FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Migrace online časopisu Mensa
Student:	Dominik Křížek
Vedoucí:	Ing. Tomáš Nováček
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

Web časopisu Mensy ČR je založen na redakčním systému MultiCMS, ale Mensa by jej ráda přemigrovala na CMS založené na systému WordPress. Protože je web již dlouhou dobu v provozu, je potřeba přemigrovat data z jednoho systému do druhého.

Cíl práce:
1) Analyzujte aktuální web časopisu Mensa ČR založený na MultiCMS.
2) Diskutujte možnost migrace databáze a přidružených souborů z MultiCMS na WordPress.
3) Implementujte řešení na migraci časopisu Mensa ČR na novou WordPress platformu.
4) Migrace musí být spustitelná vícekrát.
5) Migrace musí zaručit co nejmenší ztrátu informací.
6) Migrace musí kromě článků přemigrovat i uživatele.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 27. září 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Migrace online časopisu Mensa

Dominik Křížek

Katedra softwarového inženýrství
Vedoucí práce: Ing. Nováček Tomáš

8. dubna 2021

Poděkování

Rád bych poděkoval svému vedoucímu Tomáši Nováčkovi za skvělé vedení a korektury textu. Zároveň bych rád poděkoval Mense České republiky za příležitost pracovat na zajímavém projektu, Lucii Kopasové za motivaci a rodině za podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při 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. Dále prohlašuji, že jsem s Českým vysokým učením technickým v Praze uzavřel dohodu, na jejímž základě se ČVUT vzdalo práva na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona. Tato skutečnost nemá vliv na ustanovení § 47b zákona č. 111/1998 Sb., o vysokých školách, ve znění pozdějších předpisů.

V Praze dne 8. dubna 2021 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Křížek, Dominik. *Migrace online časopisu Mensa*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Naším cílem v této práci je vytvoření aplikace pro migraci obsahu ze systému MultiCMS do platformy WordPress.

Provádíme analýzu obou systémů pro správu obsahu, rozebíráme různé způsoby migrace dat, a dále představujeme několik nástrojů, kterými by se dalo dosáhnout cíle. Nakonec popisujeme analýzu samotného problému migrace, tedy jednotlivé entity, které musíme přemigrovat.

Dále popisujeme implementaci migrační aplikace ve frameworku Symfony, a to části ovladatelné přes konzoli, a části ovladatelné přes grafické uživatelské prostředí, které bude přístupné přes webový prohlížeč.

V závěru uvádíme další vývoj aplikace v rámci budoucí spolupráce s klientem a možné využití aplikace pro další vývojáře.

Klíčová slova migrace, systém pro správu obsahu, WordPress, MultiCMS, Symfony, databáze

Abstract

Our goal in this work is to create an application for content migration from the MultiCMS system to the WordPress platform.

We analyze both content management systems, discuss different ways of migrating data, and present several tools that could achieve the goal. Finally, we describe the analysis of the migration problem itself, like the individual entities that we need to migrate.

We also describe the implementation of the migration application in the Symfony framework, namely the part that can be controlled via the console and the part that can be controlled via the graphical user interface, which will be accessible via a web browser.

In the Conclusion, we present further development of the application within the future cooperation with the client and possible use of the application for other developers.

Keywords migration, content management system, WordPress, MultiCMS, Symfony, database

Obsah

Seznam obrázků

Úvod

Byly doby, kdy člověk, který chtěl vytvořit webovou stránku, musel umět programovat. Zvláště pak když chtěl vytvořit sofistikovaný redakční systém, přes který mohl vydávat články z pohodlí webového editoru. Dnes již tomu tak dávno není a na trhu se nabízí mnoho hotových redakčních systémů, které stačí stáhnout na server, a po pár krocích instalace jej můžeme začít používat.

Webové trendy se v čase mění, stránky vypadají jinak než před deseti lety, ovládají se jinak, můžeme je prohlížet na mobilu a redaktoři jsou zvyklí pracovat s modernějšími nástroji. I to je případ této práce, ve které Mensa České republiky chce převést svůj časopis na modernější redakční systém, aby jeho stránky dostály dnešním webovým standardům a redaktoři mohli pracovat efektivněji.

Novým zvoleným redakčním systémem má být platforma WordPress, která se časem osvědčila u spousty zákazníků a je stále oblíbenou volbou. Díky jeho velké popularitě mají vývojáři k dispozici kvalitní dokumentaci, spousty vyřešených problémů a nástrojů, se kterými mohou upravit systém natolik, aby vyhovoval přáním zákazníka.

Abychom mohli klientovi převést původní obsah časopisu na novou platformu, budeme muset využít databázových migrací, jejichž problematiku si popíšeme na následujících stránkách.

Kapitola 1

Cíl práce

Cílem je vytvořit aplikaci na migraci dat webu časopisu Mensa, založeného na systému MultiCMS, do nového systému založeném na platformě WordPress. To znamená přemístit články, rubriky, soubory včetně obrázků a uživatele. U migrace nesmí dojít k žádné ztrátě informací a musí existovat možnost ji spustit vícekrát pro nová data na původním systému.

V první řadě je tedy potřeba zjistit, jak fungují oba dané redakční systémy, tedy MultiCMS a WordPress. Dále analyzujeme možnosti migrace a vyzkoušíme několik již připravených nástrojů, které by mohly dosáhnout uvedeného cíle.

Dále implementujeme aplikaci, která nám umožní migrovat obsah z platformy MultiCMS na platformu WordPress, a nakonec po migraci dat připravíme finální web na produkčním prostředí a předáme jej zákazníkovi.

Kapitola 2

Analýza

2.1 Analýza systémů pro správu obsahu

V následujících sekcích popíšeme, co to vlastně systém pro správu obsahu je, jaké jsou jeho základní nástroje a proč bychom ho měli používat.

2.1.1 Co je to CMS

Content management system neboli systém pro správu obsahu je software, pomocí kterého spravujeme obsah dokumentů [2]. Běžně to bývají administrační rozhraní pro webové aplikace, typicky pak webové stránky se články.

K základním funkcím systému patří především správa obsahu a správa uživatelů.

2.1.2 Správa obsahu

Do obsahu patří typicky objekty jako články, kategorie, stránky, soubory, obrázky a galerie. Systém nám umožňuje tyto objekty tvořit, modifikovat, mazat a nastavovat jim různé vlastnosti. CMS bývají velice sofistikované, takže například pro články, které jsou pro návštěvníka zajímavé především titulem a samotným obsahem textu, můžeme ještě nastavit kdy má být článek publikován, v jaké kategorii se má nacházet, vlastní URL adresu, pozicování na hlavní stránce a mnoho dalšího.

Pro tvorbu samotného obsahu článku (nebo i jiných objektů) se typicky používá WYSIWYG editor. To je editor, ve kterém lze formátování textu ovládat grafickým uživatelským rozhraním, takže je vhodný pro uživatele, kteří jsou zvyklý psát např. v Microsoft Word a podobných aplikacích. Většinou podporuje i vkládání tabulek, obrázků a dalších prvků.

2.1.3 Správa uživatelů

Za obsah na stránce musí mít někdo zodpovědnost a ta bývá u větších projektů rozřazena mezi více lidí hierarchickým způsobem. Tyto systémy s touto realitou počítají, a tak nabízí i správu uživatelů. Uživatelé se přihlašují do systému svými přihlašovacími údaji a je jim přiřazena nějaká role. Role můžeme v systému vytvářet, modifikovat a přiřazovat jim různá práva. Role s největším rozsahem práv patří typicky administrátorovi. Další role, např. pro článkové weby, pak bývají redaktor, editor, šéfredaktor.

2.1.4 Proč používat CMS

Pokud máme nebo plánujeme vytvářet aplikaci, pro kterou potřebujeme spravovat obsah, je ideální využít některý z nabízených CMS. Je to již hotové řešení, takže nemusíme vyvíjet systém na míru, který by mohl zabrat několik měsíců a byl by mnohem nákladnější. U takto hotového systému můžeme počítat s vysokým zabezpečením, optimalizací, podporou a manuály k použití. Pokud potřebujeme nějakou funkcionalitu navíc, může se najít (nebo vyvinout vlastní) plugin.

Pokud však chceme systém, který neodpovídá takovému chování, je lepší použít jiné prostředky, nebo pořídit systém na míru. Například na e-shopy, vyhledávače cen a rezervační systémy se běžné CMS moc nehodí.

2.2 WordPress

WordPress je open source CMS [3]. Pohání 64 % všech webů založených na CMS řešení a 40 % webů celkově na internetu [4]. Kromě klasických CMS nástrojů podporuje přizpůsobení vzhledu, SEO, responzivní mobilní stránky a rozšiřitelnost pomocí pluginů. Je lokalizován do češtiny a díky jeho popularitě lze najít i spoustu návodů.

2.2.1 Vlastnosti WordPressu

WordPress nabízí plno možností jak můžeme vytvořit svůj web k obrazu svému. Níže popíšeme jeho klíčové vlastnosti, a seznámíme se tak s jeho možnostmi.

2.2.1.1 Obsah

Samotný obsah lze psát přes editor Gutenberg, který je k dispozici od verze 5.0 a nahradil tak bývalý editor TinyMCE [5]. Ten nám umožňuje tvořit obsah a psát články za pomocí vkládání bloků. Najdeme v něm už spoustu základních bloků jako odstavec, nadpis, obrázek, citace, galerie, video, odrážkový seznam apod. Další bloky lze získat pomocí pluginů, nebo si můžeme vytvořit vlastní.

Celý proces tvoření obsahu lze provést v administraci WordPress. Článek můžeme vytvořit, nechat si zkontrolovat gramatiku editorem, schválit od šéfredaktora a doladit od editora. Při psaní WordPress automaticky ukládá rozdělanou práci, nemusíme se tedy starat o dodatečné zálohování. Článek si můžeme prohlédnout, jak bude vypadat na webu, aniž bychom ho publikovali.

Vkládání mediálního obsahu je intuitivní. Můžeme ho nahrát klasicky přes procházení našeho počítače, popř. je zde podporován i drag-and-drop.

Publikování článku pak můžeme provést jediným tlačítkem, nebo ho můžeme naplánovat na nějaké konkrétní datum.

2.2.1.2 Administrace

Do administrace se uživatel připojí prostřednictvím přihlašovací stránky, kde musí vyplnit své údaje. V administraci pak nalezne nastavení a nástroje, na která má přidělena práva. Které práva má přidělena určuje jeho role. Správa práv pro jednotlivé role se bohužel nedá v základní instalaci WordPress nastavit, ani nelze zakládat nové role, pro tuto možnost bychom museli využít některého z nabízených pluginů.

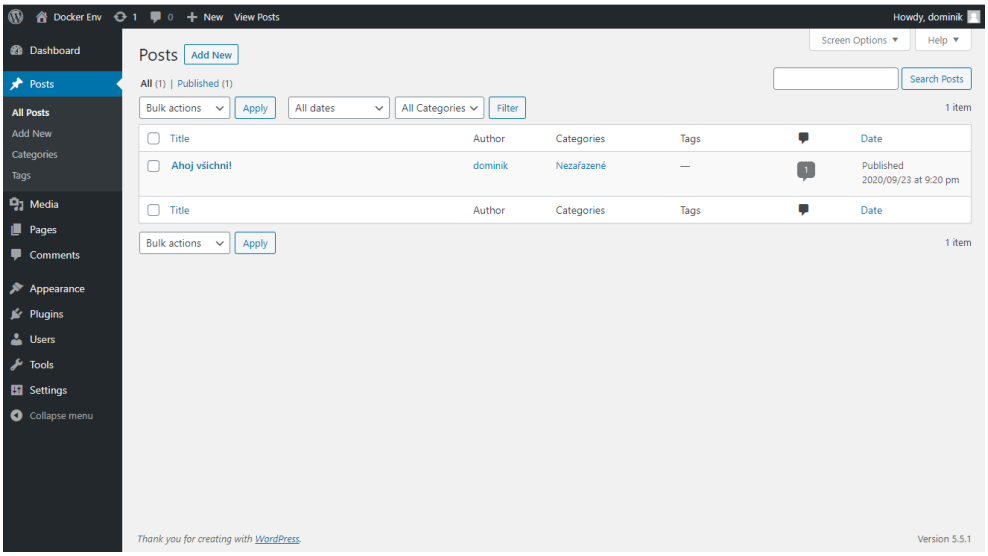
Přednastavené role jsou následující:

- Super Admin – Práva na všechno, včetně přístupu k síťové administraci,
- Administrator – Práva na všechno v rámci jedné stránky,
- Editor – Může publikovat a spravovat články i ostatních uživatelů,
- Author – Může publikovat a spravovat jen své články,
- Contributor – Může psát a spravovat své články, ale nemůže je publikovat,
- Subscriber – Může nastavovat jen svůj profil.

2.2.1.3 Rozšiřitelnost

Díky početné komunitě, která už vyřešila spoustu častých problémů, se jako základní možnost rozšíření nabízí přes 50 000 pluginů [6]. Ty mohou být jak zdarma, tak placené. WordPress nabízí rozsáhlou API a dokumentaci k ní, takže si můžeme vytvořit i vlastní pluginy, přímo pro naše specifické požadavky.

Dále můžeme využít protokol XML-RPC, který nám umožní propojit náš WordPress s jiným informačním systémem, a umožnit tak jejich vzájemnou komunikaci.



Obrázek 2.1: Náhled prostředí administrace WordPress

2.2.1.4 Vzhled

Pro vzhled máme k dispozici připravené základní šablony, které si můžeme rozšířit podle potřeby. Šablony se skládají z více souborů, které mají vlastní hierarchii, a tak se s nimi pracuje relativně snadno z vývojářského hlediska. Šablony lze sehnat již hotové, ať už placené nebo zdarma.

2.2.1.5 Bezpečnost

WordPress podporuje dnes již standardní HTTPS protokol, ba ho dokonce vyžaduje, pokud tedy chceme náš web zveřejnit na internetu, musíme si zařídit certifikát. Veškeré uživatelské vstupy, jako formuláře, jsou ošetřeny, a jsme tak chráněni i před SQL injection útoky. Dále máme možnost si nastavit dvoufázové ověřování pro přihlášení do administrace.

Také nabízí podporu pro ochranu osobních údajů (kompatibilní s GDPR), a lze tak nastavit informování uživatele a sběr jeho souhlasu. Dále nabízí nástroje pro export osobních údajů, kdyby si to některá strana vyžádala.

Abychom zachovali nejvyšší možnou bezpečnost, vyplatí se WordPress pravidelně aktualizovat, naštěstí ho můžeme aktualizovat přímo v administračním prostředí.

2.2.1.6 Komunita

WordPress umožňuje správu diskuzí, a nabízí tak návštěvníkům možnost přispívat pod jednotlivé stránky. Diskuze se dají zapnout nebo vypnout jak pro jednotlivé články, tak i pro celý web. U tvorby příspěvku se pak dají povolit

i jednotlivé HTML tagy a je přítomna i ochrana před spamem. Uživatelé se mohou registrovat a mít vlastní účet. Příspěvky diskuze lze i moderovat. Lze také nastavit notifikace, pokud přibude nový komentář.

2.2.1.7 Multisite

WordPress umožňuje spravovat více webů pod jednou instalací. Takže pokud máme více webů, můžeme je spravovat přes jeden redakční systém, a využít tak sdílených prostředků, jako např. obrázky.

2.2.1.8 Licence

WordPress je vydán pod licencí GPLv2. To nám mimo jiné umožňuje:

- Používat program pro jakýkoliv účel,
- Studovat, jak program funguje a měnit ho dle vůle,
- Redistribuovat,
- Distribuovat kopie pozměněné verze ostatním.

2.2.1.9 Požadavky

Doporučené požadavky jsou:

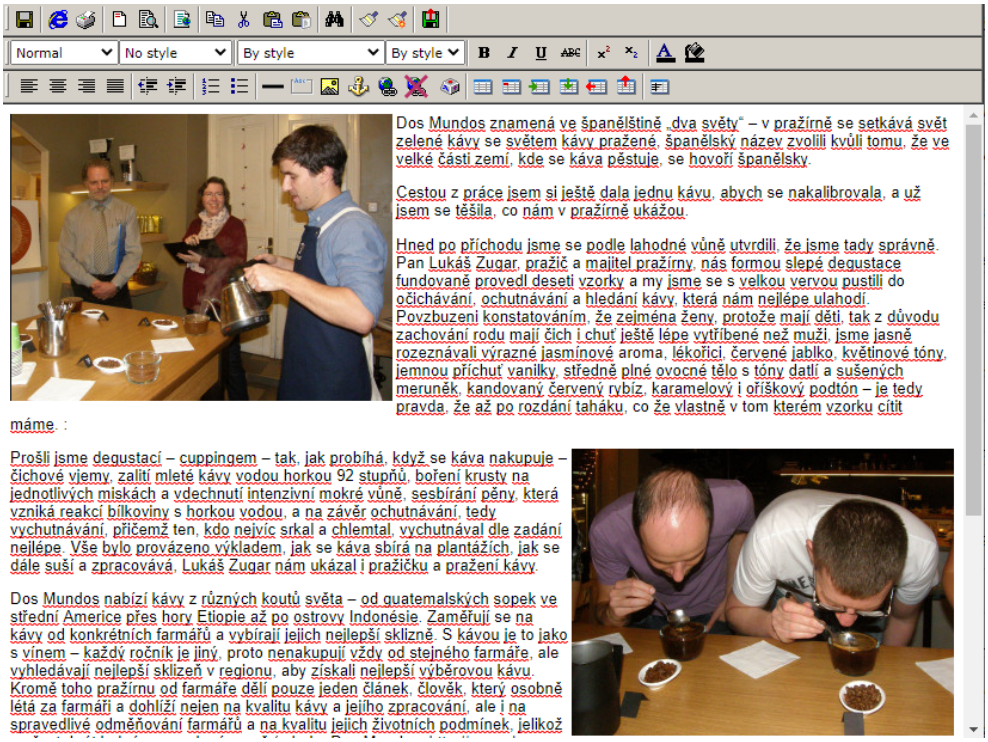
- PHP verze 7.4 a vyšší,
- MySQL verze 5.6 a vyšší, nebo MariaDB verze 10.1 a vyšší,
- Podpora HTTPS protokolu.

WordPress se sice dá zprovoznit i na starším PHP (5.6) a MySQL (5.0) verzi, ale pro tyto verze již byla ukončena oficiální podpora, a tak mohou obsahovat bezpečnostní chyby [7].

2.3 MultiCMS

MultiCMS je komerční CMS systém [8]. Bohužel jej nelze volně vyzkoušet tak, že by se dala stáhnout zkušební verze. Demoverzi si lze domluvit přes e-mail, který nalezneme na oficiálních stránkách `https://www.multicms.net/demo.html`. Na stránkách nalezneme různé edice tohoto produktu, kdy každá edice má rozdílné moduly, které nabízejí jednotlivé funkcionality CMS.

Cena za tento systém je podle dohody, lze ale nalézt orientační ceny pro jednotlivé moduly. K testování jsme měli k dispozici verzi a konkrétní implementaci MultiCMS pro web `https://casopis.mensa.cz/`, budeme tedy hlavně popisovat ji.



Obrázek 2.2: MultiCMS editor příspěvku

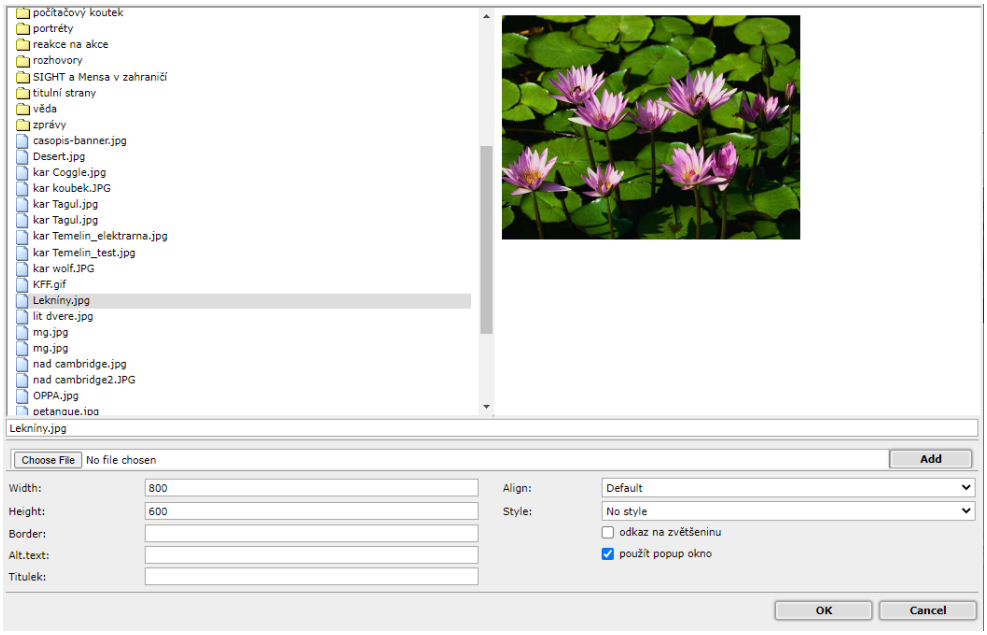
2.3.1 Obsah

Editor obsahu je klasický WYSIWYG editor připomínající starší verze Microsoft Office Word. Nalezneme zde tedy klasické nástroje pro formátování textu, vkládání odkazů, obrázků, tabulek apod. Nabízí také přepínání mezi formátovaným vzhledem a zdrojovým kódem. Nakonec v editoru najdeme i nastavení parametrů článku, kde můžeme nastavit například název článku, perex, klíčová slova, úvodní obrázek, dodatečného autora a další parametry.

Pro vkládání obrázků tu je k dispozici správce médií, kam nahráváme a také nalezneme všechny již nahrané obrázky. Správce médií je dělen na tři zóny, v levé nalezneme stromovou strukturu s názvy obrázků, v pravé pak náhled obrázku a dole jsou nastavitelné parametry jako šířka, výška, zarovnání apod. Obrázky jsou seřazeny abecedně a nelze je seřadit jinak (například podle data vložení), zároveň můžeme vidět pouze jeden náhled aktuálně označeného obrázku.

2.3.2 Administrace

V administraci pak nalezneme vlevo seznam položek, které se týkají obsahu webu. Jsou zde jednotlivé rubriky, pod kterými najdeme články, které do nich



Obrázek 2.3: MultiCMS správce médií

patří. Dále zde nalezneme nastavení menu, speciálních stránek a správce médií.

Nahoře vpravo jsou ikonky, přes které se dostaneme do dalšího nastavení, popřípadě provedeme různé akce, jako například odkázat se na samotnou stránku webu, fulltextové vyhledávání, nastavení uživatele, nebo si prohlédnout statistiku návštěvnosti.

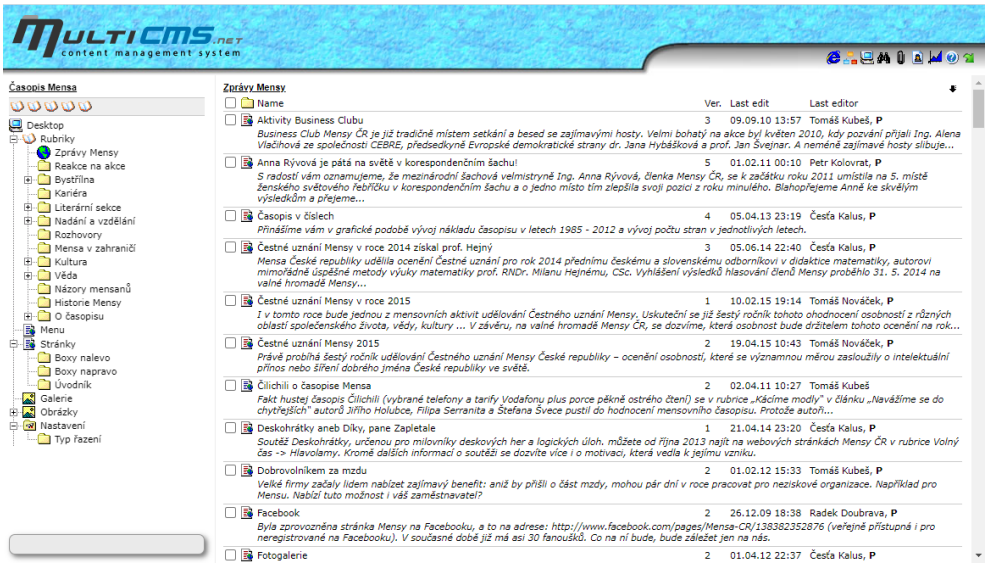
Dole pod seznamem příspěvků nalezneme akce, které souvisí s příspěvky. Je tu například vytvoření nového příspěvku, export do CSV, filtr, tisk. Dále jsou tu hromadné akce pro označené položky jako smazání, přesunutí a exportování. U seznamu příspěvků není nastaveno stránkování, takže při velkém počtu příspěvků se menu nachází poměrně daleko.

2.4 Analýza databázových migrací

V následujících sekcích popíšeme, co to jsou databázové migrace a jak je rozdělujeme.

2.4.1 Databázové migrace

Pod tímto pojmem si představujeme nějaký přesun dat z jedné databáze do druhé. Obvykle se ale při tomto procesu musí provést práce více. Cílová a zdrojová databáze se pravděpodobně budou lišit jak už ve struktuře, tak třeba i technologii. Proto musíme přesouvaná data nějak restrukturalizovat. Pokud se nám mění i datové typy, musíme data sanitizovat (odstranit neplatné znaky).



Obrázek 2.4: Náhled prostředí administrace MultiCMS

Při migraci můžeme data i různě modifikovat, doplňovat nová a mazat přebytečná. Tyto modifikace bývají běžné, pokud se jedná o migraci nejen databáze, ale migraci dat z nějakého informačního systému do jiného. To je případ i této závěrečné práce.

2.4.2 Typy databázových migrací

Mezi hlavní typy databázových migrací patří manuální přesun, databázový dump a verzování databáze. V následujících řádcích jednotlivé typy rozebereme detailněji.

2.4.2.1 Manuální přesun

Pokud máme relativně malou databázi a migrace spočívá spíše v menší restrukturalizaci, můžeme využít různé nástroje, které nám umožní manipulovat s databázemi pomocí GUI. Tyto nástroje mohou umožňovat jednoduchý import a export databází s možností přejmenování sloupců a podobných jednoduchých úprav. Popřípadě se najdou sofistikovanější nástroje, které dokážou přímo nakopírovat data z jedné databáze do druhé s jednoduchým mapováním sloupců. Mapováním sloupců myslíme, že nastavíme přesun dat z jednoho sloupce do jiného sloupce.

2.4.2.2 Databázový dump

Stejné nástroje, jako u manuálního přesunu, nám nabízí export databáze do tzv. dumpu. To je soubor, ve kterém najdeme sérii SQL příkazů, které nám

postupně vytvoří strukturu databáze a naplní daty. Tento výsledný dump samozřejmě bude konkrétní pro danou databázovou strukturu, data i technologii.

Avšak my si takovýto dump můžeme i vygenerovat. Typicky k tomu již využijeme nějaký programovací jazyk a knihovny, které nám dovolí připojit se k databázi a manipulovat s databází pomocí SQL, či ideálně pomocí nějakého jednotného formátu SQL, který není vázaný na konkrétní technologii databáze a který se nám pak na tu konkrétní technologii převede.

2.4.2.3 Verzování databáze

Zatímco předešlé dva typy spíše využijeme při jednorázovém přesunu dat, verzování nám nabízí praktickou metodiku, jak vyvíjet databázi společně s projektem (který je také ve vývoji). K verzování databáze zase najdeme velkou škálu knihoven pro všechny možné jazyky. Databázi verzujeme tak, že si necháme vygenerovat migrační soubor, který se běžně pojmenuje jako aktuální čas, nebo číslo verze, a krátký popis, co daná migrace dělá. Dále ho naplníme SQL příkazy, které modifikují databázi. Typicky píšeme SQL příkazy pro vytvoření chtěného efektu a zároveň opačný efekt, pro případ, kdybychom chtěli verzi stáhnout zase dolů. Takovýto migrační soubor zaverzujeme verzovacím systémem a při nasazení projektu by se migrace měly spustit. Ty postupně spouští příkazy z migračních souborů podle data, aktuální verze a dalších vnitřních informací.

2.4.3 K čemu potřebujeme databázové migrace

Databázové migrace využijeme například při převodu dat z jednoho informačního systému do jiného, což je i cílem této bakalářské práce. Důvod k takovému převodu může být různý, ať už končící podpora starého systému, finanční rozhodnutí, bezpečnostní důvody nebo prostě jen přechod na něco modernějšího. Tyto systémy pak budou mít rozdílnou hierarchii dat, některá data budou chybět, některá přebývat a jiná mohou nabývat úplně jiné formy. Takováto operace s daty nebývá triviální a je proto potřeba mnohdy využít kombinací více nástrojů, nebo vytvořit vlastní řešení.

Častější podoba migrací pak bývá u kopírování dat a struktury databáze jednoho konkrétního systému či aplikace. Například u webové aplikace běžně máme nějaké vývojové a produkční prostředí. Ve vývojovém prostředí programátoři, či jiné osoby zabývající se vývojem dané aplikace, testují změny, než se vydají do produkce, tedy veřejně na internet. Vývojové prostředí má i svoji databázi, která může být naplněna testovacími daty. Pokud se změna pak týká právě i databáze, např. přidání nové tabulky, je potřeba danou změnu provést i na produkční databázi. K tomu nám může pomoci široká škála nástrojů, nicméně je to další operace, kterou musí provést člověk a která by se dala automatizovat.

Takovou automatizací může být právě verzování databáze, kdy změny v databázi zapíšeme do migračních souborů a správné nastavení procesu vydávání nové verze aplikace by se měla postarat o zbytek. Díky tomu jsme nejen automatizovali proces migrace, ale i jsme zajistili lepší správu historie verzí a můžeme tak snadněji dohledat, při jaké příležitosti změna v databázi nastala.

2.4.4 Konkrétní migrační nástroje

Na následujících řádcích si představíme pár příkladů konkrétních nástrojů, které se využívají při migracích.

2.4.4.1 Navicat Premium

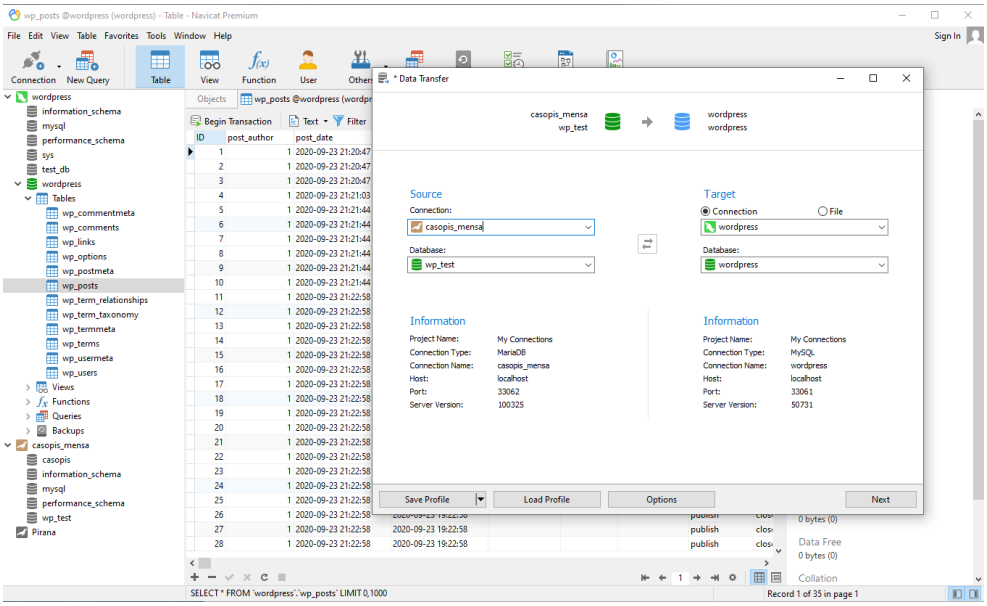
Komerční nástroj na správu databází [9], který nabízí mimo jiné možnosti přesunu jak dat, tak i struktur mezi jednotlivými databázemi.

Migrační nástroje, které nám nabízí jsou:

- Data Transfer,
- Data Synchronization,
- Structure Synchronization,
- Import a Export.

Pomocí Data Transfer můžeme zkopírovat data z jedné databáze do druhé. Výchozí nastavení počítá se stejnou strukturou a bude se snažit zkopírovat všechny záznamy ze všech tabulek, které vybereme, že chceme zkopírovat. U každé tabulky můžeme ještě učinit pokročilejší nastavení. V pokročilém nastavením můžeme určit, jak se bude tabulka jmenovat po zkopírování. Dále můžeme určit, které sloupce se mají zkopírovat a jak se budou jmenovat na druhé databázi. Nakonec ještě můžeme zvolit počet záznamů ke zkopírování, popřípadě zadat sérii filtrů. Těmi můžeme určit offset a počet záznamů, popř. zvolit komplexnější podmínky na základě hodnot ve sloupcích. Po spuštění transferu se bude proces snažit hledat existující tabulky a sloupce (i přejmenované, pokud jsme tak vyplnili). Pokud dané tabulky či sloupce nenalezne, bude se je snažit vytvořit. Tímto nástrojem tedy můžeme docílit základních migrací, pokud jde o jednoduché úpravy jako přejmenování či prohození pár sloupců a podobné.

Data Synchronization slouží, jak název napovídá, k synchronizaci dat. Poté, co vybereme zdrojovou a cílovou databázi, nám nástroj ukáže, pro které tabulky můžeme synchronizovat data. Při výběru tabulek můžeme ještě nadefinovat mapování klíčů a polí, tedy můžeme zase využít synchronizace mezi databázemi s lehce pozměněnou strukturou. Poté co máme vybrány tabulky a nastaveno mapování můžeme nechat porovnat data. To nám ukáže přehledný seznam pro jednotlivé tabulky, kolik proběhne insert, update a delete příkazů.



Obrázek 2.5: Náhled prostředí Navicat Premium

Také nám ukáže počet dat, které se již nachází v cílové databázi. Tyto tabulky můžeme vybrat, jestli se mají synchronizovat, nebo se mají ponechat. Po výběru tabulky můžeme detailně vidět, které záznamy se budou kopírovat, měnit, nebo mazat. V tomto seznamu si můžeme zvolit jednotlivé záznamy, jestli se má daná akce pro daný záznam provést, či nikoliv. Když máme vybráno, můžeme spustit synchronizaci a akce s daty se provedou. Tento nástroj je tedy vhodný pro synchronizací dat stejné databáze v různých prostředích.

Dále máme Structure Synchronization, který se bude snažit srovnat strukturu zdrojové a cílové databáze. Tento nástroj zjistí rozdíly ve struktuře a bude se snažit založit tabulky co chybí nebo smazat tabulky co přebývají. Pokud se liší struktura stejné tabulky, nástroj vytvoří chybějící sloupce a smaže přebývající. S tvořením tabulky se vytvoří jak sloupce, tak klíče a další nastavení. Tento nástroj se může hodit, pokud vyvíjíme nějakou databázi, pro kterou často měníme strukturu a nemáme nastaveno verzování databází, aby se takováto synchronizace provedla sama.

Export nám nabízí převést naší databázi na jiný formát. Import poté tento formát převést zpět na databázi. Na výběr máme celkem z deseti formátu, z nichž mezi nimi je i například Excel. Když budeme chtít exportovat naší databázi do Excelu, máme na výběr, které tabulky chceme převést a do jakých souborů. Popřípadě můžeme zvolit, že chceme všechny vybrané tabulky do jednoho souboru. V další části vybíráme, které konkrétní sloupce chceme převést. V posledním kroku vybíráme, jestli chceme zahrnout názvy sloupců a pokračovat v exportu i při případné chybě. Při importu z Excelu zase mohu

zvolit, které tabulky chci převést do databáze. Musím dále zvolit, ve kterém řádku se nacházejí jména sloupců a od kterého řádku začínají záznamy. Mohu zvolit i v jakém řádku záznamy končí a formát data. Pokud importujeme do nějaké databáze s již existujícími tabulkami, můžeme zvolit mapování importovaných tabulek. Dále můžeme pro každou tabulku zvolit primární klíč, kde ve výchozím nastavení se bere první sloupec. Poslední možnost vybíráme Import Mode. V Append mode záznamy jen přidáváme. Update Mode bude záznamy se stejným ID jen aktualizovat. Append/Update bude nové záznamy přidávat a stávající aktualizovat. Delete bude mazat záznamy se stejným ID. Copy bude také záznamy se stejným ID mazat a poté je znova přidá ze zdroje.

Obecně považujeme Navicat Premium za šikovný a přehledný nástroj pro práci s databázemi. Hodí se pro klasické procházení dat a jejich manipulaci přímo v databázi a pro jednoduché migrace totožných či lehce pozměněných databází.

Podobně pracující nástroje jsou např. HeidiSQL, který je zdarma pro Windows. Dále phpMyAdmin, který je také zdarma a můžeme ho ovládat přes webový prohlížeč, avšak umí pracovat pouze s MySQL a MariaDB.

2.4.4.2 Talend Open Studio for Data Integration

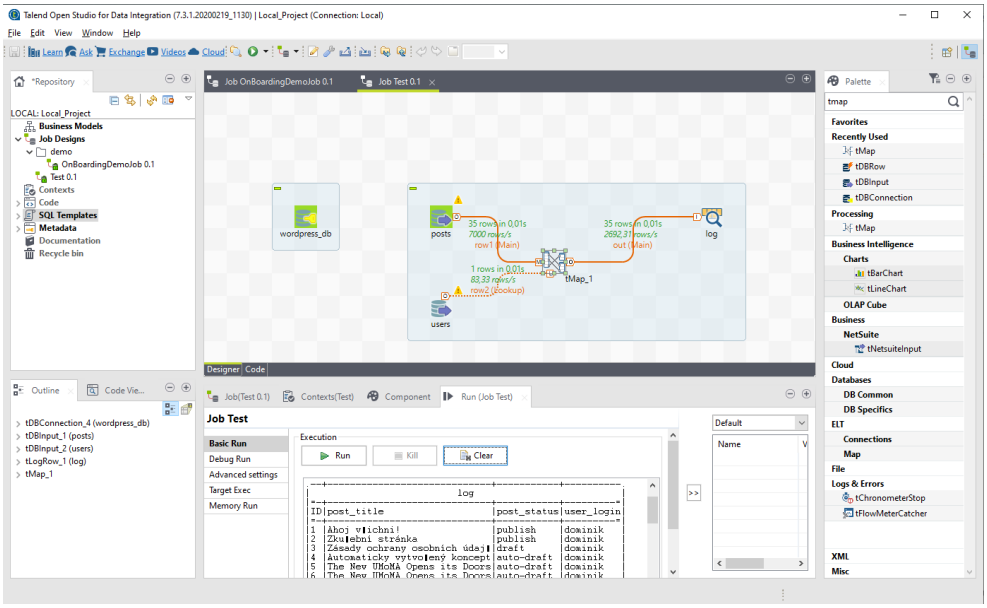
Talend Open Studio je open source nástroj pro ETL (Extract, Transform, Load) a datovou integraci [10]. V nástroji si můžeme založit projekt, který pak může provádět nějaké operace s daty, tento proces se nazývá *Job*. K nastavení *Job* máme k dispozici vizuální editor, který se dělí na 4 hlavní části.

První okno *Repository* nám umožňuje procházet hlavní komponenty *Job*, mezi které patří *Job Design*, které nejvíce souvisí s datovou migrací. Mezi další komponenty ale patří i například *Business Models*, ve kterých se dají dělat různé diagramy, nebo *Documentation*, do které můžeme vložit soubory s dokumentací.

Prostřední okno nám umožňuje editovat dané komponenty, tedy pokud vybereme nějaký náš *Job Design*, zobrazí se nám prostor pro tvorbu diagramu. Do digramu můžeme vkládat komponenty *Job Design*, které nalezneme v pravém okně *Pallete*, ty jednoduše přetáhneme myší do diagramu a tím se nám vloží do našeho *Job Design*. Komponent je veliké množství, a umožňujme nám tak nastavit vskutku komplexní manipulaci dat.

Nalezneme tu různé datové zdroje, nejen databázové, ale můžeme načíst data i ze souboru, z URL, z cloudových služeb a podobné. Dále můžeme data přetransformovat, máme k dispozici mapování, filtrování, agregaci, řazení, konvertování, a dokonce můžeme i vložit vlastní kód v Javě nebo Groovy. Nakonec budeme chtít i data někam uložit, nebo poslat, a tak můžeme využít nejen komponenty datových zdrojů, ale i různé logování, nebo dokonce poslání e-mailu.

V okně uprostřed dole potom můžeme jednotlivé komponenty nastavovat, kdy každá komponenta má své vlastní možnosti, co jí můžeme nastavit. Když



Obrázek 2.6: Náhled prostředí Talend Open Studio for Data Integration

bychom chtěli například v logu vypsat seznam názvů článků s jeho autorem, kdy autory máme v jedné tabulce a články ve druhé, spojené přes cizí klíč, mohli bychom toho docílit pěti komponentami. První komponenta by byl datový zdroj mé databáze, kde se články a autoři nacházejí. Pro tu bychom nastavili údaje k připojení a tento datový zdroj teď můžeme využívat pro další komponenty. Dále založíme dvě *DBInput* komponenty, jednu pro články, druhou pro autory. V *DBInput* komponentě můžeme pomocí SQL dotazů vybrat data z databáze, potřebujeme tedy do těchto komponent určit datový zdroj typu databáze, který jsme založili předtím. Dále pro články vyplníme, aby se nám vybraly všechny názvy článků a ID autorů. Pro autory vyplníme výběr ID a jmen autorů. Založíme novou komponentu *Map*, která nám umožní namapovat sloupce z různých tabulek do vlastního výstupu. V diagramovém okně můžeme vytvořit vazbu mezi komponentami, potáhneme tedy výstup z komponent článků a autorů do vstupu *Map*. V mapě teď můžeme propojit ID autorů se jmény autorů, a vytvořit tak nový výstup názvů článků a jmen autorů. Založíme poslední komponentu *Log*, do které potáhneme v diagramu výstup z *Map*. V *Log* nastavíme, aby se výstup do konzole vykresloval jako tabulka, a můžeme spustit náš *Job*.

Pokud jsme vše vyplnili správně, *Job* se připojí do naší databáze, vybere data, spojí je do jedné tabulky a vypíše do konzole. Pokud ne, v konzoli se objeví chyby, které máme opravit. Chyby se objevují i při procesu tvorby, ať už v dolním okně s nastavením komponent, tak i rovnou v diagramu, takže můžeme předejít chybám v konzoli úplně.

Celý *Job* by se dal určitě udělat i jednodušeji, nicméně pro názornost jsme zvolili tento přístup, abychom více znázornili práci s komponentami.

Talend Open Studio je tedy velmi komplexní nástroj, který nám nabízí mnohem více než jen datové migrace a dá se využít všude, kde je potřeba častá manipulace s daty či při potřebě data nějak prezentovat.

2.5 Analýza problému

Jelikož systémy MultiCMS a WordPress jsou do značné míry odlišné ve struktuře dat, bude jejich přesun netriviální. Budeme muset zajistit hned několik věcí, které podrobněji popíšeme dále. V následujících řádcích budeme zmiňovat názvy sloupců daných databázových tabulek, jejichž diagramy můžeme nalézt v příloze B.

2.5.1 Uživatelé

Uživatele budeme muset přemigrovat jako první, protože se na ně váže autorství článků, vlastnictví souborů a další záznamy. Jediní bohužel nepůjdou přemigrovat bezztrátově, jelikož nemáme k dispozici jejich kompletní přihlašovací údaje. Jejich migrace tedy proběhne bez hesel a uživatelé si znovu vytvoří nové heslo přes funkcionalitu zapomenutého hesla, která je v systému WordPress dostupná.

Data uživatelů máme k dispozici ve formátu CSV, což jsou údaje oddělené čárkou a záznamy oddělené řádky. Data obsahují následující údaje:

- ID,
- Uživatelské jméno,
- Celé jméno,
- E-mailovou adresu,
- Roli.

Role není vyplněna u všech záznamů, těm tedy nastavíme jako výchozí roli *Author*.

Kdybychom chtěli přemigrovat i hesla, a zajistit tak uživatelům možnost se přihlásit stejnými údaji na novém systému, museli bychom znát hašovací funkci a sůl, která byla použita ve starém systému. Dále bychom museli znát zahašovaná hesla jednotlivých uživatelů a museli bychom zajistit, aby se na novém systému používala hašovací funkce z původního systému.

Avšak ještě než přemigrujeme uživatele, budeme muset do systému přemigrovat role, které uživatelé mají. WordPress ve výchozí instalaci bohužel nenabízí správu rolí, jak jsme již zmiňovali v podsekci 2.2.1.2, použijeme tedy

plugin *Members – Membership & User Role Editor Plugin*, který můžeme stáhnout zde `https://cs.wordpress.org/plugins/members/`, nebo přímo v administraci pluginů, kde ho také nainstalujeme.

O rolích z původního systému nemáme k dispozici žádné další informace, nevíme tedy, jaké měly práva v systému. Rozhodneme se tedy pro nové role k přemigrování nastavit stejná práva, jako má výchozí role *Author*. Dodatečné nastavení práv pro nové role si pak rozhodne administrátor.

2.5.2 Obrázky a soubory

Obrázky se ve WordPress ukládají jako soubory na disku, v databázi se pak uvede název souboru a dodatečné informace. Původní obrázky z MultiCMS budeme muset stáhnout a znovu nahrát do systému WordPress, s tím, že si budeme muset držet informace, o které obrázky jde, abychom je mohli napárovat zpět do článků a převést jejich odkazy do formátu, který používá WordPress.

Seznam obrázků, které máme migrovat, máme k dispozici v SQL dumpu. Záznamy obrázků jsou spolu se záznamy článků v jedné tabulce, kdy rozlišovacím znakem, jestli se jedná o obrázek, je hodnota „1“ ve sloupci *type*. U jednotlivých obrázků pak máme k dispozici údaje jako jsou ID, název souboru, ID vlastníka, datum vytvoření a poslední úpravy. Použití obrázku ve článku je pak uloženo jako HTML tag *img* v obsahu článku, kde v atributu *src* najdeme jeho ID. Pomocí URL `https://casopis.mensa.cz/image.php?idx=ID`, kde místo „ID“, doplníme ID konkrétního obrázku, můžeme na daný obrázek nahlédnout a stáhnout jej.

Ostatní soubory jsou na tom podobně, jenom se ve článcích na ně odkazuje pomocí HTML odkazu na URL ve tvaru `https://casopis.mensa.cz/download.php?idx=ID`, kde zase místo hodnoty „ID“ uvedeme ID konkrétního souboru.

2.5.3 Rubriky

Seznam rubrik se nachází v SQL dumpu v samostatné tabulce *cat*, kde máme k dispozici údaje jako ID, název a popis rubriky, datum založení a poslední úpravy a ID nadřazené kategorie. K dispozici máme i přehledný seznam ve formátu XML na URL `https://casopis.mensa.cz/export/kategorie.php`, kde můžeme vidět na první pohled zanoření jednotlivých rubrik.

Ve WordPress aplikaci tyto informace musíme zanést do dvou tabulek. V jedné se nachází seznam rubrik s jejich názvy a popisy, ve druhé tabulce zase najdeme vazby nadřazenosti.

2.5.4 Články

V obou systémech se články píší ve WYSIWYG editoru, články jsou tedy ve formátu HTML. Ve WordPress je tento obsah článků ale ještě obohacen

o komentáře, které určují, o jaký formát bloku se jedná. WordPress využívá WYSIWYG editor jménem Gutenberg, který je založen na vkládání různých druhů bloků jako jsou odstavce, obrázky, tabulky a podobné. Tyto bloky rozpoznává právě podle HTML komentářů, které přidává do obsahu. Pokud žádný komentář před elementem uveden není, použije se obecný HTML blok. Při migraci článků tedy bude nutné tyto komentáře nejprve přidat, abychom využili potenciál Gutenberg editoru.

Články ze starého systému máme k dispozici v SQL dumpu v jedné tabulce, společně s obrázky a soubory. Jeho rozlišovací znak od souborů je hodnota „8“ ve sloupci *type*. Záznamy jednotlivých článků obsahují ID, název článku, obsah článku, perex, ID rubriky, klíčová slova oddělená čárkou, ID a jméno autora, datum založení a poslední úpravy.

Jméno autora a ID se v záznamech často liší, především pak pro autora s ID rovno „4“, jehož uživatelské jméno je „prispevatel“ a jeho e-mail je „redakce@mensa.cz“. Vypadá to tedy, že tento účet používalo více autorů a konkrétní jméno autora se doplnilo jinak, než aby bylo propojeno s uživatelským účtem. Do tohoto pole se nevyplňují pouze jména, ale můžeme tam najít i překladatele s uvedeným názvem originálního díla nebo další poznámky. Na samotném webu časopisu se data z tohoto pole propisují vždy na konec článku s označením „Autor“. Tento text budeme tedy muset také vygenerovat pro každý článek před přesunem do nové databáze.

Ve finále, kdy budeme migrovat nově převedené články do databáze, uvedeme i vazbu na konkrétní rubriku, které jsme přemigrovali již předtím.

2.5.5 Další požadavky klienta

Od klienta jsme dostali dodatečné požadavky, které se týkají funkcionality výsledného redakčního systému.

Prvním požadavkem je možnost rozkliknutí obrázku v článku do nastylované galerie. Většina pluginů implementující galerii na platformě WordPress využívají vlastnosti obrázků, nastavenou tak, že jsou obaleny odkazem na jejich zdroj. Toto budeme tedy muset také zohlednit v implementaci a budeme tedy muset obalit výskyty obrázků v článcích odkazem vedoucím na zdroj. Pro zobrazení nastylované galerie poté využijeme plugin WP Featherlight [11].

Dalším požadavkem je možnost vidět použití médií ve článcích. Tedy ve správci médií u obrázku vidět názvy článků, ve kterých je daný obrázek použit, ideálně s možností se na daný článek prokliknout. Toto jednoduše vyřešíme pluginem Attachment Usage [12].

2.6 Návrh řešení

V následujících podsekcích si popíšeme postup, kterým se budeme řídit při implementaci a jaké nástroje k tomu využijeme.

2.6.1 Vstupní data

K tomu, abychom mohli migraci provést, potřebujeme vstupní data. Ty jsme dostali ve formě databázového dumpu. V něm jsou k dispozici články, rubriky, data o použitých obrázcích a další informace. Dále máme k dispozici odkazy na XML zdroje seznamu rubrik a seznamu článků, které vrací data z databáze, kterou již máme k dispozici z dumpu, zformátované do čitelnější podoby, což se nám bude hodit při mapování sloupců. Ještě máme k dispozici vzor odkazu, přes který můžeme stáhnout obrázky, a nakonec CSV soubor s uživateli.

2.6.2 Zvolený postup

Jak jsme již zmiňovali v analýze problému, celá migrace systému bude netriviální, je proto potřeba zvolit vhodný postup. Kdybychom se vydali cestou manuálního přesunu, znamenalo by to pro nás, že budeme muset zvolit nějaký sofistikovanější nástroj, který zvládne restrukturalizaci dat a brát si zdrojová data i odjinud než z databáze, a ty zase ukládat. Tvorba takového procesu by mohla být dost složitá a mohli bychom narazit na problémy, které nám program nedokáže vyřešit. Dále by musela vzniknout dokumentace, jak daný proces obsluhovat v daném programu, aby mohl migraci obsloužit někdo jiný. Taková dokumentace by mohla být pro uživatele příliš složitá. Nakonec při případném rozšíření či modifikaci migrace se mohou věci dále komplikovat kvůli již zmíněným problémům.

Verzování databáze nám s migrací nepomůže. Nicméně je to dobrý přístup, který můžeme zvolit až po zhotovení migrace, a následném vývoji webu již na nové platformě.

Rozhodli jsme se tedy pro tvorbu vlastního řešení, které bude založeno na generování databázových příkazů a jejich následné exekuci. Tento migrační proces bude moci být spuštěn z konzole, popřípadě v přidružené webové aplikaci.

2.6.3 Použité nástroje

V této podsekci popíšeme, které nástroje jsme se rozhodli použít při tvorbě migrace.

2.6.3.1 Docker

Nástroj, který slouží pro virtualizaci prostředí v izolovaných kontejnerech [13]. Využijeme ho pro provoz lokálního prostředí, které bude složeno ze tří kontejnerů. V jednom budeme mít databázi, ve které budeme mít naimportován vstupní dump z MultiCMS. Ve druhém budeme mít databázi pro nový převedený web do platformy WordPress a ve třetím budeme mít webový server s PHP. V posledním zmíněném kontejneru budeme mít nainstalován

WordPress a také budeme vyvíjet migrační proces a jeho přidruženou webovou aplikaci.

2.6.3.2 Git

Kód migrace jsme se rozhodli verzovat v populárním, léty ověřeném nástroji Git, což je open source systém pro distribuovanou správu verzí, který je k dispozici zdarma [14].

2.6.3.3 Navicat Premium

SQL klient, který budeme používat pro náhled stavu databáze při vývoji a počátečním importu vstupního dumpu do zdrojové databáze.

2.6.3.4 Symfony

PHP framework, který nám usnadní vývoj migrace a přidružené webové aplikace [15]. Nabízí nám hned několik hotových řešení, a to zejména pro práci s konzolí [16], práci s databází [17] a tvorbou šablon [18].

Framework používá návrhový vzor MVC, který se skládá ze tří druhů objektů. *Model* je aplikační objekt, *View (Pohled)* je jeho vyjádření na obrazovce a *Controller (Řadič)* definuje způsob, jak uživatelské rozhraní reaguje na uživatelský vstup [19].

2.6.3.5 PhpStorm

Profesionální IDE nástroj pro vývoj aplikací v PHP [20], který nám usnadní vývoj naší migrace.

Kapitola 3

Implementace

3.1 Příprava lokálního prostředí

Lokální prostředí nám umožní rozběhnout webové služby na našem počítači, a tak můžeme vyvíjet a testovat migrační příkazy lokálně, místo toho abychom využívali produkční prostředí, a ovlivnili tak aktuální web. V následujících podsekcích popíšeme tvorbu tohoto lokálního prostředí.

3.1.1 Tvorba kontejnerů

V první části vytvoříme tři Docker kontejnery, se kterými budeme dále pracovat:

- Databáze pro vstupní data z MultiCMS,
- Databáze pro WordPress aplikaci,
- Webový server pro WordPress a Symfony.

Naše počáteční struktura projektu bude vypadat takto:

```
database ..... data databáze pro WordPress
importdb ..... data databáze se vstupními daty
php-httpd
    Dockerfile ..... příkazy ke tvorbě obrazu webového serveru
src
    migration ..... Symfony aplikace migrace
    wordpress ..... WordPress aplikace
docker-compose.yml ..... definice kontejnerů
```

V souboru *docker-compose.yml* mimo jiné nadefinujeme, ze kterých obrazů budeme vycházet. Pro obě databáze použijeme oficiální obraz *mariadb* [21],

pro databázi k WordPress aplikaci použijeme verzi *latest* (což je poslední stabilní verze, která v době psaní textu je 10.5), pro databázi se vstupními daty použijeme verzi *10.3*, která je uvedena v SQL dumpu. U databází také uvedeme přístupové údaje a složky, do kterých se mají ukládat perzistentní data.

Pro webový server namapujeme složku *src/* na složku */var/www/html/*, která je uvedena jako výchozí složka pro *DocumentRoot* Apache serveru. Dále uvedeme, že chceme vycházet z našeho *Dockerfile* souboru, kde budeme vycházet z oficiálního obrazu *php* [22], ve verzi *7.4-apache*, což je Apache server s PHP ve verzi 7.4.

V *Dockerfile* uvedeme instalaci následujících nástrojů:

- PHP rozšíření *mysqli*, které je nutné pro běh WordPress aplikace,
- Wget, pro stažení Symfony,
- Git, jako potřebná závislost pro Symfony,
- Symfony.

Jako poslední nastavíme uživatelské jméno a e-mail pro Git, které jsou nutné pro používání Symfony.

Tím jsou naše kontejnery připraveny a pomocí příkazu *docker-compose up*, který lze spustit v kořenovém adresáři našeho projektu, je můžeme rozběhnout.

3.1.2 Instalace WordPress

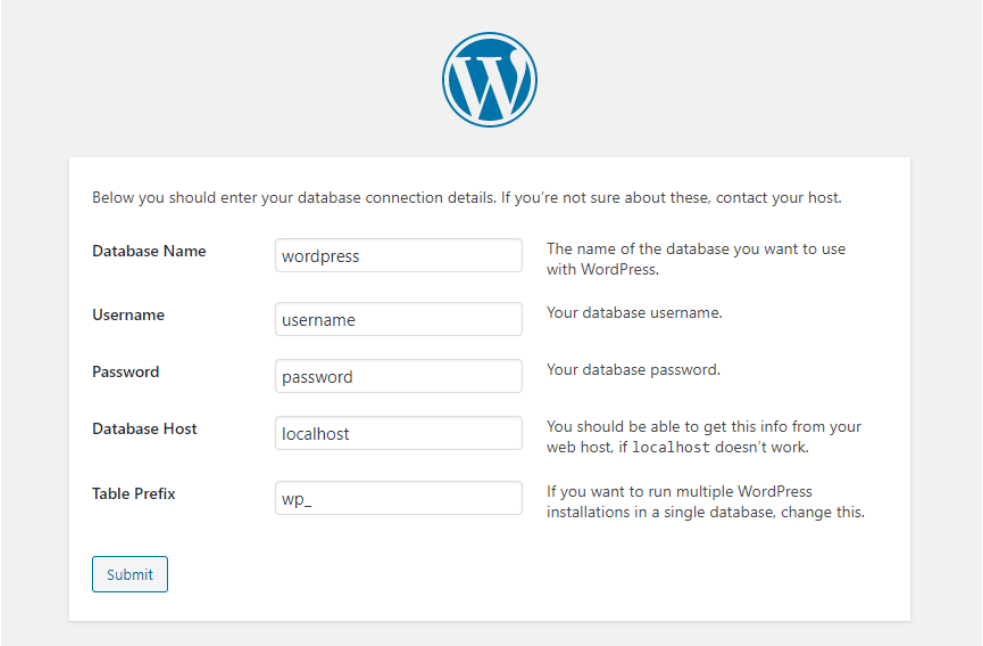
WordPress si stáhneme na oficiálních stránkách `https://wordpress.org/download/` v podobě Zip archivu, jehož obsah následně extrahujeme do složky *src/wordpress/*. Poté spustíme naše Docker kontejnery a na naší lokální adrese *localhost/wordpress* budeme mít instalaci WordPress aplikace.

V instalaci musíme vyplnit přístupové údaje k databázi, to znamená jméno databáze, uživatelské jméno a heslo, název hostitele a prefix pro tabulky. Všechny přístupové údaje vyplníme tak, jak jsme je nastavili v souboru *docker-compose.yml* a tabulkový prefix ponecháme na výchozí hodnotě „wp_“.

V dalším kroku instalace vyplníme název stránky, uživatelské jméno a heslo, pod kterým se budeme přihlašovat do administrace, dále náš e-mail a volbu, jestli chceme, aby naši stránku neindexovaly vyhledávače. Po vyplnění klikneme na tlačítko „Install WordPress“ a po chvíli nás aplikace přesměruje na přihlášení do administrace, kde už máme připravenou naší WordPress aplikaci.

3.1.3 Instalace Symfony aplikace

Nainstalování základní kostry Symfony aplikace můžeme provést přímo v našem kontejneru s PHP. Spustíme tedy naše kontejnery a poté spustíme *bash* prostředí PHP kontejneru pomocí příkazu *docker exec -it nazev_kontejneru*



Obrázek 3.1: Instalace WordPress

bash. V tomto prostředí spustíme příkaz *symfony new migration --full*, to zajistí stažení projektu *symfony/website-skeleton* a jeho inicializaci.

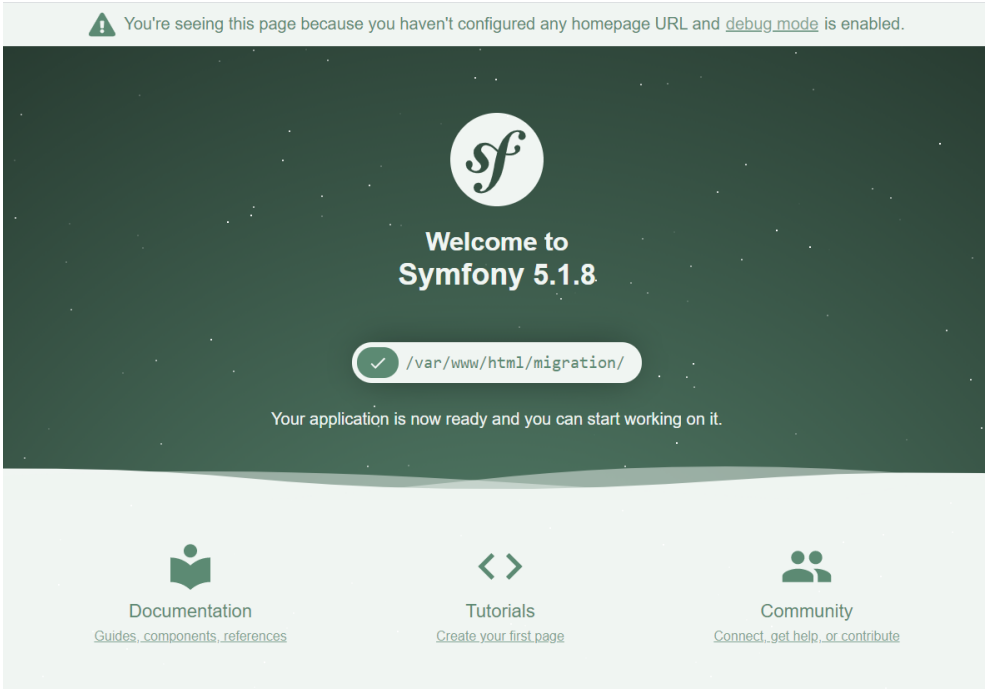
Příznak *--full* jsme použili právě proto, že zajišťuje stažení tohoto balíčku, který je vhodný pro tvorbu klasického webu. To se nám bude hodit, až budeme tvořit GUI pro naší migrační aplikaci. Pokud bychom příznak *--full* vynechali, stáhl by se balíček *symfony/skeleton*, který je úspornější a chybí v něm například knihovny pro šablonovací jazyk Twig a formuláře, které se nám právě budou hodit. Tento úspornější balíček je spíše vhodný pro mikroslužby, konzolové aplikace nebo tvorbu API.

Když instalace doběhne v pořádku, měli bychom vidět na naší lokální adrese *localhost/migration/public* uvítací stránku, viz obrázek 3.2. Tím je naše Symfony aplikace připravena pro vývoj migrační aplikace.

3.1.4 Import vstupních dat

Abychom mohli se vstupními daty pracovat pohodlněji, je vhodné tyto data převést do takové podoby, se kterou můžeme snadněji manipulovat. Vstupní data jsme dostali ve formátu SQL dumpu, tím se nám nabízí možnost tyto data naimportovat do konkrétní databáze.

Začneme tím, že spustíme naše Docker kontejnery a v Navicat se připojíme k databázi, kterou jsme si určili pro vstupní data. Dále v Navicat zvolíme *Execute SQL File*, odškrtneme výchozí nastavení *Continue on error* a spustíme.



Obrázek 3.2: Uvítací obrazovka Symfony

Data bohužel napoprvé neprošla. Z chybových hlášek můžeme vyčíst, že se snažil provést SQL příkaz *ALTER TABLE* na tabulku *discussion*, která ale nikde předtím nebyla vytvořena. Smažeme tedy vše, co se podařilo naimportovat, a podíváme se na zmíněný SQL příkaz přímo do dumpu. V něm můžeme vidět, že příkaz má přidat primární klíč na sloupec *idx* a indexové klíče na sloupce *subject* a *owner*. Dále můžeme dohledat, že se s tabulkou *discussion* nikde dál nepracuje, tedy se do ní nepřidávají ani záznamy. Můžeme tedy tento původní SQL příkaz s klidem smazat, jelikož nám nepřináší žádnou hodnotu. Po smazání příkazu spustíme znovu SQL dump v Navicat jako předtím a tentokrát vše proběhne v pořádku.

Ze vstupních dat nám ještě zbyl seznam uživatelů, který je ve formátu CSV souboru. Abychom nemuseli v migrační aplikaci potom zbytečně pracovat ještě s CSV formátem, převedeme si i tyto uživatele do naší databáze se vstupními daty. Při náhledu do souboru zjistíme, že chybí první řádek s názvy sloupců, ty tedy dopíšeme. Jsou to sloupce *id*, *username*, *fullname*, *email* a *role*.

V Navicat zvolíme možnost *Import Wizard* a vybereme, že chceme importovat CSV soubor, dále zvolíme konkrétní soubor a necháme vyplněné výchozí hodnoty až po volbu datových typů, kde u sloupce *id* zvolíme datový typ *int* a nastavíme ho jako primární klíč. U zbylých sloupců necháme výchozí datový typ *varchar* s délkou 255 znaků. Nakonec zvolíme importovací mód *Append* a spustíme převod. Tím máme vytvořenou tabulku *users* se záznamy uživatelů.

Výsledkem celého importu je naplněná databáze v podobné formě, ve kterém byla na původním systému. Díky tomu můžeme s daty pracovat pomocí SQL dotazů a příkazů, respektive databázovou abstraktní vrstvou, kterou si představíme později.

3.2 Tvorba migrační aplikace

V následující sekci budeme popisovat tvorbu Symfony aplikace, tedy jednotlivých migračních skriptů a následného GUI prostředí.

3.2.1 Nastavení připojení databáze

Aby naše migrační skripty mohly fungovat, potřebují mít nastavené připojení k naším databázím. Symfony nám nabízí k práci s databázemi projekt *Doctrine*, což je soubor několika PHP knihoven, zejména pak *Database Abstraction Layer*, který nám umožňuje pracovat s databází objektovým přístupem.

Jako první si vytvoříme soubor *.env.local*, kde uvedeme environment konstanty URL našich databází. První konstantu *DATABASE_URL* využijeme pro připojení k WordPress databázi, konstantu *DATABASE_IMPORT_URL* pak využijeme pro připojení k databázi se vstupními daty.

Poté v konfiguračním souboru *config/packages/doctrine.yaml* nastavíme dvě připojení *default* a *import*, kterým nastavíme *url* z našich environment konstant.

Tyto dvě připojení pak můžeme používat v našich jednotlivých migračních příkazech pomocí vkládání závislostí (dependency injection) [23].

3.2.2 Příkaz pro migraci uživatelů a rolí

Tento příkaz budeme realizovat jako třídu *MigrateUsers* dědící ze třídy *Command*, která je připravena jako již použitelná komponenta ze Symfony. Jako argumenty v konstruktoru uvedeme naše dvě připojení k databázím, aby se načetly pomocí dependency injection, a my tak s nimi mohli dále pracovat.

Na začátku si uvedeme pár důležitých konstant jako *DEFAULT_ROLE*, kde uvedeme roli, kterou importovaný uživatel dostane, pokud nebude mít žádnou uvedenou konkrétní roli. Dále nastavíme konstantu *CLONE_ROLE* na roli, kterou budeme klonovat pro nově importované role. Jako poslední nadefinujeme *ROLE_MAP* jako asociativní pole, kde klíčem bude role z databáze se vstupními daty, a hodnotou role, která již existuje v systému WordPress. Pomocí této konstanty rozhodneme, kterou roli nebudeme importovat, ale místo toho uživateli nastavíme jinou, již připravenou, roli, která je nejvíce podobná (nebo ekvivalentní) v systému WordPress.

Příkaz po spuštění jako první přemigruje role. Toho docílí tak, že vybere všechny různé role z tabulky uživatelů k importu. Dále získá všechny role

ze systému WordPress, které se nacházejí v tabulce *wp_options*, v záznamu s *option_name* roven „wp_user_roles“ jako serializované asociativní pole, kde jsou uvedeny role s jejich právy. Na základě těchto dat a předtím nadefinovaných konstant příkaz vytvoří nové role a připojí je do pole s rolemi (a právy). Toto výsledné pole poté serializuje a záznam s rolemi ve WordPress aktualizuje.

Když jsou role přemigrovány, příkaz dále bude migrovat uživatele. Ty jednoduše vybere z databáze se vstupními daty a pro každého uživatele vytvoří záznam v tabulce *wp_users*. Pokud již uživatel se stejným ID existuje, tak místo vložení ho aktualizuje. V systému WordPress ne všechna data týkající se uživatele najdeme přehledně v jedné tabulce (*wp_users*), ale některé další údaje nalezneme, a tím pádem musíme i doplnit, v tabulce *wp_usermeta*. Data v této tabulce fungují obdobně jako v tabulce *wp_options*, tedy záznamy jsou ve formě klíč a hodnota. Pro každého migrovaného uživatele tedy ještě vytvoříme (nebo aktualizujeme) pár meta záznamů jako jsou jméno, příjmení, a především jakou mají mít roli, kterou vyplňujeme jako hodnotu záznamu s klíčem „wp_capabilities“.

Pro migraci jak rolí, tak uživatelů jsme použili *ProgressBar*, což je další připravená komponenta v Symfony, která nám ukazuje v konzoli průběh migrace. Používá se relativně jednoduše, při její tvorbě uvedeme nějaký počet, kterého má dosáhnout. Nabízí se nám použít právě počet záznamů rolí/uživatelů. při každé dokončené migraci jednoho záznamu zavoláme metodu pro inkrementaci a na konci se jen ujistíme, že *ProgressBar* dosáhl konečného počtu. Díky této komponentě přehledně uvidíme, jaký je aktuální stav procesu migrování přímo v konzoli.

Příkazu jsme nastavili výchozí jméno „migrate:users“, které využijeme při jeho spuštění. Migraci tedy můžeme spustit v kořenu aplikace příkazem *./bin/console migrate:users*.

3.2.3 Příkaz pro migraci souborů

Jako předchozí příkaz, i tento bude dědit ze třídy *Command*, pojmenujeme jej *MigrateFiles*.

Seznam souborů k migraci budeme získávat z tabulky *items*, které následně stáhneme pomocí URL `https://casopis.mensa.cz/download.php`, s parametrem *idx* a hodnotou ID daného obrázku. Poté soubor uložíme do složky určené pro WordPress soubory a záznamy o souboru uložíme do tabulky *wp_posts*. Dále ještě k souborům uložíme pár meta záznamů, jeden z nich je název souboru s relativní cestou ze složky pro soubory systému WordPress a druhým záznamem je ID ze starého systému, toto ID používáme pro kontrolu, jestli se má soubor přidat jako nový, nebo jen aktualizovat. Relativní cestu tvoříme ve tvaru *rok/měsíc/název_souboru*, kde datum použijeme, kdy byl soubor nahrán do předešlého systému. Pokud je zároveň soubor obrázek (který poznáme přes MIME typ, což je typ internetového média), tak přidáme další meta zá-

znam, který v serializovaném asociativním poli uvádí znova název souboru, rozměry obrázku a nějaké další výchozí hodnoty.

Pokud se nám nepodaří soubor stáhnout, nebo má prázdný název souboru, tak tento záznam přeskočíme. Pokud již soubor v systému WordPress existuje a nemá v meta záznamech uvedeno ID ze starého systému, pak ke jménu souboru přidáváme umělý postfix, tak aby bylo dané jméno souboru k dispozici. Pokud soubor existuje a má přiděleno ID ze starého systému, pak tento záznam aktualizujeme. Samozřejmě, pokud byl soubor přejmenován, aktualizujeme ten daný soubor. Teprve pokud soubor neexistuje, vkládáme ho jako nový záznam.

Příkaz také využívá *ProgressBar*, takže můžeme vidět aktuální stav migrace. Výslednou migraci souborů můžeme spustit v kořenu aplikace příkazem *./bin/console migrate:files*.

3.2.4 Příkaz pro migraci rubrik

K migraci rubrik jsme od klienta získali dodatečné informace o přepracování současných rubrik na starém systému. To znamená, že články z různých rubrik se na novém systému mají vyskytovat v jiných rubrikách. Z pohledu migrací samotných rubrik to pro nás znamená, že kromě přesunu starých rubrik vzniknou i nějaké nové a nějaké můžeme při migraci ignorovat. Pokud máme uvedeno, že nějakou rubriku migrovat nemáme, nebudeme migrovat ani její podřazené rubriky.

V původním systému máme uvedeny pro rubriky ještě klíčová slova, která slouží k propsání na webu do meta tagu pro klíčová slova. Tato klíčová slova jsme se rozhodli nemigrovat hned z několika důvodů:

- Systém WordPress bez pluginů nepodporuje přidání klíčových slov k rubrice, tím pádem ani jejich vygenerování na webu do meta tagu,
- Google vyhledávač nebere na tento tag ohled [24],
- Seznam vyhledávač nebere na tento tag ohled [25].

V příkazu *MigrateCategories* si tedy uložíme informace o rubrikách ve formě konstantních polí *NEW_CATEGORIES*, kde budeme mít vypsané názvy rubrik, které máme nově založit, a *IGNORE_CATEGORIES*, kde budeme mít názvy rubrik, které máme při migraci přeskočit.

Příkaz při migraci rubriky rekurzivně zkontroluje, jestli nepatří do ignorovaných rubrik či jestli tam nepatří nějaká její nadřazená rubrika. Poté zkontroluje, jestli již rubrika v novém systému přemigrovaná je, a na základě toho bude buď rubriku zakládat, nebo aktualizovat. Rubriku příkaz vytvoří v tabulce *wp_terms*, kde kromě ID doplní ještě jméno rubriky a slug, což je formát řetězce, který se uvede v URL. Pro vygenerování slug řetězce použijeme jméno rubriky bez diakritiky s malými znaky a mezery převedeny na spojovníky.

Pokud má daná rubrika nadřazenou rubriku, uvedeme tuto informaci ještě v tabulce *wp_options* v hodnotě s klíčem *category_children*, kde máme uvedenou hierarchii rubrik jako serializované asociativní pole. Naposled vytvoříme, nebo aktualizujeme rubriku v tabulce *wp_term_taxonomy*, kde uvedeme, že se jedná o typ taxonomie „category“, popis rubriky a případné ID nadřazené rubriky.

Příkaz disponuje komponentou *ProgressBar* a můžeme ho spustit příkazem *./bin/console migrate:categories*.

3.2.5 Příkaz pro migraci štítků

Články z původního systémů obsahují klíčová slova, která se propisují na webu do meta tagu stejně jako v případě rubrik. Touhle funkcionalitou bohužel WordPress nedisponuje, nicméně rozhodli jsme se i tak využít těchto dat a předělat je na WordPress štítky. To znamená, že z klíčových slov se stanou štítky, a propíšou se tak ke každému článku. Zároveň mají tyto štítky každý vlastní stránku, kde je seznam článků s daným štítkem.

Klíčová slova jsou uvedena jako řetězec, kde jsou jednotlivé výrazy oddělen čárkou. Vybereme všechny tyto záznamy, rozdělíme podle čárky, seřadíme a odstraníme duplicity. Pro každý štítek budeme ještě potřebovat vygenerovat slug, zkontrolujeme tedy ještě jestli různá klíčová slova nevygenerují náhodou stejný slug. Naštěstí žádná taková klíčová slova nejsou, pokud by byly, museli bychom ke slug přidat například nějaký číselný postfix, popřípadě pokud by šlo o slova s rozdílným malým/velkým písmenem, mohli bychom tato klíčová slova sladit na jeden tvar.

Podobně jako v případě rubrik postupně přemigrujeme jednotlivá klíčová slova do tabulek *wp_terms* a *wp_term_taxonomy*, kde jako *taxonomy* vyplníme „post_tag“. Příkaz *MigrateTags* také obsahuje *ProgressBar* a můžeme ho spustit příkazem *./bin/console migrate:tags*.

3.2.6 Příkaz pro migraci článků

Seznam článků získáme z tabulky *items*, kde obsah článku se nachází ve dvou sloupcích, *data2* a *data3*. Obsah těchto sloupců je zdánlivě totožný, rozdíl jsme nalezli jen u pár článků, kdy *data3* vypadají jako formátované *data2*. Formátování přidává prázdné řádky, mění pořadí atributů *width* a *heigth* u obrázků a konvertuje tagy ** na tagy *<i>*. Rozhodli jsme se tedy vzít obsah ze sloupce *data2*.

Příkaz *MigrateArticles* každý článek před migrací zformátuje. První přidá dodatečného autora na konec článku, jak jsme si popisovali v kapitole 2.5.4, poté převede URL obrázků na URL ze systému WordPress a nakonec obalí vybrané elementy z konstanty *GUTENBERG_BLOCKS_COMMENTS* komentáři určující Gutenberg blok. Obrázky zároveň obalí odkazem na zdroj samotného obrázku, aby si mohli uživatelé obrázek otevřít v plné velikosti, popř.

k využití nějakého pluginu na galerie. My použijeme plugin WP Featherlight, který nám umožní obrázky po kliknutí prohlížet v ostylované galerii.

U článku ještě přeložíme původní rubriku na novou podle konstanty *CATEGORY_MAP* a začneme migrovat do tabulky *wp_posts*. Rubriky a štítky nastavíme přidáním záznamů do tabulky *wp_term_relationships*. Pro každý článek ještě ukládáme meta informaci o ID z původního systému.

Tím, že se nám po migraci vytvořili nové záznamy článků, můžeme získat jejich ID, a tak dodělat poslední dvě věci.

První aktualizujeme počty použitých taxonomií, tedy počet článků spadající pod rubriku, popřípadě pod štítek, tento údaj se totiž nepočítá v systému, ale ukládá se jako číslo do databáze. Při neopatrné migraci tedy může docházet k nekonzistenci údajů.

Nakonec přeložíme URL článků, které odkazují v rámci webu. Existují dvě varianty URL, které redaktoři používali. První varianta je URL ve tvaru „data.php?idx=*číslo*“, kde za parametr *idx* je dosazeno ID článku z původního systému. Podle tohoto čísla zjistíme ID našeho přemigrovaného článku a přepíšeme na WordPress URL článku, kterou získáme pomocí poslední přesměrované URL z GUID adresy. Druhá varianta URL je absolutní ve tvaru „casopis.mensa.cz/url/clanku“. Článků, kde se tato varianta nachází, je dohromady jen 7, bude tedy efektivnější tyto odkazy upravit ručně. Avšak pokud bychom toto chtěli automatizovat, museli bychom danou URL navštívit crawlerem a získat například obsah tagu *title*, kde bychom očekávali název článku, podle kterého bychom našli záznam ve WordPress databázi a na daný článek mohli použít URL.

Příkaz obsahuje *ProgressBar* a můžeme ho spustit příkazem *./bin/console migrate:articles*.

3.2.7 Příkaz pro migraci všeho

Příkaz *MigrateAll* zajišťuje migraci všech dat ve správném pořadí. Postupně spouští příkazy:

1. *migrate:users*
2. *migrate:files*
3. *migrate:categories*
4. *migrate:tags*
5. *migrate:articles*

Jelikož je příkaz implementován jako volání předchozích příkazů, můžeme vidět jednotlivé výpisy aktuálního stavu migrace v konzoli. Příkaz můžeme spustit pomocí *./bin/console migrate:all*.

3.2.8 Grafické uživatelské prostředí

Uživatelské prostředí implementujeme jako seznam tlačítek a přidružených ukazatelů průběhu. Po kliknutí na tlačítko se spustí příslušná migrace dané entity a začne se ukazovat průběh migrace v ukazateli. Poslední tlačítko se bude jmenovat „Migrovat vše“ a spustí po kliknutí všechna tlačítka (respektive jejich akce) ve správném pořadí. Uživatelské prostředí bude přístupné přes výchozí URL aplikace, tedy jako hlavní stránka.

Samotné provedení migrace určité entity bude provádět GET požadavek na URL */migrate/entity*, kde za *entity* dosadíme, co chceme přemigrovat. Dále budeme potřebovat adresu, která nám bude vracet aktuální průběh migrace, ta bude mít URL */migrate/progress* a získáme ho opět provedením GET požadavku na tuto URL.

Tyto akce implementujeme v třídě *MigrationController*, která dědí ze třídy *AbstractController* připravené v Symfony, a můžeme díky ní tvořit vlastní řadiče. Co je řadič jsme si představili v podsekci 2.6.3.4.

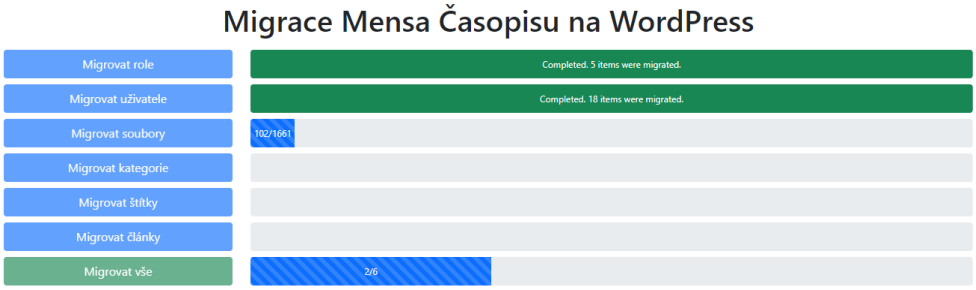
Metoda pro migraci entity jako první vypne časový limit na provedení operace, aby nám proces neskončil chybou pro překročení časového limitu. Poté vytvoří soubor, kam bude samotný migrační proces zapisovat průběh migrace. Podle zvolené entity spustí příslušný migrační proces a po jeho dokončení zapíše do souboru, který vytvořil v předchozím kroku, že je migrace dokončena. Metoda v posledním kroku nastaví zpět časový limit pro provedení operace na původní hodnotu.

Metoda pro získání aktuálního stavu migrace je vcelku prostá, přečte obsah souboru, do kterého se zapisuje průběh a vrátí ho jako odpověď. Průběh migrace se do souboru zapisuje ve formátu *progress/total*, kdy *progress* vyjadřuje počet již přemigrovaných entity a *total* celkový počet entit k migraci.

Pohled implementujeme jako šablonu v šablonovacím jazyku Twig a graficky ostylujeme pomocí CSS frameworku Bootstrap 5. Události pro kliknutí na tlačítka implementujeme pomocí jazyku JavaScript. Po kliknutí na tlačítko migrující určitou entitu jako první vypneme všechna tlačítka, aby uživatel nemohl spouštět více migrací najednou. Dále nastavíme ukazatel průběhu na počáteční stav a pošleme GET požadavek na URL, na které se nám provede migrace. Zároveň se spustí posílání požadavků pro zjištění aktuálního stavu migrace, jakmile se vrátí výsledek, posíláme okamžitě další, dokud se nevrátí, že migrace byla dokončena. Podle výsledků aktualizujeme ukazatel průběhu a když je migrace kompletní, graficky znázorníme výsledek. Poté co je migrace dokončena, zapneme znovu všechna tlačítka, aby se na ně dalo kliknout.

Tlačítko „Migrovat vše“ zavolá všechny akce tlačítek ve správném pořadí, podobně jako u konzolové verze, zároveň však nechá vypnutá tlačítka po celou dobu migrace všech entit, aby nedocházelo k problikávání a krátkodobé možnosti kliknutí. Zároveň aktualizuje svůj vlastní ukazatel průběhu, který udává, kolik druhů entit je již přemigrováno.

Pokud při migraci dojde k nějaké chybě, zobrazí se tzv. „flash message“,



Obrázek 3.3: Grafické uživatelské prostředí pro migraci

která slouží k notifikaci uživatele [26]. Uživatele tedy upozorníme, že nastala chyba, a migraci přerušíme, zároveň oznamujeme, že detail chyby může nalézt v logu (záznam chyb). Pro záznam chyb souvisejících s migrací jsme nastavili vlastní soubor *migration.log*, který jsme nadefinovali v konfiguraci knihovny Monolog, která je součástí frameworku Symfony.

3.3 Příprava produkčního prostředí

Prostředí, na kterém bude běžet výsledný web, nám připravil zákazník prostřednictvím hostingové služby WEDOS Internet. Byl připraven webový server a databáze, a byly nám zaslány přístupy. Na server jsme dostali pouze SFTP přístup, to nám znemožňuje spuštění migrace přes konzoli, a tak budeme muset využít jiný způsob, jak naplnit databázi daty. Zároveň databáze byla nastavena tak, aby k ní nešlo přistoupit z jiného místa než webu samotného. Jedinou výjimkou je administrace phpMyAdmin zřízená od hostingu. Nemůžeme tedy bohužel ani pustit migraci lokálně s tím, že bychom nastavili přístupy na produkční databázi.

Nabízí se tedy dva způsoby jak přemigrovat data do platformy WordPress. První způsob je, že bychom požádali hosting, aby nám nastavil subdoménu, nebo nějakou cestu k migrační aplikaci, a migraci bychom pak mohli spustit přes GUI. Jelikož aplikace není nijak zabezpečena autentizací uživatele, musel by hosting nastavit alespoň Basic access authentication [27] na straně webového serveru.

Druhý způsob je přemigrovat data lokálně do lokální databáze, tu následně exportovat do formátu SQL a importovat ji přes administraci phpMyAdmin. Tento způsob je jak jednodušší, tak bezpečnější, proto zvolíme jej. V exportované databázi musíme udělat jedinou věc navíc, a to najít a nahradit všechny výskyty lokální URL za produkční URL, poté je vše připraveno k importu.

Nahrajeme tedy aktuální verzi systému WordPress přes SFTP na produkční server a nainstalujeme podobně jako jsme to dělali v kapitole 3.1.2, jen doplníme údaje k produkční databázi. Poté nainstalujeme příslušné plu-

giny (Members, WP Featherlight a WP Attachment Usage) a importujeme databázi přes phpMyAdmin. Nakonec doděláme manuálně pár úprav, jako nastavení šablony, předělání dohodnutých článků na stránky a úpravy nastavení. V tuto chvíli je nový web časopisu Mensa připraven a původní uživatelé se do něj mohou přihlásit hned poté, co si přes možnost „Zapomenuté heslo“ nechají poslat odkaz na e-mail, kde si nastaví heslo nové.

Závěr

Naším cílem bylo přemigrovat stávající data časopisu Mensa v původním systému MultiCMS na platformu WordPress.

Po vysvětlení některých termínů ohledně problematiky migrace jsme si představili konkrétní nástroje, kterými by se dalo migrace docílit. Nicméně pro maximální automatizaci procesu jsme se rozhodli napsat vlastní aplikaci.

V rámci bakalářské práce jsme tedy vytvořili takovou aplikaci, která dokáže danou migraci provést. Aplikaci jsme vytvořili v PHP frameworku Symfony a umožnili migraci spouštět v konzoli a pomocí grafického uživatelského rozhraní.

Pomocí hotové aplikace jsme poté přemigrovali aktuální data klienta a pomohli tak obsahově naplnit nový web, který je již na platformě WordPress.

Vzhledem k další spolupráci s Mensou České republiky se může aplikace rozšířit o nějaké další funkcionality. Například, pokud se klient rozhodne používat nějakou jinou strukturu obsahu článků, můžeme upravit část migrace, která formátuje obsah.

Ačkoliv je aplikace určena na jednu migraci, a po jejím dokončení se může přestat používat, její části ohledně modifikace dat na platformě WordPress se mohou recyklovat v jiných nástrojích.

Literatura

[1] WordPress: Database Description. *WordPress [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://codex.wordpress.org/Database_Description`

[2] Kinsta: What Is a Content Management System (CMS)? *Kinsta [online]*, březen 2021, [cit. 2021-03-10]. Dostupné z: `https://kinsta.com/knowledgebase/content-management-system/`

[3] WordPress: Meet WordPress. *WordPress [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://wordpress.org/`

[4] Kinsta: WordPress market share. *Kinsta [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://kinsta.com/wordpress-market-share/`

[5] Michálek, M.: Gutenberg: Co je zač ten emoce vyvolávající editor (nejen) pro WordPress? *Vzhůru dolů – webová kodéřina ze všech stran [online]*, listopad 2021, [cit. 2021-03-10]. Dostupné z: `https://www.vzhurudolu.cz/prirucka/gutenberg`

[6] WordPress: Pluginy. *WordPress [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://cs.wordpress.org/plugins/`

[7] WordPress: Požadavky. *WordPress [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://cs.wordpress.org/about/requirements/`

[8] MultiCMS: Redakční systém MultiCMS. *MultiCMS [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://www.multicms.net/`

[9] Navicat: Navicat Premium. *Navicat [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://www.navicat.com/en/products/navicat-premium`

[10] Talend: Open Studio for Data Integration. *Talend [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://www.talend.com/products/data-integration/data-integration-open-studio/`

[11] WordPress: WP Featherlight – A Simple jQuery Lightbox. *WordPress [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://cs.wordpress.org/plugins/wp-featherlight/`

[12] WordPress: Attachment Usage. *WordPress [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://wordpress.org/plugins/attachment-usage/`

[13] Docker: Docker. *Docker [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://www.docker.com/`

[14] Git: Git. *Git [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://git-scm.com/`

[15] Symfony: Symfony. *Symfony [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://symfony.com/`

[16] Symfony: The Console Component. *Symfony Docs [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://symfony.com/doc/current/components/console.html`

[17] Symfony: Doctrine. *The Doctrine Project [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://www.doctrine-project.org/`

[18] Symfony: Twig. *Symfony Twig [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://twig.symfony.com/`

[19] Gamma, E.: *Návrh programů pomocí vzorů: stavební kameny objektově orientovaných programů*. Praha: Grada, 2003, ISBN 80-247-0302-5.

[20] JetBrains: PhpStorm. *JetBrains [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://www.jetbrains.com/phpstorm/`

[21] Docker: MariaDB. *Docker Hub [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://hub.docker.com/_/mariadb`

[22] Docker: PHP. *Docker Hub [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://hub.docker.com/_/php`

[23] Wikipedia: Dependency injection. *Wikipedia [online]*, 2021, [cit. 2021-03-10]. Dostupné z: `https://en.wikipedia.org/wiki/Dependency_injection`

[24] Cutts, M.: Google does not use the keywords meta tag in web ranking. *Google Search Central [online]*, září 2009, [cit. 2021-03-10]. Dostupné z: `https://developers.google.com/search/blog/2009/09/google-does-not-use-keywords-meta-tag`

[25] Seznam: Fulltextové vyhledávání. *Seznam.cz nápověda [online]*, 2021, [cit. 2021-03-10]. Dostupné z: https://napoveda.seznam.cz/cz/fulltext-hledani-v-internetu/optimalizace-faq/#keywords

[26] Symfony: Controller. *Symfony Docs [online]*, 2021, [cit. 2021-03-10]. Dostupné z: https://symfony.com/doc/current/controller.html#flash-messages

[27] Mozilla: HTTP authentication. *MDN Web Docs [online]*, 2021, [cit. 2021-03-10]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/HTTP/Authentication#basic_authentication_scheme

Příloha A

Seznam použitých zkratek

CMS Content Management System

CSV Comma-separated values

DBAL Database Abstraction Layer

GDPR General Data Protection Regulation

GUI Graphical user interface

GUID Globally Unique Identifier

HTTPS Hypertext Transfer Protocol Secure

IDE Integrated Development Environment

MIME Multipurpose Internet Mail Extensions

MVC Model-view-controller

PHP PHP: Hypertext Preprocessor

RPC Remote procedure call

SEO Search Engine Optimization

SFTP Secure File Transfer Protocol

SQL Structured Query Language

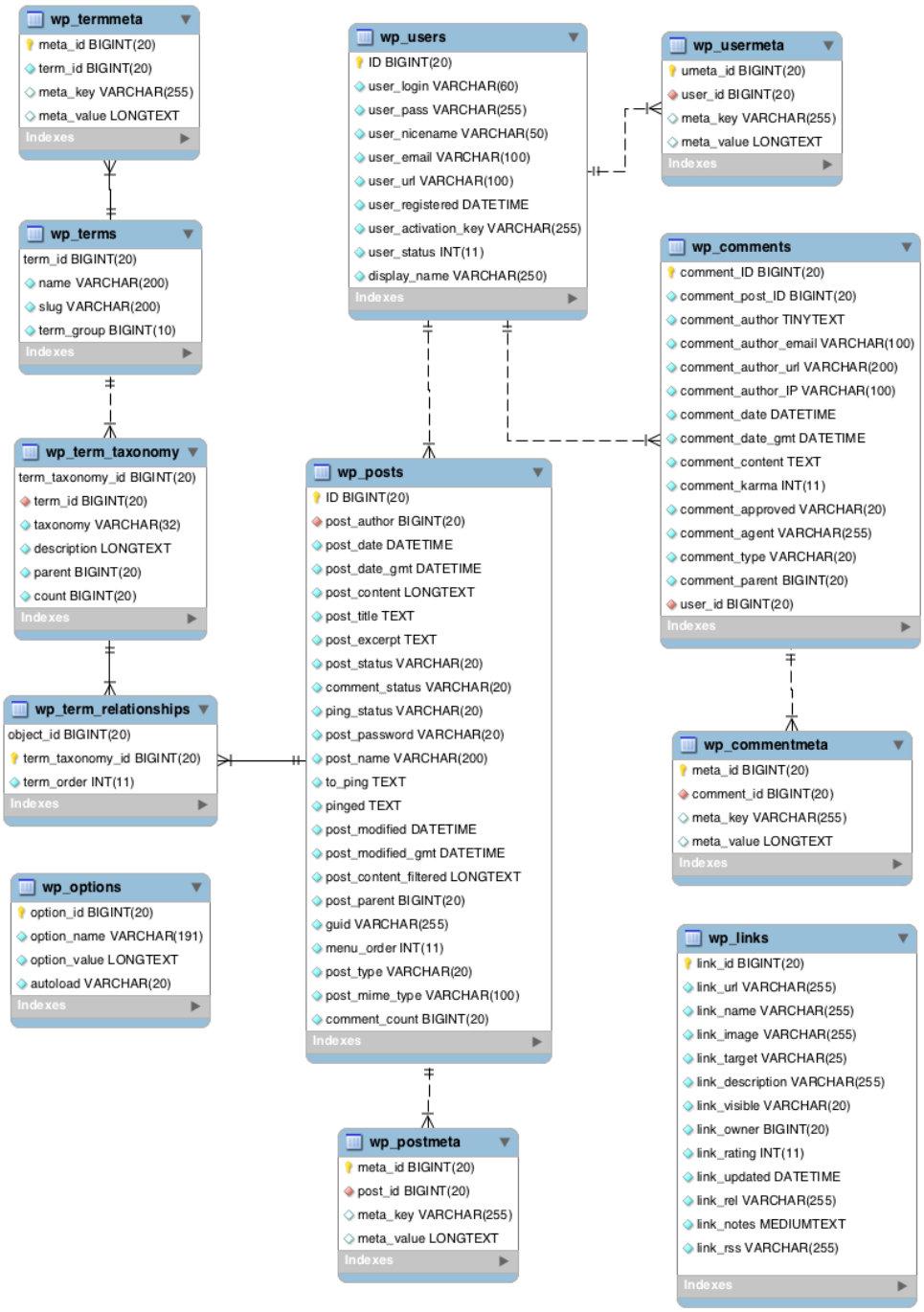
URL Uniform Resource Locator

WYSIWYG What you see is what you get

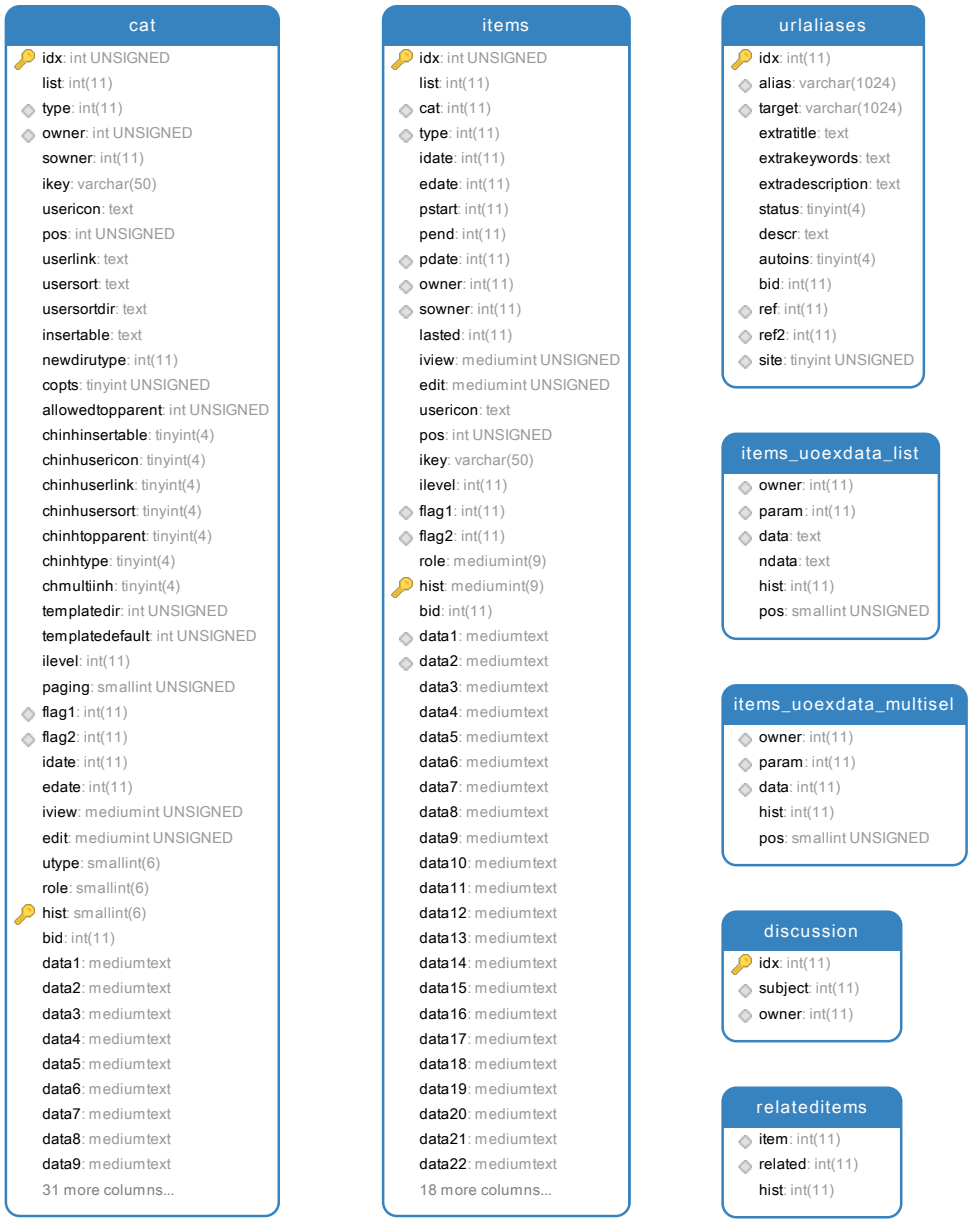
XML Extensible Markup Language

Příloha B

Databázové diagramy CMS systémů



Obrázek B.1: Databázový diagram systému WordPress [1]



Obrázek B.2: Databázový diagram systému MultiCMS

Příloha C

Obsah přiloženého SD

- `readme.txt` ..... stručný popis obsahu SD
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF