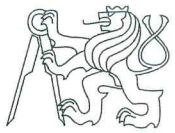
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Návrh a implementace backend projektu EBIE
Student:	David Hajčiar
Vedoucí:	Ing. Michal Valenta, Ph.D.
Studijní program:	Informatika
Studijní obor:	Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2016/17

Pokyny pro vypracování

EBIE (Extended Business Inteligence Encyclopedia) je specializovaný portál pro zobrazení metadat a byznys kontextu nad datovým skladem.
Cílem této práce je analýza, návrh a implementace backend.

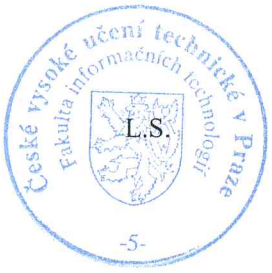
1. Seznamte se se stávající informační architekturou EBIE, vytvořte a dokumentujte konceptuální datové schéma portálu.
2. Seznamte se s obsahovými požadavky projektu EBIE a navrhněte vhodné řešení (technologie a architektura) pro správu obsahu.
3. Pro vybrané části konceptuálního schématu z bodu 1 realizujte funkční prototyp založený na řešení zvoleném v bodu 2.
4. Seznamte se se strukturou metadat v datových skladech.
5. Analyzujte, navrhněte a implementujte transformaci informací o technických a byznys metadatech ze stávajícího formátu do portálu EBIE.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

L.S.



prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 17. února 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Bakalářská práce

Návrh a implementace backend projektu EBIE

David Hajčiar

Vedoucí práce: Ing. Michal Valenta, Ph.D.

16. května 2016

Poděkování

Chtěl bych poděkovat svému vedoucímu Ing. Michalovi Valentovi, Ph.D. za vedení mé práce, vstřícnost při řešení a za konzultace, které mi velmi pomohly. Dále bych rád poděkoval členům vývojového týmu EBIE a datového skladu za spolupráci a mé přítelkyni, rodině a blízkým za trpělivost a podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 16. května 2016 .....

Odkaz na tuto práci

Hajčiar, David. *Návrh a implementace backend projektu EBIE*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací backend portálu EBIE (Extended Business Intelligence Encyclopedia). V práci je portál dokumentován, je vytvořeno konceptuální datové schéma a navrženo řešení správy obsahu, transformace metadat z datového skladu a dalších backend požadavků. Správa obsahu je řešena pomocí programu Asciidoctor. Návrhy jsou ozkoušeny a implementovány.

Klíčová slova Portál EBIE, návrh a implementace backend, metadata, Asciidoctor, Ruby on Rails

Abstract

This bachelor thesis is dealing with desing and implementation of backend of portal EBIE (Extended Business Intelligence Encyclopedia). Portal is documented in the thesis, conceptual data schema is made and content management, metadata transformation from data warehouse and other backend requirements are designed. Content management is done by program Asciidoctor. Designs are tried and implemented.

Keywords Portal EBIE, backend desing and implementation, metadata, Asciidoctor, Ruby on Rails

Obsah

Seznam obrázků

Úvod

V dnešní době provozují organizace často větší množství více či méně autonomních systémů. V takových případech může být náročné udržet si celkový přehled a dostat se k potřebným informacím. Tento problém se týká i ČVUT.

Projekt EBIE (Extended Business Inteligence Encyclopedia) usiluje o návrh a implementaci specializovaného informačního portálu nad datovým skladem. Bude poskytovat konzistentní, přehledně prezentované a vzájemně provázané informace o sestavách datového skladu, datových tržištích, business a technických metadatech a byznys kontextu (ontologická datová a procesní vrstva) příslušné části datového skladu. První verze informační architektury EBIE je již navržena. Portál bude poprvé nasazen nad vznikajícím prototypem datového skladu ČVUT.

V souvislosti s tímto portálem tvoří bakalářské práce i Michal Kopecký [1] a Cyril Černý [2].

Cíl práce

Má práce se tímto projektem (EBIE) zabývá. V první části se věnuji seznámení se s portálem EBIE, dokumentuji jej a analyzuji jednotlivé technologie. Na závěr specifikuji obsahové požadavky.

V druhé části navrhuji řešení architektury portálu, uvádím mnou vytvořené konceptuální datové schéma a navrhuji a diskutuji řešení jednotlivých požadavků.

V závěrečné části pak ukazuji, jak jsem tyto návrhy implementoval.

Kapitola 1

Analýza

V této kapitole definuji základní pojmy, které souvisí s touto prací, popisuji portál EBIE a specifikuji obsahové požadavky.

1.1 Portál EBIE - počáteční stav

Dříve, než se budu věnovat samotnému portálu EBIE, je třeba definovat pojem Business Intelligence encyklopedie, jež EBIE rozšiřuje.

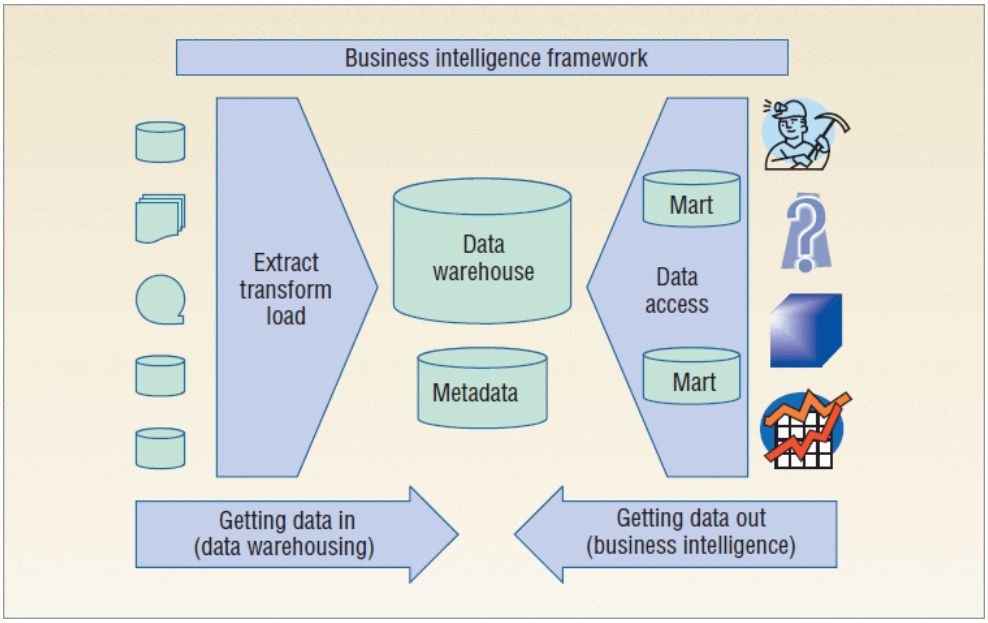
1.1.1 BI encyklopedie

Nejprve vysvětlím, co znamená Business Intelligence (BI). Vzhledem k tomu, že tento pojem není jednoznačně definován, uvádím zde dvě různé definice pro lepší pochopení: *„Business Intelligence (BI) je sada procesů, aplikací a technologií, jejichž cílem je účinně a účelně podporovat rozhodovací procesy ve firmě. Podporují analytické a plánovací činnosti podniků a organizací, a jsou postaveny na principu multidimenzionality.“* [3]

„Business intelligence (BI) is a technology-driven process for analyzing data and presenting actionable information to help corporate executives, business managers and other end users make more informed business decisions.“ [4]

Produktem BI jsou data v požadovaném stavu, která poskytují potřebné informace.

Tzv. BI Encyklopedie jsou aktuálním a úspěšným řešením potřeby velkých organizací, které z historických důvodů provozují mnoho různých více či méně autonomních a částečně spolupracujících systémů pro podporu dílčích agend, k udržení přehledu o datech a jejich souvislostech. BI Encyklopedie tedy často zobrazují data z datového skladu (více informací o datových skladech uvádím v sekci 1.2.1) [5]. Celý BI proces je ilustrativně znázorněn na obrázku 1.1.



Obrázek 1.1: Proces BI nad datovým skladem. Obrázek převzat z článku [5]

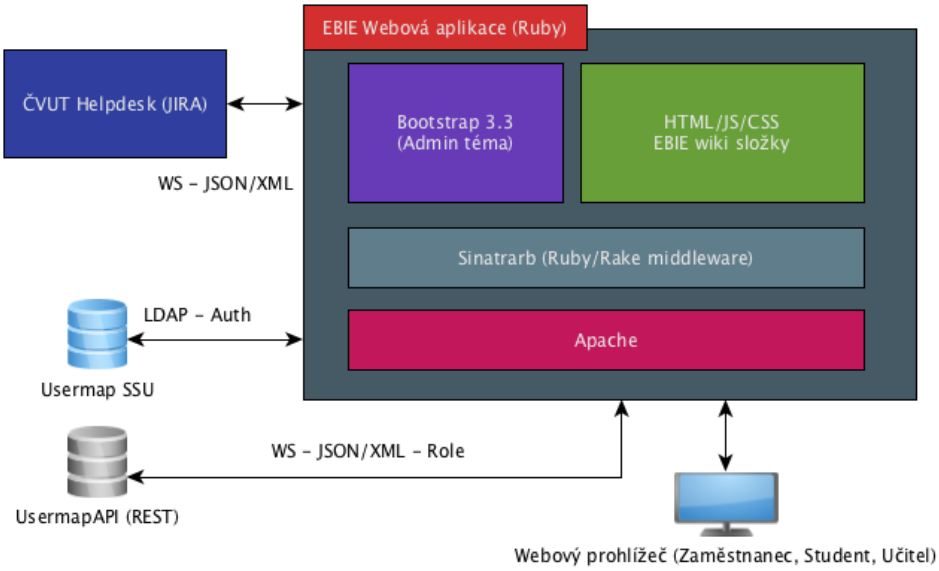
1.1.2 EBIE

EBIE - Extended Business Intelligence Encyclopedia značí v našem případě BI encyklopedii rozšířenou o konceptuální vrstvu OntoUML a DEMO (OntoUML je nadstavba UML pro konceptuální modelování [6], metodika DEMO (Desing & Engineering Methodology for Organisations) je určená k analýze a modelování procesů v organizaci [7]). Na obrázku 1.2 je vidět architektura EBIE v září 2015. Ke správě vývoje a verzí se používá gitlab, vlastní aplikace je napsaná v Ruby on Rails.

1.1.3 Použité technologie

Stručný popis (a uvedení) technologií použitých v mé práci:

- Ruby - Ruby je interpretovaný skriptovací programovací jazyk, který vytvořil Yukihiro Matsumoto [8].
- Gem - Gemy jsou v Ruby napsané knihovny a programy, které rozšiřují funkcionalitu tohoto jazyka [9].
- Ruby on Rails - Jedná se o webový framework napsaný v ruby. Podrobněji o něm píši v sekci 1.1.4.
- git - Distribuovaný systém správy verzí softwaru. [10] Aplikace EBIE je uložena v repositáři ebie-source-code.



Obrázek 1.2: Architektura EBIE v září 2015

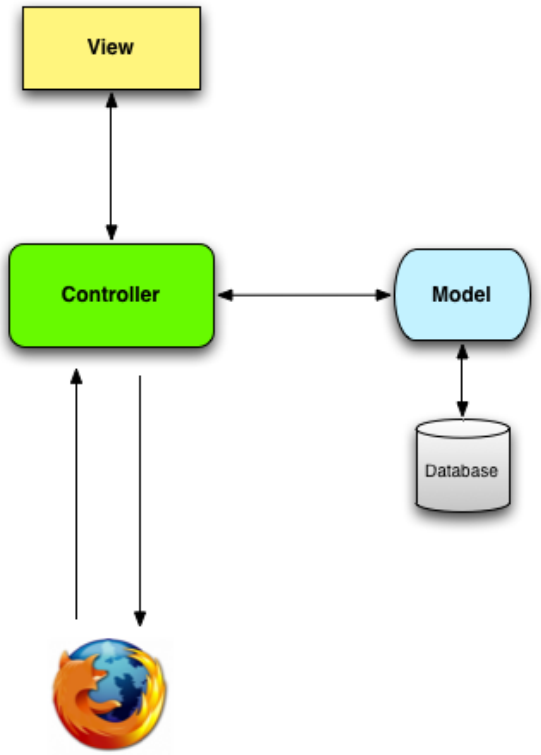
- Asciidoctor - Asciidoctor je implementaci značkovacího jazyka Asciidoc v Ruby, lze jej nainstalovat jako gem. Umožňuje konverzi prostého textu, s jednoduchými syntaktickými pravidly formátu Asciidoc, do HTML5 (a do dalších formátů, jako třeba DocBook nebo pdf) [11].
- UML - Unified Modeling Language je jazyk sloužící k specifikaci, vizualizaci a dokumentování modelů počítačových systémů i jejich struktury a designu [12].

1.1.4 Struktura Rails aplikace

Portál EBIE je psaný v Ruby on Rails. Rails používají architekturu MVC (Model-View-Controller). Jedná se o architektonický vzor, který dělí aplikaci do tří logických vrstev:

- Model - reprezentuje data v aplikaci. Je nezávislý na uživatelském prostředí.
- View (pohled) - zobrazuje jednotlivé elementy uživatelského rozhraní.
- Controller (kontroler) - zpracovává uživatelské požadavky, na jejichž základě zajišťuje spojení mezi vrstvami model a view. [13]

Tento model ilustruje obrázek 1.3. Typická aplikace v Rails (vygenerovaná příkazem rails new NázevAplikace) obsahuje jednotlivé části MVC ve složce



Obrázek 1.3: Zobrazení základní logiky MVC aplikací. [14]

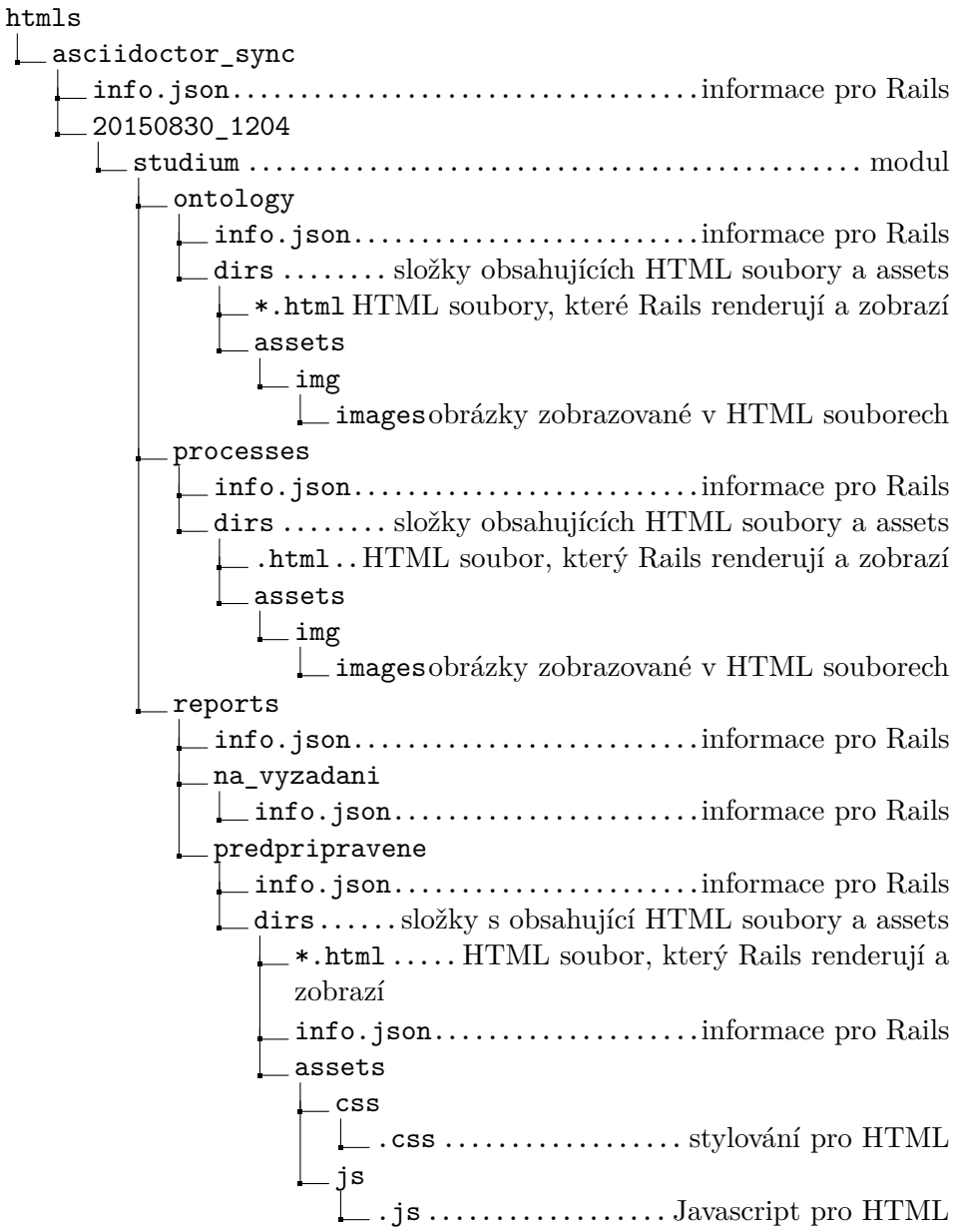
app ve svých vlastních podsložkách (app/models, app/views, app/controllers). Dále se zde nachází „helpery“ (app/helpers), které obsahují různé funkce používané v jednotlivých views a controllers (díky tomu jsou pohledy i kontrolery přehlednější) [15], „mailery“ (app/mailers), které se starají o posílání a přijímání e-mailů [16] a adresář assets, který obsahuje obrázky, javascriptové soubory, css soubory a také může obsahovat HTML soubory (tak tomu je v EBIE).

Do složky lib (v kořenovém adresáři) se mimo jiné ukládají vývojářem vytvořené generátory (tomu se více věnuji v sekci 2.3.7).

Důležitý je soubor routes.db (config/routes.rb), který definuje, jak aplikace přesměrovává příchozí dotazy (URL) na určitý controller a jeho akci [17].

V souboru gemfile jsou pak uvedené všechny aplikací používané gemy (volitelně i s požadovanou verzí jednotlivých gemů). Instalaci, update a další operace s gemy zajišťuje gem Bundler [18].

Rails aplikace má samozřejmě mnohem více části, uvedl jsem zde pouze ty, které jsou podstatné pro moji práci.



Obrázek 1.4: Strom adresářů s obsahem EBIE

1.1.5 Struktura obsahu

Dříve, než popíši strukturu aplikace EBIE, je třeba podrobně popsat obsah tohoto portálu.

Adresářová struktura obsahu EBIE (pro větší přehlednost neuvádím jednotlivé HTML, img, css nebo js soubory) je vidět na obrázku 1.4.

1.1.6 Obsah EBIE

Portál EBIE se na nejvyšší úrovni dělí na jednotlivé moduly. Aplikace načítá a zobrazuje moduly podle souboru info.json ve složce asciidoctor_sync. Jelikož se stále jedná o prototyp, je v tuto chvíli řešený pouze modul Studium. Předpřipravené, a zatím prázdné jsou moduly Ekonomický modul, Přehled středních škol a Dokumentace pro vývojáře. Dokumentace pro vývojáře bude obsahovat informace pro vývojáře datového skladu a pro aplikace, které by využívaly API (Application Programming Interface) datového skladu. Moduly se dělí na 3 sekce:

- **Reporty** (sestavy) - slouží k prezentování dat v tabulkové nebo grafické podobě. Této oblasti se více věnuje Michal Kopecký ve své bakalářské práci [1]. Sekce se dále dělí na reporty předpřipravené a na vyžádání. V tuto chvíli jsou implementované pouze ukázkové předpřipravené reporty, možnost vytváření reportů na požádání se připravuje (tomu se ve své bakalářské práci věnuje Cyril Černý [2]).

- **Ontologický katalog** - obsahuje informace o entitách, které figurují v doméně. Cílem ontologického katalogu je jednoduché popsání těchto informací pro koncového uživatele, ale přesto přesné vymezení pojmů a vztahů mezi těmito entitami. [7] Jednotlivé entity obsahují vazby na s nimi spojené procesy a reporty.

- **Procesy** - popisují činnosti organizace (uznání předmětu, vytvoření a schválení tématu závěrečné práce, přihlášení ke státní zkoušce apod.). Zobrazení procesů v portálu doplňuje informaci o změnách stavu byznys entit a o tom, jak na sebe navazují. Použitím adekvátních metodik lze procesy přehledně zobrazit, zdůraznit hlavně ty části, které vyžadují aktivní zásah příslušného aktéra (tzv. ontologická transakce), přehledně zobrazit mapování aktérů na role v organizaci a provést kontrolu úplnosti procesů.

Každá sekce má definovanou strukturu. Jelikož se ve své práci nezabývám reporty, jejich obsah popíši pouze stručně. Hlavní účel reportu je zobrazení určité sestavy buď v tabulce, nebo jako graf. Dále stránka obsahuje seznam entit vyskytujících se v reportu a seznam procesů s ním souvisejících. Na jednotlivé entity i procesy jsou vytvořené odkazy. Ve všech sekcích platí, že informace, které se týkají pouze Fakulty informačních technologií, jsou označené značkou *FIT*.

1.1.7 Obsah stránky ontologického katalogu

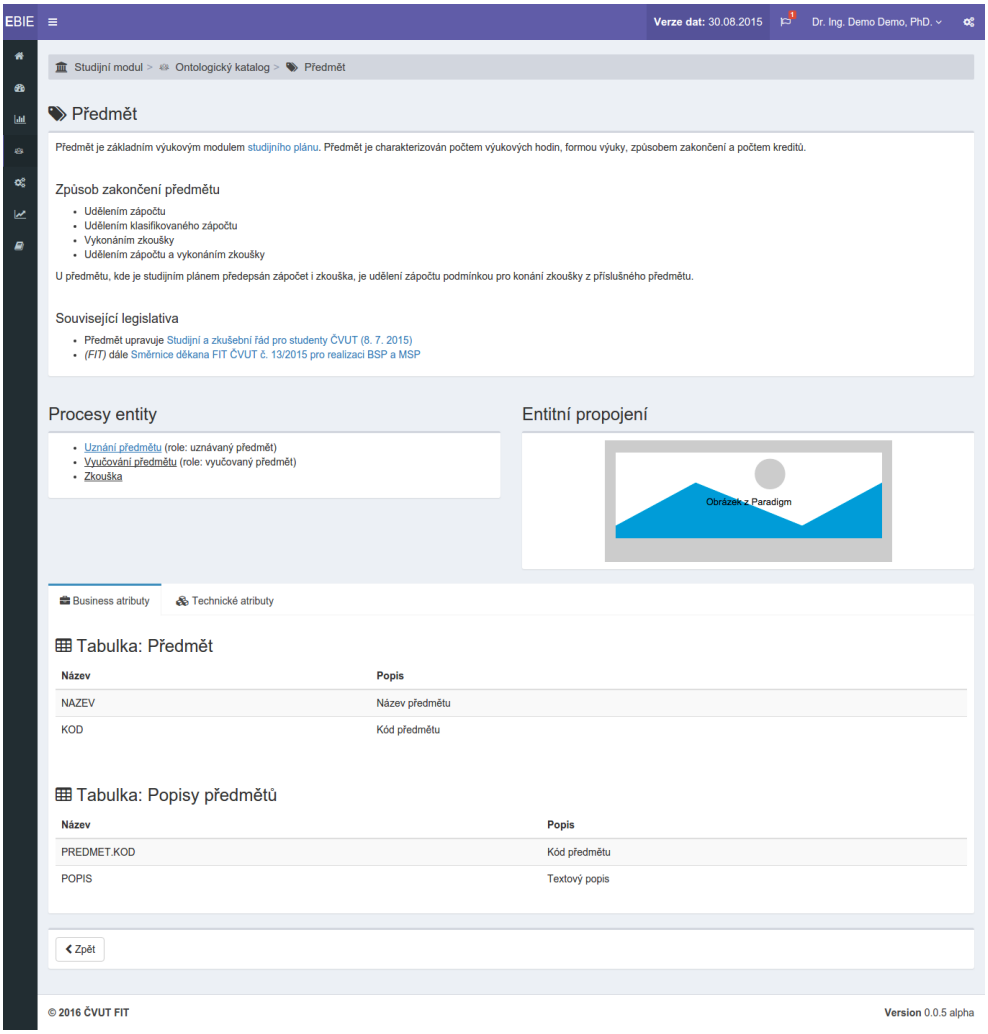
- Nadpis - Název entity (kvůli oddělení na stránce je prvním znakem mezera)

- Popis entity - Přesný popis s entity, je možné uvádět odkazy na související entity.
- Příklad instance - Uvádí praktický příklad užití entity (např. entita Zápis předmětu má jako příklad instance uvedeno: „Student si zapsal v ZS 2014/2015 předmět MI-PAR, paralelku 1.“
- Související legislativa
- Odkazy na procesy této entity - Může být uvedena i role entity v daném procesu.
- Odkazy na reporty této entity
- Obrázek entitního propojení - Náhled na relevantní část konceptuálního schématu datového skladu, která ukazuje vazby této entity na ostatní entity. Tato funkce ještě není v portálu implementována.
- Tabulka s business atributy entity
- Tabulka s technickými atributy entity

Na obrázku 1.5 je zobrazena stránka entity předmět.

1.1.8 Obsah stránky procesu

- Nadpis - Název procesu (kvůli oddělení na stránce je prvním znakem mezera)
- Popis procesu
- Popis procesu podle DEMO metodiky - Obrázek
- Rozepsání jednotlivých transakcí - Nadpis (označení a popis transakce), tabulka popisující iniciátora, exekutora a produkt transakce.
  - Podpůrné informace pro transakci - Další informace ke transakci, jako třeba odkazy na související legislativu nebo reporty.
- Tabulka popisující vazby aktorů na ontologický katalog
- Popis procesu podle BPMN (Business Process Model and Notation) metodiky - Obrázek, se zdrojem



Obrázek 1.5: Ukázka entity předmět v ontologickém katalogu portálu EBIE

1.1.9 struktura Rails aplikace EBIE

EBIE obsahuje tyto kontrolery (kontrolery jsou Ruby třídy):

- Application controller - Základní kontroler, ze kterého dědí všechny ostatní.
- bi_modules controller - Akce show zajišťuje načtení modulů.
- data_version controller - Zajišťuje zobrazení úvodní stránky.
- ontology controller - Akce index zobrazí seznam všech entit, view ukáže vybranou entitu.

- processes controller - Akce index zobrazí seznam všech procesů, view ukáže vybraný proces.

- reports controller - Akce index zobrazí výběr mezí reporty předpřipravenými a na vyžádání, u reportů na vyžádání poté zajišťuje výpis jejich seznamu, view ukáže vybraný report.

- sections controller - Dědí z něj kontrolery ontology, processes a reports (tedy sekce).

Ke každému z kontrolerů (kromě Application a sections) jsou vytvořené pohledy pro jednotlivé akce (např. pro akci index na kontrollor ontology je soubor ./app/views/ontology/index.html.erb (soubory .html.erb jsou šablony sestávající se z HTML spojeného s erb (Embedded Ruby)[19]).

Ve views/layouts jsou definované header, sidebar, controlsidebar a footer stránky. Také zde je soubor application.html.erb, ve kterém se nastavuje připojení stylů (css) a javascriptu do aplikace a zobrazení stránky podle toho, zda je uživatel přihlášený (pokud ano, použije se soubor _main.html.erb, který mimo jiné načte header, footer, sidebar a controlsidebar, jinak se zobrazí přihlašovací stránka). Jednotlivé stránky sekcí se pak renderují do stránky definované souborem view.html.erb v jejich pohledu (např. views/ontology/view.html.erb).

1.1.10 Načítání modulů a sekcí v aplikaci

Sekce ontologický katalog i procesy mají podobnou strukturu adresářů. V souboru info.json (který je v každé složce různý) je definováno, které položky se zobrazí v portálu a s jakým popisem. Jako příklad uvádím část kódu pro entitu anketa. V ontology/info.json je:

"title": "Anketa",
"description": "Hodnotící systém kvality činností na univerzitě.",
"path": "anketa",
"filename": "anketa.html"

To značí, že se zobrazí nadpis Anketa s popiskem "Hodnotící systém kvality činností na univerzitě." a že soubor anketa.html se nachází ve složce anketa). Tyto soubory jsou napsané podle notace JSON (JavaScript Object Notation) [20].

Stejným způsobem se do aplikace načítají informace o reportech i o modulech.

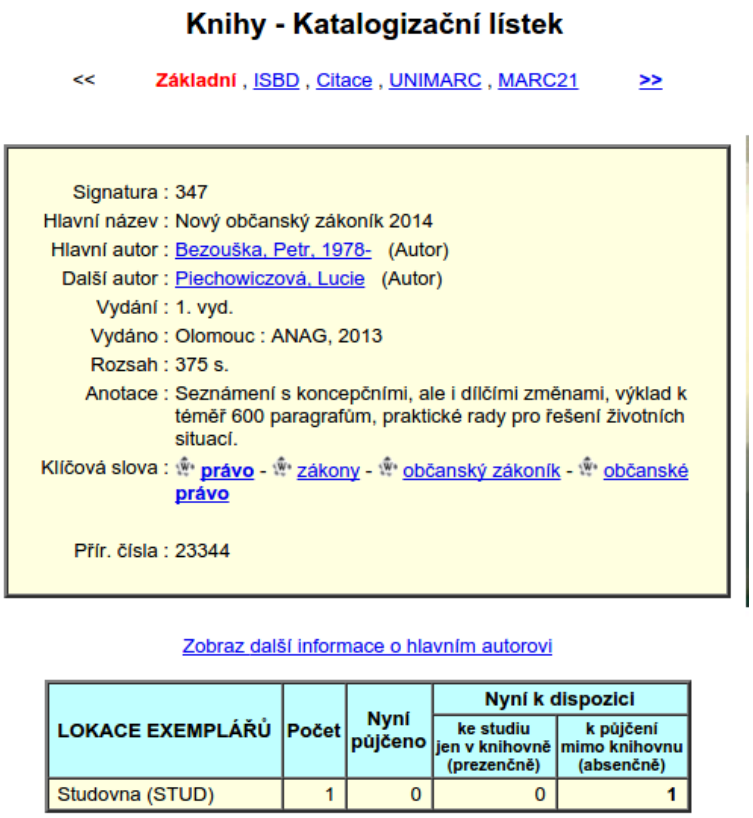
1.2 Metadata

Jeden z požadavků na tuto práci je zajistit transformaci metadat z datového skladu. V této sekci uvádím definice a popis těchto pojmů a dalších, s nimi souvisejících, pojmů.

1.2.1 Metadata

Jednoduše se metadata dají popsat jako data o datech. Výstižnější definice, která lépe vyjadřuje, co metadata značí, je *„data sdružená s objekty, které zbavují jejich potenciální uživatele nutnosti předběžné znalosti existence či charakteristik těchto objektů“* [21].

Jedna z oblastí, kde se metadata používají, je katalogizace. Na obrázku 1.6 je vidět ukázka katalogizačního lístku z elektronického katalogu knihovny, kde můžeme pozorovat použití metadat.



Obrázek 1.6: Katalogizační lístek knihovny Oslavany

1.2.2 Datový skladu

Datový sklad jako první definoval William H. Inmon. Později přišel s odlišnou definicí Ralph Kimball. Jelikož datový sklad ČVUT je budován podle jeho definice, budu se jí řídit ve své práci. Datový sklad je „The queryable source of data in the enterprise. The data warehouse is nothing more than the union of all the constituent data marts. A data warehouse is fed from the data staging area“ [22]. Data marts je označení pro datová tržiště, z nichž se datový sklad

sestává (datové tržiště je „A logical subset of the complete data warehouse“ [22]). Datovému skladu na Fakultě informačních technologií ČVUT se ve své bakalářské práci věnuje Bc. Jakub Krejčí [23].

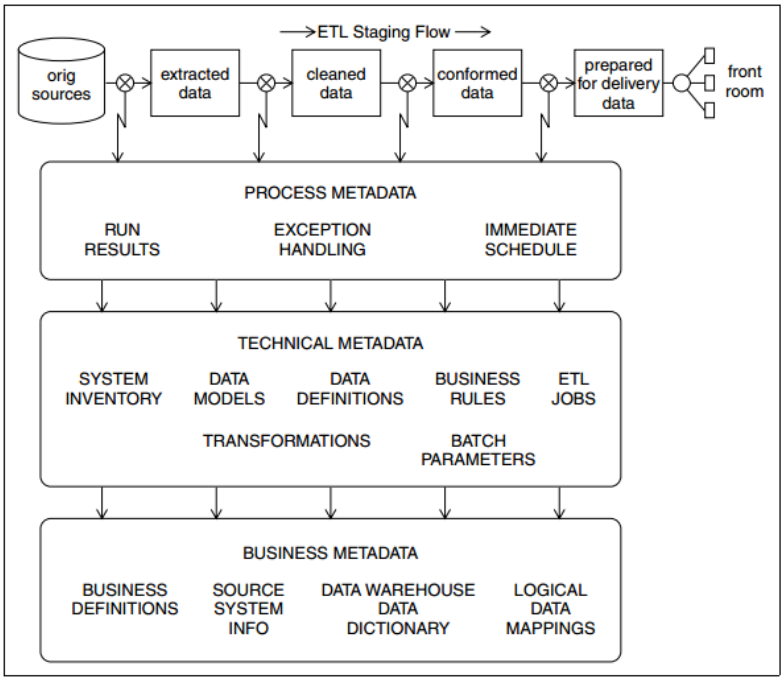
Fakultní datový sklad využívá databázový systém PostgreSQL.

1.2.3 Metadata v datovém skladu

Metadata v datových skladech můžeme rozdělit na 3 základní kategorie:

- Procesní metadata - prezentují statistiky výsledků běžících ETL procesů (ETL (Extract-transformation-load) procesy zpracovávají data a transformují je do potřebné podoby pro datový sklad).
- Business metadata - uvádí business definice („business definition is one or two sentences that describe the business meaning of an attribute“) pro data v datovém skladu
- Technická metadata - popisují data v datovém skladě z technického hlediska (jako např. atributy v tabulkách, typy uložených dat atd.) [24].

Toto rozdělení ilustruje obrázek 1.7.



Obrázek 1.7: Metadata v datovém skladu [24]

1.3 Specifikace požadavků

Na základě mého seznámení se s portálem, schůzek vývojářského týmu a konzultací s Michalem Valentou jsem určil tyto obsahové požadavky:

- Popis, diskuze a úprava architektury obsahu EBIE
- Nalezení vhodné technologie pro správu obsahu EBIE, která by splňovala tyto požadavky:
  - Možnost jednoduché úpravy stávajícího obsahu
  - Jednoduché přidávání dalšího obsahu
  - Přehlednost
  - Možnost tvorby interních odkazů
  - Použití výchozích stylů EBIE
- Oddělení obsahu EBIE od vlastní aplikační logiky
- Zpřehlednění adresářů s obsahem a zjednodušení URL pro přístup
- Zobrazení času změny obsahu a aktuální verze aplikace
- Návrh a začlenění business a technických metadat datového skladu do obsahu EBIE

Kapitola 2

Návrh

V této kapitole se věnuji návrhu řešení vzniklých požadavků a uvádím konceptuální schéma portálu EBIE.

2.1 Revidovaná architektura

Nejdříve se věnuji požadavkům na úpravu architektury.

2.1.1 Obsah adresářů

Strukturu adresářů s obsahem EBIE jsem již uvedl dříve (viz. obrázek 1.4). Navrhuji provést několik změn:

- Přejmenování adresáře Asciidoc_sync na ebie_content - název adresáře by měl vyjadřovat jeho obsah.
- Odstranění adresáře 20150830_1204 - jedná se o pozůstatek původního řešení, ve kterém se počítalo s různými verzemi dat.
  - S tím je spojeno přesunutí obsahu adresáře 20150830_1204 do složky ebie_content (cesta k souborům se zkrátí o jeden adresář).
- Odstranit podsložky v adresářích ontology a processes (veškerý obsah bude tedy přímo v těchto adresářích).
  - Obsah těchto podsložek se přesune přímo do adresářů ontology respektive processes.
  - Obsah jednotlivých složek assets se sloučí. Tím vzniká nutnost zavést konvenci pojmenování souborů, aby byly jednoznačně určitelné, a přejmenovat stávající soubory. Navrhuji tvořit jména tímto stylem pro procesy: *NázevProcesu_NázevSouboru* a pro entity: *NázevEntity_NázevSouboru*. Pro soubor bpmn.png, který náleží do

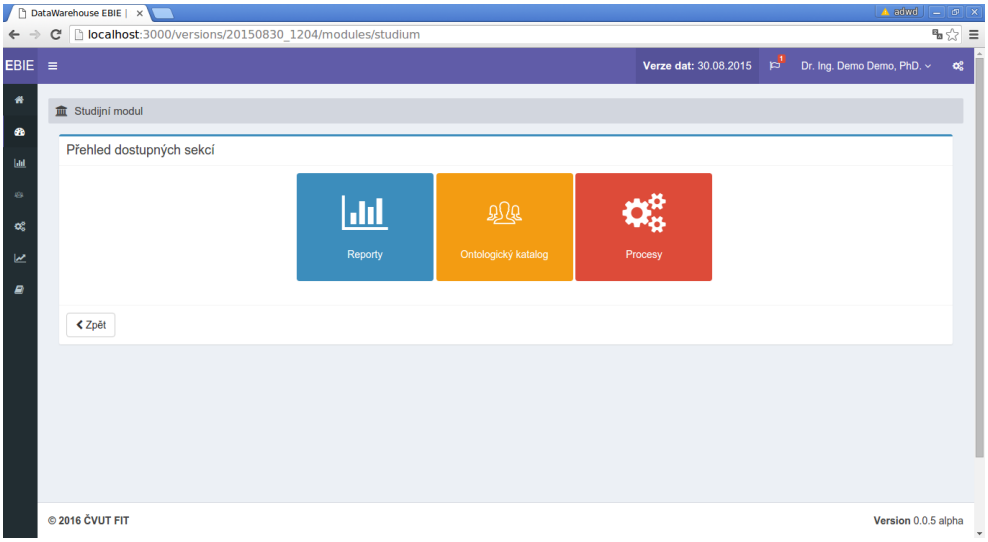
procesu státní závěrečná práce (szz), pak bude výsledný název szz_bpmn.png (pokud je soubor s entitou/procesem pojmenován zkratkou tohoto názvu (jako třeba szz), použijeme tuto zkratku).

- Zanést tyto změny do info.json souborů

Výsledný návrh uvádím až později, protože kvůli návrhům na řešení dalších požadavků jsem do výsledné struktury zahrnul další změny. Finální návrh je k vidění na obrázku 2.3.

2.1.2 Úprava URL

Kvůli původnímu úmyslu zakomponování do EBIE více různých verzí dat vyžaduje aktuální aplikační logika EBIE URL adresy, které v sobě obsahují, nyní již zbytečně, výraz „/versions/20150830_1204“ (viz. URL na snímku obrazovky 2.1 z mé lokální (aktuální) verze portálu EBIE). Díky tomu je výsledná URL adresa nejen zbytečně dlouhá, ale také není dobře čitelná. Aplikace navíc musí provádět zbytečné operace navíc.



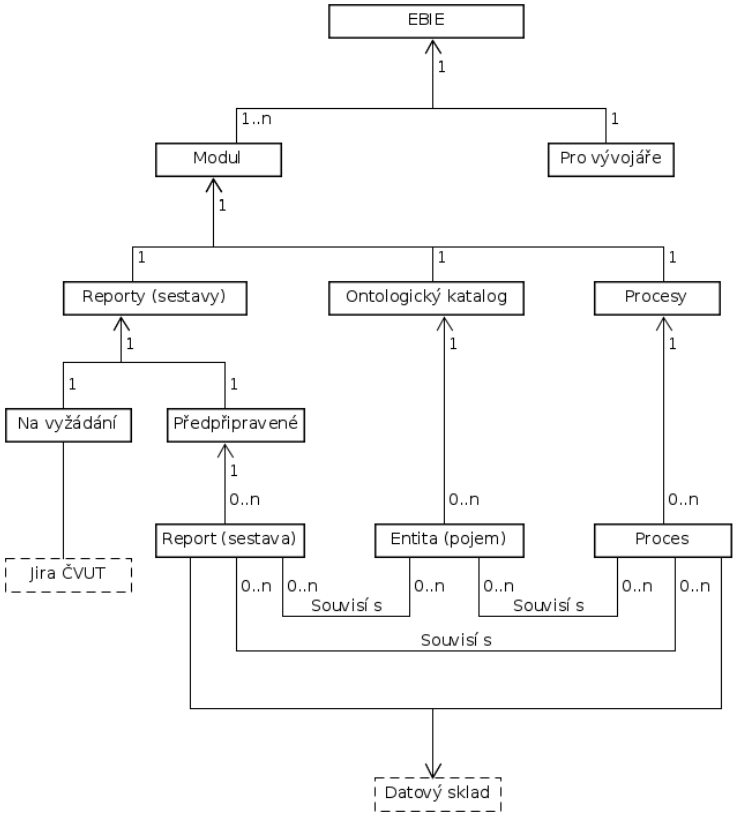
Obrázek 2.1: Snímek obrazovky ze stránky modulu studia, kde můžeme v URL adrese vidět zbytečné informace o verzi

Pokud by došlo k opravení aplikační logiky, místo aktuální URL adresy, např. „../versions/20150830_1204/modules/studium“, by nově byla adresa „../modules/studium“. Je patrné, že tento kratší zápis je mnohem přehlednější a lze z něj snadněji pochopit, co se za ním skrývá za obsah (tedy zobrazení modulu studium).

Navrhuji upravit aplikační logiku portálu EBIE tak, aby zobrazované stránky neobsahovaly číslo verze.

2.2 Datový model

Konceptuální schéma zobrazuje vztahy na nejvyšší úrovni abstraktním pohledem [25]. Na základě svého seznámení s portálem EBIE jsem vytvořil konceptuální datové schéma portálu (použil jsem nástroj Umlet). Schéma jsem vytvořil pomocí class diagramu v UML. Výsledek je přiložen jako obrázek 2.2.



Obrázek 2.2: Konceptuální schéma portálu EBIE

Na schématu lze dobře vidět základní strukturu EBIE. Samotný portál se sestává z jednotlivých modulů a z Dokumentace pro vývojáře. Každý modul poté obsahuje sekce reporty, ontologický katalog a procesy, reporty se dále dělí na reporty na vyžádání a předpřipravené.

Pokud tyto sekce mají obsah, mohou mezi nimi vznikat vazby (např. entita student se účastní procesu Státní závěrečné zkoušky). Zobrazovaný obsah reportů i informace u jednotlivých entit závisí na transformaci dat a metadat z datového skladu (to ještě není implementováno). Reporty na vyžádání se pak odesílají do systému Jira ČVUT.

Ve své práci se dále věnuji vytvoření funkčního prototypu pro transformaci

metadat do entit a řešení jednotlivých požadavků souvisejících s použitím technologií a správou obsahu (viz. sekce 1.3).

2.3 Rozbor a volba technologií

Nyní uvedu návrh na řešení zbylých stanovených požadavků společně s diskusí zvoleného řešení.

2.3.1 Správa obsahu

Pro správu obsahu se nabízí více možností, počínaje zachováním současného stavu, tedy ponecháním obsahu přímo v HTML, konče uložením obsahu do relační databáze. Řešení pouze pomocí HTML nevyhovuje stanoveným požadavkům (viz. 1.3), obsah nelze jednoduše upravovat, jelikož samotný kód není příliš přehledný, pro přidávání obsahu je nutné kromě samotného textu dopisovat i HTML tagy (a je tedy nutná znalost HTML syntaxe) a interní odkazy nejdou tvořit.

Oproti tomu relační databáze nabízí více možností, k samostatně uloženému obsahu lze lépe přistupovat a jednoduše s ním pracovat. Pro portál EBIE však toto řešení není úplně vhodné. Tzv. *database driven websites* se využijí hlavně v situacích, kdy dochází k častým změnám obsahu (jako třeba e-shop), obsah je pro jednotlivé uživatele unikátní nebo třeba pro spravování internetového fóra. [26] Oproti tomu obsah portálu EBIE je téměř neměnný (kromě vzácných drobných úprav nebo oprav textu), uživatelé vidí stejný obsah, pouze nemusí mít přístup ke všem stránkám, a internetové fórum na tomto portálu není a vzhledem k podstatě EBIE ani nebude. Další nevýhoda tohoto řešení je fakt, že opět není možné jednoduše vytvářet interní odkazy.

Je patrné, že tyto možnosti plně nevyhovují požadavkům EBIE. Jako další řešení se nabízí použití značkovacího jazyka Asciidoc s využitím jeho implementace v Ruby, Asciidoctor. Dříve, než tento jazyk popíši, uvedu důvody pro zvolení právě tohoto značkovacího jazyka.

V dnešní době existuje velké množství různých značkovacích jazyků. Patrně nejznámější je Markdown. Tento jazyk však nedosahuje stejných možností jako Asciidoctor (viz. [27]). Další výhoda pro Asciidoctor je implementace v Ruby, která umožňuje jeho snadnou integraci do projektu (jako Ruby gem). Další populární značkovací jazyk v Ruby je jazyk Textile (implementovaný do Ruby jako RedCloth). Tento jazyk však má horší čitelnost než Asciidoc, jeho syntaxe se totiž více blíží HTML syntaxi (viz. [28]). To je však v rozporu s požadavky na čitelnost textu.

Je tedy zřejmé, že nejvhodnější značkovací jazyk je Asciidoc. Text podle formátu Asciidoc je přehledný (viz. ukázka 2.3.1), psaní je jednoduché (na domovské stránce Asciidoctor se přirovnává k psaní e-mailu [29]) a základní syntaxe je snadno osvojitelná. Interní linky lze řešit pohodlně pomocí atributů (viz. [30] a ukázka 2.3.1, kde jsem předvedl použití atributu studium ke snadnému vytvoření odkazu na tento modul) a použití stylů na vzniklou stránku je zaručené umístěním přeloženého souboru do stejného adresáře, ve kterém se nyní stránky nachází.

Uvádím zde ukázku krátkého textu a výsledného HTML kódu po použití programu Asciidoctor (lze zde dobře vidět rozdíl mezi přehledností zápisu ve formátu Asciidoc a HTML, výsledný kód jsem vygeneroval příkazem „asciidoctor -a stylesheet! -s“):

```
:studium: /versions/20150830_1204/modules/studium

Samostatný odstavec

== Nadpis
* Nečíslovaný seznam, první položka
* druhá položka
** první položka na druhé úrovni

link:{studium}[Odkaz na modul studium]
```

Výsledný HTML kód (řádek s odkazem jsem kvůli jeho délce rozdělil na dva řádky):

```
<div class="paragraph">
<p>Samostatný odstavec</p>
</div>
<div class="sect1">
<h2 id="_nadpis">Nadpis</h2>
<div class="sectionbody">
<div class="ulist">
<ul>
<li>
<p>Nečíslovaný seznam, první položka</p>
</li>
<li>
<p>druhá položka</p>
<div class="ulist">
<ul>
<li>
<p>první položka na druhé úrovni</p>
</li>
</ul>
</div>
</li>
</ul>
</div>
<div class="paragraph">
<p><a href="/versions/20150830_1204/modules/studium">Odkaz na
modul studium</a></p>
```

```
</div>
</div>
</div>
```

Z ukázky je vidět, že základní překlad do HTML5 generuje velké množství HTML tagů, které nejsou vždy plně využitelné (např. pro odstavec není potřebné mít samotný odstavec (tag <p>) obalený do). Asciidoctor umožňuje tvorbu vlastního backend řešení nebo úpravu stávajícího. Navrhuji backend upravit a tím zjednodušit výsledný HTML kód.

Vzhled generovaného HTML kódu ale není požadavkem na výběr technologie, z ukázky 2.3.1 a z mých předchozích argumentů je zřejmé, že Asciidoc všechny vytyčené požadavky splňuje.

Na základě důvodů uvedených v této sekci navrhuji převést stávající obsah sekcí ontologický katalog a procesy do formátu Asciidoc, nový obsah vytvářet již v tomto formátu, a pomocí programu Asciidoctor provádět konverzi do formátu HTML.

2.3.2 Tvorba Asciidoc dokumentů

Ačkoli pomocí Asciidocu lze rychle a přehledně psát obsah, určité části HTML kódu není možné generovat přímo. Pro tyto situace existuje v Asciidocu možnost přímého vložení části textu (bez překladu, tímto způsobem lze vložit i části HTML kódu).

Nepřekládaný blok se na začátku i na konci označí čtyřmi znaky plus (++++ (pokud se jedná pouze o část textu na řádku, lze použít 3 znaky(+++)). Navrhuji tímto způsobem řešit místa, která nebude přeložitelná metodami programu Asciidoctor (jedná se např. o obalení stránky do výchozích tříd pro použití stylů).

Pro řešení interních odkazů navrhuji využít atributy. Jejich používání spočívá v tom, že po jejich zadefinování (neboli přiřazení hodnoty, textu) je lze umístit do dokumentu (obalené do závorek {}) a při překladu je místo nich dosazena jejich přiřazená hodnota.

Uživatelské atributy mohou mít hodnotu přiřazenu přímo v dokumentu nebo přes příkaz při překladu (přiřazení při překladu má vyšší prioritu, než přiřazení v dokumentu). Je třeba vytvořit atributy pro odkazy na sekce ontology, procesy a poté na jednotlivé podsekce reportů.

Další podstatná věc je vložení atributů do jednotlivé stránky. Řešení, kdy by každá stránka měla na začátku uvedené všechny atributy, není vhodné. V případě úpravy URL adresy by se musely upravit všechny atributy ve všech souborech. K tomu by došlo i při vytvoření nové podsekce v reportech nebo při potřebě definování nového atributu.

Jako řešení tohoto problému lze využít vložení obsahu souboru do tvořeného dokumentu jednoduchým příkazem „include::soubor.txt[]“. Asciidoctor

před překladem vloží obsah souboru do překládaného dokumentu a poté jej použije.

Navrhuji pro každý modul vytvořit složku lib (na úrovni jednotlivých sekcí, tedy ebie_content/lib), ve které bude soubor logic.adoc obsahující atributy definované pro daný modul. Na začátek každého Asciidoc souboru navrhuji vložit soubor logic.adoc (na základě cesty uložené v atributu lib, jehož hodnota se definuje při překladu, pokud by tedy bylo někdy potřeba složku lib přesunout nebo přejmenovat, bude to jednoduše realizovatelné). Použitím tohoto řešení bude pro každý modul k úpravě všech používaných atributů stačit změna v jediném souboru.

2.3.3 Oddělení obsahu

Vzhledem k tomu, že vývoj aplikační struktury portálu EBIE a jeho obsahu je, až na drobnosti, na sobě nezávislý, vznikl požadavek na oddělení těchto dvou částí, aby byl možný i oddělený vývoj. Jako vhodné řešení se nabízí využít git submodul. Git submoduly umožňují uchovávat Git repositář jako podadresář jiného repositáře Git. Historie úprav obou repositářů je tímto oddělená a vývoj je na sobě nezávislý.

Protože jedním z důvodů oddělení obsahu je potřeba umožnit jeho tvorbu bez nainstalované aplikace EBIE, je třeba vytvořit script na překlad Asciidoc souborů. Nemusí totiž být možné použít řešení implementované v aplikaci EBIE (viz. 2.3.7) a možnost ověřit si výsledek nové práce je pro testování i tvorbu obsahu velmi důležitá.

Navrhuji proto vytvořit jednoduchý shell script, s názvem generate.sh, který převede všechny Asciidoc soubory do formátu HTML, a nový repositář pro obsah EBIE, který by byl k repositáři ebie-source-code připojen jako submodul.

2.3.4 Snadné přidávání obsahu

Nutnost použít speciální bloků pro vkládání HTML kódu na určitá místa dokumentů (viz. sekce 2.3.2) ztěžuje a zpomaluje tvorbu nového obsahu. Aby se tomu předešlo, navrhuji vytvořit výchozí šablony pro sekce ontologický katalog a procesy, uložené do souborů ontology.adoc a processes.adoc ve složce lib v daném modulu, které by obsahovaly všechny nepřekládané části. K tomu navrhuji přidat krátké popisky jednotlivých částí dokumentu. To zjednoduší a zrychlí tvorbu nových dokumentů.

2.3.5 Transformace metadat

Konzultací s Michalem Valentou byl můj úkol upřesněn na transformaci business a technických metadat do tabulek na stránkách jednotlivých entit. Protože datový sklad ČVUT je teprve ve vývoji, domluvili jsme se, že výsledkem

mé práce bude vytvoření funkčního prototypu transformace metadat z vlastní databáze (naplněné metadaty, která jsem na otestování obdržel od vývojového týmu datového skladu), jehož funkčnost budu ilustrovat na jedné entitě.

Tabulky lze v Asciidoc tvořit jednoduchým způsobem:

```
[role="test"]
|===
|sloupec a |sloupec b

|první řádek ve sloupci a |první řádek ve sloupci b
|druhý řádek v~a |druhý řádek v~b
|===
```

Výsledek po překladu je (kvůli délce textu jsem některé řádky zarovnal):

```
<table class="tableblock frame-all grid-all spread test">
<colgroup>
<col style="width: 50%;">
<col style="width: 50%;">
</colgroup>
<thead>
<tr>
<th class="tableblock halign-left valign-top">sloupec a</th>
<th class="tableblock halign-left valign-top">sloupec b</th>
</tr>
</thead>
<tbody>
<tr>
<td class="tableblock halign-left valign-top">
<p class="tableblock">první řádek ve sloupci a</p></td>
<td class="tableblock halign-left valign-top">
<p class="tableblock">první řádek ve sloupci b</p></td>
</tr>
<tr>
<td class="tableblock halign-left valign-top">
<p class="tableblock">druhý řádek v~a</p></td>
<td class="tableblock halign-left valign-top">
<p class="tableblock">druhý řádek v~b</p></td>
</tr>
</tbody>
</table>
```

Asciidoctor navíc umožňuje vkládat obsah vnějších souborů do svého dokumentu, do tabulky lze tedy přidat řádek „include::.tabulka.adoc[]“, který

Asciidoctor při překladu načte a tabulku vygeneruje na základě tohoto souboru (původně jsem uvažoval o vkládání obsahu pomocí atributů, stejným způsobem, jako u interních odkazů. Asciidoctor ale neumožňuje tvořit tabulky pomocí textu z atributů (viz. [31]).

Navrhuji metadata načítat zvlášť pro každou entitu, poté je zpracovat a následně uložit do souborů business.adoc a technical.adoc, které se budou nacházet ve složce lib v daném modulu (název souboru značí typ v něm uložených metadat). Zpracovaná metadata, připravená na vložení do tabulky podle formátu Asciidoc, by měla mít rozložení jednotlivých řádků „|text1 |text2“, kde text1 bude název a text2 popis. Vytvořené soubory navrhuji vždy po použití odstranit, informace v nich uložené již nebudou potřebné.

2.3.6 Asciidoctor HTML5 backend

Na základě návrhu na úpravu generovaného HTML5 z Asciidoc dokumentů je třeba vybrat výchozí backend, který se bude dále upravovat. Asciidoctor nabízí 3 backend řešení pomocí různých technologií. Jedná se o ERB (Embedded Ruby), Slim a Haml. Při výběru řešení jsem bral v potaz fakt, že úpravy pro potřeby EBIE budou pouze drobné.

Všechna 3 řešení jsem otestoval. Backend ERB dosáhl nejlepšího času při testování rychlosti překladu (ke změření času jsem použil Ruby modul benchmark (viz. [32]), časy byly následující: ERB: 17,48s, Slim: 26,55s, Haml: 24,10s) a s jeho používáním jsem neměl žádné problémy. Při testování Slim backendu jsem narazil na problém s řádkováním HTML tagů, kdy jich ve výsledku bylo často vypsáno několik na jeden řádek, což značně snížilo přehlednost a čitelnost. Haml nepřeložil některé soubory kvůli chybě při překladu.

Tyto věci jsou patrně opravitelné, ale na základě toho, že volba Asciidoctor backend řešení hraje pouze malou roli pro projekt EBIE, jsem se rozhodl dále tento problém nezkoumat a zvolit funkční řešení pomocí ERB.

Navrhuji pro každý modul vytvořit složku .html5 (označující backend na překlad do HTML) na úroveň jednotlivých sekcí (tedy ebie_content/.html5).

Přebytečný kód, který navrhuji odstranit (první tři případy lze spatřit v ukázce 2.3.1, poslední v 2.3.5), je:

- Obalování odstavců do <paragraph>
- Obalení nečíslovaných seznamů do a jednotlivých položek do <p>
- Obalení bloku s nadpisem do <sectNum> (Num je číslo podle podle úrovně nadpisu), při úrovni 1 ještě do a doplňování id="nadpis"
- Nepotřebné třídy v tabulce a jejich řádcích a obalení textu do <p>

2.3.7 Transformace obsahu do HTML

V předchozích sekcích mé práce jsem sepsal jednotlivé požadavky a navrhl jejich řešení. Některé požadavky je třeba vyřešit manuálně a pouze jednou (např. úprava adresářů a aplikační logiky, převedení obsahu do formátu Asciidoc, vytvoření šablon pro sekce Ontologický katalog a procesy, úprava backendu, vytvoření a naplnění git repositáře s obsahem EBIE,...), další operace (transformace Asciidoc souborů do HTML, transformace metadat, zobrazení verze aplikace a data poslední změny) bude třeba vykonávat častěji (při updatu obsahu, aktualizaci aplikace, změně v datovém skladu atd.). Bylo by neúnosné to také provádět manuálně.

Je tedy třeba navrhnout automatizované řešení těchto věcí, které musí vykonávat tyto operace:

- Zjištění a uložení data poslední změny (commitu) submodulu ebie-content a verze aplikace EBIE
- Načtení bussiness a technických metadat z datového skladu a následně jejich transformace do ontologického katalogu
- Transformace Asciidoc souborů do HTML

Rails obsahují rozličné „generátory“, které slouží k zrychlení jednoduchých úkonů (např. tvorba nových kontrolerů i se závislostmi atd.). Rails také umožňují tvorbu vlastních generátorů [33]. Použití vytvořeného generátoru je snadné, v kořenovém adresáři se v příkazové řádce zadá příkaz „rails generate NázevGenerátoru“.

Další možností je provádět dané operace při startu serveru. Tím by ale např. při restartu docházelo ke zbytečné časové prodlevě v případě, že nedošlo k žádné změně obsahu.

Navrhuji tedy vytvořit generátor content, který bude možné spustit příkazem „rails generate content“ a který vykoná výše zmíněné operace.

2.3.8 Zobrazení data změny a verze aplikace

Git ukládá časy každého commitu do repositáře, čas posledního poslouží jako čas změny obsahu. V repositáři lze také vytvářet „tagy“, které značí v případě EBIE verzi.

Generátor z předchozího návrhu zjistí čas změny a verzi a uloží je do souborů last_commit_date.txt a version.txt. V souboru _footer.html.erb (který je v ./app/views/layouts/) je definováno, co bude vidět v zápatí stránky. Navrhuji do tohoto souboru nahrát čas změny a verzi, uložené v souborech last_commit_date.txt a version.txt (verze je již v aplikace EBIE zobrazována, ale pouze staticky, neaktualizuje se z repositáře, viz. obrázek 2.1).

2.3.9 Doplnění navržené adresářové struktury

Na základě návrhů z předchozích sekcích navrhuji doplnit návrh adresářové struktury ze sekce 2.1.1 do podoby viditelné na obrázku 2.3.

```
htmls
├── generate.sh ..... script na vygenerování HTML souborů
├── README.adoc ..... readme pro submodul ebie-content
├── last_commit_date.txt ..... soubor s časovým údajem, vytvořený
│   generátorem
├── version.txt ..... soubor s verzí, vytvořený generátorem
├── test ..... adresáře pro uložení přeloženého HTML při testování
│   └── studium
│       ├── ontology .5 *.html ..... vygenerované HTML soubory
│       └── processes .5 *.html ..... vygenerované HTML soubory
└── ebie_content
    ├── info.json ..... informace pro Rails
    └── studium ..... modul
        ├── .html5 ..... backend
        │   └── *.erb ..... jednotlivé soubory backendu
        ├── lib
        │   ├── logic.adoc
        │   ├── business.adoc ..... tento soubor zde bude vždy pouze
        │   │   krátkodobě
        │   ├── technical.adoc ..... tento soubor zde bude vždy pouze
        │   │   krátkodobě
        │   ├── ontology.adoc ..... Šablona pro ontologický katalog
        │   └── processes.adoc ..... Šablona pro procesy
        ├── ontology
        │   ├── info.json ..... informace pro Rails
        │   ├── *.html ..... HTML soubory, které Rails renderují a zobrazí
        │   ├── *.adoc ..... výchozí Asciidoc soubory
        │   └── assets
        │       └── img
        │           └── images ..... obrázky zobrazované v HTML souborech
        ├── processes
        │   ├── info.json ..... Informace pro Rails
        │   ├── *.html ..... HTML soubory, které Rails renderují a zobrazí
        │   ├── *.adoc ..... výchozí Asciidoc soubory
        │   └── assets
        │       └── img
        │           └── images ..... obrázky zobrazované v HTML souborech
        └── reports
            ├── info.json ..... informace pro Rails
            └── dirs ..... složky s jednotlivými kategoriemi pro reporty
                ├── info.json ..... informace pro Rails
                └── dirs ..... složky s obsahující HTML soubory a assets
                    ├── *.html HTML soubory, které Rails renderují a zobrazí
                    ├── info.json ..... informace pro Rails
                    └── assets
                        ├── css
                        │   └── .css ..... stylování pro HTML
                        └── js
                            └── .js ..... Javascript pro HTML
```

Obrázek 2.3: Návrh na novou strukturu adresářů s obsahem EBIE

Kapitola 3

Implementace

V této kapitole popisuji, jak jsem realizoval stanovené požadavky podle návrhů z předchozí kapitoly. Po domluvě s vedoucím mé práce, Michalem Valentou, jsem na přiložené cd umístil odkazy na repositáře, ve kterých se nachází mnou implementované části.

3.1 Změny architektury

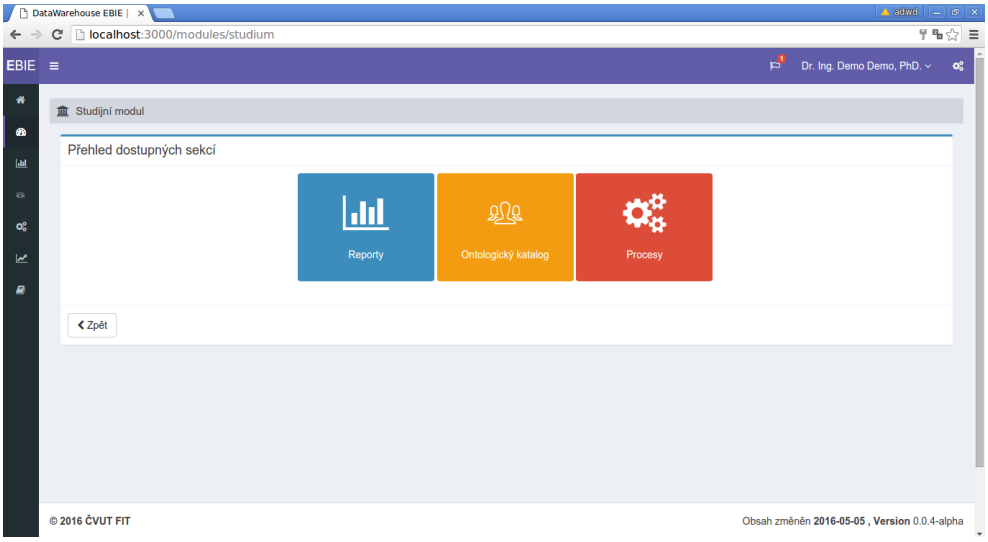
Na základě návrhu (viz. sekce 2.1) jsem provedl změny architektury portálu EBIE.

3.1.1 Zjednodušení adresářové struktury

Adresářovou strukturu jsem zjednodušil a upravil tak, že odpovídá návrhu (viz. obrázek 2.3). Podle toho jsem upravil všechny info.json soubory a v aplikaci jsem změnil název výchozího adresáře pro obsah z Asciidoctor_sync na ebie_content. Nová struktura v sekcích ontologický katalog a procesy umožňuje snadnou tvorbu interních odkazů (původní struktura rozdělení do podsložek nedovolovala jednoduchou implementaci, která by byla snadná a zřejmá na používání).

3.1.2 Upravení URL

Pro odstranění verze z URL adresy bylo třeba odstranit kontroler data_versions controller a s ním spojený view. Tento kontroler zajišťoval zobrazení úvodní stránky portálu (po přihlášení). Rozhodl jsem toto zobrazení upravit a přidat jako akci index do kontroloru bi_modules (kam i logicky patří, protože se jedná o zobrazení modulů). S tím souvisela celková úprava jednotlivých „helperů“ i dalších kontrolerů aplikace v případech, kdy se v jejich návrhu počítalo se zadáním verze.



Obrázek 3.1: Snímek obrazovky z upravené stránky modulu studia

Po provedení těchto úprav a změně souboru routes.rb vypadá URL modulu studium tak, jak jsem navrhl. Na ukázku jsem přiložil obrázek 3.1.

3.2 Správa EBIE

Následující implementace jsem vykonal na základě návrhu v kapitole 2.3.

3.2.1 Správa obsahu

Pro správu obsahu jsem vybral řešení pomocí programu Asciidoctor, který umožňuje převod Asciidoc dokumentů do formátu HTML (viz. sekce 2.3.1). Rozšířil jsem aplikační gemfile o řádek „gem 'asciidoctor'“, abych umožnil používání tohoto programu v aplikaci EBIE. Poté jsem převedl veškerý obsah ontologického katalogu a procesů, v modulu studia, do formátu Asciidoc a vytvořil k těmto sekcí vzorové šablony, která umožní snadné přidávání dalšího obsahu. (viz sekce 2.3.4).

Každý vytvořený Asciidoc soubor obsahuje tento úvod:

```
:nl: pass:[ +]
include::{lib}[]
```

Atribut nl slouží ke vložení prázdného řádku, druhý řádek obsahuje vložení obsahu souboru logic.adoc (na základě atributu lib, který je definován generátorem při překladu, viz. sekce 3.2.4).

Celkem jsem vytvořil 23 entit ontologického katalogu, 3 procesy a 2 šablony (ontology.adoc pro ontologický katalog a procesy.adoc pro procesy) ve

formátu Asciidoc (tedy celý obsah portálu EBIE kromě reportů). Šablony jsem otestoval při převodu obsahu v HTML do formátu Asciidoc, tím jsem navíc všechny entity doplnil o části procesy entity, reporty entity, entitní propojení a o tabulky na zobrazení metadat.

Jako ukázku uvádím text, který vygeneruje popis entity předmět (pro přehledné zobrazení jsem musel některé řádky rozdělit na více částí, to ale nemění vygenerovanou stránku. Výsledek odpovídá části obrázku 1.5):

```
Předmět je základním výukovým modulem
link:{ontology}studijniplan.html[studijního plánu].
Předmět je charakterizován počtem výukových hodin,
formou výuky, způsobem zakončení a počtem kreditů.

{nl}

==== Způsob zakončení předmětu

* Udělením zápočtu
* Udělením klasifikovaného zápočtu
* Vykonáním zkoušky
* Udělením zápočtu a vykonáním zkoušky

U~předmětu, kde je studijním plánem předepsán zápočet i zkouška,
je udělení zápočtu podmínkou
pro konání zkoušky z~příslušného předmětu.
```

Pro lepší výsledný HTML kód jsem upravil výchozí Asciidoctor backend, generující HTML. Na ukázku jsem vybral text již použitý v mé práci. Výsledný rozdíl je patrný na první pohled (uvedeno v sekci 3.2.3).

3.2.2 Git submodul ebie-content

Na svém školním účtu na „gitlabu“ jsem vytvořil nový repositář s názvem ebie-content, do kterého jsem nahrál mnou upravenou verzi obsahu EBIE. Struktura repositáře odpovídá návrhu (viz. obrázek 2.3). Vzhledem k tomu, že data na portálu EBIE jsou z velké části soukromá, jsem repositář založil jako skrytý a umožnil do něj přístup vývojovému týmu EBIE.

Dále jsem provedl konfiguraci repositáře ebie-source-code, který obsahuje aplikaci EBIE: odstranil jsem obsah adresáře htmls a příkazem „git submodule add https://gitlab.fit.cvut.cz/hajcidav/ebie-content app/assets/htmls/“ jsem přidal submodul.

Před použitím je (vždy po stáhnutí repositáře) nutné submodul inicializovat (příkazem „git submodule init“, který inicializuje všechny submoduly) a stáhnout aktuální obsah (příkaz „git submodule update –remote“). Po pro-

vedení těchto kroků odpovídá adresářová struktura struktuře popsané na obrázku 2.3.

Tento popis jsem přidal i do souboru README.adoc v repositáři ebie-source-code, který obsahuje návod na instalaci a spuštění aplikace.

Do repositáře ebie-content jsem také přidal jednoduchý skript, který vygeneruje výsledné HTML soubory. To umožní rychlou kontrolu přidaného obsahu (bez nutnosti instalovat aplikaci).

3.2.3 Upravený backend

Backend jsem upravil podle návrhu v sekci 2.3.6 a umístil jej do složky .html. Jeho funkčnost ukáži nejprve vygenerováním HTML ze stejného textu, jako v ukázce 2.3.1:

```
Samostatný odstavec

== Nadpis
* Nečíslovaný seznam, první položka
* druhá položka
** první položka na druhé úrovni

http://asciidoctor.org[Stránka Asciidoctor]
```

Výsledek je:

```
<p>Samostatný odstavec</p>

<h2>Nadpis</h2>

<ul>
<li>
Nečíslovaný seznam, první položka
</li>
<li>
druhá položka

<ul>
<li>
první položka na druhé úrovni
</li>
</ul>
</li>
</ul>
```

```
<p><a href="http://asciidoctor.org">stránka Asciidoctor</a></p>
```

Změny jsou patrné i ve vygenerovaná tabulce (na porovnání ukázka 2.3.5):

```
[role="test"]
|===
|sloupec a |sloupec b

|první řádek ve sloupci a |první řádek ve sloupci b
|druhý řádek v~a |druhý řádek v~b
|===
```

A výsledek:

```
<table class="test" style="width: 100%;">
<colgroup>
<col style="width: 50%;">
<col style="width: 50%;">
</colgroup>
<thead>
<tr>
<th>sloupec a</th>
<th>sloupec b</th>
</tr>
</thead>
<tbody>
<tr>
<td>první řádek ve sloupci a</td>
<td>první řádek ve sloupci b</td>
</tr>
<tr>
<td>druhý řádek v~a</td>
<td>druhý řádek v~b</td>
</tr>
</tbody>
</table>
```

Je vidět, že došlo k zpřehlednění, zjednodušení a zkrácení výsledného kódu při zachování vzhledu vygenerované stránky.

3.2.4 Generátor obsahu

Pro automatizované generování obsahu a uložení času změny (posledního commitu) submodulu a aktuální verze jsem vytvořil Rails generátor s názvem content (v aplikaci ebie jsem v adresáři ./lib vytvořil adresář content, v něm pak

soubory USAGE (obsahuje stručný popis generátoru) a content_generator.rb (který obsahuje vlastní generátor)).

Generátor jsem vytvořil tak, aby byl připravený na další rozšiřování obsahu portálu EBIE a aby byl jeho obsah srozumitelný pro další vývojáře. Uvedu zde ukázky podstatných částí kódu (kvůli přílišné délce kódu jsem na některých místech smazal mezery, změnil názvy proměnných nebo přesunul část příkazu na další řádek. Vlastní logika generátoru zůstala zachována).

Pro lepší pochopení ukázek zde ještě stručně popíši některé aspekty syntaxe jazyka Ruby. Text za znakem „#“ je komentář. Ruby nevyžaduje psaní závorek. Konec cyklů, podmínek atd. se zapisuje výrazem „end“.

Následující cyklus postupně vybere všechny vytvořené moduly (v tuto chvíli pouze studium, při vytvořeních dalších modulů není třeba nic měnit):

```
#Iterate through all ebie modules
  Dir.foreach ebie_content_path do |ebie_module_name|
    next if ebie_module_name == '.' or ebie_module_name == '..'

    #modules are the only directories in ebie-content folder
    if File.directory? ebie_content_path + ebie_module_name
```

V poli adoc_sections jsou uložené názvy jednotlivých sekcí, ve kterých se nachází Asciidoc soubory. Pokud by v budoucnu vznikla další sekce, stačí přidat další položku do sekce, když se v modulu některá sekce nenachází, program pokračuje dál:

```
#Iterate through module's sections
  adoc_sections.each do |section|

    #Select all .adoc files
    Dir.glob(module_path + section + "/*.adoc*") do |adoc_file|
```

Z datového skladu (v mém případě pouze z jednoduché testovací databáze) je načtena požadovaná tabulka do proměnné meta. Poté, co jsou metadata zpracována do potřebného formátu, jsou uložena do souborů podle jejich typu. Tyto soubory jsou po použití smazány (v proměnné meta je načtena z databáze tabulka s metadaty):

```
#Load metadata in desired format
meta.each do |row|
  b+="| "+(row['O_název']||" ")+" |"+(row['O_definice']||"")
  t+="| "+(row['sloupec']||"")+" | "+(row['datový typ']||"")
end

#Save loaded metadata to files
```

```
File.open(b_metadata, 'w') { |file| file.write business }
File.open(t_metadata, 'w') { |file| file.write technicke }

...

#Delete metadata files
if File.exists? business_metadata_file
  File.delete business_metadata_file
end

if File.exists? technical_metadata_file
  File.delete technical_metadata_file
end
```

Transformace Asciidoc souborů do HTML a následné vypsání jména přeloženého souboru:

```
#Render adoc file and print its name
system "asciidoctor -q -s -a lib=" + adoc_logic_file + " -T " \
 + backend_dir + " -a stylesheet! " + adoc_file
puts "rendered " + ebie_module_name + "/" + file_name + ".html"
```

Příkaz system znamená vykonání příkazu v subshellu [34]. Jednotlivé přepínače značí:

- -q: tichý překlad, při které se nevypisují varování (to je nutné proto, že Asciidoctor požaduje vzestupné používání nadpisů - h1, h2,... V EBIE jsou se ale vyskytují nadpisy mimo pořadí).
- -s: zamezí vložení hlavičky, kterou jinak Asciidoctor automaticky generuje.
- -a lib= adoc_logic_file_path: vložení cestu k souboru adoc_logic_file do atributu lib, pomocí něhož se do každého dokumentu EBIE načítá obsah souboru logic.adoc.
- -T backend_dir: příkaz použít upravený backend, jehož cesta je uložena v proměnné v backend_dir.
- -a stylesheet!: zakazuje použití výchozího Asciidoctor stylování.
- adoc_file: obsahuje cestu k překládanému Asciidoc souboru.

Informace o čase poslední změny obsahu a o verzi aplikace jsou získány a uloženy do souborů:

```
#Get time of last commit in submodule ebie-content as
#year-month-day
  pwd = Dir.pwd
  Dir.chdir submodule_path
  lastCommitDate=‘git log -1 --format=%ci‘.partition(’ ’).first
  Dir.chdir pwd

  #Create and fill files with date of last commit and with
  #the application version
  File.open("app/assets/htmls/last_commit_date.txt", "w")
  { |file| file.write lastCommitDate}
  File.open("app/assets/htmls/version.txt", "w")
  { |file| file.write ‘git describe --abbrev=0 --tags‘}
```

Příkazy obalené obrácenými apostrofy vykoná subshell a jako výsledek vrátí stdout (standardní výstup).
„git log -1 –format=%ci“ je git příkaz, který zobrazí 1. řádek logu repositáře jako čas commitu ve formátu podle normy ISO 8601 [35].
Příkaz „git describe –abbrev=0 –tags“ vrátí poslední uložený tag repositáře.

3.2.5 Zobrazení metadat

V předchozí sekci jsem ukázal prototyp transformace business a technických metadat do portálu EBIE. Zkušební databázi jsem vytvořil v databázovém systému PostgreSQL (aby lépe reflektovala datový sklad, který je také tvořen pomocí PostgreSQL).

Takto jsem vytvořil tabulku pro technická metadata (business metadata jsou vložena obdobným způsobem):

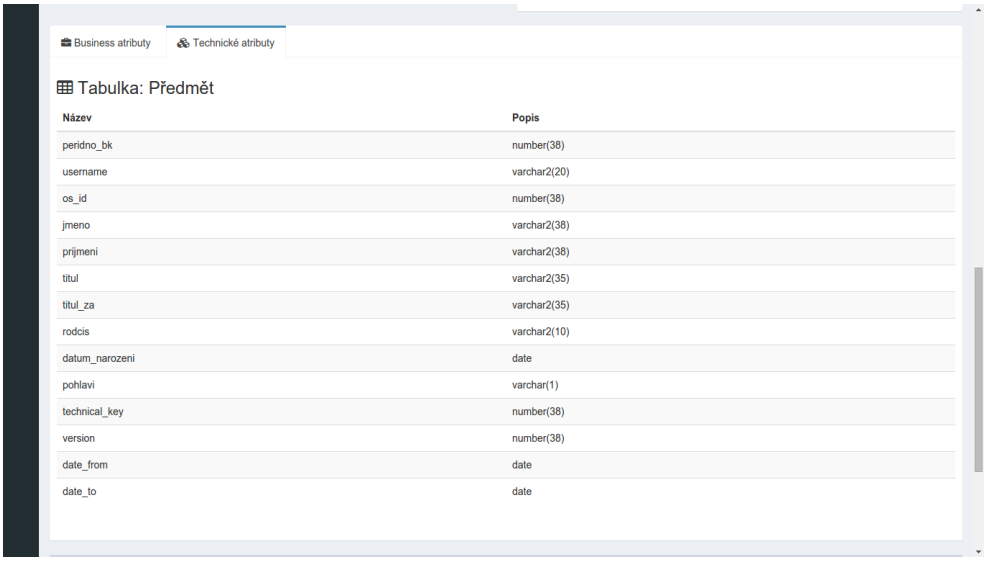
```
[role="table table-hover table-striped"]
|===
|Název |Popis

include::../lib/technical.adoc[]
|===
```

Na obrázku 3.2 lze vidět, jak se do tabulky pro technická metadata entity předmět (viz. obrázek 1.5) doplnil obsah databáze. Stejně tak byla doplněna i business metadata.

3.2.6 Zobrazení data změny a verze aplikace

V sekci 3.2.4 jsem ukázal, jakým způsobem jsou vytvořeny a naplněny soubory last_commit_date.txt a version.txt. Do souboru _footer.html.erb jsem přidal část (v ukázce jsem kvůli délce zkrátil cestu k souboru version.txt):



Obrázek 3.2: Ukázka tabulky s načtenými technickými metadaty u entity předmět

```
Obsah změněn
<strong>
<%= File.read("app/assets/htmls/last_commit_date.txt") %>
</strong>
<strong>, Version</strong> <%= File.read(".../htmls/version.txt") %>
```

Je zde vidět syntaxe ERB, „<%= ... %>“ značí vložení výsledku vykonaného příkazu do stránky. Vygenerované soubory jsou načteny a zobrazeny v zápatí stránky, jak lze vidět na obrázku 3.1.

Závěr

Ve své práci jsem se, v souladu se zadáním, seznámil s projektem EBIE, vytvořil konceptuální datové schéma a dokumentoval jej. Dále jsem se seznámil s obsahovými požadavky projektu EBIE, které byly:

- Popis, diskuze a úprava architektury obsahu EBIE
- Nalezení vhodné technologie pro správu obsahu EBIE, která by splňovala tyto požadavky:
  - Možnost jednoduché úpravy stávajícího obsahu
  - Jednoduché přidávání dalšího obsahu
  - Přehlednost
  - Možnost tvorby interních odkazů
  - Použití výchozích stylů EBIE
- Oddělení obsahu EBIE od vlastní aplikační logiky
- Zpřehlednění adresářů s obsahem a zjednodušení URL pro přístup
- Zobrazení času změny obsahu a aktuální verze aplikace
- Návrh a začlenění business a technických metadat datového skladu do obsahu EBIE

Pro všechny tyto požadavky jsem navrhl a implementoval řešení. Pro správu obsahu jsem zvolil technologii Asciidoctor, kterou jsem ozkoušel a upravil. Navíc jsem všechen relevantní obsah (tedy obsah ontologického katalogu a procesů) převedl do formátu Asciidoc a vytvořil jsem šablony pro snadné přidávání dalšího obsahu do portálu EBIE.

Obsah jsem oddělil od aplikace za použití technologie git submodule.

Dále jsem se seznámil se strukturou metadat v datových skladech a po analýze a návrhu jsem (na základě konzultace se svým vedoucím, Michalem

Valentou) vytvořil prototyp řešení transformace business a technických metadat z datového skladu do portálu EBIE.

Pro převod souborů ve formátu Asciidoc do HTML, transformaci metadat a uložení času změny obsahu a verze aplikace jsem vytvořil rails generátor, který tyto operace automaticky vykoná.

Další rozvoj

Jedním ze směrů dalšího vývoje portálu EBIE je integrace vektorových obrázků datového modelu. Poté, co bude systém nasazen, se budou ukazovat nové potřeby, které bude třeba řešit.

Literatura

[1] Kopecký, M.: *Návrh a implementace klientské části systému EBIE*. Bakalářská práce, České vysoké učení v Praze, Fakulta informačních technologií, Praha, 2016.

[2] Černý, C.: *Návrh a implementace integrace služeb do projektu EBIE*. Bakalářská práce, České vysoké učení v Praze, Fakulta informačních technologií, Praha, 2016.

[3] Novotný, O.; Pour, J.; Slánský, D.: *Business intelligence: jak využít bohatství ve vašich datech*. Praha: Grada, první vydání, 2005, ISBN 80-247-1094-3.

[4] Rouse, M.: What is Business Intelligence (BI)? *SearchDataManagement*, Oct 2014. Dostupné z: `http://searchdatamanagement.techtarget.com/definition/business-intelligence`

[5] Watson, H. J.; Wixom, B. H.: The Current State of Business Intelligence. *Computer*, ročník 40, č. 9, Sept 2007: s. 96–99, ISSN 0018-9162, doi: 10.1109/MC.2007.331.

[6] Pergl, R.: BI-OMO - Přednáška č.1 - Úvod do konceptuálního modelování. Říjen 2015, [cit. 2016-04-18].

[7] Jirovský, V.: *Konceptuální analýza datových domén Studium a Hodnocení kvality výuky s ohledem na datovou čistotu*. Diplomová práce, České vysoké učení v Praze, Fakulta informačních technologií, Praha, 2016.

[8] About Ruby. *Ruby-lang.org*, 2010. Dostupné z: `https://www.ruby-lang.org/en/about/`

[9] RubyGems guides. *Rubygems.org*, 2015. Dostupné z: `http://guides.rubygems.org/`

[10] Git. 2015. Dostupné z: https://git-scm.com/

[11] Asciidoctor. *Asciidoctor.org*, 2016. Dostupné z: http://asciidoctor.org/

[12] What is UML. *Uml.org*, July 2005. Dostupné z: http://www.uml.org/what-is-uml.htm

[13] Model-view-controller (MVC). *Http://whatis.techtarget.com/*, 2015. Dostupné z: http://whatis.techtarget.com/definition/model-view-controller-MVC

[14] Chapter 1 From zero to deploy. *Railstutorial.org*, 2016. Dostupné z: https://www.railstutorial.org/book/beginning

[15] ActionController::Helpres. *api.rubyonrails.org*, 2016. Dostupné z: http://api.rubyonrails.org/classes/ActionController/Helpers.html

[16] Action Mailer Basics - Ruby on Rails Guides. *guides.rubyonrails.org*, 2016. Dostupné z: http://guides.rubyonrails.org/action_mailer_basics.html

[17] Rails Routing from the Outside in - Ruby on Rails Guides. *guides.rubyonrails.org*, 2016. Dostupné z: http://guides.rubyonrails.org/routing.html

[18] Bundler: the best way to manage a Ruby application's gems. *Bundler.io*, 2016. Dostupné z: http://bundler.io/

[19] Creating and Customizing Rails generators & Templates - Ruby on Rails Guides. *Guides.rubyonrails.org*, 2016. Dostupné z: http://guides.rubyonrails.org/action_view_overview.html

[20] JSON. 2016. Dostupné z: http://www.json.org/

[21] Sklenák, V.: *Data, informace, znalosti a Internet.* Praha: C.H. Beck, první vydání, 2001, ISBN 80-717-9409-0.

[22] Kimball, R.; Ross, M.: *The data warehouse lifecycle toolkit: expert methods for designing, developing, and deploying data warehouses.* New York: Wiley, druhé vydání, c1998, ISBN 04-712-5547-5.

[23] Krejčí, J.: *Metadata pro datový sklad fakulty.* Bakalářská práce, České vysoké učení v Praze, Fakulta informačních technologií, Praha, 2015.

[24] Kimball, J. C. R.: *The data warehouse ETL toolkit practical techniques for extracting, cleaning, conforming, and delivering data.* Indianapolis, IN: Wiley, 2004, ISBN 07-645-7923-1.

[25] Rouse, M.: Data modeling. *SearchDataManagement*, Oct 2014. Dostupné z: http://searchdatamanagement.techtarget.com/definition/data-modeling

[26] Data-driven website. *COMPUTERBUSINESSRESEARCH.COM*, 2016. Dostupné z: http://www.computerbusinessresearch.com/Home/database/data-driven-website

[27] Setting Attributes on a Document. *Asciidoctor.org*, 2016. Dostupné z: http://asciidoctor.org/docs/user-manual/#compared-to-markdown

[28] A Textile Reference. *Redcloth.org*, 2016. Dostupné z: http://redcloth.org/hobix.com/textile/

[29] What is Asciidoc? Why do we need it? *Asciidoctor.org*, 2016. Dostupné z: http://asciidoctor.org/docs/what-is-asciidoc/#the-zen-of-writing-asciidoc

[30] Asciidoctor User Manual. *Asciidoctor.org*, 2016. Dostupné z: http://asciidoctor.org/docs/user-manual/

[31] Setting Attributes on a Document. *Asciidoctor.org*, 2016. Dostupné z: http://asciidoctor.org/docs/user-manual/#setting-attributes-on-a-document

[32] Benchmark. *Ruby-doc.org*, 2015. Dostupné z: http://ruby-doc.org/stdlib-1.9.3/libdoc/benchmark/rdoc/Benchmark.html

[33] Creating and Customizing Rails generators & Templates - Ruby on Rails Guides. *Guides.rubyonrails.org*, 2016. Dostupné z: http://guides.rubyonrails.org/generators.html

[34] Kernel. *Ruby-doc.org*, 2015. Dostupné z: http://ruby-doc.org/core-2.3.1/Kernel.html#method-i-system

[35] Git - git-log Documentation. 2015. Dostupné z: https://git-scm.com/docs/git-log

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

BI Business Intelligence

BPMN Business Process Model and Notation.

DEMO Desing & Engineering Methodology for Organisations

EBIE Extended Business Intelligence Encyclopedia

ERB Embedded RuBy

ETL Extract-Transformation-Load

HTML HyperText Markup Language

JSON JavaScript Object Notation

MVC Model-View-Controller

Rails Ruby on Rails

UML Unified Modeling Language

URL Uniform Resource Locator

Příloha **B**

Obsah přiloženého CD

- `BP_Hajčiar_David_2016.pdf` ..... text práce ve formátu PDF
- `BP_Hajčiar_David_2016.tex` ..... text práce ve formátu LaTeX
- `links.txt` ..... textový soubor obsahující odkazy na repositáře