbum`, `datumVydani`, `typ` (LP, EP, singl, koncertní záznam, výběr), `infoNahravka` (obsahuje další textové informace o nahrávce) a `idInterpret`.

Jelikož každé album obsahuje id interpreta, je tak zajištěna relace 1:N. To znamená, že interpret má jedno nebo více alb, ale album patří pouze jednomu interpretovi. To znamená, že do databáze nelze uložit například kompilace, na kterých se podílí více umělců, nebo split-singly.

Uzivatele

Tato tabulka obsahuje všechny registrované uživatele na portálu. Každý uživatel má svoje identifikační číslo `idUzivatel`. Další atributy jsou `nickname`, `heslo` (na heslo je použita šifrovací funkce a uložen je výsledný hash), `prava` a `info`.

Atribut `prava` obsahuje číselnou hodnotu, která určuje administrační oprávnění přihlášeného uživatele. Standardní uživatel má hodnotu 1 a hlavní administrátor má hodnotu 3. Hodnota 2 je rezervována pro budoucí potřeby – například pro moderátory diskuzí, nebo pro uživatele s oprávněním vkládat nové záznamy do databáze.

Hodnoceni

Tabulka propojuje tabulky Uzivatele a Alba a zajišťuje tak mezi nimi vztah M:N. To znamená, že uživatel může hodnotit jednu nebo více nahrávek a zároveň jedna nahrávka je hodnocena jedním nebo více uživateli. Jeden uživatel může hodnotit jedno album právě jednou. Je tím dosaženo podmínek, které jsou uvedeny na začátku kapitoly 5.2.

Tabulka obsahuje atributy `pocetBodu`, `komentar`, `casVlozeni` a cizí klíče `idUzivatel` a `idAlbum`.

V atributu `pocetBodu` je uloženo číselné hodnocení nahrávky a v atributu `komentar` je uložen textový komentář k nahrávce. Do atributu `casVlozeni` je uložen čas odeslání hodnocení, eventuelně čas poslední úpravy hodnocení.

Tracklist

Tato tabulka propojuje tabulky Alba a Skladby. Tak je dosažena relace M:N mezi oběma tabulkami. To znamená, že nahrávka obsahuje jednu nebo více skladeb a zároveň jedna skladba může být na jedné nebo více nahrávkách.

Obsahuje atributy `idAlbum`, `idSkladba`, `poradi` a `delka`. Atribut `poradi` určuje, kolikátá v pořadí se skladba na albu nachází a `delka` určuje, jak je skladba na daném albu dlouhá.

Skladby

V této tabulce jsou uloženy jednotlivé hudební skladby. Každá skladba má svoje identifikační číslo `idSkladba`. Druhý atribut je `jmenoSkladba`, který obsahuje název skladby.

Hodnoceniskladby

Tato tabulka je doplňující tabulkou k tabulce Hodnoceni. Propojuje tabulky Uzivatele a Skladby. Každý uživatel může u každé nahrávky ohodnotit až tři skladby podle pořadí. Tyto hodnocení jsou ukládány do této tabulky. Tabulka obsahuje atributy `idUzivatel`, `idSkladba`, `idAlbum` a `body`.

6 Aplikace a využití návrhových vzorů

Tato část práce obsahuje popis vlastní implementace vybraných částí portálu. Zaměřuje se na použití návrhových vzorů, jejich výhod a nevýhod a využití možností nabízených zvoleným frameworkem.

Zdrojové soubory jsou na přiloženém CD v adresáři BPAplikace\app. Zde se nachází adresářová struktura, která byla popsána v kapitole 5.2.

Při volbě návrhových vzorů bylo využito informací uváděných v [12].

6.1 Architektura aplikace

Popis funkce

Při vývoji grafické aplikace, vzniká potřeba jednoznačně oddělit logiku aplikace od grafického prostředí.

Začínající programátor v PHP, obvykle sáhne po některém ze starších, nicméně stále funkčních tutoriálů na výuku základů jazyka PHP. Nešvarem těchto výukových materiálů je, že jsou většinou psány stylem, který kombinuje HTML a PHP kód v jednom monolitickém celku. Potenciální student si obvykle tento návyk osvojí a kód, který vzniká, když vytváří další webové aplikace, je mnohdy i přes správné členění kódu hůře čitelný a špatně se v něm orientuje.

Zvolený návrhový vzor

Model-View-Presenter (MVP) architektura nabízí řešení, jak oddělit grafické prostředí, aplikační logiku a data. Navíc také odděluje datovou část, tzn. reprezentaci dat v rámci aplikace. Tímto oddělením je značně zpřehledněn kód. Jednolitý celek kódu je rozdělen na více částí, které jsou obvykle reprezentovány třídami a objekty. Výsledný kód aplikace je pak snazší číst a upravovat.

Architektura má následující tři hlavní části:

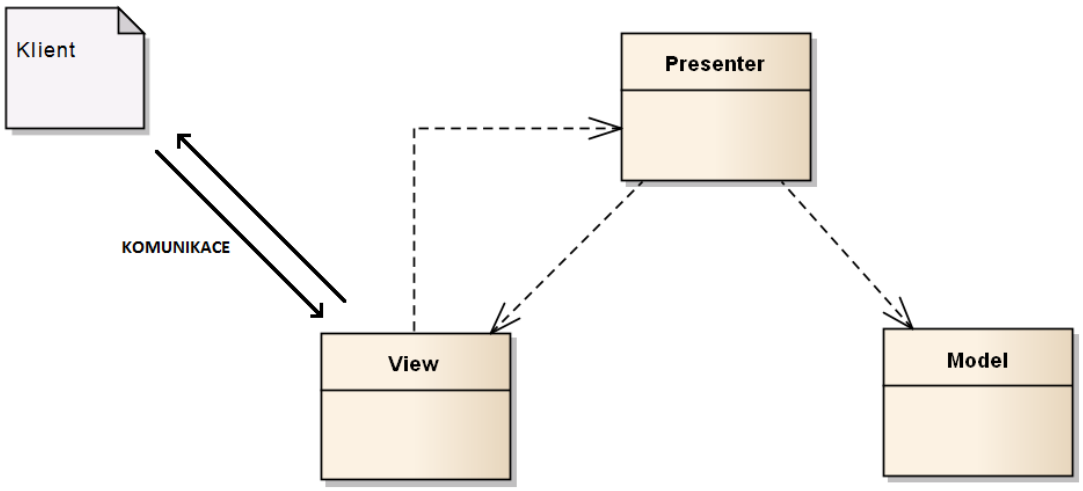
- Model: má přímý přístup k datům (obvykle k databázi) a umožňuje manipulaci s těmito daty. Obsahuje aplikační logiku a nabízí rozhraní pomocí, kterého s ním ostatní části aplikace komunikují.
- View: zajišťuje prezentaci dat na straně uživatele (obsahuje grafické uživatelské prostředí). [13] uvádí, že View v Nette představují jednotlivé šablony.
- Presenter: komunikuje s Modelem a View. Zajišťuje správnou reakci na požadavky uživatele – vyvolává příslušnou aplikační logiku, a následně požádá View o zobrazení výsledku.

Použití vzoru

[14] uvádí, že framework Nette implementuje MVC v podobě MVP, kde P označuje Presenter a je obdobou Controlleru. Nejzásadnějším rozdílem mezi MVC a MVP je podle [15] ten, že u MVC se o zachytávání uživatelského vstupu stará Controller, kdežto u MVP se o toto stará View, který obvykle po kliknutí myši volá některou z metod Presenteru.

Jak je uvedeno v [16] na počátku vývoje frameworku Nette stála myšlenka: „nechť je odkaz totéž, co zavolání funkce.“ Z toho také vychází celkové chování architektury MVP, tak jak je naimplementována a jak funguje ve frameworku Nette.

V Presenteru je pro každou vykreslitelnou šablonu, definována metoda s názvem `renderXXX()`, kde XXX značí název vykreslované šablony. V případě, že uživatel chce zobrazit danou šablonu je zavolána metoda z příslušného Presenteru.



Obrázek 4 – Schéma MVP architektury v Nette

Klient komunikuje s View – předává požadavky, dostává zobrazené výsledky. Presenter zpracovává požadavky od klienta a vyvolá aplikační logiku modelu. Presenter obdržená data od Modelu předá View spolu s žádostí o jejich vykreslení.

Jako konkrétní příklad lze uvést sekci Registrace. V `SignPresenteru` je definován formulář pro registraci, viz Obrázek 5.

```
public function createComponentRegistrace()
{
  $form = new UI\Form;

  $form->addGroup('Registrační údaje')
   ->setOption('description', 'Zadejte:');

  $form->addText('nickname', 'Přezdívka:')
   ->setRequired('Zadejte přezdívku');

   $form->addPassword('heslo', 'Heslo:')
   ->setRequired('Zadejte heslo');

  $form->addSubmit('send', 'Odeslat');

  $form->onSuccess[] = callback($this, 'registraceOdeslana');

  return $form;
}
```

Obrázek 5 – Zdrojový kód registračního formuláře

Šablona registrace obsahuje makro `{control registrace}`, které tento formulář vykreslí.

Uživatel je vyzván k zadání přezdívky a hesla. Po odeslání formuláře, pokud vyplil všechna pole, je předána presenteru informace, že uživatel provedl akci. Presenter

se nyní postará o její obsloužení. Data z formuláře jsou zpracována v metodě presenteru `registraceOdeslana()`.

```
public function registraceOdeslana($form)
{
  $data = $form->getValues();

  if (UzivatelModel::kontrola($data['nickname'])){
      $data['heslo'] = UzivatelModel::hash($data['heslo']);

      UzivatelModel::registrace($data);
      $this->flashMessage('Registrace proběhla úspěšně. Můžete se přihlásit.');

      $this->redirect("in");
  } else {
      $this->flashMessage('Vámi vybraná přezdívka je již zaregistrována. Vyberte si prosím jinou.');

      $this->redirect("this");
  }
}
```

Obrázek 6 – Zdrojový kód registrační metody

K registraci je využito metod statické třídy `UzivatelModel` z modelové vrstvy, viz Obrázek 6. V tomto případě se jedná nejdříve o metodu `kontorola()`, která v parametru obdrží uživatelem zadanou přezdívku. Metoda zjistí, zda již databáze shodnou přezdívku obsahuje, či nikoliv. V případě, že ano, vrátí metoda hodnotu `FALSE` a v presenteru je vykonána `else` větev, která zobrazí zprávu o nutnosti výběru jiné přezdívky a pošle příkaz šabloně registrace.latte k opětovnému vykreslení.

V případě, že se v databázi přezdívka nenachází je vrácena metodou `kontrola()` hodnota `TRUE` a je přistoupeno k vložení dat o uživateli do databáze.

Nejdříve je zavolána metoda `hash()` třídy `UzivatelModel`. Té je předáno heslo, které uživatel zadal. Metoda provede kombinaci hesla s předvolenou solí a zašifruje řetězec pomocí funkce md5. Výsledný hash je metodou vrácen a uložen do proměnné, která obsahovala původní heslo v nezašifrované podobě. Tento hash je porovnáván při přihlašování uživatele.

Způsob šifrování je zvolen spíše demonstračně a pro reálné použití bude lepší ho jednoduše upravit například přidáním přezdívky do šifrovaného řetězce nebo generováním dvou řetězců z hesla a přezdívky, následným spojením řetězců a vygenerováním nového řetězce. Navíc se nabízí použit i jiné šifrovací funkce než pouze md5, například sha.

Po vrácení výsledného hashe je zavolána metoda `registrace()` třídy `UzivatelModel`. Té jsou předána data – přezdívka a výsledek šifrovací funkce. V této metodě je nastavena hodnota proměnné `$data['prava']` na 1. Tím je zajištěno, že uživatel dostane základní přihlašovací práva a nezíská po přihlášení přístup

do administrační části. Následně jsou všechna data zapsána do databáze. Ta automaticky přidělí nově zaregistrovanému uživateli `idUzivatel` – vždy o jedna větší než je nejvyšší aktuální hodnota v tomto sloupci.

Následně presenter předá příkaz šabloně in.latte k vykreslení a předá jí textovou zprávu o úspěšné registraci. Šablona vykreslí přihlašovací formulář (stejným způsobem jako vykreslovala registrační formulář šablona registrace.latte) a zobrazí informační zprávu o dokončení registrace.

Zhodnocení

Využít MVC nebo MVP architekturu u webové aplikace je výhodné, pokud je vytvářen větší projekt a existuje předpoklad, že v budoucnu bude aplikace značně rozvíjena. Navíc pokud se programátor rozhodne využít některý z frameworků pro webové aplikace, obvykle se setká s tím, že je tato architektura již implementována v rámci frameworku.

Menší komplikace, v podobě prodloužení práce, může nastat v případě, kdy se programátor teprve s koncepcí MVP seznamuje. Je tedy nutné zohlednit vlastní schopnosti a při přechodu na MVP architekturu se nesnažit psát hned rozsáhlé projekty, kde může neznalost základní koncepce přinést zbytečné chyby, ale začít na jednodušších aplikacích.

Takto navržená architektura zajišťuje jednoduchou možnost úprav nebo rozšíření aplikace v budoucnu. Také umožňuje samostatný vývoj všech tří vrstev. Návrh aplikace bude flexibilnější a robustnější a bude snazší ho udržovat.

V případě webových aplikací je nutné počítat s omezením plynoucím z komunikace pomocí HTTP. Jelikož je tento protokol nestavový, webová aplikace si nemůže předávat proměnné mezi jednotlivými požadavky. Je proto nutné zajistit toto předávání, aby bylo možné využít principů návrhových vzorů. V této práci byla zvolena metoda, přenášení proměnných a objektů pomocí session.

Framework Nette k tomuto účelu nabízí persistentní parametry, které lze nastavit u presenterů. Tyto atributy se poté přenáší mezi jednotlivými metodami presenteru bez nutnosti je předávat ručně. Nevýhodou je, že Nette neumožňuje do persistentních parametrů ukládat objekty.

6.2 Vyhledávání – Strategy

Popis funkce

Uživatel má možnost vyhledat záznam v databázi a má při tom na výběr ze čtyř kritérií, podle kterých hledat. Poté, co zadá do textového pole hledaný řetězec, zvolí z roletkového menu kritérium. Po odeslání formuláře jsou zobrazeny výsledky.

Zvolený návrhový vzor

Jelikož je výběr kritéria vyhledávání ponechán na klientovi a vyhledávání se pokaždé řeší jiným způsobem, nabízí se pro tento účel návrhový vzor Strategy.

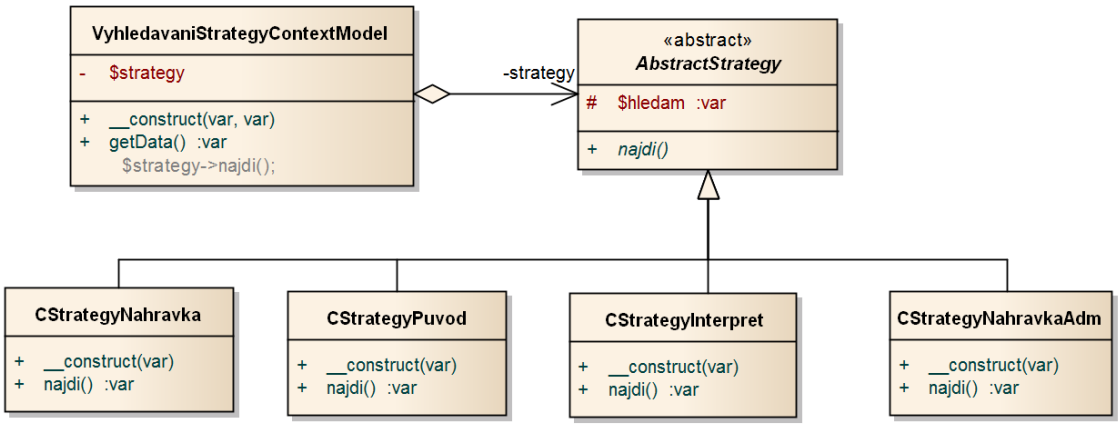
V [17] je uvedeno, že tento návrhový vzor umožňuje zapouzdřit algoritmy do samostatných objektů a jejich následnou záměnu. To, který algoritmus bude vybrán, je ovlivněno volbou klienta / uživatele.

Hlavní podmínkou pro použití tohoto vzoru je existence více podobných řešení jednoho problému. Jedná se o rodinu algoritmů, které rozdílným způsobem řeší stejnou funkci programu. Vzor má také zajistit znovu použitelnost algoritmů. Vzhledem k tomu, že algoritmy jsou implementovány jako jednotlivé objekty, je možné obejít se bez implementace jedné velké části kódu, která by obsahovala mnoho podmínek. Tím se zvyšuje čitelnost a právě znovu použitelnost jednotlivých algoritmů.

Použití vzoru

V případě, že se klient rozhodne vyhledávat, je mu zobrazen formulář, do kterého zadá vyhledávaný řetězec a zvolí kritérium, jaký typ dat chce vyhledat. Každé kritérium je v aplikaci implementováno a reprezentováno jedním objektem concrete strategy.

Po odeslání jsou tyto informace předány metodě `zpracujVysledky()`, kterou obsahuje `VyhledavaniPresenter`. Ta zajistí vytvoření instance modelu `VyhledavaniStrategyContextModel`. Vytvoří se tak kontext, se kterým uživatel komunikuje a ovládá tak celé vyhledávání. Jak je vidět v souboru VyhledavaniPresenter.php na řádku 54 jsou konstruktoru předány informace, které zadal klient. V souboru VyhledavaniStrategyContextModel.php lze vidět od řádku 16, jak je proveden výběr a uložení správné strategie, na základě požadavku klienta. Odkaz na objekt obsahující zvolenou strategii si kontext uloží do své privátní proměnné `$strategy`.



Obrázek 7 – Diagram implementace vzoru Strategy

Po vytvoření instance `VyhledavaniStrategyContextModel` je zavolána jeho metoda `getData()`. V souboru VyhledavaniStrategyContextModel.php je na řádku 45 vidět, že v metodě `getData()`, je zavolána metoda `najdi()` uložené strategie, viz Obrázek 7. Ta provádí příkaz, který v databázi vyhledá řetězec zadaný uživatelem.

```
class CStrategyNahravka extends AbstractStrategy {

    public function __construct($hledam) {
        $this->hledam = $hledam;
    }

    public function najdi() {
        return dibi::fetchAll('
            SELECT [jmenoAlbum], [datumVydani], [idAlbum], [jmenoInterpret]
            FROM [alba]
            JOIN [interpreti]
            USING (idInterpret)
            WHERE [jmenoAlbum] LIKE %~like~
            ORDER BY [jmenoAlbum]', $this->hledam
        );
    }
}
```

Obrázek 8 – Zdrojový kód třídy concrete strategy

Obrázek 8 vyobrazuje ukázku jedné ze tříd Concrete Strategy. Její metoda `najdi()` se postará o vyhledání záznamů pomocí odeslání SQL dotazu do databáze. Výsledek předá zpět presenteru.

Poté, co tímto způsobem obdrží `VyhledavaniPresenter` výsledky, odešle šabloně vysledky.latte příkaz k vykreslení a data z databáze. Tato šablona zobrazí klientovi nalezená data. To zda bylo něco nalezeno či nikoliv již není starost vzoru Strategy, ale samotného view. Při přesměrování na šablonu zobrazující nalezená data

instance Kontextu zaniká a při novém vyhledávání je tedy celá procedura opakována od začátku.

Původní návrh vyhledávání počítal s implementací tří konkrétních strategií, aby každá mohla obsluhovat jedno kritérium. Během dalšího vývoje aplikace bylo třeba v administrátorské sekci vypisovat všechny nahrávky, jejichž názvy začínají stejným písmenem. Jelikož tato funkce odpovídá vyhledávání, nabízelo se jednoduché řešení vytvořit instanci kontextu vzoru Strategy a vyhledat pomocí něj potřebná data. ConcreteStretegy, která je zde kontextu předána byla implementována dodatečně a lze tím demonstrovat snadnou rozšiřitelnost aplikace o nové strategie. Aby kontext o této strategii „věděl“, bylo třeba pouze malé úpravy v jeho konstruktoru, přidáním case větve od řádku 30 v souboru VyhledavaniStrategyContextModel.php.

Zhodnocení

Takto implementovaný návrhový vzor Strategy nepočítá se znovupoužitím instance Kontextu, ta totiž zaniká při přesměrování na stránku s výsledky a při dalším vyhledávání se vytváří instance nová. Tímto způsobem nelze dynamicky přepínat mezi strategiemi, což ovšem v tomto konkrétním případě nevadí. Ovšem ve chvíli, kdy by bylo třeba strategii změnit, bude nutné kontext zachovat a předat (např. session), protože jinak by docházelo ke stálému vytváření nových instancí.

V případě, kdy je nutné zachovat kontext, je nezbytné doimplementovat do kontextu jednoduchou metodu, která změní strategii na základě předaného parametru. Takovýto postup bude vhodný především v případě, že s výsledky hledání rovnou zobrazíme i formulář pro nové vyhledávání. Nebylo by již třeba vytvářet novou instanci kontextu, ale stačilo by pouze zavolat metodu `setNewStrategy()` a předat jí jako parametr novou strategii.

Je tedy třeba si především uvědomit, na co, jak a kde chceme tento vzor používat. Je nutné dopředu počítat s použitím prostředku pro předávání kontextu, i kdyby to nakonec nebylo nezbytné.

Jedná se tedy o ukázku toho, že návrhový vzor nemusíme nutně chápat zcela dogmaticky, ale zároveň si musíme uvědomit, že tato implementace sice funguje, ale není zcela správná. Je proto lepší naučit se používat vzory, tak jak se předpokládá. Mohlo by se poté stát, že budeme chtít použít stejný vzor v nějakém ze silně typovaných jazyků a narazíme na problémy, kdy něco nebude fungovat, i přes to, že budeme mít v povědomí, že jinde to fungovalo.

Je nutné uvést také poznámku o dynamickém a slabém typování jazyka PHP. Na základě těchto vlastností totiž není nezbytné definovat konkrétním strategiím společného předka a aplikace přesto bude fungovat. PHP umožňuje do proměnné `$strategy` třídy `VyhledavaniStrategyContextModel` vložit bez problémů prakticky cokoliv. To znamená, že nejen odkaz na některou z konkrétních strategií, ale třeba textový řetězec nebo číslo. Je nutné toto brát v potaz a v případě práce na projektu ve více lidech zajistit, aby nemohl být obsah proměnné změněn „někým z venku".

6.3 Administrace – State

Popis funkce

Klient, v tomto případě uživatel s právy administrátora, má přístup do administrační části aplikace. V této sekci má možnost editovat data obsažená v databázi. Může editovat nahrávky, interprety a smazat registrované uživatele.

Zvolený návrhový vzor

Pokaždé se jedná o úpravu dat v databázi, ale vždy jde o data jiného typu. Bude tedy vhodné využít jednoho objektu, který nabídne klientovi metody ke správě databázových dat. Tímto objektem je kontext z návrhového vzoru State, u kterého bude stav dynamicky měněn podle toho, která data zrovna administrátor/klient potřebuje upravit. Klient se tedy vůbec nezajímá o to, jak je zařízeno, že jsou vždy provedeny operace nad správnými daty. Změna stavu v kontextu je prováděna automaticky – při zvolení upravovaných dat je kontextu předán stav, který k těmto datům náleží.

[18] uvádí, že umožňuje změnu chování objektu na základě jeho vnitřního stavu.

Použití vzoru

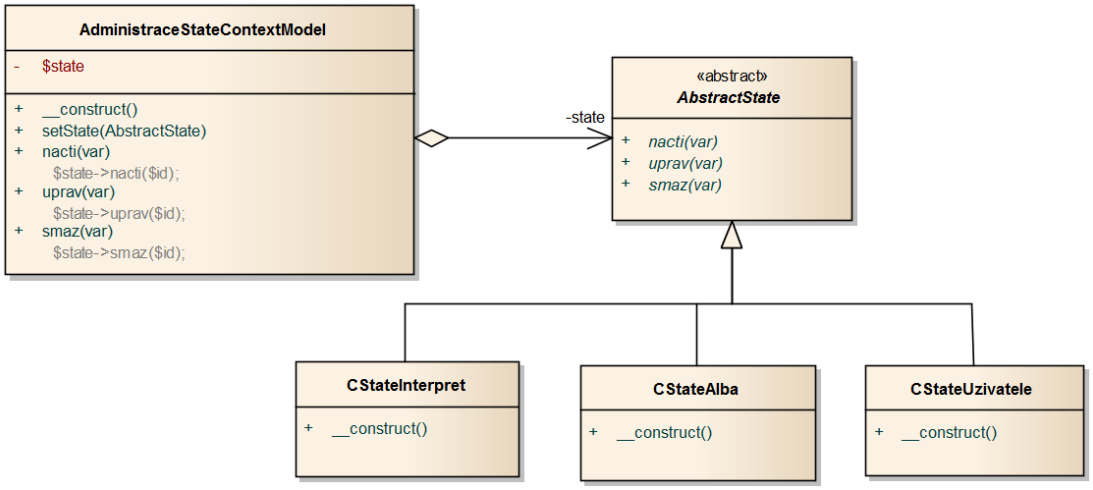
Pokud administrátor vstoupí do administrační sekce, je mu jako úvodní nabídka zobrazeno staticky generované abecední menu. V této části je v `AdminPresenteru` vytvořena instance objektu `AdministraceStateContextModel` (viz soubor AdminPresenter.php, řádek 24). Pomocí tohoto objektu uživatel provádí veškeré požadavky. Třída `AdministraceStateContextModel` je implementována jako Singleton, což je rozvedeno dále v části Administrace – Singleton.

Při zvolení počátečního písmene, jsou zobrazeny tři tabulky obsahující interprety, nahrávky a uživatele. Objekty konkrétních stavů jsou vytvářeny ve chvíli, kdy administrátor zvolí některý ze záznamů z databáze a bude ho chtít editovat.

V případě, že zvolí editaci interpreta je vytvořen objekt `CStateInterpret` a tato instance je předána kontextu jeho metodou `setState()` jako parametr. Je tím nastaven stav, který odpovídá editaci interpretů. Ten kromě toho, že je uložen v kontextu do atributu `$state`, je také uložen do session, aby se při návratu k administraci interpretů nevytvářela nová instance třídy `CStateInterpret`, ale byla použita již vytvořená, jak to předepisuje návrhový vzor.

Na stránce je vykreslen formulář pro editaci interpreta, ten je po odeslání obsloužen příslušným stavem uloženým v kontextu.

Pro zjištění existence instance stavového objektu je na začátek skriptu přidána podmínka. Pokud již stav někdy nastal, instance se nevytváří, ale použije se ta, která je uložená v session.



Obrázek 9 – Diagram implementace vzoru State

V případě, že administrátor chce editovat nahrávky nebo uživatele, je postup analogický. V metodě `AdminPresenteru`, která přísluší k editaci nahrávek nebo uživatelů, je vytvořen další concrete state – `CStateAlba` nebo `CStateUzivatele`. Ten je opět předán kontextu a tím pozměněna jeho funkce. Zároveň je uložen do session, ze stejného důvodu jako předcházející `CStateInterpret`. Když tedy bude administrátor upravovat některý ze záznamů, budou volány metody odpovídajícího stavu, viz Obrázek 9.

Podle [18] jsou dva přístupy k tomu, zda zachovávat objekty concrete state v paměti. První je vytvoření objektů, až ve chvíli jejich potřeby a jejich následné zničení ihned po použití. Druhý je takový, že jsou všechny objekty concrete state vytvořeny najednou a poté uchovávány v paměti pro následné použití.

V této implementaci je využitá kombinace obou přístupů. Objekty concrete state jsou v paměti vytvořeny až ve chvíli, kdy jsou potřeba. Následně jsou v paměti uchovány a předávány pomocí session a dosazovány pomocí metody `setState()` do kontextu. Je tím dosaženo menší paměťové náročnosti. Neaktivní concrete state objekty čekají v paměti, než bude vyvolán stav, ke kterému náleží. V případě, že některý ze stavů vůbec nenastane, není třeba vytvářet instanci k němu přiřazeného objektu. Je tak dosaženo toho, že aplikace je lazy a v paměti se opět šetří místo.

Jako příklad fungování je zde uvedena volba smazání záznamu z databáze. Ve chvíli, kdy administrátor v editačním formuláři zadá volbu „Smazat z databáze" u některého ze záznamů je zavolána metoda `AdminPresenteru smazat()`, která zpracuje požadavek (soubor AdminPresenter.php, řádek 248). V metodě je zavolána metoda `getInstance()` třídy `AdministraceStateContextModel` (viz kapitola 6.4) a tím je získána instance kontextu, který obsahuje aktuálním stav. Následně je zavolána metoda kontextu `smaz()`. Zde volání metod kontextu funguje na stejném principu jako u vzoru Strategy. To znamená, že v metodě kontextu je volána adekvátní metoda uloženého objektu concrete state. Po vykonání akce je zobrazen opět výpis záznamů v databázi. Toho je docíleno pomocí funkce redirect, kterou nabízí framework. Ta obsahuje dva parametry. První je název šablony, která bude po přesměrování vykreslena. Tou je šablona prehled. Druhý parametr ve funkci redirect je předán metodě, která souvisí s vykreslovanou šablonou. V tomto případě se jedná o metodu `renderPrehled()` presenteru `AdminPresenter`.

Zhodnocení

Při použití tohoto vzoru v jazyce PHP je třeba zajistit, aby byl zachován objekt kontext a zvolit přístup k vytváření instancí tříd concrete state. V případě, že by nebyly uchovány a místo toho by byly na začátku vykonávání každého skriptu znovu vytvořeny a na konci zničeny, je třeba vzít v potaz, zda toto neustálé vytváření objektů nebude mít závažný dopad na dostupnou paměť. Jak uvádí [19] v bodě 293, záleží na ceně vytvoření těchto instancí.

V aplikaci je tento problém řešen pomocí předávání kriticky důležitých objektů (všechny vytvořené concrete state) pomocí session. Uchování kontextu je podrobněji rozvedeno v kapitole 6.4.

Stejně jako u vzoru Strategy (kapitola 6.2) je nutné díky slabému a dynamicky typovanému PHP zajistit, aby kontext vždy obsahoval referenci na správný typ objektu

– tedy některý z konkrétních stavů. V opačném případě, by mohlo dojít ke kritické chybě v aplikaci.

6.4 Administrace – Singleton

Popis funkce

Administrátor má možnost upravovat data v databázi. Je třeba zajistit, aby objekt, přes který se tyto úpravy provádí a je zároveň vstupním bodem do administrace, byl pouze jeden.

Zvolený návrhový vzor

Jak již bylo naznačeno v předchozí kapitole 6.3, je kontext z návrhového vzoru State implementován zároveň jako Singleton. V administrační části je žádoucí vytvořit pro administrátora (klienta) pouze jednu instanci kontextu, se kterou bude komunikovat po celou dobu editací a tak vytvořit pouze jeden přístupový bod do administrace. Toho dosáhneme použitím vzoru Singleton, který zajistí, že bude v paměti po celou dobu vytvořena pouze jedna instance kontextu a objekt sám o sobě již nedovolí vytvořit instanci další.

V [20] je uvedeno, že vzor má za účel zajistit vytvoření pouze jedné instance od dané a poskytnou k ní globální přístup.

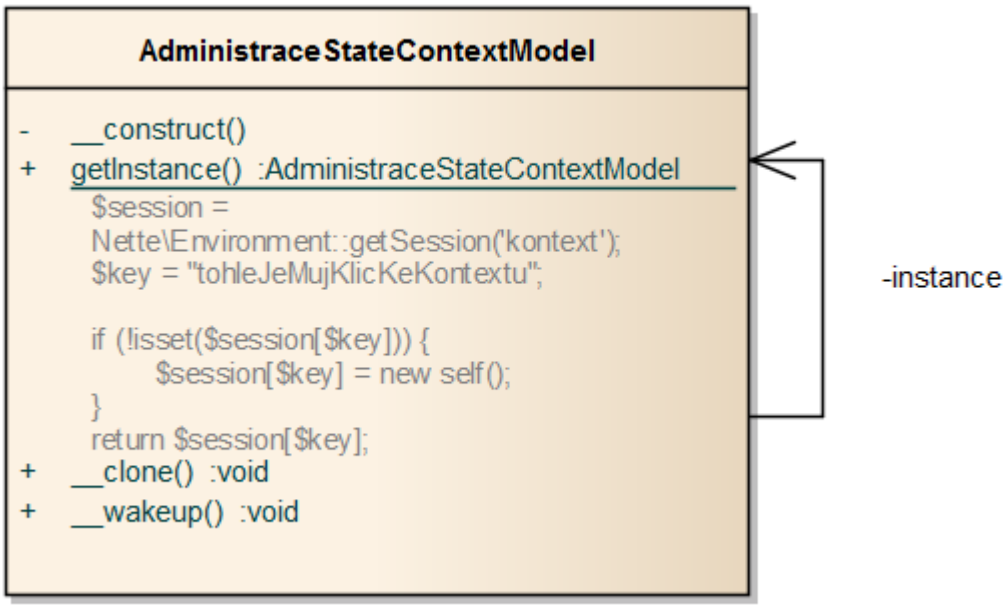
Použití vzoru

U webových aplikací v PHP tento vzor naráží na nestavové prostředí. Po vykonání skriptu jsou všechny instance tříd zničeny. Není zachován ani obsah statických atributů tříd. To znamená, že je vhodné použít v tomto vzoru jinou formu uložení jediné instance, než právě ukládání do statické proměnné instanciované třídy. V aplikaci je vzor implementován za využití session.

V souboru AdminPresenter.php na řádku 24 je vidět, že instance je získána statickou metodou `getInstance()` třídy `AdministraceStateContextModel` a není vytvářena pomocí konstruktoru.

Tato metoda zjistí, zda instance kontextu nebyla vytvořena a uložena do session. Pokud ne je v této metodě otevřena session a vytvořena nová instance Kontextu. Ta je následně uložena do otevřené session, která zde supluje výše zmiňovaný statický atribut třídy, do kterého je obvykle jediná instance třídy ukládána (soubor

AdministraceStateContextModel.php, řádek 22, také viz Obrázek 10). Metoda vytvořenou instanci vrátí do `AdminPresenteru`, kde s ní je dále pracováno.



Obrázek 10 – Diagram implementace vzoru Singleton

Jak již bylo uvedeno objekt implementovaný jako Singleton, je nutné nějakým způsobem předávat. V případě, že vytvoříme instanci v jedné sekci a objekt nepředáme, tak se bude opět vytvářet nová, jelikož ta původní po vykreslení šablony související s aktuální metodou presenteru zaniká. Jiná metoda presenteru, která používá stejný objekt, by tak nevěděla o tom, že již nějaká instance dříve existovala. Došlo by tím tak k neustálému vytváření nových instancí, což neodpovídá funkci návrhového vzoru Singleton.

V dalších metodách presenteru, které využívají instanci kontextu, nyní stačí opět zavolat statickou metodu `getInstance()` třídy `AdministraceStateContextModel`. Ta otevře session a vrátí instanci v ní uloženou a nic nevytváří, viz Obrázek 10.

I přes to, že je kontext vytvořen pouze jednou a následně předáván pomocí session, je stále nutné zabránit možnosti vytvoření další instance. V PHP tohoto dosáhneme několika kroky. Prvním, jak již bylo uvedeno, je, že třída by měla mít statickou proměnnou, kde bude uložená její jediná instance. V této implementaci tuto proměnnou zastupuje uložení do session. Také musí mít statickou metodu, která jedinou instanci vrací.

Dalším krokem je nastavení privátního konstruktoru. Tím, že zajištěno, že nebude možné v aplikaci vytvořit další instance voláním operátoru `new`.

V PHP je navíc nutné brát v potaz všechny funkce, které jazyk nabízí pro klonování, kopírování a vytváření objektů. Je tedy nutné přepsat metody `__clone()` a `__wakeup()`, které nabízí samo PHP. Jeden z možných způsobů ilustruje Obrázek 11. Tento kód je k nalezení na [21].

```
public function __clone()
{
    trigger_error('Clone is not allowed.', E_USER_ERROR);
}

public function __wakeup()
{
    trigger_error('Unserializing is not allowed.', E_USER_ERROR);
}
```

Obrázek 11 – Ukázka úprav metod třídy Singleton

Zhodnocení

Opět zde narážíme na problém spojený s tím, že všechny proměnné a objekty jsou po vykonání PHP skriptu zrušeny a samy se neuchovávají v žádné paměti. Pokud by toto nebylo zajištěno, existovala by sice v paměti vždy pouze jedna instance singletonu, což je v pořádku, ovšem tato instance by se vytvářela a zanikala vždy pouze s jedním během skriptu. Tím by v paměti nezůstala žádná instance dané třídy. To ale neodpovídá povaze návrhového vzoru. Navíc s takovýmto přístupem, by nešlo použít singleton například k uchování stavové informace.

U tohoto vzoru je tedy potřeba zajistit předávání vytvořené instance mezi jednotlivými částmi aplikace, v této konkrétní implementaci pomocí session. Pak je skutečně možné využít účel návrhového vzoru a vytvořit tak jednu instanci objektu. V tomto případě byl návrhový vzor Singleton užit v kombinaci s kontextem návrhového vzoru State Pattern, viz kapitola 6.3. Tím bylo zajištěno, že v administraci nevznikne více než jedna instance kontextu.

Je nutné se opět zmínit o typování PHP. Jelikož je odkaz na instanci ukládán do session, je nezbytné opět zajistit nemožnost modifikace obsahu této session. Zde je dobré dbát na zásady skrývání implementace. V momentě kdy neposkytneme informace o tom, jak instanci uchováváme, předejdeme tím problémům s možnou záměnou kontextu za jiný objekt.

6.5 Aktivita interpretů – Factory

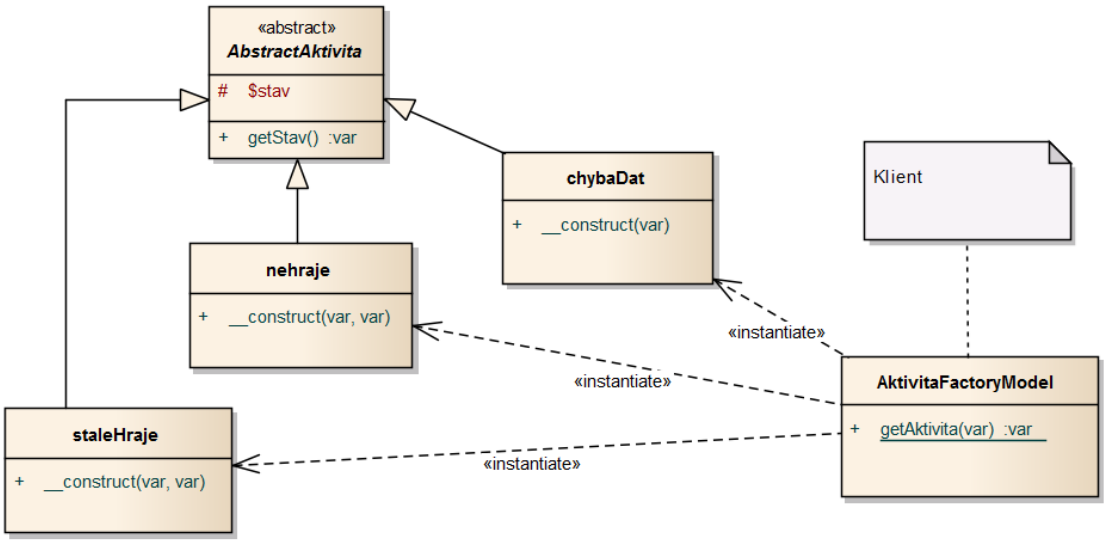
Popis funkce

U interpreta je na základě dat v databázi zobrazena jeho aktivita (rok založení, rok ukončení, a zda hraje, či nikoliv). Před samotným načtením dat z databáze a před vykonáním programového kódu, není možné určit, jakým způsobem má být naloženo s informacemi z databáze – jaká je má zpracovat třída. Je proto vhodné tyto informace předat tovární metodě určené třídy, která na základě těchto dat vybere správný objekt. Klientovi vrátí instanci vybraného objektu, který data následně zpracuje. Je tedy vhodné zvolit návrhový vzor Factory.

Zvolený návrhový vzor

Podle [22] návrhový vzor Factory řeší problém, kdy během nebo celkově před spuštěním programu, nevíme, jaký bude přesně potřeba na určitém místě objekt a jakým způsobem ho vybrat až za běhu programu. Můžeme mít totiž více druhů objektů, které dědí od jednoho předka, ale každý z těchto objektů může provádět nad daty různé operace. Toto využijeme v případě, že je potřeba jednu situaci řešit různými způsoby, ale dopředu nevíme, který ze způsobů bude třeba vybrat. Objekt Factory, kterému uživatel předá vstupní parametry, vytvoří na základě těchto parametrů a implementované logiky výběru adekvátní objekt se kterým bude uživatel dále pracovat.

Použití vzoru



Obrázek 12 – Diagram implementace vzoru Singleton

Následující tabulka ilustruje všechny možné kombinace proměnných uložených v databázi.

Tabulka 1 – Aktivita interpreta

Rok založení	Rok ukončení	Je kapela aktivní?	Výsledný stav
`NULL`	`NULL`	0	Interpret již nehraje; není známo období, kdy působil
`NULL`	`NULL`	1	Interpret stále hraje; není znám rok založení
`NULL`	Hodnota	0	Interpret již nehraje; není znám rok založení, ale je znám rok ukončení
`NULL`	Hodnota	1	Zakázaný stav – Interpret má příznak, že hraje, ale zároveň má i rok ukončení.
Hodnota	`NULL`	0	Interpret již nehraje; je znám rok založení, ale není znám rok ukončení
Hodnota	`NULL`	1	Interpret hraje a je znám rok založení
Hodnota	Hodnota	0	Interpret již nehraje a je znám rok založení i ukončení
Hodnota	Hodnota	1	Zakázaný stav – Interpret má příznak, že hraje, ale zároveň má i rok ukončení.

V případě, že uživatel zadá příkaz k vykreslení stránky s interpretem je předáno řízení aplikace metodě `renderProfil()` `InterpretiPresenteru`. Zde jsou od modelu `InterpretModel` zprostředkovávajícího komunikaci s databází přebrány data o interpretovi na základě jeho id. Informace o aktivitě interpreta jsou následně předána jako parametr tovární metodě třídy `AktivitaFactoryModel`. Tovární třída vyhodnotí obdržená data a na základě těchto dat a implementované logiky výběru zvolí správný objekt, kterému předá informace ke zpracování. K dispozici jsou tři třídy odvozené od společného předka. Třídy `staleHraje` a `nehraje` zpracují data a vrátí správný údaj o aktivitě interpreta. V případě, že se v databázi nachází chybná data, v tabulce ilustrovány zakázanými stavy, zvolí tovární metoda třídu `chybaDat`.

Ta předá textový řetězec s informací, že v databázi jsou chybová data a proto nemůže být aktivita zobrazena.

Zhodnocení

Funkce vzoru Factory je vhodné využít v případě, kdy před začátkem vykonávání programu nevíme, z jakých dat budeme objekt tvořit a co od něj následně budeme očekávat za funkce. V tomto případě je implementace vzoru zvolena spíše demonstračně. Objekty nenabízejí kromě funkce vypsání aktivity nic navíc. Je ale možné, že se v budoucnu bude aplikace rozšiřovat a objekty najdou další využití. Následné doimplementování potřebných funkcí bude usnadněné díky použití vzoru a rozdělení aplikační logiky mezi několik objektů.

7 Závěr

Byla vytvořena webová aplikace – komunitní portál, zaměřený na hudební scénu. Díky možnosti hodnotit hudební nahrávky registrovanými uživateli, by měl portál nabídnout nejen většinové hodnocení v podobě průměrné známky, ale také slovní komentáře od jednotlivých uživatelů. Sami uživatelé si poté mohou udělat úsudek o tom, zda chtějí věnovat svůj čas poslechu pro ně neznámé nahrávky. Mohou tak učinit pouze na základě celkového hodnocení, nebo se rozhodnout podle hodnocení některých uživatelů, kteří mají podobný hudební vkus. Sami pak mohou nahrávku ohodnotit a okomentovat. Navíc, lidé se s časem vyvíjí a s tím se částečně mění i jejich hudební vkus – mnohou proto svůj komentář a hodnocení editovat a připsat například názor z dlouhodobějšího hlediska.

Aplikace byla vytvořena za použití moderních programovacích technologií, jako je OOP a návrhové vzory. Jak je patrné z dílčích zhodnocení v předchozí části práce, využití návrhových vzorů v prostředí webových aplikací je sice snadné, ale je třeba počítat s danými specifiky tohoto prostředí. I přes to, že v posledních verzích nabízí PHP možnosti využívat mnoho moderních technik programování, je třeba si uvědomit, že se jedná stále především o skriptovací jazyk pro tvorbu dynamických webových stránek. Z toho vyplývají jistá omezení. Stejně tak je třeba uvědomit si vlastnosti protokolu HTTP.

V první řadě se jedná o fakt, že HTTP je nestavový a nedokáže tak přenášet mezi jednotlivými částmi aplikace proměnné, objekty a další stavové informace. Toto omezení je možné překonat pomocí nástrojů jazyka PHP, které umožňují práci s cookies nebo session. V této práci byla zvolena metoda předávání proměnných a objektů pomocí session, aby bylo možné co nejsnáze aplikovat uvedené poznatky nezávisle na volbě případného frameworku, nebo i v případě, že žádný framework nepoužijeme.

V případě, že framework bude nabízet jednodušší možnost, jak přenášet objekty mezi jednotlivými požadavky, bude lepší použít ten. Zvolený framework Nette nabízí možnost peristentních parametrů, které jsou automaticky přenášeny, ale jak bylo uvedeno ve zhodnocení kapitoly 6.1, nelze v nich přenášet objekty.

Druhý problém může nastat v případě slabého typování v jazyce PHP. Jak bylo uvedeno především v kapitolách 6.2 a 6.3, je nutné zajistit, aby proměnné, ve kterých se nachází odkazy na objekty, nemohly být volně měněny. V případě, kdy by toto nebylo zajištěno, může nastat situace, kdy bude obsah proměnné přepsán a nejen že proměnná

již nebude obsahovat instanci některého z objektů, ale může dokonce obsahovat jednoduchý datový typ. Tím by byla značně narušena funkčnost aplikace a ve chvíli, kdy bude nutné zavolat metodu příslušného objektu, skončí vykonávání programu chybou.

Jestliže budeme brát zřetel na tyto dvě výrazná specifika, již od začátku návrhu aplikace, mohou být návrhové vzory využity stejně jako u klasických desktopových aplikací. Využití návrhových vzorů ve vhodných situacích s sebou přinese zpřehlednění a lepší čitelnost kódu a případné rozšiřování aplikace v budoucnu bude jednodušší.

Také je vždy při použití návrhových vzorů přihlédnout k faktu, že není nutné se za každou cenu striktně držet předepsanými pravidly. Je tedy potřeba si uvědomit, že při použití jazyka PHP může nastat situace, kdy se dá návrhový vzor naimplementovat funkčním, ale ne zcela správným způsobem, viz kapitola 6.26.5. Tohoto je dobré se vyvarovat a naučit se používat vzory správně. Podobné návyky by mohly mít nepříjemný dopad ve chvíli, kdy bude programátor chtít využít návrhových vzorů v jiném programovacím jazyce, který má silné typování, např. Java nebo C#.

Zároveň je ale třeba si také uvědomit, že se v některých případech dá vzor naimplementovat funkčně a zároveň dobře, ale ne zcela podle původního postupu, který vzor stanovuje, viz kapitola 386.4.

Musíme tedy k návrhovým vzorům přistupovat způsobem, aby nám práci především usnadňovali a nekladli před nás zbytečné překážky.

Literatura

[1] BOUDA, Radek. Seriál návrhových vzorů – 1. díl. Programujte.com [online]. 10. 4. 2012 [cit. 2012-05-17]. Dostupné z: http://programujte.com/clanek/2012032900-serial-navrhovych-vzoru-1-dil/

[2] THE PHP GROUP. PHP: Hypertext Preprocessor [online]. © 2001-2012, Thu May 17 15:21:18 2012 [cit. 2012-05-17]. Dostupné z: http://php.net/

[3] NETTE FOUNDATION. Nette Framework: Rychlý a pohodlný vývoj webových aplikací v PHP [online]. © 2008, 2012, 12. 4. 2012 [cit. 2012-05-17]. Dostupné z: http://nette.org/

[4] Základní principy objektově orientovaného programování. ARI [online]. [cit. 2012-05-17]. Dostupné z: http://ari.wikidot.com/zakladni-principy-objektove-orientovaneho-programovani

[5] Zapouzdření a skrývání implementace. PECINOVSKÝ, Rudolf. Myslíme objektově v jazyku Java: kompletní učebnice pro začátečníky. 2., aktualiz. a rozš. vyd. Praha: Grada, 2009, s. 148. ISBN 978-80-247-2653-3.

[6] TOMAN, Ivo. Poznámky OOP I: základní pojmy. Ivorius weblog [online]. 4.10.08 [cit. 2012-05-17]. Dostupné z: http://ivorius.com/weblog/poznamky-oop-zacatecnika-1

[7] DOSTÁL, Aleš. OOP v PHP 2: základy. Finc weblog [online]. Leden 30th, 2007 [cit. 2012-05-17]. Dostupné z: http://blog.irminsul.cz/2007/01/30/oop-v-php-2-zaklady/

[8] MROZEK, Jakub. OOP v PHP: Základy OOP. ZONER SOFTWARE, a.s. Interval.cz [online]. 15. 02. 2006 [cit. 2012-05-17]. Dostupné z: http://interval.cz/clanky/oop-v-php-zaklady-oop/

[9] GRUDL, David et al. Výchozí Latte makra. NETTE FOUNDATION. Nette Framework [online]. 17. 6. 2011, 4. 2. 2012 [cit. 2012-05-17]. Dostupné z: http://doc.nette.org/cs/default-macros

[10] GRUDL, David et al. Quick Start. NETTE FOUNDATION. Dibi [online]. 28. 5. 2008, 12. 10. 2010 [cit. 2012-05-17]. Dostupné z: http://dibiphp.com/cs/quick-start

[11] GRUDL, David. NETTE FOUNDATION. Dibi: Dibi is Database Abstraction Library for PHP 5. [online]. © 2008, 2012, 19. 1. 2012 [cit. 2012-05-17]. Dostupné z: http://dibiphp.com/cs/

[12] HERRINGTON, Jack D. Five common PHP design patterns. IBM. DeveloperWorks [online]. 18 Jul 2006 [cit. 2012-05-17]. Dostupné z: http://www.ibm.com/developerworks/library/os-php-designptrns

[13] PANDA et al. Vytvoření presenteru. NETTE FOUNDATION. Nette Framework [online]. 23. 12. 2011, 26. 2. 2012 [cit. 2012-05-17]. Dostupné z: http://doc.nette.org/cs/quickstart/presenter

[14] GRUDL, David et al. MVC aplikace & presentery. NETTE FOUNDATION. Nette Framework [online]. 17. 6. 2011, 4. 2. 2012 [cit. 2012-05-17]. Dostupné z: http://doc.nette.org/cs/presenters

[15] BERNARD, Borek. Prezentační vzory z rodiny MVC. Zdroják [online]. 11. 5. 2009 [cit. 2012-05-17]. Dostupné z: http://www.zdrojak.cz/clanky/prezentacni-vzory-zrodiny-mvc/

[16] GRUDL, David. Nette Framework: MVC & MVP. Zdroják [online]. 24. 3. 2009 [cit. 2012-05-17]. Dostupné z: http://www.zdrojak.cz/clanky/nette-framework-mvc--mvp/

[17] Strategy. GAMMA, Erich, Richard HELM, Ralph JOHNSON a John VLISSIDES. Design Patterns: Elements of reusable object-oriented software [online]. Boston: Addison-Wesley, 1995, s. 266 [cit. 2012-05-17]. ISBN 0-201-63361-2. Dostupné z: http://www.saeedsh.com/resources/Design%20Patterns.pdf

[18] State. GAMMA, Erich, Richard HELM, Ralph JOHNSON a John VLISSIDES. Design Patterns: Elements of reusable object-oriented software [online]. Boston: Addison-Wesley, 1995, s. 258-261 [cit. 2012-05-17]. ISBN 0-201-63361-2. Dostupné z: http://www.saeedsh.com/resources/Design%20Patterns.pdf

[19] Příliš mnoho rozhodování: (Stav - State). PECINOVSKÝ, Rudolf. Návrhové vzory. Brno: Computer Press, 2007, s. 224. ISBN 978-80-251-1582-4.

[20] Singleton. GAMMA, Erich, Richard HELM, Ralph JOHNSON a John VLISSIDES. Design Patterns: Elements of reusable object-oriented software [online]. Boston: Addison-Wesley, 1995, s. 109 [cit. 2012-05-17]. ISBN 0-201-63361-2. Dostupné z: http://www.saeedsh.com/resources/Design%20Patterns.pdf

[21] Patterns. THE PHP GROUP. PHP [online]. © 2001-2012, 11 May 2012 [cit. 2012-05-17]. Dostupné z: http://www.php.net/manual/en/language.oop5.patterns.php

[22] DVOŘÁK, Miloš. Factory Pattern. Objekty [online]. 12.06.2005 [cit. 2012-05-17]. Dostupné z: http://objekty.vse.cz/Objekty/Vzory-Factory

Příloha A

Obsah přiloženého CD

Adresář BP2012 – Obsahuje vlastní text bakalářské práce ve formátu pdf

Adresář BPAplikace – Obsahuje zdrojové kódy webové aplikace