VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF CONTROL AND INSTRUMENTATION

VÝVOJOVÉ PROSTŘEDÍ PRO GENEROVÁNÍ CLIENT/SERVER DATABÁZOVÝCH APLIKACÍ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

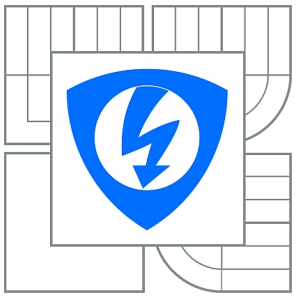
AUTOR PRÁCE
AUTHOR

Bc. MILAN LEMPERA

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF CONTROL AND INSTRUMENTATION

VÝVOJOVÉ PROSTŘEDÍ PRO GENEROVÁNÍ CLIENT/SERVER DATABÁZOVÝCH APLIKACÍ

DEVELOPMENT ENVIRONMENT (IDE) FOR DEVELOPING CLIENT/SERVER APPLICATIONS.

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

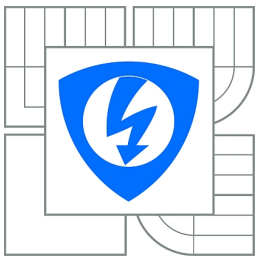
AUTOR PRÁCE — Bc. MILAN LEMPERA

AUTHOR

VEDOUCÍ PRÁCE — Ing. RADOVAN HOLEK, CSc.

SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav automatizace a měřicí techniky

Diplomová práce

magisterský navazující studijní obor
Kybernetika, automatizace a měření

Student:	Bc. Milan Lempera	***ID:***	73014
Ročník:	2	***Akademický rok:***	2010/2011

NÁZEV TÉMATU:

Vývojové prostředí pro generování client/server databázových aplikací

POKYNY PRO VYPRACOVÁNÍ:

1. Navrhněte základní architekturu vývojového prostředí. Předpokládá se, že vývojové prostředí i cílové aplikace budou provozované v prostředí tenkého klienta.
2. Zvolte vhodný framework, šablonovací systém a dostupné knihovny pro práci s databází.
3. Navrhněte datový a procesní model pro implementaci administračního prostředí.
4. Realizujte základní část vývojového prostředí.
5 Na vzorovém projektu ověřte dosažené výsledky při generování cílové aplikace.

DOPORUČENÁ LITERATURA:

Maslakowski M., Naučte se MySQL za 21 dní. Praha: Computer Press, 2001. 478 s. ISBN 80-72226-448-6.
Brázda J., PHP 5 začínáme programovat. Praha: Grada Publishing a.s., 2006. 244 s. ISBN 80-247-1146-X.
Dle pokynů vedoucího práce a vlastního výběru.

Termín zadání:	7.2.2011	***Termín odevzdání:***	23.5.2011

Vedoucí práce: Ing. Radovan Holek, CSc.

prof. Ing. Pavel Jura, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Diplomová práce se zabývá návrhem vývojového prostředí pro generování client/server databázových aplikací. Definuje architekturu a vlastnosti správné databázové aplikace a navrhuje vhodné koncepty pro unifikaci systémových částí aplikace. Dále analyzuje vlastnosti vývojového prostředí, které usnadní implementaci databázových aplikací. Je navržen datový model, implementován prototyp vývojového prostředí a ověřena jeho funkčnost a správnost navržených konceptů.

KLÍČOVÁ SLOVA

vývojové prostředí, generování zdrojového kódu, databázové aplikace client/server, Nette framework, PHP, MySQL

ABSTRACT

This thesis deals with the development environment for generating client/server database applications. The thesis defines the characteristics of good architecture of database applications and proposes appropriate concepts for the unification of system parts of the application. It also analyzes the characteristics of the development environment to facilitate the implementation of database applications. In thesis is proposed data model. There is implemented a prototype of development enviroment and verified its functionality and correctness of the proposed concepts.

KEYWORDS

development environment, source code generation, database client/server applications, Nette framework, PHP, MySQL

LEMPERA, Milan *Vývojové prostředí pro generování client/server databázových aplikací*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2011. 109 s. Vedoucí práce byl Ing. Radovan Holek, CSc.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Vývojové prostředí pro generování client/server databázových aplikací“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Radovanu Holkovi, CSc., za čas, který mi věnoval, a za ochotu se kterou zodpovídal mé dotazy.
Můj upřímný dík patří mé manželce Ivě a rodičům za jejich podporu a pochopení.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Při vývoji databázových aplikací se často setkáváme s vlastnostmi, které jsou pro řadu z nich stejné, nebo velmi podobné. Vlastnosti jako např. stránkování, filtrování, nebo řazení jsou potřeba ve většině databázových aplikací. Obdobná situace je i u systémové části informačního systému, kterou uživatel při své práci mnohdy ani nevnímá, přesto v každém kvalitním informačním systému nesmí chybět. Jedná se např. o validaci dat, kontrolu životních cyklů entit, nebo přístupových oprávnění.

Při vývoji informačního systému často programátor věnuje určitou část své práce opakujícím se činnostem, které by bylo možné, při správném návrhu systémové části aplikace zjednodušit, nebo automatizovat. Bylo by tedy výhodné, vytvořit prostředí, které umožní zvolit a nakonfigurovat požadované vlastnosti aplikace a pak aplikaci podle této konfigurace vygenerovat. Vzhledem k tomu, že některá vstupní data lze zjistit automaticky (např. informace o datovém modelu z ER diagramů, nebo přímo z databáze), mohlo by vývojové prostředí vykazovat jistou míru autonomnosti a inteligence.

Tato práce se zabývá návrhem vývojového prostředí, které by umožňovalo co nejvíce automatizovat, zjednodušit a unifikovat implementaci informačního systému v prostředí tenkého klienta.

1 VLASTNOSTI INFORMAČNÍHO SYSTÉMU

Tato kapitola uvádí výčet vlastností, které jsou v informačních systémech typické a často se opakují. Jejich jednotné řešení by přineslo jednodušší a čistější kód a v neposlední řadě také úsporu času a nákladů. Kapitola je rozdělena do tří částí.

1.1 Architektura aplikace

Aby byla aplikace co nejlépe udržovatelná, měla by být zvolena vhodná architektura, která bude definovat oddělení prezentační logiky od logiky aplikační. Vhodnou se jeví architektura MVC, nebo některá její varianta.

1.1.1 Použití frameworku a volně dostupných knihoven

Ve fázi návrhu by mělo být zváženo využití některého frameworku jak pro generované aplikace, tak pro samotné vývojové prostředí. Bylo by vhodné porovnat několik volně dostupných řešení. Také by bylo vhodné zvážit použití dalších volně dostupných knihoven.

1.2 Systémová část informačního systému

1.2.1 Podpora jazykových mutací

Podpora jazykových mutací je dnes standardním požadavkem na téměř každý informační systém. Proto by měly být navrženy principy pro jednotnou a pohodlnou správu vícejazyčných údajů. Uložení a přístup k jazykovým datům by měl být zvolen tak, aby bylo možné přidávat další jazykové mutace bez zásahů do systému.

1.2.2 Přístupová oprávnění

Jedním z nejdůležitějších prvků systémové části aplikace je modul řídící přístupová oprávnění. V jednodušších systémech bývají oprávnění svázána přímo s konkrétním uživatelem, nebo mají hierarchickou strukturu. Tyto přístupy jsou však značně omezující.

Jako nejflexibilnější se tedy jeví využití modelu na principu rolí, které opravňují k přístupu k jednotlivým zdrojům. Uživatel může mít přidělen libovolný počet rolí. Informace o přidělení rolí jsou uloženy v databázi, takže je možné uživatelská oprávnění pohodlně spravovat bez zásahů do systému.

V rámci návrhu vývojového prostředí bude zváženo využití tohoto modelu oprávnění na přístup k modelové části aplikace, kde je navíc třeba řešit další omezení, jako je např. stav záznamu, vlastnictví záznamu, a další.

1.2.3 Životní cykly entit

Další důležitou vlastností, kterou by měl informační systém poskytovat, je řízení životních cyklů entit. Jednotlivé entity mají definované stavy, kterými ve svém životním cyklu procházejí. V každém stavu mohou s entitou pracovat pouze některé role. Výčet akcí, které mohou provádět, je kontrolován, stejně jako možnosti editace jednotlivých atributů entity.

Přechody mezi stavy jsou taktéž povoleny jenom pro konkrétní role. Před samotným přechodem je třeba zkontrolovat, zda je entita validní a vykonat přechodové kontroly, bez jejichž úspěchu nemůže být přechod proveden.

1.2.4 Variantní položky

V některých informačních systémech se setkáváme s požadavkem na variantní položky. Jde o položky, které existují ve více variantách a vhodná varianta je vybrána dle určité podmínky. Podmínky mohou být buď časové (platnost varianty je od – do), nebo určené logickým výrazem (uživatel patří do skupiny, nebo hodnota atributu entity se rovná konkrétní hodnotě,...).

Při návrhu systémové části by měla být podpora variant zvážena. Její využití by přicházelo v úvahu u formulářů a sestav, u číselníků a také u šablon.

1.3 Uživatelská část

1.3.1 Navigace

Menu systém byl měl spolupracovat se systémem oprávnění a zobrazovat uživateli vždy pouze ty položky, k nimž má přístup. Měl by být vhodně rozdělen do několika úrovní a graficky reprezentován tak, aby bylo zřejmé, kde se uživatel právě nachází a jaké operace může provádět.

1.3.2 Uživatelská nastavení

Současné informační systémy nabízejí uživateli řadu uživatelských nastavení při práci s daty – filtry, řazení podle více parametrů apod. Většinou však tyto nastavení neukládají, případně je udržují pouze po dobu přihlášení uživatele v systému.

Užitečné by tedy bylo definovat službu, která by se o ukládání těchto nastavení starala. Jednotlivé komponenty by nemusely řešit ukládání nastavení, pouze by využily metody poskytované touto službou.

1.3.3 Jednotné prvky

Jak již bylo řečeno v úvodu i v uživatelské části systému jsou prvky, které se opakují napříč různými systémy. Kromě již zmíněných filtrů a stránkování se jedná např. o LOV (List Of Value – formulářový prvek, který umožňuje vybrat hodnotu ze seznamu) prvky, které umožňují výběr z více hodnot a které jsou často řešeny pouze formulářovými prvky `select`. Toto řešení však selhává v případě většího množství položek. Výběr je potom značně nesnadný. Použití LOV by přineslo řadu výhod od lepší ovladatelnosti až po možnost řazení a filtrování v datech.

1.3.4 Zobrazení formulářových prvků dle oprávnění

V některých situacích by bylo užitečné, vytvořit jeden formulář, který by různým uživatelům umožnil vidět a editovat pouze ty prvky, ke kterým má příslušná oprávnění. Potom by nebylo nutné vytvářet několik verzí formulářů pro různé role/uživatele, což by vedlo ke zjednodušení případných úprav systému.

1.3.5 Nezávislost vzhledu na obsahu

Oddělení prezentační a aplikační logiky by mělo být dodržováno v celém systému. Proto by i formuláře měly být navrženy tak, aby bylo možno měnit jejich vzhled, tedy pozice jednotlivých prvků a styl vykreslení bez zásahu do jejich logiky.

2 VLASTNOSTI VÝVOJOVÉHO PROSTŘEDÍ

Tato kapitola popisuje vlastnosti a funkce vývojového prostředí. Nejprve jsou popsány obecné vlastnosti, dále vlastnosti související s datovým modelem. V další kapitole jsou rozebrány vlastnosti pro řízení přístupu a životních cyklů. Dále je definováno rozdělení prezentační části aplikace a v poslední kapitole očekávaný výstup generátoru.

2.1 Obecné vlastnosti

2.1.1 Správa projektů

Vývojové prostředí by mělo umožnit spravovat více projektů, ukládat nastavení a metadata jednotlivých projektů do databáze. Mělo by zahrnovat jednoduchou správu projektů s možností přenesení nastavení mezi jednotlivými projekty (např. číselníky), zálohování a obnovení jednotlivých projektů i kompletní databáze projektů.

2.1.2 Jazykové mutace

Podpora jazykových mutací by měla být součástí vývojového prostředí. U každého projektu by měla být možnost zvolit povolené jazyky a výchozí jazyk.

2.1.3 Nezávislost na konkrétním databázovém systému

Vývojové prostředí by mělo být navrženo tak, aby umožňovalo tvorbu databázových aplikací nezávislých na konkrétní databázi. Podpora pro konkrétní databáze by měla být modulární, aby bylo možné jednoduše doplnit podporu jiných databází. Stejně tak struktura pro ukládání metadat vývojového prostředí by měla být navržena tak, aby nebyla závislá na implementaci generátoru v konkrétním jazyce. Tím by měla být zachována možnost použití vývojového prostředí ke generování aplikací v různých programovacích jazycích.

2.1.4 Varianty

Prostředí by mělo obsahovat i podporu pro variantní položky. Jde o položky, které existují ve více variantách a vhodná varianta je vybrána dle určité podmínky. Podmínky mohou být buď časové (platnost varianty je od – do), nebo určené logickým výrazem (uživatel vlastní roli, uživatel patří do skupiny, nebo se hodnota atributu

záznamu rovná konkrétní hodnotě, ...). V systému by variantnost měla být implementována u číselníků a šablon.

2.2 Vlastnosti související s datovým modelem

2.2.1 Automatické načítání datového modelu

Metadata nesoucí informace o struktuře databázových tabulek, atributech, jejich datových typech, indexech, primárních i cizích klíčích a vztazích mezi atributy je možné načíst přímo z databáze přiřazené ke konkrétnímu projektu, nebo ze zakládacího SQL kódu databáze. Dále by měla být implementována možnost editace načtených metadat.

2.2.2 Využití SQL komentářů

Text uvedený v komentáři databázových tabulek bude taktéž prostředím ukládán a bude použit jako nápověda u formulářů obsahujících tento atribut. Pokud nebude zvoleno jinak, budou komentáře uloženy jako nápovědy pro výchozí jazykovou verzi.

2.2.3 Slovník názvů atributů

Při načítání metadat databázového modelu bude využíván slovník názvů atributů. Jde o jazykově závislou strukturu, která nese údaj o názvu atributu entity v databázovém modelu a názvu, kterým byl nahrazen v konkrétním jazyce. Slovník bude zahrnovat informace ze všech projektů, bude preferovat nejpoužívanější název, v editaci pak nabídne možnost výběru z dříve použitých možností.

2.2.4 Vypočítaná pole

V metadatech by měly být také informace o vypočítaných polích, které budou dále zobrazovány ve formulářích a výstupních sestavách. Vypočítaná pole bude možné zadat k jednotlivým entitám jako SQL kód, nebo vztah pro jejich výpočet. Při tvorbě formuláře pak uživatel vybere vypočítané pole, které se má zobrazit a to pak bude zařazeno do SQL dotazu pro získání dat, aby byly do dotazu zařazeny pouze aktuálně potřebná pole.

2.2.5 Výchozí sloupec

Libovolný atribut entity bude možné označit jako výchozí. Takto označený sloupec bude automaticky použit jako náhrada cizího klíče při zobrazení dat entity a při její

editaci ve formuláři typu LOV.

2.2.6 Speciální typy atributů

Atribut může obsahovat data, která vyžadují speciální přístup. Je možné označit speciální typ atributu. Atribut je potom kontrolován regulárním výrazem, který k danému typu přísluší. Mohlo by se jednat například o následující typy:

- Hašovací funkce MD5, SHA1,
- HTML kód,
- HTML barva,
- URL adresa,
- emailová adresa,
- telefonní číslo,
- PSČ,
- IP adresa,
- regulární výraz.

Implementace pomocí regulárních výrazů by měla umožnit snadné rozšíření o další typy.

2.2.7 Dědičnost tříd

Entity mohou dědit funkce od předem vytvořených vzorových tříd. Mohlo by se jednat například o tyto funkce:

- manipulace se stromovou architekturou (metodou traverzování kolem stromu),
- manipulace s atributem pořadí pro řazení záznamů,
- funkce pro verzování dat v tabulce.

Mělo by být umožněno jednoduché vytváření dalších vzorových tříd.

2.2.8 Omezení vztahů mezi entitami

V rámci metadat datového modelu je možné zadat omezení pro vztahy mezi entitami. A to pomocí min/max účasti master tabulky, nebo min/max účasti slave tabulky.

2.2.9 Referenční integrita

V rámci udržení referenční integrity je možno nastavit několik typů chování při mazání záznamu s cizím klíčem:

- nastavit na `NULL`,
- nastavit výchozí hodnotu,
- kaskádové mazání,

- zamezit smazání,
- popřít (deaktivovat záznam),
- vykonat SQL funkci.

Při editaci záznamu nebude možno změnit primární klíč, proto není potřeba editaci tímto způsobem ošetřovat.

2.3 Vlastnosti pro řízení přístupu a životních cyklů

Aplikace vygenerovaná navrhovaným vývojovým prostředím bude k řízení přístupu používat model rolí, kde každý uživatel může vlastnit libovolný počet rolí. Vlastněné role pak uživatele opravňují k přístupu k jednotlivým modulům, formulářům a sestavám. Role budou využity pro řízení přístupu k entitám v konkrétních stavech životního cyklu. Proto je vhodné zadat všechny role ještě před vytvořením životního cyklu entit. Při vytváření životního cyklu je pak možné přímo definovat přístup pro konkrétní role.

2.3.1 Zadání rolí

Ve vývojovém prostředí budou předpřipraveny základní uživatelské role např. role Nepřihlášený uživatel a Přihlášený uživatel. Ty bude možno deaktivovat. Bude možno přidávat další role.

2.3.2 Zadání životních cyklů entit

Zadání životních cyklů lze rozdělit do tří kroků:

Označení entit dle životního cyklu nejprve je třeba rozlišit entity, které nemají žádný životní cyklus. Vývojové prostředí takto automatiky označí ty entity, u kterých nenajde atribut nesoucí informaci o stavu. Uživateli bude umožněno přidat entitu do skupiny s životním cyklem, nebo je z ní naopak odebrat. Stejně tak lze změnit atribut nesoucí stavovou informaci.

Entity s jednoduchým životním cyklem entity, které mají životní cyklus budou dále rozděleny podle toho, zda se jedná pouze o jednoduchý životní cyklus se stavy 0 a 1, nebo jde o složitější životní cyklus. Stavy 0/1 jsou vygenerovány automaticky s popiskem Aktivní/Neaktivní.

Entity se složitějším životním cyklem specifický životní algoritmus musí uživatel do prostředí zadat ručně.

2.3.3 Stavově závislá validační pravidla

Na životním cyklu entity mohou záviset validační pravidla pro konkrétní atributy, nebo jejich kombinace. Ty bude možné v konkrétním stavu zapnout do režimů chyba, varování, nebo vypnout.

2.3.4 Přechodové kontroly

Se životními cykly souvisí také kontroly, které by měly být v systému taktéž implementovány. Jedná se o SQL funkce, které jsou spouštěny při některé z následujících událostí:

- přechod ze stavu A do B,
- vstup do stavu,
- opuštění stavu,
- změna dat ve stavu.

Kontrolní funkce jsou určeny pro kontrolu podmínek přechodu. Pokud jsou podmínky splněny, vrátí funkce hodnotu `TRUE` a přechod může být uskutečněn. Pokud podmínky splněny nejsou, vrací funkce hodnotu `FALSE`. Přechodové kontroly nikdy nemění data, k tomuto účelu slouží přechodové funkce. V tomto případě by bylo vhodné, aby byly vykonány vždy všechny přechodové kontroly a ve výsledném chybovém hlášení byly vypsány všechny nedostatky, kvůli kterým k přechodu nedošlo. Dále by mohla být přínosná možnost spustit některé kontroly pouze v režimu varování. Uživatel by musel potvrdit, že bere varování na vědomí a poté by mohl být přechod uskutečněn i při nesplnění přechodové kontroly.

2.3.5 Přechodové funkce

Přechodové funkce mohou být spouštěny při stejných událostech. Mohou měnit data a jejich návratová hodnota bude vyhodnocována volitelně – buď jako chyba, jako varování. Opět se jedná o SQL funkce (případně procedury).

2.4 Moduly a formuláře

V předchozích částech byly popsány vlastnosti, které zaručují správu datového modelu, kontrolu hodnot atributů, životního cyklu. Nyní bude popsána ta část systému, která slouží k vytváření modulů, formulářů a sestav.

2.4.1 Moduly

Moduly jsou základním blokem pro správu formulářů a sestav. Moduly mohou obsahovat další podřízené moduly, formuláře, nebo sestavy.

2.4.2 Formuláře

Formuláře umožňují pracovat s daty. Bude použito pět základních typů formulářů: Query Form (QF), Record List (RL), Insert Form (IF), View Form (VF), List Of Value (LOV). Vedle vytváření formulářů by měla být implementována i možnost správy šablon pro formuláře, jejich editace, přidávání a klonování, stejně jako klonování formulářů.

2.4.3 Sestavy

Jsou určeny pro výstup dat z generované aplikace. V podstatě je možné je rozdělit na tiskové sestavy a exportní sestavy. V tiskových sestavách by měla být možnost výběru mezi tiskem na tiskárnu a do PDF souboru. U exportních sestav je možné vybrat z formátů XML, CSV a HTML. V rámci modulů bude dále zvážena možnost importu dat a její zařazení do formulářů, nebo do sestav.

2.4.4 Přístupová oprávnění k modulům a formulářům

Vývojové prostředí by mělo umožnit řízení přístupu na několika úrovních:

- přístup k modulům,
- přístup k jednotlivým formulářům a sestavám v modulu,
- přístup k jednotlivým entitám,
- přístup k jednotlivým atributům entit ve formulářích a sestavách.

Poslední dva případy by měly dále zohledňovat životní cyklus entit, ke kterým je přistupováno.

2.5 Generátor

Výstupem z vývojového prostředí bude generovaný kód databázové aplikace. Generovaný kód by měl být rozdělen do několika částí:

- systémová část:
  - třídy, podporující systémové vlastnosti, případně framework,
- generované zdrojové kódy:
  - modelové části aplikace,
  - prezentační části aplikace,

- SQL kód:
  - s definicí systémových tabulek,
  - se systémovými daty.

Součástí by mohl být také instalační soubor, který zajistí vykonání SQL kódu na serveru.

3 PROSTŘEDKY PRO IMPLEMENTACI

Prostředí client/server pro generované aplikace je dané zadáním. U vývojového prostředí bylo s ohledem na budoucí využití zvoleno taktéž prostředí client/server. Tato volba umožní používání principů navržených pro generované aplikace přímo ve vývojovém prostředí, což přinese zjednodušení vývoje a zároveň umožní ověření navržených principů v praxi.

V této kapitole budou popsány vybrané technologie, volba frameworku a šablonovacího systému. Poslední kapitola uvádí přehled volně dostupných knihoven, které by mohly být při implementaci využity.

3.1 Zvolené technologie

Pro implementaci byly vybrány technologie v současné době rozšířené a hojně používané. Webový server Apache, skriptovací jazyk PHP a databázový systém MySQL (volba databázového systému se vztahuje k vývojovému prostředí, generované aplikace by měly být navrženy nezávisle na zvolené databázi). Vybrané technologie budou dále krátce představeny.

3.1.1 Apache

V současné době nejrozšířenější webový server. Je dostupný na všechny hlavní platformy (Windows, Linux, ...) a je vyvíjen jako open source. Podporuje protokol HTTP/1.1. V kombinaci s PHP a MySQL je nejčastěji využívaným řešením pro tvorbu dynamických stránek. Jeho oblibu lze přičíst také rozšiřitelnosti díky velkému množství modulů a nastavení. Více v [1].

3.1.2 MySQL

Je relační databázový systém, který jeho autor, Michael Widenius, vyvinul nejprve pro vlastní potřeby při tvorbě webových aplikací, později jej uvolnil na internetu. V současné době MySQL vlastní firma Oracle a je poskytován pod GNU GPL licencí. MySQL je možné provozovat téměř na všech systémech od různých linuxových distribucí až po komerční systémy jako např. Mac OS X, Windows a další. Také díky tomu patří k nejrozšířenějším databázovým technologiím.

Jak název napovídá, komunikace probíhá pomocí jazyka SQL, respektive jeho mírné modifikace s některými rozšířeními. MySQL je optimalizovaný pro rychlost, je tedy jeden z nejrychlejších databázových systémů, ale za cenu určitých zjednodušení.

Některé pokročilejší funkce (např. podpora pohledů, triggerů a uložených procedur), byly přidány až v posledních verzích . Další informace v [4].

3.1.3 PHP

PHP je jedním z nejpoužívanějších jazyků pro tvorbu dynamicky generovaných WWW stránek. Autor Rasmus Lerdorf jej vytvořil pro svou osobní potřebu a publikoval pod názvem „Personal Home Page Tools“, zkráceně PHP. Později se ujal název „PHP: Hypertext Preprocessor“.

Tento jazyk se stal populární zejména díky možnosti integrovat ho přímo do HTML stránky. Lze tak snadno vytvářet webové aplikace. V jednom zdrojovém souboru se kombinuje statická část stránky spolu s výkonným kódem. PHP obsahuje rozsáhlé knihovny funkcí pro zpracování textu, grafiky, práci se soubory, přístup k většině databázových serverů a podporu celé řady internetových protokolů. Ve verzi 5 byl navíc kompletně přepracován objektový model, takže styl objektového programování se blíží k Javě nebo C++ [5]. Verze 5.3 navíc přidává další koncepty známe z uvedených jazyků, např. jmenné prostory a anonymní funkce. [7]

3.2 Volba Frameworku

Tato kapitola zdůvodňuje potřebu frameworku pro aplikaci. Jsou v ní stanoveny vlastnosti, které by měl vhodný framework mít a dále jsou porovnány vybrané frameworky. V poslední části je pak zdůvodněna volba konkrétního frameworku.

3.2.1 Proč použít framework

Frameworky jsou knihovny, které mají ulehčit práci při programování. Cílem frameworku je odstínit programátora od typických problémů určité oblasti, aby se mohl lépe soustředit na účel aplikace, místo konkrétních implementačních problémů. Při návrhu vývojového prostředí bylo třeba rozhodnout, zde bude aplikace postavena na framworku, či nikoli. Výhodou je, že frameworky mají dobře zvolenou architekturu (v případě webových frameworku se většinou jedná o MVC architekturu, nebo některou její variantu), využívají návrhových vzorů, jsou navrženy tak, aby mohly být dále rozšiřovány a obecně lze říci, že vedou k lepším návykům i programátora. Kód frameworku je znovupoužitelný, což zrychluje vývoj a zjednodušuje údržbu. Za většími frameworky stojí většinou komunita vývojářů a uživatelů, což přináší výhody hlavně při řešení problémů.

Při generování aplikace vývojovým prostředím tedy existují dvě cesty: generátor vygeneruje kód, který nevyužívá framework a nezávisí na žádných knihovnách.

Tento kód je složitější, rozsáhlejší a v neposlední řadě dochází k opakování stejného kódu na více místech, naproti tomu kód generovaný pro framework nemusí obsahovat základní funkce, protože ty jsou už obsaženy ve frameworku. Je jednodušší, přehlednější. Vykonávání takovéhoto kódu však bude vždy pomalejší, než v prvním případě.

Z těchto dvou přístupů byl zvolen druhý, využívající framework. To by mělo zjednodušit a zpřehlednit kód. Mírné zpomalení vykonávání není pro generované aplikace podstatný.

3.2.2 Požadavky na framework

Pro vývojové prostředí a generované aplikace jsou důležité následující vlastnosti frameworku:

- dobrá architektura – architektura, která rozděluje aplikaci do nezávislých vrstev,
- plně objektový návrh,
- vhodná open source licence, která neomezuje další užití aplikace,
- dobré zabezpečení,
- dobrá podpora nových technologií – zvláště AJAX a HTML 5,
- dobré ladící nástroje,
- aktivní komunita,
- podpora knihoven pro přístup k různým databázím,
- modulárnost,
- rychlost,
- velikost,
- možnosti pluginů a rozšíření.

3.2.3 Porovnání frameworků

Pro porovnání byly vybrány: Nette Framework, Zend Framework a Prado.

Nette Framework

Jedná se o výkonný, komponentově orientovaný framework. Je napsaný v PHP 5. Jeho licence, která vychází z BSD, patří k těm nejvolnějším. V současné době je kolem Nette asi nejaktivnější komunita českých PHP vývojářů. Framework je koncipován jako otevřený, je tedy možné používat jeho části samostatně, nebo společně s jinými otevřenými frameworky. Framework využívá MVP architekturu, má velmi propracovaný šablonovací systém, je však možné jej používat i s jinými šablonovacími systémy. Nette má také velmi propracovanou práci s formuláři a je orientován na

bezpečnost a má velmi dobrou podporu pro AJAX. Framework naopak neřeší modelovou část architektury. Neobsahuje třídy pro práci s databází, ale umožňuje využít knihovnu Dibi (která je od stejného autora jako Nette), nebo ORM knihovny – např. Doctrine 2. Velikost Frameworku je méně než 1MB, existuje i v jedno-souborové verzi s velikostí 0,4 MB a podle testů [9] patří k nejrychlejším PHP frameworkům. K Nette jsou k dispozici různé pluginy a rozšíření. Framework je dostupný na [12].

Zend Framework

Jedná se o objektově orientovaný framework licencovaný pod New BSD licencí. Užívá modulární architekturu a stejně jako Nette je koncipován jako otevřený. Framework obsahuje komponenty pro MVC architekturu, autorizaci, autentifikaci, implementuje různé druhy cache, filtrů, validátorů pro uživatelská data, ... Obsahuje taktéž třídy pro práci s formuláři a třídy pro práci s databází. Velikost Frameworku je řádově v desítkách MB. Robustnost tohoto frameworku se projevuje i v rychlosti – oproti Nette je až 10x pomalejší (více v [9]). I pro Zend Frameworku existují pluginy a rozšíření. Framework je k dispozici na webu [13].

Prado

Prado je taktéž objektově orientovaný framework s BSD licencí. Je založen na komponentovém přístupu a programování událostí a má modulární architekturu. Framework odděluje aplikační a prezentační logiku, má zabudovaný šablonový systém a podporuje AJAX. Obsahuje také podporu pro nejpoužívanější databáze. Podporuje databázový vzor Active-record i kompletní objektové mapování. Podle rychlosti můžeme framework zařadit mez Nette a Zend Framework. Oproti předchozím dvěma není Prado tak rozšířen a jeho dokumentace není tak kvalitní. Framework je k dispozici na [14].

3.2.4 Výběr frameworku

Po prostudování vlastností a dokumentace jednotlivých frameworků jsem se rozhodl využít ve vývojovém prostředí Nette Framework. Hlavní předností Nette je jeho malá velikost, rychlost a mimo jiné také to, že Nette se nesnaží komplexně pokrýt celou oblast tvorby webových stránek. Zaměřuje se na konkrétní oblasti (např. formuláře, architektura MVP, ...), ale zároveň nabízí uživateli velkou volnost při tvorbě ostatních částí aplikace. Díky koncepci otevřeného systému je snadné zahrnout do projektu moduly třetích stran. Nezanedbatelnou výhodou je také aktivní komunita vývojářů.

3.3 Volba šablonovacího systému

Šablonovací systém je vrstva aplikace, která zajišťuje oddělení aplikační a prezentační logiky. Pracuje se šablonami, (nejčastěji HTML dokument se speciálními značkami), které jsou nahrazovány, manipulovány či vyhodnocovány šablonovacím systémem. Výsledkem zpracování je výstupní dokument (nejčastěji webová stránka).

Ve své bakalářské práci [15] jsem využíval vlastní implementaci šablonovacího systému, který umožňoval výpis jednotlivých proměnných (tagy), výpis prvků pole v cyklu (`loop`) a vyhodnocení jednoduchých podmínek (`if - else`). Tyto tři jednoduché typy šablonových proměnných jsou často dostačující, nicméně existují situace, kdy by bylo vhodnější, rozšířit tuto šablonovací logiku o další prvky.

Jednoduchý šablonovací systém neumí efektivně řešit například výpis jedné proměnné v různém formátu – např. Text s HTML formátováním nemůžeme použít jako titulek stránky – je třeba nejprve odebrat HTML značky. Ve výše zmíněném šablonovacím systému je tuto situaci třeba řešit pomocí dvou proměnných, kde jedna obsahuje původní text s HTML a druhá je ošetřena odebráním HTML značek. Toto řešení není samozřejmě ideální, protože logiku zobrazení musíme řešit ve vrstvách s aplikační logikou. Ideální by bylo ošetřit tyto situace v šabloně.

Zmíněný šablonovací systém taktéž neřeší problémy jako je rekurze, ošetření proměnných v jiném kontextu než HTML (např. Javascript). Tyto situace je nutné řešit v aplikační logice, a až poté je předat šablonovacímu systému, což narušuje oddělení aplikační a prezentační logiky.

Pro vývojové prostředí by bylo lepší využití složitějšího šablonovacího systému, který řeší výše popsané nedostatky. Protože Nette Framework, který byl pro vývojové prostředí vybrán, obsahuje vlastní šablonovací systém, budou v následující kapitole zhodnoceny vlastnosti tohoto šablonovacího systému. Na základě jeho vlastností bude rozhodnuto o jeho využití, případně o volbě jiného šablonovacího systému.

3.3.1 Šablonovací systém Nette Frameworku

Nette framework zahrnuje vlastní šablonovací systém, (využitelný i bez frameworku), je snadno rozšířitelný a nabízející pokročilé funkce. Text šablony je upraven filtry. Při této úpravě dochází k expanzi maker a aplikaci helperů. Tyto pojmy budou nyní vysvětleny.

Filtry

Jsou využívány pro předzpracování šablony. Jedná se o funkce, které dostanou jako parametr obsah šablony a vrátí ho v pozměněném stavu. Jako filtr je možno registrovat libovolný callback, nebo anonymní funkci. Základní filtry jsou obsaženy

přímo v distribuci frameworku např. Latte filtr, filtr pro převod generování URL adres nebo doplnění absolutních adres.

Makra

Jsou pravidla pro výpis jednotlivých prvků šablon. V tabulce 3.1 jsou uvedena základní makra Latte filtru. Další makra a detailní popis naleznete v [16].

Tab. 3.1: Základní makra Latte filtru.

Zápis v Latte	**PHP ekvivalent nebo význam**
`{$variable}`	Vypíše kontextově escapovanou proměnnou.
`{!$variable}`	Vypíše proměnnou bez escapování.
`{*text komentáře*}`	Komentář, bude odstraněn
`{plink ...}`	Vypíše kontextově escapovaný odkaz.
`{link ...}`	Odkaz nad komponentou
`{if ?} ...`	`<?php if (?): ?> ...`
`{elseif ?} ...`	`<?php elseif (?): ?> ...`
`{/if}`	`<?php endif ?>`
`{foreach ?} ...`	`<?php foreach (?): ?> ...`
`{/foreach}`	`<?php endforeach ?>`
`{for ?} ...`	`<?php for (?): ?> ...`
`{/for}`	`<?php endfor ?>`
`{while ?} ...`	`<?php while (?): ?> ...`
`{/while}`	`<?php endwhile ?>`
`{include dir/file.phtml}`	Vloží podšablonu

Helpery

Jedná se o pomocné funkce, umožňující úpravu proměnných v šabloně. Standardní helpery obsažené v Nette zvládají např. převod velkých a malých písmen, zkrácení textu, odstranění HTML entit, formátovaný výpis data, nebo čísla. Je možné rozšíření vlastními helpery. Stejně jako u filtrů se může jednat o callback, nebo anonymní funkci, jen s tím rozdílem, že jako parametr dostává proměnnou, kterou vrátí v pozměněném stavu. Před použitím je třeba helpery v šabloně registrovat. S použitím Latte filtru je zápis helperů velice jednoduchý, jak je vidět z následující ukázky.

```
{$title|stripTags|upper}
```

Jsou zde použity dva helpery, `stripTags` a `upper`. Tato šablona vypíše proměnnou `$title`, zbavenou HTML entit a převedenou na velká písmena.

Kontextově-senzitivní escapování

Jedná se o další vlastnost Latte filtru. Rozlišují se tyto kontexty:

- HTML text,
- Hodnota HTML atributu (uzavřená do uvozovek nebo apostrofů),
- Značky `<style>` a `<script>`, s
- HTML atributy `style` a atributy akcí začínající písmeny `on-`,
- HTML komentáře.

Escapování probíhá automaticky, výstup bude tedy vždy ošetřen dle příslušných pravidel. Pokud chceme vypsat neošetřenou proměnnou (např. HTML kód), uvedeme před název proměnné vykřičník. Latte filtr podporuje také dědičnost v šablonách a to pomocí mechanizmu bloků. Díky tomu je možné upravit blok nadřazené šablony, který jinak podřazená šablona ovlivnit nemůže.

Zpracování šablony

Jakmile se šablona vykresluje, aplikují se na ní všechny zaregistrované filtry. Šablona se přeloží do čistého php souboru a ten se poté kešuje – to umožní odstínit php od html a v šablonách používat pouze syntax šablonových filtrů. Pro opakované vykreslení se používá kešovaná verze, což zvyšuje rychlost šablonovacího systému.

3.3.2 Zhodnocení šablonovacího systému Nette frameworku

Z přehledu vlastností šablonovacího systému je zřejmé, že odstraňuje všechny nedostatky jednoduchého šablonovacího systému popsaného na začátku této kapitoly. Díky vhodnému návrhu je možné šablonovací systém modifikovat a rozšiřovat. Jeho velkou výhodou je také důraz kladený na bezpečnost. Pro navrhované vývojové prostředí se tento šablonovací systém jeví jako velmi vhodný, bude tedy při jeho implementaci použit.

3.4 Volně dostupné knihovny

V prostředí internetu se v současné době rozvijí poměrně velké množství projektů, které jsou uvolňovány jako open-source a jsou určeny pro použití v dalších projektech. Některé z těchto knihoven by bylo možné využít při implementaci vývojového prostředí. V této kapitole bude popsáno několik knihoven, jejichž použití by bylo přínosem.

3.4.1 jQuery

jQuery je JavaScriptový framework [17] který usnadňuje programování v JS. Umožňuje jednoduchou manipulaci s DOM, pohodlnou obsluhu událostí, komplexní selektory, manipulaci s CSS a jednoduchou obsluhu AJAXu. Framework obsahuje také základní animace prvků stránky, jeho hlavní výhodou je snadná rozšířitelnost a s ní související dostupnost pluginů, které framework dále rozšiřují. JavaScriptových knihoven a frameworků existuje větší množství, jQuery byla vybrána pro jednoduchost použití, její velké rozšíření a rozšířitelnost. V práci by mohly být využity pro zpříjemnění a zpřehlednění uživatelského rozhraní a pro zpracování AJAXových požadavků. Například by mohly být využity pluginy pro automatické dokončování, validaci na straně klienta, nebo úpravu obrázků.

3.4.2 Datagrid

DataGrid [18] je komponenta, pro Nette Framework, která výrazně zjednodušuje tvorbu přehledových tabulek a zajišťuje vizuální prezentaci těchto dat uživateli. Umožňuje data rychle třídit, filtrovat a manipulovat s nimi. Podporuje také AJAX. V administračním prostředí by bylo možné tuto komponentu využít pro tvorbu formuláře RL, případně pro zobrazení přehledu záznamů.

3.4.3 Knihovny pro přístup k databázi

Dále uvedené knihovny slouží k přístupu do databáze, před samotnou implementací bude nutné zhodnotit, která z uvedených knihoven bude nejvhodnější, tak, aby bylo možno splnit požadavky kladené na aplikaci co nejefektivněji. Zde je uveden pouze jejich přehled.

Dibi

Dibi [20] je PHP knihovna pro práci s databázemi, zapouzdřuje funkce pro práci s databází do objektového rozhraní. Automaticky konvertuje datové typy pro databázi a obsahuje rozhraní pro zpracování výsledků dotazu a generování výjimek v případě chyb. Umožňuje zápis pomocí fluent syntaxe, dále obsahuje třídu DibiDataSource, která zapouzdřuje SQL dotaz i vrácený výsledek a vykoná jej až ve chvíli, kdy jsou data skutečně potřeba. Pro Dibi existují ORM nadstavby, nejsou však příliš rozšířené.

NotORM

NotORM [21] je PHP knihovna určená pro jednoduchou práci s daty v databázi. Velmi zajímavou vlastností je podpora snadné práce s relacemi. Knihovna je velice rychlá, autor uvádí, že NotORM může skutečně běžet rychleji než nativní ovladač.

Doctrine 2

Doctrine [22] 2 je nový ORM framework pro jazyk PHP, velmi silně inspirovaný Java knihovnou Hibernate. Knihovna postavená na návrhovém vzoru Data Mapper, je vůči zbytku aplikace reprezentovaná zejména fasádou v podobě Entity Manageru. Pro definici vlastností jednotlivých entit se elegantně využívají komentářové anotace, je zde definován DQL (Doctrine Query Language), kde se místo názvů tabulek a atributů používají výhradně názvy entit a jejich členských proměnných, k dispozici je cachování na několika úrovních, lazy-loading.

4 ANALÝZA A NÁVRH ŘEŠENÍ SYSTÉMOVÉ ČÁSTI APLIKACE

Při analýze byla nejprve definována architektura a struktura aplikace. Poté byly pečlivě prozkoumány požadavky stanovené v kapitole 1. Výstupem analýzy bylo několik konceptů, které umožňují uvedené požadavky splnit. Jednotlivé koncepty budou nyní podrobně vysvětleny.

4.1 Architektura aplikace

Architektura aplikace je do značné míry dána zvoleným frameworkem, který odpovídá vzoru MVP [10], ten bude stručně popsán v následující kapitole. Framework dále určuje adresářovou strukturu a do jisté míry také rozdělení tříd do jmenných prostorů.

4.1.1 MVP

Architektura MVP je velmi podobná architektuře MVC, proto nejprve pár slov o této architektuře.

MVC (Model-View-Controller) rozděluje aplikaci do tří vrstev: na datový model, uživatelské rozhraní a řídicí logiku. Přičemž modifikace některé z nich má pouze minimální vliv na ostatní. MVC určuje vztah jednotlivých komponent:

- **Model** zajišťuje přístup k datům a manipulaci s nimi,
- **View** (pohled) převádí data reprezentovaná modelem do podoby vhodné k prezentaci uživateli,
- **Controller** (řadič) reaguje na události pocházející od uživatele a zajišťuje změny v modelu nebo v pohledu.

Pojem MVC je dnes často používán, ale spíše než o implementaci původní specifikace jde o volně převzatou myšlenku oddělení jednotlivých vrstev aplikace.

MVP (Model-View-Presenter) přináší oproti MVC některé změny:

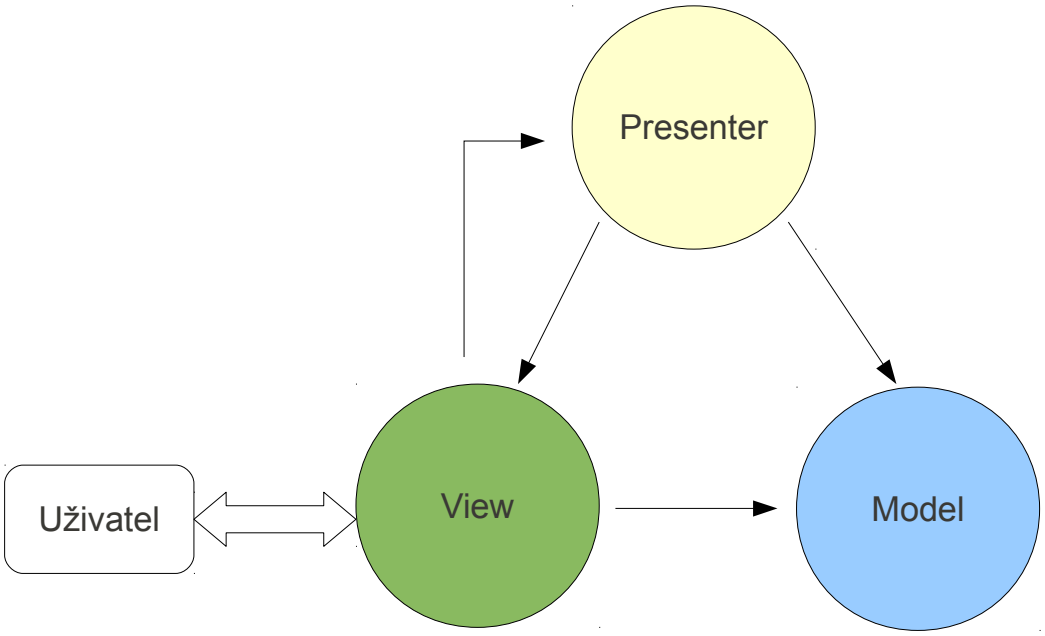
- uživatelský vstup i výstup plně kontroluje View (skrze ovládací prvky uživatelského),
- primární motivací pro oddělení View a Presenteru už není nutnost ošetření vstupu, důvody jsou čistě architektonické (prezentační vrstva s MVP se udržuje lépe než monolitické View),
- View má typicky přímou vazbu na Presenter,

- Presenter v mnoha případech přímo pracuje s View, takže i tato vazba je silnější než v případě vzoru MVC.

Funkce jednotlivých částí, tak jsou zobrazeny na obr. 4.1, jsou následující:

- **Model** zajišťuje přístup k datům a manipulaci s nimi.
- **View** navíc oproti MVC zpracovává uživatelský vstup. Typicky dělá pouze to, že např. v reakci na kliknutí myši zavolá nějakou metodu na Presenteru – pouze deleguje uživatelské akce.
- **Presenter** obsahuje aplikační a prezentační logiku. Manipuluje s Modelem, což pomocí systému notifikací zajistí aktualizaci View, nebo ovlivňuje View přímo.

Podrobnější popis zmíněných architektur naleznete v [11].



Obr. 4.1: Architektura Model-View-Presenter.

4.1.2 Hierarchie prezentační vrstvy

Aplikace bude mít modulární strukturu. Moduly rozdělují aplikaci do logických celků, mohou být zanořené – tzn. přiřazení rodičovskému modulu.

Každý presenter je spojen právě s jedním modulem. Jeden presenter poskytuje metody pro práci s jedno entitou modelu. K modelu přistupuje přes službu, jejíž název je v presenteru definován.

Každá metoda presenteru potom představuje jednu „obrazovku“ aplikace – tedy jeden formulář, nebo sestavu.

4.1.3 Adresářová struktura

Adresářová struktura je zvolena podle doporučení Nette frameworku. Zjednodušená struktura zobrazuje nejdůležitější adresáře a jejich obsah:

```
app/
    DbmModelModule/                # složka Front modulu
        forms/
            Entity
                EntityForm.latte   # třídy s˜definicí formuláře
                EntityForm.php     # šablona formuláře
        presenters/                # presentery modulu
            EntityPresenter.php
        templates/                 # šablony modulu
            Entity/                # šablony entity presenteru
    presenters/                    # presentery společné všem modulům
        BasePresenter.php          # abstraktní předek presenterů
    templates/                     # šablony využívané všemi moduly
        @layout.latte              # hlavní šablona
    models/                        # část modelu
                                   # rozdělení této části bude popsáno dále
    bootstrap.php                  # zaváděcí soubor aplikace
    config.neon                    # konfigurační soubor aplikace
    service.neon                   # konfigurace aplikačních služeb
    modelService.neon              # konfigurace služeb modelů
document_root/                     # root složka domény
    index.php                      # vstupní soubor aplikace
libs/                              # knihovny třetích stran
    Nette/
    Doctrine/
temp/                              # dočasné soubory frameworku a knihoven
```

4.1.4 Struktura jmenných prostorů

Jmenné prostory rozdělují aplikaci do několika částí a aplikaci zpřehledňují. Bylo zvoleno následující rozdělení:

Jmenné prostory modelu začínají názvem aplikace, pokračují slovem „Model“ a názvem hlavní entity (bude vysvětleno později).

Jmenné prostory presenterů jsou určeny názvem modulu.

Ostatní třídy jsou buď přímo ve jmenném prostoru aplikace, nebo, pokud se jedná o větší celky, mají vlastní jmenný prostor.

Jmenný prostor Kwido je používán pro znovupoužitelné části, často abstraktní třídy a rozhraní, u nichž se počítá s budoucím využitím v dalších projektech.

4.2 Modelové části aplikace

Po zhodnocení dostupných knihoven byla pro práci z databází zvolena ORM knihovna Doctrine 2 [22]. V následující kapitole je popsána architektura této knihovny a dále pak její zapojení do modelu.

4.2.1 Doctrine 2

Knihovna zajišťuje mapování objektů na relační databázi. Vytváří tedy abstrakci nad databází a odstiňuje zbytek aplikace od konkrétní databáze. V současné době podporuje databáze MySQL, PgSQL, Oracle, Sqlite. Splňuje tedy požadavek na nezávislost na databázi a díky dobře navržené architektuře je vhodným základem pro budování dalších pokročilých funkcí.

Doctrine 2 je postavena na návrhovém vzoru Data Mapper, od základu myšleno na optimalizaci výkonu, k dispozici je cachování na několika úrovních, používá se lazy-loading.

Doctrine 2 je rozdělena do tří vrstev:

Common definuje základní obecná rozhraní, třídy a knihovny. Například nástroje pro práci s kolekcemi, anotacemi, cachováním, událostmi apod.

DBAL (DataBase Abstraction Layer) abstrahuje zbytek aplikace od konkrétního typu databáze. Zavádí notaci DQL (Doctrine Query Language). DBAL je závislý na Common.

ORM (Object-Relational Mapping) nejvyšší vrstva, která zajišťuje mapování aplikačních objektů na relační databázi, jejich persistování a načítání. ORM je závislá na DBAL i Common.

EntityManager

Třída je vlastně jen fasádou, která zprostředkovává přístup k dalším částem ORM. Nabízí metody pro načítání, perzistenci a mazání dat. Níže uvedená ukázka kódu přibližuje typického použití EntityManageru.

Entity mohou být z pohledu EntityManageru v různých stavech, podle toho, jestli jsou nové, persistované nebo třeba smazané. Informace o stavu se uchovává mimo entitu, drží si ho sám EntityManager, přesněji řečeno UnitOfWork. Podle stavu každé entity se EntityManager rozhoduje, jak bude s entitou nakládat a co je s ní potřeba dělat. Doctrine 2 rozlišuje celkem čtyři různé stavy entit – nová, spravovaná, odpojená a smazaná.

Doctrine 2 využívá návrhový vzor Identity Map, který zajišťuje, že jeden záznam databáze reprezentuje vždy jen jeden objekt – při několika požadavcích na jeden záznam je vždy vrácena reference na jeden objekt. To zároveň eliminuje vícenásobné pokládání stejného dotazu do databáze.

Typické použití EntityManageru.

```
// $em je instance EntityManageru

$product = $em->find('Product', 12);
    // nacteni produktu z~DB
    // $product ma stav "spravovana"
$newProduct = new Product();
$em->persist($newProduct);
    // $newProduct ma stav "nova"

$category = $em->get('Category', 34);
$em->remove($category);
    // $category ma stav "smazana"

$em->flush();
    // preneseni vsech zmen do DB
```

Definice entit

Entity jsou základní kameny v Doctrine 2 a vůbec celé aplikace. Každá entita reprezentuje nějaký objekt reálného světa, takzvaný doménový objekt. Doctrine 2

podporuje několik způsobů definice – Yaml, XML, nebo PHP anotace. V práci byl zvolen způsob zápisu pomocí anotací, který je vidět na níže uvedené ukázce.

Podrobnější popis knihovny je k dispozici v [23].

Ukázka definice entity pomocí anotací.

```
<?php
/**
 * @Entity @Table(name="products")
 */
class Product
{
    /** @Id @Column(type="integer") @GeneratedValue */
    public $id;
    /** @Column(type="string") */
    public $name;
}
```

4.2.2 Organizace modelové části

Při klasické práci s EntityManagerem je třeba uvádět názvy tříd, se kterými chceme pracovat. Což je patrné z dříve uvedené ukázky.

Pro odstínění od konkrétních názvů bychom mohli vytvořit další vrstvu, která by se starala o jednu entitu modelu. Zajišťovala by její vytváření, načítání a další operace. Presenter by pak spolupracoval pouze s touto třídou a nemusel by znát názvy dané třídy.

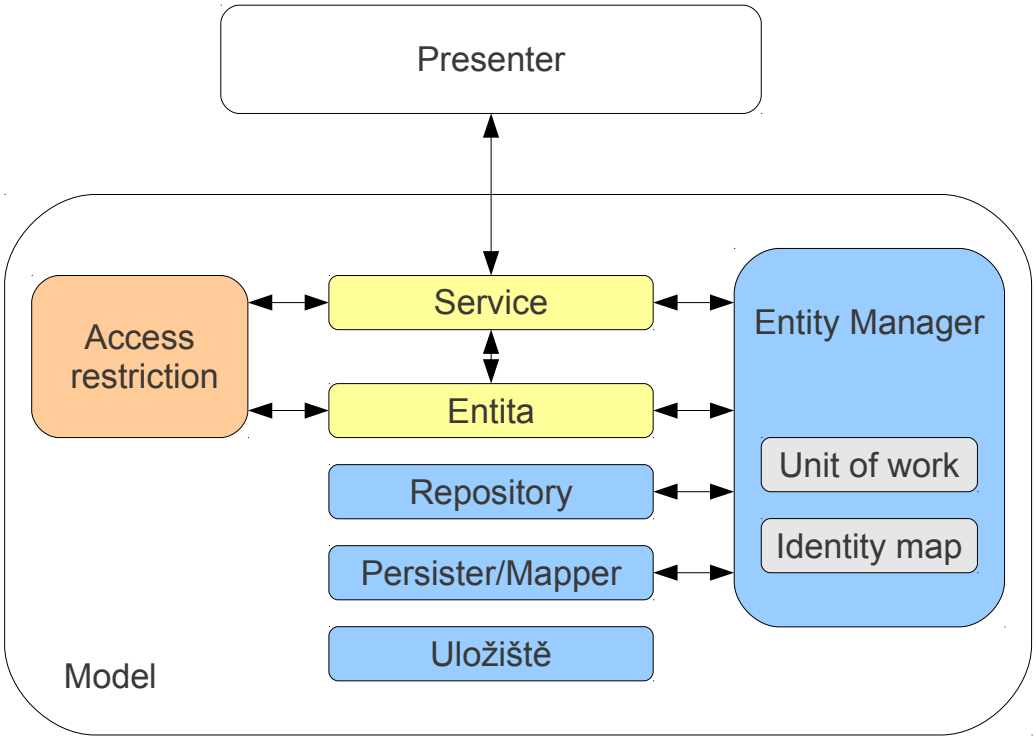
Takto navržené třídy služeb byly použity například v [24]. Služby jsou navíc získávány ze servisního kontejneru, jsou tedy vytvářeny až když je jich skutečně potřeba a v aplikaci existuje vždy jen jedna instance služby.

Toto řešení je velmi flexibilní a při vhodné modifikaci by umožnilo snadno kontrolovat přístup k jednotlivým entitám. Proto modelová část využívá toto rozložení vrstev, zobrazené na obr. 4.2.

Modře označené objekty jsou součástí Doctrine 2, oranžovou barvou je zvýrazněna třída zajišťující kontrolu přístupových oprávnění a žlutou barvou potom objekty definované konkrétní aplikací.

Oproti původnímu řešení v [24] došlo k několika změnám:

Doctrine entity jsou členěny do skupin. Ve skupině je vždy jedna hlavní entita a několik podřízených entit, které s hlavní entitou souvisí. Podřízené entity reprezentují například jazykové tabulky, nebo M:N tabulky.



Obr. 4.2: Rozdělení modelu do vrstev.

Definice/vyjasnění pojmů

Protože pojem entita je běžně používán v několika významech, přistoupíme nyní ke krátké definici pojmů.

- Doctrine ORM označuje jako entitu třídu, která odpovídá jedné databázové tabulce.
- Při popisu datového modelu pojem entita většinou reprezentuje objekt, nebo událost z reálného světa, kterou chceme v databázi zaznamenat. Entity jsou v databázi uloženy v tabulkách – jedna entita může být v případě potřeby rozložena do více tabulek.

V této práci je pojmem entita myšlen objekt z reálného světa, nezávisle na způsobu jeho uložení v databázových tabulkách. Má-li autor na mysli doctrine entitu, je to přímo uvedeno, nebo je použit pojem třída entity. Stejně tak pojmy hlavní a vedlejší (podřízená) entita značí, že se jedná o doctrine entity, a navíc popisují i jejich vzájemný vztah – respektive vztah databázových tabulek v relační databázi, které tyto doctrine entity představují.

Pro objasnění je uveden ještě stručný postup při definici entit s příkladem:

- při analýze identifikujeme entity (např. produkt),

- při návrhu datového modelu provedeme normalizaci a stanovíme strukturu databázových tabulek, v níž budou entity uloženy. (produkt obsahuje jazykově závislé položky, které budou uloženy odděleně v jazykové tabulce, ostatní informace budou uloženy v tabulce produkt),
- na základě databázových tabulek jsou generovány doctrine entity (vzniknou tedy 2 třídy entit, jedna pro produkt a jedna pro jazykové položky produktu),
- je proveden výběr hlavní entity – to je ta doctrine entita, která jednoznačně identifikuje celou entitu, tedy obsahuje primární klíč (hlavní entitu představuje třída Produkt, která obsahuje identifikátor entity – id produktu),
- ostatní entity jsou označeny jako vedlejší.

Přes službu je možné přistoupit pouze k hlavní entitě, k vedlejším entitám lze přistupovat pouze přes metody hlavní entity.

Jmenné prostory a adresáře doctrine entit

Rozdělení na hlavní a vedlejší entity reflektuje i adresářová struktura. V adresáři `models` jsou složky s názvy hlavních entit. V každé složce je pak třída hlavní entity, několik podřízených entit, pokud jimi hlavní entita disponuje.

Název jmenného prostoru je složen ze několika částí:

- jmenný prostor aplikace,
- označení modelové části,
- název hlavní entity.

Ve jmenném prostoru je ještě třída služeb (`Service`), která prezentační části aplikace poskytuje přístup k entitě.

Volitelně se zde může nacházet třída `Finder`, která umožňuje skládat dotazy do repozitáře. Instanci této třídy je možná získat ze servis třídy `Service` pomocí metody `getFinder()`. Použití `Finder` třídy je taktéž inspirováno prací [24].

Velká část funkcí modelu je poskytována abstraktními třídami, které byly vyčleněny do jmenného prostoru Kwido. Do budoucna je počítáno s jejich využitím v dalších projektech.

4.3 Přístupová oprávnění

Správa přístupových oprávnění byla rozdělena do dvou částí. První jsou přístupová oprávnění k prezentační části aplikace, tedy k modulům, presenterům a jejich metodám. Druhá část řeší přístupové oprávnění k modelu.

Základní princip je u obou částí stejný, definujeme zdroje, oprávnění a role. Rolím dále definujeme oprávnění k jednotlivým zdrojům. Uživatel pak vlastní určité role, které ho opravňují k přístupu ke zdrojům.

Zdroje jsou v případě modelu a prezentační části natolik odlišné, že musí být odděleny. Stejně tak oprávnění vykazují jisté rozdíly. Jednotlivé části budou nyní podrobně vysvětleny.

4.3.1 Oprávnění v prezentační části

Pro tuto část je možné použít třídu `Permission` z Nette frameworku [25], která implementuje námi požadovanou činnost. Dovoluje definovat role, zdroje a oprávnění a vytvářet pravidla. Třída však řeší pouze logiku definice a kontroly přístupu. Jak budou definovány jednotlivé položky třída neřeší. Umožňuje však podporu dědičnosti rolí a zdrojů.

Protože chceme oprávnění řídit daty, bude vytvořena třída `Acl`, která je potomkem třídy `Permission` a rozšiřuje její funkčnost o načítání jednotlivých položek z databáze.

Tab. 4.1: Databázové tabulky – Řízení přístupových oprávnění prezentační části

Entita	**Popis**
system_acl_role	Role
system_acl_privilege	Oprávnění
system_acl_resource	Zdroje
system_acl	Pravidla
system_modul	Moduly
system_presenter	Presentery
system_method	Metody
system_mpm	kombinace MPM (Modul-Presenter-Metoda)
system_acl_resource_2_system_mpm	M:N vztah Zdroje-MPM

Databázové tabulky

Pro uložení informací o oprávněních v prezentační části byly navrženy tabulky uvedeny v tab. 4.1, ER diagram je zařazen jako příloha B.1.1. Tabulky pro uložení rolí a zdrojů umožňují použití dědičnosti.

Zdroj je definován jako jedna nebo více kombinací MPM (Modul-Presenter-Metoda). V definici kombinace MPM je také možné vynechat metodu, a presenter. Je tedy možné definovat:

- modul (zahrnuje všechny presentery uvedeného modulu a jejich metody)
- modul a presenter (zahrnuje všechny metody uvedeného presenteru)

- modul-presenter-metoda (specifikuje právě jednu „obrazovku“ systému)

V tabulce `system_acl` jsou uloženy jednotlivá pravidla pro přístupová oprávnění.

4.3.2 Oprávnění v modelu

Role jsou společná pro oprávnění k modelu i k prezentační části. Oprávnění jsou uloženy ve stejné databázové tabulce, jsou však identifikována podle typu.

Zdroj je v modelové části definován jako kombinace entita, stav, tabulka (odpovídá doctrine entitě) a atribut. Definuje několik úrovní přístupu:

- přístup k entitě - povoluje přistupovat ke službě, která entitu zprostředkovává,
- přístup k záznamu:
  - vložit,
  - editovat,
  - smazat,
  - zobrazit,
- přístup k atributům záznamu:
  - vložit,
  - editovat,
  - zobrazit .

Každá z těchto úrovní může být dále omezena stavem entity.

Takto definované přístupy pokrývají poměrně velkou část požadavků na řízení přístupu, přesto ale najdeme situace, které uvedený model nepokryje. Jedná se např. o vlastnictví záznamu (uživatel může editovat pouze záznamy, které vytvořil). Pro splnění těchto požadavků byl navržen koncept omezení.

Omezení

Jejich účelem je omezit přístup uživatele k modelu na základě dalších podmínek. Omezení mohou být dvou typů.

Omezení nezávislá na datech záznamu jsou dána vlastnostmi, která nejsou závislá na datech modelu. Naopak jsou závislá na nějakém vnějším stavu. Jako příklad lze uvést časové omezení – data může role editovat pouze v pracovní době.

Tato omezení je tedy potřeba zkontrolovat před samotným načítáním dat. Výstupem pak značí povolení, nebo zamítnutí operace.

Omezení bude implementováno jako třída, která je potomkem abstraktní třídy `EvaluableRestriction`. Ta definuje jednu metodu `check()`, kterou každá třída omezení překryje vlastní implementací. O zavolání této metody na všech zvolených omezeních se stará třída zajišťující kontrolu oprávnění.

Omezení závislá na datech záznamu jsou formulována jako podmínky, které budou připojeny do požadavku na získání dat. Podmínky se poté objeví v SQL dotazu položeném do databáze a ta již zajistí, že vrácená data tuto podmínku splňují. Příkladem může být již zmíněné vlastnictví záznamu. Do dotazu je doplněna podmínka `uživatel = aktuální uživatel`.

Konfigurace takovéhoto omezení obsahuje cestu (atribut), na který se bude omezení aplikovat a třídu, která se stará o získání hodnoty (např. identifikátor aktivního uživatele).

Omezení bude implementováno jako třída, která je potomkem abstraktní třídy `ConditionalRestriction`, která definuje jednu abstraktní metodu `getConditionValue()`, kterou každá třída omezení překryje vlastní implementací. Další metody (nastavení cesty a dalších parametrů omezení) jsou definované přímo v rodičovské třídě.

Počet omezení pro přístupová pravidla není nijak omezen. Omezení v kombinaci výše definovanou kontrolou přístupu rozšiřuje možnosti řízení přístupu tak, že by neměl být problém splnit libovolně složitý požadavek.

Tab. 4.2: Databázové tabulky – Řízení přístupových oprávnění modelu

Název tabulky	**Popis**
system_acl_role	Role
system_acl_privilege	Oprávnění
system_acl_restriction	Omezení
system_dbm_resource	Zdroje
system_dbm_acl	Pravidla
system_acl_restriction_2_system_dbm_acl_combination	Kombinace omezení
system_dbm_entity	Entity
system_dbm_table	Tabulky
system_dbm_attribute	Atributy
system_dbm_state	Stavy
system_dbm_state_transition	Přechody
system_dbm_state_transition_list	Oprávnění k přechodům

Databázové tabulky

Seznam tabulek, které řízení přístupu k modelu využívá je uveden v tab. 4.2. Pro uložení rolí a oprávnění jsou použity stejné tabulky jako v případě řízení přístupu k prezentační vrstvě. ER diagram je zařazen jako příloha B.1.1.

4.4 Životní cykly entit

Některé z entit v informačním systému mají definován životní cyklus. Záznam takové entity během své existence prochází různými stavy. Změna stavu je označována jako přechod. V jednotlivých stavech platí pro záznam jiná pravidla, ať už jde o přístup k záznamu, možné prováděné akce, nebo validační pravidla. Tyto pravidla je potřeba v systémové části kontrolovat.

Obecně můžeme podle životního cyklu rozdělit entity do tří skupin:

- entity bez životního cyklu
- entity z jednoduchým životním cyklem – mají pouze dva stavy, v jednom je záznam aktivní (platný), v druhém ne.
- entity s vlastním životním cyklem – definují vlastní stavy a přechody mezi nimi.

Toto dělení bude dále používáno.

V navrhované koncepci modelu je životní cyklus definován pro celou entitu. Toto zdánlivé omezení však ve skutečnosti nečiní problémy, protože vedlejší entity nejsou z logiky věci samostatnými celky v modelu. Pouze reflektují způsob strukturování dat v relačních databázích. Podporu životního cyklu tedy nepotřebují.

S existencí životního cyklu bylo počítáno už při návrhu řízení přístupu k modelu, kontrolu oprávnění řeší tedy tento modul. I pro jeho potřeby je však třeba jednotlivé stavy ukládat. Stavy jsou sice definovány číslem, pro zobrazení by však měly být pojmenovány. Protože jedním z požadavků je podpora vícejazyčných aplikací, měla navržena struktura pro ukládání vícejazyčných popisku. Obdobná situace je u přechodů – ty jsou taktéž označeny názvem a podporují vícejazyčnost.

Přechody mezi jednotlivými stavy budou omezeny jenom na určité role. To musí být v databázi také zohledněno.

Tab. 4.3: Databázové tabulky – Správa životního cyklu

Název tabulky	**Popis**
system_dbm_entity	Entity
system_dbm_state	Stavy
system_dbm_state_transition	Přechody
system_dbm_state_transition_language	Popisky
system_dbm_state_transition_list	Oprávnění k přechodům
system_dbm_state_transition_list_language	Popisky

Databázové tabulky

Databázové tabulky, které ponesou informace o životním cyklu jsou uvedeny v tab. 4.3. ER diagram zobrazující tyto entity je zařazen jako příloha B.2.

4.5 Podpora jazykových mutací

Chceme-li v relační databázi ukládat vícejazyčná data tak, aby bylo možné libovolně přidávat další jazyky, volíme rozdělení dat do dvou databázových tabulek, kdy jedna obsahuje data jazykově nezávislá, druhá jazykově závislá data, doplněná o atribut specifikující jazyk a atribut nesoucí informaci o příslušnosti k jazykově nezávislým datům. Vztah mezi tabulkami je 1:N, tedy jednomu jazykově nezávislému záznamu může patřit několik jazykově závislých záznamů.

Takováto databázová struktura je použita i u entit souvisejících s životním cyklem, které byly popsány v minulé kapitole (viz příloha B.2).

V modelové části aplikace budou tyto entity detekovány, tabulky s jazykově nezávislými položkami bude označena jako hlavní, tabulky s jazykovými položkami bude zařazena do téže skupiny a označena jako jazyková.

V hlavní entitě budou vytvořeny metody pro přístup k jazykovým položkám.

4.6 Číselníky

Číselníky slouží k ukládání dat definovaných výčtem hodnot. Protože výčet těchto hodnot se může v čase měnit a může nastat požadavek na udržování starých hodnot číselníků pro staré záznamy a aktuálních hodnot pro záznamy nové. Proto byly číselníky navrženy jako variantní. ER diagram je zařazen jako příloha B.3.

Číselník může mít několik variant, kde platnost konkrétní varianty je dána buď časové, splněním definované podmínky, nebo stavem. Konkrétní číselník obsahuje libovolné množství hodnot, které mohou být taktéž časově, nebo jinak podmíněné.

Každý číselník pak může být podřízen jinému číselníku, jeho hodnoty je pak možno omezit na základě volby hodnoty nadřazeného číselníku.

4.7 Šablony

Šablona v Nette Frameworku je složena z jedné hlavní šablony (layout), do které může být vložen libovolný počet bloků a v nich mohou být rekurzivně vloženy další bloky. Struktura, která umožňuje uložit hierarchii šablon v databázi je vložena jako příloha B.4.

Protože, stejně jako u číselníků, může dojít k požadavku na změnu šablony a na zobrazení starších dat ve starších šablonách a nových dat v nových šablonách, byla struktura databáze navržena tak, aby umožnila ukládat jednotlivé varianty. Platnost varianty je možné určit stavem, časem platnosti (od-do), nebo podmínkou.

4.8 Navigace

V rámci návrhu řízení přístupu oprávnění byla definována kombinace MPM (Modul-Presenter-Metoda). Její využití pro navigaci je nasnadě, protože jednoznačně specifikuje jednotlivé částí prezentační vrstvy, na které chceme z navigace odkazovat. Položky v navigaci jsou jazykově závislé, proto jsou popisky definovány ve speciální jazykové tabulce.

Navigace by měla být hierarchická, stejně jako je tomu u kombinace MPM. Položky v menu by však měly umožnit i jinou hierarchii, než jen tu danou hierarchií modulu. Zároveň je třeba určovat i pořadí položek v rámci jedné úrovně. Proto byla pro uložení těchto dat zvolena struktura traverzování kolem stromu.

Pro umístění navigace v šabloně jsou definovány pozice, závislé na konkrétní šabloně. Do těchto pozic jsou potom jednotlivé položky menu přiřazeny a to tak, že jedna položka může být v několika pozicích.

Tab. 4.4: Databázové tabulky – Navigace

Název tabulky	**Popis**
system_menu_item	Položka v menu
system_menu_item_language	Jazyková tabulka položky
system_mpm	kombinace MPM
system_menu_item_2_system_menu_position	Umístění položky do pozic
system_menu_position	Pozice pro menu

Databázové tabulky a ER diagram

Databázové tabulky jsou uvedeny v tab. 4.4. ER diagram zobrazující tyto entity je zařazen jako příloha B.5.

4.9 Uživatel, role a uživatelská nastavení

K uložení dat o uživateli slouží tabulka `system_user`. Ta obsahuje základní informace, které jsou k identifikaci uživatele potřeba. Pokud by bylo potřeba v některé

aplikaci ukládat o uživateli další data, je možné tuto tabulku rozšířit o další atributy.

V použitém modelu oprávnění může uživatel vlastnit více rolí. Jednotlivé uživatelské role jsou ukládány v tabulce `system_user_acl_role`. Ta, kromě cizích klíčů definujících vztah M:N, obsahuje ještě dva atributy. Atribut `using` značí, jestli může uživatel tuto roli používat, atribut `assing` značí, zda může tuto roli přidělovat jiným uživatelům (za předpokladu, že má přístup k potřebnému formuláři).

Uživatelská nastavení se mohou vztahovat k jedné „obrazovce“ prezentační vrstvy – tedy k jedné akci presenteru, k celému presenteru, k modulu, nebo k celé aplikaci. Pro zaznamenání této závislosti je uživatelské nastavené přiřazeno ke konkrétní kombinaci MPM (Modul-Presenter-Metoda). Pokud tato není uvedena, platí nastavení pro celou aplikaci. Nastavení bude pojmenováno tagem.

Protože formát dat, které budou jednotlivé komponenty ukládat může být značně rozdílný, nebudou data uložena v databázi strukturovaně, ale pouze jako text. Pro jejich uložení by bylo vhodné použít formát JSON, nebo obdobný formát.

Tab. 4.5: Databázové tabulky – Uživatelé, role a nastavení.

Název tabulky	**Popis**
system_user	Uživatel
system_user_acl_role	Role uživatele
system_user_settings	Uživatelská nastavení

Databázové tabulky a ER diagram

Výpis navržených tabulek je uveden v tab. 4.5. Pro uložení uživatelského nastavení byla navržena tabulka `system_user_settings`. Ta kromě zmíněného nastavení a tagu, kterým je pojmenováno, obsahuje také závislost na konkrétním uživateli (tabulka `system_user`) a na kombinaci MPM (tabulka `system_mpm`). Oba tyto vztahy jsou nepovinné - aby bylo možné uložit nastavení platné pro všechny uživatele a pro celou aplikaci. ER diagram zobrazující tyto entity je zařazen jako příloha B.6.

4.10 Formuláře řízené dle oprávnění

Chceme-li měnit styl vykreslení formulářových prvků podle oprávnění, jeví se jako nejvhodnější řešení definovat výčet všech možných prvků formuláře. V definici budou obsaženy všechny informace potřebné pro vykreslení prvku v libovolném oprávnění. Při vykreslování potom porovnáme výčet s atributy pro které máme potřebná oprávnění a podle oprávnění je vykreslíme jako viditelné, editovatelné, případně je nevykreslíme vůbec.

Následujících kapitoly se věnují definici a vykreslení formulářů typu VF a IF. Formulářům typu RL a QF, kde je situace mírně odlišná, se věnuje kapitola 4.11

Zkrácena ukázka definice formuláře ve formátu neon.

```
name:
    type: text
    label: Nazev
lifeCycle:
    type: codebook
    label: Life Cycle
state:
    type: state
    label: Stav
```

4.10.1 Definice formuláře

Pro definici formuláře je používáno konfigurační pole. Pro přehlednost je zapisováno ve formátu NEON [26], který je lépe čitelný. Třída pro parsování formátu neon a jeho převod na pole je součástí Nette frameworku.

V uvedené ukázce je vidět definici tří formulářových prvků, různých typů. Jednotlivé typy jsou popsány v následující kapitole. První část definice určuje název formulářového prvku a musí odpovídat názvu atributu v datovém zdroji. Pro každý prvek je dále definován typ a popisek (label).

Tab. 4.6: Typy prvků pro definici formuláře.

Typ	**Popis**
text	textové pole - jednořádkové
password	pole pro heslo
textArea	textové pole - víceřádkové
file	pole pro upload souboru
hidden	skryté pole
checkbox	volba Ano/Ne
select	výběr ze seznamu
multiSelect	vícenásobný výběr
state	stav (výběr ze seznamu)
codebook	číselník (výběr ze seznamu)
relation	cizí klíč (výběr ze seznamu)
multiContainer	zobrazení vedlejší entity

4.10.2 Formulářové prvky

Výčet formulářových typů je uveden v tabulce 4.6. Prvních osm prvků je převzato z Nette frameworku a odpovídá jednotlivým formulářovým prvkům HTML. Nyní budou popsány prvky, které byly navrženy v rámci této práce.

Typ state

Typ je určen pro zobrazení/editaci stavu entity. Je možné ho použít pouze u atributu, který představuje stav. Samozřejmostí je zobrazení lokalizovaných textových popisků stavu namísto jejich číselné reprezentace. Prvek je vykreslen jako typ select.

Typ codebook

Tento typ je určen pro zobrazení číselníku. Jeho použití je omezeno na atribut, který ukazuje na číselník. Hodnoty jsou reprezentovány lokalizovaným textovým popiskem. Prvek je vykreslen jako typ select.

Typ relation

Tento typ slouží k zobrazení vazby mezi entitami. Jeho použití je omezeno na atribut, který reprezentuje vztah. Stejně jako předchozí prvky je vykreslen jako typ select.

Typ select

V Nette formulářový typ select reprezentuje stejnojmenný HTML prvek – tedy roletku s hodnotami – nezávisle na jejich počtu. Při velkém počtu hodnot je však tento prvek zcela nevyhovující. Proto se jeví vhodnější přidat do objektu logiku, která rozhodne o způsobu vykreslení podle počtu záznamů. Pokud je počet hodnot nízký, bude prvek vykreslen jako roletka, při větším počtu jako formulář typu LOV.

Typ multiContainer

Typ je určen pro zobrazení kolekce záznamů z podřízené entity, které je praktické editovat hromadně. Nejčastěji se jedná o jazykové položky, kdy chceme v jednom formuláři editovat jak jazykově nezávislé hodnoty, tak texty pro všechny lokalizace. Tento typ představuje vlastně definici několika menších formulářů pro jednotlivé prvky v kolekci. Jeho definice je tedy složitější. Je třeba definovat iterátor, který určuje záznamy (např. které jazyky), bude možno editovat. Dále je třeba definovat datový zdroj pro získání kolekce.

4.10.3 Nezávislost vzhledu na obsahu

Jedním z požadavků na aplikace je oddělení prezentační a aplikační logiky u formulářů. Formuláře v Nette mají hierarchickou strukturu. Formulář je kontejner, do něhož lze vkládat jednotlivé formulářové prvky nebo jiné kontejnery. Při vykreslování formuláře je pak tato struktura postupně procházena a vykreslována.

Kromě standardního vykreslování nabízí Nette také tzv. „ruční vykreslování“, které umožňuje definovat pro formulář šablonu. Formulář je potom vykreslen do této šablony, která obsahuje speciální makra [27].

Šablona pro ruční vykreslování může definovat libovolný layout. Na místě kde má být vykreslen formulářový prvek je použito makro `{input nazevPrvku}`, pro popisek prvku existuje makro `{label nazevPrvku}`.

Protože jsou formuláře řízeny oprávněním, nemusí být vždy daný prvek definován. Proto je zobrazení prvků podmíněno jeho existencí, která je testována makrem `ifset`. Jak již název napovídá, makro provádí testování podmínky `if (isset(...))`. Na níže uvedené ukázce je definice šablony formuláře s použitím uvedených maker.

Jak již název napovídá, možnost ručního vykreslování počítá s tím, že programátor definuje kompletní strukturu v šabloně ručně. Proto neřeší procházení této hierarchie, které je ale pro daty řízené formuláře velmi užitečné. Původní implementace obsahuje pouze makro `{formContainer nazevKontejneru}` pro nastavení daného kontejneru jako aktivního. Pro pohodlnější definici šablon byly proto definovány další prvky.

- makro `containersAs` – pro procházení jednotlivých kontejnerů v aktuálním kontejneru
- makro `controlsAs` – pro procházení jednotlivých formulářových prvků v aktuálním kontejneru
- `$_this` – tato proměnná ukazuje vždy na aktuálně aktivní kontejner.

Díky těmto makrům je možné dynamicky procházet strukturu formulářových kontejnerů a prvků. Pro ověření existence prvku není třeba znát celou hierarchii, stačí použít `$_this`.

Šablonu formuláře je tedy možné libovolně upravovat, aniž by to jakkoli ovlivnilo jeho funkčnost. Díky nově definovaným makrům je navíc umožněn snadný výpis opakujících se prvků v cyklu.

4.10.4 Vykreslení formulářových prvků podle oprávnění

Jednotlivé formulářové prvky se vykreslují až při zpracování šablony. To jak se daný prvek vykreslí je určeno třídou, jejíž instanci představuje. Pro každý prvek formuláře existuje v Nette konkrétní třída. Pro správné vykreslení stačí tedy pouze vytvořit správné instance ve formuláři.

Vytvoření formulářových prvků je rozděleno do dvou částí. Nejprve se vytváří instance prvků, které jsou editovatelné. Z datového zdroje je získán seznam formulářových prvků, ke kterým má uživatel oprávnění pro editaci. Vytvoření příslušné instance mají na starosti tzv. tovární metody. Podle typu prvku je tedy volána tovární metoda, která instanci vytvoří a přiřadí ji příslušnému formulářovému kontejneru.

V následujícím kroku se celý postup opakuje pro viditelné prvky. Kromě různých továrních metod se vytváření editovatelných a viditelných prvků tedy neliší.

Pro prvky, které nejsou ani viditelné ani editovatelné nejsou vytvořeny žádné instance. V šabloně jsou tyto prvky vynechány z vykreslování.

Ukázka šablony formuláře.

```
{form $form}
<fieldset>
    <p n:ifset="$_this['name']">
        {label name}
        {input name}
    </p>
    <fieldset n:ifset="$_this['DbmAttributeLanguage']">
        <legend>Jazykove mutace</legend>
        {formContainer DbmAttributeLanguage}
            {containersAs $cont}
                {formContainer $cont}
                    {ifset $_this['id']}
                        {input id}
                    {/ifset}
                    <p n:ifset="$_this['label']">
                        {label lngLabel}
                        {input lngLabel}
                    </p>
                {/formContainer}
            {/containersAs}
        {/formContainer}
    </fieldset>
    <p n:ifset="$_this['state']">
        {label state}
        {input state}
    </p>
    <p>{input s~value => "Save"}</p>
</fieldset>
{/form}
```

4.10.5 Načítání dat z datového zdroje

Pro předávání dat je formuláři přidělen datový zdroj a jednotlivé formulářové prvky se snaží načíst svoji hodnotu z tohoto zdroje, dle svého názvu.

Pro tvorbu složitějších formulářů je vhodné využít hierarchii – tedy pokud je prvek umístěn v kontejneru, požádá kontejner o datový zdroj. Pokud kontejner nemá definovaný žádný zdroj, požádá o něj svého předka – tedy nadřazený kontejner, nebo formulář, který je prakticky také kontejnerem (třída `EntityForm`, která reprezentuje formulář je potomkem třídy `FormContainer`).

Protože Nette tuto funkčnost nepodporuje je třeba do třídy`FormContainer` doplnit metody pro nastavení a získání zdroje a vlastnost pro jeho uložení.

4.11 Formuláře Record List a Query Form

Jak již bylo zmíněno, situace u formulářů typu RL a QF je poněkud odlišná. Hlavním rozdílem je, že tyto formuláře nemají jako datový zdroj jeden záznam. Podle změn hodnot v QF je třeba sestavit dotaz, pomocí kterého potom RL získá data.

Pro základní implementaci bude pro tyto formuláře využita komponenta Datagrid [18], která tyto dva formuláře slučuje do jedné komponenty. Její nevýhodou je, že neumožňuje filtrovat nad atributy, které nejsou zobrazeny. Toto omezení nám při základní implementaci nebude vadit. Datový zdroj by však měl být implementován bez tohoto omezení, aby v případě jiné prezentační komponenty bylo toto omezení odstraněno.

4.11.1 Definice formuláře

Datagrid podporuje různé typy datových zdrojů, mezi nimi jsou i Doctrine, respektive QueryBuilder – to je třída, která umožňuje tvořit dotazy do databáze. Pracuje na úrovni doctrine entit a jejich atributů. O sestavení SQL dotazu se stará knihovna Doctrine.

Na níže uvedené ukázce definice datagridu s doctrine entitami je patrné, že definice není příliš přehledná. Jednotlivé položky (sloupce) je třeba nejprve zapsat do dotazu. Potom je třeba uvést sloupec v mapování a přiřadit mu alias. Tím je definován datový zdroj, který je předán datagridu. Teprve nyní může být sloupec definován v datagridu, může mu být přiřazen filtr, případně další nastavení.

Ukázka použití Datagridu s Doctrine (převzato z [19]).

```
//$em instanceof Doctrine\ORM\EntityManager
$grid = new \DataGrid\DataGrid;

//prepare datasource
$dataSource = new \DataGrid\DataSources\Doctrine\QueryBuilder(
$em->createQueryBuilder()
                //columns to be used
    ->select('u.id, u.name, u.regTime, a.city')
                //master table
    ->from('Models\User', 'u')
                //joined table (one-to-one association)
    ->join('u.address', 'a')
);

//mapping betweeen DataGrid's column names and entity columns
$dataSource->setMapping(array(
    'id'    => 'u.id',
    'name'  => 'u.name',
    'time'  => 'u.regTime',
    'city'  => 'a.city',
));

//finally, set datasource to DataGrid
$grid->setDataSource($dataSource);

//now we're working with mapped fields
$grid->addNumericColumn('id', 'ID')->addFilter();
$grid->addColumn('name', 'Jmeno')->addFilter();
$grid->addColumn('city', 'Mesto')->addFilter();
$grid->addDateColumn('time', 'Datum registrace')->addDateFilter();
```

Uvedený způsob je zdlouhavý, nepřehledný a v případě změny vyžaduje úpravy kódu na více místech. Jako řešení se jeví definovat formát konfigurace, na jejímž základě proběhnou tyto nastavení automaticky. Zároveň je možné lépe kontrolovat přístup k modelu. Pro lepší čitelnost je pro konfiguraci opět použit formát neon.

Ukázka definice formuláře ve formátu neon.

```
id:
    mapping: id
    type: numeric
    label: Id
e_name:
    mapping: dbmEntity.name
    type: text
    label: Entita
name:
    mapping: name
    type: text
    label: Name
    filter:
        type:text
label:
    mapping: languages.label
    type: text
    label: Label
type:
    mapping: type
    type: codebook
    label: Type
    filter:
        type: codebook
state:
    mapping: state
    type: state
    label: Stav
    filter:
        type: state
```

Definice je podobná jako u formulářů VF a IF, navíc jsou zde položky:

- `mapping` – ta udává název atributu, nebo cestu k němu oddělenou tečkami, pokud se nejedná přímo o atribut hlavní entity. Není již potřeba vytvářet QueryBuilder, ten je z konfigurace vytvořen automaticky. Podle potřeby jsou automaticky mapovány další entity. Mapování je realizováno metodou `leftJoin`, takže jsou vždy vytvořeny všechny kombinace s připojovanou tabulkou,
- `filter` – definuje jestli bude možné podle atributu filtrovat a jakým způsobem bude filtr zobrazen.

4.11.2 Typy formulářových prvků

Přehled prvků je uveden v tabulce 4.7. Většina prvků je definována datagridem. Přidány byly prvky `codebook` a `state`. U nich dochází k nahrazení hodnoty textovou položkou a do filtrů jsou přidány hodnoty, kterých může atribut nabývat. Filtr je typu `select`. Typ `relation` zde není potřeba, libovolné položky ze vztahů je možné namapovat pomocí tečkové notace.

Tab. 4.7: Typy prvků pro definici formuláře.

Typ	**Popis**
text	textové pole
numeric	číslo
date	datum
checkbox	hodnota Ano/Ne
position	pozice (lze měnit)
image	obrázek
select	výběr ze seznamu
codebook	číselník
state	stav

4.11.3 Omezení

Jak již bylo uvedeno, entity jsou přidávány metodou `leftJoin`. Pokud tedy existuje záznam v hlavní tabulce a k němu dva záznamy v podřízené tabulce (např. popis ve dvou jazycích) bude tento záznam zobrazen v datagridu jako 2 záznamy - každý s jedním jazykovým záznamem. To může být někdy nežádoucí. Pro doplnění dalších omezujících podmínek pro zobrazené záznamy můžeme použít mechanizmus omezení, který byl již popsán v kapitole 4.3.2.

Omezení je pak definováno buď přímo ve službě, která se stará o vytvoření QueryBuilderu, nebo při definici formuláře. Omezení nastavená v řízení přístupu jsou samozřejmě aplikována automaticky. Tyto doplněná omezení jsou určena pro omezení datového zdroje v celé aplikaci, nebo pro konkrétní formulář.

Praktickým příkladem použití je např. zobrazení jazykově závislých popisků. Ačkoliv uživatel má oprávnění vidět a editovat více jazykových verzí, je zbytečné zobrazovat každou položku v RL vícekrát. Na datový zdroj je tedy aplikováno omezení na aktuální používaný jazyk.

4.12 ER diagram systémové části aplikace

Některé části ER diagramu byly popsány. V tabulce C.1 je kompletní výpis tabulek systémové části aplikace. Uvedené tabulky podporují všechny dříve uvedené vlastnosti. V příloze B.7 je zařazen ER diagram, který zobrazuje všechny entity a vazby. Pro přehlednost jsou tabulky barevně odlišeny podle svého účelu:

- červená – je použita pro jazykové tabulky,
- zelená – vyznačuje tabulky pro řízení přístupových práv,
- modrá – označuje ostatní, datové tabulky.

5 VÝVOJOVÉ PROSTŘEDÍ

Na základně analýzy požadavků na vývojové prostředí byly stanoveny jednotlivé části aplikace a navržen jejich datový model. Rozdělení jednotlivých částí je stanoveno podle souvisejících entit, se kterými pracují a podle pořadí, ve kterém je třeba jednotlivé údaje zadávat. První podkapitola uvádí postup činností při vytváření projektu ve vývojovém prostředí. V dalších kapitolách jsou popsány jednotlivé části vývojového prostředí, jejich účel a datový model.

Některé entity jsou jen rozšířením entit navržených v systémové části, upravené tak, aby bylo možné uložit data pro více projektů. V takovém případě budou popsány hlavně rozdíly oproti systémovým entitám.

5.1 Postup činností ve vývojovém prostředí

Postup vytváření projektu ve vývojovém prostředí by měl být uspořádán následovně:

1. Projekt
   - Nový projekt
     - Založení projektu
     - Nastavení projektu
   - Existující projekt
     - Aktivace projektu
2. Databázový model
   - Načtení existujícího modelu
   - Možnost automatického překladu jazykově závislých položek
   - Nastavení a úpravy
     - detekovaných entit
     - načtených tabulek
     - načtených atributů
     - načtených vztahů
     - speciálních typů sloupců
3. Role
   - Přidání nových rolí
   - Úprava výchozích rolí
4. Životní cykly
   - Rozdělení entit podle životního cyklu
   - Definice vlastních životních cyklů entit
     - Definice stavů
     - Definice přechodů
   - Nastavení validačních pravidel pro stavy

    - Nastavení kontrolních funkcí
5. Přístupová oprávnění modelu
    - Možnost generovat oprávnění
    - Úpravy oprávnění
6. Prezentační část
    - Moduly
    - Presentery
    - Komponenty
    - Přístupová oprávnění k prezentační části
7. Generátor
    - php kód
        - modelová část aplikace
        - prezentační část aplikace
    - SQL
        - zakládací script
        - systémová data

5.2 Správa projektu

Tento modul umožňuje zakládat nové projekty, nastavovat vlastnosti projektu – použitý databázový systém, podporované jazyky, připojení k databázi.

Před prací s dalšími moduly je třeba vybrat projekt se kterým se pracuje. Tento modul je tedy vstupním bodem do celého vývojového prostředí.

Tab. 5.1: Databázové tabulky – Správa projektu

Název tabulky	**Popis**
project	Projekt
database_connection	Připojení k databázi
database_schema	Databázové schema
language	Tabulka jazyků
project_language_version	Jazykové verze projektu
dictionary	Slovník pro překlad názvů db. modelu
database_system	Podporované databázové systémy
db_content_type	Typ obsahu
db_date_type	Datový typ
database_system_type_alias	Alias pro datový typ

5.2.1 Databázové tabulky

Tabulky pro uložení informací o projektu jsou uvedeny v tab. 5.1. K projektu je přiřazen databázový systém a databázové schema, s nímž projekt pracuje. Vazbu na entitu Projekt mají i další entity, které jsou vázané ke konkrétnímu projektu.

Aby nebyl systém závislý na konkrétní databázi, nejsou použity datové typy konkrétní databáze, ale je vytvořen model, který umožňuje větší míru abstrakce. Jsou definovány typy obsahu (např. krátký text, dlouhý text, celé číslo, ...). Z typů obsahu potom vychází datové typy - ty přidávají informaci o rozsahu možných hodnot v případě číselných hodnot, nebo maximální délce v případě řetězců.

Nakonec jsou definovány aliasy datových typů pro různé databázové systémy. Při načítání dat jsou datové typy určovány právě podle těchto aliasů.

ER diagram je zařazen jako příloha D.1

5.3 Datový model

Entity pro uložení datového modelu projektu ve vývojovém prostředí kopírují strukturu modelu tak, jak byla popsána v kapitole 4.2.2. Došlo však k jejich rozšíření o informace, na základě kterých bude později generován zdrojový kód. Entita Atribut byla například rozšířena o atributy jako datový typ, rozsahy, ...

V rámci informací o datovém modelu se počítá i s jazykově závislými daty (popisky atributů použité např. ve formulářích). Ty jsou uloženy v jazykových tabulkách.

Nově jsou zde definovány tzv. vypočítané atributy – tedy atributy, jejichž hodnotu lze získat výpočtem, nebo jinou operací, z ostatních hodnot. Takový atribut je určen jen ke čtení a ve vygenerovaném modelu bude reprezentován metodou, která hodnotu vypočítá a vrátí. V systémových tabulkách bude takový atribut uveden v tabulce atributů a bude určen pouze ke čtení.

Dále jsou definovány zobrazovací transformace – tedy metody, které zajistí změnu formátu hodnoty při práci s ní. S využitím těchto funkcí je počítáno např. pro lokalizace formátu data.

Data pevně daná datovým modelem budou zobrazeny pouze ke čtení a jejich změna nebude možná. U ostatních dat bude možné provádět změny.

5.3.1 Databázové tabulky

Seznam tabulek je uveden v tab. 5.2. Větší část uvedených entit slouží k uložení informací načtených z datového modelu a informací, které budou zadávány manuálně, ale s databázovým modelem souvisí. ER diagram je zařazen jako příloha D.2.

U jazykových tabulek nejsou, pro lepší přehlednost, zobrazeny definované atributy. Jejich obsah je vždy stejný – obsahují atributy:

- `label` – krátký popisek,
- `description` – delší popis,
- cizí klíče k rodičovské tabulce,
- cizí klíče k tabulce jazyků.

Stejně tak tabulky, které již byly uvedeny dříve nebudou na dalších diagramech zobrazeny podrobně.

Tab. 5.2: Databázové tabulky – Datový model

Název tabulky	**Popis**
project	Projekt
dbm_entity	Entita
dbm_entity_language	Entita – jazyková tabulka
dbm_table	Tabulka
dbm_table_language	Tabulka – jazyková tabulka
dbm_attribute	Atribut
dbm_attribute_language	Atribut – jazyková tabulka
dbm_attribute_calculated	Vypočítaný atribut
dbm_attribute_calculated_language	Vypočítaný atribut – jazyková tabulka
dbm_foreign_key	Cizí klíč
dbm_index	Index
dbm_index_attribute	Atributy indexu
codebook	Číselník
database_schema	Databázové schéma
db_date_type	Datový typ
view_transformation	Transformace zobrazení

5.4 Životní cykly

Entity pro stavy, přechody a oprávnění k nim jsou definovány stejně jako v systémové části. Navíc jsou definovány entity pro uložení kontrolních funkcí. Data z těchto entit budou při generování převedena do kódu. Proto se tyto entity v systémové části nenachází.

Počítá se s vytvořením kontrolních funkcí pro každou entitu. Definice funkce a mapování parametrů by mělo být odděleno. Kontrolní funkci s namapovanými

parametry označíme jako kontrolu. Vykonání kontroly je možné navázat na životní cyklus entity – ke konkrétnímu stavu a události.

5.4.1 Databázové tabulky

Výpis tabulek je uveden v tab. 5.3. Pro uložení kontrolních funkcí je navržena tabulka `revision_function`. Pro parametry, které funkce přijímá je navržena tabulka `revision_function_param`. Mapování atributů na parametry je definováno v tabulce `dbm_revision_function_param`. To umožňuje použití jedné kontrolní funkce několikrát, pokaždé s jinými parametry. Kombinace kontrolní funkce a mapování atributů je označeno jako kontrola a uloženo v tabulce `dbm_revision`. K navázání kontrol na životní cyklus entity je definována tabulka `dbm_revision_event`. ER diagram je uveden v příloze D.3.

Tab. 5.3: Databázové tabulky – Životní cykly

Název tabulky	**Popis**
acl_role	Role
dbm_entity	Entity
dbm_revision	Kontroly
dbm_revision_event	Události
dbm_revision_function_param	Mapování parametrů
dbm_revision_language	Kontrola – jazyková tabulka
dbm_state	Stavy
dbm_state_language	Stavy – jazyková tabulka
dbm_state_transition	Přechody
dbm_state_transition_language	Přechody – jazyková tabulka
dbm_state_transition_list	Oprávnění k přechodům
revision_function	Kontrolní funkce
revision_function_param	Parametry kontrolních funkcí

5.5 Validace

Validační pravidla můžeme rozdělit do dvou skupin. První skupina je určena datovým modelem. Taková pravidla musí být splněna za všech okolností a nebudou se v průběhu životního cyklu nijak měnit. Kontrolu těchto pravidel označíme jako statickou validaci. Druhý typ validačních pravidel označíme jako validaci dynamickou.

5.5.1 Statická validace

Jak již bylo řečeno, statická validační pravidla jsou dána datovým modelem. Nejčastěji se jedná o kontrolu vyplnění povinné položky, délku textových řetězců, ...

Tyto pravidla jsou aplikována automaticky, protože je možné je načíst z databázového modelu. Ve vývojovém prostředí mohou být tyto pravidla také měněna – změna má smysl pouze v případě přidávání pravidel. Odebírat pravidla daná datovým modelem aplikace povede k nekompatibilitě s datovým modelem v databázi a chybám ve vygenerované aplikaci.

Definici statických validačních pravidel je možné při generování umístit přímo do jednotlivých tříd modelu. Při ukládání hodnot do databáze proběhne kontrola pravidel. Pro kontrolu pravidel bude použita validační služba z frameworku Symfony [28]. Tato služba podporuje několik způsobů definice validačních pravidel a lze ji snadno navázat na události Doctrine ORM. V Nette frameworku jsou také k dispozici metody pro validaci, ty jsou ale příliš svázány s prezentační logikou. Tato část frameworku má být sice v nejbližší době přepracována, ale aktuální implementace není pro použití k uvedeným účelům vhodná.

Ve třídách entit budou tedy definována pravidla pro každý atribut. Před uložením entity do databáze volá Doctrine ORM událost `onFlush`. K této události je jako posluchač registrována třída `ValidationSubscriber` – jedná se o návrhový vzor Pozorovatel (Observer), viz [6]. Třída `ValidationSubscriber` žádá validační službu o validaci každé změněné entity. Pokud nejsou validační pravidla splněna, dojde k vyhození výjimky, kterou potom zpracuje prezentační část. Validace je navržena tak, aby vždy proběhla kontrola všech pravidel a byly vždy vypsány všechny chyby najednou.

5.5.2 Dynamická validace

Dynamická validace obsahuje pravidla platná v různých stavech životního cyklu entity. Pro validaci těchto pravidel bude také využita validační služba, jako v případě statických pravidel. Definice pravidel pro validaci však bude probíhat dynamicky na základě stavu entity.

Databázové tabulky

V tab. 5.4 jsou uvedeny tabulka pro uložení dynamických validačních pravidel. Validační funkce jsou uloženy v tabulce `validity_function`, jejich parametry pak v tabulce `validity_function_parameter`. Mapování atributů na parametry funkce je uloženo v tabulce `dbm_attribute_parameter_mapping`. Kombinace validační funkce

Tab. 5.4: Databázové tabulky – Dynamická validace

Název tabulky	**Popis**
validity_function	Validační funkce
validity_function_language	Validační funkce – jazyková tabulka
validity_function_parameter	Parametry validační funkce
dbm_attribute_parameter_mapping	Mapování parametrů
dbm_attribute_validity_function	Validační pravidla
dbm_state_attribute_validity_function	Validačních pravidla podle stavu
dbm_attribute_validity_function_language	Validační pravidlo – jazyková tabulka

a namapovaných parametrů definuje validační pravidlo. Pro jejich uložení je navržena tabulka `dbm_attribute_validity_function`. Validační pravidla jsou potom přiřazena ke stavům, v kterých mají být kontrolována. Navíc je možné zvolit, zda nesplnění pravidla bude označeno jako chyba a zabrání uložení dat do databáze, nebo jestli bude vypsáno pouze jako varování. ER diagram je zařazen jako příloha D.4.

5.6 Vzorové třídy

Mezi plánovanými vlastnostmi vývojového prostředí byla uvedena dědičnost od vzorových tříd (viz 2.2.7). Vzhledem k tomu, že v současném návrhu modelu musí být třídy entit odvozeny od společného předka, který implementuje metody společné všem třídám a je využíván při typových kontrolách, bylo ustoupeno od použití dědičnosti.

Metody ze vzorových tříd budou při generování vloženy do třídy entity, s použitím atributů podle mapování. Tak bude umožněno použití více vzorových tříd zároveň, což by při použití dědičnosti nebylo možné – PHP nepodporuje vícenásobnou dědičnost.

Databázové tabulky

Výpis tabulek je uveden v tab. 5.5. Jsou definovány tabulky pro vzorové třídy (`class_pattern`), jejich atributy (`class_pattern_attribute`) a metody (`class_pattern_method`). Mapování atributů vzorové třídy na atributy entity je uloženo zvlášť pro běžné atributy (`mapping_attribute`) a pro vypočítané atributy (`class_pattern_attribute_calculated`).

Pokud dvě třídy z nějakého důvodu kolidují a není možné je použít společně u jedné entity, jsou tyto třídy označeny jako vzájemně se vylučující a zaznamenány

Tab. 5.5: Databázové tabulky – Vzorové třídy

Název tabulky	**Popis**
class_pattern	Vzorové třídy
class_pattern_language	Vzorové třídy – jazyková tabulka
class_pattern_method	Metody vzorových tříd
class_pattern_exclusion	Vzájemně se vylučující třídy
class_pattern_attribute	Atributy vzorových tříd
class_pattern_attribute_calculated	Vypočítané atributy vzorových tříd
mapping_attribute	Mapování atributů
mapping_attribute_calculated	Mapování vypočítaných atributů
dbm_entity_using_class_pattern	Entity využívající vzorové třídy

v tabulce (`class_pattern_exclusion`). Tabulka `dbm_entity_using_class_pattern` udržuje informace o tom, jaké vzorové třídy entity využívají. ER diagram je zařazen jako příloha D.5.

5.7 Přístupová oprávnění k modelu

Entity nesoucí informace i datovém modelu byly oproti systémové části rozšířeny o vypočítané atributy. Definice zdroje tedy musí být o tento vztah také rozšířena. Dále je u některých entit doplněna vazba na projekt.

5.7.1 Databázové tabulky

Výpis databázových tabulek pro kontrolu přístupu k modelu je uveden v tab. 5.6. Jak již bylo uvedeno, oproti systémovým entitám jsou zde odděleně uvedeny vypočítané atributy a doplněna závislost na projektu. ER diagram je uveden jako příloha D.6.

5.8 Prezentační část

Aplikace je rozdělená na moduly, které mohou mít hierarchickou strukturu. Modul obsahuje několik presenterů. Presenter je třída prezentační vrstvy, která poskytuje grafické rozhraní pro práci s jednou částí modelu (s jednou entitou).

Tab. 5.6: Databázové tabulky – Přístupová oprávnění k modelu

Název tabulky	**Popis**
acl_role	Role
acl_privilege	Oprávnění
acl_restriction	Omezení
dbm_resource	Zdroje
dbm_acl	Pravidla
acl_acl_2_acl_restriction	Kombinace omezení
dbm_entity	Entity
dbm_table	Tabulky
dbm_attribute	Atributy
dbm_attribute_calculated	Vypočítané atributy
dbm_state	Stavy
dbm_state_transition	Přechody
dbm_state_transition_list	Oprávnění k přechodům

Dalším prvkem jsou komponenty. Jsou definovány dva základní typy komponent a další podtypy:

- formulář:
  - Query From,
  - Record List,
  - Insert Form,
  - View Form ,
- sestava:
  - tisková sestava,
  - exportní sestava.

Je možné definovat další podtypy. Např. exportní sestavy by mohli být dále rozděleny na export do CSV a export do XML, ... Při generování jsou komponenty převedeny do kódu, proto v systémové části nejsou v databázi uvedeny. Komponenty mohou mít různá nastavení, ty jsou závislá na typu formuláře.

Presentery obsahují metody – jedna metoda představuje jednu „obrazovku“ informačního systému. Každá metoda má přiřazenou komponentu, kterou zobrazuje. Jedna komponenta může být používána více metodami presenteru. Dále jsou definovány akce komponent – akcí je například odeslání formuláře.

5.8.1 Databázové tabulky

Seznam databázových tabulek je uveden v tab. 5.7. U tabulek `modul`, `presenter` a `component` je kromě jazykové tabulky ještě jedna jazykově závislá tabulka se sufixem `_text`. Ta je určena pro uložení textů, které se vztahují k danému objektu.

Jak již bylo uvedeno, při generování jsou komponenty převedeny na zdrojový kód, proto jsou zde oproti systémovým tabulkám navíc. Ostatní entity ze systémových tabulek vycházejí, jsou pouze doplněny o závislost na projektu. ER diagram je uveden jako příloha D.7.

Tab. 5.7: Databázové tabulky – Prezentační část

Název tabulky	**Popis**
modul	Moduly
modul_language	Moduly – jazyková tabulka
modul_text	Moduly – texty
presenter	Presentery
presenter_language	Presentery – jazyková tabulka
presenter_text	Presentery – texty
method	Metody presenteru
mpm	Kombinace Modul-Presenter-Metoda
component	Komponenty
component_language	Komponenty – jazyková tabulka
component_text	Komponenty – texty
component_attribute	Atributy, používané komponentou
component_attribute_calculated	Vypočítané atributy, používané komponentou
component_action	Akce komponent
component_action_language	Akce komponent – jazyková tabulka
component_action_text	Akce komponent – texty
component_type	Typy komponent
component_settings	Nastavení komponent
component_settings_value	Hodnoty nastavení

5.9 Přístupová oprávnění k prezentační části

Entity pro definici přístupových oprávnění vychází ze systémové části aplikace, popsané v kapitole 4.3.1. Kromě přidané závislosti na projektu nebyly entity nijak změněny.

Na ER diagramu (příloha D.8) je zobrazena tabulka `system_user`, zvýrazněná žlutě. Ta samozřejmě patří do systémové části. Zde je zobrazena pro zdůraznění rozdělení v systému – tabulky `project`, `database_connection` a další tabulky jsou vázány na konkrétního uživatele vývojového prostředí. Informace o tomto uživateli jsou v systémové části v tabulce `system_user`.

Při definici oprávnění pro novou aplikaci ve vývojovém prostředí jsou informace o uživatelích aplikace uvedeny v tabulce `user`. Jedná se tedy o budoucí uživatele generovaného systému. Při generování jsou pak informace o uživatelích, definovaných pro daný projekt, vygenerovány jako data pro tabulku `system_user`.

5.10 Navigace

Navigace koresponduje s entitami navigace v systémové části (4.8). Doplněna je zde pouze vazba na konkrétní projekt u položek menu, modulů a šablon. Závislost na projektu by sice bylo možné odvodit z tabulky mpm, ale pro snazší generování jsou uvedeny přímo. ER diagram je uveden jako příloha D.9.

5.11 Šablony

Entity pro uložení šablon jsou prakticky stejné jako v systémové části aplikace, popsané v kapitole 4.7. Jsou zde 2 důležité rozdíly:

- závislost na projektu – šablony patří vždy ke konkrétnímu projektu. Proto má tabulka `template` cizí klíč, který tento vztah reprezentuje.
- jazykové tabulky – texty v šabloně a blocích jsou jazykově závislé. Při generování se počítá s uložením těchto dat v lokalizačních souborech. Proto tyto tabulky v systémové části databáze nejsou.

ER diagram je uveden jako příloha D.10.

5.12 Číselniky

U entit pro číselníky je situace podobná. Datový model je prakticky stejný jako v systémové části aplikace, popsané v kapitole 4.6. Protože číselníky se během provozu aplikace mění, jsou jazykové položky i v systémových číselnících. Jediným rozdílem je tedy závislost na projektu. ER diagram je uveden jako příloha D.11.

5.13 Kompletní seznam tabulek vývojového prostředí

Kompletní seznam tabulek vývojového prostředí je uveden jako příloha E. V příloze jsou také ER diagramy se skupinami entit, navržených pro konkrétní části systému. Kompletní ER diagram není uveden z důvodu velkého množství tabulek a vztahů (tabulek navržených pro vývojové prostředí je téměř 100).

5.14 Konvence očekávané u datového modelu

Aby bylo možné správně identifikovat typy atributů, tabulek, je třeba stanovit určité konvence, které boudou při načítání modelu očekávány. Dodržováním těchto konvencí vlastně vnášíme do datového modelu další informace, které potom nebude nutné zadávat ručně. Konvence je možné rozdělit na povinné a doporučené.

5.14.1 Povinné konvence

Pro správnou funkčnost vygenerované aplikace je nutné tyto konvence dodržovat. Snahou bylo co nejvíce počet povinných konvencí snížit. Vývojové prostředí vyžaduje dodržení těchto tří pravidel:

- každá tabulka, která není vazební musí obsahovat jednoduchý primární klíč,
- vazební tabulka (pro vztah M:N) může používat složený primární klíč pouze v případě, že kromě tohoto složeného klíče neobsahuje žádné jiné atributy,
- jednoduchý primární klíč musí být vždy pojmenován `id`.

Splnění těchto požadavků by nemělo být problematické. Definice jednoduchého primárního klíče patří k pravidlům dobrého návrhu databáze. Jeho jednotné pojmenování je běžně používané.

Vyžadování jednoduchého primárního klíče u vazebních tabulek, pokud obsahují další atributy, je dáno použitím tabulky v kontextu doctrine modelu. Pro vazební tabulku nebude generována vlastní třída, bude uvedena pouze jako pravidlo pro načtení kolekce. Pokud tabulka obsahuje další datové atributy, je třeba ji definovat jako běžnou datovou tabulku, pro kterou bude generována třída .

5.14.2 Doporučené konvence

Pojmenování tabulek

Z názvů tabulek jsou odvozovány jejich typy a názvy entit a tříd. Jejich nedodržení sice nezpůsobí nefunkčnost systému, ale znemožní získat z modelu některé další informace, které potom bude nutné zadávat manuálně.

- vazební tabulka – názvy tabulek odděleny `_2_`, nebo `_has_`, např. `dbm_acl_2_acl_restriction`,
- jazyková tabulka – název odpovídá názvu příslušné tabulky, je doplněn sufixem `_language`, nebo `_jazyk` např. `modul_language`,
- všechny tabulky – názvy psány malými písmeny, v jednotném čísle, jednotlivá slova oddělena podtržítkem.

Pojmenování atributů

Z názvu atributů je detekován pouze atribut nesoucí informaci o stavu, protože ten není možné rozlišit jiným způsobem. Dále jsou názvy použity k odvození názvů atributů a metod doctrine entit. Jejich hodnoty lze sice manuálně upravit, ale přináší to zbytečnou práci navíc.

- všechny atributy – názvy psány malými písmeny, jednotlivá slova oddělena podtržítkem,
- atributy reprezentující stav – název `state`, nebo `stav`,
- cizí klíče – název tabulky v jednotném čísle, doplněný prefixem, `id_`, nebo sufixem `_id`,
- atributy - cizí klíče směřující do jedné tabulky – název jako u běžných cizích klíčů, doplněný o část označující význam. Umístění této části nehraje roli, např. `id_codebook_master, id_codebook_slave`, nebo `id_start_state, id_target_state`.

5.15 Načítání datového modelu projektu

Jedním z požadavků na vývojové prostředí je automatické načítání datového modelu nezávislé na konkrétní databázi. Toho bude dosaženo rozdělením této funkčnosti mezi několik tříd.

Bylo definováno rozhraní interface `IDbSchemaLoader`, které deklaruje metody pro načtení tabulek, atributů, indexů a vztahů.

Dále je definována třída `DbModelBuilder`. Ta očekává jako parametr konstruktoru objekt implementující zmíněné rozhraní `IDbSchemaLoader`.

Třída `DbModelBuilder` dokáže zpracovat informace o modelu, identifikovat entity, typy sloupců a tabulek podle dříve uvedených konvencí a tyto informace uložit do databáze. O to z jakého zdroje a jak jsou informace o datovém modelu získány se nestará, využívá pouze metod deklarovaných rozhraním.

Odpovědnost za načítání datového modelu mají jednotlivé loader třídy. Pro základní implementaci bude postačující třída `MysqLoader`, které bude načítat data z databáze MySQL pomocí schématu `information_schema`.

Implementovat v budoucnu další třídy pro další databáze, nebo zdroje (např. sql soubor) nebude problém. Stačí, aby třída implementovala požadované rozhraní.

5.16 Generátor

Generátor je rozdělen do několika částí – generátor modelu, generátor prezentační části a generátor SQL kódu.

5.16.1 Generátor modelu

Výstupem generátoru bude kompletní kód a konfigurace modelové části aplikace, tedy:

- adresářová struktura modelu,
- třídy služeb pro entity,
- třídy hlavních a vedlejších entit,
- konfigurace služeb (konfigurační soubor ve formátu neon),
- třída `ProjectInfo`, která poskytuje informace o projektu.

5.16.2 Generátor prezentační vrstvy

Výstupem generátoru bude kompletní kód a konfigurace prezentační části aplikace, tedy:

- adresářová struktura modulů,
- třídy presenterů,
- třídy jednotlivých formulářů,
- šablony

5.16.3 Generátor SQL kódu

SQL kód bude rozdělen do dvou souborů. První bude obsahovat zakládací skript pro všechny systémové tabulky, druhý soubor bude obsahovat data.

6 IMPLEMENTACE

Z výčtu navržených vlastností vývojového prostředí i generované aplikace je patrné, že se jedná o rozsáhlé systémy. Jejich kompletní implementace přesahuje možnosti této práce. Proto byly vybrány základní části, jejichž implementací bude ověřena správnost navržených konceptů.

6.1 Výběr základních částí pro implementaci

Základní části byly vybrány tak, aby byl v rámci této práce vytvořen funkční prototyp vývojového prostředí, který bude využívat koncepty navržené v kapitole 4. Tím bude ověřeno, že uvedené koncepty jsou v praxi opravdu použitelné.

Ze systémové části je nutné implementovat:

- třídy podporující modelovou část aplikace
- přístupová oprávnění k modelu,
- jazykové položky,
- číselníky,
- formuláře řízené dle oprávnění.

V rámci vývojového prostředí byly k implementaci vybrány tyto části:

- základní funkce správy projektů,
- načítání datového modelu a jeho správa,
- správa rolí,
- správa prezentační části aplikace,
- generátor.

Prototyp vývojového prostředí se bude orientovat hlavně na funkční části. V rámci jeho implementace proto není třeba realizovat grafická rozhraní pro všechny požadované části, hlavní důraz bude kladen na funkční část.

6.2 Dosažené výsledky

Jako první byly implementovány třídy, které podporují systémové funkce. Poté bylo realizováno načítání informací o datovém modelu z MySQL databáze a generování modelových tříd. Díky tomu bylo možné vygenerovat modelovou vývojového prostředí. Generovaný model odpovídá návrhu a definuje pravidla pro statické validace. Byl použit jako základ modelové části a dále byl rozšířen o potřebné funkce.

Při generování modelu jsou generovány zároveň přístupová oprávnění k němu. Protože u prototypu nebylo počítáno s implementací grafického rozhraní pro správu

oprávnění k modelu, jsou automatiky vygenerována plná oprávnění ke všem položkám pro všechny role. Pokud má být přístup pro některé role omezený, stačí smazat příslušná pravidla přímo v databázi.

Bylo implementováno grafické rozhraní pro správu projektu, rolí a pro správu datového modelu. Formuláře jsou definovány pomocí konfiguračních polí, jak je navrženo v kapitole 4.10 a zobrazení položek reflektuje oprávnění k modelu. To je vidět u formulář datového modelu, kde položky pevně dané databází není možné editovat. Je zde využit jednoduchý životní cyklus entit a číselníky.

V rámci ladění generátoru prezentační části byly vygenerovány formuláře pro práci s číselníky. Všechny generátory používají šablonovací systém, takže případná úprava generovaného kódu je snadno realizovatelná.

Podle definovaného návrhu byly implementovány všechny požadované části prototypu vývojového prostředí. Jednotlivé části jsou přímo ve vývojovém prostředí použity a tím byla ověřena jejich funkčnost.

6.3 Životní cykly entit vývojového prostředí

Při návrhu byly definovány životní cykly pro vybrané entity. Životní cykly však nebyly zařazeny do výběru základních vlastností pro implementaci, ale v rámci dalšího vývoje je s nimi počítáno. Proto byly zařazeny jako příloha F.

V práci není uveden procesní model, který zobrazuje procesy právě ve vztahu k životnímu cyklu a rolím. Je to proto, že v prototypu je použita pouze jedna role a entity mají jenom jednoduché životní cykly, procesní model by tedy nepřinesl žádnou přidanou hodnotu.

7 POUŽITÉ NÁSTROJE

7.1 NetBeans IDE

Vývojové prostředí NetBeans IDE [29] je nástroj, pomocí kterého programátoři mohou psát, překládat, ladit a distribuovat aplikace. Samotné vývojové prostředí je vytvářeno v jazyce Java – ovšem podporuje prakticky jakýkoliv programovací jazyk. Existuje rovněž velké množství modulů, které toto vývojové prostředí rozšiřují. Vývojové prostředí NetBeans je bezplatně šířený produkt a jeho užívání není nijak omezeno. Je vyvíjeno pod licencí Open Source a je možné je bezplatně používat v komerčním i nekomerčním prostředí.

7.2 Mysql Workbeanch

Je grafický nástroj pro správu databáze, dostupný na [30]. Poskytuje funkce pro:

- Datové modelování – umožňuje vytvářet databázové tabulky, vztahy, sloupce, definovat indexy a triggery. Lze využít také synchronizaci s databází, nebo export do SQL kódu.
- SQL editor – umožňuje připojení k databázi, zjednodušuje vytváření SQL dotazů a editaci dat v databázi.

Aplikace je dostupná ve dvou verzích – „Community Edition“ a „Standard Edition“. První uvedená je k dispozici zdarma.

7.3 Git

Git [31] je distribuovaný systém správy verzí vytvořený Linusem Torvaldsem, původně pro vývoj jádra Linuxu. Původně se mělo jednat o nízkoúrovňový základ pro vývoj různých systémů správy verzí, ale časem se Git vyvinul do samostatně použitelného systému správy verzí. Dnes je používán mnoha známými projekty. Je šířen pod GPL verze 2, jedná se o svobodný software.

8 ZÁVĚR

Práce se zabývá návrhem vývojového prostředí pro generování client/server databázových aplikací.

První kapitola definuje vlastnosti dobře navrženého databázového systému a hledá nedostatky u současných aplikací, které by mohly být odstraněny návrhem vhodných systémových řešení. Druhá kapitola popisuje vlastnosti vývojového prostředí. Ve třetí kapitole je popsán výběr technologií, frameworku a knihoven. Stěžejní části práce jsou popsány ve čtvrté a páté kapitole.

Čtvrtá kapitola analyzuje požadavky na databázový systém, navrhuje koncepty pro jejich řešení a podrobně je rozebírá. Je popsána architektura aplikace, architektura modelové části. Řízení přístupových oprávnění je rozděleno na dvě části – zvlášť pro prezentační a modelovou část aplikace. Pro prezentační část je použit klasický koncept rolí a zdrojů. U modelové části je řízení přístupu taktéž založeno na rolích, zdroje jsou však definovány na několika úrovních a zohledňují životní cyklus entity. Také je definován koncept omezení, který rozšiřuje možnosti řízení přístupu a umožňuje splnit libovolně složitý požadavek. V prezentační části je navržen konfigurační formát pro jednodušší definici formulářů. Je navržen způsob řízení přístupu k formulářovým prvkům na základě oprávnění k modelu.

Pátá kapitola rozebírá vlastnosti vývojového prostředí, uvádí postup činností ve vývojovém prostředí, popisuje jednotlivé moduly a definuje datový model pro uložení dat o jednotlivých projektech. Definuje povinné a doporučené konvence, který usnadňuje autodetekci při načítání datového modelu.

V šesté kapitole jsou vybrány základní části vývojového prostředí, které jsou implementovány. Nejprve byly naprogramovány systémové části, nezbytné pro fungování prototypu vývojového prostředí. Při implementaci základních částí byl zvolen takový postup, který umožňoval použít již hotové části vývojového prostředí pro generování dalších částí. Modelová část vývojového prostředí je tedy generovaná, do tříd je pouze doplněna další funkčnost. Stejně tak je tomu u některých tříd prezentační vrstvy a řízení přístupu k modelu. Tím byla zároveň ověřena funkčnost navržených konceptů i samotného vývojového prostředí.

V práci bylo ověřeno, že zvolená architektura a navržené principy odpovídají stanoveným požadavkům a je možné na nich dále stavět. Při realizaci práce bylo dosaženo stanovených cílů.

LITERATURA

[1] Kabir, M. J. *APACHE SERVER 2 kompletní příručka administrátora.* Vydání první. Brno : Computer Press, 2004. 722 s. ISBN: 80-251-0319-6.

[2] HERNANDEZ, Michael J. *Návrh databází.* Přeložil Jan Bouda. 1. vyd. Praha : Grada Publishing a.s., 2006. 408 s. ISBN 80-247-0900-7.

[3] VRÁNA, Jakub. *Adminer.cz : Adminer Pro - Generátor administračního rozhraní.* [online]. 2009 [cit. 2010-04-24]. Dostupné z URL: <http://www.adminer.org/pro/>.

[4] KOFLER, Michael. *Mistrovství v MySQL 5 : Kompletní průvodce webového vývojáře.* Vydání první. Brno : Computer Press, 2007. 805 s. ISBN 978-80-251-1502-2.

[5] GUTMANS Andi;BAKKEN Stig Saether;RETHANS Dereck. *Mistrovství v PHP 5.* 1.vyd. Brno: Computer Press, a.s.,2005. 520 s. ISBN 80-251-0799-X.

[6] PECINOVSKÝ Rudolf. *Návrhové vzory - 33 vzorových postupů pro objektové modelování.* Vydání první. Brno: Computer Press, a.s.,2007. 520 s. ISBN 978-80-251-1582-4.

[7] The PHP Group. *PHP: Hypertext Preprocessor* [online]. 2011 [cit. 2011-11-04]. Dostupné z URL: <http://php.net>.

[8] HERNANDEZ, Michael J.; VIESCAS, John L. *Myslíme v jazyku SQL : Tvorba dotazů.* Vydání první. Praha : Grada Publishing a.s., 2004. 380 s. ISBN 80-247-0899-X.

[9] DANĚK, Petr. *Velký test PHP frameworků. Root.cz* [online]. 11. 9. 2008, 3, 4, [cit. 2010-11-04]. Dostupné z URL: <http://www.root.cz/serialy/velky-test-php-frameworku>.

[10] Nette Foundation. *Model-View-Presenter (MVP).* [online]. 2011 [cit. 2011-05-04]. Dostupné z URL: <http://doc.nette.org/cs/model-view-presenter>.

[11] BERNARD Borek. *Prezentační vzory z rodiny MVC.* [online]. 2009-05-07 [cit. 2011-05-04]. Dostupné z URL: <http://zdrojak.root.cz/clanky/prezentacni-vzory-zrodiny-mvc>.

[12] Nette Foundation. *Nette Framework.* [online]. 2010 [cit. 2010-11-04]. Dostupné z URL: <http://nette.org>.

[13] Zend Technologies Ltd. *Zend Framework.* [online]. 2010 [cit. 2010-11-04]. Dostupné z URL: <http://framework.zend.com>.

[14] PRADO Group. *PRADO PHP Framework.* [online]. 2010 [cit. 2010-11-04]. Dostupné z URL: <http://www.pradosoft.com>.

[15] LEMPERA, M. *Databázový systém pro elektronický obchod.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií 2009. 63 s., 5 s. příloh. Vedoucí bakalářské práce Ing. Tomáš Macho, Ph.D.

[16] Nette Foundation. *Latte filter.* [online]. 2010 [cit. 2010-11-04]. Dostupné z URL: <http://doc.nette.org/cs/sablony/latte-filter>.

[17] The jQuery Project. *JQuery is a new kind of JavaScript Library : write less, do more.* [online]. 2010 [cit. 2010-11-04]. Dostupné z URL: <http://jquery.com>.

[18] Nette Foundation. *DataGrid - Doplňky, pluginy a komponenty.* [online]. 2010 [cit. 2010-11-04]. Dostupné z URL: <http://addons.nette.org/cs/datagrid>.

[19] SKLENÁŘ Roman. *DataGrid - Doplňky, pluginy a komponenty.* [online]. 2011 [cit. 2011-05-04]. Dostupné z URL: <https://github.com/romansklenar/nette-datagrid/tree/dev>.

https://github.com/romansklenar/nette-datagrid/tree/dev

[20] Nette Foundation. *Dibi : tiny 'n' smart database layer.* [online]. c2008-2009 , [cit. 2010-12-19]. Dostupné z URL: <http://dibiphp.com/cs>.

[21] VRÁNA, Jakub. *NotORM.* [online]. 2010 [cit. 2010-12-19]. Dostupné z URL: <http://www.notorm.com>.

[22] Doctrine Project. *Doctrine - PHP Object Persistence Libraries and More.* [online]. 2010 [cit. 2010-12-19]. Dostupné z URL: <http://www.doctrine-project.org>.

[23] TICHÝ, Jan. *Seriál Doctrine 2.* [online]. 2011 [cit. 2011-01-19]. Dostupné z URL: <http://zdrojak.root.cz/serialy/doctrine-2>.

[24] MAREK, Jan. *Informační portál města Kladna.* [online]. Fakulta elektrotechnická , 2011. 50 s. Bakalářská práce. České vysoké učení technické v Praze. Dostupné z URL: <https://dip.felk.cvut.cz/browse/pdfcache/marekj16_2011bach.pdf>.

[25] Nette Foundation. *Nette\Security\Permission.* [online]. 2011 [cit. 2011-05-04]. Dostupné z URL: <http://doc.nette.org/cs/nette-security-permission>.

[26] Nette Foundation. *NEON sandbox.* [online]. 2011 [cit. 2011-05-05]. Dostupné z URL: <http://ne-on.org>.

[27] MAREK, Jan;TVRDÍK Jan. *FormMacros | Addony pro Nette Framework.* [online]. 2011 [cit. 2011-05-05]. Dostupné z URL: <http://nette.merxes.cz/form-macros/>.

[28] Extreme Sensio. *Validation.* [online]. 2010 [cit. 2011-05-10]. Dostupné z URL: <http://symfony.com/doc/2.0/book/validation.html>.

[29] Oracle Corporation. *Netbeans IDE.* [online]. 2011 [cit. 2011-05-10]. Dostupné z URL: <http://netbeans.org>.

[30] Oracle Corporation. *MySQL Workbench.* [online]. 2010 [cit. 2011-05-10]. Dostupné z URL: <http://dev.mysql.com/downloads/workbench>.

[31] CHACON Scott. *Git - the fast version control system.* [online]. 2010 [cit. 2011-05-10]. Dostupné z URL: <http://git-scm.com>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

ACL Access control list – seznam oprávnění, určující kdo nebo co má povolení přistupovat k objektu a jaké operace s ním může provádět

AJAX Asynchronous JavaScript and XML – obecné označení pro technologie vývoje interaktivních webových aplikací, které mění obsah svých stránek bez nutnosti jejich znovu-načítání

CSS Cascading Style Sheets – jazyk pro popis způsobu zobrazení stránek napsaných v jazycích HTML, XHTML nebo XML

DOM Document Object Model – objektově orientovaná reprezentace XML nebo HTML dokumentu, umožňuje přistupovat k dokumentu jako ke stromu

ERD, ER diagram Entity Relationship Diagram – metoda datového modelování používaná pro abstraktní a konceptuální znázornění dat

IF Insert Form – formulář pro vložení záznamu

JS JavaScript – multiplatformní, objektově orientovaný skriptovací jazyk, používá se pro WWW stránky, jsou jím obvykle ovládány různé interaktivní prvky GUI , nebo tvořeny animace a efekty obrázků

JSON JavaScript Object Notation – JavaScriptový zápis objektů – formát pro výměnu dat

LOV List Of Value – formulářový prvek, který umožňuje vybrat hodnotu ze seznamu

MVC Model view controller – softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent

MVP Model view presenter – softwarová architektura, která rozděluje datový model aplikace, uživatelské rozhraní a řídicí logiku do tří nezávislých komponent

OOP Objektově orientované programování – metodika vývoje softwaru, založená na objektech jako prvcích modelované reality, jejich atributech a vzájemných vztazích

ORM Object-relational mapping – programovací technika, mapující objekty a relace v databázi na třídy a asociace

PHP	PHP: Hypertext Preprocessor – skriptovací programovací jazyk, určený především pro programování dynamických internetových stránek
QF	Query Form – formulář pro filtrování záznamů
RL	Record List – formulář zobrazující seznam záznamů
SQL	Structured Query Language – standardizovaný dotazovací jazyk používaný pro práci s daty v relačních databázích
VF	View Form – formulář pro zobrazení a editaci záznamu
HTML	HyperText Markup Language – hypertextový jazyk pro tvorbu webových stránek
WWW	World Wide Web - systém provázaných hypertextových dokumentů na internetu

SEZNAM PŘÍLOH

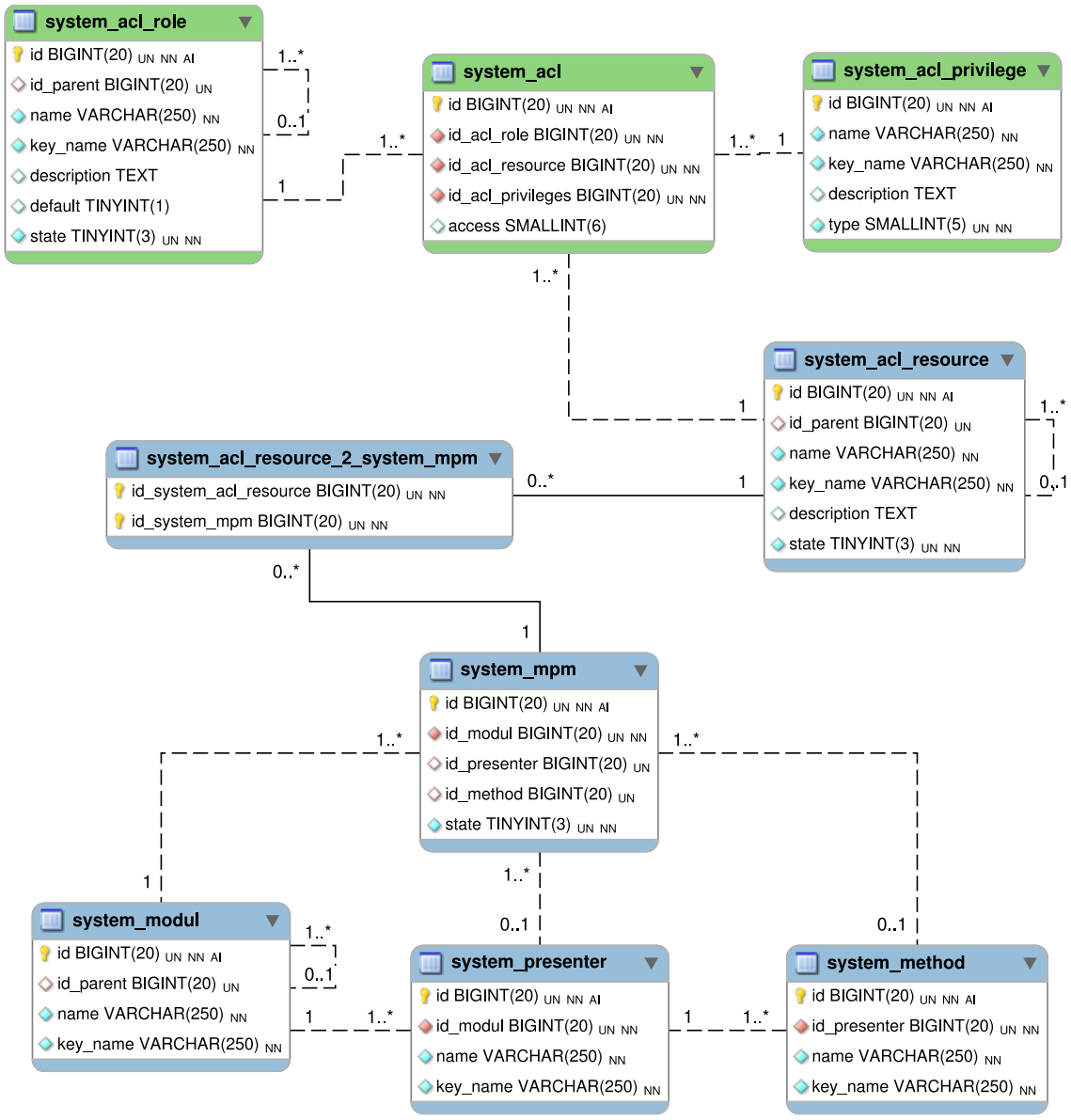
A OBSAH PŘILOŽENÉHO CD

- adresář `DP` – obsahuje elektronickou verzi diplomové práce
- adresář `PHP` – obsahuje zdrojové kódy vývojového prostředí
  - adresář `_generated` – soubory vygenerované vývojovým prostředím
  - ostatní adresáře – dle struktury popsané v práci
- adresář `SQL`
  - `admingenerator.sql` – SQL soubor pro založení databáze vývojového prostředí
  - adresář `models` – obsahuje datový model vytvořený v aplikaci MySQL Workbench

B ER DIAGRAMY – SYSTÉMOVÁ ČÁST

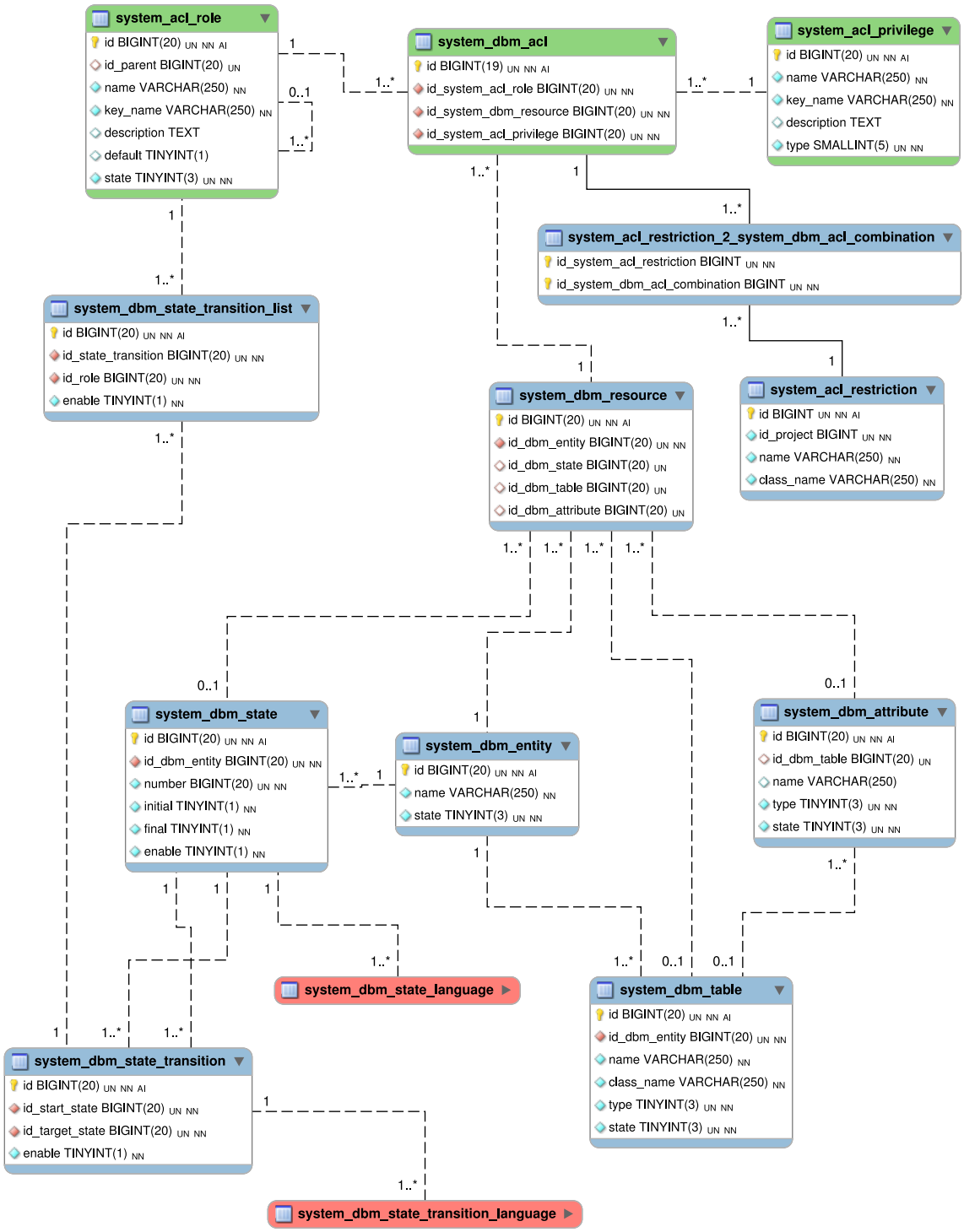
B.1 Přístupová oprávnění

B.1.1 Oprávnění v prezentační části



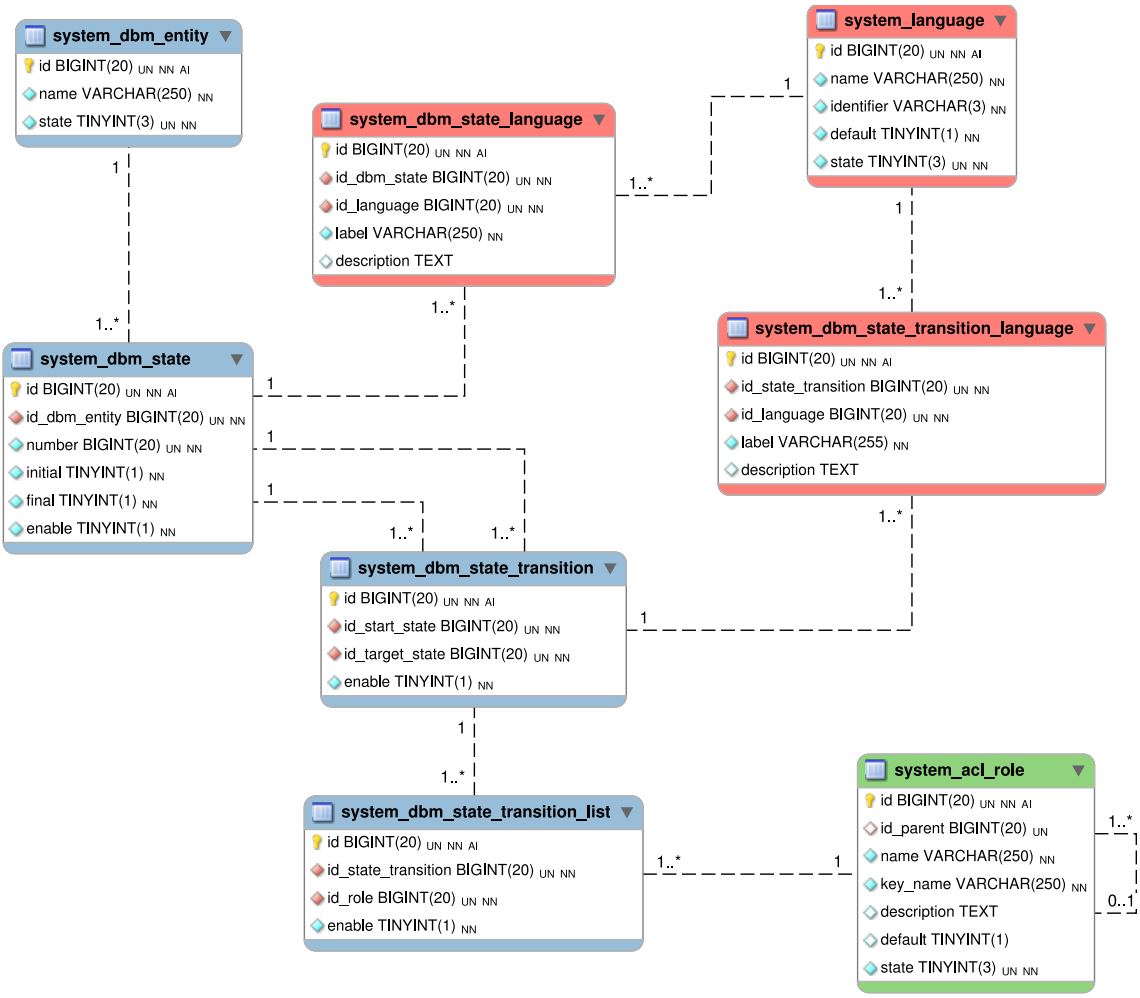
Obr. B.1: ER diagram – Oprávnění v prezentační části.

B.1.2 Oprávnění v modelu



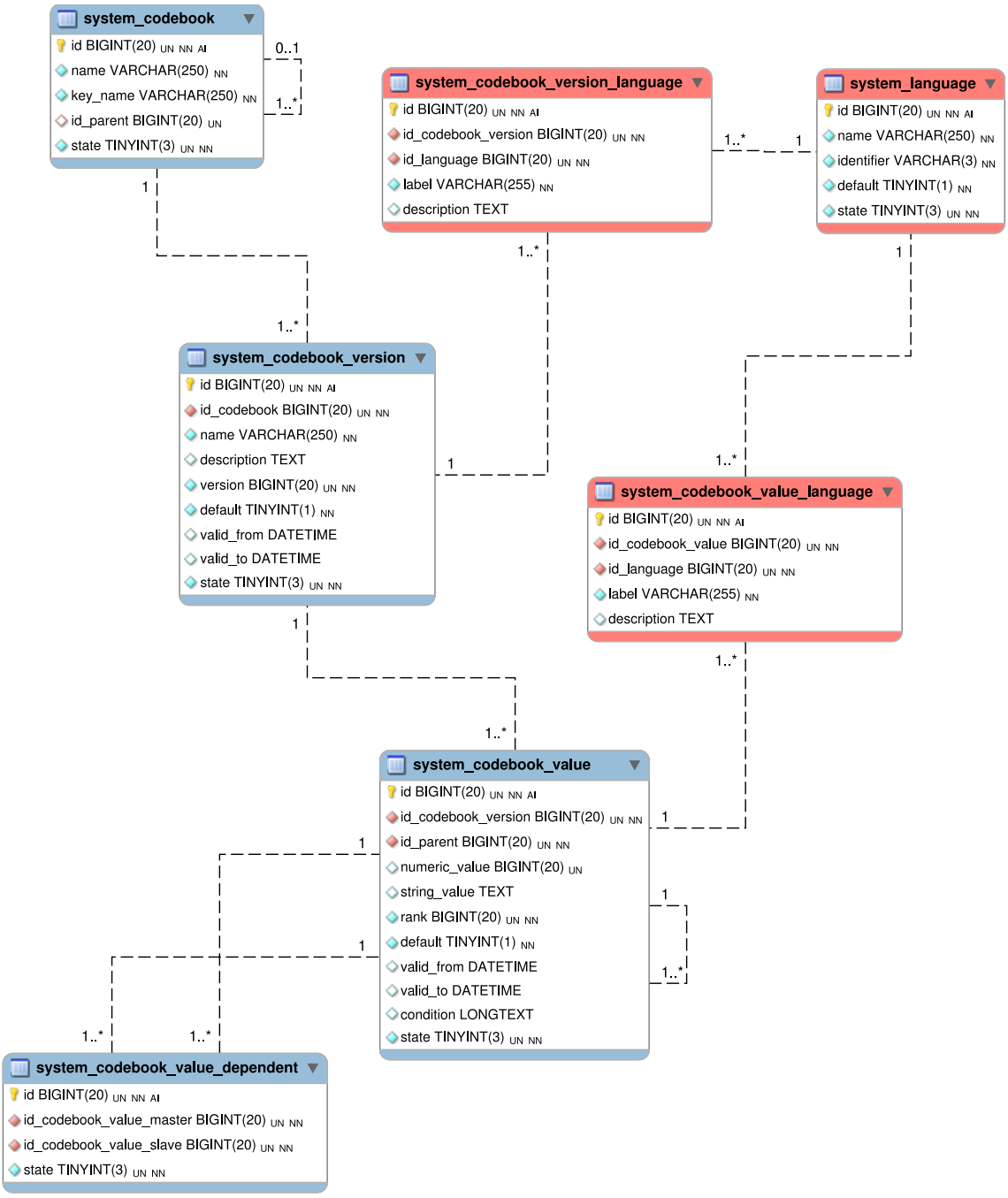
Obr. B.2: ER diagram – Oprávnění v modelu.

B.2 Životní cykly



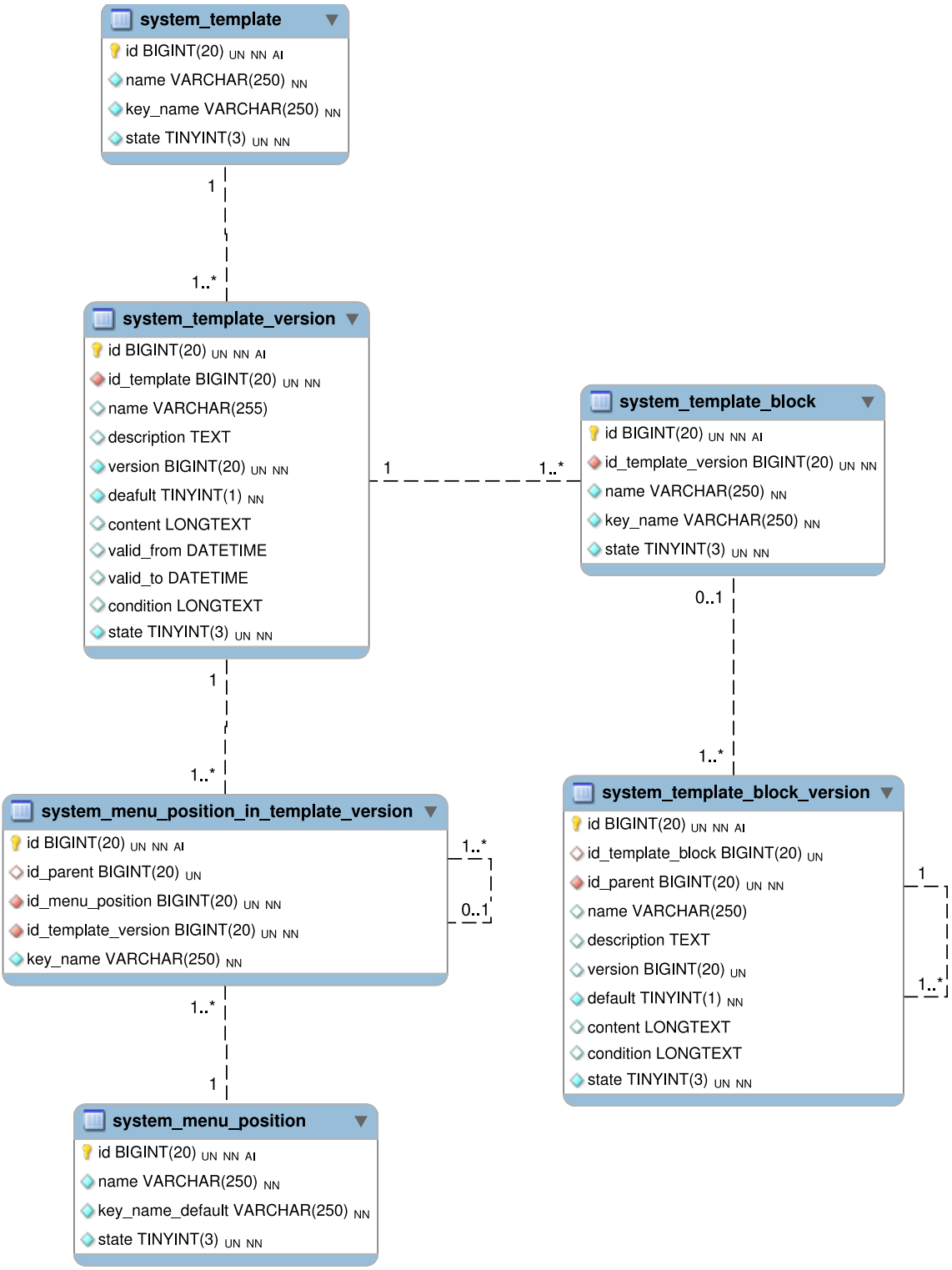
Obr. B.3: ER diagram – Životní cykly.

B.3 Číselníky



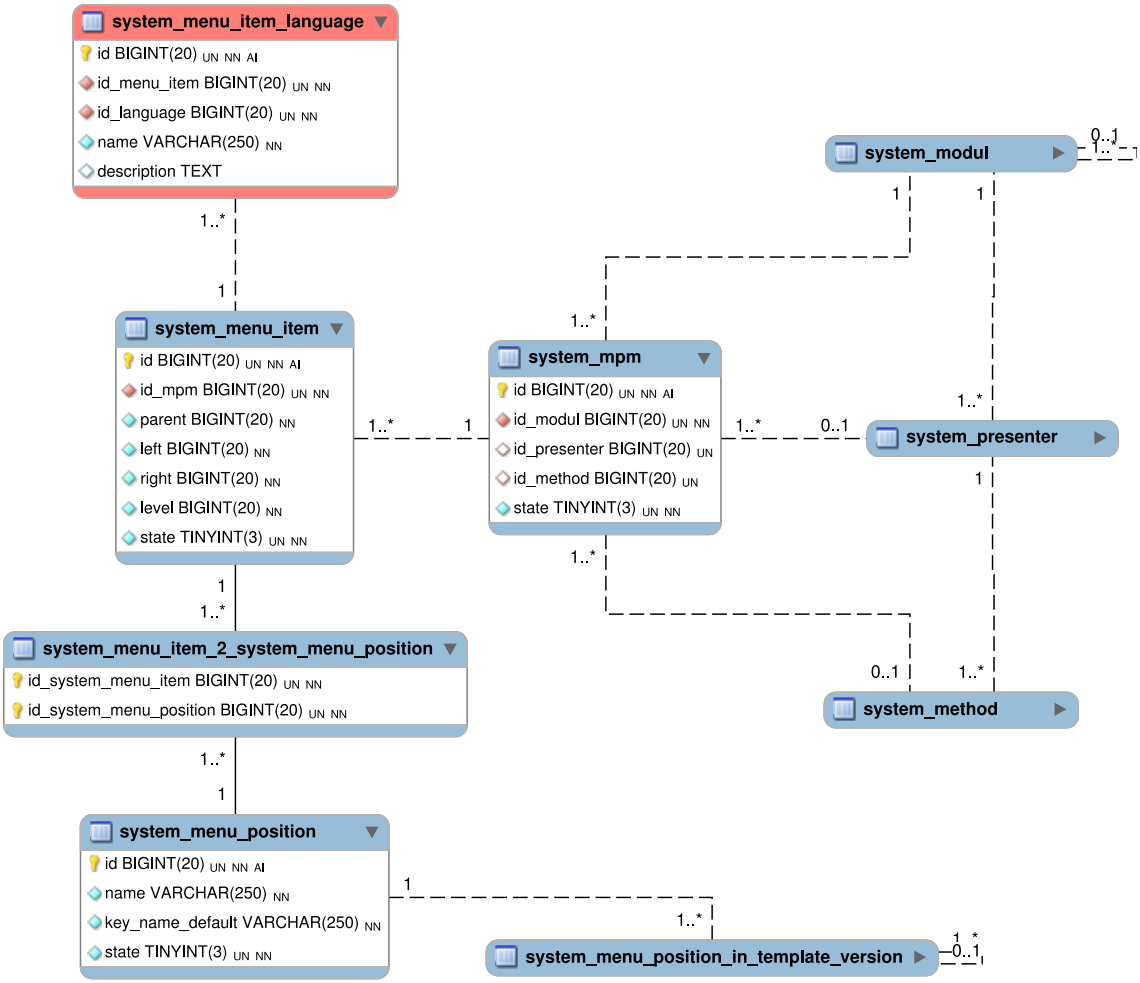
Obr. B.4: ER diagram – Číselníky.

B.4 Šablony



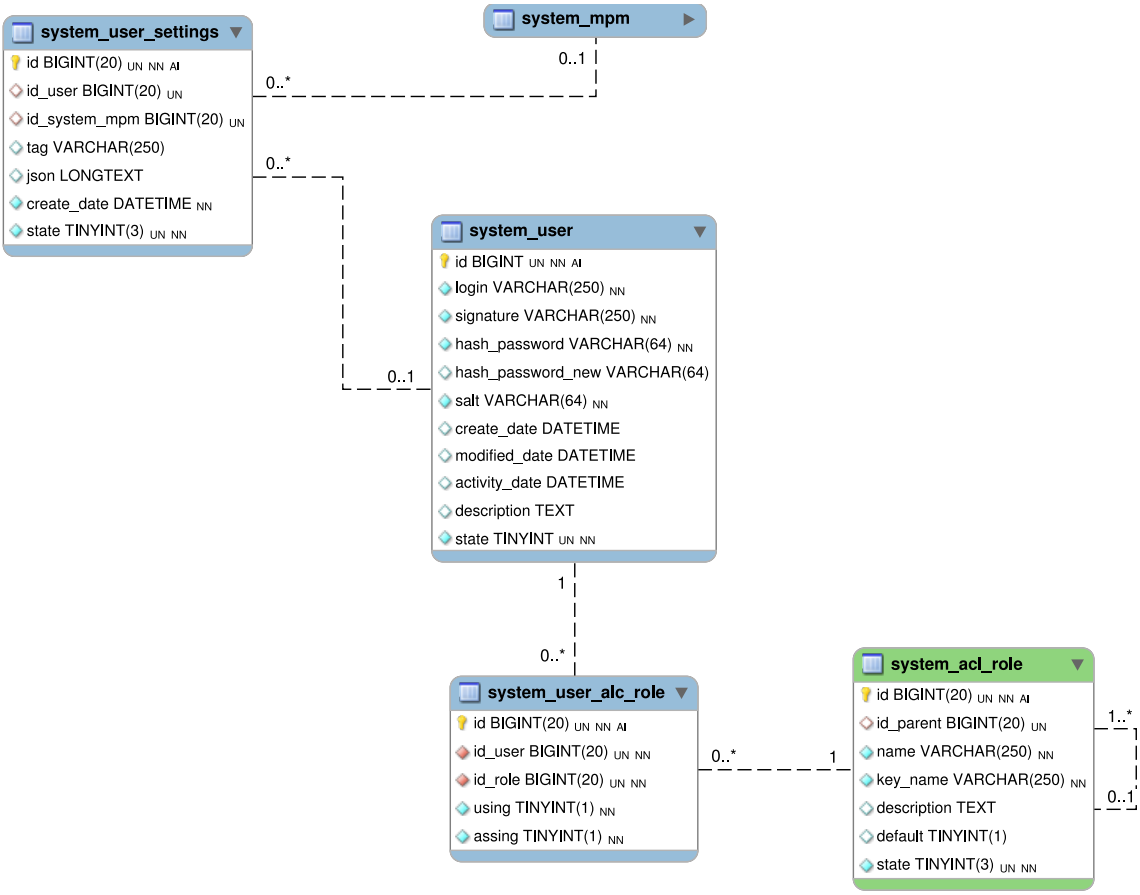
Obr. B.5: ER diagram – Šablony.

B.5 Navigace



Obr. B.6: ER diagram – Navigace.

B.6 Uživatel, role a uživatelské nastavení



Obr. B.7: ER diagram – Uživatel, přidělení rolí a uživatelské nastavení.

B.7 Systémové tabulky

Tato příloha je vložena volně.

C KOMPLETNÍ SEZNAM TABULEK SYSTÉMOVÉ ČÁSTI APLIKACE

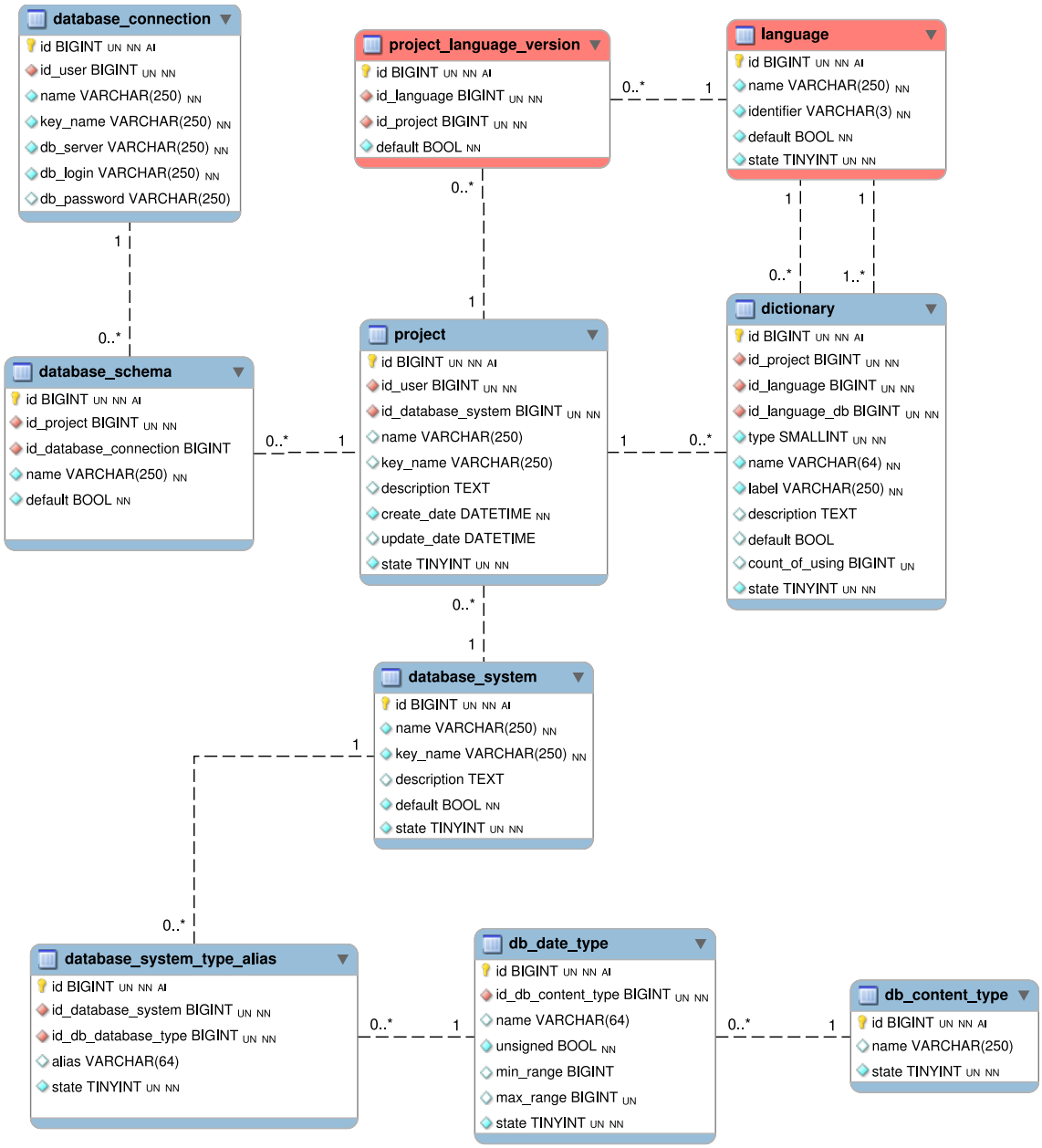
Tab. C.1: Kompletní seznam tabulek systémové části aplikace.

Název tabulky	**Popis**
system_acl	Přístupová pravidla (prezentační část aplikace)
system_acl_privilege	Oprávnění
system_acl_resource	Zdroje (prezentační část aplikace)
system_acl_resource_2_system_mpm	Vazební tabulka Zdroje : Model-Presenter-Metoda
system_acl_restriction	Omezení (modelová část aplikace)
system_acl_restriction_2_system_dbm_acl	Vazební tabulka Pravidla : Omezení
system_acl_role	Role
system_codebook	Číselníky
system_codebook_value	Hodnoty číselníku
system_codebook_value_dependent	Závislé hodnoty
system_codebook_value_language	Hodnoty číselníku – jazyková tabulka
system_codebook_version	Verze číselníku
system_codebook_version_language	Verze číselníku – jazyková tabulka
system_dbm_acl	Přístupová pravidla (modelová část aplikace)
system_dbm_attribute	Atributy
system_dbm_entity	Entity
system_dbm_resource	Zdroje (modelová část aplikace)
system_dbm_state	Stavy
system_dbm_state_language	Stavy – jazyková tabulka
system_dbm_state_transition	Přechody mezi stavy
system_dbm_state_transition_language	Přechody mezi stavy – jazyková tabulka
system_dbm_state_transition_list	Oprávnění pro přechody mezi stavy
system_dbm_table	Tabulky
system_language	Tabulka jazyků

Název tabulky	Popis
system_menu_item	Položky v menu
system_menu_item_2_system_menu_position	Vazební tabulka Položky v menu : Menu pozice v šabloně
system_menu_item_language	Položky v menu – jazyková tabulka
system_menu_position	Menu pozice v šabloně
system_menu_position_in_template_version	Menu pozice v konkrétní verzi šablony
system_method	Metody presenterů
system_modul	Moduly
system_mpm	kombinace Modul-Presenter-Metoda
system_presenter	Presentery
system_template	Šablony
system_template_block	Bloky v šabloně
system_template_block_version	Verze bloků v šabloně
system_template_version	Verze šablon
system_user	Uživatelé
system_user_alc_role	Přiřazení rolí uživatelům
system_user_settings	Uživatelské nastavení

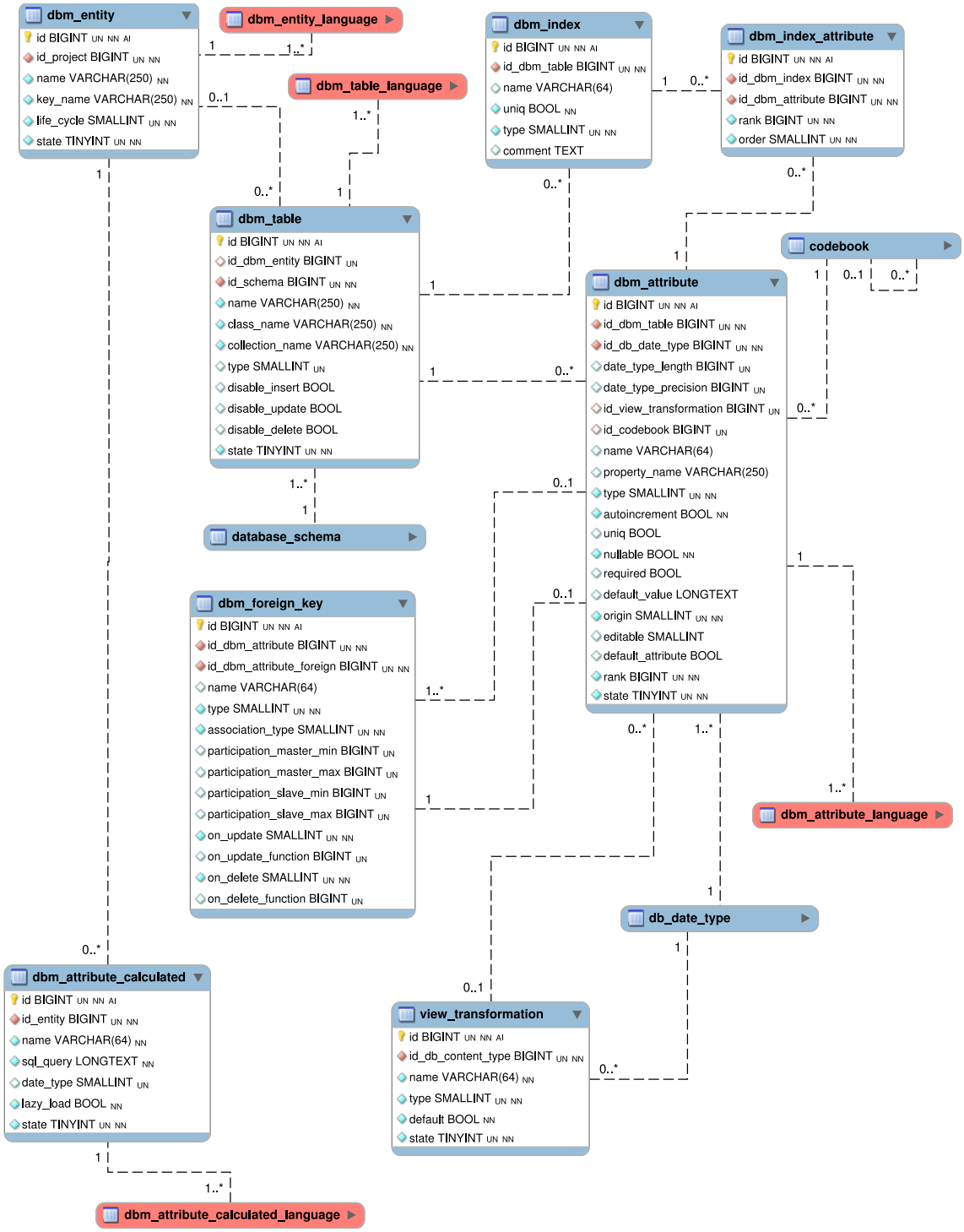
D ER DIAGRAMY – VÝVOJOVÉ PROSTŘEDÍ

D.1 Projekt a jeho nastavení



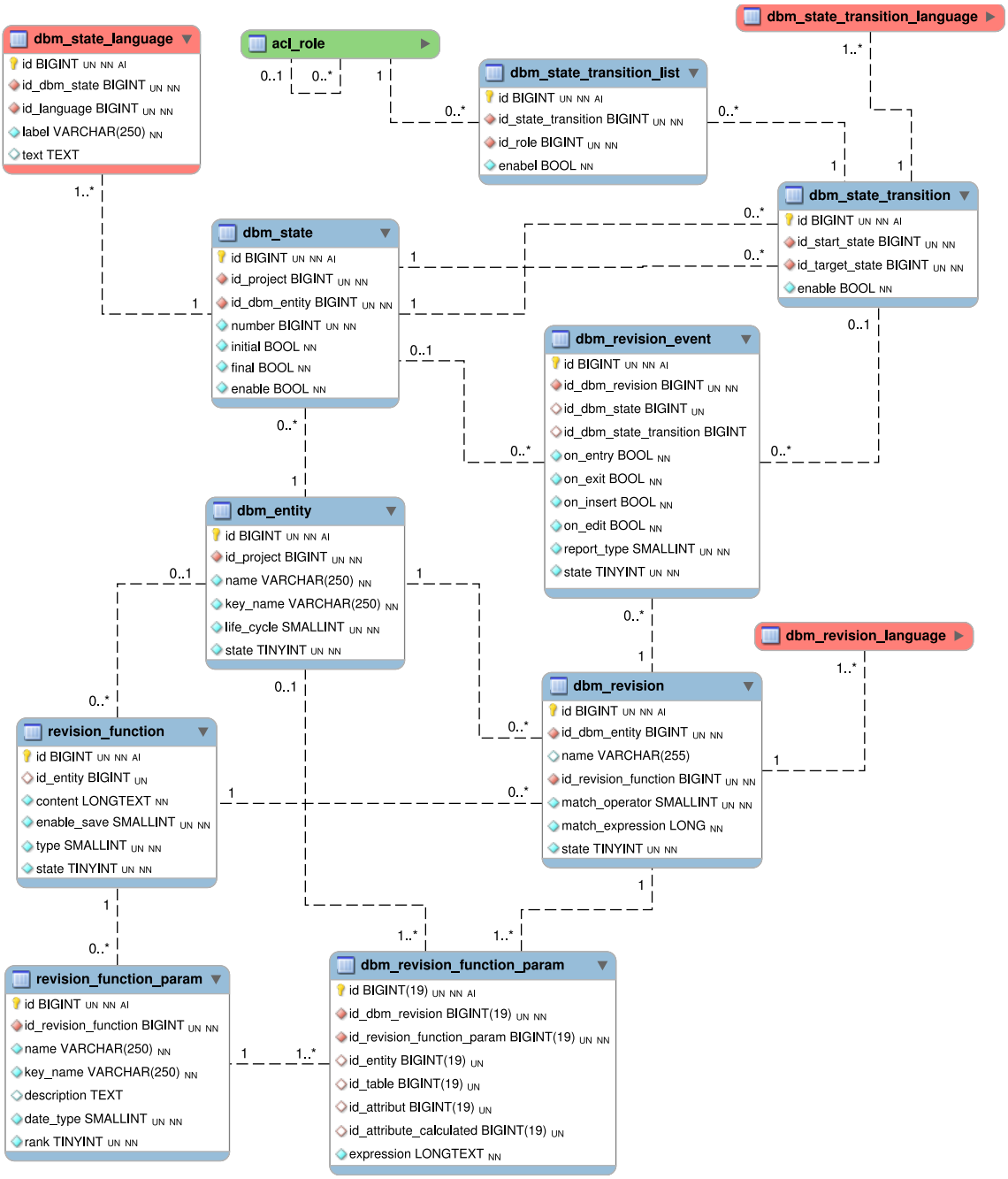
Obr. D.1: ER diagram – Projekt a jeho nastavení.

D.2 Datový model



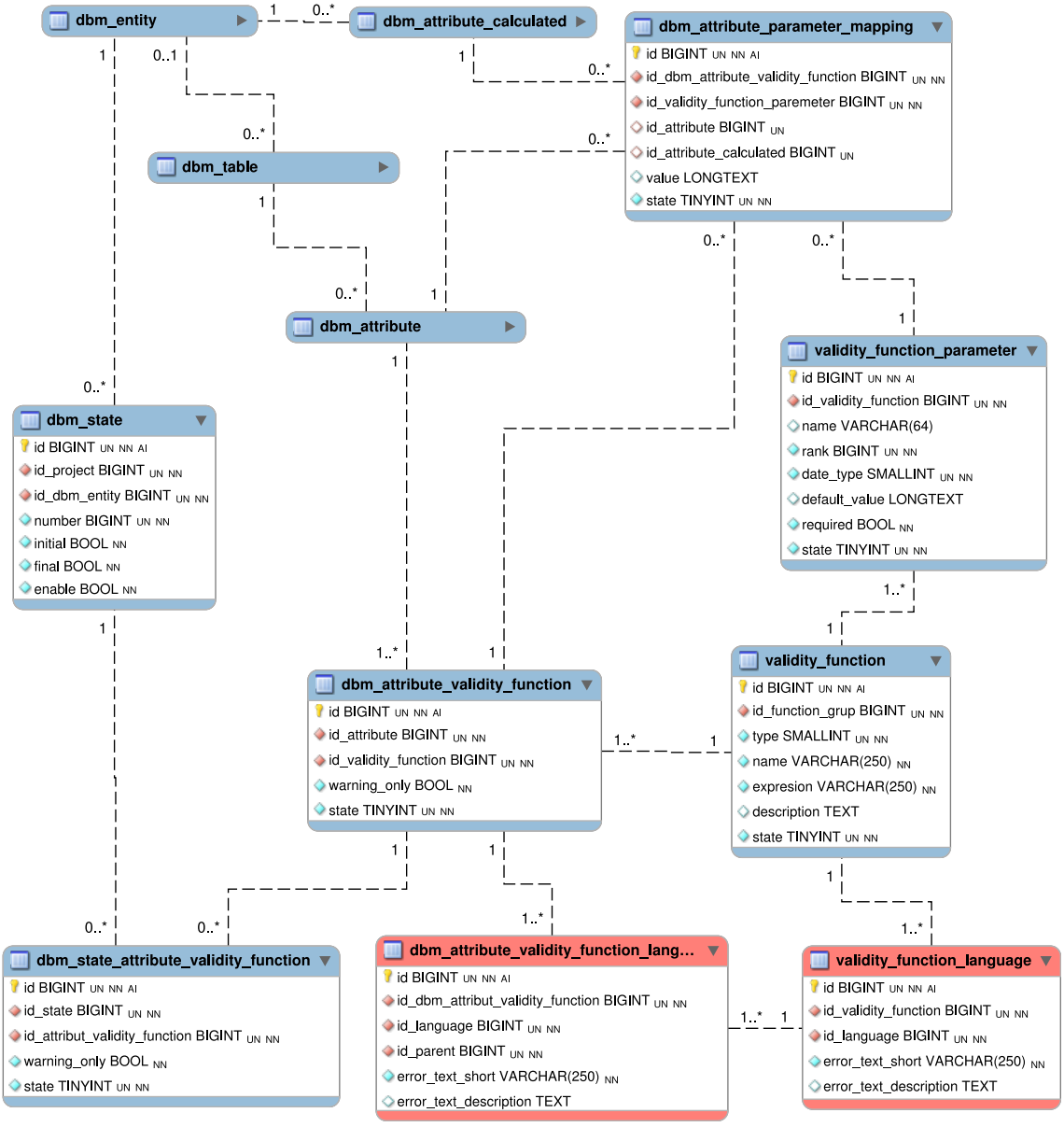
Obr. D.2: ER diagram – Datový model.

D.3 Životní cykly



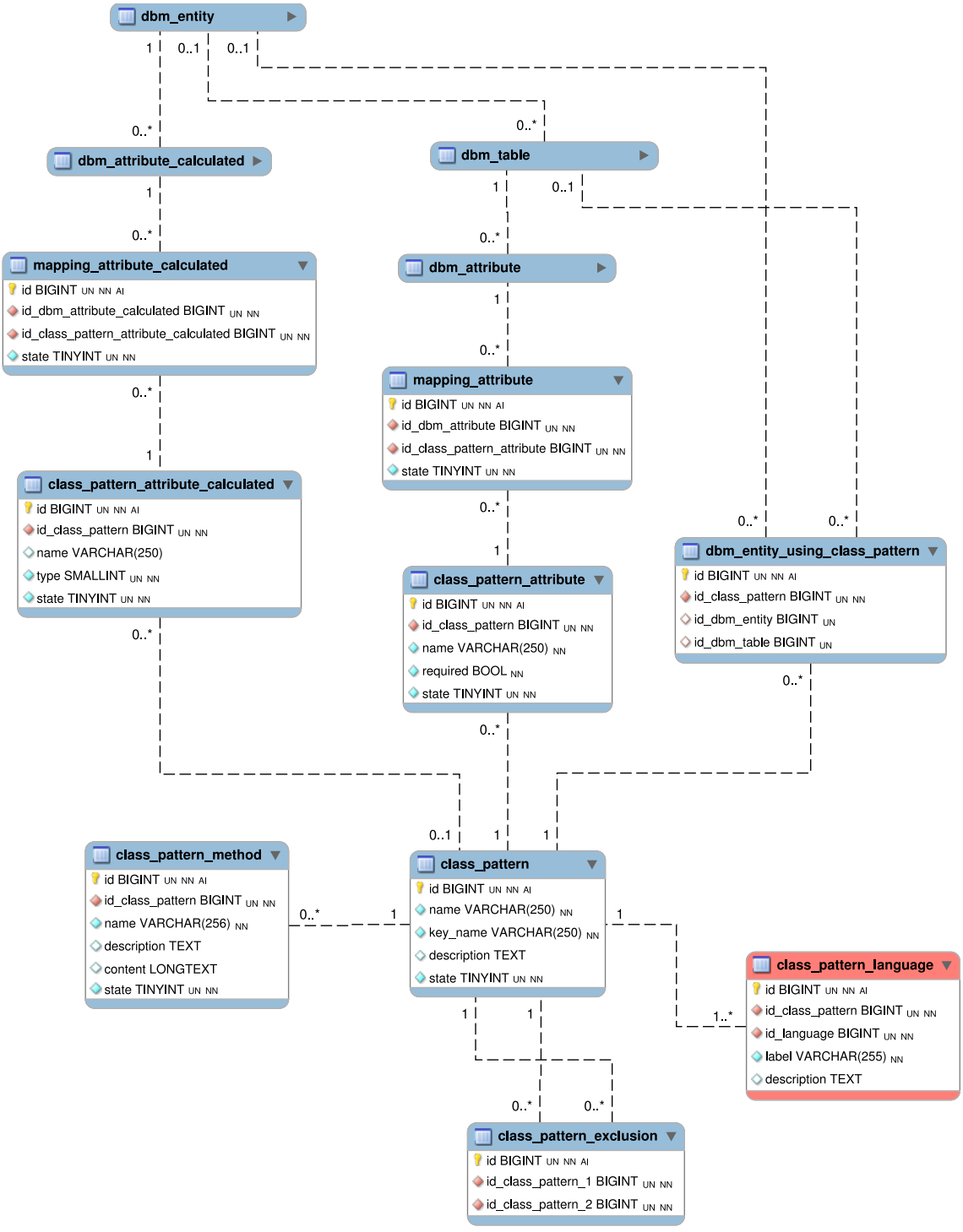
Obr. D.3: ER diagram – Životní cykly.

D.4 Dynamická validace



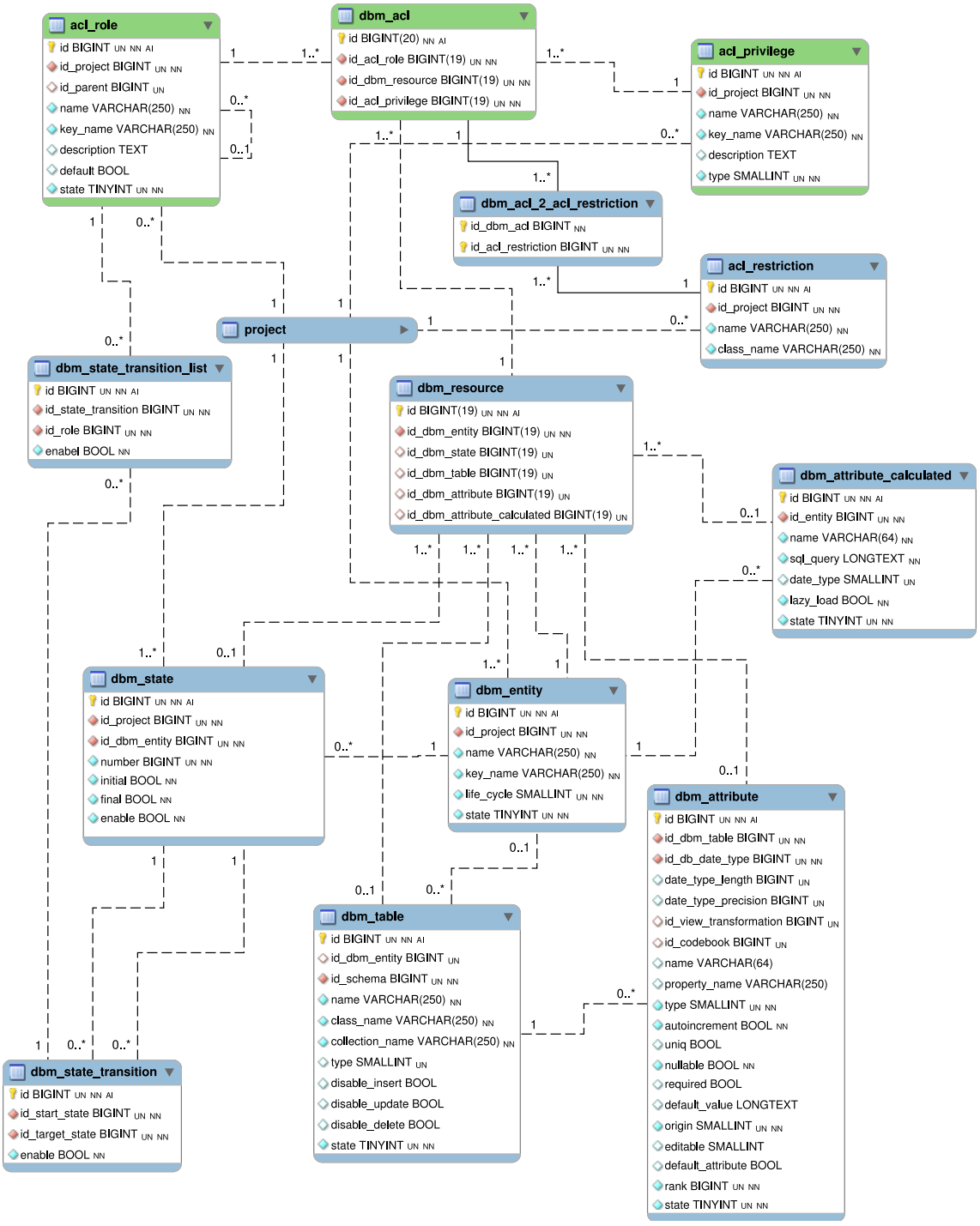
Obr. D.4: ER diagram – Dynamická validace.

D.5 Vzorové třídy



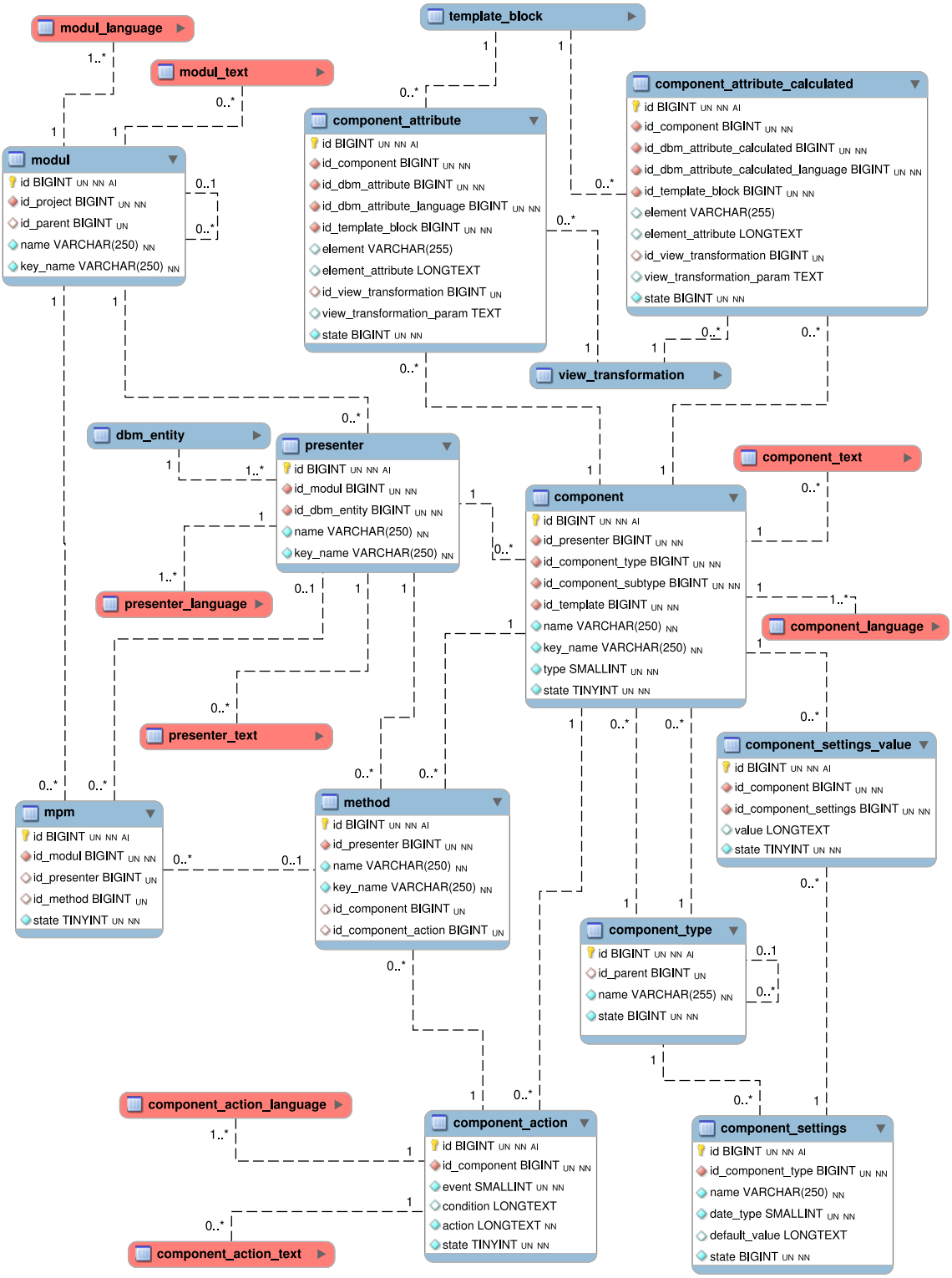
Obr. D.5: ER diagram – Vzorové třídy.

D.6 Přístupová oprávnění k modelu



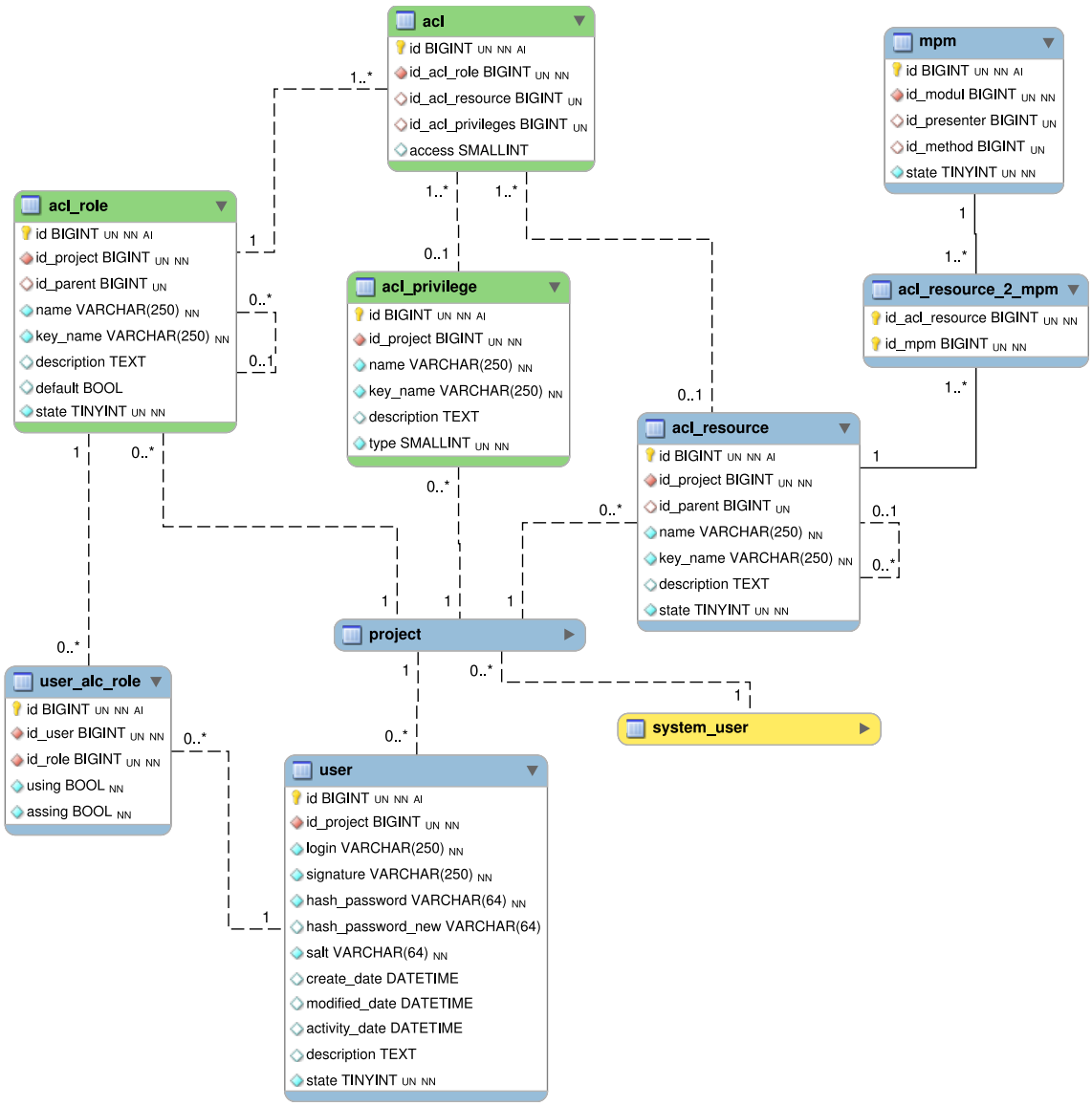
Obr. D.6: ER diagram – Přístupová oprávnění k modelu.

D.7 Prezentační část



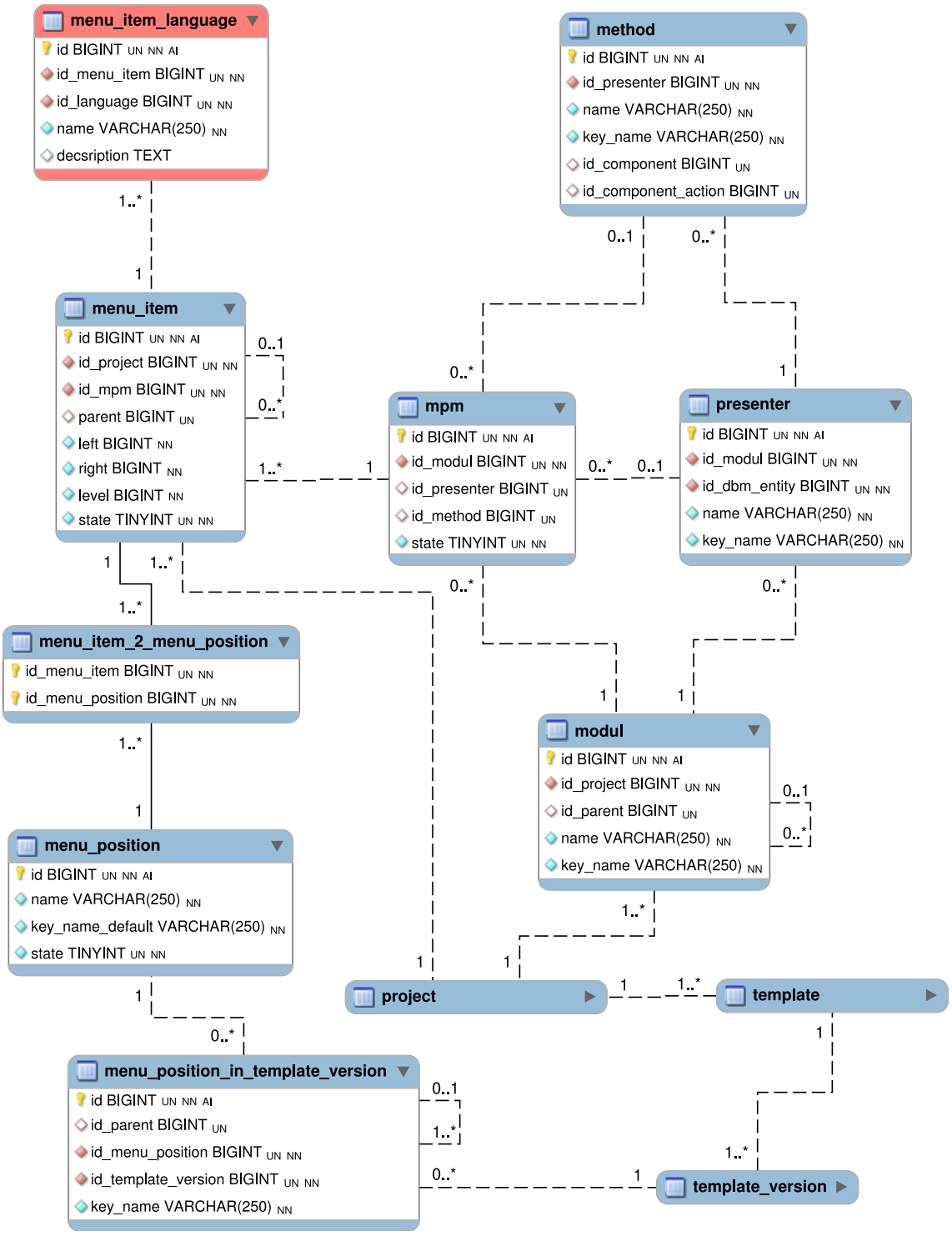
Obr. D.7: ER diagram – Prezentační část.

D.8 Přístupová oprávnění k prezentační části



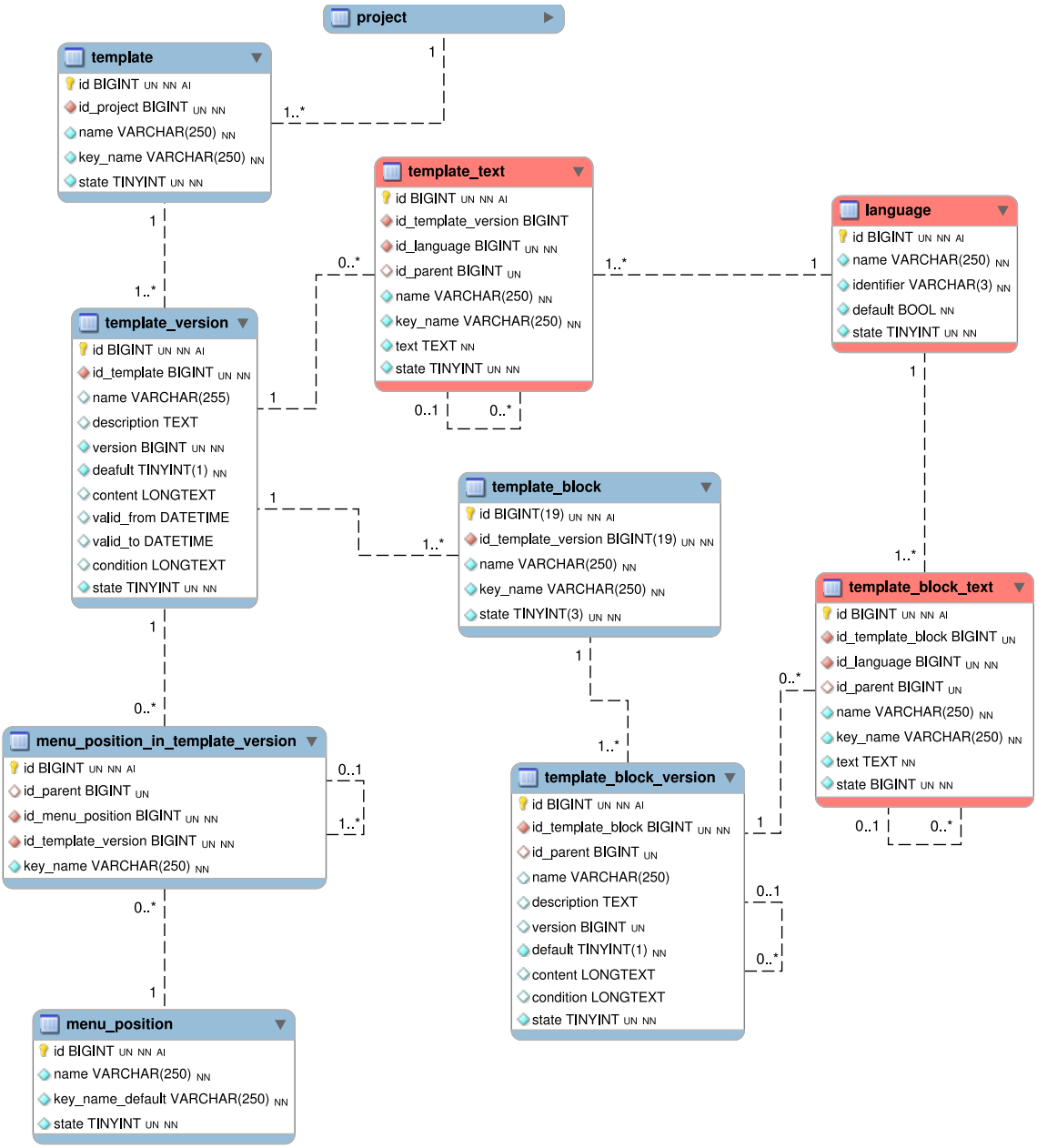
Obr. D.8: ER diagram – Přístupová oprávnění k prezentační části.

D.9 Navigace



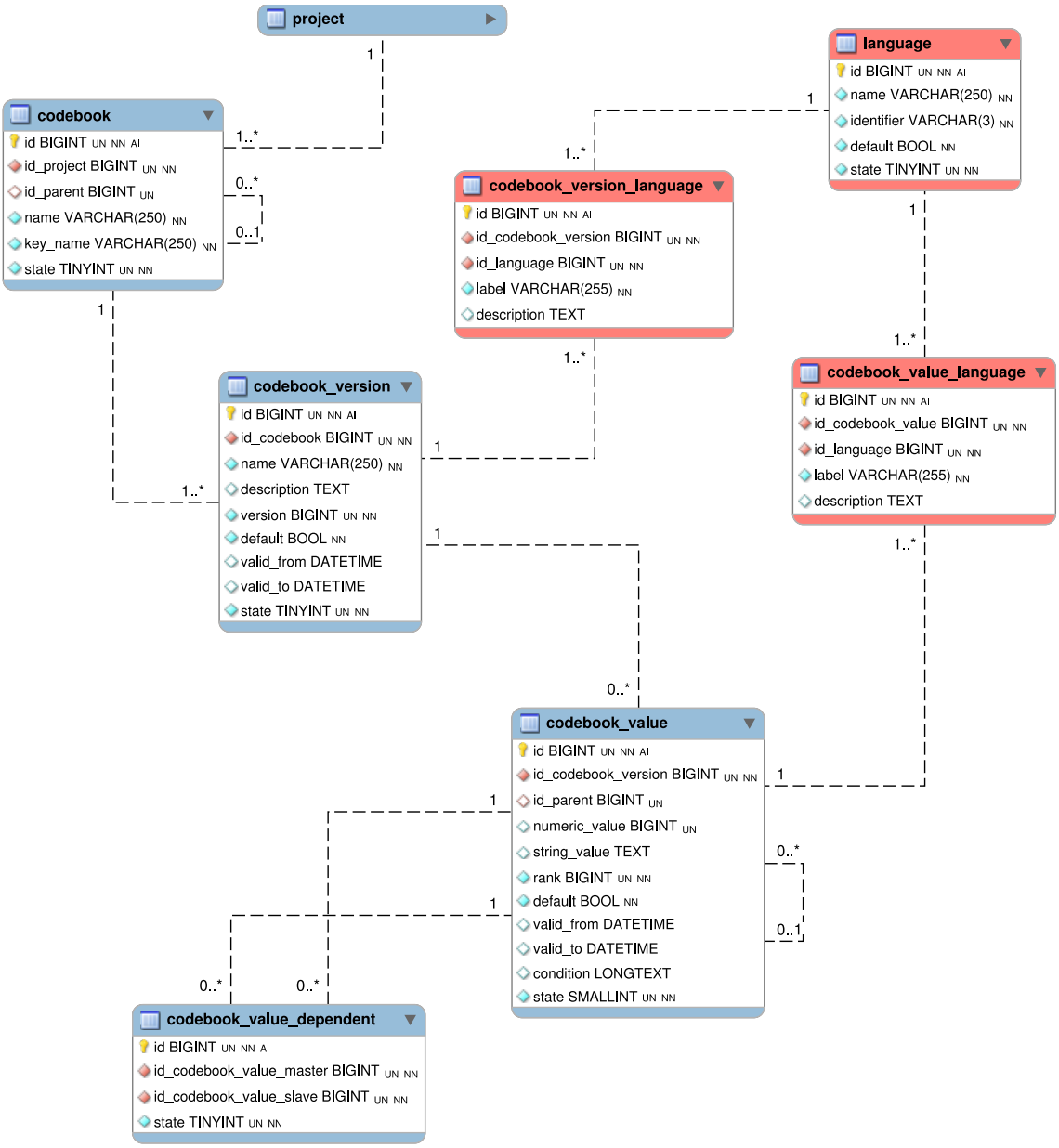
Obr. D.9: ER diagram – Navigace.

D.10 Šablony



Obr. D.10: ER diagram – Šablony.

D.11 Číselníky



Obr. D.11: ER diagram – Číselníky.

E KOMPLETNÍ SEZNAM TABULEK VÝVOJOVÉHO PROSTŘEDÍ

Tab. E.1: Kompletní seznam tabulek vývojového prostředí

Název tabulky	**Popis**
acl	Přístupová pravidla (prezentační část aplikace)
acl_privilege	Oprávnění
acl_resource	Zdroje (prezentační část aplikace)
acl_resource_2_mpm	Vazevní tabulka Zdroje : Model-Presenter-Metoda
acl_restriction	Omezení (modelová část aplikace)
acl_role	Role
dbm_attribute_parameter_mapping	Mapování parametrů
class_pattern	Vzorové třídy
class_pattern_attribute	Atributy vzorových tříd
class_pattern_attribute_calculated	Vypočítané atributy vzorových tříd
class_pattern_exclusion	Vzájemně se vylučující třídy
class_pattern_language	Vzorové třídy – jazyková tabulka
class_pattern_method	Metody vzorových tříd
codebook	Číselníky
codebook_value	Hodnoty číselníků
codebook_value_dependent	Závislé hodnoty
codebook_value_language	Hodnoty číselníku – jazyková tabulka
codebook_version	Verze číselníku
codebook_version_language	Verze číselníku – jazyková tabulka
component	Komponenty
component_action	Akce komponent
component_action_language	Akce komponent – jazyková tabulka
component_action_text	Akce komponent – texty
component_attribute	Atributy, používané komponentou
component_attribute_calculated	Vypočítané atributy, používané komponentou
component_language	Komponenty – jazyková tabulka
component_settings	Nastavení komponent
component_settings_value	Hodnoty nastavení
component_text	Komponenty – texty
component_type	Typy komponent

Název tabulky	Popis
database_connection	Připojení k databázi
database_schema	Databázové schema
database_system	Podporované databázové systémy
database_system_type_alias	Alias pro datový typ
db_content_type	Typ obsahu
db_date_type	Datový typ
dbm_acl	Pravidla
dbm_acl_2_acl_restriction	Kombinace omezení
dbm_attribute	Atribut
dbm_attribute_calculated	Vypočítaný atribut
dbm_attribute_calculated_language	Vypočítaný atribut – jazyková tabulka
dbm_attribute_language	Atribut – jazyková tabulka
dbm_attribute_validity_function	Validační pravidla
dbm_attribute_validity_function_language	Validační pravidlo – jazyková tabulka
dbm_entity	Entita
dbm_entity_language	Entita – jazyková tabulka
dbm_entity_using_class_pattern	Entity využívající vzorové třídy
dbm_foreign_key	Cizí klíč
dbm_index	Index
dbm_index_attribute	Atributy indexu
dbm_resource	Zdroje
dbm_revision	Kontroly
dbm_revision_event	Události
dbm_revision_function_param	Mapování parametrů
dbm_revision_language	Kontrola – jazyková tabulka
dbm_state	Stavy
dbm_state_attribute_validity_function	Validačních pravidla podle stavu
dbm_state_language	Stavy – jazyková tabulka
dbm_state_transition	Přechody
dbm_state_transition_language	Přechody – jazyková tabulka
dbm_state_transition_list	Oprávnění k přechodům
dbm_table	Tabulka
dbm_table_language	Tabulka – jazyková tabulka
dictionary	Slovník pro překlad názvů db. modelu
language	Tabulka jazyků
mapping_attribute	Mapování atributů
mapping_attribute_calculated	Mapování vypočítaných atributů

Název tabulky	Popis
menu_item	Položka v menu
menu_item_2_menu_position	Umístění položky do pozic
menu_item_language	Položka v menu – jazyková tabulka
menu_position	Pozice pro menu
menu_position_in_template_version	Pozice ve verzi šablony
method	Metody presenteru
modul	Moduly
modul_language	Moduly – jazyková tabulka
modul_text	Moduly – texty
mpm	Kombinace Modul-Presenter-Metoda
presenter	Presentery
presenter_language	Presentery – jazyková tabulka
presenter_text	Presentery – texty
project	Projekt
project_language_version	Jazykové verze projektu
revision_function	Kontrolní funkce
revision_function_param	Parametry kontrolních funkcí
template	Šablony
template_block	Bloky v šabloně
template_block_text	Bloky v šabloně – texty
template_block_version	Verze bloků v šabloně
template_text	Šablona – texty
template_version	Verze šablon
user	Uživatelé
user_alc_role	Přiřazení rolí uživatelům
validity_function	Validační funkce
validity_function_language	Validační funkce – jazyková tabulka
validity_function_parameter	Parametry validační funkce
view_transformation	Transformace zobrazení