Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Návrh a implementace redakčního systému v ASP.NET Core

Bc. Jaromír KEJDA

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jaromír Kejda

Informatika

Název práce

Návrh a implementace redakčního systému v ASP.NET Core

Název anglicky

Design and Implementation of Content Management System in ASP.NET Core

---

Cíle práce

Hlavním cílem této práce je implementace základní funkcionality webového redakčního systému pro tvorbu webových stránek s využitím technologie ASP.NET Core MVC a jazyka C#. Vedlejším cílem práce je provést návrh tohoto systému prostřednictvím standardních postupů softwarového inženýrství.

Metodika

Metodika této práce je založena na analyticko-syntetickém přístupu k odborným literárním a internetovým zdrojům, které poskytnou teoretický základ této práce. Prostřednictvím standardních postupů softwarového inženýrství autor nejprve provede návrh redakčního systému, a poté přistoupí k samotné implementaci prostřednictvím nástrojů platformy ASP.NET Core MVC a jazyka C#. Nakonec autor provede diskusi, kde kriticky zhodnotí výsledek své práce a navrhne další možný vývoj aplikace.

Doporučený rozsah práce

60-80 stran

Klíčová slova

ASP.NET, MVC, .NET Core, C#, Microsoft, CMS, client-side, server-side, ORM, datová perzistence

---

Doporučené zdroje informací

BARKER, Deane. Web content management: systems, features, and best practices. Boston: O'Reilly, 2016. ISBN 978-1-491-90812-9.

FREEMAN, Adam. Pro ASP.NET Core MVC. New York, NY: Apress, 2016. ISBN 978-1-4842-0398-9.

MACDONALD, Matthew. Beginning ASP.NET 4.5 in C#. New York, NY: Apress, 2012. ISBN 978-143-0242-529.

VRANA, Ivan. Projektování informačních systémů s UML. V Praze: Česká zemědělská univerzita, Provozně ekonomická fakulta, 2008. ISBN 978-80-213-1817-5.

Předběžný termín obhajoby

2018/19 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 24. 1. 2019

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 24. 1. 2019

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 01. 03. 2019

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Návrh a implementace redakčního systému v ASP.NET Core" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 1. 3. 2019 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Jiřímu Brožkovi, Ph.D. za vstřícné a inspirativní vedení, stejně tak jako za jeho odborné znalosti a návrhy, které se podílely nemalým dílem na podobě této práce.

Návrh a implementace redakčního systému v ASP.NET Core

Abstrakt

Hlavním cílem této práce je implementace základní funkcionality webového redakčního systému pro správu obsahu webových stránek s využitím technologie ASP.NET Core MVC a jazyka C#. Praktická část práce popisuje postup samotné implementace tohoto softwarového řešení s ohledem na klíčové aspekty fungování aplikace. Pozornost je věnována implementaci doporučených postupů použitých technologií, například využití objektově-relačního mapování pomocí technologie Entity Framework Core, nebo implementace návrhového vzoru *dependency injection*. Prostor je také věnován vlastnímu systému konfigurace aplikace, jednotlivých vzhledů a mapování URL adres. Poslední kapitoly práce se věnují jednotlivým implementovaným funkcionalitám a popisují nasazení aplikace do provozu.

Vedlejším cílem práce je provést návrh tohoto systému prostřednictvím standardních postupů softwarového inženýrství. K tomuto účelu byly použity modelovací prostředky jazyka UML. Autor se v práci věnuje návrhu funkcionality aplikace, databáze a adresářové struktury výsledného programu.

Klíčová slova: ASP.NET, MVC, .NET Core, C#, Microsoft, CMS, client-side, server-side, ORM, datová perzistence

Design and implementation of content management system in ASP.NET Core

Abstract

The primary objective of this thesis is to implement the basic functionality of a web content management system for a website administration using ASP.NET Core MVC technology and C# language. The practical part of this work describes the implementation process of this software solution and focuses on the key features of the application. The project incorporates the best practices within the used technologies, for instance, the usage of an object-relational mapping of the Entity Framework Core, or a dependency injection pattern. This work also deals with my own custom configuration system of the application, themes and URL mapping. The last chapters of this thesis describe the individual application's features and deployment.

The secondary objective of this thesis is to conduct a design of this system using standard methods of software engineering. For this purpose, the author used standard tools of the UML language. This thesis also further discusses the functionality, database, and the directory structure of the application.

Keywords: ASP.NET, MVC, .NET Core, C#, Microsoft, CMS, client-side, server-side, ORM, data persistence

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Seznam použitých zkratek

ADO.NET – *ActiveX Data Objects*, sada .NET knihoven pro připojení k databázi

AJAX – *Asynchronous JavaScript And XML*, technologie pro asynchronní HTTP volání

API – *Application Programming Interface*, aplikační rozhraní pro tvorbu programů

ARPANET – *Advanced Research Projects Agency Network*, počítačová síť, která se stala zárodkem dnešního internetu

ASP – *Active Server Pages*, technologie pro tvorbu webových aplikací

CIL – *Common Intermediate Language,* nejnižší člověkem čitelný programovací jazyk na platformě .NET

CLR – *Common Language Runtime*, běhové prostředí vykonávající běh programů platformy .NET

CRUD – zkratka pro operace *create, read, update* a *delete*

CSRF – *Cross-site request forgery,* jedna z metod útoku na webové aplikace

CTS – *Common Type System,* společný typový systém

GDPR – *General Data Protection Regulation,* obecné nařízení o ochraně osobních údajů

HTTP – *Hypertext Transfer Protocol*, internetový protokol pro výměnu hypertextových dokumentů ve formátu HTML

IANA – *Internet Assigned Numbers Authority*, autorita pro přidělování IP adres a správy DNS (*Domain Name System*)

IDE – *Integrated Development Environment,* software pro vývoj aplikací

IIS – *Internet Information Services*, webový server od společnosti Microsoft

I/O operace – *input-output operace*, operace čtení nebo zápisu dat

ISOC – *Internet Society*, mezinárodní organizace pro vedení internetových standardů

MS SQL Server – *Microsoft SQL Server*, databáze od společnosti Microsoft

MVC – *Model-view-controller*, jedná se o softwarovou architekturu

NPM – *Node Package Manager,* balíčkovací systém

ORM – *Object-relational mapping*, objektově-relační mapování

SEO – *Search Engine Optimization*, optimalizace pro vyhledávače

TCP/IP – *Transmission Control Protocol/ Internet Protocol,* rodina protokolů pro komunikaci v počítačové síti

UWP – *Universal Windows Platform*, platforma od společnosti Microsoft pro tvorbu univerzálních aplikací

UX Design – *User Experience Design*, návrh interakce uživatele se systémem

W3C – *World Wide Web Consortium*, mezinárodní konsorcium vyvíjející webové standardy

WPF – *Windows Presentation Foundation*, knihovna pro tvorbu grafického uživatelského rozhraní na platformě Windows

WWW – *World Wide Web*, jedná se o službu internetu poskytující systém webových stránek

1 Úvod

Jedním ze současných trendů vztahující se k oblasti informačních a komunikačních technologií je nutnost efektivní sebeprezentace, ať už se jedná o snahu podnikatelů zviditelnit svůj podnik, nebo o pouhé osobní webové stránky sloužící jako pracovní portfolio či jako volnočasový web. Díky masovému rozšíření počítačů a stále dostupnějšímu připojení k internetu se k tvorbě webového obsahu uchyluje i širší skupina laiků za cílem, jak často uvádějí reklamní slogany: „Vytvořit webové stránky snadno a rychle bez předchozích znalostí.“ Tento cíl je možné realizovat například prostřednictvím speciálních aplikací, které jsou nazývány webové redakční systémy.

Webové redakční systémy slouží ke snadné správě online obsahu, který je typicky šířen pomocí celosvětové sítě internet, nebo v případě interního podnikového nasazení také v rámci intranetu. Tyto systémy umožňují i méně technicky zdatným uživatelům vytvářet webové prezentace i bez předchozích znalostí jazyků jako jsou HTML, JavaScript či CSS. O samotný provoz, údržbu a pravidelné zálohy se poté stará provozovatel redakčního systému, který typicky poskytuje tuto službu ve velkém a nabízí ji více zákazníkům současně.

Webových redakčních systémů pro správu obsahu existuje mnoho a liší se různými atributy, například cenou, použitou technologií, dostupností služby, disponibilní funkcionalitou a dalšími doplňkovými službami, které více či méně souvisí s konkrétně použitým redakčním systémem. Předmětem této práce je zaměřit se na tento typ aplikací, navrhnout a implementovat jednoduchý redakční systém s využitím relativně nové technologie ASP.NET Core, který využije nových vylepšení a prostředků, která tato platforma přinesla, za cílem umožnit vytvoření jednoduché webové prezentace jak laikům, tak i zkušeným uživatelům.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem této práce je implementace základní funkcionality webového redakčního systému pro správu obsahu webových stránek s využitím technologie ASP.NET Core MVC a jazyka C#. Vedlejším cílem práce je provést návrh tohoto systému prostřednictvím standardních postupů softwarového inženýrství.

2.2 Metodika

Metodika teoretické části práce je založena na studiu českých i zahraničních elektronických a literárních pramenů. Autorem vybrané technické aspekty a konstrukce uvedené v teoretické části práce následně autor aplikuje vhodným způsobem v části praktické.

Praktická část práce vychází z provedeného návrhu a spočívá v samotné implementaci redakčního systému, ke které bude použit framework ASP.NET Core MVC od společnosti Microsoft spolu s dalšími server-side a client-side technologiemi.

3 Teoretická východiska

Teoretická část práce čtenáře nejprve uvede do problematiky webového vývoje vzhledem ke klientské a serverové části aplikace. Následně se práce velmi zběžně zabývá problematikou webových redakčních systémů a zmiňuje několik světově rozšířených řešení. Po tomto úvodu je již popsána hlavní použitá technologie, tedy ASP.NET Core MVC, kde jsou vysvětleny základní principy vývoje na této platformě. Jelikož problematice datové perzistence, které se tato práce dotýká, jsem se podrobněji věnoval ve své bakalářské práci: *Prostředky platformy .NET pro zpracování dat a zajištění datové perzistence,* budu zde tuto problematiku adresovat již jen ke vztahu k webovým aplikacím.

3.1 Úvod do problematiky webového vývoje

Vývoj software pro určitou platformu s sebou nese určitá specifika. Ať už hovoříme o různých hardwarových či softwarových platformách, aplikace jsou tvořeny pomocí odlišných programovacích jazyků, frameworků, vývojových nástrojů, jsou určeny pro odlišná zařízení a uživatele s různorodými potřebami. Web, jako specifická platforma vývoje má své charakteristiky, kterým se věnují následující kapitoly.

3.1.1 Specifika webového vývoje

Od roku 1969, kdy byla vytvořena síť ARPANET, prošla celosvětová síť internet mnohými změnami a trendy. Tyto změny šly ruku v ruce s vývojem síťové konektivity, vznikem internetových protokolů a byly ovlivněny i vznikem organizací dohlížející na standardizaci v síti internet, například IANA, ISOC, nebo W3C. Za počátek webu tak, jak ho známe dnes se dá považovat rok 1990, kdy Tim-Berners Lee vytvořil službu WWW a spustil první webový server na světě *info.cern.ch*. Ve stejném roce vznikl i celosvětově úspěšný značkovací jazyk HTML pro tvorbu hypertextem provázaných webových stránek. (*World Wide Web*, nedatováno)

Většina služeb internetu, včetně již zmíněné služby WWW, funguje ve výpočetním modelu klient/server. Pro výměnu webových stránek po síti se poté využívá bezestavový protokol HTTP. Již zmíněná služba WWW pro komunikaci používá rodinu protokolů TCP/IP. Mezi výhody modelu klient/server patří například centralizace zdrojů a mezi nevýhody to, že každá aplikace musí mít svého vlastního klienta, což ale v případě služby WWW odpadá,

protože tuto roli hraje webový prohlížeč, který slouží jako univerzální tenký klient. (Peterka, 2014)

3.1.2 Server-side

Celosvětová síť internet je decentralizovaná síť, kde jednotlivými uzly sítě jsou v případě služby WWW webové servery, které obsluhují požadavky klientů. Ve srovnání například s klasickou desktopovou monolitickou aplikací je zde tedy nutné webovou aplikaci rozdělit na dvě části, přičemž jedna běží na serveru a druhá tvoří klientskou část (Peterka, 2014). Mezi typické zástupce sever side technologií patří například v této práci použitý ASP.NET Core MVC framework, nebo dále jazyky PHP, ASP, jazyk Java společně s technologií JSP, nebo Python (*Server-side scripting*, nedatováno).

3.1.3 Client-side

V roli tenkého klienta poté vystupuje internetový prohlížeč, který pomocí HTTP požadavků získává ze serveru odpovědi v podobě webových stránek, nebo jiného obsahu. Jak píše Peterka (2014), řez mezi client-side a server-side by měl být volen tak, aby se přenášelo co nejméně dat. Tato situace tedy vypadá tak, že client-side je strana, kde se nachází uživatel a server-side je tam, kde se nachází sdílená data pro více uživatelů. Dále Peterka (2014) uvádí, že client-side je aktivní stranou, zatímco server je pasivní a čeká na požadavky od klienta. Mezi client-side technologie patří dle stránek[1] Microsoftu například jazyky HTML, CSS, JavaScript, a další frameworky jako JQuery, Angular, React apod.

3.1.4 Webové redakční systémy

Webové redakční systémy jsou jedním z mnoha systémů pro správu obsahu, jedná se o takzvané CMS (*Content Management Systems*). Jak píše Barker (2016, s. 5): „*A content management system (CMS) is a software package that provides some level of automation to the tasks required to effectively manage content*.“ Barker zde pojmem redakční systém má na mysli webový redakční systém, jak předesílá v úvodu své knihy *Web Content Management Systems*. Za jeden z prvních pokusů o správu obsahu se dá považovat alexandrijská knihovna, což byla jedna z největších a nejslavnějších knihoven starověku. V dnešní terminologii byli tehdejší knihovníci správci obsahu. Od počátku 90. let, kdy byl

[1] Více informací například zde: https://docs.microsoft.com/en-us/dotnet/standard/modern-web-apps-azure-architecture/common-client-side-web-technologies

zrozen web, rostla potřeba vývoje systémů, které usnadní správu obsahu, jehož množství strmě narůstalo a stále narůstá. (Barker, 2016, s. 5)

Jak již bylo řečeno, redakční systémy slouží pro správu obsahu. Jak píše Barker (2016, s. 7), pojem *content*, česky *obsah*, se od ostatních informací liší tím, že vzniká redakčním procesem, tedy procesem přípravy informace pro publikaci, která zahrnuje autorskou tvůrčí práci, editování, kritiku, schvalování, porovnávání a kontrolování. K tvorbě obsahu se poté váží otázky, co má být předmětem obsahu, pro koho je obsah určen, jak dlouhý obsah má být a jestli má být doplněn dalšími multimediálními materiály. (Barker, 2016, s. 7)

3.1.4.1 Wordpress

Wordpress je open-source webový redakční systém pro vytvoření a spravování moderní webové stránky, který je přístupný i méně technicky zdatným uživatelům. Wordpress byl původně určen pro blogy, ale postupem času se vyvinul v bohatý a plnohodnotný CMS. Systém WordPress byl vydán v květnu 2003. (Król, 2015, s. 1)

Na Wordpressu v roce 2017 je postaveno 27 % internetových stránek, je dostupný v 50 jazycích a má k dispozici více než 3000 zdarma dostupných vzhledů a 48 500 pluginů (*Shocking WordPress Stats,* 2017). V žebříčku podílu na trhu mezi ostatními redakčními systémy strojí WordPress na vrcholu s podílem téměř 60 % (*Popular CMS by Market Share*, 2017).

3.1.4.2 Joomla

Joomla je podobně jako WordPress open-source CMS pro vytvoření a spravování webové stránky, který je dostupný od roku 2005. V roce 2018 má na trhu podíl pouhých 6,6 % (*Joomla vs. WordPress Comparison*, 2018). Mezi další redakční systémy s podobně nízkým podílem patří například Drupal, Magento, Blogger, Shopify, nebo třeba PrestaShop. Již zmíněné redakční systémy WordPress a Joomla byly napsány v jazyce PHP. Pro jejich nasazení se často používá softwarový balíček známý pod zkratkou LAMP (Linux, Apache, MySQL, PHP).

3.1.4.3 Orchard

Systém Orchard je zdarma dostupný, open-source a na komunitu zaměřený CMS, postavený na technologii ASP.NET MVC. Verze *OrchardCore* je implementovaná v technologii ASP.NET Core MVC. (*Orchard CMS*, 2018)

Orchard byl uvolněn firmou Microsoft v lednu 2011 spolu s technologiemi ASP.NET MVC 3, IIS Express, SQL CE 4 a například i programem WebMatrix. Cílem těchto technologií bylo lépe zpřístupnit webový vývoj na systému Windows. Vývoj systému Orchard započal pod organizaci *Outercurve Foundation (*dříve známá také jako *CodePlex Foundation),* což byla nezávislá a nezisková organizace založená firmou Microsoft v roce 2009. (Zablocki, 2012, s. 10)

Od roku 2014 je Orchard financován organizací *.NET Foundation*, která zaštiťuje i další projekty, jako například .NET Core, EntityFramework, Managed Extensibility Framework, NuGet, nebo redakční systém Umbraco. (*.NET Foundation Projects*, 2018)

3.2 Platforma .NET a webový vývoj

Existuje nespočetné množství dohledatelných definic, vysvětlení a pohledů na to, co vlastně .NET Framework je. Je to dáno především obrovskou komplexitou celého tohoto vývojářského ekosystému, který se již dávno neomezuje pouze na operační systém Windows. Navíc platforma .NET v posledních letech získala další přírůstek do své infrastruktury, a to multiplatformní framework .NET Core. (*.NET Core Guide*, 2018)

Ve své bakalářské práci (*Prostředky platformy .NET pro zpracování dat a zajištění datové perzistence,* 2017, s.18-21) jsem se podrobněji věnoval infrastruktuře platformy .NET, běhovému prostředí CLR, společnému typovému systému CTS a mezilehlému jazyku CIL. Dále jsem ve zmíněné publikaci věnoval kapitolu jazyku C#. Z tohoto důvodu se již zde nebudu věnovat zmíněným položkám podrobně, ale pouze uvedu platformu .NET do kontextu webového vývoje.

V oficiální dokumentaci společnosti Microsoft je možné se dočíst krátkou definici platformy .NET: *„.NET is a general purpose development platform.“.* Dokumentace dále shrnuje klíčové atributy tohoto frameworku: podpora více programovacích jazyků, asynchronní programování, automatická správa paměti, práce s neřízenými (*unamnaged*) zdroji, typová bezpečnost, delegáty, lambda výrazy (jazyk LINQ) a podpora generických typů. (*Tour of .NET*, 2017)

Každá .NET aplikace je vyvíjena a provozována na jedné nebo více implementaci technologie .NET. Mezi implementace technologie .NET patří (*.NET architectural components*, 2017):

- **.NET Framework** – původní framework, existující od roku 2002
- **Mono –** multiplatformní implementace .NET od společnosti Xamarin
- **UWP** – slouží k vytváření moderních aplikací mimo jiné pro tablety, chytré telefony, Xbox One a počítače s Windows 8 nebo novější verzí
- **.NET Core** – multiplatformní implementace .NET běžící pod Windows, macOS a Linux, dostupná od června 2016

Každá z těchto implementací má jedno nebo více běhových prostředí, svoje knihovny (*class library),* dále volitelně může v sobě obsahovat aplikační framework (tím je například ASP.NET, Windows Forms, nebo WPF) a popřípadě doplňkové vývojářské nástroje. *(.NET architectural components,* 2017)

Dále existuje API specifikace společná všem těmto implementacím, která se nazývá *.NET Standard*. Implementací této specifikace je tzv. BCL (*Base Class Library*). Kód vyhovující této specifikaci může běžet pod libovolnou výše vypsanou implementací .NET. (*.NET architectural components*, 2017)

Platforma .NET byla vydána 13. února 2002 (*.NET Framework version history*, 2018). Co se týká technologií od společnosti Microsoft pro tvorbu webových stránek před tímto rokem, byla používána technologie ASP, později označována jako Classic ASP z roku 1996. Tato technologie podporovala jazyky VBScript, JScript, nebo PerlScrtipt (*Active Server Pages*, 2018).

Pro přehled uvádím seznam ASP technologií pro tvorbu webových aplikací (*ASP and ASP.NET Tutorials*, nedatováno):

- Classic ASP
- ASP.NET Web Forms
- ASP.NET MVC (verze MVC 3 přidala tzv. Razor Syntax)
- ASP.NET Web Pages
- ASP.NET API
- ASP.NET Core (obsahuje architekturu MVC, nové Razor Pages a přepracované ASP.NET API)

3.2.1 ASP.NET

Technologie ASP.NET vznikla v roce 2002 a byla navázána na server IIS od Microsoftu. Tato technologie, také známá jako ASP.NET Web Forms se snažila skrývat podstatu fungovaní bezestavového protokolu HTTP a skrývala před programátorem dokonce i kód HTML, který byl generován automaticky na základě použitých tzv. *server controls*. Výsledné automaticky vygenerované HTML často nesplňovalo aktuální standardy a nebylo jej možné lehce stylovat pomocí CSS a snadno používat s JavaScriptem, kvůli dlouhým automaticky vygenerovaným identifikátorům. ASP.NET Web Foms byla ohromná abstrakce nad celým výpočetním modelem a imitovala klasický desktopový vývoj aplikací pomocí událostí. Cílem Microsoftu bylo co nejvíce přiblížit webový vývoj tomu desktopovému. Celá stránka se chovala jako jeden velký formulář, který při každém požadavku ukládal informace do tzv. *view state*, což neúměrně zatěžovalo síťový provoz zbytečně velkým objemem přenášených dat. (Freeman, 2016, s. 4)

3.2.2 ASP.NET MVC

V říjnu roku 2007 Microsoft oznámil novou platformu ASP.NET MVC, jejímž cílem bylo odstranit nedostatky Web Forms. Microsoft dále využil v této technologii JQuery framework, zbavil se automaticky generovaného HTML a uvolnil zdrojové kódy pro veřejnost. Tento nový framework, využívající návrhový vzor MVC, se ale neoprostil zcela od minulosti a stále využíval knihovny předešlé technologie ASP.NET. Celá situace byla zkomplikována dále tím, že Microsoft začal přidávat nové vlastnosti z ASP.NET MVC

zpětně do staré verze ASP.NET, což působilo zmatky. Microsoft pochopitelně pokračoval v rozvíjení ASP.NET MVC a přidával další frameworky, jako například Web API, nebo SignalR, které měly svá vlastní specifická nastavení a chování. Výsledný obraz celého frameworku byl značně nesourodý a nepřehledný. (Freeman, 2016, s. 5)

3.2.3 ASP.NET Core MVC

Nakonec v roce 2015 Microsoft oznámil ASP.NET Core MVC – bezplatný a open-source webový framework, který jsem použil v praktické části této práce. Jedná se o další generaci ASP.NET vyvíjenou společností Microsoft a komunitou. První verze byla uvolněna 27. 6. 2016, druhá verze pak 14. 8. 2017. V době psaní této práce je aktuální verze 2.2 z prosince 2018. Platforma ASP.NET Core může běžet jak pod plnohodnotným .NET Frameworkem, nebo pod multiplatformním .NET Core. Jedná se o zcela nový přepis předchozích technologií ASP.NET MVC a ASP.NET Web API do jednoho produktu. Mezi komponenty této nové platformy patří (*ASP.NET Core*, 2018):

- Entity Framework Core
- Identity Core
- MVC Core
- Razor Core

Pro shrnutí, ASP.NET Core je technologie postavená na multiplatformní technologii .NET Core, což je obdoba .NET Frameworku, ale bez specifických API pro operační systém Windows. Výslednou aplikaci je tedy možné spustit nejenom na systému Windows, ale i Linux, nebo OS X/macOS. V případě použití .NET Core knihoven je nabídka funkcí menší, ale Microsoft postupně dodává implementace dalších známých knihoven z .NET Frameworku. (Freeman, 2016, s. 5)

3.2.3.1 MVC architektura v ASP.NET Core

Podobně jako předchozí technologie ASP.NET MVC, se nová technologie ASP.NET Core MVC také opírá o návrhový vzor MVC (*model-view-controller*). Tento návrhový vzor má své počátky již v roce 1978, kdy byl formulován jako část systému Smalltalk v Palo Alto Research Center, přičemž ASP.NET Core MVC je implementací tohoto vzoru. Návrhový vzor MVC je vhodný pro webové aplikace díky jejich životnímu cyklu: Uživatel provede

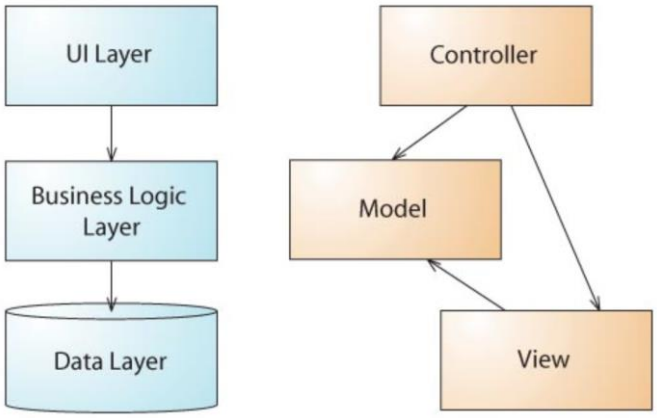
akci pomocí HTTP volání skrze uživatelské rozhraní, aplikace změní svá určitá data a vrátí uživateli nový pohled s novými daty. Další skutečností je to, že se webové aplikace rozdělují do několika logických vrstev, kde každá vrstva má svůj účel a přičemž lze změnit implementaci jedné vrstvy bez nutnosti měnit vrstvy ostatní. Hovoříme poté o *n-tier* architektuře (z hlediska hardwarové implementace), nebo o *n-layer* architektuře (z hlediska softwarové implementace). (Freeman, 2016, s. 6)

Následující tabulka zjednodušeně znázorňuje, vztah jednotlivých vrstev:

Tier (HW)	**Layer (SW)**
databáze	datová vrstva (DAL) – *data access layer*
aplikační server, webový server	business logika
client	prezentační vrstva – GUI, CLI

Tabulka 3.1 – Vícevrstvá architektura aplikace

Jak do tohoto *n-tier/n-layer* schématu vstupuje architektura MVC? Tyto architektury se vzájemně nevylučují a je možné je používat zároveň. Následující obrázek schematicky porovnává sledované architektury (Heathcote, 2011):



Obrázek 3.1 – Třívrstvá architektura a MVC

Zdroj: (Heathcote, 2011)

V levé části obrázku se nachází schéma třívrstvé architektury. V pravé části obrázku je znázorněna architektura MVC. Jak je vidět, v třívrstvé architektuře vrstva grafického rozhraní má přístup pouze k vrstvě business logiky a ta má přístup pouze k datové vrstvě.

Skutečně se tedy jedná o vrstvovitou architekturu. Architektura MVC se liší v tom, že controller je rozhodovacím bodem, který má přístup jak k pohledu, tak k modelu. Schéma MVC tedy proto není vrstvovité, ale tvoří tvar trojúhelníku. (Heathcote, 2011)

Jak vysvětluje smalltalkový programátor a autor architektury MVC, Trygve Reenskaug (1979, s. 1), model, view a controller mají tento účel:

- Model – představuje určitou znalost
- View – jedná se o vizuální reprezentaci modelu
- Controller – jedná se o spojovací článek mezi uživatelem a systémem

Pokud uvedené vztáhneme k ASP.NET Core, které implementuje architekturu MVC, zjistíme, že platí (Atwood, 2008):

- **Model** – Jedná se o třídy (soubory s příponou *.cs*), které pracují s daty, provádějí tedy CRUD operace.
- **View** – Je to uživatelské rozhraní (soubory s příponou *.cshtml*), které obsahují především client-side kód, ale také i server-side kód.
- **Controller** – Jedná se o speciální třídu dědící od třídy *Controller*, spravující uživatelský vstup, změny v datech modelu a to, jaký pohled bude uživateli vrácen.

3.2.3.2 Routing systém v ASP.NET Core

Stejně jako v ASP.NET MVC, i nová verze Core obsahuje routovací systém. Tento systém mapuje jednotlivé fyzické soubory na URL adresy, které již nezávisí na umístění těchto fyzických souborů v adresářovém systému. Díky tomu URL adresy nevypadají například takto: */website/folder1/file1.aspx*, ale mají čitelnější podobu v závislosti na jménu controlleru a akční metody: */home/article/first-tutorial*. Kromě toho, že je druhá URL adresa čitelnější, tak tento tvar má také význam pro vyhledávače, neboť ti přisuzují váhu klíčovým slovům v rámci URL adresy. Tyto vylepšení URL adresy jsou také lépe zapamatovatelné a nevyzrazují vnitřní adresářovou strukturu webu potenciálním útočníkům. (Freeman, 2016, s. 7)

3.2.3.3 Další charakteristiky ASP.NET Core

Když po přibližně patnácti letech vývoje a existence ASP.NET Microsoft oznámil ASP.NET Core, jednalo se o jednu z nejradikálnějších změn na této platformě. Díky tomu, že ASP.NET Core je postaveno na .NET Core, je možné využívat všechny poslední novinky jazyka C#, jako je klíčové slovo *await*, lambda výrazy, anonymní typy, dynamické typy, LINQ apod. U předchozích verzí ASP.NET bylo nutné vyvíjet webové aplikace na systému Windows. Nyní je .NET Core aplikace možné vyvíjet a spouštět i na ostatních nejpoužívanějších operačních systémech. Jelikož je ASP.NET Core MVC open-source projekt, je možné jít ve vývoji poměrně daleko – upravit zdrojové kódy a zkompilovat si vlastní verzi ASP.NET Core MVC. (Freeman, 2016, s. 8)

3.2.3.4 Persistence dat v ASP.NET Core MVC

Jelikož je HTTP protokol bezestavový (ang. *stateless*), není možné pomocí něho přímo ukládat stav jednotlivých proměnných mezi HTTP požadavky. U klasické desktopové aplikace na platformě .NET stačí vytvořit proměnnou, přiřadit jí hodnotu, a tato hodnota zůstane uložena v paměti počítače do té doby, než bude přepsána jinou hodnotou, nebo uvolněna (ať už ručně, nebo automaticky pomocí *garbage collectoru*). Desktopovou aplikaci také velmi často používá v jeden okamžik jeden uživatel, a proto se perzistence dat realizuje například serializací dat do binárního souboru, nebo jiného formátu – tento přístup není vhodný ve webových aplikacích s mnoha aktivními uživateli současně, protože I/O operace jsou často pomalejší než databázové operace a je také potřeba řídit současný přístup ke sdílenému datovému zdroji. Pochopitelně toto ale nevylučuje ukládání určitých dat do

souboru na straně serveru, za předpokladu že tyto soubory nejsou příliš frekventovaně otevírány a zavírány (ať už pro zápis, nebo pro čtení). Proto je dobrou praxí ukládat často používaná data ve webových aplikacích do databáze. (Kejda, 2017, s. 25-28)

Datová perzistence mezi jednotlivými dotazy, sloužící k uchování stavu aplikace (*state management*), může být realizována v ASP.NET Core MVC různými způsoby, například:

Cookies – Cookies ukládají data napříč HTTP požadavky a posílají se s každým HTTP požadavkem, proto je potřeba jejich velikost snížit na minimum. Většina prohlížečů stanovuje maximální velikost cookies a dále je počet cookies limitován na doménu. Ačkoliv mohou být cookies ručně upraveny uživatelem a může vyprchat jejich platnost, stále se jedná o nejvíce trvalý způsob datové perzistence na straně klienta. Prakticky se cookies používají například pro identifikaci uživatele a následné personalizaci webu. Při používání cookies je potřeba mít na paměti GDPR. (*Session and app state in ASP.NET Core,* 2018)

Session State – Tento způsob datové perzistence se používá pro uložení informací během toho, když uživatel prochází jednotlivými stránkami webu. Takto uložená data mají být dle dokumentace považována za efemérní a aplikace musí fungovat i při jejich absenci. Session data jsou uložena na straně serveru a jsou přiřazena klientovi na základě *session ID*, které je uloženo v *cookies*, které se posílají s každým požadavkem. Již zmíněné *session ID* je vygenerováno pro prohlížeč, který zaslal požadavek na server, a tudíž není sdíleno mezi prohlížeči. Defaultní chování je takové, že pokud uběhne 20 minut od posledního požadavku na server, tak se session odstraní. Další podstatnou informací je, že se server nemůže nijak dozvědět o tom, že uživatel zavřel svůj prohlížeč. Proto zavření prohlížeče na session nemá vliv. Z hlediska bezpečnosti je velice nebezpečné ukládat jakékoliv citlivé údaje do session. (*Session and app state in ASP.NET Core,* 2018)

TempData – TempData je proměnná, která obsahuje uložená data do té doby, než jsou přečtena (ačkoliv existují metody pro přečtení těchto dat bez jejich smazání). TempData mohou používat poskytovatele (tzv. *provider*) buď ze Session State, nebo z Cookies. (*Session and app state in ASP.NET Core,* 2018)

QueryStrings – Další možností, jak předávat data mezi jednotlivými požadavky je použití URL, do které jsou uloženy jednotlivé hodnoty proměnných. Díky tomu je možné s jiným uživatelem sdílet URL odkaz, který v sobě obsahuje nastavení stavu požadované stránky. Jelikož si může kdokoliv tyto data z URL adresy přečíst, je bezpečnostní hrozbou využívat tento způsob pro uložení citlivých dat (hesel, osobních údajů apod.). Pomocí URL adresy s danými parametry je možné zavolat příslušnou akční metodu na serveru, a proto je možné URL adresu s parametry použít k CSRF útoku na web, není-li stránka dostatečně zabezpečená. (*Session and app state in ASP.NET Core*, 2018)

Hidden fields – Webové HTML formuláře mohou obsahovat skrytá pole, která jsou odesílána na server. V případě metody GET jsou data připojena k URL adrese a v případě metody POST jsou data připojena do těla HTTP požadavku. *(HTML <form> method Attribute*, nedatováno)

Cache – Cachovaná data nejsou spojena s konkrétním uživatelem, HTTP voláním, ani session. Slouží k ušetření výkonu, který by byl nutný ke generování obsahu, který je uložen v cache. ASP.NET Core obsahuje více implementací cachingu, například *IMemoryCache,* což jsou data uložená v paměti webového serveru. (*Cache in-memory in ASP.NET Core,* 2016)

Dependency injection – ASP.NET Core MVC bylo navrhováno od samého počátku pro podporu dependecy injection. Dependency injection je technika pro vytváření volných vazeb (*loose coupling*) mezi objekty a jejich závislostmi (tzv. *dependencies*). Jednotlivé závislosti se poté nejčastěji injektují do daných objektů pomocí konstruktorů, tzv. *constructor injection*. Díky použití dependency injection jsou potom vkládané závislosti dostupné odkudkoliv v kódu aplikace skrze konstruktor třídy – například v controlleru, nebo přímo v jednotlivých pohledech pomocí klíčového slova *inject*. V praktické části této práce bylo použito dependecy injection databázového kontextu v metodě *ConfigureServices* následovně (metoda je z redakčních důvodů zkrácena):

```
public void ConfigureServices(IServiceCollection services)
{

  services.AddDbContext<DatabaseContext>();

}
```

Výpis 3.1 – Konfigurace dependecy injection databázového kontextu

Výše uvedená metoda *ConfigureServices* se nachází v souboru *Startup.cs* a je automaticky volána běhovým prostředím. Třída *DatabaseContext* je třída databázového kontextu dědící od třídy *DbContext,* což je třída použitého frameworku Entity Framework Core. Objekt databázové kontextu je poté dostupný v každém controlleru skrze konstruktor následovně:

```
public class HomeController : Controller
{
   private readonly DatabaseContext _databaseContext;

   public HomeController(DatabaseContext dbContext)
   {
       this._databaseContext = dbContext;
   }
}
```

Výpis 3.2 – Získání databázového kontextu skrze konstruktor

Databáze – Další možností, jak uchovat stav aplikace, je pomocí databáze, což souvisí s již zmíněnou technikou dependency injection. Pochopitelně není vůbec nutné používat žádné objektově-relační mapování (např. Entity Framework) a lze se vyhnout i používání dependency injection a vrátit se k používání knihovny ADO.NET, která byla preferovaným způsobem přístupu k databázi před příchodem Entity Frameworku. Samotný Entity Framework je sám totiž postaven na této knihovně. (*Entity Framework*, nedatováno)

ASP.NET Core podporuje množství databází, skrze takzvané poskytovatele (*database providers*), ať už vlastních, nebo třetích stran. Mezi podporované databáze patří například MS SQL Server, SQLite, PostgreSQL, MySQL, SQL Server Compact, Db2, nebo Microsoft Access files. (*Database Providers, 2018*)

4 Vlastní práce

Předmětem této kapitoly je na základě poznatků z teoretické části práce navrhnout a implementovat webový redakční systém, autorem pojmenovaný jako **JasperSite**, pro tvorbu webových stránek a správu jejich obsahu. Na návrhu a implementaci redakčního systému jsem pracoval sám a není součástí žádného komerčního nebo jiného řešení.

4.1 Stanovení požadavků na aplikaci

Po zběžném studiu již existujících redakčních systémů jsem se rozhodl klást na výsledný systém konkrétní požadavky. Systém umožňuje:

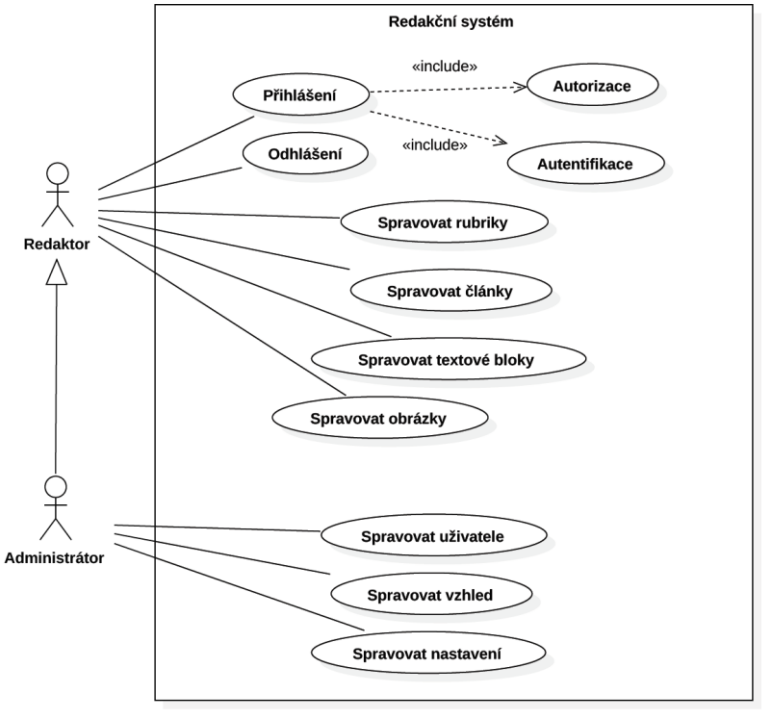
- Zobrazení webové stránky s příspěvky v daných kategoriích a zobrazení bloků textů bez příslušnosti k dané kategorii.
- Přihlášení a odhlášení správce obsahu webové stránky do administrace.
- Přidávání nových rubrik, článků, textových bloků, obrázků, tabulek a videí.
- Správu uživatelů.
- Osobě se znalostní programování vytvořit a za běhu aplikace přidat vlastní vzhled pro webovou stránku bez nutnosti rekompilace projektu. Osoba bez znalosti programování může již existující vzhledy přidávat nebo mazat.
- Provést instalaci redakčního systému pomocí instalačního formuláře, s možností volby více druhů databází.

4.2 Návrh redakčního systému

Před zahájením návrhu aplikace bylo potřeba se alespoň povrchně seznámit s některými již etablovanými existujícími webovými redakčními systémy. K tomuto účelu jsem využil svých uživatelských znalostí systémů Wordpress a Prestashop, u kterých jsem se více či méně inspiroval. Pro návrh základní funkčnosti systému byl použit *Use case diagram*, pro datové modelování jsem využil standardní UML C*lass Diagram* a poté variantu UML diagramu od Scotta W. Amblera pro modelování schématu relační databáze. Návrhu grafického uživatelského rozhraní a interakčnímu designu se tato práce nevěnuje, stejně tak jako není věnována pozornost přístupnosti a SEO optimalizaci. Kvůli většímu rozsahu celého redakčního systému jsou v následujících diagramech vypuštěny některé méně důležité součásti systému.

4.2.1 Use case diagram

Use case diagram (diagram případů užití) je prvním z formálních prostředků softwarového inženýrství, který jsem zvolil pro návrh aplikace, jelikož slouží jako velmi hrubý nástroj pro popis struktury funkčnosti systému a poskytne obecnou představu o tom, jaké skupiny metod bude aplikace obsahovat a kdo bude se systémem interagovat. (Vrana, 2008, s. 59)



Obrázek 4.1 – Use case diagram

Na levé straně diagramu jsou uvedeni aktoři komunikující se systémem. Mezi uvedenými aktory se nachází Administrátor a Redaktor. Na implementační úrovni jednotlivě pojmenovaní aktoři nebudou muset nezbytně nutně odpovídat stejnojmenným třídám, neboť jak uvádí specifikace UML: *„Note an actor does not necessarily represent a specific physical entity but merely a particular facet (i.e., "role") of some entity"* (Unified Modelling Language, 2007, s. 586, 587). Tedy jednotliví aktoři budou realizováni pomocí rolí, kterých bude moci třída Uživatel (*User*) nabýt.

4.2.2 Datový slovník tříd objektů

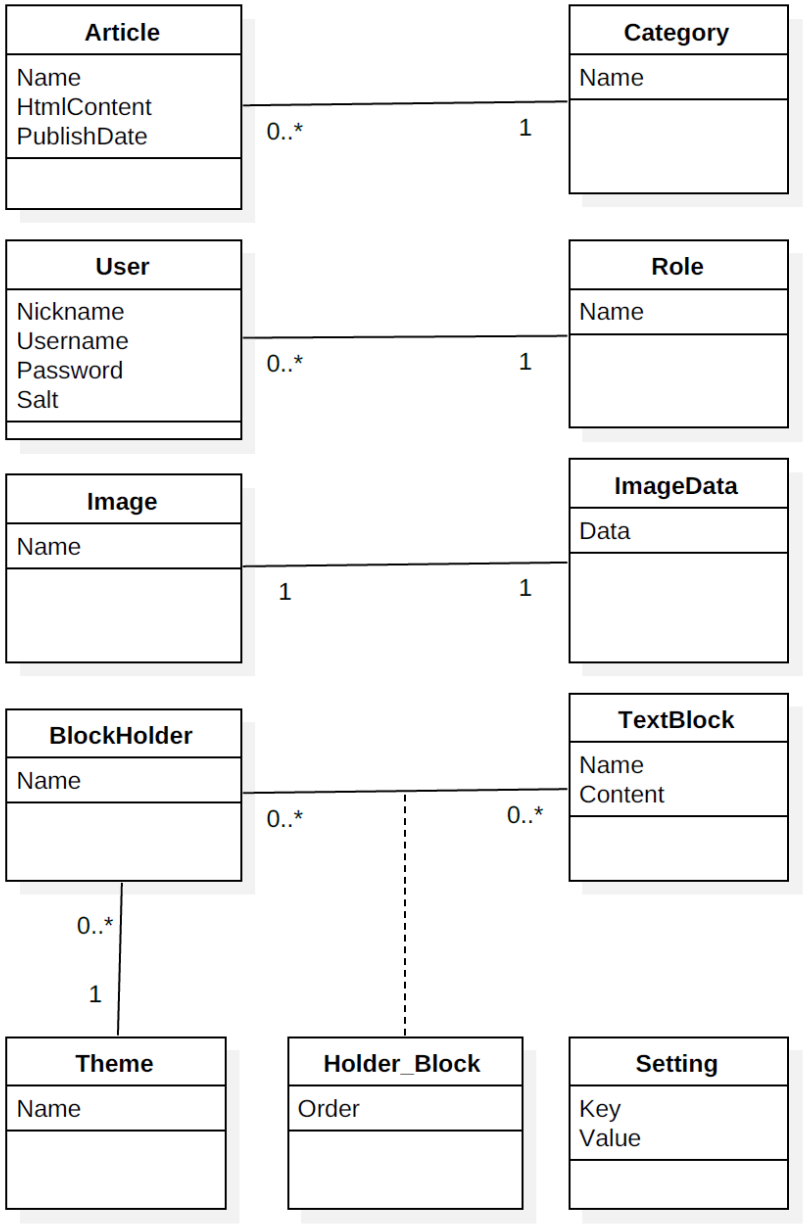
Následující tabulka stručně popisuje význam jednotlivých tříd objektů vyskytujících se v následujících UML diagramech. Vrana (2008, s. 85) tuto tabulku nazývá tzv. datovým slovníkem:

Název třídy	Popis
Article	Jedná se o publikovaný **článek**, který může obsahovat text, obrázky, tabulky, video apod.
Category	Jedná se o **rubriku**, pod kterou články spadají.
User	Jedná se o **uživatele** systému, tedy například administrátora nebo redaktora.
Role	**Role** udává, jaká oprávnění bude mít její nabyvatel.
Image	**Obrázek** je seznam pojmenovaných obrázků a odkazů na jejich binární data.
ImageData	**Data obrázku** je samostatně vyčleněná třída obsahující samotná binární data obrázku. Tato data jsou v samostatném objektu, aby bylo možné využít techniky *lazy loadingu*, kdy jsou obrázky skutečně načteny až když jsou zapotřebí.
BlockHolder	**Kontejner na bloky** je třída seskupující textové bloky do určité oblasti na stránce.
TextBlock	**Textový blok** je třída sloužící k prezentaci nejen textového obsahu mimo pevně dané kategorie. Může se jednat o uvítací text stránky, nebo i zápatí webu.
Theme	**Vzhled** je objekt představující vizuální reprezentaci webové stránky, která je nezávislá na obsahové stránce.
Holder_Block	Tato třída je dle standardu UML tzv. propojovací třídou, což je částečně vazba a částečně třída. Umožňuje realizovat vazbu M:N mezi třídami TextBlock a BlockHolder.
Setting	Třída **nastavení** uchovává některá konfigurační data redakčního systému, například název webové stránky.

Tabulka 4.1 – Datový slovník tříd objektů

4.2.3 Doménový třídní diagram

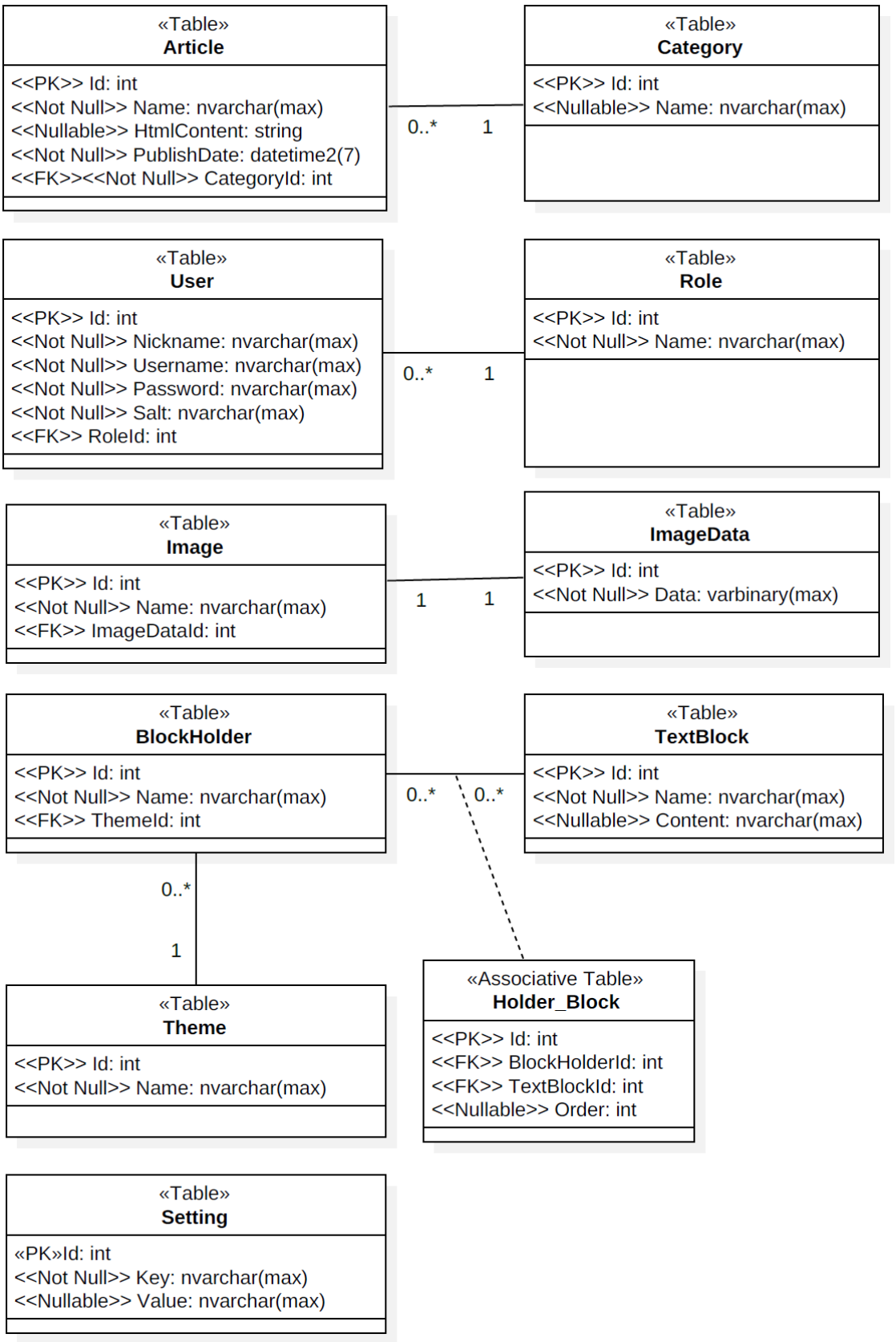
Následující schéma představuje doménový diagram tříd (*Domain class diagram*) (Vrana, 2008, s. 79), na který budou namapovány databázové tabulky technikou ORM (*Object-relational mapping*).



Obrázek 4.2 – Domain class diagram

Jednotlivé uvedené třídy budou tedy odpovídat stejnojmenným relačním tabulkám (pojmenovaných dle konvence v množném čísle) a jednotlivé atributy odpovídají stejnojmenným sloupcům v těchto tabulkách.

4.2.4 UML physical data model



Obrázek 4.3 – UML physical data model

Na předchozím diagramu se nachází tzv. *UML Physical data model,* dle Scotta Amblera (kanadský softwarový inženýr a autor mnoha publikací o jazyku UML). Jak píše Scott W. Ambler (2012), jazyk UML nemá žádný zvláštní standardizovaný diagram pro popis interního schématu databáze, jednotlivých tabulek, jejich a tributů a vzájemných vazeb. Ambler dále píše, že pro popis databázového schématu si nelze vystačit pouze s nástroji třídního UML diagramu, neboť ten neposkytuje standardizovaný diagram pro zachycení databázových spouští, pohledů, indexů, složených primárních nebo kandidátních klíčů a podobně. Více informací k této problematice Ambler podává i ve své prezentaci (*Towards a UML Persistence Model, 2000)*. Alternativně pro účel databázového modelování se dá použít Chenův ER diagram, který ale není součástí standardu UML a používá značení vizuálně odlišné od jazyka UML.

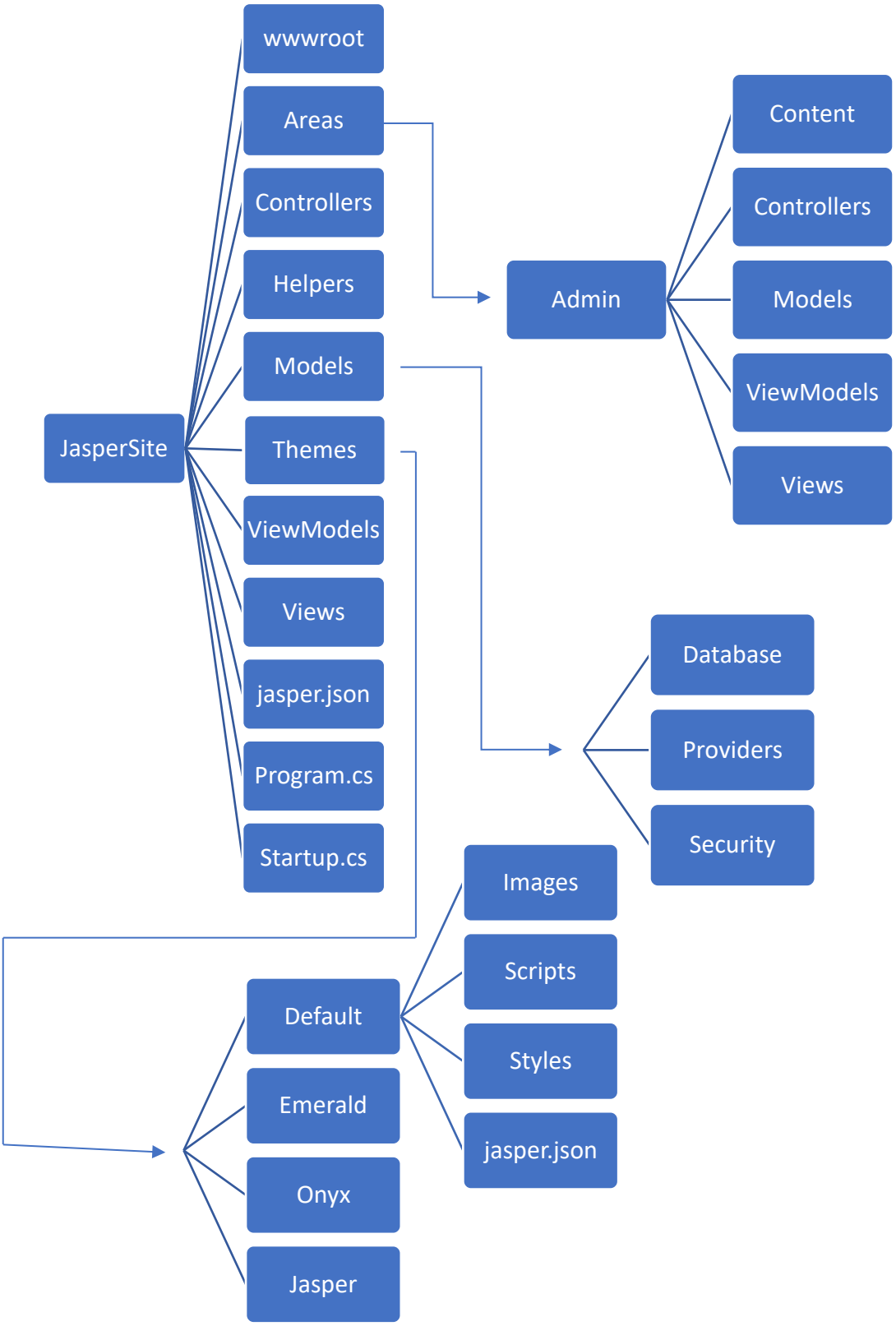
4.2.5 Zajištění perzistence dat a optimalizace dotazů

Dva výše uvedené diagramy popisovaly uložení dat na dvou úrovních. *UML Physical data model* popsal uložení databázových entit na úrovni relační databáze, zatímco *Class diagram* popsal již překlopená data z databáze do objektů (což bude v projektu prakticky realizováno automaticky pomocí techniky ORM).

Z výše uvedeného je zřejmé, že data nacházející se ve třídách za běhu programu budou mít transientní povahu až do okamžiku, než bude proveden příslušný SQL příkaz pro jejich uložení do databáze. Samotný SQL kód pro vytvoření databáze bude pomocí příslušného frameworku vygenerován automaticky z uvedeného *Class diagramu* (tzv. *code-first* přístup) a proto není předmětem této kapitoly sestavení a optimalizace kódu SQL. O již zmíněnou optimalizaci automaticky vygenerovaných SQL dotazů na databázi se totiž na platformě .NET Core stará běhové prostředí samo, pokud využijeme k dotazování jazyk LINQ.

Některá konfigurační data potřebná pro instalaci systému je nutné mít k dispozici ještě před vytvořením samotné databáze. K tomuto účelu bude program využívat jednoduché ukládání dat do souboru ve formátu JSON, přímo v adresáři aplikace, přičemž přístup k těmto souborům bude mít pouze aplikace samotná a oprávněný uživatel.

4.2.6 Návrh adresářové struktury projektu



Obrázek 4.4 – Adresářová struktura projektu

Na předchozím schématu je možné vidět zjednodušenou adresářovou strukturu projektu, místy s uvedenými podstatnými soubory, o kterých se bude hovořit dále. Velká část této adresářové struktury je vynucená použitým frameworkem a považuje se za *best practice* jí takto používat. Ze schématu byl pro zjednodušení vynechán projekt *JasperSite.Test*, který obsahuje jednotkové testy (unit testy).

Co se týče adresářové struktury i samotné funkčnosti aplikace, redakční systém je možné rozdělit na dvě podstatné části. Jedná se o administrační sekci, kde uživatel spravuje svojí webovou stránku (složka Areas/Admin) a poté se jedná o samotnou webovou stránku, kterou vidí návštěvník (vzhled této stránky je uložen ve složce Themes).

Jak je po bližším prozkoumání zřetelné, v kořenové složce projektu se nachází soubor *jasper.json*, stejně tak jako poté v každé složce se vzhledy (složky Jasper, Onyx, Emerald a Default). První zmíněný soubor, nacházející se v kořenovém adresáři, je konfigurační soubor celého redakčního systému. Druhý zmíněný soubor ve složce vzhledů figuruje jako konfigurační soubor pro každý jednotlivý vzhled.

Návrhem adresářové struktury projektu bylo nutné se začít zabývat již před samotnou implementací, neboť jedním z požadavků na výsledný program bylo, aby uživatel mohl za běhu aplikace dynamicky přidávat vlastní vzhledy. Jelikož použitý framework kompiluje výsledný program do několika knihoven (*.dll, dynamic linking library*), bylo potřeba předem navrhnout, které soubory, popřípadě adresáře se nebudou kompilovat a budou vyčleněny mimo *.dll* knihovnu, aby je bylo možné za běhu aplikace upravovat, bez nutnosti rekompilace celého projektu. Toto nastavení se nachází v souboru *startup.cs* viditelném na předchozím diagramu.

4.3 Implementace redakčního systému

Jak již bylo uvedeno, serverová část CMS byla implementována pomocí technologie ASP.NET Core MVC a programovacího jazyka C#. Následující kapitoly se krátce věnují nástrojům, technologiím a programovacím či skriptovacím jazykům použitým při implementaci tohoto projektu.

4.3.1.1 Nástroje použité při vývoji

Při vývoji redakčního systému jsem mimo jiných používal i následující nástroje:

Název	**Popis**
Microsoft Visual Studio 2017	Integrované vývojové prostředí (IDE) nejenom pro tvorbu webových aplikací na platformě Windows.
Visual Studio Code[2]	Alternativa k Visual Studiu – jedná se o jednodušší[3] a multiplatformní nástroj pro vývojáře.
Microsoft SQL Server Management Studio 2017 a přidružené nástroje.	Nástroj pro správu MS SQL databáze.
MySQL Workbench 8.0 CE	Nástroj pro správu MySQL databáze.
StarUML	Tento nástroj byl použit pro prototypování databáze a tvorbu use case diagramu.
GIT	Systém pro správu verzí.
Servery IIS a Kestrel.	Tyto dva servery se používají zároveň, kdy IIS Server (popřípadě jeho odlehčená verze Express) slouží jako reverse proxy server[4] pro Kestrel.

Tabulka 4.2 – Nástroje použité při vývoji

4.3.1.2 Client-side frameworky a balíčky

Pro tvorbu grafického uživatelského rozhraní jsem v projektu použil následující prostředky (ale i mnohé další), dostupné pomocí balíčkovacího nástroje NPM:

Název	**Popis**
Bootstrap	CSS Framework pro tvorbu grafického rozhraní.
JQuery	JS Framework.
Chart.js	JS knihovna pro tvorbu grafů.
FontAwesome	CSS knihovna pro přidávání vektorové grafiky.
Prism	JS knihovna pro zvýrazňování syntaxe programovacích jazyků.
Tinymce	Knihovna pro bohatý textový editor s možností přidávat nejenom text, ale i obrázky, videa apod.

Tabulka 4.3 – Client-side technologie

[2] Od 1.11. 2017 společnost Microsoft ukončila podporu alternativního nástroje WebMatrix.
[3] Pojmem jednodušší je zde myšlena odlehčená velikost instalačního souboru, méně možností samotného IDE a nižší nároky na hardware počítače.
[4] Alternativně by šlo jako reverse proxy použít Nginx nebo Apache server.

4.3.1.3 Server Side technologie a NuGet balíčky

Název	**Popis**
Microsoft ASP.NET Core MVC	Multiplatformní framework pro tvorbu webových aplikací od společnosti Microsoft.
Microsoft.AspNetCore.All	Jedná se o NuGet balíček všech potřebných knihoven pro vývoj aplikací svázaný s výše uvedeným ASP.NET fremeworkem. Obsahuje například Entity Framework Core, který se stará o objektově-relační mapování.
Microsoft.NETCore.App	NuGet balíček obsahující základní knihovny fremeworku .NET Core. Zahrnuje například jazyk LINQ.
Newtonsoft.Json	NuGet balíček pro práci s *.json* soubory.
Pomelo.EntityFrameworkCore.MySql[5]	NuGet balíček umožňující připojení k MySQL databázi.
Moq	Framework pro jednotkové testy databázového kontextu.
NUnit	Framework pro jednotkové testy.

Tabulka 4.4 – Server-side technologie

4.3.1.4 Implementace datové perzistence

Samotná aplikace používá dva způsoby ukládání dat. Většina dat jako jsou například články, rubriky, uživatelské účty a jejich hesla apod. jsou uloženy v relační databázi. Aplikace momentálně podporuje dvě databáze, a to MySQL a MSSQL. Některá ostatní data, jako například připojovací řetězec k databázi, nebo název aktuálního vzhledu se nachází v konfiguračním souboru *jasper.json* v kořenovém adresáři. Nastavení týkající se vzhledů, tedy například názvy jednotlivých kontejnerů pro textová pole, nebo mapování URL adres na fyzické soubory se nachází v souboru *jasper.json* v příslušné složce daného vzhledu.

Pro vytvoření samotné databáze byla využita technika objektově relačního mapování (ORM) a technika *code first design*, kdy byl nejprve napsán kód daných tříd a poté na jeho základě byla automaticky vygenerována databázová struktura odpovídajících relací. V **příloze A** je možné si prohlédnout kód těchto jednotlivých tříd.

[5] Konkurenční balíček MySql.Data.EntityFrameworkCore v době psaní této práce trpěl neduhem neschopnosti správně namapovat datový typ byte[] na odpovídající datový typ v MySql databázi. Problém byl adresován například zde: https://stackoverflow.com/questions/41234762/how-to-store-blob-type-in-mysql-with-entity-framework-core-using-byte

K jednotlivým třídám, které jsou již namapované na databázové tabulky, se přistupuje pomocí speciální třídy, která dědí od třídy *DbContext* (zjednodušeno):

```
public class DatabaseContext: DbContext,IDatabaseContext
{

  protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
  {
     string typeOfDatabase = Configuration.GlobalWebsiteConfig.TypeOfDatabase;
     switch (typeOfDatabase)
            {
case "mssql":
optionsBuilder.UseSqlServer(Configuration.GlobalWebsiteConfig.ConnectionString);
break;

case "mysql":
optionsBuilder.UseMySql(Configuration.GlobalWebsiteConfig.ConnectionString); break;
default: throw new NotSupportedDatabaseException();
            }
    }
 }
```

Výpis 4.1 – Třída databázového kontextu

Výše uvedená třída *DatabaseContext* dále implementuje rozhraní *IDatabaseContext* kvůli možnosti jejího jednotkového testování pomocí frameqorku *Mock* a *Nunit*. Třída dále obsahuje rozhodovací konstrukci *switch*, která rozhoduje, jestli bude použit poskytovatel pro připojení k MSSQL databázi, nebo k databázi MySQL.

Dalším aspektem datové perzistence, která se prolíná celým projektem, je použití techniky *dependency injection.* V praxi to znamená, že databázový kontext není sdílen mezi HTTP požadavky například pomocí statické třídy, ale je předáván jako nová instance při každém požadavku a toto předávání je realizováno pomocí konstruktoru příslušného *controlleru*, který požadavek obsluhuje.

Kromě databáze jsou některá data uložena v již zmíněných konfiguračních souborech s názvem *jasper.json*. Jedná se především o data, která je potřeba znát ještě před samotným automatickým vytvořením databáze. Následující výpis znázorňuje obsah souboru *jasper.json* v kořenovém adresáři projektu:

```
{
  "themeName": "Default",
  "connectionString": "Server=localhost;Database=jaspermysql;Uid=root;Pwd=root",
  "typeOfDatabase": "mysql",
  "installationCompleted": "True"
}
```

Výpis 4.2 – Obsah souboru jasper.json

Tento konfigurační soubor obsahuje informace o názvu aktuálního vzhledu, připojovacím řetězci k databázi, typu databáze a informace o tom, jestli byla instalace redakčního systému dokončena. Data z tohoto souboru se pochopitelně nečtou při každém požadavku na webovou stránku, ale jsou uložena do paměti serveru a v případě potřeby jsou opět znovu načtena.

4.3.1.5 Vlastní routing systém a prezentace obsahu pomocí vzhledů

Framework ASP.NET Core MVC obsahuje vestavený routing systém, který mapuje jednotlivé fyzické soubory na příslušné URL adresy v závislosti na jménu controlleru a akční metody (*action method*). Jelikož jedním z požadavků na systém byla možnost tvorby vzhledů ostatními uživateli, bylo nutné vyřešit problém toho, jak umožnit změnu části aplikace přímo za běhu. Jelikož použitý framework defaultně kompiluje výslednou aplikaci do soustavy *.dll* souborů, které jsou poté spuštěny běhovým prostředím na serveru, nebylo nejprve možné nijak do již zkompilovaného projektu zasahovat. Zatím jako jediné řešení tohoto problému se mi podařilo nalézt možnost vypnout kompilaci jednotlivých pohledů (*views*) pomocí úpravy souboru projektu *JasperSite.sln*:

```
<PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
    <MvcRazorCompileOnPublish>false</MvcRazorCompileOnPublish>
</PropertyGroup>
```

Výpis 4.3 – Zakázání kompilace pohledů

Ačkoliv tímto nestandardním nastavením aplikace již neušetří výpočetní čas předkompilováním pohledů, je dosaženo požadovaného efektu – je skutečně možné během běhu aplikace měnit jednotlivé vzhledy stránek, tedy jinými slovy editovat příslušné *.cshtml* soubory pohledů. Jelikož projekt v sobě již obsahuje ukázkové vzhledy, bylo potřeba nastavit jejich překopírování do výstupní složky (taktéž editací *.sln* souboru):

```
<ItemGroup>
        <None Include="Themes/**" CopyToPublishDirectory="Always" />
</ItemGroup>
```

Výpis 4.4 – Překopírovaní vzhledů do výstupní složky

Další novinkou oproti starší verzi ASP.NET MVC, je příchod takzvaných *middleware* komponent, pomocí nichž je možné nakonfigurovat chování HTTP požadavků ještě před tím, než je obslouží příslušný controller. ASP.NET Core MVC defaultně neumožňuje obsluhovat požadavky na statické soubory, čímž jsou myšleny soubory fyzické, které server vrátí na požadavek bez účasti controlleru. Je nutné proto zapnout tuto funkci v souboru *Startup.cs* v metodě *Configure* pomocí:

```
app.UseStaticFiles();
```

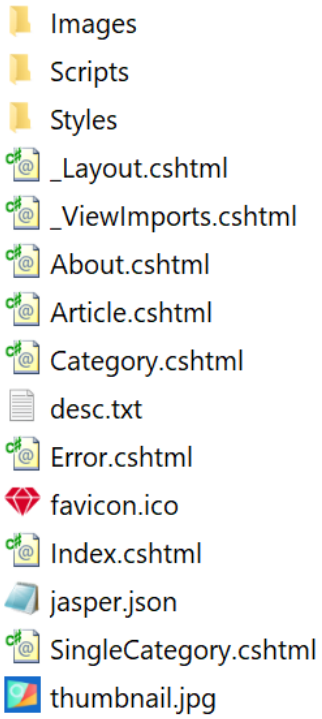
Výpis 4.5 – Povolení statických souborů v adresáři wwwroot

Toto povolení se ale defaultně týká pouze adresáře *wwwroot* a statické soubory, jako například obrázky, css a javascriptové soubory by mohl server vracet pouze z této složky. Toto je nedostačující pro požadovaný systém vzhledů. Je proto nutné také povolit přístup přes HTTP volání ke statickým souborům ve složce se vzhledy, čehož se dosáhne zavedením následujícího *middleware*:

```
app.UseStaticFiles(new StaticFileOptions()
{
FileProvider = new PhysicalFileProvider(
Path.Combine(Directory.GetCurrentDirectory(),Configuration.ThemeFolder)),
            RequestPath = new PathString("/" + Configuration.ThemeFolder)
});
```

Výpis 4.6 – Povolení statických souborů ve složce se vzhledy

Výše uvedený *middleware* mapuje fyzickou cestu ke složce s vzhledy na příslušnou URL adresu, ze které budou statické soubory přes HTTP volání dostupné. Po zkompilování projektu vypadá poté struktura složky libovolného vzhledu podobně jak uvádí následující obrázek:



Obrázek 4.5 – Složka vzhledu

Složka vzhledu obsahuje podsložku *Images*, která obsahuje obrázky použité na stránce, složka *Scripts* obsahuje CSS a javascriptové soubory. Soubor *_Layout.cshtml* je hlavní stránkou, do které se poté načítá obsah jednotlivých podstránek, například obsah pohledu *Index.cshtml*. Soubor *_ViewImports.cshtml* obsahuje kód, který je sdílený mezi všemi pohledy (tedy jinak řečeno mezi všemi *.cshtml* soubory v této složce). Pohledy *About.cshtml*, *Article.cshtml*, *Category.cshtml* a *SingleCategory.cshtml* jsou volitelné stránky vzhledu, které se autor vzhledu rozhodl implementovat. Soubor *desc.txt* obsahuje popis vzhledu a informace o autorovi, které se ukazují v redakčním systému a *favicon.ico* je ikona zobrazovaná v liště webových prohlížečů. Obrázek *thumbnail.jpg* je zobrazen v redakčním systému pro snazší rozpoznání vzhledu uživatelem.

Jedním z nejdůležitějších a zatím nepopsaných souborů je soubor *jasper.json*, který zde již není konfiguračním souborem pro celý web, ale vztahuje se jen k aktuálnímu vzhledu. Používání takovýchto konfiguračních souborů specifických pro danou složku není žádnou novinkou, inspiroval jsem se například u technologie ASP.NET, což je technologie dvě generace zpátky za aktuálně použitým frameworkem. Obsah souboru *jasper.json* vypadá následovně:

```
{
  "urlRewriting": "True",
  "articleFile": "Article.cshtml",
  "articleRoute": "/home/article/",
  "blockHolders": [ "MainPageHolder", "FooterHolder", "AboutPageHolder" ],
  "missingImagePath":  "./Images/missing.jpg",
  "routingList": {
    "homePage": [
      "/home/index",
      "/home",
      "/",
    ],
    "homePageFile": "Index.cshtml",
    "errorPageFile": "Error.cshtml"
  },
```

```
  "customPageMapping": [
    {
      "routes": [
        "/home/category",
        "/category/",
      ],
      "file": "Category.cshtml"
    },
    {
      "routes": [
        "/home/tutorial",
      ],
      "file": "Tutorial.cshtml"
    },
    {
      "routes": [
        "/home/article",
      ],
      "file": "Article.cshtml"
    },
    {
      "routes": [
        "/home/singlecategory",
      ],
      "file": "SingleCategory.cshtml"
    },
    {
      "routes": [
        "/home/about",
      ],
      "file": "About.cshtml"
    }
  ]
}
```

Výpis 4.7 – Konfigurační soubor vzhledu jasper.json

Uvedený soubor si zajisté zaslouží podrobnější komentář. Atributy *urlRewriting*, *articleFile* a *articleRoute* jsou nepovinné a týkají se automatického přepisování URL adres článků na adresy snáze zapamatovatelné. Dále každý vzhled sestává z určitých pojmenovaných kontejnerů (v kódu označovány jako tzv. *holders*), do kterých je možné načítat textové bloky (v kódu často označováno jako *text blocks*). Tyto textové bloky pochopitelně mohou obsahovat i další prvky (obrázky, tabulky, videa apod.). Jak si již čtenář mohl všimnout, vzhled sám o sobě neobsahuje žádné controllery, pouze pohledy. Řešeným problémem je tedy situace, že vzhled potřebuje notifikovat redakční systém o tom, jaké kontejnery v sobě zahrnuje. Ačkoliv pohledy mohou obsahovat kód jazyka C# a bylo by možné systém notifikovat skrze něj, zvolil jsem řešení skrze konfigurační soubor, který umožňuje předat podstatné informace redakčnímu systému. Na pátém řádku na výpisu 4.7 je tedy seznam pojmenovaných kontejnerů, které vzhled deklaruje, že je bude používat. Zde se jedná o *MainPageHolder*, *FooterHolder* a nakonec *AboutPageHolder*. Jak z názvů těchto kontejneru vyplývá, budou sloužit pro informace na hlavní stránce, v patičce stránky a na

stránce „O Autoru“. Tyto kontejnery se poté do HTML kódu začlení pomocí volání například takto:

```
@J.Components.Holder("MainPageHolder")
```

Výpis 4.8 – Vložení kontejneru do stránky vzhledu

Kde „*J*“ je název třídy obsahující datovou složku „*Components*“ pro výpis přidružených textových bloků pomocí metody „*Holder*“. Argumentem je název kontejneru, který byl deklarován dříve v konfiguračním souboru *jasper.json*. Instance třídy „*J*“ je dostupná v každém pohledu díky souboru *_ViewImports.cshtml*, ve kterém je provedena následující injektáž:

```
@inject IJasperDataService J;
```

Výpis 4.9 – Injekce pomocného objektu

Pokud se nyní vrátíme zpět ke konfiguračnímu souboru *jasper.json,* tak následujícím konfiguračním nastavením je *missingImagePath*, které udává relativně[6] cestu k obrázku, který se načte jako zástupný symbol při chybějícím obrázku v databázi.

Dalším nastavením je *routingList*. Jedná se objekt, který v sobě obsahuje:

- Pole *homePage:* Jedná se o pole řetězců udávající, na jakou URL adresu bude namapována hlavní stránka.
- Řetězec *homePageFile:* Tento řetězec udává relativní umístění souboru domovské stránky, typicky *Index.cshtml*.
- Řetězec *errorPageFile* udává relativní umístění souboru chybové stránky, typicky pro tento redakční systém soubor *Error.cshtml*.

[6] Cesty je skutečně možné zadávat relativně ke složce vzhledu, jelikož redakční systém sám automaticky předřadí absolutní cestu k danému adresáři.

Nakonec se dostáváme k jádru fungování celého systému vzhledů, protože poslední nastavení *customPageMapping* je pole objektů, které se stará o mapování fyzických souborů na URL adresy a díky tomu může autor vzhledu upravovat, jak budou vypadat jednotlivé URL adresy daného vzhledu. Je také pochopitelně možné na jeden fyzický soubor nasměrovat více URL adres. Daný konfigurační objekt z pole *customPageMapping* obsahuje vlastnost *routes* a vlastnost *file*. *Routes* představuje pole řetězců jednotlivých URL adres, které jsou namapovány na relativně adresovaný fyzický soubor vlastnosti *file*. Tento systém mapování URL adres funguje především díky akční metodě *Index* controlleru *Home*, neboť ten zachytí všechny HTTP požadavky (které nejdou do administrace webu) a provede dané mapování na základě aktivního vzhledu. Fungování této akční metody je značně komplexní a zahrnuje rozhodování o tom, jaká konkrétní stránka bude zobrazena. Dále tato metoda řeší zobrazování chybových stránek, které se liší pro přihlášeného a pro nepřihlášeného uživatele.

Na konci této kapitoly se budu věnovat vysvětlení vztahu článků a textových bloků. Jak již bylo vysvětleno, textové bloky jsou nezávislé na rubrice a jsou agregovány v příslušných kontejnerech, které mohou být kdekoliv na stránce. Nejčastěji budou vystupovat jako informace v hlavičce stránky, zápatí stránky, může se jednat o uvítací text, nebo i reklamní sdělení. Díky tomu, že jeden a ten samý kontejner se může opakovat na celém webu, i na jedné stránce vícekrát, mohou textové bloky sloužit i k zobrazování stejných informací na více místech a není nutné proto textové bloky duplikovat.

Pokud ale hovoříme o článcích, tak ty jsou charakteristické tím, že spadají právě vždy do jedné rubriky. Stránka poté musí obsahovat seznam rubrik a odkazy na jednotlivé články, které se zobrazují vždy na samostatné stránce, a nikoliv jako součást kontejneru. Adresa pro zobrazení článku poté vypadá například následovně: *http://mujweb.cz/home/article?id=1*. Z adresy je zřejmé, že k nalezení článku je potřeba parametr *id* na konci URL adresy. Toto řešení není nijak estetické a lze použít postupy pro používání hezčích adres článků, například *http://mujweb.cz/clanky/webova_grafika_uvod*. Tato funkcionalita je v redakčním systému zahrnuta také. Je nutné je ale aktivovat pomocí již zmíněných atributů *urlRewriting, articleFile* a *articleRoute* v konfiguračním souboru *jasper.json*. V sekci úpravy článku se poté odemkne možnost zadat vlastní URL adresu pro libovolný článek.

Co se týká implementačních podrobností výpisu obsahu jednotlivých článků, byl k tomu použit nový nástroj frameworku ASP.NET Core MVC, a to vlastní *Tag Helpers*[7]. Jedná se o možnost používat vlastnoručně navržené a pojmenované značky, zapisované podobně jako jakékoliv jiné platné HTML značky. Použití vlastních značek ukazuje následující výpis:

```
@model int
    <j-article id="@Model">
        <div class="jumbotron">
            <h1> <j-name /></h1>
            <small>Publikováno: <j-date format="dd/MM/yyyy" /></small>
        </div>
        <j-content />
    </j-article>
```

Výpis 4.10 – Použití vlastních Tag helpers

Nejprve je modelu pohledu předáno identifikační číslo požadovaného článku, které je poté předáno jako hodnota atributu *id* značky <*j-article*>. Speciální tagy jsou na obrázku znázorněny fialovou barvou. Jedná se o:

- <j-article id="@Model"> – Tento tag pochopitelně není standartní HTML značkou a má význam pouze pro běhové prostředí ASP.NET Core MVC a to pouze v tomto projektu, kde byla tato značka implementována. Tag *j-article* má dále atribut *id*, který slouží pro specifikaci daného článku, který bude vykreslen tam, kde se nachází tato značka.
- <j-name /> – Tento tag je specifikován tak, že má význam pouze jako přímý potomek tagu <j-article>. Dále se jedná o samouzavíratelný tag, a je bez atributů. Místo tohoto tagu poté běhové prostředí dosadí název článku.
- <j-date /> – Tento tag je podobně jako předchozí nahrazen datem publikace článku.
- <j-content /> – Tato značka signalizuje, kde se má vypsat obsah článku.

Jak je vidět z předchozího výpisu kódu, díky použití těchto vlastních značek máme možnost jednoduše měnit rozložení toho, kde se bude vypisovat název, datum a obsah článku. Dále je možné tyto vlastní tagy uzavírat do standardních HTML tagů, například <h1> apod. s příslušnými výslednými efekty.

[7] Více informací například zde: https://docs.microsoft.com/cs-cz/aspnet/core/mvc/views/tag-helpers/authoring?view=aspnetcore-2.1

4.4 Bezpečnost, dokumentace a unit testy

Co se týká bezpečnosti, bylo nutné uvažovat několik aspektů:

- **Přístupnost konfiguračních souborů** – pomocí odpovídajícího *middleware* bylo zajištěno, aby webový server nikdy neodpověděl na požadavek na získání konfiguračního souboru *jasper.json* přes HTTP volání.
- **Přístupnost statických souborů** – opět pomocí příslušného *middleware* byly zpřístupněny statické soubory ve standardním adresáři *wwwroot* a poté také dodatečně ve složce se vzhledy.
- **Vstup do administrace** – vstup do administrační sekce webu je řešen pomocí přihlášení uživatele pomocí uživatelského jména a hesla, které je uloženo v databázi v zahashované podobě a pro zvýšení bezpečnosti i doplněno saltem.
- **Uživatelské role** – systém momentálně podporuje dvě uživatelské role, a to role administrátora a redaktora. Administrátor má plná práva, redaktor nesmí provádět změny vzhledů, konfigurační nastavení a nastavení uživatelských rolí a hesel.

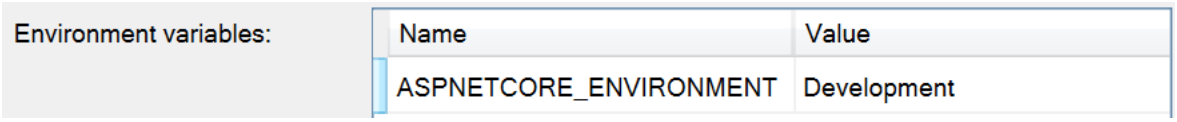
Dokumentace

K celkové bezpečnosti systému přispívá také její dokumentace a co největší pokrytí kódu unit testy. Dokumentace byla psána společně s kódem pomocí XML anotací a je možné jí vyexportovat a následně pomocí nástrojů, jako je například program *Sandcastle*[8] ji přetvořit v online webovou dokumentaci.

Analýza zranitelností

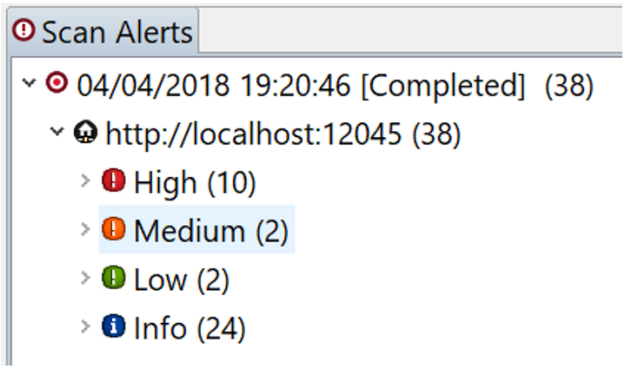
Během mého studia na České zemědělské univerzitě jsem také v rámci předmětu Bezpečnost IS provedl analýzu zranitelností tohoto redakčního systému. Při testování zranitelností jsem použil určitá zjednodušení: Testovaný web byl v režimu zapnutého debuggingu, a proto zobrazoval ladící informace místo chybových hlášek určených pro uživatele. Na platformě ASP.NET Core MVC je podstatná proměnná ASPNETCORE_ENVIRONMENT, která udává, jestli se aplikace má chovat podle nastavení pro běh na lokálním, nebo produkčním prostředí, což má bezpečnostní dopady.

[8] Ke stažení například zde: https://www.microsoft.com/en-us/download/details.aspx?id=10526



Obrázek 4.6 – Proměnná ASPNETCORE_ENVIRONMENT ve Visual Studiu

Testovaný redakční systém je provozován na lokálním vývojářském serveru IIS Express, který nově s příchodem ASP.NET Core předává požadavky na webový server Kestrel. Jako databáze je použita vývojářská databáze SQL Server Express LocalDB. Více o této databázi a o uložení dat je možné se dočíst v mé již obhájené bakalářské práci*: Prostředky platformy .NET pro zpracování dat a zajištění datové perzistence*. Použitá verze operačního systému je Windows 10. K lokálnímu testu byl použit open-source nástroj Vega od společnosti Subgraph[9]. Výstup daného testu vypadá následovně:



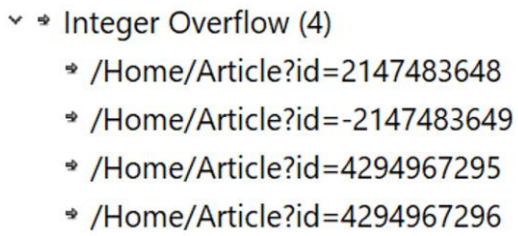
Obrázek 4.7 – Výstup testu programu Vega

Z důvodu omezeného rozsahu této práce se zde jen okrajově zmíním dle mého názoru nejzávažnější nalezené zranitelnosti. Zdrojem vysvětlení dané chyby je použitý program Vega:

- **Cleartext Password over HTTP**: Týká se adres: /Admin/Login/ a /Admin/Login (bez lomítka na konci). Zadané heslo by se nemělo posílat skrze nezabezpečený HTTP protokol v podobě čistého textu, protože by heslo mohlo být po síti odposlechnuto. **Řešení:** Po nasazení webu na produkční server nastavit přihlašovací stránce použití HTTPS protokolu, popřípadě nastavit HTTPS na localhostu pro lokání testování pomocí *self-signed* certifikátu.

[9] Více informací zde: https://subgraph.com/vega/

- **Integer overflow**: Následující obrázek uvádí problémové URL adresy:



Obrázek 4.8 – Problémové URL adresy

Nastalé přetečení datového typu INT použitého pro reprezentaci identifikátoru konkrétního článku v databázi může mít různé dopady. Při překročení maximální kladné velikosti čísla doje k přetečení do nejmenšího možného čísla a databáze může vrátit jiný článek, než bylo zamýšleno, dále může dojít k neočekávanému vyhodnocení kontrolních podmínek v kódu aplikace, nebo daný článek nebude existovat. Jedním z řešení je omezit maximální velikost parametru ID rovnou v routing systému architektury MVC, která do systému nepředá požadavky s větším než stanoveným číslem. Dalším možným řešením je v aplikaci použít klíčové slovo jazyka C#: *check* – jedná se o konstrukci signalizující, zdali došlo k přetečení zásobníku či nikoliv.

- **Page Fingerprint Differential Detected, Possible Local File Include** – Jedná se o chybu, kdy program Vega detekoval citlivé informace o aplikaci, které byly zobrazeny na chybové stránce, protože debugging nevyl vypnut. Řešením je skutečně debugging vypnout.

- **Possible Social Security Number Detected** – V udávaném souboru obsahujícím CSS styly se žádné rodné číslo nenachází a jedná se jen o číslo splňující testovací regulární výraz programu Vega.

Jednotkové testy

Co se týče jednotkových testů, rozhodl jsem se pro potřeby této práce jen přibližně nastínit provedení testu nad databázovým kontextem na příkladu jedné metody. V praxi by pochopitelně mělo smysl psát testy tak, aby došlo co k největšímu pokrytí kódu. K provedení samotných jednotkových testů jsem použil frameworky NUnit a Moq:

```
[TestFixture]
class DbHelperTest
{
/// <summary>
/// This method automatically creates DbSet mock objects
/// </summary>
/// <typeparam name="T">Class to be mocked.</typeparam>
/// <param name="elements">Underlying data source.</param>
/// <returns></returns>
private static Mock<DbSet<T>> CreateDbSetMock<T>(List<T> elements) where T : class
{
var elementsAsQueryable = elements.AsQueryable();
var dbSetMock = new Mock<DbSet<T>>();

dbSetMock.As<IQueryable<T>>().Setup(m =>
m.Provider).Returns(elementsAsQueryable.Provider);

dbSetMock.As<IQueryable<T>>().Setup(m =>
m.Expression).Returns(elementsAsQueryable.Expression);

dbSetMock.As<IQueryable<T>>().Setup(m =>
m.ElementType).Returns(elementsAsQueryable.ElementType);

dbSetMock.As<IQueryable<T>>().Setup(m =>
m.GetEnumerator()).Returns(elementsAsQueryable.GetEnumerator());

dbSetMock.Setup(d => d.Add(It.IsAny<T>())).Callback<T>(s => elements.Add(s));
dbSetMock.Setup(d => d.Remove(It.IsAny<T>())).Callback<T>(s => elements.Remove(s));
return dbSetMock;
}

[Test]
public void DeleteCategory_IdExists_DeletesOnlyCategoryNotArticles()
{
// Arrange categories
List<Category> categories = new List<Category>() {
new Category(){Id=1,Name="Nezařazeno"},
new Category(){Id=2,Name="Programování"},
new Category(){Id=3,Name="Škola"},
};
Mock<DbSet<Category>> mockSetCategories = CreateDbSetMock(categories);

// Arrange articles
List<Article> articles = new List<Article>() {
new Article(){Id=1,Name="Nezařazeno - článek 1"},
new Article(){Id=2,Name="Programování - článek 1"},
new Article(){Id=3,Name="Škola - článek 1"},
};
Mock<DbSet<Article>> mockSetArticles = CreateDbSetMock(articles);

// Arrange dbContext
Mock<IDatabaseContext> mockContext = new Mock<IDatabaseContext>();
mockContext.Setup(e=>e.Categories).Returns(mockSetCategories.Object);
mockContext.Setup(e => e.Articles).Returns(mockSetArticles.Object);

// Act - deleting first category
DbHelper helper = new DbHelper(mockContext.Object);
int idToBeDeleted = 2;
helper.DeleteCategory(idToBeDeleted);
```

```
 // Assert
int expectedNumberOfCategories = 2;
int actualNumberOfCategores = mockContext.Object.Categories.Count();
Assert.That(actualNumberOfCategores,Is.EqualTo(expectedNumberOfCategories));

int expectedNumberOfArticles = 3;
int actualNumberOfArticles = mockContext.Object.Articles.Count();
Assert.That(actualNumberOfArticles, Is.EqualTo(expectedNumberOfArticles));

}
}
```

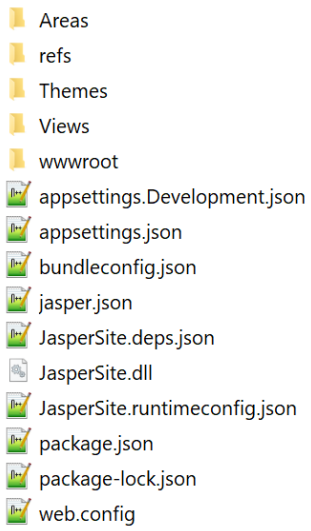
Výpis 4.11 – Jednotkový test

Uvedený kód představuje jednotkový test, který testuje metodu *DeleteCategory* třídy *DbHelper*. Tato metoda má na starost odstranění rubriky na základě jejího *Id*. Metoda se dále stará o to, že všechny články přiřazené do této rubriky nebudou smazány, ale budou přesunuty do rubriky „Nezařazeno“. Tuto funkcionalitu testuje metoda jednotkového testu s názvem: *DeleteCategory_IdExists_DeletesOnlyCategoryNotArticles*, která dodržuje doporučenou konvenci pojmenovávání metod jednotkových testů. Metoda *CreateDbSetMock* je pouze pomocná metoda pro vytvoření jednotlivých objektů typu *DbSet*, které odpovídají jednotlivým tabulkám v relační databázi. Samotný jednotkový test poté simulovanou (mockovanou) databázi naplní testovacími daty. Jelikož se jedná pouze o zástupný objekt databáze bez vazby na databázi skutečnou, není nijak do skutečné databáze zasahováno a testy nemají na reálnou databázi dopad. Díky tomu poté není testován datový zdroj sám o sobě (reálně databáze může být nedostupná, nebo vrátí neočekávaná data) a dochází k testování funkčnosti jen dané metody.

4.5 Instalace a použití

Celý redakční systém byl od počátku navrhován tak, aby se výsledná aplikace co nejvíce blížila praktické použitelnosti. Z tohoto důvodu byly alespoň v redukované podobě implementovány funkcionality systému, kterým se věnuje tato kapitola. Určité funkcionality se dají považovat za stěžejní a některé slouží pouze ke zrychlení práce uživatele.

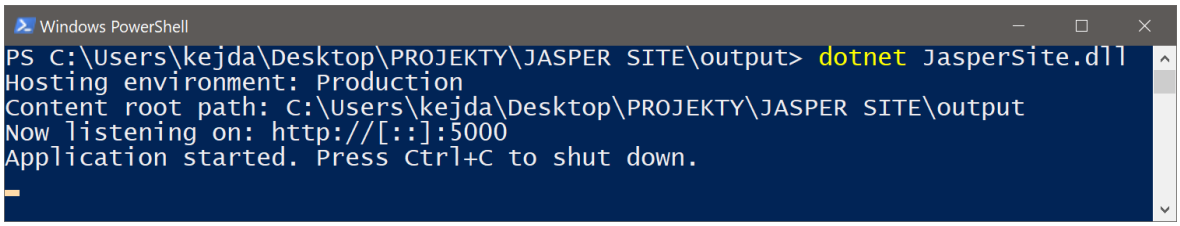
4.5.1 Instalace redakčního systému



Obrázek 4.9 – Adresář zkompilovaného projektu – zkráceno

Následující obrázek ukazuje adresářovou strukturu zkompilovaného redakčního systému, který je připraven k nasazení do provozu. Uvedený zkrácený výpis znázorňuje hlavní *.dll* knihovnu obsahující zkompilovaný projekt, dále konfigurační soubor *jasper.json* a další konfigurační soubory. Jelikož nejsou jednotlivé pohledy kompilovány, tak složka *Areas* obsahuje jednotlivé pohledy (tedy *.cshtml* soubory) samotného redakčního systému. Složka *Themes* obsahuje jednotlivé vzhledy.

Při nasazení redakčního systému na lokální server Kestrel je možné redakční systém spustit pomocí příkazového řádku Windows pomocí příkazu *dotnet JasperSite.dll*. Program se spustí za předpokladu, že na cílovém počítači je nainstalováno běhové prostředí .NET Core:



Obrázek 4.10 – Spuštění programu pomocí nástroje PowerShell

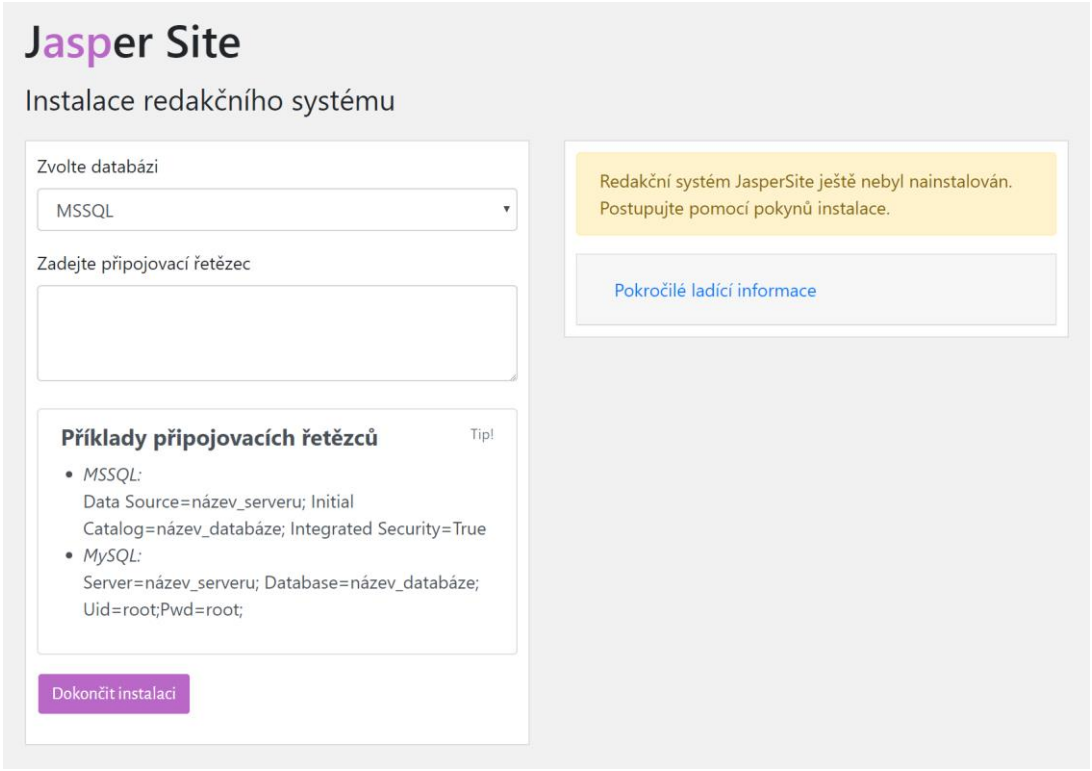
Za předpokladu, že nebyl ručně upraven konfigurační soubor *jasper.json*, se zobrazí instalační okno redakčního systému. Následující výpis ukazuje obsah konfiguračního souboru *jasper.json* před zahájením instalace a založením databáze:

```
{
  "themeName": "",
  "connectionString": "",
  "typeOfDatabase": "",
  "installationCompleted": ""
}
```

Výpis 4.12 – Obsah konfiguračního souboru jasper.json před instalací

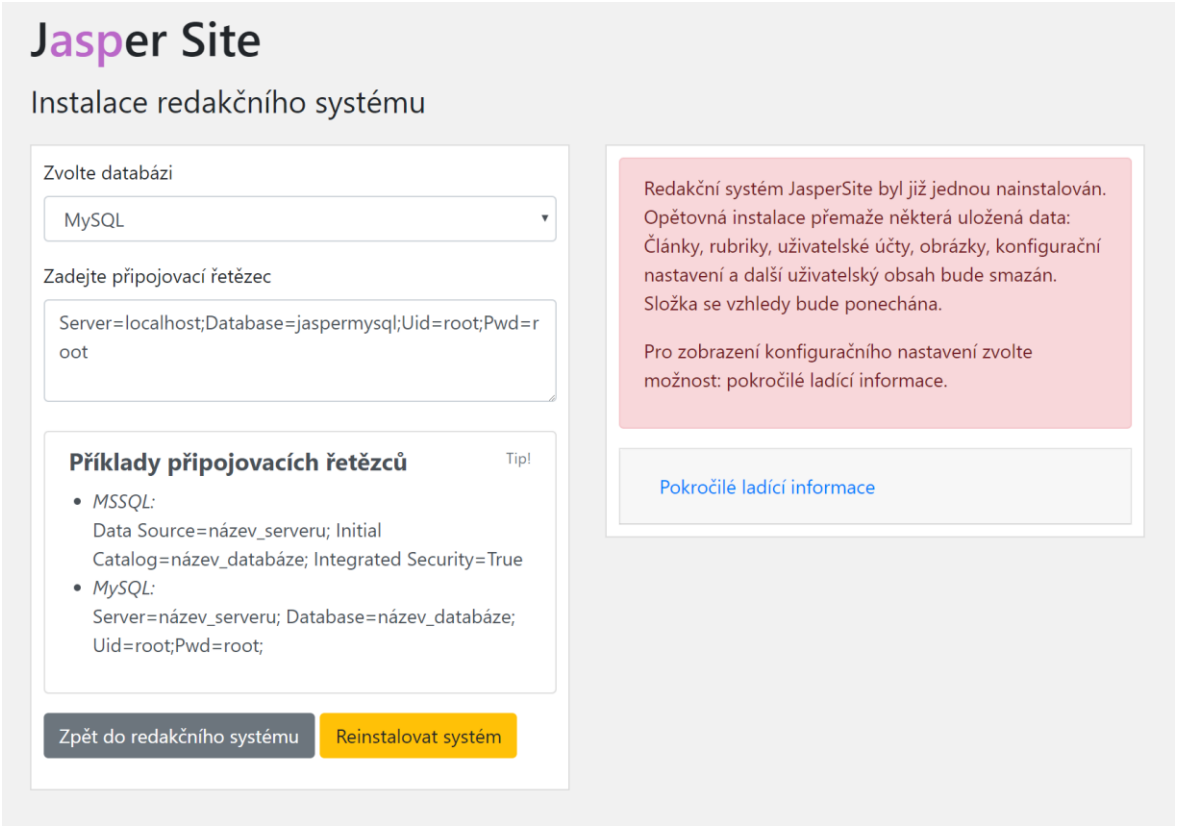
Pokud tento konfigurační soubor v adresáři není přítomen, dojde k jeho automatickému vytvoření.

Následující obrázek ukazuje hlavní instalační stránku redakčního systému:



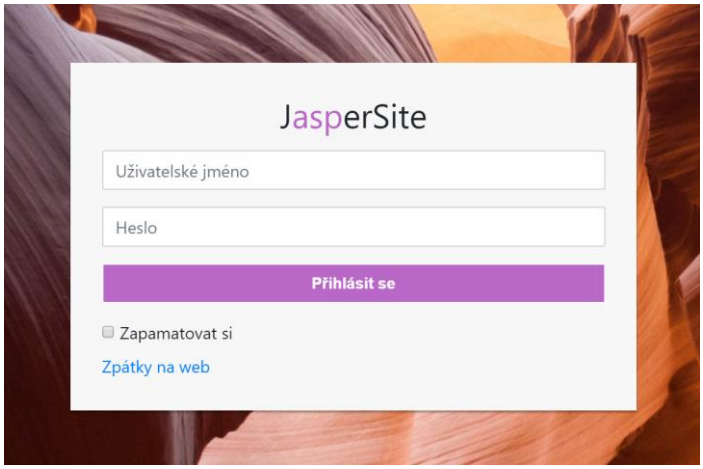
Obrázek 4.11 – Hlavní instalační stránka redakčního systému

V případě již dokončené instalace tato samá stránka slouží pro reinstalaci systému:



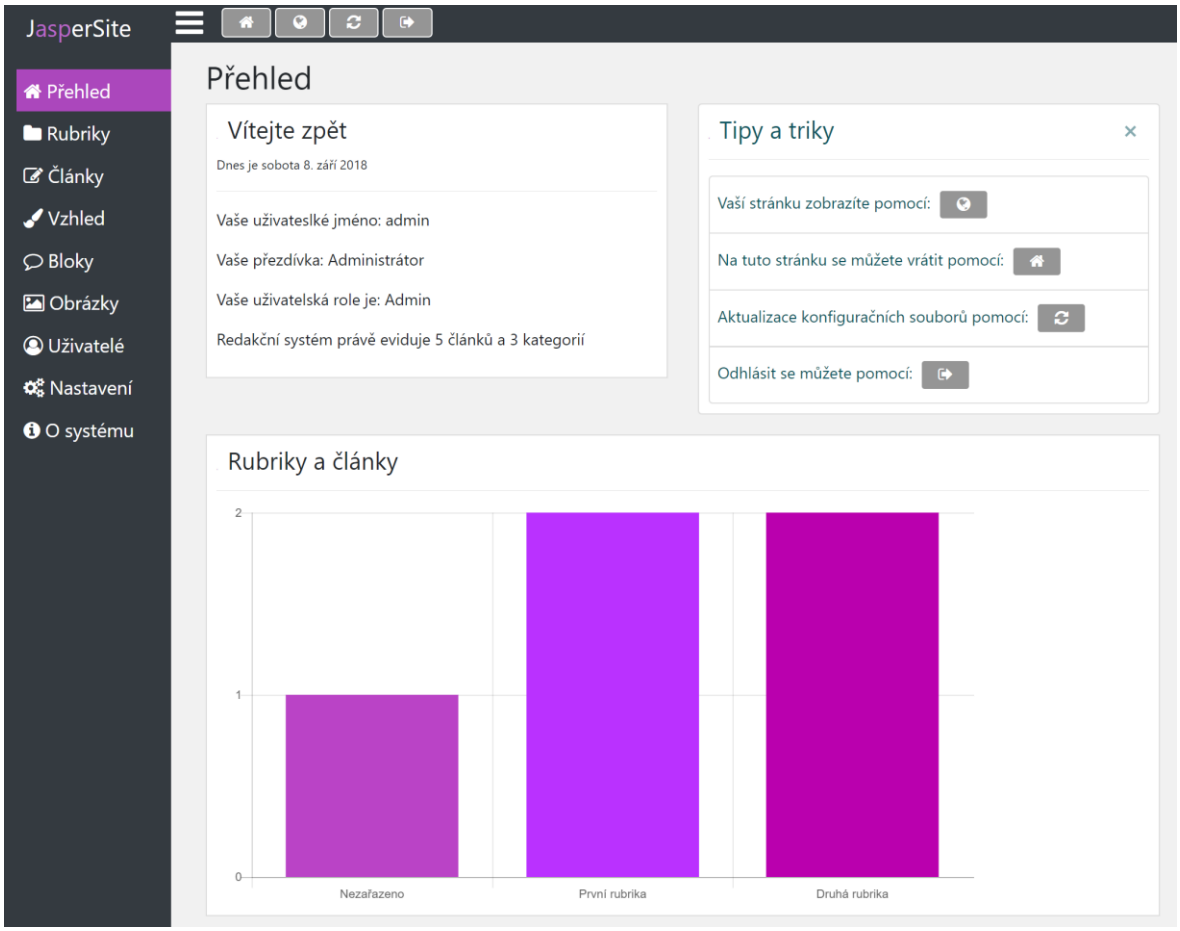
Obrázek 4.12 – Reinstalace systému

Po zvolení typu databáze a zadání připojovacího řetězce je uživatel přesměrován na přihlašovací obrazovku, kde zadá předpřipravené uživatelské jméno: *admin* a heslo: *admin*, popřípadě jméno: *redaktor* a heslo: *redaktor*:



Obrázek 4.13 – Přihlášení do administrace

4.5.2 Hlavní stránka redakčního systému

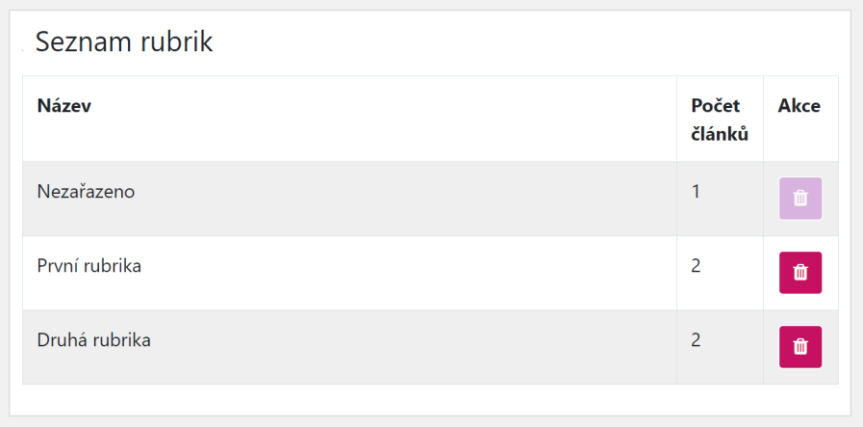


Obrázek 4.14 – Hlavní stránka administrace

Následně je uživatel přesměrován do hlavní administrace webové stránky, jak ukazuje předchozí obrázek. Administrační sekce v levé části obsahuje jednotlivé položky redakčního systému. Hlavní lišta obsahuje odkaz na hlavní stránku redakčního systému, ikona planety otevře v novém okně samotnou spravovanou webovou stránku, šipky v kruhu slouží k načtení konfiguračních dat a na konci se nachází tlačítko pro odhlášení aktuálně přihlášeného uživatele. Hlavní oblast stránky obsahuje mimo jiné i graf počtu rubrik a jejich příslušných článků.

4.5.3 Správa rubrik

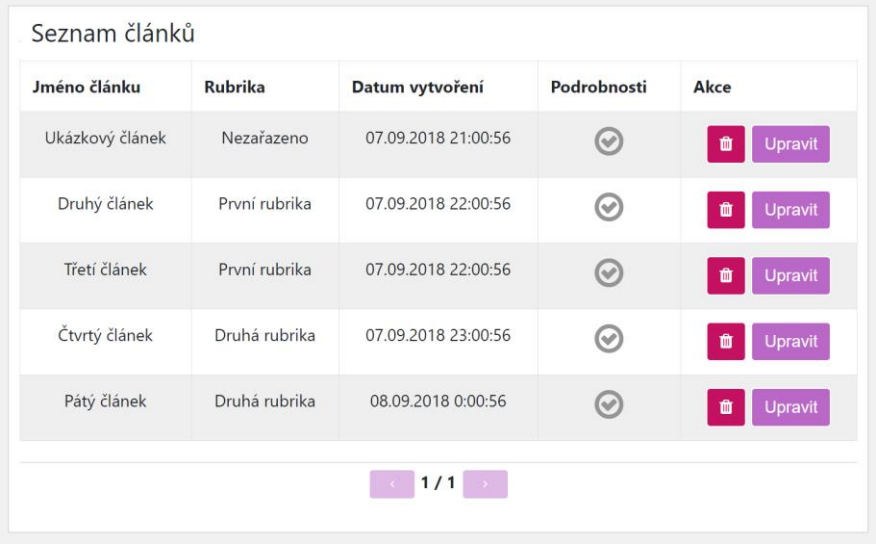
Záložka rubriky umožňuje přidávat a odstraňovat rubriky. Odstraněním rubriky s přiřazenými články dojde k přesunu všech příslušných článků do rubriky „Nezařazeno“.



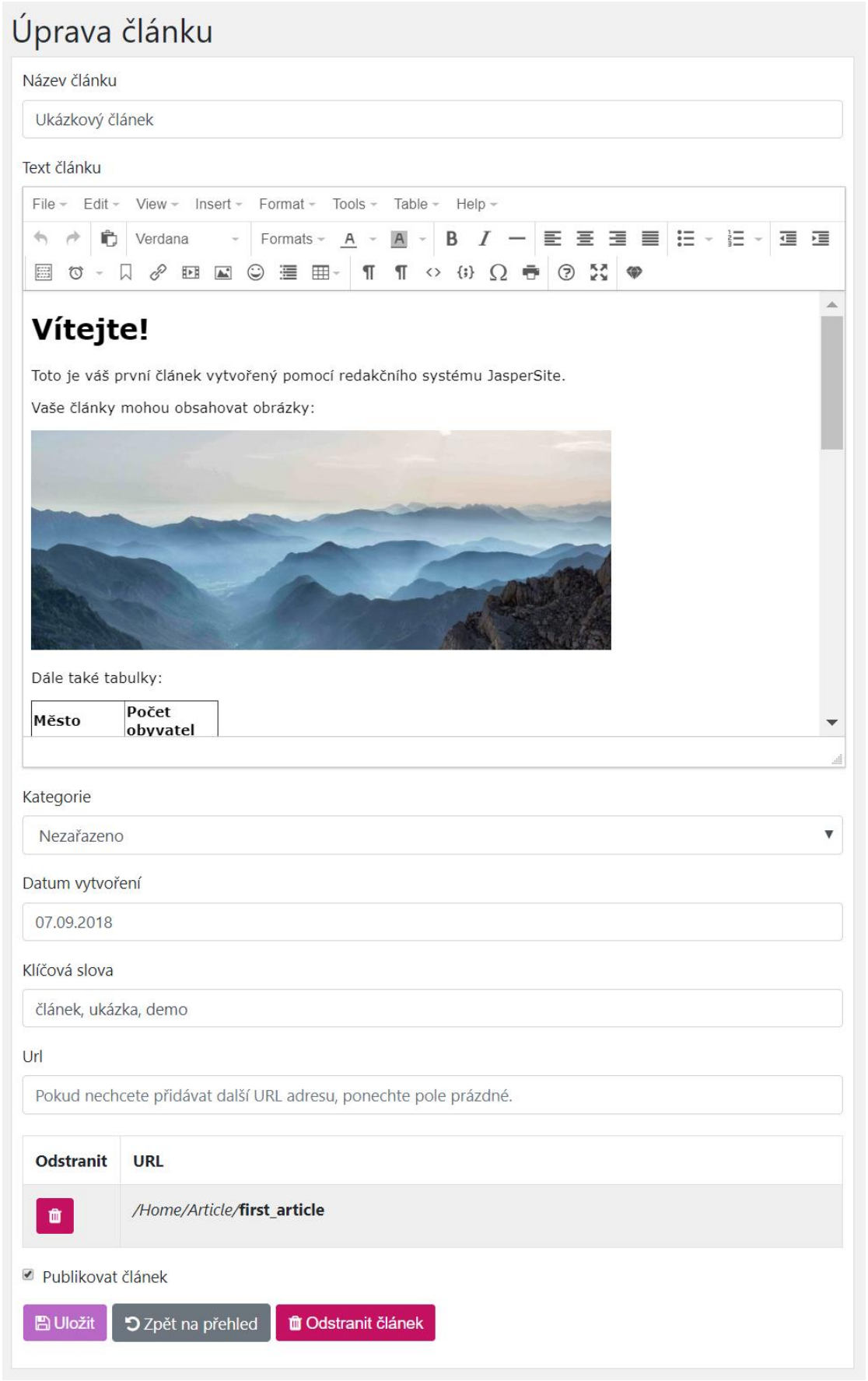
Obrázek 4.15 – Sekce rubriky

4.5.4 Správa článků

Záložka články slouží k přidávání, mazání článků a jejich editaci:



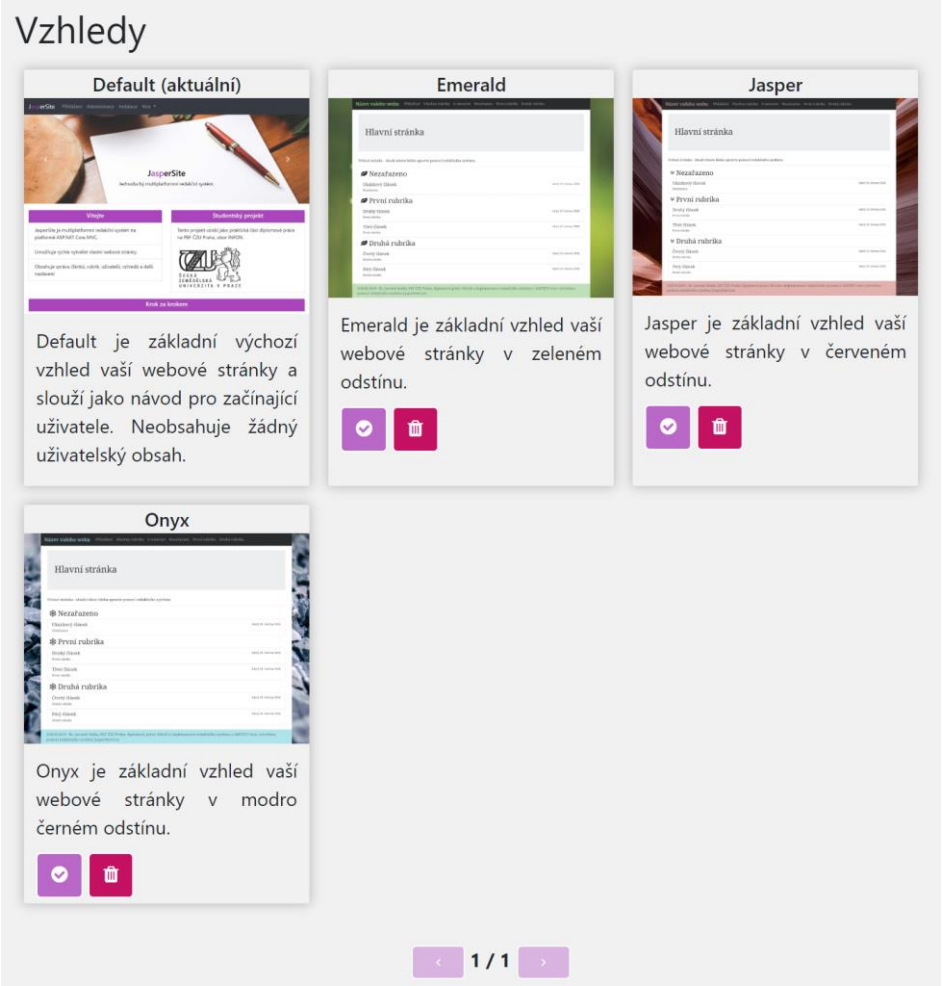
Obrázek 4.16 – Seznam článků



Obrázek 4.17 – Editace článku

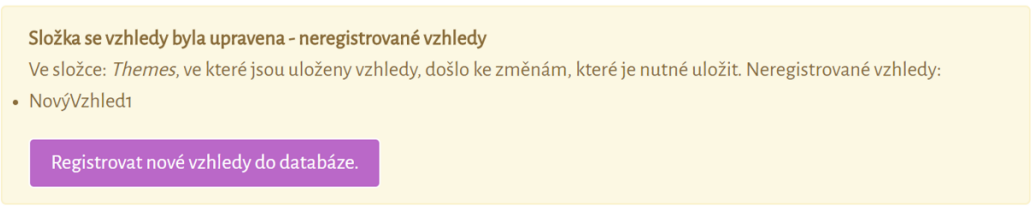
4.5.5 Správa vzhledů

Záložka vzhledy umožňuje přepínat a odstraňovat nainstalované vzhledy:

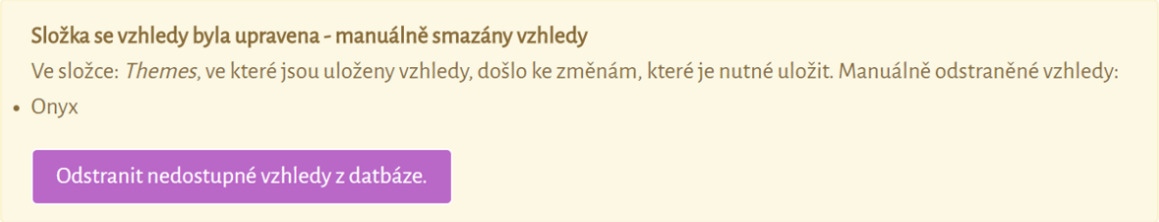


Obrázek 4.18 – Správa vzhledů

Redakční systém obsahuje algoritmus pro inteligentní změnu vzhledu. Pokud nově aktivovaný vzhled obsahuje stejně pojmenované kontejnery, tak se textové bloky automaticky přiřadí k novým kontejnerům. Pokud dojde k ručnímu smazání, nebo přidání vzhledu uživatelem, redakční systém je schopný tyto změny detekovat a uložit je do databáze:

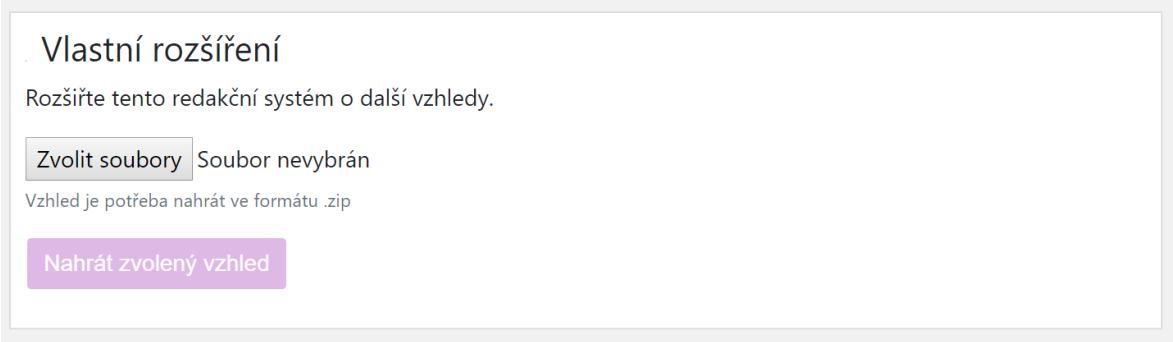


Obrázek 4.19 – Neregistrované vzhledy



Obrázek 4.20 – Manuálně odstraněné vzhledy

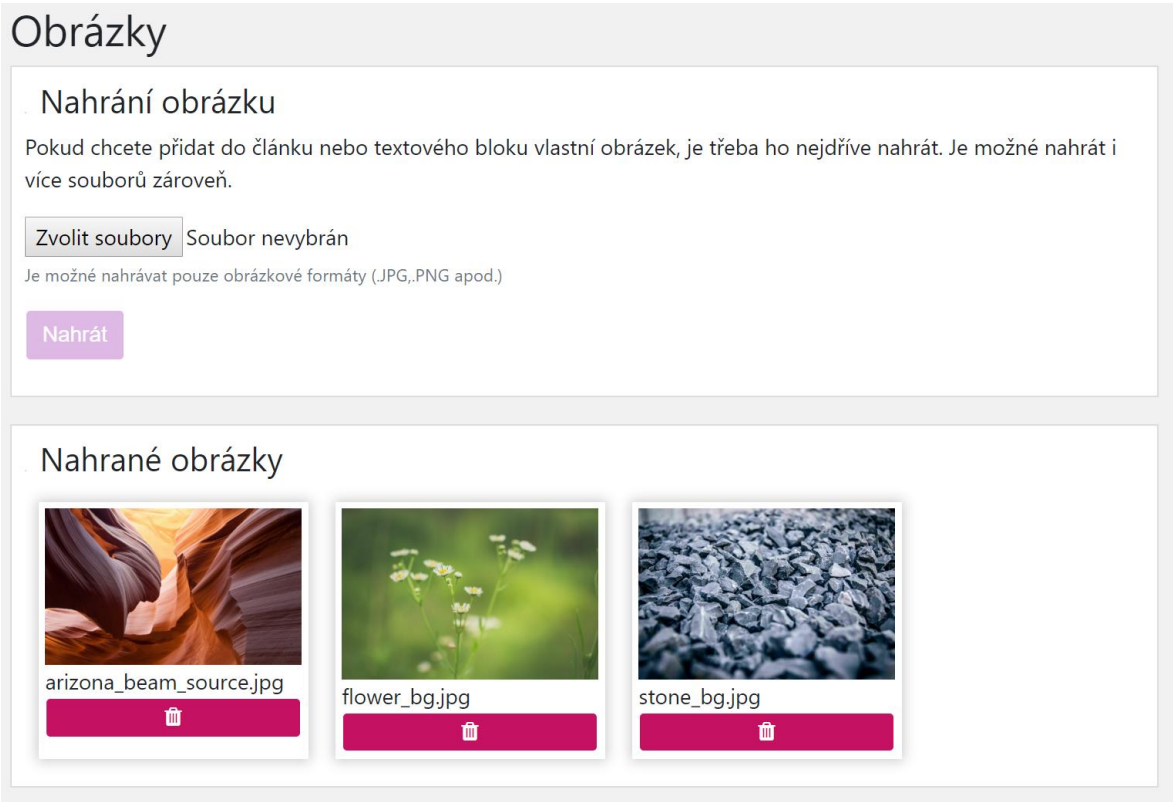
Vlastní vzhledy je možné nahrávat i přes grafické rozhraní:



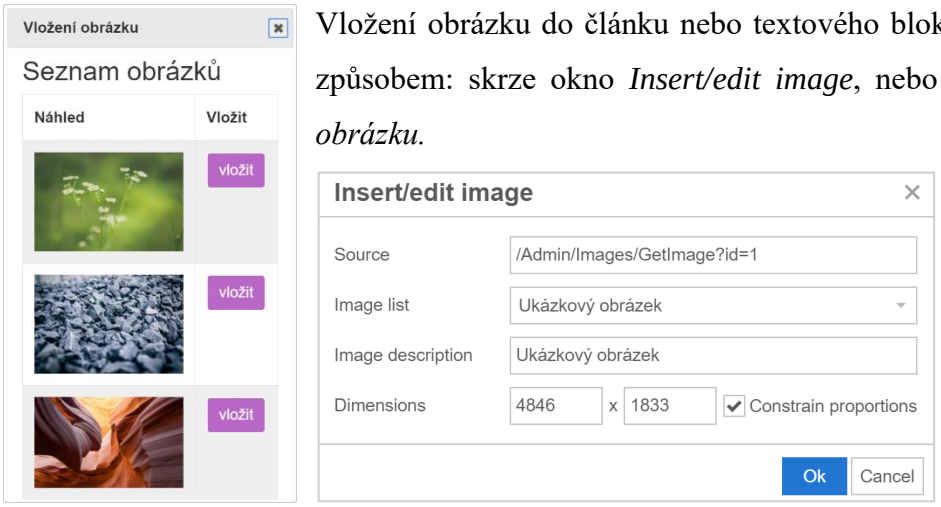
Obrázek 4.21 – Nahrání vzhledu přes grafické rozhraní

4.5.6 Správa obrázků

Další záložkou jsou obrázky, které je možné přidávat, mazat a poté následně přidávat do článků a textových bloků:



Obrázek 4.22 – Nahrání obrázku

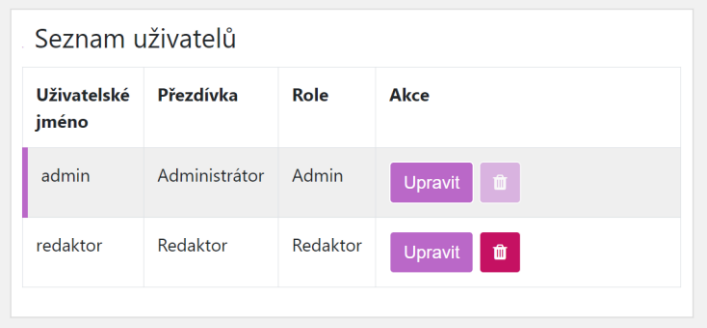


Vložení obrázku do článku nebo textového bloku je možné dvojím způsobem: skrze okno *Insert/edit image*, nebo přes okno *Vložení obrázku.*

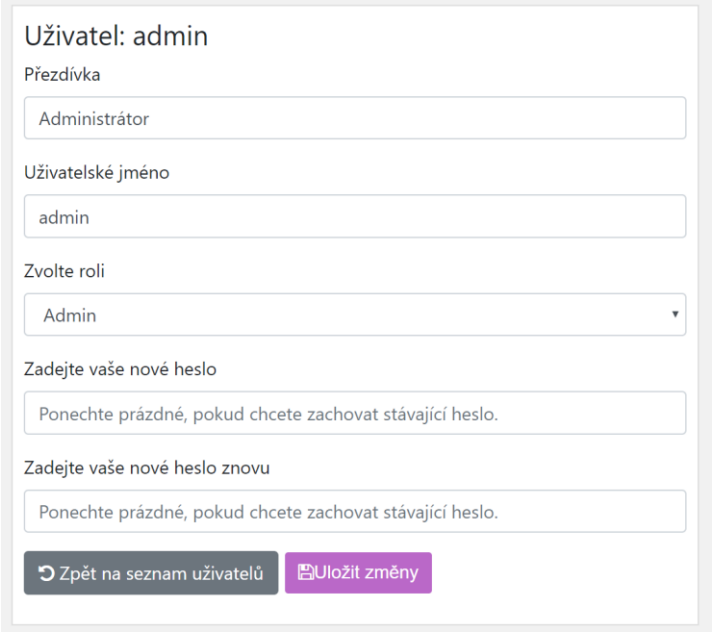
Obrázek 4.23 – Vložení obrázku do článku

4.5.7 Správa uživatelů

Záložka uživatelé umožnuje spravovat uživatele a upravovat jeho detaily:



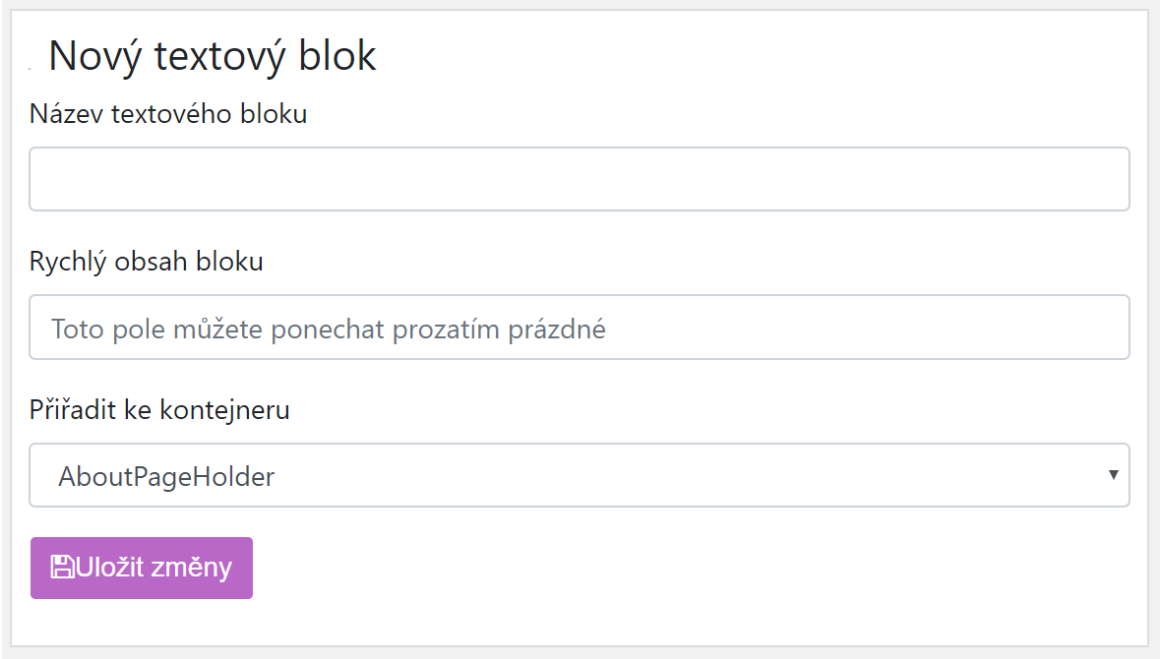
Obrázek 4.24 – Seznam uživatelů



Obrázek 4.25 – Detaily uživatele

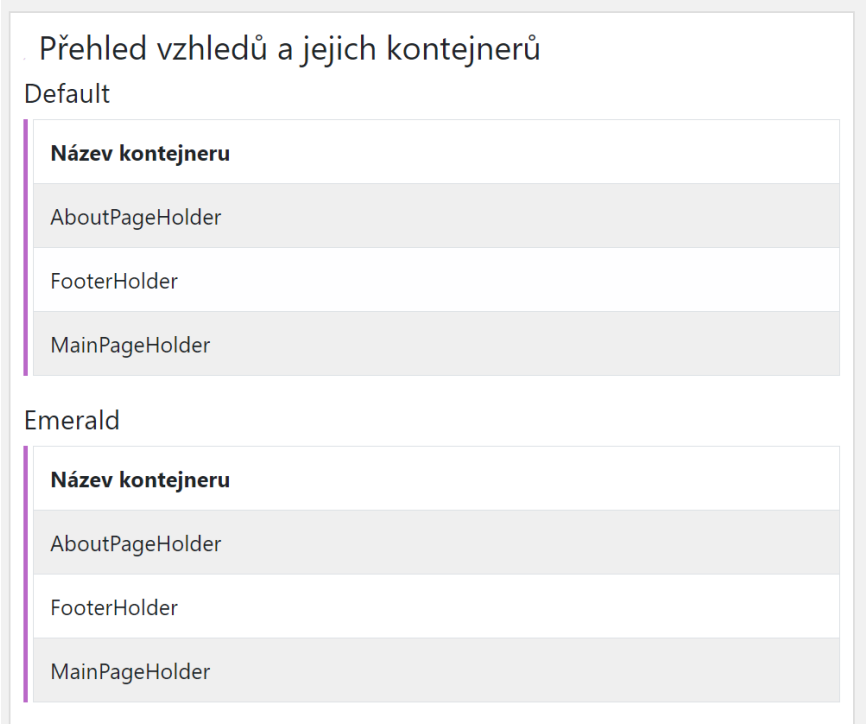
4.5.8 Správa textových bloků

Jednou z obsáhlejších sekcí administrace je správa textových bloků a jejich přiřazených kontejnerů:



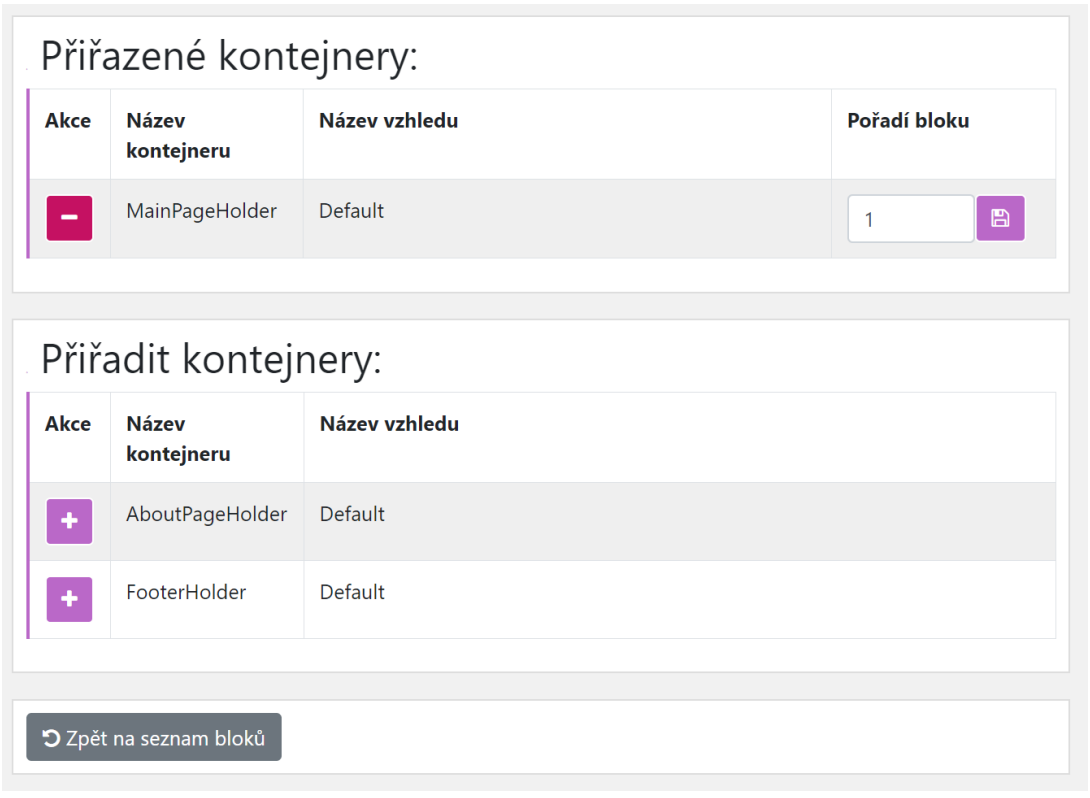
Obrázek 4.26 – Přidání nového textového bloku

Obrázek 4.27 – Přiřazené kontejnery textových bloků



Obrázek 4.28 – Přehled vzhledů a jejich kontejnerů

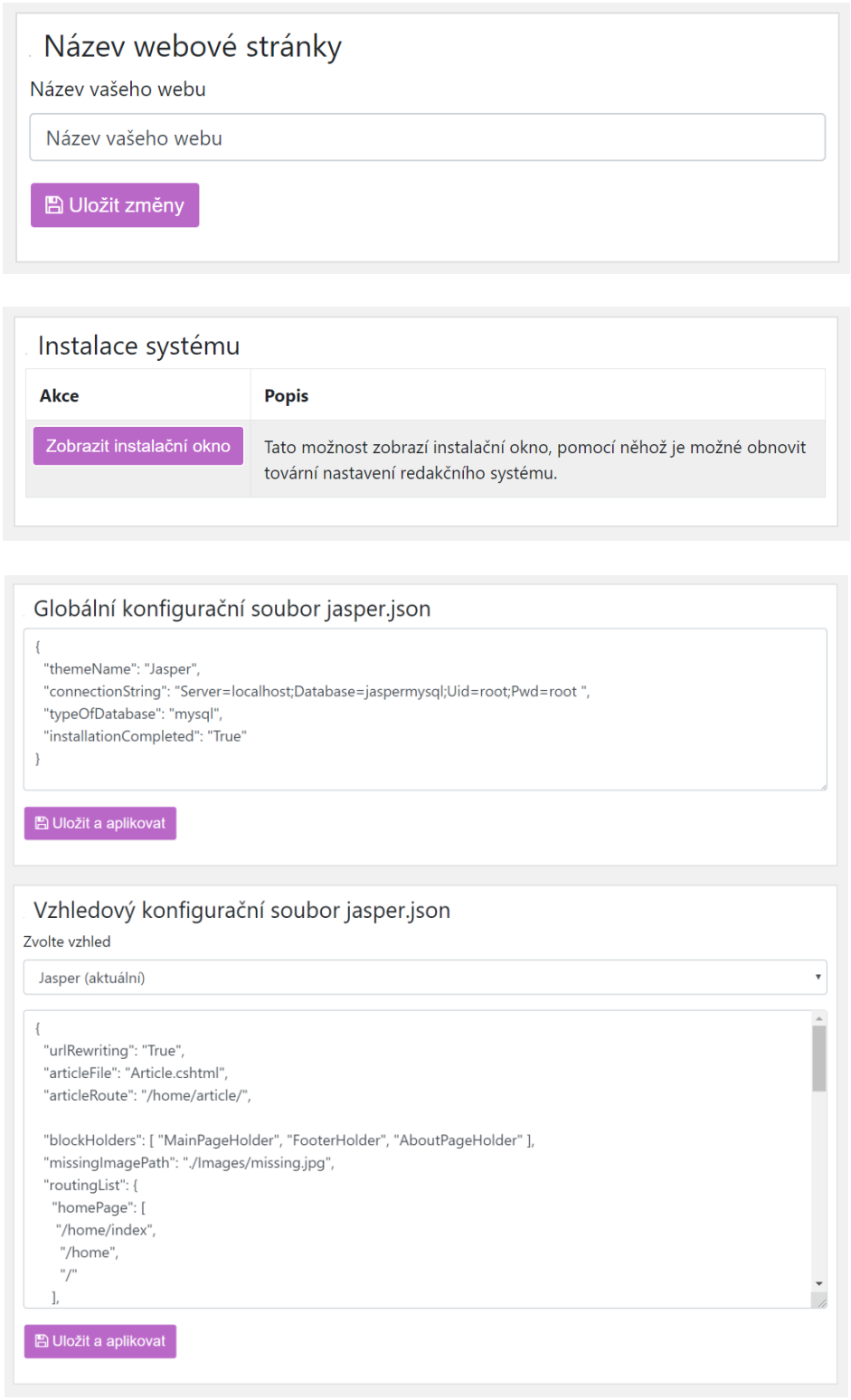
Úprava textového bloku obsahuje stejný pokročilý textový editor, jak tomu bylo u článků a možnosti přiřadit textový blok k pojmenovaným kontejnerům. Dále je možné stanovit pořadí bloku v kontejneru:



Obrázek 4.29 – Úprava textového bloku

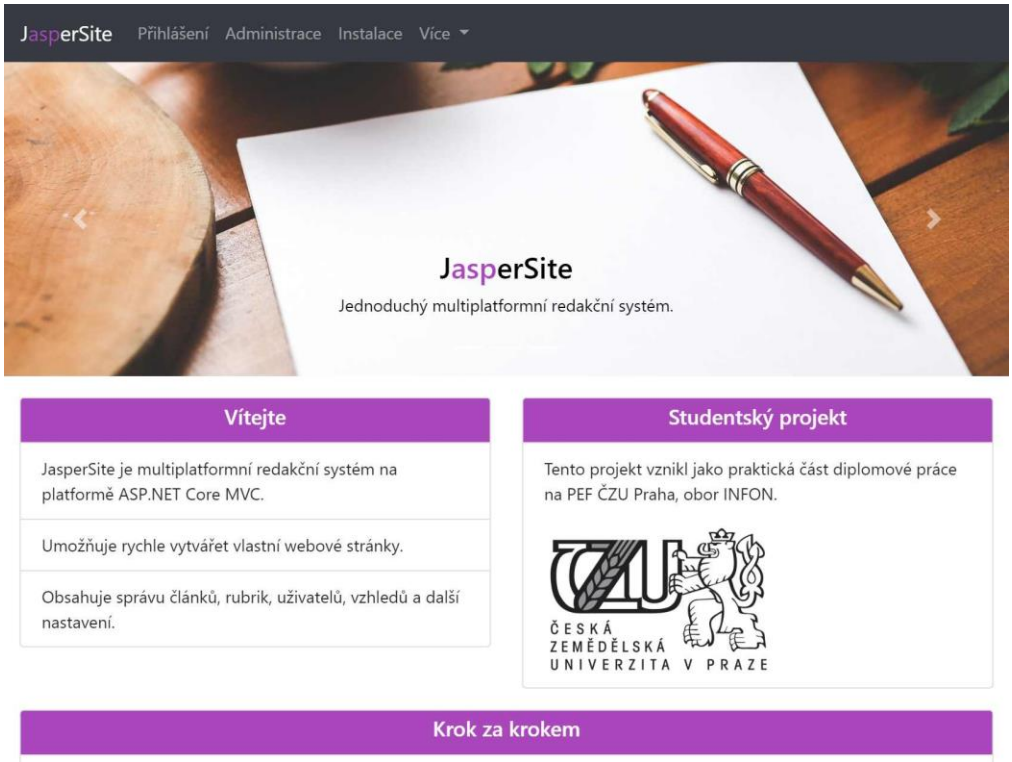
4.5.9 Nastavení

Záložka nastavení skýtá možnost změnit název webu, spustit opětovnou instalaci, nebo upravit konfigurační nastavení:

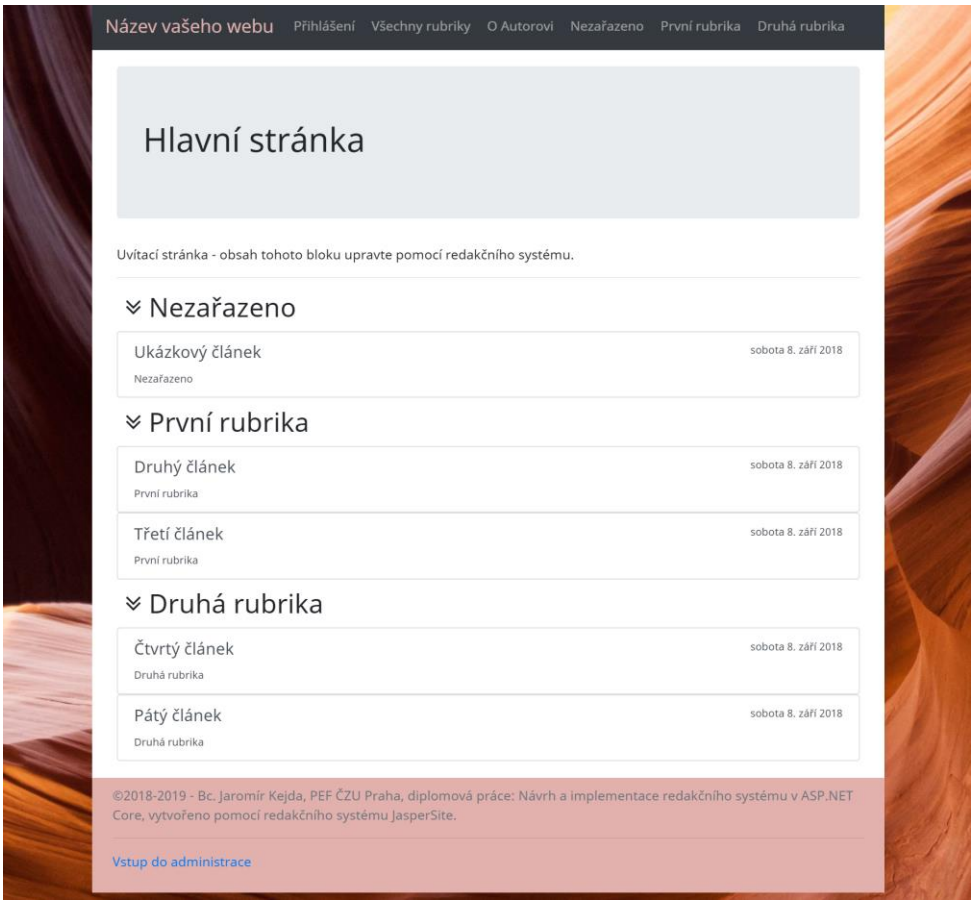


Obrázek 4.30 – Sekce nastavení

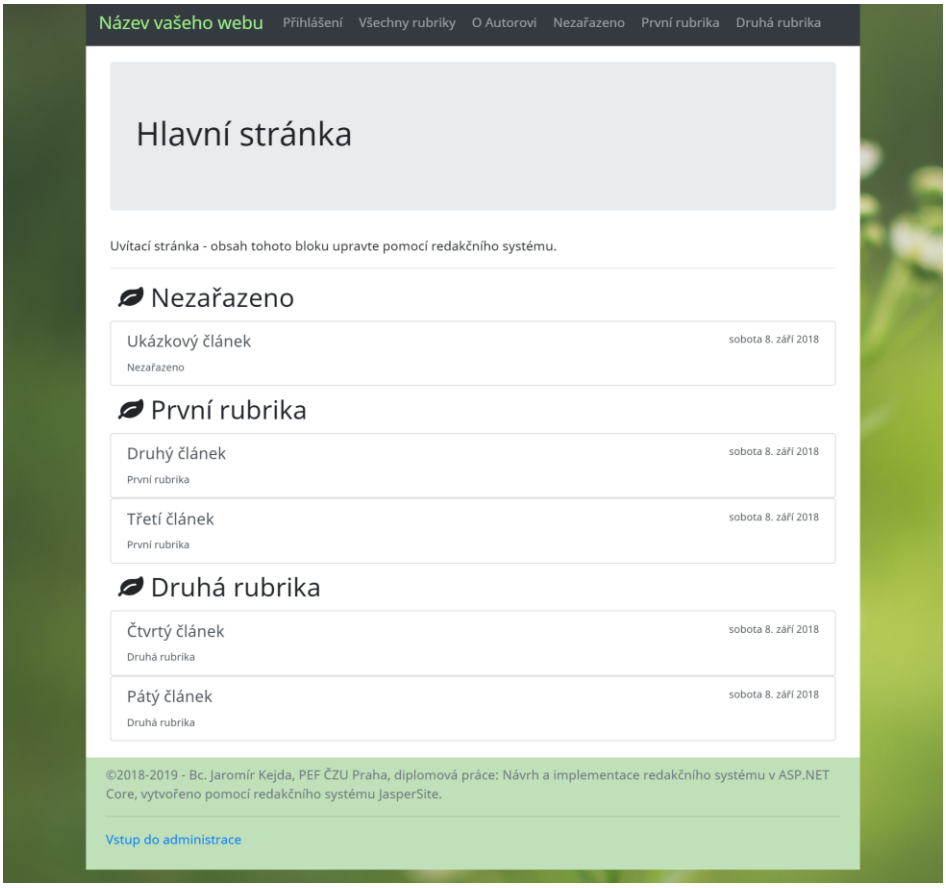
4.5.10 Vzhled stránky



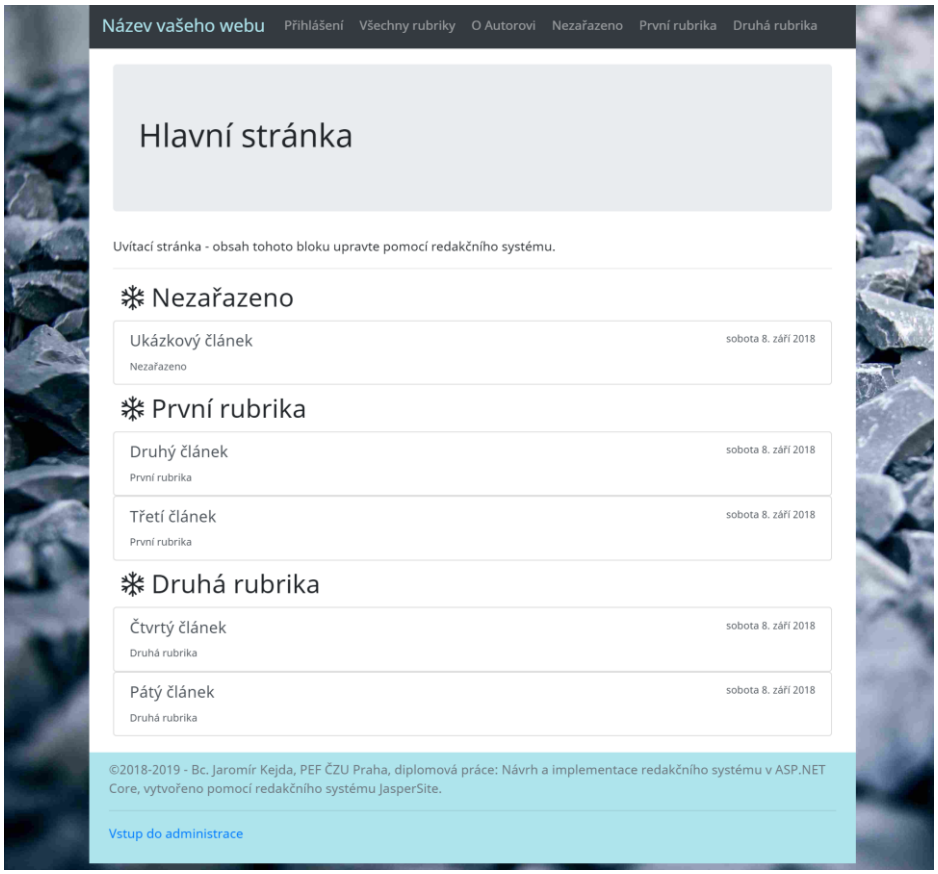
Obrázek 4.31 –Výchozí vzhled



Obrázek 4.32 – Vzhled Jasper



Obrázek 4.33 – Vzhled Emerald



Obrázek 4.34 – Vzhled Onyx

Název vašeho webu Přihlášení Všechny rubriky O Autorovi Nezařazeno První rubrika Druhá rubrika

Ukázkový článek

Publikováno: sobota 8. září 2018, 14:14:38

Vítejte!

Toto je váš první článek vytvořený pomocí redakčního systému JasperSite.

Vaše články mohou obsahovat obrázky:

Dále také tabulky:

Město	Počet obyvatel
Praha	1,3 mil

Je možné používat i zvýraznění syntaxe pro jazyky jako: C#, F#, CSS, HTML, JavaScript, Smalltalk a další:

```
// Ukázka kódu tohoto redakčního sysétmu
private static Mock<DbSet<T>> CreateDbSetMock<T>(List<T> elements) where T : class
        {
            var elementsAsQueryable = elements.AsQueryable();
            var dbSetMock = new Mock<DbSet<T>>();

            dbSetMock.As<IQueryable<T>>().Setup(m => m.Provider).Returns(elementsAsQueryable.Provider);
            dbSetMock.As<IQueryable<T>>().Setup(m => m.Expression).Returns(elementsAsQueryable.Expression);
            dbSetMock.As<IQueryable<T>>().Setup(m => m.ElementType).Returns(elementsAsQueryable.ElementType);
            dbSetMock.As<IQueryable<T>>().Setup(m => m.GetEnumerator()).Returns(elementsAsQueryable.GetEnumerator());
            dbSetMock.Setup(d => d.Add(It.IsAny<T>())).Callback<T>(s => elements.Add(s));
            dbSetMock.Setup(d => d.Remove(It.IsAny<T>())).Callback<T>(s => elements.Remove(s));
            return dbSetMock;
        }
```

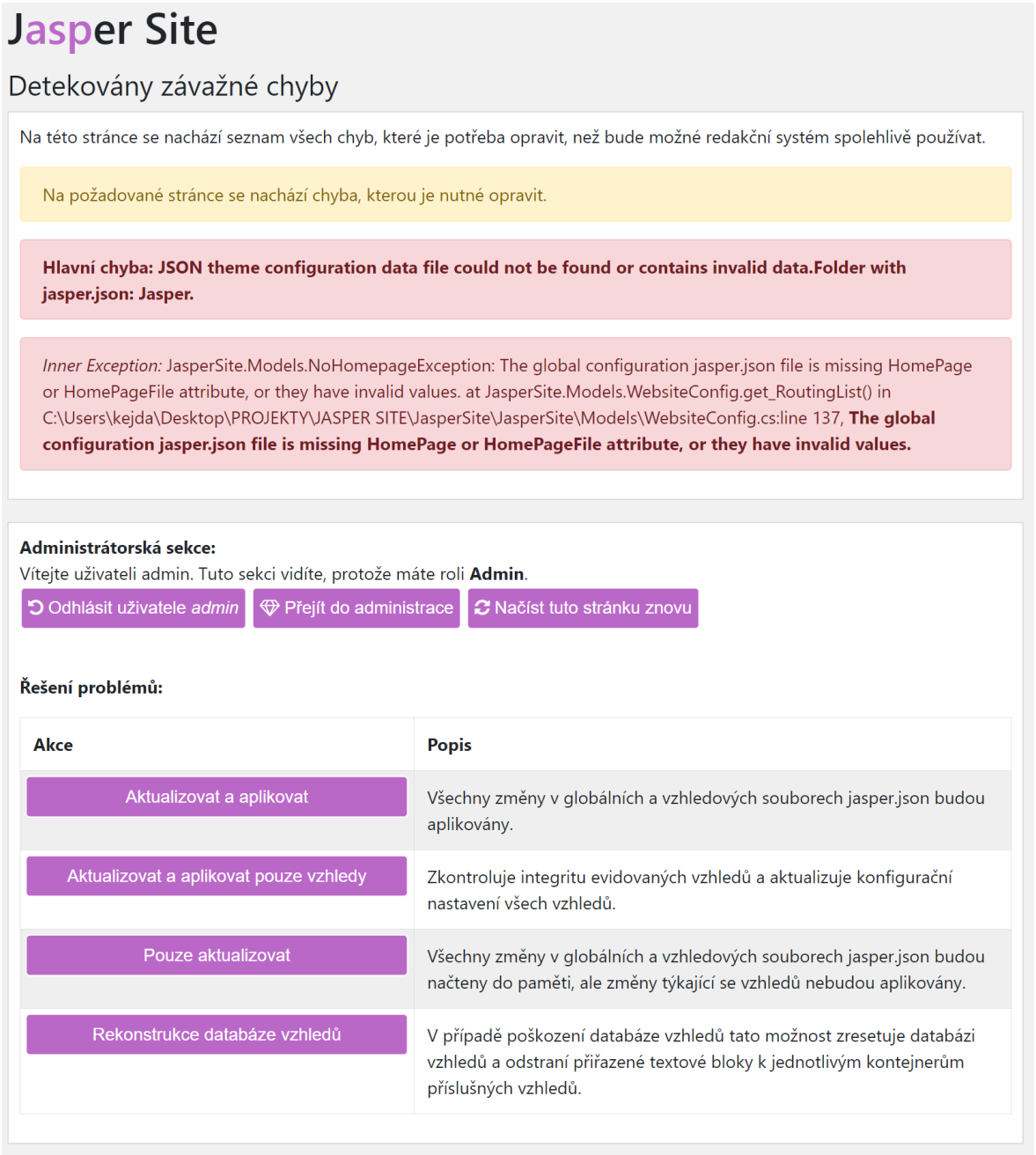
Můžete přidávat i videa, odkazy, nebo emotikony 😃 :

Vstup do administrace

Obrázek 4.35 – Ukázka článku na webu

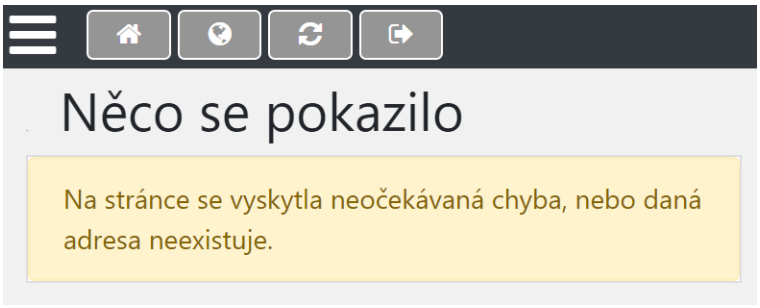
4.5.11 Detekce chyb a chybová hlášení

Jednou z podstatných vlastností softwarových systémů je schopnost detekovat neočekávané situace a zotavit se z nich. Redakční systém JasperSite byl budován s ohledem na robustnost celého řešení, a tudíž obsahuje několik úrovní zachytávání chyb. Nejhrubším systémem zachycování chyb je kontrola tzv. **závažných chyb**, při kterých není možné spustit redakční systém. Tyto chyby může uživatel řešit pomocí následující stránky:



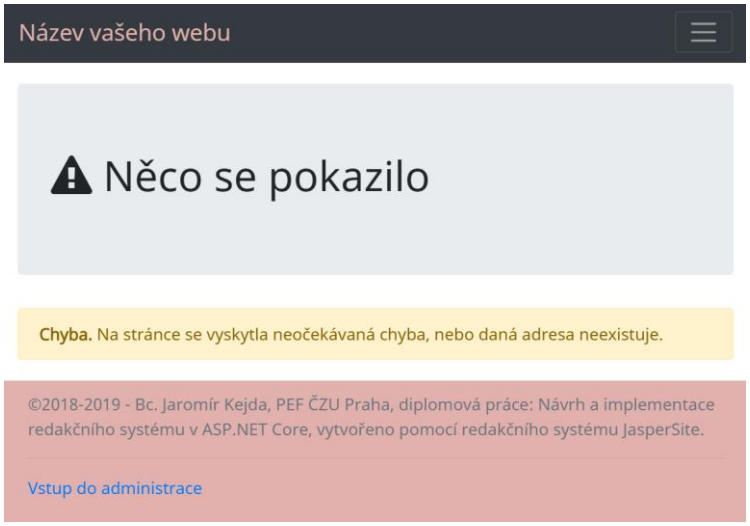
Obrázek 4.36 – Stránka závažných chyb

Dalším druhem chyb jsou **chyby administrace redakčního systému**, které jsou reprezentovány následující chybovou hláškou:



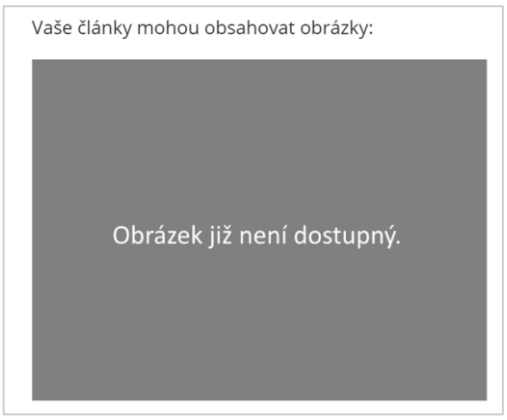
Obrázek 4.37 – Chyba administrace systému

Dalším typem chyb jsou **chyby samotné webové stránky**, které uvidí návštěvník, pokud zadá neexistující adresu, nebo nastane-li nespecifikovaná chyba redakčního systému, která umožňuje zobrazit toto hlášení:



Obrázek 4.38 – Chyba samotné webové stránky

Specifickým druhem chyby samotné webové stránky je nedostupnost nějakého zdroje, zde například obrázku, který byl ze serveru smazán, nebo je nedostupný z jiného důvodu:



Obrázek 4.39 – Nedostupný zdroj

4.6 Nasazení řešení na Azure

Redakční systém JasperSite by postrádal smysl, kdyby ho nebylo možné bezproblémově nasadit do provozu. Pro tento účel jsem se rozhodl v této kapitole využít služby Azure. Společnost Microsoft uvádí na svých stránkách: *"Microsoft Azure je neustále se rozšiřující sada cloudových služeb, které pomáhají vaší organizaci překonávat překážky v podnikání."* Microsoft Azure slouží pro vytváření, testování, nasazování a řízení aplikací a služeb prostřednictvím datových center společnosti Microsoft. Jednou ze služeb Microsoft Azure je možnost hostovat ASP.NET Core webové aplikace. Další službou, která je potřeba pro nasazení redakčního systému JasperSite je služba MSSQL nebo MySQL databáze. (*What is Azure*, nedatováno)

V rámci akademické licence společnost Microsoft nabízí studentům zdarma k vyzkoušení služby Azure po dobu jednoho roku spolu s disponibilním kreditem 100$. (*Vytvoření bezplatného účtu Azure,* nedatováno)

Po registraci do služby Azure je nutné přiřadit ke svému účtu prostředky nutné pro běh .NET Core aplikace, na obrázku č. 4.40 uvedeno jako *App Service*. Dále byla vytvořena instance MSSQL databáze. Dále je v seznamu uveden prostředek *Application Insights*, který je podporován samotným redakčním systémem a umožňuje monitoring aplikace.

Zdroj obrázku: https://portal.azure.com

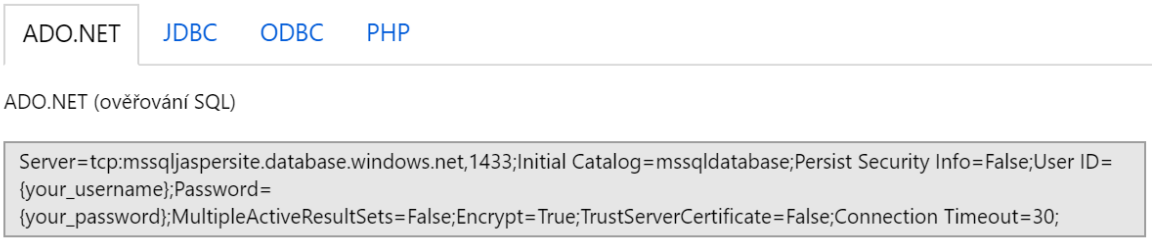
NÁZEV	TYP	UMÍSTĚNÍ
jaspersite	Application Insights	Západní Evropa
jaspersite	App Service	Západní Evropa
jaspersitehostingplan	Plán služby App Service	Západní Evropa
mssqljaspersite	SQL Server	Západní Evropa
mssqldatabase (mssqljaspersite/mssqldatabase)	Databáze SQL	Západní Evropa

Obrázek 4.40 – Seznam aktivovaných prostředků služby Azure

Samotnou aplikaci je možné nahrát na webový server například prostřednictvím Visual Studia. Azure poskytuje možnost si bezplatně zvolit doménu třetího řádu, v tomto případě byla zvolena doména *jaspersite.azurewebsites.net.* Po nahrání redakčního systému na server

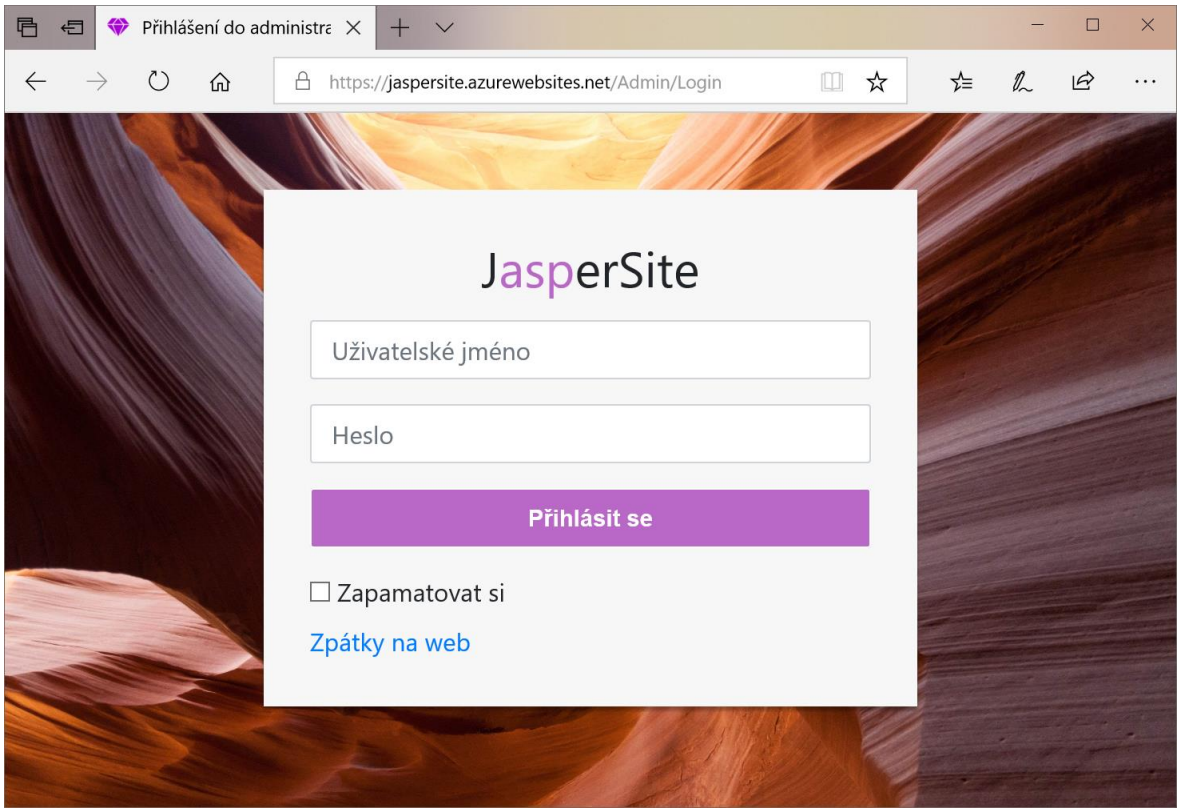
je třeba zadat připojovací řetězec k databázi. Ten je možné získat pomocí uživatelského rozhraní služby Azure:

Zdroj obrázku: https://portal.azure.com



Obrázek 4.41 – Připojovací řetězec k databázi ve službě Azure

Po dokončení instalace se zobrazí přihlašovací okno do administrátorské sekce. Jak je zřetelné z URL adresy prohlížeče, služba Azure implicitně použila protokol HTTPS:



Obrázek 4.42 – Nasazený redakční systém

5 Výsledky a diskuse

Implementaci redakčního systému předcházel jeho návrh, který byl proveden především prostřednictvím nástrojů jazyka UML. Tento návrh byl při následné implementaci dodržen.

Výsledná podoba implementovaného redakčního systému splnila požadavky, které byly autorem z počátku stanoveny. Pro shrnutí, redakční systém umožňuje spravovat příspěvky v daných kategoriích, zobrazovat textové bloky, obsahuje správu uživatelů a jejich rolí, podporuje přidávání vlastních vzhledů za běhu aplikace, podporuje nejenom příspěvky v podobě čistého textu, ale i obrázků, tabulek a videí, a v neposlední řade obsahuje i instalační stránku.

Co se týče jednotkových testů, tak kód byl pokryt jednotkovými testy jen ukázkově pro demonstrační potřeby této práce. Naznačený postup testování by měl být ideálně proveden i u ostatních částí kódu. Finálnímu testování výsledné aplikace byl věnován již velmi malý prostor a byla testována jen základní funkcionalita. Veškeré možné alternativy chování systému nebyly testovány a je zaručena spolehlivá funkčnost jen základních funkcí redakčního systému. Jsem si tedy vědom určitých nedostatků ve správném fungování systému, které by měly být adresovány před reálným nasazením redakčního systému do praxe.

Aplikace dále zlepšuje uživatelskou zkušenost použitím responzivního designu a technologie AJAX, a díky tomu je grafické rozhraní neustále responzivní na uživatelské podněty.

Jednou z kladných vlastností systému, které v práci nebyly zmíněny je použití nástrojů pro minifikaci a bundling javascriptových a CSS souborů pomocí nástrojů Webpack a Gulp. Díky tomu je webový server zatěžován zřetelně méně HTTP požadavky v rámci jedné stránky.

Dalším návrhem na možné zlepšení by bylo použití frameworku pro tvorbu rozšiřujících modulů pro redakční systém. V momentální podobě není kromě vzhledů možné nijak program modulárně rozšiřovat. Toto by mohlo změnit použití frameworku MEF (*Managed Extensibility Framework*).

Krátce bych zde zmínil také problematiku, kterou MacDonald (2012, s. 506) nazývá jako *concurrency checking*. Jedná se o skutečnost, že ve stejný okamžik mohou na jednom článku v redakčním systému pracovat současně dva uživatelé. Pokud první uživatel uloží svoji práci dříve než ten druhý, tak pochopitelně druhý uživatel stále pracuje se starou verzí a nové změny svému kolegovi přemaže. Tento problém není v této práci nijak řešen a je předmětem dalšího možného vývoje.

Redakční systém JasperSite je v momentální podobě teprve velmi mladým systémem, který si ani vzdáleně nemůže zadat s již etablovanými redakčními systémy. Přesto se ale jedná o systém, který doplňuje spíše řídké řady redakčních systémů na platformě .NET a ještě řidší řadu redakčních systémů na platformě .NET Core. Dle mého názoru se jedná o jednoduchý multiplatformní redakční systém pro menší webové stránky vhodný pro nenáročné uživatele za předpokladu, že projde celý systém procesem podrobného testování a opravením některých chyb.

Další rozvoj a budoucnost redakčního systému JasperSite vidím především v open source komunitním vývoji, který bude realizován prostřednictvím portálu GitHub, kde bude k dispozici i dokumentace.

6 Závěr

Hlavním cílem této práce byla implementace základní funkcionality webového redakčního systému pro správu obsahu webových stránek s využitím technologie ASP.NET Core MVC a jazyka C#. Tento cíl byl naplněn v praktické části této práce, konkrétně v rámci kapitoly 4.3. Tato kapitola také popsala nástroje použité při vývoji client-side a server-side části aplikace. Vzhledem k hlavnímu cíli práce byla v kapitole 4.3.1.4 popsána implementace datové perzistence a v kapitole 4.3.1.5 implementace vlastního routing systému a vzhledů.

Vedlejším cílem této práce bylo provést návrh tohoto systému pomocí standardních nástrojů softwarového inženýrství. V kapitole 4.1 byly stanoveny požadavky na výsledný systém a kapitola 4.2 se zabývala samotným návrhem. Následující kapitoly se věnovaly návrhu funkcionality systému, relačního databázového schématu, objektového modelu a adresářové struktury projektu.

Předmětem kapitoly 4.5 byl popis nasazení a použití implementovaného redakčního systému. V páté kapitole této práce nakonec autor kriticky zhodnotil dosažené výsledky a navrhl opravy či dodatečná vylepšení redakčního systému.

7 Seznam použitých zdrojů

Tištěné publikace

BARKER, Deane. *Web content management: systems, features, and best practices*. Boston: O'Reilly, 2016. ISBN 978-1-491-90812-9.

FREEMAN, Adam. *Pro ASP.NET Core MVC*. New York, NY: Springer Science Business Media, 2016. ISBN 978-1-4842-0398-9.

KEJDA, Jaromír. *Prostředky platformy .NET pro zpracování dat a zajištění datové perzistence*. Praha, 2017. Bakalářská práce. Česká zemědělská univerzita. Vedoucí práce Ing. Jiří Brožek, Ph.D.

KRÓL, Karol. *WordPress 4.x Complete*. 2015. Packt Publishing.ISBN 978-1-78439-090-7.

MACDONALD, Matthew. *Beginning ASP.NET 4.5 in C#*. New York, NY: Apress, 2012. ISBN 978-143-0242-529.

VRANA, Ivan. *Projektování informačních systémů s UML*. V Praze: Česká zemědělská univerzita, Provozně ekonomická fakulta, 2008. ISBN 978-80-213-1817-5.

ZABLOCKI, John. *Orchard CMS: up and running*. Beijing: O'Reilly, [2012]. ISBN 978-1-449-32021-8.

Elektronické zdroje

Active Server Pages. *Wikipedia* [online]. 3.9.2018 [cit. 2018-09-09]. Dostupné z: https://en.wikipedia.org/wiki/Active_Server_Pages

AMBLER, Scott, Eric HARTFORD a Andreas RUECKERT. *A UML Profile for Data Modeling* [online]. 2003 [cit. 2018-06-17]. Dostupné z: http://www.agiledata.org/essays/umlDataModelingProfile.html

AMBLER, Scott. *Towards a UML Persistence Model* [online]. 2000 [cit. 2018-06-17]. Dostupné z: https://www.omg.org/news/meetings/workshops/presentations/uml_presentations/4-3%20Ambler%20-%20Ambler_UML_Persistence.pdf

ATWOOD, Jeff. Understanding Model-View-Controller. *Coding Horror*[online]. 2008 [cit. 2018-06-23]. Dostupné z: https://blog.codinghorror.com/understanding-model-view-controller/

ASP and ASP.NET Tutorials. *W3schools* [online]. [cit. 2018-06-23]. Dostupné z: https://www.w3schools.com/asp/

ASP.NET Core. *Wikipedia* [online]. 2018 [cit. 2018-06-22]. Dostupné z: https://en.wikipedia.org/wiki/ASP.NET_Core

Cache in-memory in ASP.NET Core. *Microsoft Docs* [online]. 2016 [cit. 2018-06-23]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/performance/caching/memory?view=aspnetcore-2.1

Database Providers. *MS Docs* [online]. 2018 [cit. 2018-06-23]. Dostupné z: https://docs.microsoft.com/en-us/ef/core/providers/rk

Entity Framework. *Wikipedia* [online]. [cit. 2018-06-23]. Dostupné z: https://en.wikipedia.org/wiki/Entity_Framework

HEATHCOTE, Bryan. MVC vs 3 – Tier Pattern. *All Things CS* [online]. 2011 [cit. 2018-06-23]. Dostupné z: http://allthingscs.blogspot.com/2011/03/mvc-vs-3-tier-pattern.html

HTML <form> method Attribute. *W3schools* [online]. [cit. 2018-06-23]. Dostupné z: https://www.w3schools.com/tags/att_form_method.asp

Joomla vs. WordPress Comparison 2018. Firstsiteguide [online]. 2018 [cit. 2018-06-24]. Dostupné z: https://firstsiteguide.com/wordpress-joomla/

.NET architectural components. *Microsoft Docs* [online]. 2017 [cit. 2018-06-22]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/components

.NET Core Guide. *Microsoft Docs* [online]. 1.8.2018 [cit. 2018-09-09]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/core/

NET Foundation Projects [online]. [cit. 2018-06-22]. Dostupné z: https://www.dotnetfoundation.org/projects?q=Orchard&type=project

.NET Framework version history. *Wikipedia* [online]. 5.9.2018 [cit. 2018-09-09]. Dostupné z: https://en.wikipedia.org/wiki/.NET_Framework_version_history

Orchard CMS [online]. [cit. 2018-06-22]. Dostupné z: http://orchardproject.net/

PETERKA, Jiří. *Počítačové sítě, verze 4.0: Vývoj výpočetního modelu* [online]. 2014 [cit. 2018-06-20]. Dostupné z: http://www.earchiv.cz/l226/slide.php3?l=10&me=15

Popular CMS by Market Share. *WebsiteSetup* [online]. 2017 [cit. 2018-06-24]. Dostupné z: https://websitesetup.org/popular-cms/

REENSKAUG, Trygve. *MODELS - VIEWS - CONTROLLERS* [online]. 1979 [cit. 2018-06-23]. Dostupné z: http://heim.ifi.uio.no/~trygver/1979/mvc-2/1979-12-MVC.pdf

Server-side scripting. *Wikipedia* [online]. [cit. 2018-06-24]. Dostupné z: https://en.wikipedia.org/wiki/Server-side_scripting

Session and app state in ASP.NET Core. *Microsoft Docs* [online]. 14.8. 2018 [cit. 2018-06-23]. Dostupné z: https://docs.microsoft.com/cs-cz/aspnet/core/fundamentals/app-state?view=aspnetcore-2.1#tempdata

Shocking WordPress Stats 2017. *Whoishostingthis* [online]. 2017 [cit. 2018-06-24]. Dostupné z: https://www.whoishostingthis.com/compare/wordpress/stats/

Tour of .NET. Microsoft Docs [online]. 2017 [cit. 2018-06-22]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/standard/tour

Unified Modeling Language [online]. 2007 [cit. 2018-02-09]. Dostupné z: http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF

Vytvoření bezplatného účtu Azure. *Microsoft Azure* [online]. [cit. 2018-09-08]. Dostupné z: https://azure.microsoft.com/cs-cz/free/students/

What is Azure. *Microsoft Azure* [online]. [cit. 2018-09-08]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-azure/

World Wide Web. *Wikipedia* [online]. [cit. 2018-06-24]. Dostupné z: https://cs.wikipedia.org/wiki/World_Wide_Web

8 Přílohy

A. Entity databázového kontextu

```
public class Article
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required]
    public string Name { get; set; }

    public string HtmlContent { get; set; }

    [Required]
    public DateTime PublishDate { get; set; }

    [Required]
    [ForeignKey("Category")]
    public int CategoryId { get; set; }
    public Category Category { get; set; }

    public ICollection<User> Users { get; set; }
}

public class Category
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required]
    public string Name { get; set; }

    public ICollection<Article> Articles { get; set; }
}

public partial class User
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required]
    public string Nickname { get; set; }

    [Required]
    public string Username { get; set; }

    [Required]
    public string Password { get; set; }

    [Required]
    public string Salt { get; set; }
```

```
    [Required]
    [ForeignKey("Role")]
    public int RoleId { get; set; }
    public Role Role { get; set; }
}

public class Role
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required]
    public string Name { get; set; }
}

public class Setting
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id{ get; set; }

    [Required]
    public string Key { get; set; }

    public string Value { get; set; }
}

public class Theme
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required]
    public string Name { get; set; }
}

public class BlockHolder
{

    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required]
    public string Name { get; set; }

    [ForeignKey("Theme")]
    public int ThemeId { get; set; }
    public Theme Theme { get; set; }

}

public class TextBlock
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
```

```
    public int Id { get; set; }

    [Required]
    public string Name { get; set; }

    public string Content { get; set; }
}

public class Holder_Block
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [ForeignKey("BlockHolder")]
    public int BlockHolderId { get; set; }
    public BlockHolder BlockHolder { get; set; }

    [ForeignKey("TextBlock")]
    public int TextBlockId { get; set; }
    public TextBlock TextBlock { get; set; }

    public int Order { get; set; }
}

public class Image
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required]
    public string Name { get; set; }

    [ForeignKey("ImageData")]
    [Required]
    public int ImageDataId { get; set; }
    public ImageData ImageData { get; set; }
}

public class ImageData
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int Id { get; set; }

    [Required]
    public byte[] Data { get; set; }
}
```

B. Optický disk

Na přiloženém optickém disku se nachází zdrojové kódy redakčního systému JasperSite a jeho produkční verze připravená k nasazení do provozu.