Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Webový informační systém pro recenze filmů, divadelních her a dalších uměleckých představení

Bc. Jakub Kolebaba

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jakub Kolebaba

Informatika

Název práce

Webový informační systém pro recenze filmů, divadelních her a dalších uměleckých představení

Název anglicky

Web information system on movie, theatre, and other art perfomances reviews

---

Cíle práce

Cílem práce je navrhnout a vytvořit internetový deník ve formě webové aplikace dle předem stanoveného písemného zadání od zástupce České novinářské obce. Deník bude přinášet informace o kulturním dění v Čechách a na Moravě jako jsou například recenze a pozvánky na kulturní představení. Webová aplikace bude poskytovat několik úrovní přístupu dle rolí v redakci a také přehledy o návštěvnosti jednotlivých článků. Základní úroveň přístupu bude zdarma a bude zobrazovat samotné recenze. Vyšší úrovně přístupu budou placené a navíc budou umožňovat odbornou diskusi.

Metodika

Budou dodržovány standardy softwarového inženýrství a UML. V první části bude vypracována literární rešerše o dostupných a použitých technologiích a teoriích. V druhé části práce bude projektová dokumentace samotného díla.

Doporučený rozsah práce

60 – 120 stran

Klíčová slova

UML, Javascript, NodeJS, MongoDB, Webová aplikace

Doporučené zdroje informací

HAVERBEKE, Marijn, [2019]. Eloquent JavaScript: a modern introduction to programming. Third edition. San Francisco: No Starch Press. ISBN 978-159-3279-509.

KANISOVÁ, Hana a Miroslav MÜLLER, 2006. UML srozumitelně. 2., aktualiz. vyd. Brno: Computer Press. ISBN 8025110834.

MACCAW, Alex, c2011. JavaScript Web Applications. Sebastopol: O'Reilly. ISBN 978-1-449-30351-8.

Předběžný termín obhajoby

2018/19 LS – PEF

Vedoucí práce

doc. Ing. Vojtěch Merunka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Konzultant

Dušan Spáčil, redaktor a spisovatel

Elektronicky schváleno dne 26. 2. 2019

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 26. 2. 2019

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 03. 03. 2019

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Webový informační systém pro recenze filmů, divadelních her a dalších uměleckých představení" 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 27.3.2019 _____

Poděkování

Rád bych touto cestou poděkoval vedoucímu diplomové práce doc. Ing. Vojtěchu Merunkovi, Ph.D. za odborné vedení a trpělivost při zpracovávání diplomové práce.

Webový informační systém pro recenze filmů, divadelních her a dalších uměleckých představení

Abstrakt

Tato diplomová práce se zabývá problematikou softwarového inženýrství a vývoje webových aplikací. V teoretické části jsou popsány jednotlivé fáze systematického přístupu k vývoji software v softwarovém inženýrství. Dále je popsána architektura webových aplikací a technologií, které jsou pro jejich tvorbu využívány. V praktické části je dle softwarového inženýrství systematicky analyzován, navržen a vyvinut systém pro recenze filmů, divadelních her a dalších uměleckých představení.

Klíčová slova: Softwarové inženýrství, UML, Webová aplikace, JavaScript, NodeJS, GraphQL, MongoDB

Web information system on movie, theatre, and other art performances reviews

Abstract

This master's thesis deals with software engineering and the development of web applications. Theoretical part elaborates on systematic approach for software development in software engineering and architecture of web applications and technologies used for their development. Practical part describes systematic analysis, design and development of web information system on movie, theatre, and other art performances reviews.

Keywords: Software engineering, UML, Web application, JavaScript, NodeJS, GraphQL, MongoDB

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Práce se zabývá softwarovým inženýrstvím a vývojem webových aplikací. V teoretické části jsou popsány základní pojmy softwarového inženýrství, webových aplikací a technologií, které jsou pro jejich návrh a vývoj využívány.

Praktická část práce aplikuje pojmy a technologie popsané v teoretické části na reálnou problematiku vývoje informačního systému pro recenze filmů, divadelních her a dalších uměleckých představení ve formě webové aplikace. Pro systém jsou nejprve stanoveny požadavky, které jsou analyzovány a dále využity pro návrh klíčových částí systému pomocí UML diagramů, drátěných modelů a také pro grafický návrh uživatelského rozhraní. Ve vývojové fázi jsou popsána implementační specifika klientské a serverové části systému včetně ukázky konkrétního řešení vybraných požadavků. Na závěr je popsána příprava a optimalizace systému pro nasazení na veřejný server a samotný proces nasazování.

Projekt, který je v rámci této diplomové práce zpracováván byl navržen a zpočátku zpracováván ve spolupráci s redaktorem a spisovatelem Dušanem Spáčilem. Dokončení projektu je ale realizováno pouze jako diplomová práce z důvodu odchodu hlavního sponzora projektu.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je vytvoření webového informačního systému pro recenze filmů, divadelních her a dalších uměleckých představení dle písemného zadání s ohledem na zásady a principy softwarového inženýrství a UML.

2.2 Metodika

Pro dosažení cílů vychází práce z teoretických východisek uvedených v její první části, ve které je stanoven pojem systémové inženýrství a přiblíženy jednotlivé fáze procesu vývoje software. Tyto informace jsou čerpány z dostupných zdrojů a jsou doplněny praktickými znalostmi a ukázkami.

Vlastní práce aplikuje metody softwarového inženýrství na tvorbu reálného webového informačního systému. Nejprve bude provedena analýza zadání a informací získaných prostřednictvím rozhovorů s redaktorem a spisovatelem Dušanem Spáčilem. Výsledky analýzy budou dále využity pro identifikaci uživatelských rolí, vytvoření návrhu systému pomocí UML diagramů a volbu vhodných technologií. Po samotném vývoji systému bude následovat jeho důkladné otestování připravenými testovacími scénáři, jednotkovými testy a testy výkonu. Výsledkem práce bude funkční systém nasazený na veřejný server a jeho dokumentace. V závěrečné diskuzi bude výsledný systém zhodnocen z hlediska dalšího vývoje a možných rozšíření.

3 Teoretická východiska

Teoretická východiska práce popisují problematiku vývoje software v softwarovém inženýrství včetně detailního popisu zvolených principů a technologií.

3.1 Softwarové inženýrství

Softwarové inženýrství je disciplína zabývající se metodami a nástroji pro systematický přístup k vývoji softwarových produktů a systémů (Tsui, 2018). Ian Somerville (Sommerville, 2013) tento systematický přístup nazývá softwarovým procesem a definuje pro něj čtyři základní aktivity:

- Specifikace vyvíjeného software a omezení jeho činnosti.
- Vývoj a návrh software.
- Validace software, zda odpovídá požadavkům zákazníka.
- Evoluce a úprava software dle nových požadavků a skutečností.

Tyto obecné aktivity lze v pak v závislosti na charakteru vyvíjeného software uspořádat různými způsoby a popsat na různých úrovních podrobnosti. Často jsou využívány fáze analýzy, návrhu, vývoje, validace, nasazení a evoluce.

3.1.1 Modely vývoje software

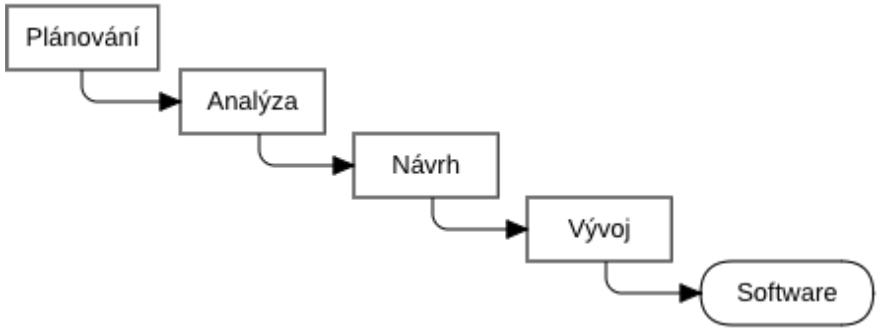
Modely vývoje software popisují celý životní cyklus, který začíná představou o software a končí, jakmile software zaniká nebo je jiným software nahrazen. Identifikací jednotlivých fází životního cyklu lze určit časové a finanční náklady a také dospět k úplné, nebo částečné automatizaci procesu. V praxi jsou tyto modely využívány pro podporu řízení projektů. Modelů vývoje software je celá řada a různé typy projektů vyžadují aplikaci odlišných modelů (Daoust, 2012).

3.1.1.1 Vodopádový model

Ve vodopádovém modelu (Obrázek 1) softwarového procesu jednotlivé fáze probíhají sekvenčně a není možné začít další fázi, pokud není současná fáze ukončena. Výsledkem každé fáze jsou rozsáhlé dokumenty, které je nutné před ukončením schválit. Výhodou tohoto přístupu je identifikace a analýza systémových požadavků před vývojem, díky čemuž dochází k minimálním změnám specifikace během vývoje. Dvěma hlavními

nevýhodami je nutnost kompletní specifikace před začátkem vývoje a dlouhá doba trvání mezi zadáním a doručením systému. Systém také může vyžadovat významné změny z důvodu změny prostředí (Dennis, 2012).

Obrázek 1 - Vodopádový model

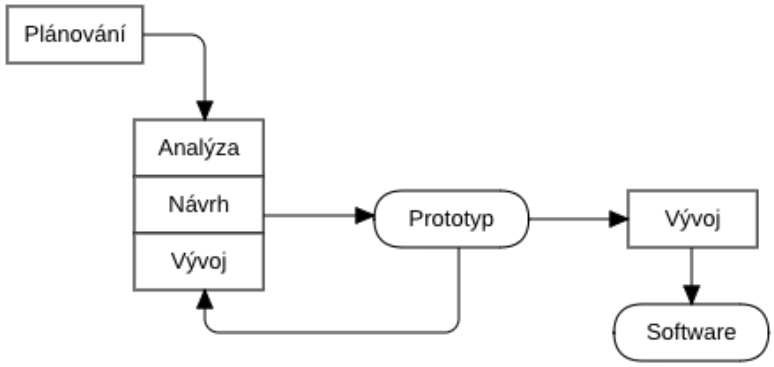


Zdroj: Dennis, 2012, vlastní zpracování

3.1.1.2 Prototypování

Modely založené na prototypování (Obrázek 2) provádí analýzu, návrh a vývoj souběžně. Všechny tyto fáze jsou prováděny opakovaně v cyklu, dokud není systém dokončený. Vývoj systému je zahájen základní analýzou a návrhem a produktem je jednoduchý program, který poskytuje minimální množství funkcí. Tento jednoduchý program, neboli prototyp, je prezentován zákazníkovi, který k němu vznese výhrady využité pro další analýzu, návrh a opětovný vývoj produktu. Cyklus analýzy, návrhu a vývoje pokračuje do doby, než je dosaženo shody se zákazníkem a teprve poté je nainstalován. Hlavní výhodou prototypového modelu je velmi rychlé poskytnutí funkčního produktu zákazníkovi, pro kterého je pak jednodušší systém pochopit, než když je specifikace provedena pouze na papíře (Dennis, 2012).

Obrázek 2 - Prototypový model



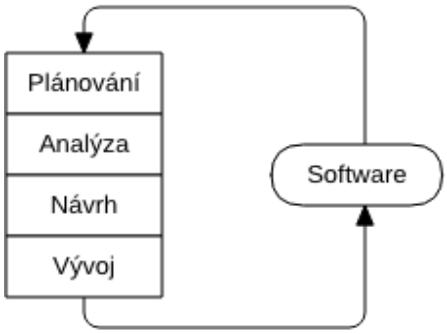
Zdroj: Dennis, 2012, vlastní zpracování

3.1.1.3 Agilní metody

Moderním způsobem vývoje software jsou agilní metody (Obrázek 3), založené na agilním manifestu. Nejvyšší prioritou manifestu je uspokojení zákazníka včasným a opakovaným dodáváním software a řešení problémů změnou požadavků místo vývoje s přesnými a detailními plány. Všechny tyto metody sledují několik principů agilního software (Dennis, 2012):

- Software je doručován rychle a neustále během jakékoli fáze.
- Změny požadavků jsou podporovány, i když se vyskytnou během vývoje.
- Zákazník a řešitel spolu úzce spolupracují na vyřešení problému.
- Jednotlivcům je důvěřováno jsou podporováni v řešení problémů, jsou jim poskytnuty nástroje, které potřebují.
- Komunikace mezi členy týmu.
- Hlavním ukazatelem postupu je fungující software.
- Vyhýbání se nadbytečné práci.
- Pravidelná zpětná vazba od členů týmu pro zlepšení procesů.
- Týmy, které se samy organizují vytvářejí nejlepší architektury a design.

Obrázek 3 - Agilní model vývoje software



Zdroj: Dennis, 2012, vlastní zpracování

3.1.2 Analýza

Analytická fáze vývoje software se zabývá především otázkami, kdo bude systém využívat a jaká bude jeho funkce. Prvním krokem analytické fáze je zkoumání současných systémů, identifikace příležitostí k rozvoji a vytvoření konceptuálního modelu nového systému. Druhým krokem je specifikace požadavků (Dennis, 2012). Předmětem analýzy jsou nejčastěji neznámé a nepřesně formulované systémy. Produktem je kromě konceptuálních modelů systému také specifikační dokument, který je úplným podkladem pro návrh funkčního řešení. Specifikační dokument obsahuje požadovaný výsledek, cílový stav a další důležité parametry jako je přínos a cena (Stephens, 2015).

Konceptuální modely slouží ke správnému pochopení vlastností systému. Ideálním modelem by byla realizace systému, kde by uživatel mohl chování systému ohodnotit, nicméně toto řešení je finančně náročné a v praxi nevyužitelné. Pro analýzu jsou proto využity jednodušší modely, popřípadě prototypování (Dennis, 2012).

Specifikace požadavků je klíčovou fází, kdy dochází ke kontaktu mezi zadavatelem a řešitelem. Požadavky mohou být specifikovány neformálně v přirozeném jazyce a být tak srozumitelné všem uživatelům, manažerům a řešitelům. Pokud neformálně specifikované požadavky systém dostatečně přesně nepopisují, lze je využít jako podklad pro vypracování podrobné specifikace. Během vývojové fáze životního cyklu je pak specifikace využita pro plánování a také pro validaci, zda se vytvořený software shoduje s požadavky zadavatele (Stephens, 2015).

3.1.3 Návrh

Fáze návrhu slouží k převedení výsledků analýzy do modelů na technologické úrovni s detailnějšími schématy, které dále slouží jako podklad pro vývoj. Návrh systému zpřesňuje fungování systému v oblasti infrastruktury, uživatelského rozhraní, formulářů a výkazů které by měl systém poskytovat. Návrh by měl být přesný, v souladu s požadavky, rozpočtem a řešit specifikované problémy (Subramanian, 2015). Systém lze navrhovat na několika úrovních:

- **Návrh architektury –** Prvním krokem je ujasnění, zda bude systém vyvíjen interně, outsourcován[1] nebo zakoupen. Tento krok vede k základnímu návrhu architektury systému, která popisuje využitý hardware, software a síťovou infrastrukturu.
- **Návrh uživatelského rozhraní –** Návrh uživatelského rozhraní specifikuje, jak se budou uživatelé v systému pohybovat. Pro tento návrh jsou často využity drátěné modely, které jednoduše popisují rozložení ovládacích prvků na obrazovce.
- **Datový model –** Popisuje strukturu ukládaných dat v databázi.

Na konci fáze návrhu jsou všechny jeho části předány vývojovému a analytickému týmu, který přezkoumá analýzu a společně s vedením pak rozhodne, zda v projektu pokračovat či nikoli.

3.1.4 Vývoj

Hlavním produktem softwarového inženýrství je fungující program. Vývoj je fáze, kdy dochází k převádění technologických modelů z předchozích fází do algoritmů zvoleného programovacího jazyka. Psaní programového kódu ale není jedinou aktivitou vývoje. Kód musí být také laděn a také kompilován do spustitelné formy. Často je zapotřebí zavést verzovací systém pro sledování změn ve zdrojovém kódu software. Vytvoření kvalitního software se při fázi vývoje řídí několika charakteristikami dobrého programového kódu (Tsui, 2017):

- **Čitelnost** – Snadná čitelnost ostatními programátory.
- **Udržovatelnost** – Snadná změna a správa kódu.

[1] Outsourcing je vyčleňování činností externí firmě, která se na danou činnost specializuje.

- **Výkon** – Kód by měl být vykonáván co možná nejrychleji.
- **Správnost** – Kód by měl řešit požadovaný problém.
- **Úplnost** – Všechny systémové požadavky jsou splněny.

Charakteristiky jsou mezi sebou zaměnitelné, například zvýšení čitelnosti kódu zvýší i udržovatelnost a obě tyto charakteristiky přispívají ke správnosti (Tsui, 2017).

3.1.5 Validace

Cílem validace software je ověření, zda odpovídá definovaným požadavkům. Zahrnuje testování při vývoji (jednotkové, integrační testovaní) a také akceptační testování, kdy je software testován zákazníkem (Tsui, 2017). Testování při vývoji přináší výhody ve zkvalitnění vyvíjeného software a také zkrácení času, který by byl nutný na nalezení a opravu chyb v produkčním prostředí. Testy jsou výhodné i pro vývojáře, kterým mohou sloužit jako příklad použití určité části kódu (Hahn, 2013).

Jednotkový test je část kódu, která využívá a ověřuje správnost jiné části kódu. Každý jednotkový test by měl být automatizovaný, opakovatelný, snadno pochopitelný a snadno spustitelný (Saleh, 2013). Výhodou jednotkových testů je rychlé zachycení a oprava chyb. Každá funkční část software by ideálně měla být pokryta jednotkovým testem. Jakmile jsou jednotlivé části kódu pokryty jednotkovými testy, je třeba mezi těmito částmi vytvořit vazby. Testování těchto vazeb a správnosti komunikace mezi nimi se nazývá integrační testování. Integrační testy se typicky zaměřují na nově přidaný kód a kontrolu, zda s dalšími metodami komunikuje správně. Systémy lze dále testovat například z hlediska přístupnosti, výkonu a bezpečnosti (Tsui 2017; Stephens, 2015).

3.1.6 Nasazení

Nasazení zahrnuje vydání a instalaci vyvinutého software do produkčního prostředí, kde bude dostupný koncovému uživateli (Subramanian, 2015). Instalace je proces, kdy je starý systém nahrazen za nový. Pro tento proces existuje několik strategií – souběžná, pilotní, postupná a nárazová. Při souběžné strategii současný systém zůstává v provozu a současně je instalován nový systém. Jakmile nový systém pracuje spolehlivě a jsou s ním zaměstnanci seznámeni, dojde k přechodu na nový systém. Tato strategie je bezpečná, ale náročná na zaměstnance, kteří v krátkém časovém úseku musejí pracovat s oběma systémy zároveň. Dalším typem je pilotní strategie, při které je nový systém instalován pouze

v jedné organizační jednotce a po ověření následně zaveden v celé společnosti. Pro tuto strategii je nutné vybrat takovou organizační jednotku, která je poměrně náročná a lze na ní ověřit co nejvíce problémových oblastí. Postupná strategie je využívána především u rozsáhlých informačních systémů. Nejprve jsou zavedeny důležité části, na kterých je zbytek systému závislý, a poté postupně zaváděny ostatní části. Důraz je kladen na správné naplánování této strategie z důvodu časové náročnosti. Nárazová strategie spočívá v okamžitém ukončení současného systému a k okamžité instalaci nového. Tento způsob je náročný na přípravu a organizační opatření. Jakmile je fáze nasazení ukončena, následuje fáze evoluce, kdy je systém dále rozvíjen a upravován, aby reflektoval změny vyžadované uživatelem a případné změny v legislativě (Mallach, 2009).

3.2 Unified Modelling Language

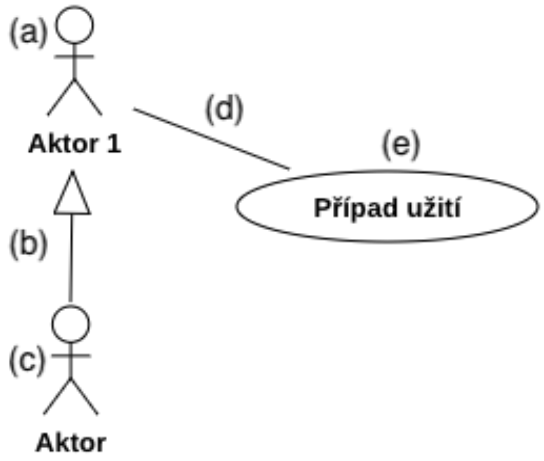
Unified Modelling Language (UML) je standard obsahující řadu technik pro tvorbu diagramů a poskytující grafickou reprezentaci vhodnou k modelování jakéhokoli systému. Motivací pro vznik tohoto standardu bylo mnoho nestandardizovaných technik (Daoust, 2012). UML ve verzi 2.4 popisuje sadu čtrnácti diagramů a technik pro modelování systémů. Rozděleny jsou do dvou hlavních skupin – strukturální a behaviorální. Pomocí strukturálních diagramů lze reprezentovat data a jejich vztahy v systémech. Obsahují například diagramy třídy, objektů, balíčku a nasazení. Behaviorální diagramy jsou určeny pro modelování dynamických vztahů mezi instancemi objektů a také pro modelování chování jednotlivých objektů během jejich existence. Pro analytiky jsou podporou při modelování funkčních požadavků systému. Obsahují diagramy aktivit a případů užití, dále sekvenční, komunikační, interakční a stavové diagramy (Dennis, 2012). V této kapitole jsou představeny vybrané UML diagramy.

3.2.1 Diagram případů užití

Diagram případů užití zachycuje vnější pohled na chování systému z hlediska uživatele. Základní komponenty diagramu jsou zobrazeny na obrázku (Obrázek 4). Aktor systému, který je zobrazen na části (a) a (c) reprezentuje uživatele, popřípadě vnější systém, který se systémem interaguje. Mezi jednotlivými případy užití – na obrázku část (e) – a aktory jsou pomocí asociačních spojení vytvářeny vztahy – na obrázku část (d) – určující, že se aktor na daném případu užití účastní. Posledním zobrazeným vztahem na obrázku je vztah

zobecnění – na obrázku část (b) – pomocí kterého lze určit generalizaci a specializaci objektů (Daoust, 2012).

Obrázek 4 - Komponenty diagramu případů užití



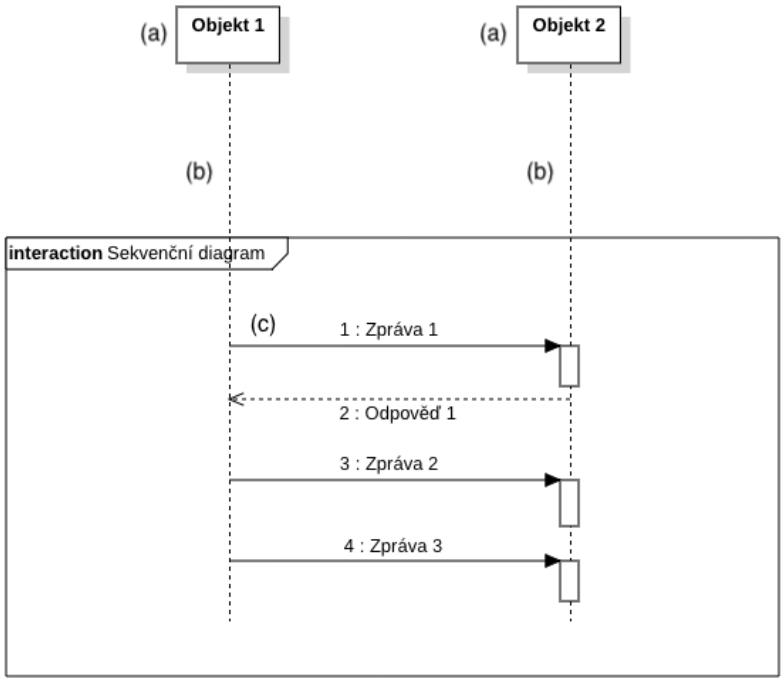
Zdroj: vlastní zpracování

Diagram případu užití lze vytvořit různými způsoby podle hlediska zaměření. Při systémovém zaměření jsou případy užití modelovány jako systémové funkce. Business zaměření obsahuje aktora vně podniku (zákazník, dodavatel), který interaguje s podnikem pro dosažení cíle využitím podnikových procesů. Komponentové zaměření případů užití nachází využití při dokumentaci požadavků na určitou software komponentu. Aktoři jsou v tomto případě dalšími komponentami systému, popřípadě vnější systémy. Výhodou diagramu případů užití je rychlé získání hrubého přehledu o funkcích systému ale poskytují také možnost získat bližší informace studiem scénářů pro jednotlivé případy užití (Daoust, 2012).

3.2.2 Sekvenční diagram

Sekvenční diagramy jsou využívány pro ilustraci toku času a výměny zpráv scénáře případu užití. Na obrázku (Obrázek 5) jsou jednotlivé objekty účastnící se na výměně zpráv zobrazeny v horní části diagramu – na obrázku část (a) – běh času je pak znázorněn vertikální čárou (lifeline) – na obrázku část (b) – a samotné výměny zpráv jako šipky mezi těmito čárami – na obrázku část (c) (Daoust, 2012).

Obrázek 5 - Sekvenční diagram

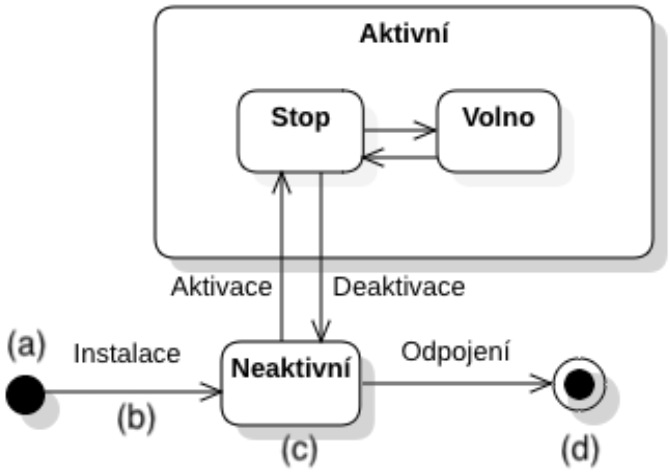


Zdroj: vlastní zpracování

3.2.3 Stavový diagram

Stavový diagram popisuje životní cyklus jedné položky, typicky instance třídy. Diagram (Obrázek 6) obsahuje všechny možné stavy – na obrázku část (c) – dané položky a přechody mezi nimi – na obrázku část (b). Dalšími důležitými komponentami stavového diagramu je počáteční stav – na obrázku část (a) – a koncový stav – na obrázku část (d). Mohou být využívány pro ilustraci funkčních požadavků (Daoust, 2012).

Obrázek 6 - Stavový diagram



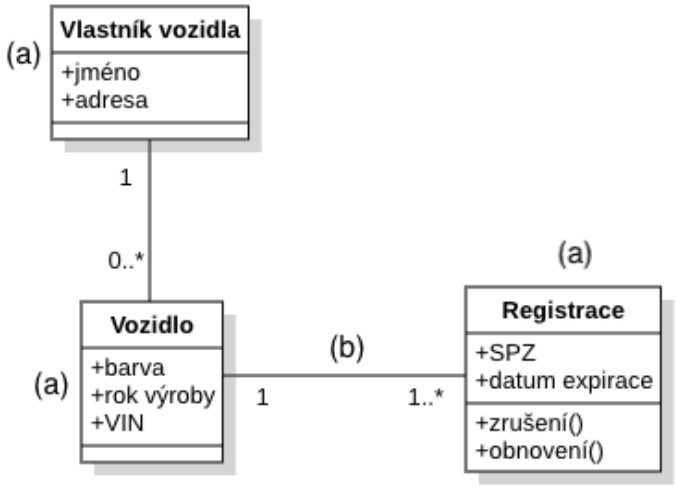
Zdroj: Daoust, 2012, vlastní zpracování

Příkladem pro využití stavového diagramu je modelování stavů semaforu přechodu pro chodce. Po instalaci a aktivaci semaforu zapojením do elektrické sítě se semafor může nacházet pouze ve stavu stop nebo volno dokud nedojde k jeho opětovné deaktivaci a odstranění (Daoust, 2012).

3.2.4 Diagram tříd

Diagramy tříd zobrazují relevantní entity systému (lidé, místa, události apod.), vztahy mezi nimi a operace, které na nich mohou být provedeny. Na diagramu (Obrázek 7) jsou obsaženy třídy – na obrázku část (a) – které se skládají ze tří polí – jména, atributů a operací. Atributy a operace nejsou vyžadovány, přesto jsou často pro přehlednost uváděny jako prázdná pole. Vztahy mezi třídami – na obrázku část (b) – obsahují mimo jiné i kardinalitu (Daoust, 2012).

Obrázek 7 - Diagram tříd



Zdroj: Daoust, 2012, vlastní zpracování

3.3 Architektura webových aplikací

Webové aplikace jsou aplikace typu klient-server, ke kterým je přistupováno prostřednictvím internetu, případně intranetu. Klientskou částí je ve webové aplikaci webový prohlížeč, serverem počítač, který poskytuje přístup k datům. Webová aplikace dokáže uživateli podle parametrů v požadavku zobrazovat dynamický obsah. Využívány jsou například jako e-shopy, informační systémy a podnikové intranety (Shklar, 2009).

3.3.1 Rozhraní

Webové servery a prohlížeče spolu komunikují výměnou HTTP zpráv. Servery obdrží a interpretují přijaté zprávy, načtou požadované zdroje, vygenerují odpověď a odešlou ji zpět. Protokol HTTP garantuje spolehlivé doručení dat, tedy že nebudou poškozena, zkreslena nebo zničena. Každá zpráva odesílaná tímto protokolem musí obsahovat metodu (Tabulka 1), která určuje, jaká akce má být na serveru provedena (Gourley, 2002).

Tabulka 1 Vybrané HTTP metody

METODA	POPIS
GET	Získání zdroje ze serveru.
DELETE	Smazání zdroje na serveru
POST	Odeslání dat z klienta na server, typicky při vytváření nového záznamu v databázi.
HEAD	Získání hlavičky daného zdroje.

Zdroj: vlastní zpracování

Zdrojem na webovém serveru může být mnoho různých datových typů, proto HTTP označuje každý zdroj přenášený po internetu identifikátorem datového formátu MIME. Původně byl MIME navržen pro přenos zpráv mezi různými emailovými klienty, nicméně dobré využití našel i v protokolu HTTP, který si MIME brzy adaptoval právě pro typování multimediálního obsahu. Webový prohlížeč zkoumá MIME typy u objektů, které ze serveru obdrží, aby dokázal rozhodnout, zda s objektem dokáže pracovat. Moderní prohlížeče podporují několik stovek typů, nejčastěji pro obrázky a HTML soubory. Každá odpověď HTTP protokolu obsahuje stavový kód (Tabulka 2). Stavový kód je trojciferný numerický kód, který klienta informuje, zda byl požadavek dokončen úspěšně, popřípadě o jakou chybu se jedná (Gourley, 2002).

Tabulka 2 - Vybrané HTTP kódy odpovědí

STAVOVÝ KÓD	POPIS
200	OK. Dokument byl úspěšně vrácen.
302	Přesměrování. Dokument byl přesunut na jiné umístění.
401	Vyžadována autentizace
404	Zdroj nebyl nalezen.
500	Chyba serveru

Zdroj: vlastní zpracování

Společně se stavovým kódem je také odesílána krátká textová zpráva, která chybu popisuje podrobněji (Gourley, 2002).

3.3.1.1 GraphQL

GraphQL je deklarativní dotazovací jazyk pro načítání dat a také serverová služba, která spouští dotazy podle definované datové struktury. Komunikace je typicky realizována prostřednictvím protokolu HTTP metodou POST. Služba je databázově nezávislá a pouze definuje atributy a jejich typy. Pro každý atribut pak musí na serveru existovat funkce pro získání jeho hodnoty. Podle Bankse (Banks, 2018) jsou dotazy v GraphQL hierarchické, silně typované a specifikované klientem. Silné typování je zajištěno typovým systémem obsaženým ve schématu služby, které dotazy a vstupní hodnoty validuje. Struktura dotazu je stejná jako struktura odpovědi, což klientům umožňuje dotazovat se pouze na taková data, která využijí.

GraphQL definuje tři typy operací – *query, mutation* a *subscribtion*, které popisují, jaký typ operace má být na serveru proveden. Operace typu *query* je obdobná jako metoda GET protokolu HTTP a slouží tedy k načítání zdrojů, případně odesílání parametrů, podle kterých má být zdroj vybrán. Tato metoda by ale neměla provádět žádné akce s vedlejšími účinky jako je například vkládání do databáze, úprava a mazání, ke kterým slouží operace *mutation.* Posledním typem operace je *subscription*, která umožňuje serveru odesílat data svým klientům, pokud nastane určitá událost. Výhodou implementace komunikace mezi klientem a serverem pomocí GraphQL je například menší zatěžování přenosového kanálu, protože klient se vždy dotazuje na data, která potřebuje. Problém, kdy odpověď serveru obsahuje nadbytečné informace se nazývá overfetching (Banks, 2018). Lze ho ilustrovat na příkladu načítání jména uživatele, ve kterém by tělo odpovědi HTTP serveru vypadalo následovně:

Obrázek 8 - Tělo odpovědi HTTP serveru

```
{
  "id": 1,
  "name": "John Doe",
  "username": "johndoe",
  "email": "john@doe.eu",
}
```

Zdroj: vlastní zpracování

Pokud ale klient využije pouze hodnotu atributu *name*, ostatní data zůstávají nevyužita a dochází právě k overfetchingu. Tento problém řeší GraphQL díky formulací dotazů, ve kterých je klient schopen specifikovat strukturu odpovědi (Obrázek 9).

Obrázek 9 - Dotaz na jméno uživatele v jazyce GraphQL

```
query {
  user(id: 1) {
    name
  }
}
```

Zdroj: vlastní zpracování

V dotazu klient volá operaci typu *query* pro načtení objektu typu *user*, konkrétně takového, jehož atribut *id* je roven *1*, jak je uvedeno v parametru dotazu. Tělo dotazu dále popisuje atributy jejichž hodnoty mají být načteny, zde je to tedy *name* (Obrázek 10).

Obrázek 10 - Odpověď GraphQL služby se jménem uživatele

```
{
  "data": {
    "user": {
      "name": "John Doe",
    }
  }
}
```

Zdroj: vlastní zpracování

Je zřejmé, že v tomto případě při využití GraphQL dochází ke zmenšení velikosti odpovědi, která obsahuje pouze data využitá klientem (Banks, 2018). GraphQL sice přináší určité výhody, nicméně čistá komunikace pomocí HTTP stále převyšuje výkonem, protože není nutné vyžívat další knihovny a dále se hodí do projektů, kdy je zapotřebí docílit nezávislého vývoje klienta a serveru a případně do serverově řízených aplikací (Bojinov, 2018).

3.3.2 Klientská část

Klientskou částí webových aplikací je webový prohlížeč. Pro definování základní struktury dokumentu a obsahu je využívána technologie HTML, vzhled dokumentů je definován v souborech CSS, které jsou do HTML vkládány pomocí odkazu v hlavičce, popřípadě jsou definovány přímo v HTML dokumentu. Třetí technologií využívanou pro vytváření aplikací ve webovém prohlížeči je JavaScript, který je určen pro definici chování jednotlivých prvků (Saternos, 2014).

3.3.2.1 JavaScript

JavaScript byl původně vyvinut pro prohlížeč Netscape Navigator, nicméně dnes je podporován ve většině webových prohlížečů. Jedná se o poměrně sofistikovaný objektově orientovaný programovací jazyk využívaný pro manipulaci s prezentační vrstvou, typicky vytvořenou v jazyce HTML (MacCaw, 2011). Technologie HTML sama o sobě neobsahuje mechanismus pro dynamickou změnu již vykreslené stránky, a tak každá elementární operace vyžaduje serverové zpracování a vykreslení nové stránky z odpovědi serveru. Postupem času proto byla do HTML zavedena podpora pro manipulaci s událostmi – kód, který je spuštěn, jakmile v prohlížeči nastane určitá událost jako klik myší, nebo přejetí přes prvek. Pro implementaci těchto manipulátorů je využíván právě JavaScript (Shklar, 2009). Rozšíření manipulátorů vedlo k vzniku přístupu k vývoji webových aplikací, který se nazývá SPA (Single Page Application). V tomto přístupu je vykreslena pouze jedna stránka a její obsah je pomocí JavaScriptu dynamicky překreslován. Výhodou tohoto přístupu je zlepšení uživatelského zážitku při prohlížení webové stránky, protože nedochází k přerušování opětovným načítáním stránky. Popularita SPA v posledních letech vzrostla především díky rozšíření JavaScriptu a vzniku knihoven, které vývoj usnadnily. Jednou z nejpoužívanějších knihoven pro vykreslování komponent a jejich aktualizaci při změně dat je ReactJS (Monteiro, 2014).

3.3.2.2 Typescript

JavaScript je často kritizován pro jeho dynamickou typovou kontrolu, která probíhá za běhu programu. Při vývoji v dynamicky typovaném jazyce není zapotřebí specifikovat datový typ při vytváření proměnné a mohou proto obsahovat hodnotu jakéhokoli typu. Nevýhodou tohoto přístupu je vyšší náchylnost k běhovým chybám. Tento problém řeší právě programovací jazyk TypeScript od společnosti Microsoft, který rozšiřuje JavaScript o statickou typovou kontrolu. Při kompilaci jsou datové typy ověřeny a případná chyba je programátorovi ihned indikována. Po úspěšné kompilaci je jazyk transpilován[2] do JavaScriptu očištěného o typy a typové definice jsou vygenerovány do samostatných souborů. Alternativou ke statickému typování pomocí TypeScriptu je knihovna Facebook

[2] Transpilace je proces překladu zdrojového kódu jednoho jazyka do zdrojového kódu jiného jazyka. Tento proces provádí programy nazývané transpilery.

Flow, která také doplňuje JavaScript o typové definice, nicméně neobsahuje transpiler tak jako TypeScript (Rauschmayer, 2015).

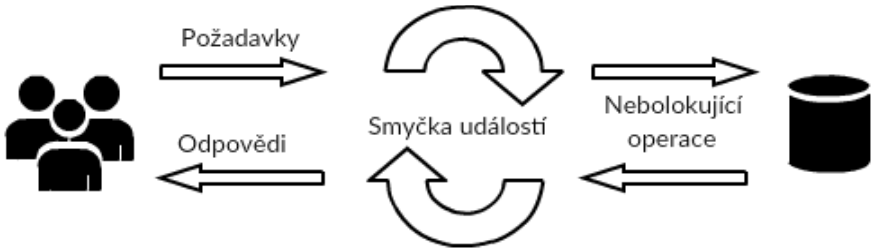
3.3.3 Serverová část

V této kapitole jsou popsány vybrané technologie využívané pro vývoj webových aplikací na straně serveru.

3.3.3.1 NodeJS

NodeJS je platforma pro vývoj webových aplikací, aplikačních serverů, síťových serverů a klientů postavená na Google V8 JavaScript engine s podporou spouštění JavaScript kódu mimo webový prohlížeč. Platforma rozšiřuje JavaScript o knihovny pro práci se soubory, volání systémových funkcí a o komunikaci po síti. NodeJS tvoří hlavní program, který je spouštěn v jednom vláknu jako nekonečná smyčka událostí, která komunikuje se souborovým systémem a databází a manipuluje se zpětným voláním. Tento model (Obrázek 11) umožňuje zpracování velkého množství dat na vstupu a výstupu, nicméně běh programu v jednom vláknu přináší i určité nevýhody, například při složitých výpočtech. Pokud je spuštěna úloha náročná na procesor, dochází k blokaci smyčky událostí a každý další požadavek bude zdržen ve frontě, dokud nebude blokující úloha dokončena (Herron, 2011; Vlăduțu, 2014).

Obrázek 11 - Architektura NodeJS



Zdroj: Herron, 2011, vlastní zpracování

Další výhodou je spojení s balíčkovacím systémem npm, standardně dodávaným s platformou NodeJS. Tento systém umožňuje NodeJS fungovat modulárně a vyžívat tak již existující kusy kódu. Sdílení a opětovné využívání kódu vytváří prostředí pro produktivní vývoj a rychlé řešení složitých úkolů pouhým zahrnutím modulu (Vlăduțu, 2014; Düüna, 2016).

3.3.3.2 MongoDB

MongoDB je jedna z nejrozšířenějších NoSQL databází. NoSQL jsou databáze s jiným ukládáním a zpracováním dat, než je tabulkové schéma relační databáze. Motivací k tomuto přístupu je jednoduchost návrhu, horizontální škálovatelnost, lepší kontrola dostupnosti a vysoká propustnost čtení a zápisu (Herron, 2011). Nelze obecně říci, zda je vhodnější NoSQL nebo relační databáze, protože oba tyto přístupy nacházejí využití v závislosti na řešeném problému (Banker, 2016).

MongoDB využívá flexibilní dokumentové schéma, kdy dokumenty ve stejné kolekci mohou mít různou strukturu, a intuitivní datový model ukládání dokumentů ve formátu JSON. Pro udržování vztahů mezi dokumenty je možné využít dva přístupy. Prvním přístupem je zanořování souvisejících dokumentů dovnitř rodičovského dokumentu. Výhodou tohoto přístupu je získání všech potřebný dat jedním dotazem do databáze. Nevýhodou je duplicita údajů, na které je vztahem odkazováno. Druhý přístup řeší duplicitu údajů ukládáním referencí na objekt, nicméně pro získání všech údajů je nutné provést několik dotazů do databáze (Edward, 2015).

3.3.4 Bezpečnost

Hlavní charakteristikou webových aplikací je jejich dostupnost odkudkoli, proto je nutné se zabývat jejich bezpečností. Vybrané útoky:

- **Cross Site Scripting (XSS)** – Nedostatečně zabezpečené vstupy umožní útočníkovi vložit do webové stránky skript za účelem přístupu k důvěrné informaci.
- **SQL Injection (SQi)** – Útočník využije bezpečnostní trhliny k podvržení a spuštění SQL dotazu díky čemuž může s důvěrnými informacemi manipulovat.
- **Cross-site request forgery (CSRF)** – Útočník využívá požadavky na URL adresy webové aplikace pro vykonání určité akce přihlášených uživatelů.

Základní ochranou proti výše uvedeným útokům je zabezpečení formulářů aplikace, nepovolení vložení skriptů do webové stránky, využití zabezpečeného přenosového protokolu a zabezpečení přenosu přihlašovacích údajů uživatelů.

3.3.4.1 Autentizace

Autentizace je proces ověřování identity uživatele. Ve webových aplikacích je autentizace typicky provedena dotazem na uživatelské jméno a heslo. Základním a také často

využívaným autentizačním protokolem je Basic Authentication, který byl původně uveden ve specifikaci HTTP/1.0. Protokol kombinuje uživatelské jméno a heslo oddělené dvojtečkou a tuto kombinaci zakóduje metodou base-64 (Boyd, 2012). Proces vytvoření autentizačních údajů v protokolu Basic Authentication lze provést ve čtyřech krocích (Tabulka 3).

Tabulka 3 - Proces Autentizace metodou Basic Authentication

KROK	POPIS	DATA
1	Zadání uživatelského jména a hesla	Uživatelské jméno: admin Heslo: 1234
2	Transformace zadaných údajů	admin:1234
3	Base-64 kódování	YWRtaW46MTIzNA==
4	Odeslání autentizačních údajů z klienta na server	Authorization: Basic YWRtaW46MTIzNA==

Zdroj: Boyd, 2012, vlastní zpracování

Problémem Basic Authentication je nízká úroveň zabezpečení, protože kódování base-64 je jednoduché dekódovat. Zásadním pokrokem v oblasti zabezpečení je OAuth 2.0. V současné době je to jedna mála metod, která je skoro 100 % spolehlivá. Její spolehlivost tkví ve vytváření unikátních tokenů pro každého uživatele. Pokud dojde ke kompromitaci tokenu, je smazán a vyžádán nový. OAuth v porovnání s Basic Authentication poskytuje vyšší úroveň zabezpečení, protože každý požadavek na přihlašovací údaje je prováděn pod protokolem SSL a také proto, že vygenerovaný token je pouze dočasný.

3.3.4.2 Autorizace

Autorizace slouží k ověření, zda má uživatel příslušná přístupová práva k provedení určité akce, jako je například čtení nebo zápis dokumentu. Autorizace je prováděna až po úspěšné autentizaci uživatele, aby bylo možné zjistit jaká práva uživatel má (Boyd, 2012).

3.3.4.3 Zabezpečení HTTPS

Zabezpečení samotného HTTP serveru v některých případech nemusí být dostatečné. Pokud komunikace se serverem obsahuje důvěrné informace, je vhodné spojení zabezpečit šifrováním, tedy protokolem HTTPS. Struktura zprávy je v protokolu HTTPS stejná, před odesláním ale dojde k zašifrování protokolem SSL a po přijetí zprávy serverem k dešifrování. SSL protokol podporuje širokou škálu algoritmů k ověření klienta a serveru, odesílání certifikátů a vytvoření bezpečnostních klíčů. Zabezpečení komunikace je velmi důležitou součástí bezpečnosti webových aplikací. Přihlášení uživatele do systému je nedostatečné, pokud je právě uživatelské jméno a heslo odesíláno nešifrovanou formou do internetu (Shklar, 2009).

3.3.5 SEO

SEO (podle anglického Search Engine Optimization) je metodologie pro získávání návštěvnosti webové stránky zvýšením viditelnosti pro webové vyhledávače. Jednou z možností je placená inzerce, dále pak zvyšování kvality samotné webové stránky. Měření kvality webové stránky lze shrnout do dvou hlavních faktorů – on-page a off-page. On-page faktory zahrnují všechny ovlivnitelné oblasti jako jsou klíčová slova, nadpisy, titulky, sémantiku a rychlost načítání stránky. Off-page faktory není možné ovlivnit, ale jsou vytvářeny komunitou. Jedná se například o zmiňování na sociálních sítích, v médiích a na fórech (Shenoy, 2016).

4 Vlastní práce

Vlastní práce popisuje proces vývoje webového informačního systému s ohledem na proces softwarového inženýrství, který byl popsán v teoretické části práce.

Nejprve byly definovány a analyzovány požadavky a výsledky analýzy použity pro návrh systému. Během samotného vývoje byl systém průběžně testován a nasazován na testovací prostředí na veřejném serveru. Výsledkem práce je funkční webový informační systém pro recenze filmů, divadelních her a dalších uměleckých představení.

4.1 Sběr a analýza požadavků

První částí softwarového procesu této práce byl sběr a analýza požadavků. Požadavky byly v první fázi získány v tištěné podobě od zadavatele, kterým byl redaktor a spisovatel Dušan Spáčil. Další požadavky a upřesnění zadání byly dodatečně získány na osobních schůzkách.

4.1.1 Specifikace

Zadavatel požaduje systém ve formě internetového deníku určený jak pro širokou veřejnost, tak i profesionálům, který přináší informace o kulturním dění v Čechách a na Moravě. Systém umožní vkládání článků, jejich úpravu a mazání. Každý článek se bude skládat z krátkého obsahu a obrázku (popřípadě videa), a bude možné ho sdílet na sociální síti. Články bude vytvářet asi 20 odborných recenzentů a po editačních úpravách publikovat dvou až pěti členná redakční rada. Registrovaní uživatelé, kteří registraci uhradí převodem na účet, budou moci články hodnotit, vést odbornou diskuzi a hodnotit příspěvky v diskuzi. Neregistrovaní uživatelé budou mít přístup ke článkům, hodnocení a komentářům pouze ke čtení.

4.1.2 Seznam požadavků

Analýzou neformální specifikace byly stanoveny funkční a nefunkční požadavky na systém (Tabulka 4). Každému funkčnímu požadavku byl přiřazen unikátní identifikátor a priorita na základě které byly tyto funkční požadavky dále implementovány. Nefunkčním požadavkem je návrh systému ve formě webové aplikace a využití již vytvořeného loga.

Tabulka 4 - Seznam požadavků

IDENTIFIKÁTOR	PRIORITA	POPIS
FR1	10	Vytvoření článku
FR2	10	Úprava článku
FR3	10	Smazání článku
FR4	9	Hodnocení článku
FR5	9	Diskuze ke článkům
FR6	5	Hodnocení komentářů v diskuzi
FR7	3	Sdílení článků na sociální síti
FR8	10	Publikování článků
FR9	3	Přehledy návštěvnosti
FR10	6	Vyhledávání článku
FR11	10	Registrace uživatelů
FR12	10	Přihlašování uživatelů
FR13	8	Nastavení data expirace členství registrovaným uživatelům
NR1	10	Systém ve formě webové aplikace
NR2	8	Využití stávajícího loga

Zdroj: vlastní zpracování

Škála priorit byla v seznamu požadavků nastavena na rozsah 1–10, kde 1 je nejméně důležitý požadavek a 10 požadavek nutný pro akceptaci výsledného systému. Ze zadání byly dále identifikovány uživatelské skupiny. Základní skupinou je nepřihlášený uživatel. Nepřihlášení uživatelé si mohou články zobrazit, filtrovat dle kategorií, vyhledávat v nich, a také zobrazit diskuzi ke článku. Přihlášení uživatelé mají přidělenou roli, podle které je jim umožněn přístup k různým funkcím systému.

- **Registrovaný** – U uživatelů s rolí „registrovaný“ je ukládáno datum expirace, což je datum, do kterého mají členství zaplaceno. Pokud jejich členství ještě nevypršeno, mají v systému možnost hodnotit články, diskutovat pod článkem a hodnotit komentáře v diskuzi.

- **Externista** – Externisté mají v systému právo pouze přidávat a upravovat své články.
- **Redakce** – Členové redakce mají přístup ke všem funkcí systému, kromě správy kategorií, jak je znázorněno tabulkou s požadavky na přístupová práva (Tabulka 5). Nejdůležitějším modulem pro redakci je modul pro správu článků, ve kterém mohou články schvalovat a popřípadě editovat.
- **Administrátor** – Administrátor má oproti redakci navíc přístup ke konfiguraci systému. Jedná se o roli s nejvyššími právy v systému.

Podrobná specifikace požadavků na přístupová práva k jednotlivým funkcím systému je znázorněna následující tabulkou (Tabulka 5).

Tabulka 5 - Požadavky na přístupová práva

	NEREGISTR.	REGISTR.	EXTERNISTA	REDAKCE	ADMIN
Zobrazení článků	✓	✓	✓	✓	✓
Zobrazení odborné diskuze	✓	✓	✓	✓	✓
Sdílení na soc. síti	✓	✓	✓	✓	✓
Vytvoření komentáře v diskuzi	×	✓	✓	✓	✓
Hodnocení článku a komentářů v diskuzi	×	✓	✓	✓	✓
Vytváření a úprava článku	×	×	✓	✓	✓
Publikování článku	×	×	×	✓	✓
Zobrazení přehledů	×	×	×	✓	✓
Správa uživatelů	×	×	×	✓	✓
Správa kategorií	×	×	×	×	✓

Zdroj: vlastní zpracování

4.1.3 Výběr technologií

Pro vývoj systému byla pro klientskou i serverovou část aplikace zvolena technologie JavaScript. Její výhodou je možnost vývoje na obou částech systému a také snadná integrace a komunikace s databázovým systémem MongoDB. Jako komunikační mezivrstva byla zvolena knihovna GraphQL, která kromě zjednodušení implementace zajišťuje i validaci vstupních dat. Na klientské části aplikace byla navíc využita knihovna ReactJS pro vytváření uživatelských rozhraní a také knihovna komponent Ant Design. Automatizace a nasazení systému bylo zajištěno cloudovou službou Bitbucket v kombinaci s aplikační platformou Heroku.

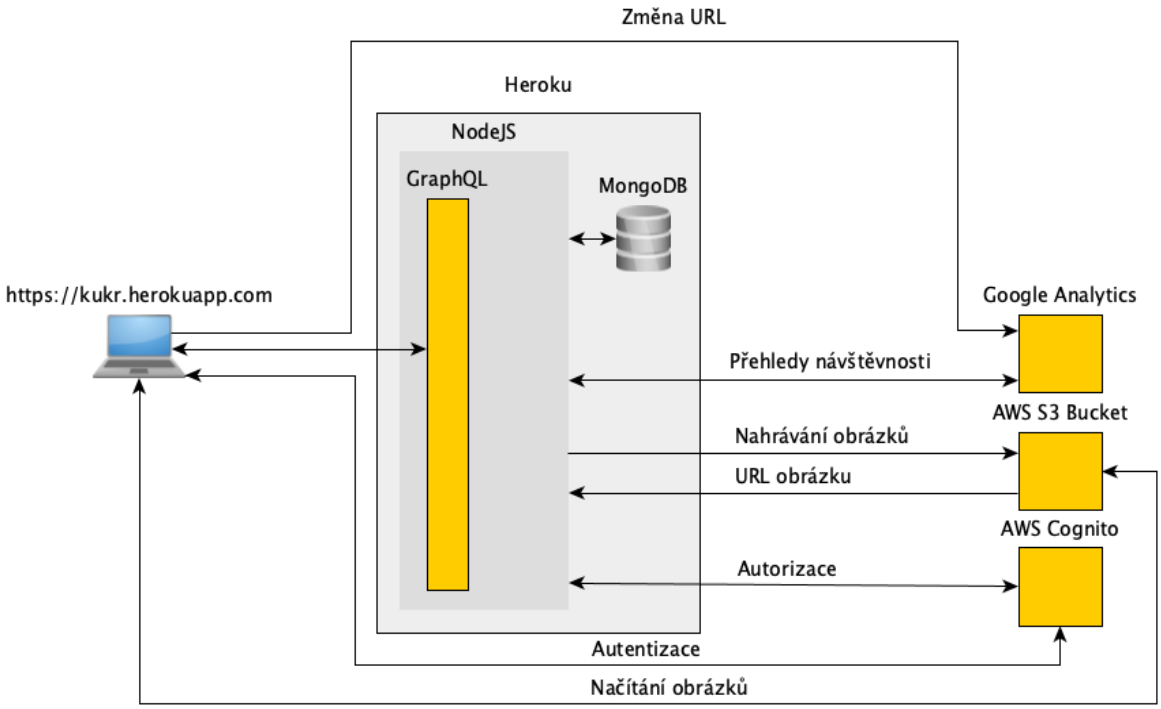
4.2 Návrh

Systém byl nejprve navržen z hlediska architektury, kdy byly modelovány jednotlivé komponenty systému, komunikace mezi nimi a vazby na externí služby. Požadavky na systém byly navrženy pomocí diagramu případů užití a funkční části systému sekvenčními a stavovými diagramy. Pro návrh uživatelského rozhraní byly využity drátěné modely a také grafický návrh.

4.2.1 Architektura systému

Architektura byla navržena s ohledem na podstatu aplikace, při které bylo zapotřebí brát ohled na SEO optimalizaci, pro kterou je důležité serverové vykreslování stránek. Pokud by stránky byly vykreslovány až na straně klienta, prvotní načtení stránky by obsahovalo pouze jednoduchou strukturu HTML bez obsahu a s vloženým JS souborem. Navržený přístup (Obrázek 12) je výhodný i pro uživatele webové aplikace z důvodu rychlejšího načtení při prvním přístupu na webovou stránku.

Obrázek 12 - Schéma architektury

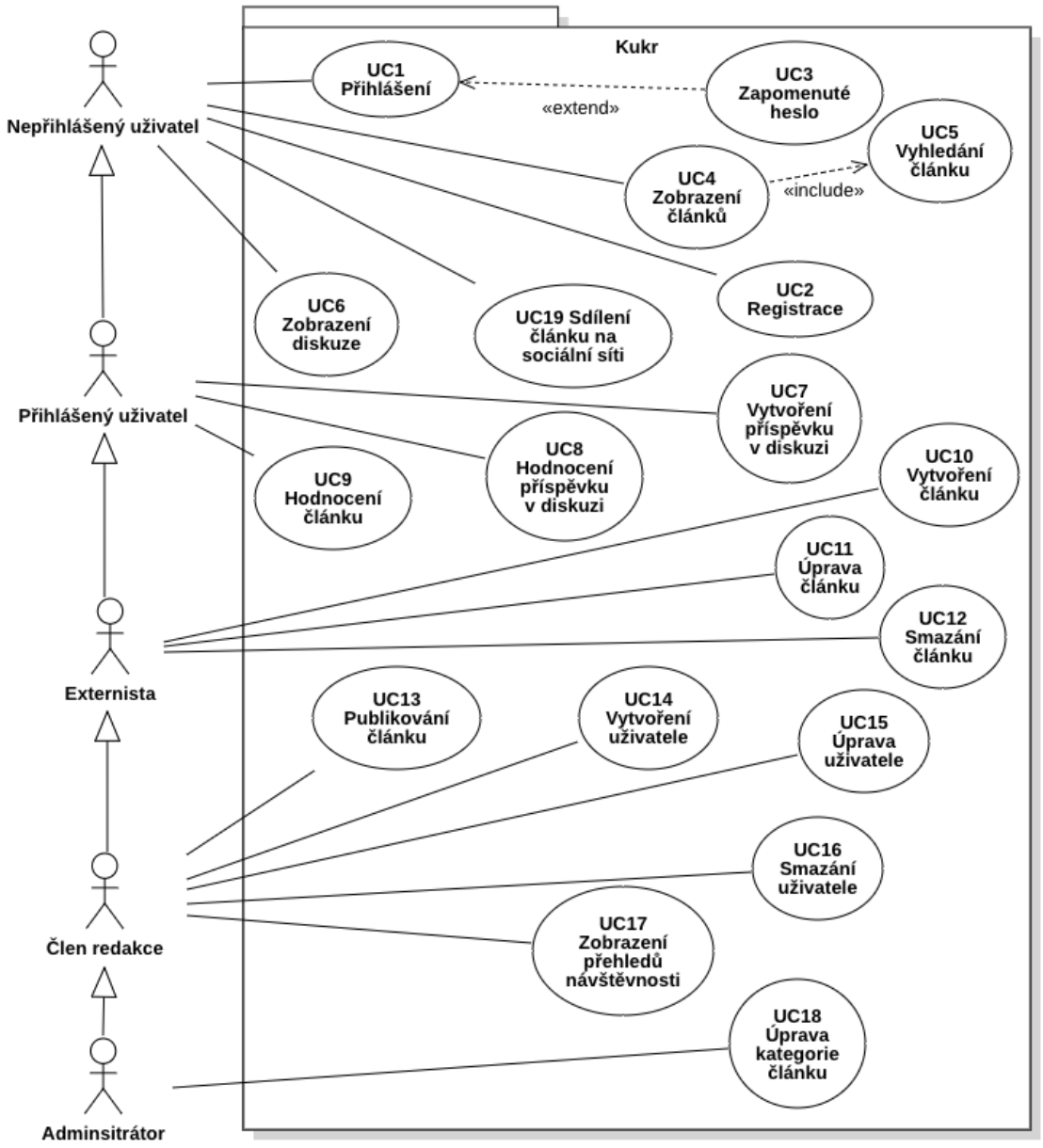


Zdroj: vlastní zpracování

4.2.2 Případy užití

Funkční požadavky na systém byly dále zpracovány a pro zpřehlednění znázorněny diagramem případů užití (Obrázek 13). V diagramu byla mezi aktory využita vazba generalizace znázorňující přístupová práva do systému pro jednotlivé uživatelské role. Vybrané případy užití byly dále detailně popsány pomocí scénářů.

Obrázek 13 - Diagram případů užití



Zdroj: vlastní zpracování

4.2.2.1 Scénáře případů užití

Prvním případem užití, pro který byl vytvořen scénář je registrace uživatele. Předpokladem tohoto scénáře je nepřihlášený uživatel (A), který se nachází na libovolné stránce. Reakce systému jsou znázorněny písmenem S. Scénář případu užití pro registraci uživatele (Tabulka 6) obsahuje ve čtvrtém kroku alternativní tok, kdy při vyplňování registračních údajů systém detekuje neunikátní uživatelské jméno nebo email, který není ve správném

formátu. Systém v tomto případě pokračuje krokem 4.1, ve kterém uživatele o této skutečnosti informuje.

Tabulka 6 - Scénář případu užití: Registrace a uživatele

KROK		POPIS		
1	A	Uživatel zvolí ovládací prvek „registrace“.		
2	S	Vygenerování formuláře pro registraci a jeho zobrazení v modálním okně.		
3	A	Vyplnění uživatelského jména, emailu a odeslání formuláře.		
4	S	Systém zkontroluje, zda je uživatelské jméno unikátní a email ve správné formě. Pokud ano, je uživateli odeslán email s informacemi k prvnímu přihlášení a modální okno s formulářem uzavřeno.		
		4.1	S	Pokud uživatelské jméno není unikátní nebo email není validní, systém tuto informaci uživateli zobrazí.

Zdroj: vlastní zpracování

Dalším případem užití je přihlášení registrovaného uživatele (Tabulka 7). Dva alternativní toky mohou nastat ve čtvrtém kroku. Prvním případem je detekce nesprávných přihlašovacích údajů systémem. Druhým případem je detekce prvního přihlášení uživatele, kdy systém uživateli zobrazí formulář pro nastavení nového hesla a tok dále pokračuje v tabulce kroky 4.2.1 a 4.2.2.

Tabulka 7 - Scénář případu užití: Přihlášení

KROK		POPIS				
1	A	Uživatel zvolí ovládací prvek „přihlášení“.				
2	S	Systém zobrazí přihlašovací formulář v modálním okně.				
3	A	Uživatel vyplní přihlašovací údaje.				
4	S	Systém ověří přihlašovací údaje				
		4.1	S	Pokud přihlašovací údaje nejsou validní, systém uživateli zobrazí zprávu o nesprávném uživatelském jménu / heslu.		
		4.2	S	Pokud systém detekuje první přihlášení, zobrazí uživateli formulář pro nastavení hesla obsahující dva vstupy – jeden pro heslo a druhý pro jeho potvrzení.		
				4.2.1	A	Uživatel vyplní nová hesla
				4.2.2	S	Systém uživateli nastaví nové heslo, provede přihlášení a informuje uživatele o úspěšném přihlášení.

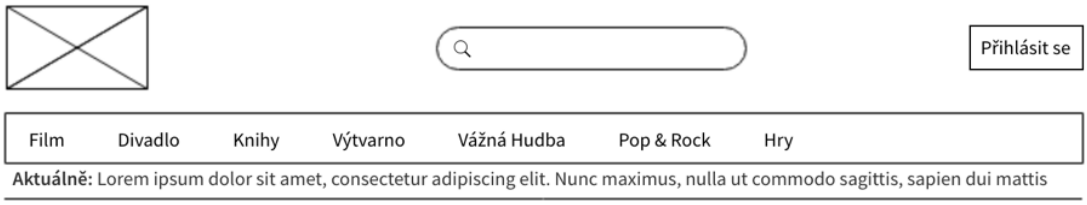
Zdroj: vlastní zpracování

4.2.3 Uživatelské rozhraní

Po stanovení případů užití a architektury systému byl vytvořen návrh uživatelského rozhraní. Návrh byl realizován v první fázi jako drátěné modely, které sloužily pro prezentaci rozložení jednotlivých stránek zadavateli. Po schválení drátěných modelů byl vytvořen návrh grafického vzhledu, který již obsahoval detailnější pohled na aplikaci i z hlediska barev.

Komponenta pro hlavičku (Obrázek 14) je taková část stránky, která se opakuje na všech stránkách přístupných bez přihlášení uživatele s příslušným oprávněním. Obsahuje logo, vyhledávací pole, tlačítko pro přihlášení a registraci a menu, ve kterém uživatel může vybrat zvolenou kategorii článků.

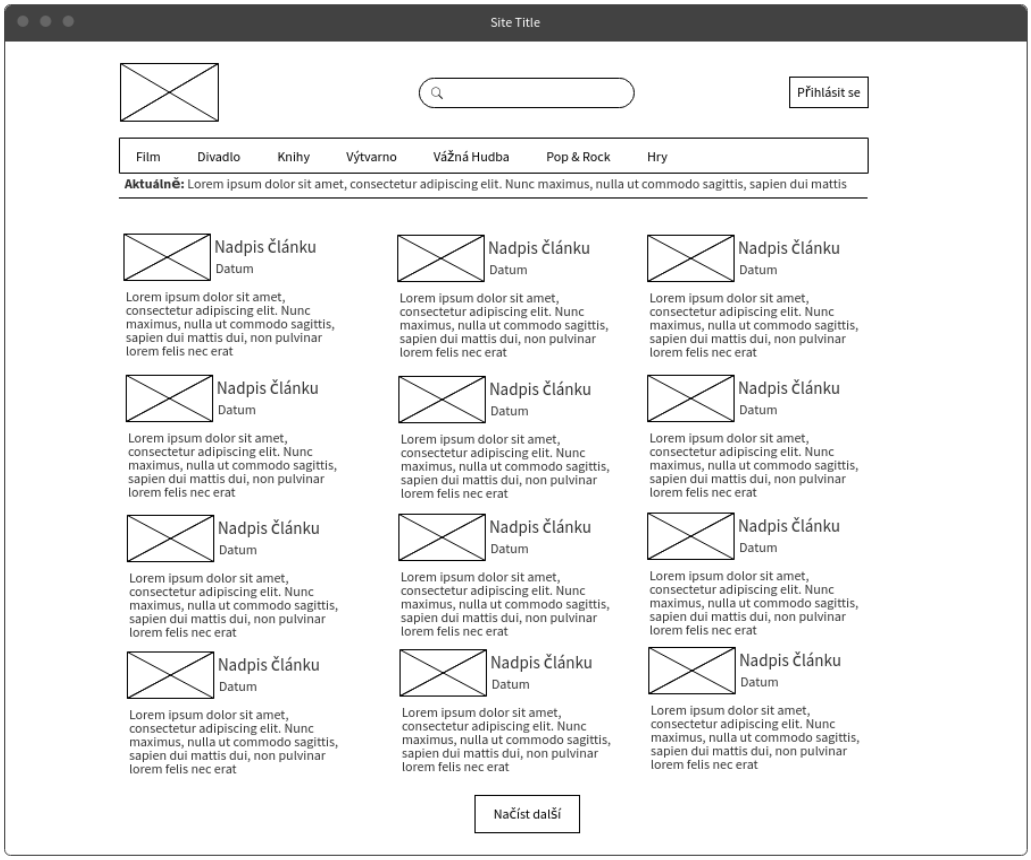
Obrázek 14 - Drátěný model: Hlavička



Zdroj: vlastní zpracování

Hlavní strana (Obrázek 15) obsahuje sloupce se třemi typy článků – Zákulisí, Recenze, Přehledy. Další články jsou načteny stisknutím tlačítka „Načíst další“ na konci stránky. Články jsou na hlavní stránce zobrazeny pouze se zmenšeným náhledem obrázku a s omezením na počet znaků textu.

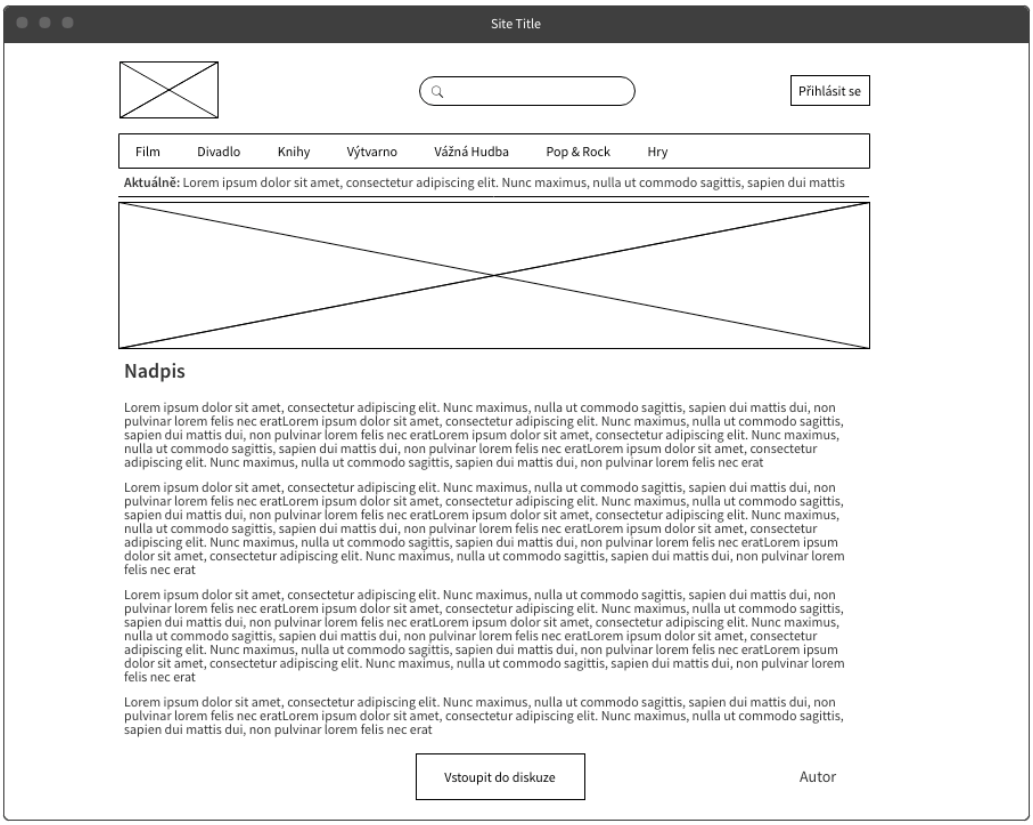
Obrázek 15 - Drátěný model: Hlavní strana



Zdroj: vlastní zpracování

Detail článku obsahuje obrázek v plné velikosti a text, který není redukován na počet znaků, a další aktivní prvky pro hodnocení článku a na konci stránky tlačítko pro vstup do diskuze (Obrázek 16).

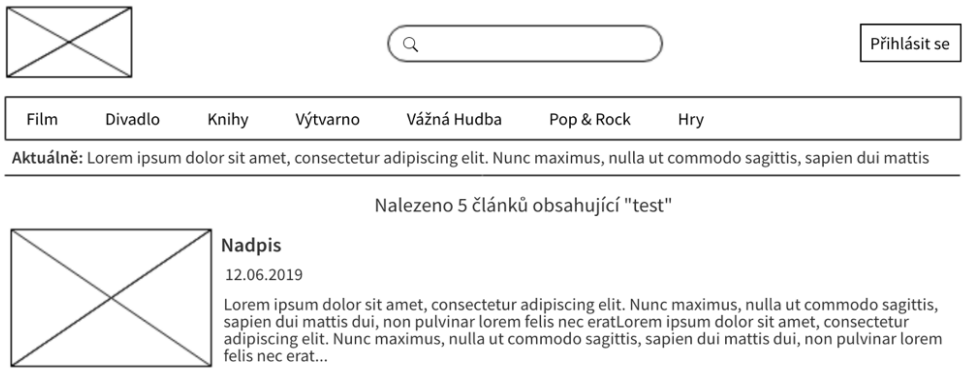
Obrázek 16 - Drátěný model: Detail článku



Zdroj: vlastní zpracování

Kategorie článků obsahuje výpis článků z jedné kategorie, které jsou vykresleny v jednom sloupci a náhled obrázku je větší než na hlavní straně. Stránka má obdobné rozložení jako stránky s výsledkem vyhledávání (Obrázek 17), liší se pouze v zobrazení informace o počtu nalezených článků, která na výpisu kategorie není k dispozici.

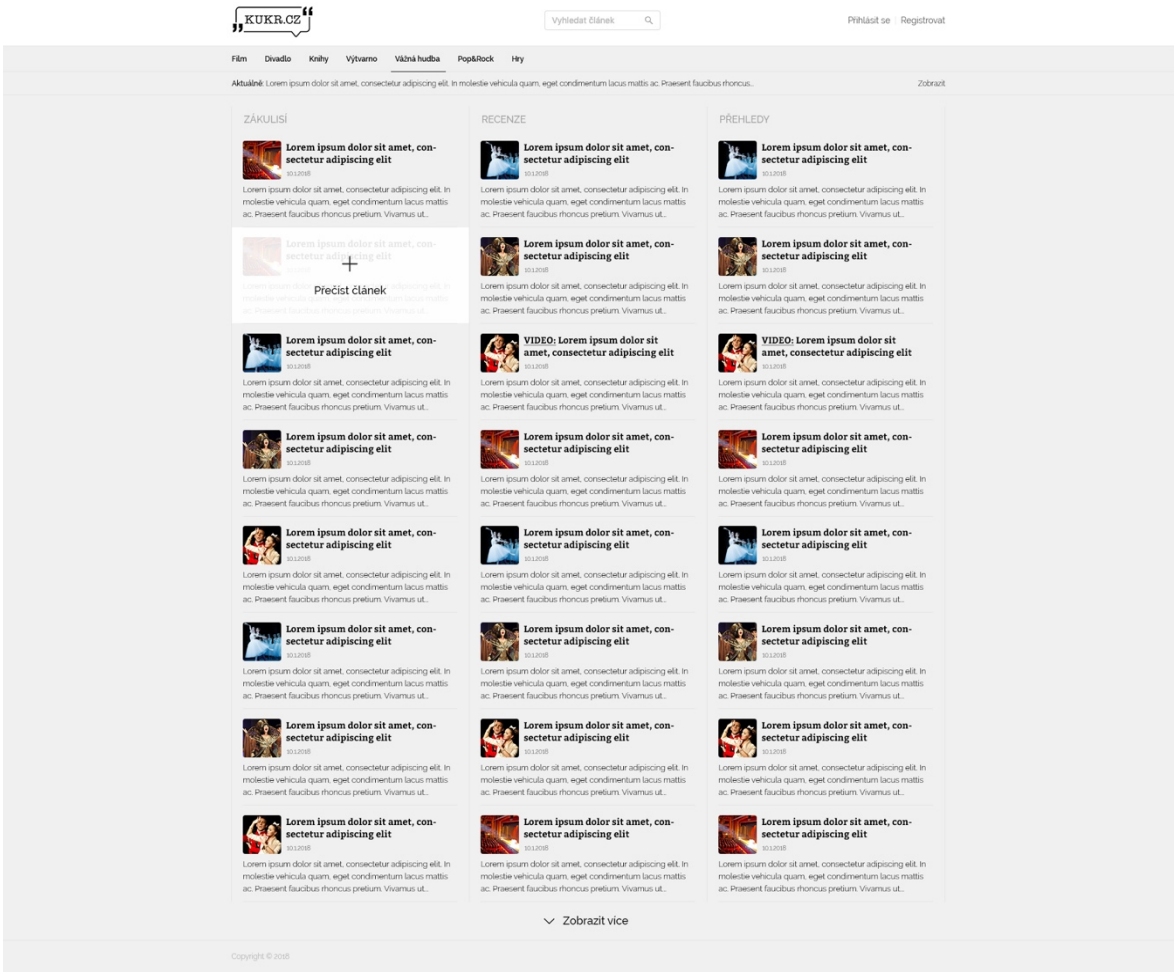
Obrázek 17 - Drátěný model: Výsledek vyhledávání



Zdroj: vlastní zpracování

Drátěné modely byly po schválení zadavatelem zpracovány do grafického návrhu (Obrázek 18). Při návrhu byl brán ohled na nefunkční požadavek na využití již vytvořeného loga.

Obrázek 18 - Grafický návrh



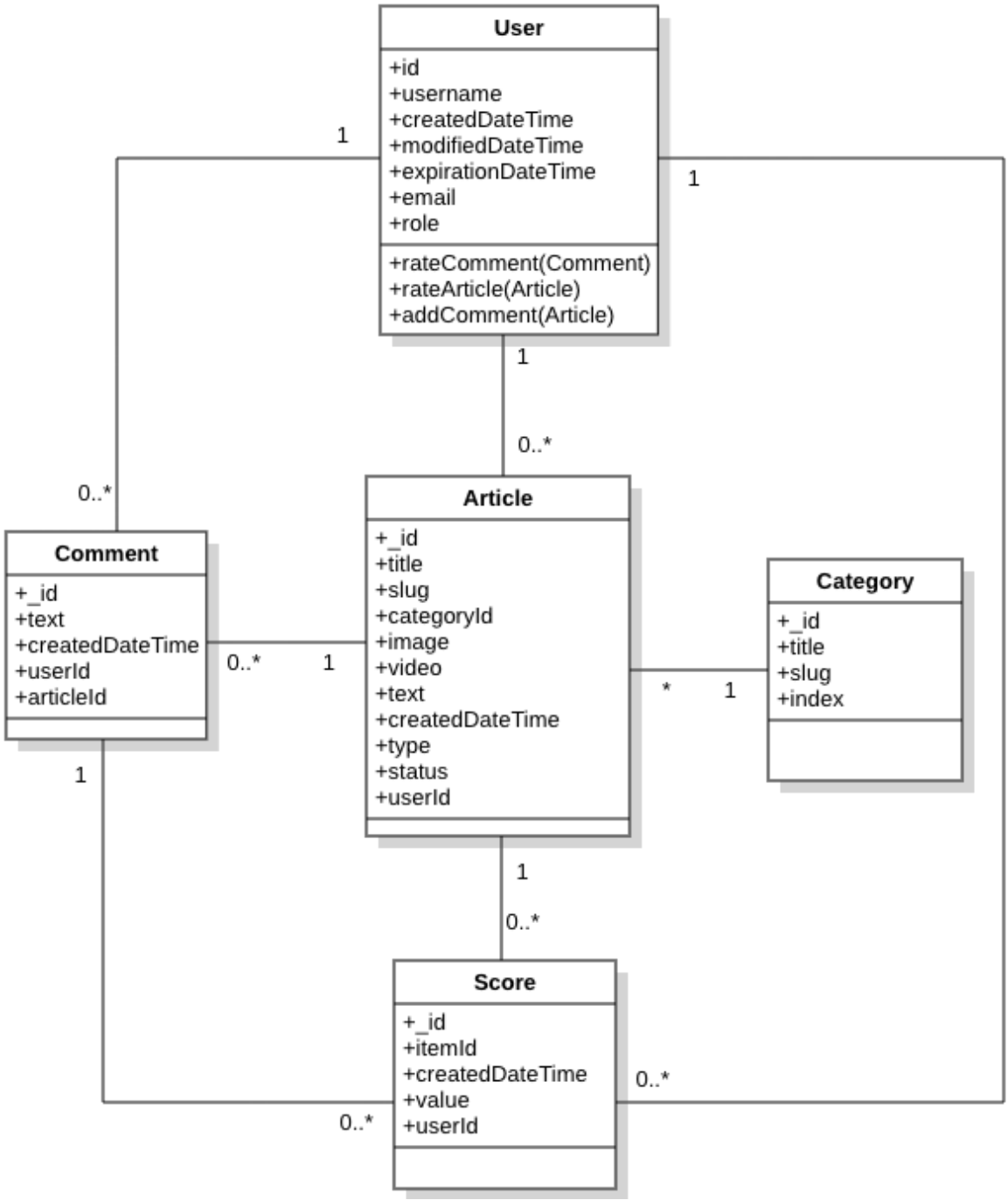
Zdroj: vlastní zpracování

Grafický návrh obsahuje výpis článků na hlavní straně dle dříve vytvořeného drátěného modelu a také náhled aktivního prvku, který je zobrazen jako překryvná vrstva s ikonou a nápisem „přečíst článek“ po najetí myší na libovolný článek.

4.2.4 Diagram tříd

Schéma objektové databáze MongoDB bylo znázorněno diagramem tříd (Obrázek 19). Článek popisuje třída Article s vazbou na uživatele, který článek vytvořil a kategorii článku. Každý článek musí být přiřazený do nějaké kategorie. Komentáře (třída Comment) obsahují vazbu na článek a na uživatele, který jej vytvořil. Hodnocení článků a komentářů je řešeno třídou Score, která kromě vazby na příslušný objekt uchovává i informace o hodnocení a uživateli, který hodnocení vytvořil.

Obrázek 19 - Diagram tříd



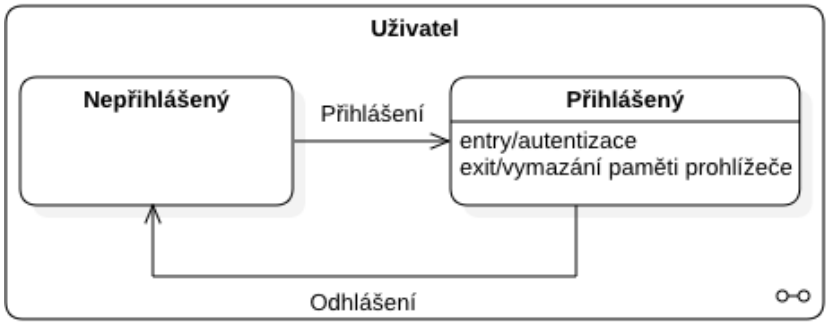
Zdroj: vlastní zpracování

4.2.5 Stavové diagramy

Některé objekty v systému mění svůj vnitřní stav. Prvním takovým objektem je uživatel, který udržuje informaci o přihlášeném uživateli (Obrázek 20). Informace o přihlášeném

uživateli jsou udržovány v paměti webového prohlížeče a vymazány, jakmile dojde k jeho odhlášení.

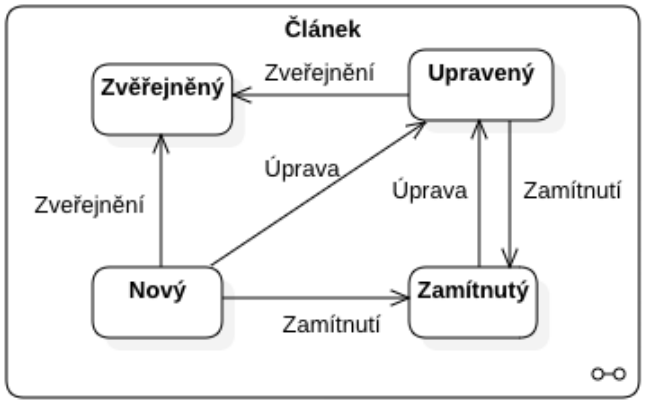
Obrázek 20 - Stavový diagram uživatele



Zdroj: vlastní zpracování

Dalším objektem s vnitřním stavem je článek (Obrázek 21), který se po jeho vytvoření nachází ve stavu „Nový“. Redakce může článek z tohoto stavu přesunout do stavů „Zamítnutý“ a „Zveřejněný“, popřípadě článek upravit. Úpravou se článek automaticky přesouvá do stavu „Upravený“. Jakmile je článek ve zveřejněném stavu, není dále možné měnit jeho stav.

Obrázek 21 - Stavový diagram článku



Zdroj: vlastní zpracování

4.3 Vývoj

V této kapitole jsou popsány nástroje a postupy, které byly využity pro vývoj systému a pro nastavení lokálního vývojového prostředí. Během vývoje byly dodržovány postupy

pro splnění určitých charakteristik dobrého programového kódu, které byly podrobněji popsány v kapitole 3.1.4.

- Snadná čitelnost a s tím související udržovatelnost byla zvýšena využitím softwarového nástroje Prettier, který po spuštění kód automaticky formátuje do předem předepsané podoby.
- Čitelnost a udržovatelnost byla dále zvýšena společně s výkonem pomocí nástroje TSLint pro statickou analýzu kódu, který kód kontroluje z funkčního hlediska.
- Určitého zlepšení bylo také dosaženo zvoleným programovacím jazykem, kterým byl na klientské i serverové straně TypeScript.

4.3.1 Lokální vývojové prostředí

Pro vývoj systému na lokálním prostředí byla zapotřebí instalace knihovny NodeJS. Tato knihovna obsahuje balíčkovací systém npm, pomocí kterého jsou instalovány další potřebné závislosti. Tento balíčkovací systém byl na lokálním prostředí nahrazen systémem yarn, který je oproti npm výkonnější. Dalším nutným předpokladem pro správný běh systému na lokálním prostředí je instalace a spuštění databáze MongoDB. Pro přístup k databázi byl využit nástroj Robo 3T. Ostatní závislosti jsou již obsaženy v konfiguračním souboru projektu a nainstalovány pomocí systému yarn.

4.3.2 Struktura projektu

Projekt se skládá ze dvou hlavních částí – serverové a klientské – které jsou spolu propojeny tak, že serverová část využívá komponent klientské části pro serverové vykreslování. Obě části projektu byly vyvinuty v programovacím jazyce TypeScript. Komunikační rozhraní mezi klientskou a serverovou částí bylo vytvořeno v GraphQL a dotazy optimalizovány, aby nebylo nutné při načítání seznamu dokumentů načítat všechny záznamy databáze. Na klientské části aplikace byla pro tyto účely vytvořena komponenta, která načte pouze pevný počet záznamů a další načítá až při požadavku uživatele na další načtení. Díky tomuto řešení nedochází k zahlcení přenosu ani k blokování uživatelského rozhraní vykreslováním velkého množství dat.

4.3.3 Serverová část

Serverová část projektu byla implementována v prostředí NodeJS ve spojení s Express Framework pro dosažení dostatečného výkonu aplikace a pro efektivní vývoj. Framework poskytuje zdroje na několika URL adresách:

Tabulka 8 - URL zdrojů webového serveru

URL	POPIS
/graphql	Spuštěná služba GraphQL
/public	URL pro načítání statických souborů, jako jsou například ikony.
*	Serverově vykreslené HTML obohacené o potřebná data a CSS.

Zdroj: vlastní zpracování

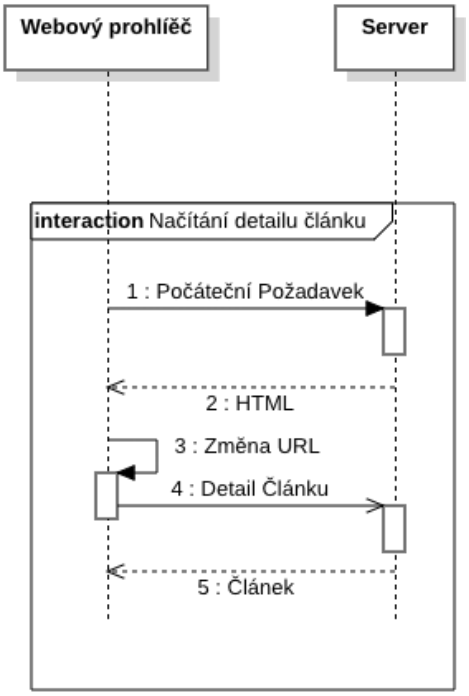
V posledním záznamu je uvedena URL jako *, což znamená, že server vrací HTML stránku na jakýkoli dotaz, jehož URL neodpovídá prvním dvěma záznamům. Tato konfigurace je zvolena právě z důvodu serverového vykreslování. Systém byl dále integrován s databází MongoDB prostřednictvím knihovny Mongoose, která poskytuje jednoduché rozhraní pro práci s uloženými dokumenty. Jako komunikační rozhraní pro odesílání a přijímání dat byla na serveru spuštěna a nakonfigurována služba GraphQL. Definice vlastností a jejich datových typů sloužila jako prostředek pro validaci vstupních dat.

4.3.4 Klientská část

Klientská část aplikace byla vyvíjena jako single page aplikace (SPA) se serverovým vykreslováním prvního načtení. Všechna logika je tedy umístěna na straně klienta a data jsou načítána asynchronně, nicméně při prvním dotazu je ze serveru odesílán celý HTML dokument obohacený o CSS a potřebná data. Tato architektura kombinuje výhody SPA a tradičního přístupu, kdy je pro každý další dotaz se serveru odesílán celý HTML dokument. Výhodou je rychlé načítání obsahu při přechodu mezi stránkami díky asynchronnímu načítání dat a překreslování jednotlivých komponent a také bezproblémová indexace roboty, protože při prvním načtení je HTML dokument vždy k dispozici celý bez

nutnosti spouštění JavaScript kódu. Obrázek (Obrázek 22) znázorňuje situaci, kdy uživatel při počátečním požadavku přistoupí na hlavní stránku a následně zobrazí některý ze článků. První odpověď serveru pak obsahuje celý HTML dokument, druhá odpověď už obsahuje pouze data daného článku.

Obrázek 22 - Diagram načítání detailu článku



Zdroj: vlastní zpracování

Samotné uživatelské rozhraní bylo vytvořeno ve frameworku ReactJS ve spojení s komponentami AntDesign. Komponenty byly v některých případech vizuálně upraveny pro dodržení grafického návrhu. Klientská část byla dále pro potřeby sledování návštěvnosti napojena na službu Google Analytics, na kterou odesílá informace o návštěvníkovi při každém přístupu a změně URL.

4.3.4.1 GraphQL

Pro komunikaci se službou GraphQL na straně serveru byla využita knihovna Apollo GraphQL, která po konfiguraci umožňuje jednoduché odesílání dotazů a ukládání odpovědí do mezipaměti. Pro vytvoření dotazu na server je nejprve nutné formulovat dotaz, který je pro načtení jednoho konkrétního článku vyobrazen na obrázku (Obrázek 23).

Obrázek 23 - GraphQL dotaz na článek

```
export const ARTICLE = gql`
  query article($slug: String!) {
    article(slug: $slug) {
      _id
      image
      title
      slug
      text
      type
      video
      category {
        _id
        title
        slug
      }
      averageScore
    }
  }
`;
```

Zdroj: vlastní zpracování

Dotaz obsahuje operaci typu *query* pro načtení objektu typu *article*. Parametrem je v tomto případě URL adresa daného článku, podle které je nalezen, vybrán z databáze a následně odeslán v odpovědi klientovi. Datový typ a povinnost parametru je nutné specifikovat už v dotazu a musí odpovídat nastavenému schématu serverové služby. Pokud hodnoty neodpovídají, dojde k selhání dotazu a vypsání chybového hlášení. Tento dotaz je dále využit v komponentě z knihovny Apollo (Obrázek 24), která samotný požadavek na GraphQL službu provede.

Obrázek 24 - Komponenta pro načítání a vykreslení článku

```
<Query query={ARTICLE} variables={{ slug: articleSlug }}>
  {({ loading, error, data }) => {
    if (loading) {
      return <Loader />;
    }

    if (error) {
      return <ErrorMessage message={error.message} />;
    }

    return (
      <React.Fragment>
        <Helmet title={data.article.title} />
        <Detail {...data.article} />
      </React.Fragment>
    );
  }}
</Query>
```

Zdroj: vlastní zpracování

Výše uvedená komponenta slouží k načtení dat ze serveru a vykreslení článku. Parametry pro nalezení článku je nutné nastavit do objektu *variables*, v tomto případě je tedy pouze nutné nastavit hodnotu *slug*. Odpovědí serveru může být několik:

- *Loading* je nastaven na hodnotu *true* – server stále na požadavek neodeslal odpověď.
- *Error* obsahuje chybové hlášení – při načítání dat nastala chyba.

Pokud jsou data načtena a odpověď neobsahuje chyby, je možné článek zobrazit k čemuž slouží komponenty Helmet (která nastavuje titulek stránky) a Detail (která zobrazuje samotný článek). Další komponenty jako například diskuze k článku, výpis kategorie a administrační rozhraní jsou řešeny obdobně, liší se pouze ve formulaci dotazů, hodnotou parametrů a komponentami, které jsou při úspěšném načtení dat ze služby GraphQL vykresleny.

4.3.5 Bezpečnost

Zabezpečení aplikace je především řešeno komunikací pouze pomocí protokolu HTTPS. Komunikaci protokolem HTTP bylo zamezeno automatickým přesměrováním na zabezpečenou verzi. Na serveru byla dále nastavena hlavička odpovědi X-Frame-Options na hodnotu *deny*, čímž je zamezeno případným zahrnutím stránky v rámci a s tím souvisejícími útoky. Přihlášení uživatele je realizováno službou Amazon Web Services

(dále jako AWS) Cognito, na kterou je aplikace napojena a která je využita pro správu uživatelů. Po úspěšné autentizaci je do webového prohlížeče uživatele uložen token, který je odesílán v hlavičce požadavků a na serveru dále využíván pro ověření uživatele k provedení zabezpečené akce.

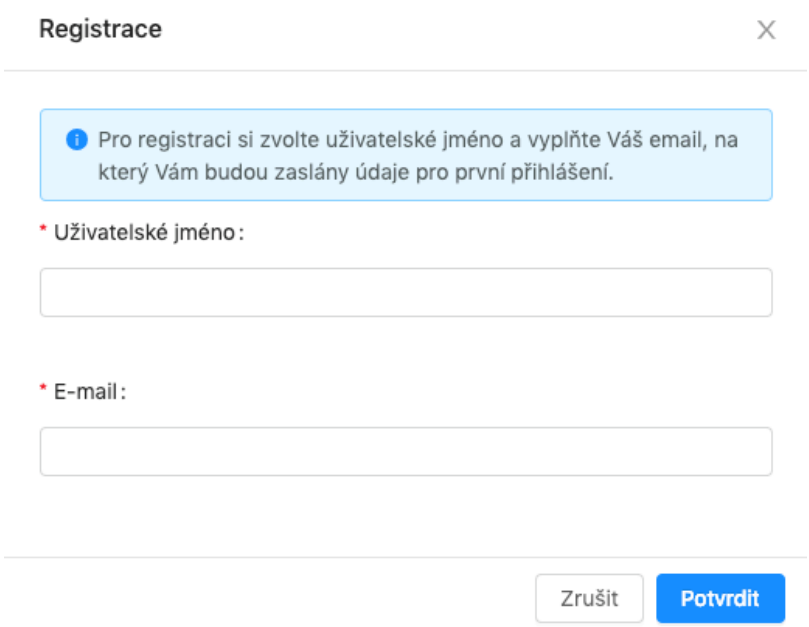
4.3.6 Realizace požadavků

Požadavky na vložení, úpravu, mazání a publikování článků, zobrazení přehledů a na správu uživatelů byly realizovány v administračním rozhraní aplikace, které je přístupné přihlášeným uživatelům s příslušným oprávněním, do kterého lze vstoupit prostřednictvím odkazu *admin* v pravém horním rohu aplikace, popřípadě přístupem na URL */admin*. Ostatní funkce jsou realizovány v uživatelském rozhraní systému, ve kterém jsou dle role přihlášeného uživatele zobrazovány příslušné ovládací prvky.

4.3.6.1 Registrace a přihlášení uživatelů

Uživatelé se mohou do aplikace zaregistrovat zvolením odkazu „registrovat“ v jejím pravém horním rohu. Registrační formulář (Obrázek 25) je zobrazen v modálním okně a obsahuje vstup pro uživatelské jméno a email. Oba vstupy jsou validovány a nemohou být odeslány nevyplněné.

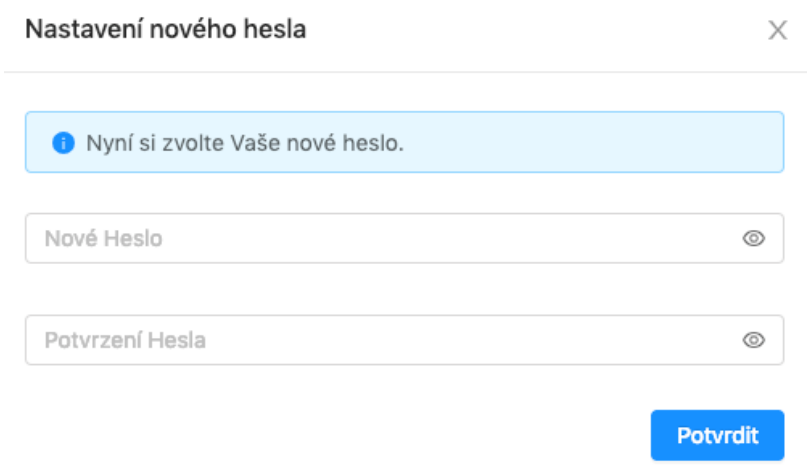
Obrázek 25 - Registrace uživatele



Zdroj: vlastní zpracování

Validace dále probíhá i na straně serveru z hlediska unikátního uživatelského jména. Jakmile je formulář úspěšně potvrzen, je uživateli odeslán email s informacemi pro první přihlášení, konkrétně tedy s uživatelským jménem a vygenerovaným heslem.
Při prvním přihlášení je pak uživatel po zadání uživatelského jména a hesla přesměrován na formulář pro zvolení vlastního hesla do systému (Obrázek 26). Formulář obsahuje dva vstupy pro heslo, které musejí obsahovat stejnou hodnotu pro ověření správnosti hesla a na serverové straně je navíc validován na dostatečnou složitost hesla, kterou je možné nastavit v administračním rozhraní AWS Cognito. Aktuálně je heslo nastavené na minimální délku 8 znaků a je vyžadována alespoň jedna číslice.

Obrázek 26 - Nastavení nového hesla



Zdroj: vlastní zpracování

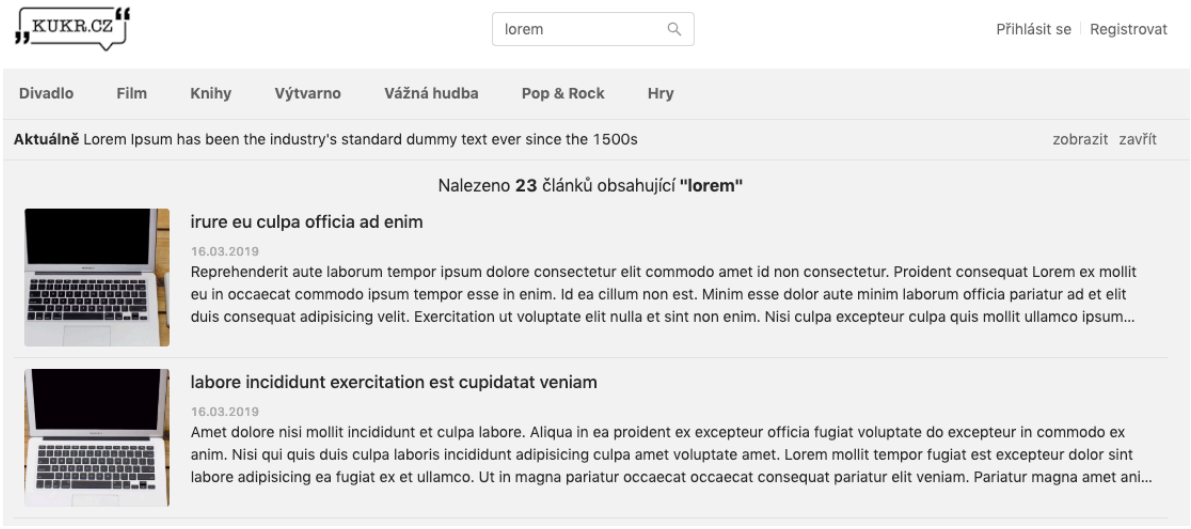
4.3.6.2 Expirace členství uživatelů

Pro nastavení expirace členství uživatelů bylo nutné do AWS Cognito přidat vlastní atribut pro ukládání této hodnoty. Protože Cognito podporuje pouze jednoduché datové typy, je hodnota ukládána jako textový řetězec. Hodnota tohoto atributu je následně využívána pro autorizaci uživatele a registrovaným uživatelům zobrazována pro informační účely.

4.3.6.3 Vyhledání článku

Články je možné z uživatelského rozhraní vyhledávat pomocí pole umístěného v horní části aplikace. Po zadání vyhledávaného řetězce a stisknutí klávesy ENTER nebo kliknutím na ikonu lupy je odeslán požadavek na službu GraphQL, která vybere z databáze takové články, které v textu nebo v nadpisu obsahují vyhledávanou frázi. Seznam těchto článků je odeslán v odpovědi klientovi a zobrazen jako seznam společně s počtem odpovídajících článků (Obrázek 27).

Obrázek 27 - Výsledek vyhledávání



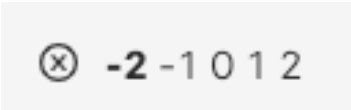
Zdroj: vlastní zpracování

Vyhledávaná fráze je ukládána do URL adresy, uživatel se tedy po obnovení stránky ocitne opět na seznamu vyhledávání. Výhodou je také možnost ukládání vyhledávaných frází do záložek prohlížeče.

4.3.6.4 Hodnocení článků

Přihlášený uživatel, který si zobrazí detail článku má možnost tento článek ohodnotit na škále od -2 do 2. Z tohoto hodnocení je následně vypočítáván průměr pro získání přehledu o kvalitě článku. Pokud již uživatel článek ohodnotil, je jeho hodnocení zvýrazněno tučným písmem a také má možnost své hodnocení smazat pomocí tlačítka které se po ohodnocení zobrazí vedle hodnotící škály (Obrázek 28).

Obrázek 28 - Hodnocení článků



Zdroj: vlastní zpracování

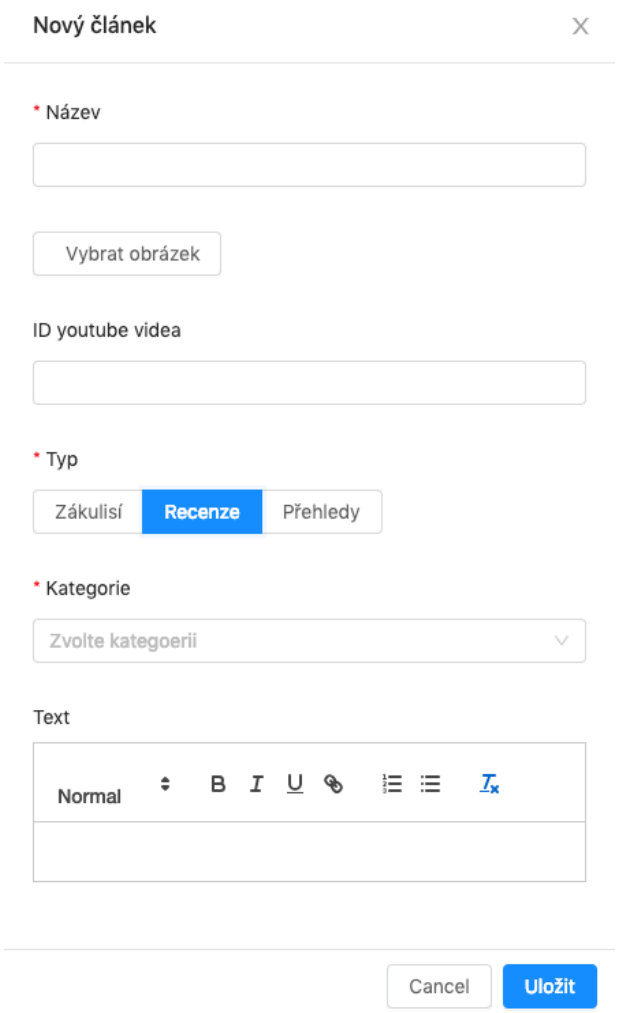
4.3.6.5 Odborná diskuze a hodnocení komentářů

Do odborné diskuze lze přejít z detailu článku pomocí ovládacího prvku *vstoupit do diskuze* na jeho konci. Přihlášení uživatelé mají kromě výpisu komentářů možnost také vytvářet a hodnotit komentáře. Hodnocení komentářů je obdobné jako hodnocení článků.

4.3.6.6 Vložení, úprava a smazání článku

V administračním rozhraní je možné zobrazit seznam článků aktuálně přihlášeného uživatele na záložce *Mé články*. Seznam obsahuje název článku, jeho status, kategorii a tlačítka pro úpravu a smazání. Tyto akce jsou zobrazovány v závislosti na statusu článku a nejsou k dispozici, pokud je článek zveřejněný. Formulář pro vytvoření článku (Obrázek 29) je zobrazen v modálním okně po zvolení *Vytvořit článek.* Pro úspěšné uložení je třeba vyplnit název článku, nahrát obrázek a vybrat typ. Pokud je zvolen typ *Recenze,* je ve formuláři navíc zobrazen prvek pro výběr kategorie.

Obrázek 29 - Formulář pro vložení článku



Zdroj: vlastní zpracování

Vytvořením se článek automaticky přesouvá do stavu *Nový*, jeho úpravou do stavu *Upravený*. Obrázek je nahráván do cloudového úložiště AWS S3. Po vložení, úpravě a smazání článku dochází k automatickému obnovení seznamu článků, konkrétně takové stránky seznamu, na které se uživatel nachází.

4.3.6.7 Redakční úpravy a zveřejňování

Pro redakční úpravy je v administračním rozhraní vytvořena záložka *Redakce*, kde je, obdobně jako na záložce *Mé články,* zobrazen seznam článků. U každého článku je navíc zobrazen jeho autor a je možné měnit jeho status. Status článku je možné měnit dle stavového diagramu uvedeného v kapitole 4.2.5.

4.3.6.8 Přehledy návštěvnosti

Zobrazení přehledů návštěvnosti bylo realizováno napojením na službu Google Analytics, ze které jsou načteny nejvíce navštěvované stránky za posledních 14 dní. Informace o těchto stránkách jsou spárovány s informacemi v databázi, aby v přehledu bylo možné zobrazit existující články a případně si je zobrazit. Další informace je pak možné získat přímo v účtu Google Analytics.

4.4 Validace

Systém byl v průběhu vývoje testován jednotkovými testy a testy pro kontrolu kvality kódu. Po dokončení byly validovány jednotlivé testovací scénáře vytvořené dle diagramu případu užití a dále byl otestován celkový výkon systému.

4.4.1 Testovací scénáře

Průchody testovacími scénáři byly otestovány manuálně s předem připravenými daty. Každý scénář má definován akci, kterou uživatel provede a případně data, která tato akce vyžaduje. Vytvořeny byly testovací scénáře pro funkční požadavky, které měly nejvyšší prioritu stanovenou v seznamu požadavků (Tabulka 4).

4.4.1.1 Přihlašování

Pro přihlášení byly vytvořeny dva testovací scénáře, první pro uživatele, který již provedl první přihlášení a nastavení hesla a druhý pro uživatele, který provádí své první přihlášení. Cílem prvního testu (Tabulka 9) je úspěšné přihlášení uživatele. Předpokladem je nepřihlášený uživatel nacházející se na libovolné stránce.

Tabulka 9 - Testovací scénář: Přihlašování

KROK	POPIS	DATA	POŽADOVANÝ VÝSLEDEK
1	Zvolení odkazu "Přihlásit se" v pravém horním rohu		Zobrazen přihlašovací formulář v modálním okně
2	Vyplnění uživatelského jména	kukr.admin	Formulářový prvek obsahuje zadané uživatelské jméno
3	Vyplnění hesla	kukr1234	Formulářový prvek obsahuje heslo ve skryté formě
4	Odeslání formuláře tlačítkem "Přihlásit"		Formulář je odeslán, tlačítko je během přihlašování ve stavu načítání. Po přihlášení je uživateli zobrazena notifikace o úspěšném přihlášení.

Zdroj: vlastní zpracování

Cílem druhého testu je ověření procesu prvního přihlášení uživatele (Tabulka 10). Úspěšný konec testu nastane, jakmile je uživatel přihlášený. Předpokladem je nepřihlášený uživatel s novým účtem, který ještě neprovedl první přihlášení a nastavení hesla.

Tabulka 10 - Testovací scénář: První přihlášení

KROK	POPIS	DATA	POŽADOVANÝ VÝSLEDEK
1	Zvolení odkazu "Přihlásit se" v pravém horním rohu		Zobrazen přihlašovací formulář v modálním okně
2	Vyplnění uživatelského jména	test	Formulářový prvek obsahuje zadané uživatelské jméno
3	Vyplnění hesla, které uživatel obdržel v e-mailu	tmppassword	Formulářový prvek obsahuje heslo ve skryté formě
4	Odeslání formuláře tlačítkem "Přihlásit"		Formulář je odeslán, tlačítko je během přihlašování ve stavu načítání.
5	Vyplnění nového hesla	kukr1234	
6	Vyplnění potvrzení hesla	kukr1234	
7	Odeslání formuláře tlačítkem "Potvrdit"		Formulář je odeslán, po úspěšném nastavení hesla je uživatel přihlášen.

Zdroj: vlastní zpracování

4.4.1.2 Registrace

Cílem testovacího scénáře pro registraci (Tabulka 11) je úspěšné uložení nového uživatele do databáze s rolí „registrovaný“ a odeslání informací k prvnímu přihlášení novému uživateli na zadaný email.

Tabulka 11 - Testovací scénář: Registrace uživatele

KROK	POPIS	DATA	POŽADOVANÝ VÝSLEDEK
1	Zvolení odkazu "Registrovat" v pravém horním rohu		Zobrazen registrační formulář v modálním okně
2	Vyplnění uživatelského jména	test	Formulářový prvek obsahuje zadané uživatelské jméno
3	Vyplnění e-maili	test@kukr.cz	Formulářový prvek obsahuje zadaný email
4	Odeslání formuláře tlačítkem "Potvrdit"		Formulář je odeslán, tlačítko je během registrace ve stavu načítání. Po registraci uživatel obdrží přihlašovací údaje na zadaný e-mail

Zdroj: vlastní zpracování

4.4.1.3 Vytvoření článku

Cílem testu (Tabulka 12) je úspěšné vložení článku a jeho zobrazení v seznamu článků. Předpokladem pro spuštění je přihlášený uživatel s právy pro vytváření článků.

Tabulka 12 - Testovací scénář: Vytvoření článku

KROK	POPIS	DATA	POŽADOVANÝ VÝSLEDEK
1	Zvolení odkazu "admin" v pravém horním rohu		Zobrazena úvodní stránka administračního rozhraní
2	Zvolení odkazu "mé články" v menu		Zobrazen seznam článků vytvořených aktuálně přihlášeným uživatelem
3	Stisknutí tlačítka "Vytvořit článek"		Zobrazeno modální okno s formulářem
4	Vyplnění názvu článku	Testovací článek	Formulářový prvek obsahuje zadanou hodnotu
5	Výběr obrázku	Libovolný obrázek z disku	Název obrázku je zobrazen pod formulářovým prvkem pro nahrávání
6	Zvolení typu článku	Recenze	Hodnota "recenze" je zvýrazněna. Zobrazení formulářového prvku pro výběr kategorie
7	Výběr kategorie	Divadlo	Zvolená hodnota je zobrazena jako vybraná
8	Zadání textu článku	Lorem ipsum	Formulářový prvek obsahuje zadanou hodnotu
9	Odeslání formuláře tlačítkem "Uložit"		Formulář je odeslán, tlačítko pro uložení je ve stavu načítání, dokud není ukládání dokončeno. Po uložení je článek zobrazen v seznamu článků

Zdroj: vlastní zpracování

4.4.1.4 Smazání článku

Cílem testu (Tabulka 13) je úspěšné smazání článku a jeho následné odstranění ze seznamu článků. Předpokladem pro spuštění je přihlášený uživatel s alespoň jedním článkem, který není ve stavu "zveřejněný" a který se nachází na stránce s výpisem článků v administračním rozhraní.

Tabulka 13 - Testovací scénář: Smazání článku

KROK	POPIS	POŽADOVANÝ VÝSLEDEK
1	Stisknutí tlačítka pro smazání u libovolného článku	Zobrazeno okno pro potvrzení akce
2	Potvrzení smazání článku	Zpráva o úspěšně provedené akci a odstranění článku ze seznamu

Zdroj: vlastní zpracování

4.4.1.5 Úprava článku

Cílem testu (Tabulka 14) je úspěšné upravení existujícího článku a jeho následná aktualizace v seznamu. Předpokladem pro spuštění je přihlášený uživatel s alespoň jedním článkem, který není ve stavu "zveřejněný" a který se nachází na stránce s výpisem článků v administračním rozhraní.

Tabulka 14 - Testovací scénář: Úprava článku

KROK	POPIS	DATA	POŽADOVANÝ VÝSLEDEK
1	Stisknutí tlačítka pro úpravu u libovolného článku		Zobrazeno modální okno s formulářem s předvyplněnými hodnotami
2	Změna názvu článku	Testovací článek úprava	Formulářový prvek obsahuje zadanou hodnotu
3	Odeslání formuláře tlačítkem "Uložit"		Formulář je odeslán, tlačítko pro uložení je ve stavu načítání, dokud není ukládání dokončeno. Po uložení je článek aktualizován v seznamu článků

Zdroj: vlastní zpracování

4.4.1.6 Publikování článku

Cílem testu (Tabulka 15) je úspěšná změna stavu existujícího článku a také kontrola, zda se zveřejněný článek nachází ve výpisu článků přístupném neregistrovaným uživatelům. Předpokladem spuštění je přihlášený uživatel s alespoň jedním článkem, který není ve stavu „zveřejněný“ a který se nachází na stránce „redakce“ v administračním rozhraní.

Tabulka 15 - Testovací scénář: Publikování článku

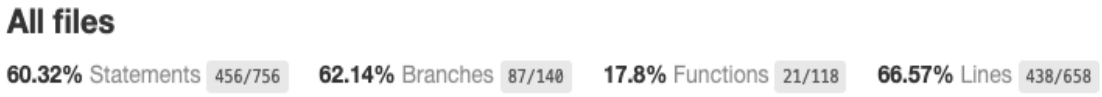
KROK	POPIS	POŽADOVANÝ VÝSLEDEK
1	Zvolení hodnoty ve výběrovém poli na výpisu u libovolného článku na „zveřejněný“	Hodnota ve formulářovém prvku je změněna
2	Načtení hlavní stránky	Seznam článků obsahuje právě zveřejněný článek.

Zdroj: vlastní zpracování

4.4.2 Jednotkové testy

Jednotkové testy byly vytvářeny průběžně při vývoji systému. Systém je pokryt pouze určitým procentem jednotkových testů, protože na některé části kódu nelze jednotkový test aplikovat, popřípadě nedává smysl takovýto test vytvářet. Dle vygenerovaného reportu (Obrázek 30) nástrojem Jest je v systému pokryto zhruba 66% řádek kódu. Při dalším rozvoji systému by bylo vhodné toto číslo dále zvyšovat.

Obrázek 30 - Report pokrytí testy nástrojem Jest

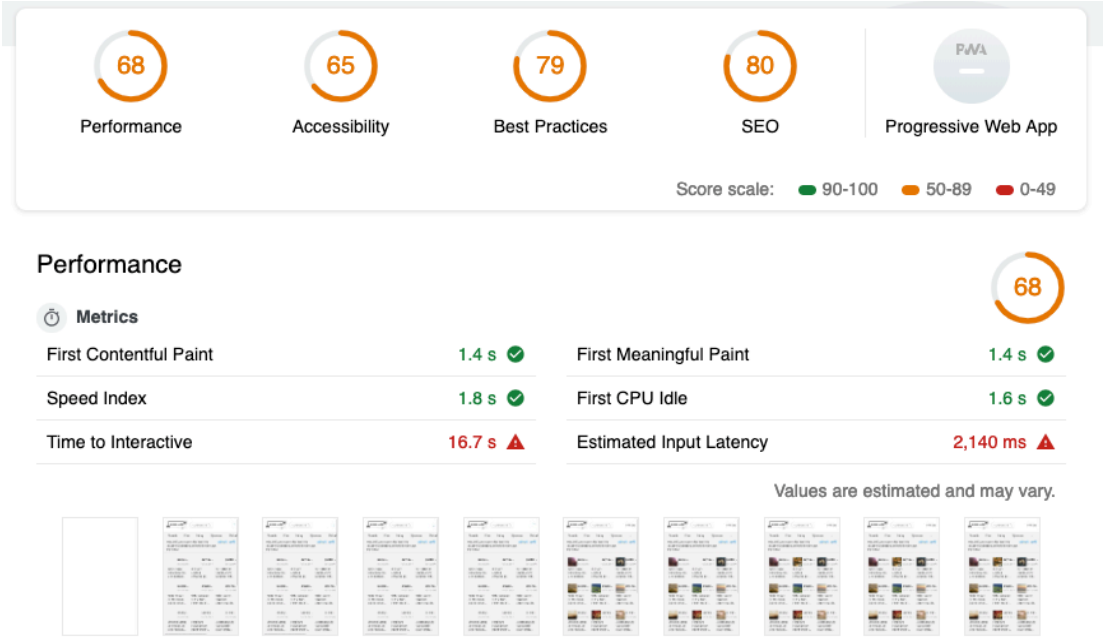


Zdroj: vlastní zpracování

4.4.3 Výkon

Systém byl z hlediska výkonu testován nástrojem Google Lighthouse, pomocí kterého byl vygenerován report s přehledem výsledků výkonnostních testů (Obrázek 31). Před otestováním výkonu bylo do databáze vygenerováno 1000 záznamů a následně spuštěn test pro hlavní stránku.

Obrázek 31 - Report vygenerovaný nástrojem Google Lighthouse



Zdroj: vlastní zpracování

Výsledky z nástroje ukazují, že systém zaostává při měření latence a času do interaktivity. Hlavní ukazatel výkonu ale s hodnotou 68/100 ukazuje solidní výkon, který by uživatele při používání systému neměl nijak omezovat. Pro dosažení tohoto skóre bylo nutné komprimovat JavaScript a CSS soubory a nastavit ukládání obrázků do mezipaměti prohlížeče. V dalších krocích by pro zvyšování výkonu bylo vhodné optimalizovat velikost obrázků a zvýšit výkon serveru.

Pro generování článků do databáze byla pro účely testování systému vytvořena funkce, která v cyklu vytváří články. Každý článek se skládá z náhodně vybraného typu (zákulisí, přehledy, recenze), uživatele a stavu (nový, upravený, zveřejněný, zamítnutý) a URL adresy obrázku vybrané z online generátoru obrázků *https://picsum.photos/*. Pokud se jedná o článek typu recenze, je k němu navíc přiřazena náhodně vybraná kategorie. Dále je pomocí knihovny *lorem-ipsum* vygenerovaný obsah a titulek.

4.5 Nasazení

Před nasazením aplikace na veřejné prostředí bylo zapotřebí nejprve zdrojový kód zkompilovat do spustitelné formy. Pro kompilaci byl zvolen nástroj *Webpack*, který převádí všechny závislosti do statických souborů. Další konfigurací tohoto nástroje bylo

docíleno i optimalizace souborů pro produkční prostředí, při kterém byly kompilované zdrojové kódy následně minifikovány[3] a komprimovány[4]. Výsledkem kompilace bylo několik souborů.

Tabulka 16 - Výsledek kompilace projektu

NÁZEV	**POPIS**
browser.js	Klientská část aplikace
browser.css	CSS styly pro klientskou část aplikace
Server.js	Serverová část aplikace
*.gz	Komprimované soubory

Zdroj: vlastní zpracování

Nasazení projektu probíhá automaticky díky nástrojům průběžné integrace a průběžného nasazení. Vzhledem k nezávislosti projektu, kdy nebylo nutné provádět integraci s dalšími systémy zákazníka, bylo pro nasazení pouze nutné nastavit verzovací systém a prostředí na veřejném serveru pro nasazování aplikace. Verzování projektu bylo zabezpečeno systémem Git společně s jeho vzdáleným repozitářem v cloudobé službě Bitbucket. Každá změna v tomto vzdáleném repozitáři spouští řadu příkazů (Tabulka 17) pro otestování a odeslání aplikace do produkčního prostředí na veřejném serveru.

[3] Minifikace je proces automatizované změny názvů proměnných na jednoznakové, odstranění komentářů a formátování kódu vymazáním mezer za účelem zmenšení velikosti zdrojového kódu.
[4] Soubory aplikace byly komprimován do formátu GZip.

Tabulka 17 - Sada příkazů spouštěná při nasazování

PŘÍKAZ	POPIS
yarn	Instalace závislostí projektu pomocí balíčkovacího systému yarn.
yarn lint	Kontrola zdrojového kódu, zda odpovídá předepsaným pravidlům formátování.
yarn test	Spuštění testů
yarn build	Vytvoření produkční verze aplikace
./heroku-deploy.sh	Skript pro odeslání souborů na veřejný server

Zdroj: vlastní zpracování

Pokud jsou všechny kroky úspěšně dokončeny, je aplikace dostupná na adrese *kukr.herokuapp.com*. První načtení této stránky trvá přibližně půl minuty, což je způsobeno zvoleným plánem cloudové služby Heroku, na které je tato aplikace hostována a který je poskytován zdarma. Při nutnosti zvýšení výkonu se jedná pouze o nastavení této služby, která poskytuje široké možnosti škálování.

4.6 Přístup k systému

Systém je nasazený a přístupný na URL adrese *kukr.herokuapp.com.* Pro účely otestování byly vytvořeno několik uživatelů s různými rolemi. Tito uživatelé a jejich přihlašovací údaje jsou uvedeny v následující tabulce:

Tabulka 18 - Přístupové údaje k systému

UŽIVATELSKÉ JMÉNO	ROLE	HESLO
kukr.admin	administrátor	kukr1234
kukr.redakce	redakce	kukr1234
kukr.externista	externista	kukr1234
kukr.registrovany	registrovaný	kukr1234

Zdroj: vlastní zpracování

Pro lokální spuštění je nutné nejprve spustit databázi MongoDB a dále nainstalovat závislosti systému ve složce se zdrojovými kódy spuštěním příkazu *yarn*. Dalším krokem

je kompilace aplikace příkazem *yarn build.* Pokud je kompilace dokončena úspěšně, je možné sytém spustit příkazem *yarn start:build* a následně otevřít webový prohlížeč na adrese *localhost:3333.* Systém nebude obsahovat žádná data, nicméně přihlášení je sdílené, proto je možné se i do aplikace spuštěné na lokálním prostředí přihlásit přihlašovacími údaji z výše uvedené tabulky (Tabulka 18).

5 Výsledky a diskuse

Vytvořený systém je plně funkční a připravený na použití zadavatelem. Využití systému není ale omezeno pouze pro tyto účely, protože návrh a realizace je kromě uživatelského rozhraní dostatečně abstraktní pro jakékoli zpravodajství a informační účely.

5.1 Možnosti rozšíření

Systém by bylo vhodné rozšířit napojením na platební bránu a poskytnout tak uživatelům možnost automatického prodloužení členství bez nutnosti zásahu administrátora a kontroly bankovního účtu. Dále by bylo možné rozšířit algoritmus pro výpočet celkového hodnocení článků a komentářů, pro který je nyní využit pouhý aritmetický průměr, jakmile by byl systém schopen vypočítat váhu jednotlivých uživatelů, bylo by vhodné využít vážený průměr.

Rozšířením administrační části o správu reklamy by bylo možné sledovat a řídit aktuálně zobrazované reklamy. Systém je v tuto chvíli schopen reklamu zobrazovat, nicméně neposkytuje přihlášeným uživatelům rozhraní pro nahrávání reklamních bannerů.

5.2 Vhodné úpravy

Vhodnou úpravou systému je zamezit hodnocení článků a komentářů, které aktuálně přihlášený uživatel vytvořil. Systém v současné době dovoluje hodnotit své vlastní články a komentáře. Zlepšit by bylo dále možné i SEO optimalizaci stránky z hlediska popisu a klíčových slov. Tyto informace totiž na stránce zatím nejsou zobrazovány. První možností řešení tohoto problému je rozšíření formuláře pro vkládání a úpravu článku o další prvky, pomocí kterých by bylo možné tyto informace vkládat. Druhou možností je automatické generování těchto informací algoritmem. Další vhodnou úpravou je zlepšení procesu nahrávání obrázků na cloudové úložiště. Velikost obrázků totiž není upravována a jsou tak načítány v plné velikosti i pro náhled. Řešením je uložení těchto obrázků ve třech velikostech – v malé pro hlavní stranu, střední pro výpis kategorie a výsledek vyhledávání a v plné velikosti pro detail článku. Úpravou by došlo ke zrychlení načítání stránky. Při využití v reálném provozu by bylo také vhodné nastavit vlastní doménové jméno a nevyužívat URL *kukr.herokuapp.com*.

6 Závěr

Cílem práce bylo vytvoření webového informačního systému pro recenze filmů, divadelních her a dalších uměleckých představení s dodržením postupů a principů softwarového inženýrství a UML.

V teoretické části byly definovány základní pojmy softwarového inženýrství, webových aplikací a technologií, které byly pro návrh a vývoj využívány. Ve vlastní práci byla provedena analýza písemného zadání a následně specifikovány funkční a nefunkční požadavky na systém a požadavky na přístupová práva k jednotlivým funkcím systému. Tyto informace byly využity pro návrh systému pomocí UML diagramů, drátěných modelů a pro grafický návrh uživatelského rozhraní. Prvním krokem vývojové fáze bylo nastavení podpůrných nástrojů pro splnění definovaných charakteristik dobrého programového kódu, příprava prostředí na vzdáleném serveru a automatického vydávání nových verzí a testování pomocí nástrojů průběžné integrace a průběžného nasazování. Dále byly popsány implementační specifika serverové a klientské časti včetně ukázky konkrétního řešení vybraných požadavků. Během vývoje byl systém testován jednotkovými a výkonnostními testy a testovacími scénáři, které byly vytvořeny pro funkční požadavky s nejvyšší prioritu stanovenou v seznamu požadavků. Na závěr byla popsána příprava a optimalizace systému pro nasazení na veřejný server a samotný proces nasazování. Výsledkem práce je funkční systém nasazený na veřejný server a připravený k použití.

7 Seznam použitých zdrojů

BANKER, Kyle, Peter BAKKUM, Shaun VERCH, Douglas GARRETT a Tim HAWKINS, 2016. *MongoDB in action*. Second edition. Shelter Island, NY: Manning. ISBN 978-161-7291-609.

BANKS, Alex a Eve PORCELLO, 2018. *Learning GraphQL*. Sebastopol, CA: O'Reilly. ISBN 9781492030706.

BOJINOV, Valentin, 2018. *RESTful Web API Design with Node.js 10*. Third Edition. Packt Publishing.

BOYD, Ryan, 2012. *Getting started with OAuth 2.0*. Sebastopol, Calif.: O'Reilly. ISBN 978-144-9311-605.

COCKBURN, Alistair, 2005. *Use Cases: jak efektivně modelovat aplikace*. Brno: CP Books. ISBN 8025107213.

DAOUST, Norman, 2012. *UML requirements modeling for business analysts*. Westfield, NJ: Technics Publications. ISBN 978-1-9355042-4-5.

DENNIS, Alan, Barbara Haley WIXOM a David Paul TEGARDEN, 2012. *Systems analysis design, UML version 2.0: an object oriented approach*. 4th ed. Hoboken, NJ: John Wiley. ISBN 978-1-118-03742-3.

DÜÜNA, Karl, 2016. *Secure your Node.js web application: keep attackers out and users happy*. Dallas, Texas: The Pragmatic Bookshelf. ISBN 978-168-0500-851.

EDWARD, Shakuntala Gupta, 2015. *Practical MongoDb: architecting, developing, and administering MongoDB*. New York, NY: Springer Science Business Media. ISBN 978-1484206485.

GOURLEY, David a Brian TOTTY, 2002. *HTTP: the definitive guide*. Sebastopol, CA: O'Reilly. ISBN 978-156-5925-090.

HAHN, Evan, 2013. *JavaScript testing with Jasmine*. Sebastopol, CA: O'Reilly Media. ISBN 14-493-5637-0.

MACCAW, Alex, 2011. *JavaScript Web Applications*. Sebastopol: O'Reilly. ISBN 978-1-449-30351-8.

MALLACH, Efrem, 2009. *Information System Conversion Strategies: A Unified View* [online]. USA [cit. 2019-03-03]. Dostupné z: http://www.irma-international.org/viewtitle/3950/

MONTEIRO, Fernando, 2014. *Learning single-page web application development: build powerful and scalable single-page web applications using a full stack JavaScript environment with Node.js, MongoDB, AngularJS, and the Express framework*. Birmingham: Packt Publishing. ISBN 978-1-78355-209-2.

SALEH, Hazem, 2013. *JavaScript Unit Testing*. Packt Publishing. ISBN 9781782160625.

SATERNOS, Casimir, St. Laurent, Simon ST. LAURENT, SIMON, Allyson MACDONALD a Rebecca DEMAREST, 2014. *Client-server web apps with JavaScript and Java*. Sebastopol, CA: O'Reilly Media. ISBN 978-144-9369-330.

SHENOY, Aravind a Anirudh PRABHU, 2016. Introducing SEO: your quick-start guide to effective SEO practices. India: Apress. ISBN 978-1-4842-1853-2.

SHKLAR, Leon a Rich ROSEN, 2009. *Web application architecture: principles, protocols and practices*. 2nd ed. Hoboken, NJ: Wiley. ISBN 978-047-0518-601.

SOMMERVILLE, Ian, 2013. *Softwarové inženýrství*. Brno: Computer Press. ISBN 978-80-251-3826-7.

STEPHENS, Rod, 2015. *Beginning software engineering*. Indianapolis, IN: John Wiley. ISBN 11-189-6914-6.

SUBRAMANIAN, Chandramouli, Saikat DUTT, Chandramouli SEETHARAMAN a B. G. GEETHA, 2015. *Software Engineering*. Pearson Education India. ISBN 9789332558298.

TSUI, Frank F., Orlando KARAM a Barbara BERNAL, 2017. *Essentials of software engineering*. Fourth edition. Burlington, Massachusetts. ISBN 978-1-2841-0600-8.

VLĂDUȚU, Alexandru, 2014. *Mastering Web Application Development with Express*. Birmingham B3 2PB, UK: Packt Publishing. ISBN 978-1-78398-108-3.

8 Přílohy

8.1 Přiložené CD

Přiložené CD obsahuje zdrojové kódy aplikace.