Univerzita Hradec Králové
Fakulta informatiky a managementu
KIKM

Moderní klient pro Usenet news

Bakalářská práce

Autor: Adam Kvasnička
Studijní obor: AI3

Vedoucí práce: Ing. Pavel Kříž, Ph.D.

Hradec Králové duben 2019

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

vlastnoruční podpis

V Hradci Králové dne:

Poděkování:
Děkuji vedoucímu práce Ing. Pavlu Kříži, Ph.D. za cenné rady, čas a hlavně pevné nervy.

Anotace

Práce nabízí základní pohled do zákulisí výměny diskuzních článku pomocí distribučního diskuzního systému Usenet.

První část práce je zaměřena na základní informace o Usenet news a síťovém protokolu NNTP. Následně budou otestována stávající dostupná klientská řešení včetně jejich výhod a nevýhod. V neposlední řadě bude popsán postup implementace samotného moderního Usenet news klienta.

Výsledkem celé práce je webová aplikace. Určena je zejména Usenet poskytovatelům, ti ji mohou ve svém vlastním prostředí nasadit a nabídnout tak přístup k jejich článkům skrze moderní rozhraní. Samotná aplikace je vyvinuta pouze za pomoci programovací jazyka JavaScript s využitím na něm založených technologií jako je Node.js nebo React.js.

Klíčová slova

Usenet, Webnews, NNTP, React.js, Koa, JavaScript, Newsreader, Node.js

Annotation

Title: Modern Usenet news client

This thesis offers a basic view into the background of discussion article exchanges, using the distributional discussion system called Usenet.

The first part of thesis focuses on basic information about Usenet and NNTP network protocol. Next, the existing client solutions will be run through tests, which will determine their flaws and advantages. Last but not least, the implementation process of the modern Usenet news client itself will be described.

The final result will be a web application. It is intended mainly for Usenet providers, who will be able to deploy this application alongside their existing Usenet provider, providing easy access to its articles for their users.

The application is developed using only the JavaScript programing language and technologies based on it, such as Node.js or React.js.

Obsah

Seznam obrázků

Seznam tabulek

Seznam grafů

Seznam příkazů

1 Úvod

Cílem této práce je návrh a implementace moderního více platformního softwarového produktu, umožňující práci se sítí Usenet za použití protokolu NNTP.

Autor této práce je aktivním čtenářem mnoha volně dostupných Usenet poskytovatelů. Jedním z příkladů může být volně dostupný Usenet server veřejného spolku HKFree.org, jehož je členem. Bohužel v dnešní době existuje pouze malá hrstka vskutku moderních volně dostupných čteček, jež by umožnovala příjemnou práci s Usenety. Hlavním důvodem výběru tohoto tématu je zejména tvorba takového řešení, které bude sám autor rád používat. Protože je autor velkým fanouškem programování a zejména kreativní tvorby, práce pro něj zajisté bude spojení zábavného s užitečným.

Usenet jako takový byl do volného světa vypuštěn roku 1980. Tedy o dekádu dříve než klasický *World Wide Web.* To z něj dělá jednu z nejstarších počítačových komunikačních sítí. Ve zkratce se jedná se o celosvětový distribuční diskuzní systém. Usenet v mnoha směrech připomíná *Bulletin Board Systém*. V mnoha ohledech se též jedná o předchůdce klasických internetových fór. Uživatel pomocí Usenetu může číst a vkládat články, takzvané *news*. Ty jsou vkládány do jedné nebo více diskuzních skupin zvaných newsgroups. Hlavním rozdílem Usenetu oproti výše zmíněným službám je absence centrálního prvku a dedikované administrace. Usenet je distribuován mezi velké množství, konstantně se měnících serverů. Ty si navzájem, pomocí protokolu NNTP, mezi sebou články vyměňují. Díky tomu vzniká takzvaný *news feed* [1].

2 Protokol NNTP

Network News Transfer Protocol, dále pouze NNTP, je síťový přenosový protokol, slouží k distribuci, odběru, vyhledávání a odesílání článku do určených diskuzních skupin takzvaných Usenet newsgroups. NNTP protokol je tedy standardizovaná sada příkazů sloužící pro definování komunikace mezi klientem a Usenet serverem [2].

Standardizace protokolu je definována například v RFC dokumentu 977 [2].

2.1 Popis protokolu

Usenet server, poskytuje své služby pomocí spolehlivého síťového protokolu TCP, nejběžnější, v praxi používaný, síťový port, na kterém host naslouchá je 119.

Komunikace klienta a serveru probíhá výhradně užitím textových ASCII příkazů, skládající se z jednoho klíčového slova, které přesně definuje požadovaný příkaz, a dalšími, ve většině případech volitelnými, parametry. Každý příkaz musí být zakončen odřádkováním, tedy kombinací byte-kódových znaků CR (carriage return, nebo také‘\r’) a LF (line feed, nebo‘\n’).

Odpověď serveru se následně skládá ze stavové části, stavové části a jejich možné příklady jsou uvedeny v kapitole **2.5.2**, samotný stavový kód je většinou následován textovou odpovědí skládající, jež se může skládat buď z jednoho nebo většího množství řádků, stejně jako při odesílání příkazů je každý řádek odpovědi zakončen kombinací znaků CR a LF. Veškeré serverové odpovědi, musí být za každé okolnosti zakončeny tečkou ‘.’ [3].

2.2 Rozšíření protokolu

Protokol samozřejmě díky růstu popularity, zejména v 90. letech, nabýval různorodosti, co se implementací pro mnoho různých platforem a operačních systému týče. Bylo tedy uznáno, že je zapotřebí provést celkovou revizi RFC dokumentu 977 [2]. Práce na revizi započali roku 1991, tato práce bohužel v nový standard nakonec nevyústila. Se změnou přišel až rok 2000, kdy si mnoho nápadů a myšlenek vzniklých z předchozí práce našlo svou cestu do implementace zejména řad nových příkazů. Současně bylo použito mnoho rozšíření vytvořených zejména autory softwaru určených k usnadnění práce s Usenety, tedy klientů

určených pro čtení. Všechny tyto změny a implementace byly přesně definovány a specifikovány v novém RFC dokumentu s číslem 2980 [4]. Je však potřeba brát v potaz že tento dokument, nepředstavuje nový standard, nýbrž pouze zaznamená implementaci a specifikaci možných rozšířeních, která součástí standardu NNTP nebyly.

Práce na rozšířeních poté neustala a v roce 2006 spařilo světlo světa nová specifikace protokolu definována v RFC dokumentu číslo 3977 [5]. Ten již oficiálně aktualizuje dosluhující RFC dokument 977 [2] a klasifikuje některé jeho neurčitosti. Přidány byly nové bázní funkcionality a specifikován mechanismus pro přidání standardizovaných přídavků protokolu NNTP. Ačkoliv je tato specifikace většinově zpětně kompatibilní s RFC dokumentem 977 [2], přišla také řada změn, které by mohly starším Usenet klientům způsobit problémy, jedná se zejména o níže uvedené změny:

- Základní sada znaku byla změněna z US-ASCII na UTF-8.
- Řada příkazů, které byly v RFC dokumentu 977 [2] definovány jako nepovinné, nebo byly přebrány z RFC dokumentu 2980 [4], jsou nyní implementačně povinné.
- Byl přidán příkaz **CAPABILITIES**, který umožňuje klientům získat výčet funkcionalit, jež daný host nabízí.

Další změnou, tentokrát týkající se zabezpečení, přineslo v téže roce rozšiřující specifikace definovaná v RFC dokumentu 4643 [6]. Tradičně byl přístup do diskuzních skupin veřejný, v některých případech byla autentifikace uživatelů velmi užitečná, jedním z takových případů byla nutnost kontroly konzumace zdrojů nebo zamezení zneužívání příkazu **POST**. Pro tento účel bylo nutné zajistit jakoukoliv vhodnou formu autentifikace mezi klientem a serverem. Mimo jiné taky definovat kroky negociačního procesu a bezpečností vrstvy pro veškeré příkazy po dobu trvání aktuální interakce mezi serverem a uživatelem. Hlavní změnou z oblasti zabezpečení tedy byla aktualizace a formalizace příkazů **AUTHINFO USER**

a **AUTHINFO PASS**, které nahradily zastaralé **AUTHINFO SIMPLE** a **AUTHINFO GENERIC** definovaných v RFC dokumentu 2980 [4] [6].

Následně v tom samém roce přišla další specifikace, tentokrát rozšiřující RFC dokument 2980 [4], s číslem 4644 [7]. Hlavní myšlenkou této specifikace byl efektivnější přenos článků mezi serverem a klientem. Pro tento účel byl definován asynchronní, někdy také označovaný jako *streamovaný*, přenos článků. Výsledem byla aktualizace a formalizace příkazů **CHECK** a **TAKETHIS**, a vyřazení zastaralého **MODE STREAM** příkazu [7].

V roce 2010 přišlo zatím poslední rozšíření protokolu s číslem 6048 [8]. Tento RFC dokument přináší zejména vylepšení příkazu **LIST**, předtím specifikovaného v RFC 3977 [5]. Další novinkou byla aktualizace a formalizace příkazů **LIST DISTRIBUTIONS** a **LIST SUBSCRIPTIONS**, definovaných v RFC 2980 [4]. V neposlední řadě došlo k přidání příkazů **LIST COUNTS**, **LIST MODERATORS** a **LIST MOTD**. Dále byly specifikovány nové návratové hodnoty příkazu pro zjištění stavu diskuzních skupin **LIST ACTIVE** [8].

Při tvorbě této práce bylo vycházeno z norem definovaných zejména v RFC dokumentech, konkrétně se jedná především o RFC dokument 3977[5], RFC dokument 4643 [6] a RFC dokument 6048 [8].

2.3 Software pro správu

Jak je možná z kapitol výše zřetelné Usenet server je poměrně komplikovaný software, jenž by měl nejen implementovat velkou řadu výše specifikovaných služeb, ale hlavně nabízet tyto definované služby velkému množství klientům. Musí být tedy být schopen zvládat masivní síťový provoz. Ať už se jedná o vyřizování velkého množství požadavků nebo zajištění rychlé odezvy při práci s velkým množstvím binárních souborů [9].

Pro tyto účely vzniklo mnoho řešení. Mezi nejoblíbenější kompletní řešení, pro operační systémy Microsoft Windows, se v době psaní práce řadí *Dnews*, ten již bohužel není aktivně vyvíjen ani udržován. Velice populární jsou také řešení určená

pro platformu Linux. Příkladem může být volně šiřitelný *Leafnode*, *INN2* nebo novější *WendzelNNTPd*.

Pro potřeby této práce byl použit veřejný Usenet server občanského sdružení *HKFree*, jež běží na výše zmíněném Linuxovém řešení INN2.

2.4 Software pro čtení

Jak už bylo zmíněno v kapitolách výše protokol NNTP vlastně představuje klasický znakový síťový protokol, proto je pro práci s ním plně dostačující vhodný operační systém, jež podporuje terminálový příkaz TELNET.

Samozřejmě toto řešení není uživatelsky pohodlné. Z tohoto důvodu pak v minulých letech vznikla řada úspěšných i méně úspěšných softwarových řešení, jež mají sloužit zejména jako čtečka informací, které Usenet server klientům poskytuje. Často se jedná o součást jiného produktu, nejčastěji emailových klientů.

Příklady úspěšných řešeních a jejich výhody či nevýhody jsou podrobněji upřesněny v kapitole číslo **3**.

2.5 Použité příkazy protokolu

V rámci této práce bylo použito pouze malé množství z celkové množiny příkazů definovaných v použitých RFC dokumentech. Kompletní seznam použitých příkazů, použité parametry a dokument ve kterém byly poprvé představeny je uveden v tabulce níže.

Příkaz	Parametry příkazů	Definice	Úprava
GROUP	<název skupiny>	RFC 977 [2]	RFC 3977 [5]
HEAD	<identifikační kód článku>	RFC 977 [2]	RFC 3977 [5]
ARTICLE	<identifikační kód článku>	RFC 977 [2]	RFC 3977 [5]
LISTGROUP	<název skupiny>	RFC 977 [2]	RFC 3977 [5]
OVER	<číslo článku>	RFC 3977 [5]	Bez úprav
POST		RFC 977 [2]	RFC 3977 [5]
LIST		RFC 977 [2]	RFC 6048 [8]
QUIT		RFC 977 [2]	RFC 3977 [5]

Tabulka 1. Použité NNTP příkazy

2.5.1 Struktura článku

Pro úplné pochopení výše zmíněných příkazů a zejména použitých parametrů, je vhodné zmínit definici vnitřní struktury Usenet článků. Obsah této kapitoly čerpá z [10].

Každý článek by se dle, na Usenet nezávislém, RFC dokumentu 1036 [10], jež říká, jak by měla standardní struktura implementace vypadat, musí skládat ze dvou částí, hlavičky a těla. V daném dokumentu je rovněž uvedeno, že zmíněnou definici hlavičky je možné pro speciální případy implementačně upravit přidáním vlastních parametrů. To ale pouze za předpokladu, že veškeré nadstandardní parametry budou začínat předponou 'X-'.

V hlavičce článku jsou pro nás z uživatelského hlediska důležité zejména první tři parametry. První z parametrů je označován klíčovým slovem **From**. Jedná se o parametr označující identifikátor odesílatele článku. Dalším parametrem je **Newsgroups**. Ten nám říká, do jaké diskuzní skupinu námi zvolený článek patří. Posledním uživatelsky důležitým parametrem je **Subject**. Ten představuje předmět odeslaného článku. V případě odesílání článku uživatelem musí být výše zmíněné parametry povinně definovány.

Z hlediska správy a práce nad články je důležitá hlavička **Message-ID**, kterou musí každý článek povinně obsahovat. Představuje totiž jedinečný globální identifikátor článku, díky čemuž je možné zaručit, že napříč celému světu existuje pouze jeden článek s daným identifikátorem. Ačkoliv to není doporučované, tuto hlavičku je možno nastavit ručně. Ideálním způsobem je ale nechat doplnění na Usenet poskytovateli, jenž daný článek přijímá. Poslední hlavičkou, určenou zejména pro práci s články, je **Message-Number**. Ta reprezentuje číselný identifikátor článku ve zvolené diskuzní skupině. Na rozdíl od **Message-ID** je tato hlavička unikátní pouze v dané diskuzní skupině, nikoliv globálně.

V RFC dokumentu 1036 [10] je dále zmíněno mnoho dalších hlaviček, ty ale nejsou pro potřeby této práce natolik důležité jako výše zmíněné.

Tělo článku nemá oproti hlavičce žádné povinné parametry, či strukturu, je tedy možné tělo článku v jedinečných případech i vynechat. Jedná se čistě o

uživatelsky definovaný obsah článku. V nejjednodušším případě se skládá pouze z čistého nebo formátovaného textu. případě potřeby je tělo také místem k umístění volitelných binárních příloh, odesíláním binárních příloh a k tomu potřebnému kódování je vyhrazena samostatná kapitola **5.3.1.1**, v praktické části této práce.

2.5.2 Návratové stavové kódy

Abychom mohli dále s protokolem pracovat, je třeba správná interpretaci každé odpovědi Usenet serveru. Kompletní podrobný popis návratových stavů je popsán v dokumentu RFC 977 [2].

Každá odpověď na libovolný uživatelský příkaz musí za všech okolností začínat stavovým kódem. Ten je specifikován tříciferným numerickým číslem. Vzhledem k celkovému počtu definovaných NNTP příkazů jsou tři cifry dostačujícím počtem, pro pokrytí všech možných stavů, jež mohou nastat. V seznamech níže jsou vysvětleny významy částí stavových kódu.

První cifra odpovědi indikuje obecné kategorie příkazu, zejména zdali byl příkaz úspěšný, neúspěšný nebo zdali jde o pokračování k naplnění předcházejícího příkazu.

1xx. Informační zpráva
2xx. Úspěšné splnění příkazu bez pokračování
3xx. Úspěšné splnění příkazu s následným pokračováním
4xx. Příkaz byl v pořádku, ale z neznámého důvodu nebylo možné ho splnit
5xx. Daný příkaz byl buď nekorektně interpretován, nebyl implementován, nebo nastala vážná chyba programu

Druhá cifra indikuje, pod jakou konkrétní příkazovou kategorii daný příkaz spadá.

x0x. Připojení, nastavení nebo podružné informační zprávy
x1x. Zvolení diskuzních skupin
x2x. Selekce vybraného článku
x3x. Distribuční funkcionality
x4x. Odesílání článků

x8x. Nestandardní rozšíření, například soukromá implementace

x9x. Výstup v případě debuggování

Třetí cifra označuje již konkrétní stavový kód. Ten je specifický pro každý příkaz.

2.5.3 Popis použitých příkazů

Pro úplný přehled příkazů použitých v rámci praktické části této práce, je níže uveden přesný popis stavových kódu, které mohou při použití vzniknout. Pro každý příkaz je následně pomocí protokolu *TELNET* uveden názorný příklad jeho použití. Veškeré ukázky znázorňují pouze kladné vyřízení příkazu, uživatelský vstup bude zvýrazněn šedým pozadím, odpovědi Usenet serveru zůstanou bez zvýraznění.

U všech příkazu je třeba brát v potaz nutnost úspěšného připojení k požadovanému serveru, příklad úspěšného připojení je uvedeno v příkazu číslo 1.

Jako první je specifikována technologie, pomocí které budeme komunikovat, v našem případě se jedná o *TELNET*, následuje adresa serveru, může se jednat jak o doménovou adresu, tak klasickou IP adresu, příkaz je zakončen číslem síťového portu, na kterém server naslouchá. V případě úspěšného připojení je navrácen stavový kód 200.

telnet news.hkfree.org 119
200 news.gogo.stezery.hkfree.org InterNetNews NNRP server INN 2.5.3 ready (posting ok)

Příkaz 1. Úspěšné připojení k Usenetu

QUIT – příkaz je preferovaná metoda sloužící k oznámení Usenet serveru, že klient dokončil veškeré požadované transakce. Definován byl v RFC dokumentu 977 [2] a odpovědí serveru by měl za každého případu být číselný návratový kód 205.

QUIT
205 Bye!

Příkaz 2. Ukončení spojení s Usenetem

GROUP – příkaz definovaný již v RFC dokumentu 977 [2], jako parametr očekává název diskuzní skupiny a slouží k označení dané skupiny jako používané. Návratovou hodnotou v případě úspěšného pokusu je návratový kód 211. Následován je souhrn informací o dané diskuzní skupině. Informace obsahují: návratový kód, celkový počet článků, číslo prvního článku, číslo posledního článku a název skupiny.

GROUP hkfree.test
211 679 3628 4347 hkfree.test

Příkaz 3. Výpis informací o dané diskuzní skupině

HEAD – příkaz je definovaný jako implementačně povinný. Všechny dostupné implementace Usenet serveru by jej měli jakýmkoliv způsobem poskytovat [2]. Příkaz samotný slouží k získání veškerých informací v hlavičce jednoho článku, použití je možné třemi formami uvedenými níže.

1. Zaslání příkazového slova, následovaného globálně jedinečným identifikačním kódem článků. Pokud není daný článek nalezen je navrácen chybový stav 430.
2. Zaslání příkazového slova následovaného číslem článku jedinečného pouze v aktuálně zvolené diskuzní skupině. V případě nezvolené diskuzní skupiny je navrácen chybový stav 412. Pokud článek s daným číslem v dané diskuzní skupině neexistuje je navrácen chybový stav 423.
3. Posledním způsobem je zaslání pouze příkazového slova. V tomto případě je navrácena hlavička aktuálně zvoleného článku, ve zvolené diskuzní skupině. Pokud není zvolena žádná diskuzní skupina je navrácen chybový stav 412. V případě, že není zvolen článek je navrácen chybový stav 420.

V případě úspěšného použití příkazu je navrácen stavový kód 221, následován hlavičkou článku.

HEAD 4348
221 4348 <q47b7i$j8e$1@news.gogo.stezery.hkfree.org> head Path: news.gogo.stezery.hkfree.org! .POSTED!not-for-mail From: “test” <test@test.cz> Newsgroups: hkfree.test Subject: test Date: Fri, 15 Feb 2019 21:33:38 +0000 (UTC) Organization: HKFree Lines: 1 Message-ID: <q47b7i$j8e$1@news.gogo.stezery.hkfree.org> NNTP-Posting-Host: 127.0.0.1 X-Trace: news.gogo.stezery.hkfree.org 1550266418 19726 127.0.0.1 (15 Feb 2019 21:33:38 GTM) X-Complaints-To: <usenet@news.gogo.stezery.hkfree.org> NNTP-Posting-Date: Fri, 15 Feb 2019 21:33:38 +0000 (UTC) Xref: news.gogo.stezery.hkfree.org hkfree.test:4348 .

Příkaz 4. Výpis hlavičky vybraného článku

ARTICLE – slouží k získání jak hlavičky, tak těla článku. Obě informace musí být dle definice v RFC dokumentu 977 [2] odděleny prázdným řádkem. Příkaz má podobně jako příkaz **HEAD,** tři identické formy použití. Na rozdíl od příkazu **HEAD**, ale není implementačně povinný.

1. Forma očekává zaslání příkazového slova následováno globálně jedinečným identifikačním kódem článků. Když není daný článek nalezen je navrácen chybový stav 430.
2. Forma očekává příkazové slovo následováno číslem článku jedinečného v aktuálně zvolené diskuzní skupině. V případě že není zvolena skupina je navrácen chybový stav 412. Naopak neexistuje-li v dané diskuzní skupině číslo článku je navrácen stav 423.

3. Forma potřebuje pouze příkazové slovo. Pro identifikaci článku je použita aktuálně zvolena diskuzní skupina a článek. V případě nezvolené diskuzní skupiny je navrácen stav 412. Naopak není-li zvolen článek je odpovědí chybový stav 420.

V kladně vyřízeném požadavku je navrácen stavový kód 220 následován hlavičkou a tělem článku.

ARTICLE 4348
220 4348 <q47b7i$j8e$1@news.gogo.stezery.hkfree.org> article Path: news.gogo.stezery.hkfree.org! .POSTED!not-for-mail From: “test” <test@test.cz> Newsgroups: hkfree.test Subject: test Date: Fri, 15 Feb 2019 21:33:38 +0000 (UTC) Organization: HKFree Lines: 1 Message-ID: <q47b7i$j8e$1@news.gogo.stezery.hkfree.org> NNTP-Posting-Host: 127.0.0.1 X-Trace: news.gogo.stezery.hkfree.org 1550266418 19726 127.0.0.1 (15 Feb 2019 21:33:38 GTM) X-Complaints-To: <usenet@news.gogo.stezery.hkfree.org> NNTP-Posting-Date: Fri, 15 Feb 2019 21:33:38 +0000 (UTC) Xref: news.gogo.stezery.hkfree.org hkfree.test:4348 Testovací tělo článku .

Příkaz 5. Výpis hlavičky a těla vybraného článku

LISTGROUP – je vylepšením příkazu **GROUP**. Parametrem je tedy obdobně název požadované diskuzní skupiny, ale na rozdíl od příkazu **GROUP** je návratovou odpovědí kolekce číselných identifikátorů všech dostupných článku dané skupiny

[5]. Kolekce je v případě úspěšného požadavku následována stavovým kódem 211, v opačném případě je navrácen chybový stav 411. Nastane-li situace, že zvolená diskuzní skupina neexistuje je navrácen chybový stav 412.

LISTGROUP hkfree.test
211 680 3628 4348 hkfree.test 3628 3629 3630 …… 4346 4347 4348

Příkaz 6. Výpis číselných identifikátorů článku vybrané diskuzní skupiny

OVER – je příkaz specifikován v novějším RFC dokumentu 3977 [5]. Návratovou hodnotou je obsah všech informačních polí daného článku v databázi Usenet poskytovatele. Příkaz lze použít jednou ze tří forem popsaných níže. Pro použití každé z forem je potřeba mít vybranou libovolnou diskuzní skupinu. Pokud není diskuzní skupina zvolena je navrácen chybový stav 412.

1. formou je použití příkazového slova následováno globálně jedinečným identifikačním kódem článků. Použití této formy je implementačně volitelné. Pokud daný Usenet tuto formu neposkytuje, musí být při použití navrácen generický chybový stavový kód 503.
2. formou je použití rozsahu čísel článků dané skupiny. Nastane-li situace, že neexistuje žádný článek v daném rozsahu je navrácen chybový stavový kód 423. Parametr rozsahu může obsahovat jednu ze tří možností.
    1. Pouze číslo článku. Návratovou hodnotou je informace pouze o jediném článku.

2. Číslo článku následováno pouze pomlčkou, která definuje zvolení všech následujících čísel článků.
3. Číslo článku následováno pomlčkou, kterou následuje další číslo článku.

3. forma je použita v případech, když příkaz není následován žádným doplňujícím parametrem. Jako parametr se automaticky zvolí číslo aktuálně zvoleného článku. Pokud není žádný článek zvolen je odpovědí chybový stav 420.

Úspěšný požadavek je identifikován, množinou informací daného článku. V případě použití rozsahu kolekce článků, je navrácen stavový kód 224.

GROUP hkfree.test
211 680 3628 4348 hkfree.test
OVER 4348
224 Overview information for 4348 follows 4348 test "test" <test@test.cz> Fri, 15 Feb 2019 21:33:38 +0000 (UTC) <q47b7i$j8e$1@news.gogo.stezery.hkfree.org> 566 1 Xref: news.gogo.stezery.hkfree.org hkfree.test:4348 .

Příkaz 7. Výpis informačních polí článku pomocí číselného identifikátoru

POST – slouží k odeslání nových článku Usenet poskytovateli. Příkaz je jedinečným v tom smyslu, že se na rozdíl od příkazů předešlých skládá ze dvou na sobě navazujících částí. První část slouží k ověření, zdali daný Usenet poskytovatel podporuje příjem uživatelských článků. Ověření probíhá zasláním příkazového slova **POST**. Návratovou hodnotou musí bez výjimky být stavový kód 340. Daný kód označuje schopnost Usenet serveru přijímat uživatelské články. Pokud daný poskytovatel tuto funkcionalitu neposkytuje je vrácen chybový stavový kód 440, značící neschopnost přijímat články. Důvod může být buď implementačního nebo organizačního charakteru. Druhá část přímo navazuje na tu první, a je podmíněna jejím úspěšným provedením. Obsahem požadavku je již samotný uživatelův článek.

Ten musí splňovat požadovaný formát dat, tedy blok více řádkových informací. Každý řádek musí být povinně oddělen tečkou. Bezproblémové přijetí dat je oznámeno návratovým stavem 240. V případě opačném chybovým stavem 441 [5].

POST
340 Input article; end with <CR-LF>.<CR-LF>
From: “test” <test@test.cz> Newsgroups: hkfree.test Subject: Test article Organization: hkfree.org Testovací článek .
240 Article received OK

Příkaz 8. Odeslání nového článku

LIST – příkaz slouží k výpisu všech dostupných diskuzních skupin daného Usenetu. Návratovou hodnotou je v kladném případě návratový stav 215, následovaný více řádkovým blokem dat. Každý řádek obsahuje informace o jedné dostupné diskuzní skupině. V záporném případě je navrácen chybový stav 432 [8].

LIST
215 Newsgroups in form “group high low status” hkfree.sit 000025544 21661 y hkfree.games 000012569 12354 y ……. hkfree.bazar 000044767 25807 y

Příkaz 9. Výpis všech dostupných diskuzních skupin

3 Rešerše existujících klientů

V následující části se autor pokusí popsat vlastní zkušenosti s používáním jednotlivých Usenet klientů a jednotlivá řešení objektivně zhodnotit. V potaz budou brány momentálně nejpoužívanější platformy. Za klasické stolní počítače a notebooky, se bude jednat o libovolnou Linux distribuci, Apple OSX nebo Microsoft Windows. U mobilních zařízeních se bude jednat pouze o zařízení poháněná operačním systémem Android nebo iOS. Klienti byly zvoleni dle oblíbenosti případně dle počtu stažení mezi uživateli jednotlivých platforem.

3.1 Mozilla thunderbird

Jedná se volně šiřitelné více platformní desktop řešení poštovního klienta, od společnosti Mozilla. Spatřilo světlo světa v roce 2013, a v prvních deseti dnech bylo staženo přes milion kopií [11]. Jedná se o výchozí poštovní klient v Linuxové distribuci Ubuntu. Nejaktuálnější verze v době psaní práce je 60.3.3.

Ačkoliv se jedná zejména o poštovního klienta, jeho součástí je také news feed (RSS), chatovací služby (XMPP, IRC, Twitter) nebo plnohodnotná Usenet čtečka.

Nespornou výhodou aplikace Thunderbird je možnost připojení k neomezenému množství Usenetů zároveň. Čtení je umožněno odebíráním libovolných diskuzních skupin daného serveru. Jednotlivé články dané skupiny jsou následně zobrazeny v přehledné stromové struktuře. Klient umožňuje odpovídat na články, pouze formou čistého textu. Samozřejmostí je zobrazení a odesílání binárních příloh. Veškeré články odebíraných skupin je možné uložit do lokálního paměťového média pro další použití. Bonusem je možnost archivace článků, a jejich označení pomocí příznaků.

3.1.1 Klady

- Možnost označit článek pomocí poskytnutých nebo vlastních příznaků
- Zobrazení binárních příloh
- Efektivní filtrování dle hlaviček
- Možnost ukládání článků offline

3.1.2 Zápory

- Možnost odpovědi pouze pomocí čistého textu
- Pro neznalé uživatele obtížnější nastavení poskytovatele a odebírání článků

3.1.3 Shrnutí

Ačkoliv se jedná v první řadě o plnohodnotného poštovního klienta, funkcionálně nabízí vše potřebné k pohodlné četbě Newsgroup článků a prací nad nimi, ať už se jedna o archivací jedním kliknutím nebo bleskurychlé vyhledávání článku dle informací v hlavičce. Ze všech dostupných řešení považuje autor práce Thunderbird za jasného favorita.

3.2 NewsGroup Reader

Je nativní aplikace určená pouze pro mobilní zařízení s operačním systémem android. Vydána byla na platformě Google Play v roce 2013. K dispozici je jak neplacená, tak placená verze, která oproti verzi neplacené neobsahuje reklamu a přidává řadu užitečných funkcí. Poslední aktualizace s číslem 1.65 přišla v roce 2014.

Aplikace již v neplacené verzi nabízí veškerou funkcionalitu potřebnou pro práci s Usenety. Samozřejmostí je podpora připojení k více než jednomu Usenetu, odebírání diskuzních skupin, pokročilé funkce pro filtrování článků nebo možnost ukládání článku pro jejich čtení i bez internetového připojení.

3.2.1 Klady

- Již v neplacené verzi je nabízena veškerá potřebná funkcionalita
- Pokročilé možnosti filtrování přijatých článků
- Offline režim

3.2.2 Zápory

- Zastaralý a uživatelsky nepřívětivý vzhled
- Momentálně bez další podpory ze strany výrobce

3.2.3 Shrnutí

Plnohodnotná aplikace, která i v bezplatné verzi nabídne vše potřebné. Vzhledem k ukončené aplikační podpoře není na novějších verzích operačního systému Android zaručena správná funkčnost. Jedná se o dostačující řešení zejména pro starší zařízení.

3.3 PhoNews Newsgroup Reader

Je dalším nativním mobilním klientem pro operační systém Android. Poslední aktualizace s číslem 2.6 byla vydána v roce 2016.

Obdobně jako předešlá aplikace nabízí PhoNews jak placenou, tak bezplatnou verzi. Neplacená verze nabízí pouze základní funkce jako je připojení k více poskytovatelům nebo offline režim. Pro možnost odesílání článku je zapotřebí přejít na verzi placenou. Všudypřítomné reklamy v neplacené verzi k celkové uživatelské přívětivosti nepřidávají. Příjemnou nadstandardní funkcionalitou naopak je možnost propojení s cloud účty jako je Dropbox nebo Google Drive a následné volitelné ukládání binárních příloh na daná úložiště.

3.3.1 Klady

- Možnost ukládání binárních příloh do cloudových úložišť
- Intuitivní ovládání díky modernímu uživatelskému rozhraní

3.3.2 Zápory

- Nemožnost odesílání článků v bezplatné verzi
- V bezplatné verzi velmi vysoké množství reklam
- Na méně výkonných zařízeních dochází k zamrzání

3.3.3 Shrnutí

Díky použití Google Material vzhledu působí aplikace na první pohled moderních dojmem. Hlavní předností je zejména podpora Cloud úložišť, ale pro plnohodnotnou práci je třeba zainvestovat do placené verze. Aplikace v době psaní této práce, podporuje i poslední verzi operačního systému Android 8.

3.4 NewsTap Lite

Je Usenet čtečka určená pro mobilní zařízení Apple iPhone, iPad a iPod Touch. Jedná se o bezplatnou verzi populární aplikace NewsTap.

V základní verzi nabízí NewsTap Lite pokročilé funkcionality jako je podpora offline režimu, rozsáhlá podpora binárních příloh, podpora filtrování na základě příznaků nebo uživatelsky intuitivní stromový pohled na jednotlivé články vybrané diskuze. Hlavním nedostatkem bezplatné verze je podpora připojení pouze k jednomu Usenet poskytovateli a možnost odebírat nanejvýš dvě diskuzní skupiny.

Aplikace je nadále aktivně vyvíjena.

3.4.1 Klady

- Podpora zobrazování binárních příloh
- Uživatelsky přívětivé UI
- Možnost zobrazení komunikace se serverem pomocí prohlížení logu
- Velké možnosti nastavení
- Možnosti ukládání článků na iCloud

3.4.2 Zápory

- Bezplatná verze podporuje pouze jednoho Usenet poskytovatele

3.4.3 Shrnutí

Aplikace díky velkému množství nastavení nabízí velkou svobodu individuálních úprav. Společně s uživatelský přívětivým vzhledem a podporou Apple iCloud se jedná o jednoho z nejlepších klientů pro zařízení s operačním systémem iOS. Velice zajímavou funkcí je pak také možnost nahlédnutí do logu komunikace zařízení s Usenetem, a to včetně zobrazení použitých příkazů.

3.5 Newsy

Jsou poslední probíranou aplikací určenou pro mobilní zařízení od společnosti Apple s operačním systémem iOS. Tato aplikace je nadále aktivně vyvíjena vývojářem Arnoldem Tangem.

Oproti konkurenčním aplikacím byl při vývoji této aplikace brán potaz na vysokou oblíbenost sociálních síti. Hned v základu je v aplikaci přednastavena množina nejoblíbenějších Usenetů, na které je pohlíženo jako na zdroj témat, které by mohli uživatele zajímat a diskutovat o nich. K tomu samozřejmě patří jak podpora binárních příloh, možnost zobrazení GIF animací nebo hromadné odeslání až 20 binárních souborů v případě podpory Usenetem. Oproti předešlým aplikací nebylo zanedbáno ani uživatelské rozhraní, které je inspirováno moderními trendy. Například klasická zobrazení pomocí stromové struktury bylo nahrazeno moderním dlaždicovým pohledem. V případě preference starší stromové struktury je možnost jejího přepnutí v nastavení. V rámci následování moderních trendů je pak samozřejmostí podpora nativních notifikací na nové články v uživatelem sledovaných diskuzích.

3.5.1 Klady

- Moderní vzhled
- Možnost přepínání mezi stromovým nebo dlaždicovým pohledem
- Nativní podpora notifikací

3.5.2 Zápory

- Nekonzistentní uživatelské rozhraní na různých rozlišeních
- Oproti konkurenci velice drahé možnosti vylepšení základní verze
- Vysoké množství reklam

3.5.3 Shrnutí

Ze všech zástupců se jedná o první skutečně moderně pojatou Usenet čtečku, pomocí začlenění prvků spíše známých ze světa sociálních sítí. Největší výhodou oproti veškeré konkurenci je velice moderní a intuitivní vzhled. Bohužel aplikace kromě velice drahých přídavných funkcí, trpí v době psaní práce řadou menších technických neduhů, na novějších verzích operačního systému iOS, testováno na verzi 12.2. Ty mohou práci s aplikací značně znepříjemnit.

3.6 Celkové shrnutí

Ačkoliv většina aplikací nabízí veškerou funkcionalitu, kterou uživatel při práci s Usenety potřebuje. Většinou se jedná o placené aplikace. Těm vzhledem k jejich většinové zastaralosti, chybí kromě moderního a přístupného uživatelského rozhraní také v dnešní době řada moderních a pro mnoho uživatelů nepostradatelných prvků. Příkladem je podpora nativních notifikací nebo synchronizace uživatelského účtu na více zařízeních. Všechny tyto funkcionality by měly být součástí výstupu praktické části této práce, jehož použití bude volně šiřitelné zdarma.

4 Nástroje

Vzhledem ke zkušenostem z kapitoly výše byla identifikace požadavků, jež by měla praktická část práce implementovat, velice jednoduchá. Jednou z hlavních funkcionalit by měla být možnost použití aplikace na co největším množstvím uživatelských platforem. Finálním, rozhodnutím byla implementace webové jedno stránkové aplikace.

Protože se převážná část webových aplikací skládá ze dvou částí, serverové a klientské, tak ani tato práce není výjimkou. Byla proto zapotřebí identifikace nástrojů, jež umožní rychlý a snadný vývoj, jak na prostředí serveru, tak na straně klienta.

Pro realizaci chování na straně serveru byl použit vývojový framework *Koa* běžící v runtime prostředí Node.js. Na straně klienta byla pro tvorbu uživatelského rozhraní použita JavaScript knihovna *React*. Pro potřeby ukládání uživatelských dat napříč uživatelovými zařízeními byl použit relační databázový systém SQLite. Snadné nasazení finální aplikace na velkém množství produkčních strojů má na starosti kontejnerová technologie Docker.

Veškeré zde zmíněné nástroje budou podrobněji popsány v této kapitole.

4.1 JavaScript

Je více platformní, interpretovaný nebo JIT kompilovaný programovací jazyk využívající takzvaných funkcí první třídy. Nejvíce známý je jako skriptovací jazyk pro webové stránky, ale také pro mnoho ne-webových prostředí. Například stále populárnější běhové prostředí Node.js nebo na poli databází produkt Apache CouchDB. Jako více paradigmatický, dynamický skriptovací jazyk, podporuje imperativní, objektově-orientovaný, prototypovací a funkcionální programovací přístup [12].

Hlavním výhodou JavaScriptu, zejména při běhu ve webovém prohlížeči, je zejména jeho rychlost a jednoduchost. Další nespornou výhodou je obrovská popularita a vysoké množství podpůrných knihoven, které zefektivní a zlevní vývoj budoucí aplikace. Na druhou stranu jsou tyto výhody vykoupeny náchylností na bezpečností díry, protože se jedná o kód běžící v přístroji uživatele. Tomu nepomáhá

ani rozdílná interpretace stejných částí kódu v různých webových prohlížečích, což může vést k dalším nežádoucím účinkům.

JavaScript vytvořil Brendan Eich v roce 1995 v době jeho působení v tehdejší firmě Netscape Communications. Inspiraci čerpal u programovacích jazyků jako je Java, Scheme nebo Self [13].

4.2 React

Je volně šiřitelná JavaScript knihovna vydaná v květnu roku 2013 zaměstnancem společnosti Facebook, Jordanem Walkem. React byl vyvinut za účelem snadné a rychlé tvorby uživatelských rozhraní pro webové a mobilní aplikace. Přestože je React poměrně mladou technologií v době psaní práce se, co se počtu stažení týče, jedná o jednu z globálně nejpopulárnější JavaScript knihovnu.

Základním konceptem aplikací napsaných pomocí knihovny Reactu je znuvupoužitelnost. Uživatelské rozhraní vzniká skládáním reaktivních prvků zvaných jako komponenty. Ty zapouzdřují veškerou aplikační logiku a jejich skládáním můžeme docílit rychlého vývoje komplexních uživatelských rozhraních. Vysoký výkon je zajištěn utilizací takzvaného virtuálního DOMu [14].

V současnosti je React nadále udržován jak samotnou společností Facebook, tak i komunitou nezávislých vývojářů, nebo jinými technologickými firmami. V době psaní práce je nejaktuálnější stabilní verze 16.8.3. Což je také verze, kterou využívá klientská aplikace.

4.3 Node.js

Je opět volně šiřitelné více platformní běhové prostředí postavené na JavaScript enginu V8. Totožný engine využívá pro svou činnost také internetový prohlížeč Google Chrome. Byl vytvořen Ryanem Dahlem v roce 2019. Použitím je umožněn rychlý a škálovatelný vývoj na serveru běžících aplikací a dynamických webových stránek pouze s pomocí programovacího jazyka JavaScript.

Základní stavebním prvkem platformy Node.js je použití takzvané *event-driven* architektury, a *non-blocking* I/O modelu [15].

Aktuální verze v době psaní bakalářské práce je 11.4.0. Pro potřeby této práce byla použita starší stabilní verze 10.14.2.

4.3.1 Event-driven

Základním konceptem *event-driven* architektury je podpora paralelního zpracování procesů pomocí konceptu zasílání zpráv takzvaných *eventů* a jejich zpětného volání pomocí speciálních funkcí, takzvaných *callbacků*. Této funkcionality je dosaženo použitím *event-loop* smyčky, která pro dané zprávy naslouchá a při jejich zaznamenání je schopna pomocí zpětného volání reagovat a daný proces bleskově obsloužit [16]. To vše je díky *non-blocking* modelu možné při použití pouze jednoho vlákna procesoru.

Blocking model, znamená, že než je jakýkoliv proces obsloužen, musí čekat na dokončení veškerých operací předchozího procesu. Na druhou stranu *non-blocking* model, nečeká na vyřízení operací. Místo toho začne obsluhovat další proces v pořadí. Jakmile jsou všechny operace prvního procesu vyřízeny, dojde k pokračování pozastavených procesů. Díky tomuto přístupu je možné *paralelně* zpracovávat více jak jeden proces zároveň [17].

4.4 Koa

Je expresivní Framework postavený na JavaScript specifikaci ECMAScript 6. Jeho hlavním cílem je agilní a rychlá tvorba serverových webových aplikací nebo rozhraní běžících na platformě Node.js. V základu jsou součástí frameworku pouze základní funkcionality, které je možné dále rozšířit použitím konceptu takzvaných *middleware* funkcí.

4.4.1 Middleware

Middleware je v kontextu Koa frameworku běžná či asynchronní JavaScript funkce, která může provést jakoukoliv operaci, ať už se jedná o I/O požadavek, či obyčejné logování do konzole. Důležité je, že po každém provedení operace, předá řízení dalšímu *middlewaru* v pořadí. Tento koncept funguje oboustranně je tedy možné předávat řízení směrem dolů či probublat nahoru. Tímto způsobem je umožněno posílání dat mezi *middlewary*, a jejich následnému zpracování.

4.5 Docker

Je podobně jako ostatní nástroje volně šiřitelný nástroj sloužící vývojářům k usnadnění tvorby, nasazení a spouštění aplikací, bez ohledu na hardwarové rozdíly mezi vývojovým a produkčním prostředním za pomoci takzvaných kontejnerů.

Principiálně může nástroj *docker* připomínat virtuální stroj. Na rozdíl od něj, ale nevytváří *docker* celý virtuální operační systém se všemi jeho částmi, nýbrž využívá pouze jeho kernel, což znamená menší hardwarovou zátěž na samotný fyzický stroj.

Docker vyvinul v roce 2013 Solomon Hykes, jako interní projekt společnosti dotCloud.Jako volně přístupný byl Docker spuštěn 13. března téže roku [18].

4.5.1 Kontejner

Kontejner je možné si představit jako obal, do kterého jsme schopni zabalit aplikaci společně se všemi jejími částmi, ať už se jedná o knihovny, konfigurační soubory případně jakékoliv jiné závislosti.

Kontejnery jsou vytvářeny pomocí takzvaných *obrazů*, které přesně specifikují jejich obsah. Obrazy jsou často vytvářeny kombinací a modifikací standardních obrazů stažených z veřejných repositářů [18].

4.6 SQLite

Je více platformní relační databázový systém. Na rozdíl od konkurenčních databázových systému nejedná se o typ databáze klient-server, databáze tedy žádným způsobem nekomunikuje s aplikací, ale jedná se o její součást. Čtení a ukládání dat je následně prováděno buď přímo do systémového souboru nebo operační paměti. Následně jsou všechny databázové informace o tabulkách, indexech, spínačích a pohledech uchovány přímo v daném aplikačním databázovém souboru [19].

Mezi hlavní výhody *SQLite* se řadí například možnost paralelního čtení více něž jednou instancí nebo prakticky nulová nutnost počáteční konfigurace.

Autorem SQLite je D. Richard Hipp. První verze byla vydána v roce 2000 pod veřejnou doménovou licencí [19]. Pro účel této práce byla použita stabilní verze 3.26.0

5 Návrh a implementace řešení

Jakákoliv práce na softwarovém produktu by měla v ideálním případě zahrnovat alespoň tři části, analýzu, vývoj, a testování. Díky tomu je možné se na budoucí produkt dívat z více úhlů. Správná a důkladně promyšlená analýza by měla počítat se všemi možnými případy použití, ale zároveň být dostatečně flexibilní na případná budoucí vylepšení. Chybná analýza může naopak, zejména při vývoji, vést k řadě špatných rozhodnutích, které pak výrazně prodlouží vývoj produktu a znemožní budoucí efektivní testování. Důkladné analýze a promyšlení finální implementace byla v této práci věnovaná nemalá část času.

5.1 Návrh produktu

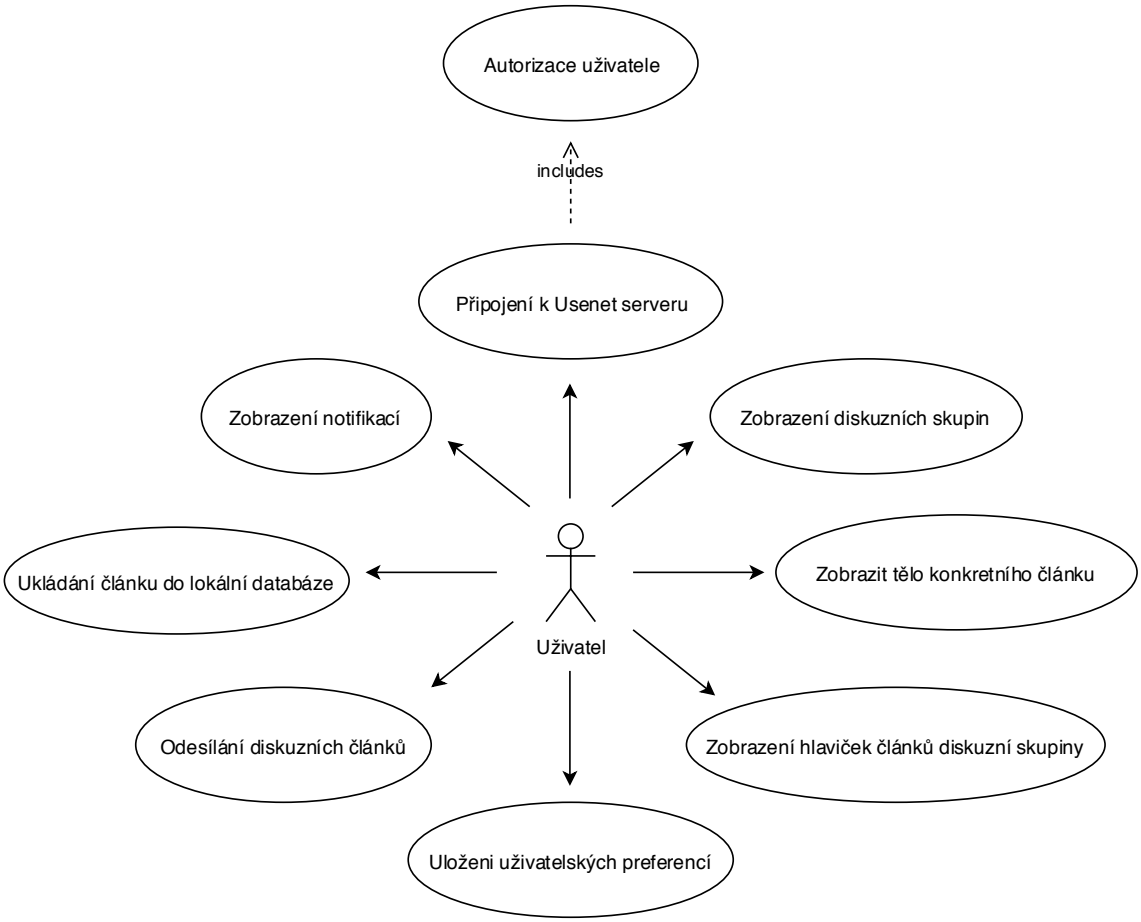
Návrh produktu má jeden jasný úkol, definici požadavků budoucích uživatelů produktu a převedení do takové podoby, aby byl vývoj jednoduchý a zároveň efektivní. Je tedy nutné ujasnit detaily daného softwaru, definovat a ideálně vytyčit množinu funkcionalit jež bude produkt uživatelům nabízet. Příkladem mohou být možná budoucí vylepšení nebo naopak jaké části aplikace se jeví jako přebytečná.

Finální výčet funkcionalit v našem případě vyplívá zejména z praktických zkušeností autora práce s jinými Usenet klienty a samotnou filozofií protokolu NNTP. Hlavním cílem praktické části této práce je vytvoření co nejvíce stabilního jádra produktu, které zajistí veškeré základní funkce pro příjemnou práci s Usenety, a následně pomocí, pokud možno dále nezávislých, modulů přidávat jednotlivé nadstandardní funkcionality.

Finálním výsledkem této analýzy je následující množina požadovaných a následně nadstandardních funkcionalit, které by měla finální verze produktu uživateli poskytnout.

- Jádro produktu
    1. Připojení ke zvolenému Usenet serveru, včetně autorizace
    2. Stažení a zobrazení veškerých diskuzních skupin pro danou autorizaci
    3. Stažení a zobrazení hlaviček článků zvolené diskuzní skupiny
    4. Zobrazení zvoleného článku, včetně binárních příloh
    5. Možnost odpovědi na libovolný článek, včetně přiložení binárních příloh

- Nadstandardní funkcionalita
    1. Uložení zvoleného článku do lokální databáze pro umožnění offline režimu
    2. Uložení uživatelských meta-dat určených k synchronizaci
    3. Upozornění na nové události pomocí notifikací
    4. Podpora většiny uživatelských zařízeních, podporující webový prohlížeč



Obrázek 1. Diagram požadovaných funkcionalit

5.1.1 Hranice systému

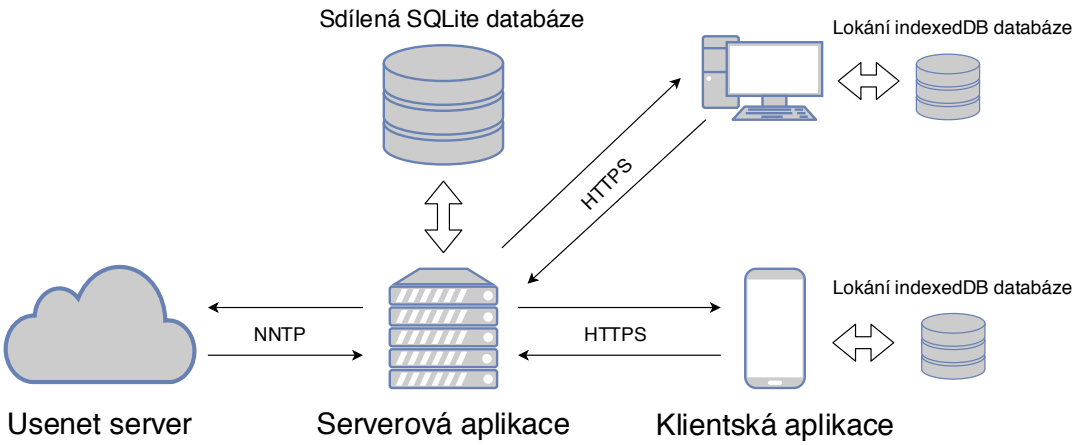
Ještě, než začneme s popisem jednotlivých částí výsledné aplikace je třeba jasně a přehledně definovat a popsat hranice, jakých může aplikace nabývat.

Usenet uzlem může být většina moderních Usenet implementací, jedním z příkladů kompatibilních poskytovatelů za linuxové prostředí je *INN2*.

Hlavním aplikačním uzlem je serverová aplikace, ta má na starosti dva hlavní úkoly. První a klíčový úkol je zprostředkování komunikace s Usenetem, to vše za pomoci síťové komunikace prostřednictvím protokolu NNTP. Druhým úkolem je zajištění autorizace uživatelů a ukládání jejich meta-dat do sdílené relační SQLite databáze.

Klientským uzlem je takzvaná *jedno stránková* aplikace, běžící ve webovém prohlížeči uživatele, jejím účelem je zobrazení dat získaných ze serverové aplikace pomocí síťového protokolu **HTTPS**, aplikace by měla být schopna tato data ukládat do lokální NoSQL *indexedDB* databáze přímo ve webovém prohlížeči.

Hranice systému jsou graficky znázorněny v diagramu níže.



Obrázek 2. Grafická reprezentace hranic systému

5.2 Klientský program

Klientský program je, jak bylo už výše zmíněno, *jedno stránková* webová aplikace, vytvořená pomocí JavaScript knihovny React. Základní kostra klientské aplikace byla vytvořena pomocí veřejně dostupného konzolové skriptu figurujícího pod názvem *create-react-app*. Tato kostra v základu nabízí konzolový příkaz ***eject***, jež uvolní veškeré konfigurační soubory pro jejich následnou úpravu. Tento příkaz byl vzápětí použit, aby byly umožněny případné budoucí nadstandardní úpravy. Příkladem může být konfigurace balíčkového nástroje *Webpack*, s jehož pomocí je možné spustit nejen vytvořit vývojové prostředí aplikace, ale také aplikaci zabalit a docílit menšího objemu pro následné produkční použití.

5.2.1 Analýza uživatelské aplikace

Před začátkem vývoje jakékoliv aplikace založené na knihovně React je potřeba nejprve vyjasnit a jasně definovat finálního hranice.

První věcí, na kterou je třeba se při vývoji zaměřit, je jakým způsobem bude řešen globální stav aplikace, případně jakým způsobem k němu bude přistupováno. Jedním z možných řešeních a v praxi dosti zneužívaných, je použití specializovaných knihoven, například industriálně oblíbená knihovna jménem *Redux*. Bohužel většina knihoven s sebou kromě zbytečných závislostí, jež zvyšují celkovou paměťovou náročnost aplikace, přináší také značnou režii, co se logiky finálního produktu týče. Od této možnosti bylo hned ze začátku práce opuštěno. Další možností je držení stavu uvnitř samotných komponent. Zde ale nastávají problémy u větších aplikací, a věci jako kolování vlastností napříč několika větvemi komponent se z programátorského hlediska stává, bez nadsázky, noční můrou. Posledním, a zároveň využívaným, řešením bylo využít relativně nový způsob správy vnitřního stavu, který je od verze 16.6 zabudován přímo v samotné knihovně React. Řeč je o takzvané *Context API*, což je rozhraní fungující na principu takzvaných *Providerů*, tedy komponent, které svůj vnitřní stav nabízí ostatním, a *Consumerů*, komponentách, které daný stav čerpají pro své vlastní potřeby. Výhodou tohoto způsobu je, že ke zdrojům jednotlivých *Providerů* je možné přistoupit ve kterékoliv komponentě bez ohledu na celkové zanoření ve finální stromové aplikační struktuře.

Dále je nutné vymyslet jakým způsobem bude vyřešena práce s takzvanými *side-effecty*, tedy práce s asynchronním kódem uvnitř našich synchronních komponent. Pro toto řešení nám opět samotný React ve verzi 16.8 přinesl funkcionalitu zvanou *Hooks* a speciální komponentu *Suspense*, kde pomocí přibalené funkce *useEffect* byla značně usnadněna práce se vzdálenými zdroji dat. Pomocí výše zmíněné komponenty s názvem *Suspense* bylo umožněno vykreslování komponent asynchronních, tedy těch, které jsou závislé na vzdálených datech, protože bez nich by nemělo dané komponenty smysl zobrazovat.

Poslední věcí, nad kterou je třeba se zamyslet je jakým způsobem budou samotné uživatelské komponenty tvořeny a ze kterých bude ve finále složeno

kompletní uživatelské rozhraní. Opět existuje více cest, kterými je možné se vydat, jednou z možností je použít již jednu z mnoha knihoven s již hotovými komponenty, včetně vyřešeného stylování, příkladem může být například jedna velice populární knihovna s názvem *material-ui*. Bohužel v době, kdy došlo k začátku vývoje klientské aplikace nebyla knihovna ve stavu pro použití pro produkci, byla tedy zvolena cesta vlastního řešení se všemi potřebnými obecnými komponentami. Výsledkem této snahy je knihovna, která je momentálně nabízena v rámci MIT licence na veřejně přístupném portálu s balíčky *npm*. Knihovnu lze nalézt pod názvem *Bachelor-ui*. Dále více o této knihovně v následují kapitole.

5.2.2 Použité komponenty

Jak již byla zmíněno v předešlé kapitole, pro stavbu uživatelského rozhraní byla použita vlastní implementace jak veškerých obecných komponent, tak komponent specifických pouze pro výslednou aplikaci.

Pro stylování komponent byl využit způsob, který se v posledních letech těší čím dál větší oblibě, řeč je o takzvaném *css-in-js*, tedy řešení, kde mají veškeré komponenty svůj vlastní nezávislý styl, ve většině případu se dané stylování nalézá v jednom souboru společně s výslednou komponentou. Pro toto řešení byla využita pomocná knihovna *styled-components* jež značně urychluje vývoj zmíněným způsobem a zároveň nabízí řadu pokročilých funkcí, například možnost změny motivu již vykreslených komponent. Této funkcionality využívá například funkce temného módu finálního klientského produktu, kdy je při pouhé změně komponenty spínače změněn vzhled již vykreslené a běžící aplikace buď na tmavý nebo světlý motiv.

Veškeré vlastní komponenty jsou implementovány tak, aby splňovaly alespoň základní prvky přístupností pro většinu uživatelů, ale také, aby poskytovaly velice pestré rozhraní. To umožní budoucím vývojářům rychlou změnu chování či vzhledu komponenty dle všech potřebných uživatelských požadavků. Knihovnu je tedy bez problémů možné použít i v dalších, na této práci nezávislých, projektech.

5.2.3 Mobilního pohled

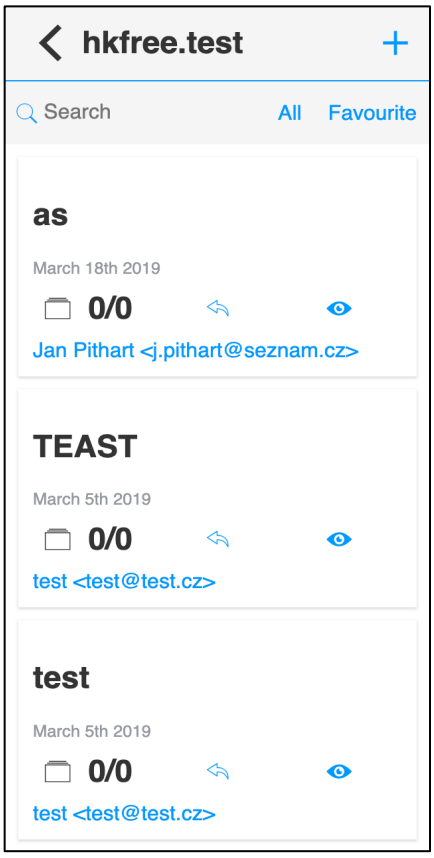
Rozložení mobilní verze aplikace je složeno z několika obrazovek, z nichž každá má svou specifickou funkcionalitu a vykresluje jednu z hlavních zobrazovacích aplikačních komponent.

Úvodní stránka je vykreslena pomocí komponenty ***Navigation***, ta se v první řadě zabývá vykreslením seznamu oblíbených diskuzních skupin uživatele. V případě potřeby sledování nové diskuzní skupiny lze ve spodní částí listu stisknout tlačítko s nápisem *more*, následně dojde k zobrazení všech dostupných diskuzních skupin, ze kterých si uživatel může vybrán přesně tu, která ho zajímá a kliknutím ji přidat mezi oblíbené. Odebrání skupin probíhá po kliknutí na textový řetězec *edit*, po stisku dojde k zobrazení ikony odstranění u detailu každé skupiny. Komponenta slouží také jako přístup do nastavení aplikace, k němu dojde při stisku ikony ozubeného kola v pravém horním rohu.

Obrázek 3. Úvodní obrazovka mobilního pohledu

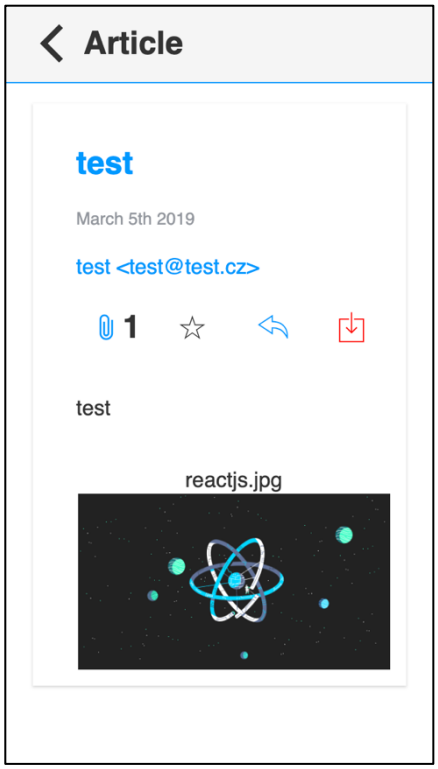
Při stisknutí detailu jakékoliv uživatelem oblíbené diskuzní skupiny, dojde k přesměrování na novou stránku, tato událost vyvolá vykreslení komponenty ***ArticleList***. Tato komponenta má jasný cíl vykreslit hlavičky určité množiny článků

dané skupiny ve formě seznamu, články jsou seřazeny dle data vytvoření od nejnovějších po nejstarší. Při prvním vykreslení je zobrazeno pouze prvních dvacet článků, toto množství je možné zvětšit při stisku na poslední položku seznamu kde se nachází speciální položka seznamu s tlačítkem *Load More*. Součástí této komponenty je v horní části také navigační lišta, která vykresluje vyhledávací komponentu ***SearchField***. Při vyplnění dojde k zobrazení článků, jejichž hlavička obsahuje alespoň část vyplněného textového řetězce. Dále navigační lišta obsahuje dvojici tlačítek. Po jejich stisknutí dojde k zobrazení všech nebo pouze uživatelem oblíbených článků aktuálně zvolené diskuzní skupiny.



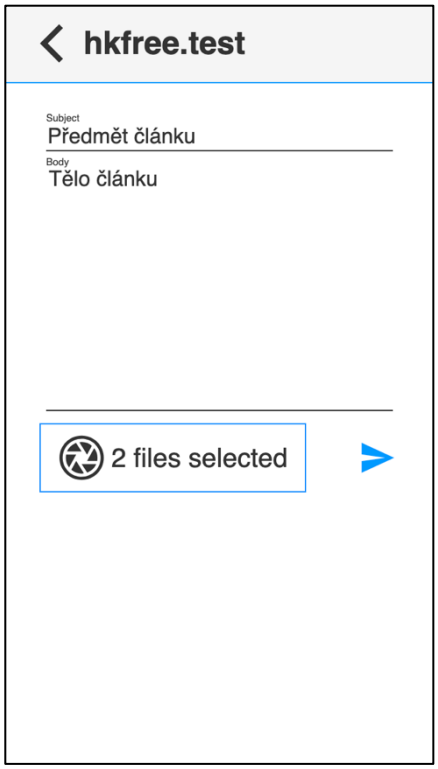
Obrázek 4. Mobilní pohled na hlavičky zvolené diskuzní skupiny

Po kliknutí na libovolnou hlavičku je uživatel přesměrován na stránku, která je vykreslena za pomoci komponenty ***Article.*** Ta pomocí listové komponenty ***TreeList***, zobrazuje hierarchii všech článku v přehledné stromové struktuře, to vše včetně zobrazení veškerých binárních příloh.



Obrázek 5. Mobilní pohled na detail článku

K přesměrování na poslední stránku dochází v případě, že si uživatel přeje na libovolně zvolený článek odpovědět. Po přechodu je vykreslena komponenta ***ArticleForm***, v té lze za pomocí komponent ***TextField***, nadefinovat předmět a tělo odpovědi. Za účelem přikládání libovolného počtu binárních příloh byla vytvořena komponenta ***FilePicker***. Jejím účelem je nejen kontrola, zdali nebyla překročena maximální velikost odeslaných souboru, ale také o zamezení odeslání nepodporovaných souborových formátů. Samotné odeslání článků proběhne po stisknutí ikony obálky.



Obrázek 6. Mobilní pohled formuláře nového článku

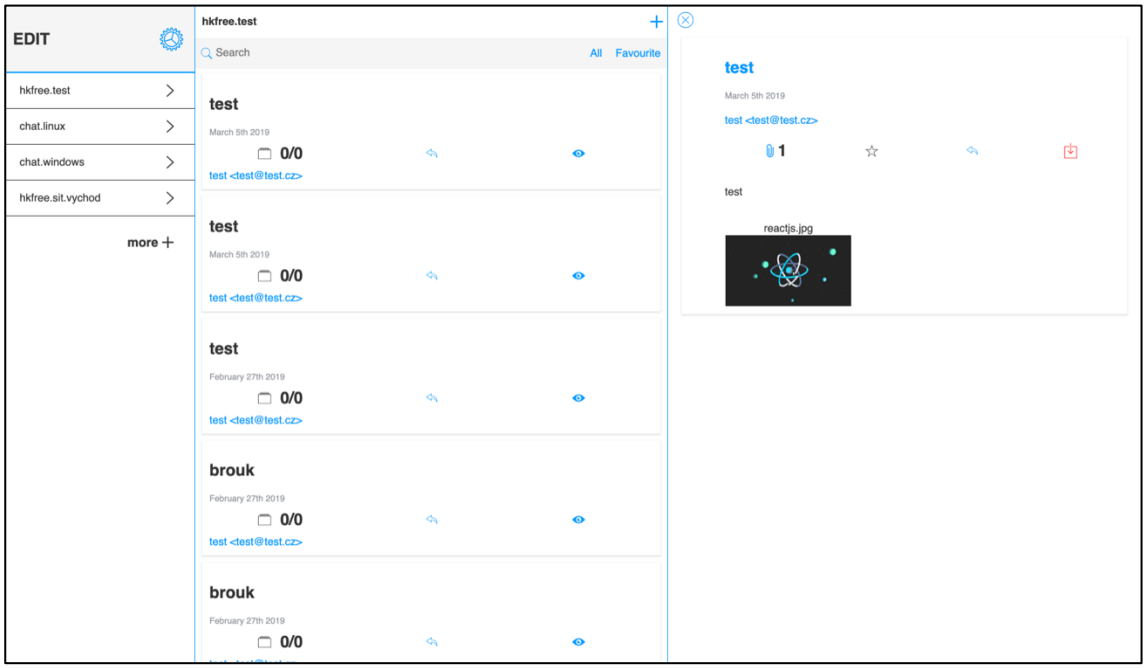
5.2.4 Desktop pohled

Na rozdíl od aplikace mobilní, desktopová verze skládá pouze z jedné jediné obrazovky, ta je pomocí komponenty ***Grid***, rozložena buď na dvě případně tři části v závislosti na tom, zdali je aktuálně vybrán libovolný článek. Druhým rozdílem oproti mobilní verzi je zobrazení komponenty ***ArticleForm***, ta je místo na samostatné stránce, zobrazena pomocí komponenty ***Dialog*** zobrazena jako vyskakovací okno.

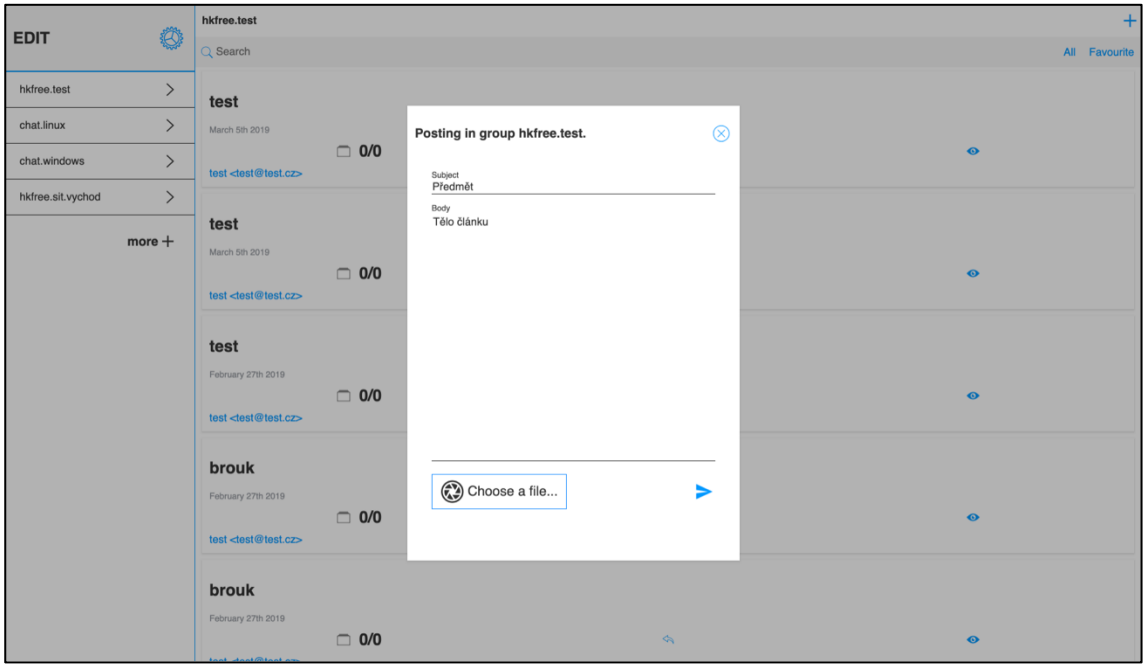
Levá část uživatelské rozhraní je vykreslena pomocí komponenty ***Navigation***, její chování je naprosto identické jako ve verzi mobilní.

Prostřední část zabírá komponenta ***ArticleList***. Ta má opět obdobnou funkcionalitu jako v případě mobilní verze, jediným rozdílem, je proměnlivá velikost. Jak již bylo zmíněno výše, velikost této komponenty je závislá na tom, zdali je vybrán konkrétní článek či nikoliv. Stejně jako v mobilní verzi lze v horní části komponenty využít buď textové pole pro vyhledávání článků či zaškrtnutí zobrazení pouze oblíbených článků.

Pravá strana je vykreslena pouze za předpokladu, že uživatel klikne na libovolný článek, tím dojde k vykreslení komponenty ***Article***, ta obdobně jako v mobilní verzi zobrazí detail článku, pomocí listové komponenty **TreeList**.



Obrázek 7. Desktop verze aplikace



Obrázek 8. Desktop pohled s otevřeným formulářem

5.2.5 PWA

Jedním z dalších požadavků byla možnost používat aplikaci i bez neustálého připojení k síti internet. Kvůli tomu bylo zapotřebí nějakým způsobem na klientském zařízení ukládat statické aplikační soubory, které nám serverová aplikace při webovém požadavku dodá. Pro potřeby této aplikace bylo nutné zajistit zejména ukládání do mezi paměti vstupního HTML souboru *index.html*. Dále hlavní aplikační kaskádový stylový soubor *styles.css*, jež obsahuje veškeré potřebné stylistické úpravy finálního produktu. V neposlední řadě také zaváděcí JavaScript soubor *app.js*, který má na starosti veškerou klientskou aplikační logiku. Mechanismus ukládání do mezi paměti, který byl po důkladné analýze zvolen jako nejvhodnější se nazývá Progresivní webová aplikace (v anglickém originále Progressive Web Application ve zkratce PWA).

Implementace PWA byla v našem případě velice jednoduchá, protože jak již bylo zmíněno v úvodu této kapitoly, klientská aplikace vychází z námi upravené verze *create-react-app*, která při vytvoření veškeré potřebné soubory generuje. Jedná se o soubor *serviceWorker.js*, jehož kód má na starosti nejen start a následné nastavení speciální funkcionality takzvaného *service-workera* ve webovém prohlížeči, ale také předání konfiguračních parametrů. S pomocí zmíněného souboru je možné webovému prohlížeči efektivně sdělit jaké aplikační soubory je nutné uložit do klientské mezi paměti pro správnou funkcionalitu offline režimu. Druhým souborem nutným pro korektní funkcionalitu PWA je přítomnost konfiguračního souboru *manifest.json*. V tomto souboru je zapotřební definovat veškeré povinné parametry nutné například k uložení aplikační přístupové ikony na ploše klientského zařízení. Dále je v tomto souboru definováno jméno aplikace, její zkratka, cesta k obrazovému souboru ikony aplikace, startovací adresa aplikace nebo v jaké obrazové orientaci máme aplikaci zobrazit.

Je nutné brát v potaz, že implementaci *service-workerů*, v době psaní práce, podporuje pouze část nejnovějších webových prohlížečů. V případě, že klient používá prohlížeč, který danou implementaci neposkytuje nebude mít možnost využít offline přístupu do aplikace. Aplikace bude nadále fungovat za předpokladu, že klient bude neustále připojen k síti internet.

5.2.5.1 Service workers

Jedná se o samostatně běžící JavaScript skript. Ten je na rozdíl od normálního JavaScript souboru dokonale oddělen on samotné webové aplikace. Hlavním účelem je otevřít dveře k těm funkcionalitám, které pro svou činnost nepotřebují samotnou webovou aplikaci, nebo uživatelskou interakci. V době psaní práce mezi nejžádanější funkcionality patří zobrazení notifikací nebo synchronizace na pozadí.

Při použití je třeba brát v potaz, že *service-worker* nemá přístup k aktuálnímu DOMu webové stránky. Z tohoto důvodu existuje mechanismus zasílání zpráv. Díky tomu mezi sebou může webová aplikace a *service-worker* komunikovat. Naopak nespornou výhodou je, že *service-worker* je schopen pracovat s rozhraním uživatelské databáze *IndexedDB*, jež tato práce využívá.

Z důvodu zabezpečení *service-workery* pro správnou funkcionalitu, vyžadují zabezpečené připojení pomocí protokolu **HTTPS** [20].

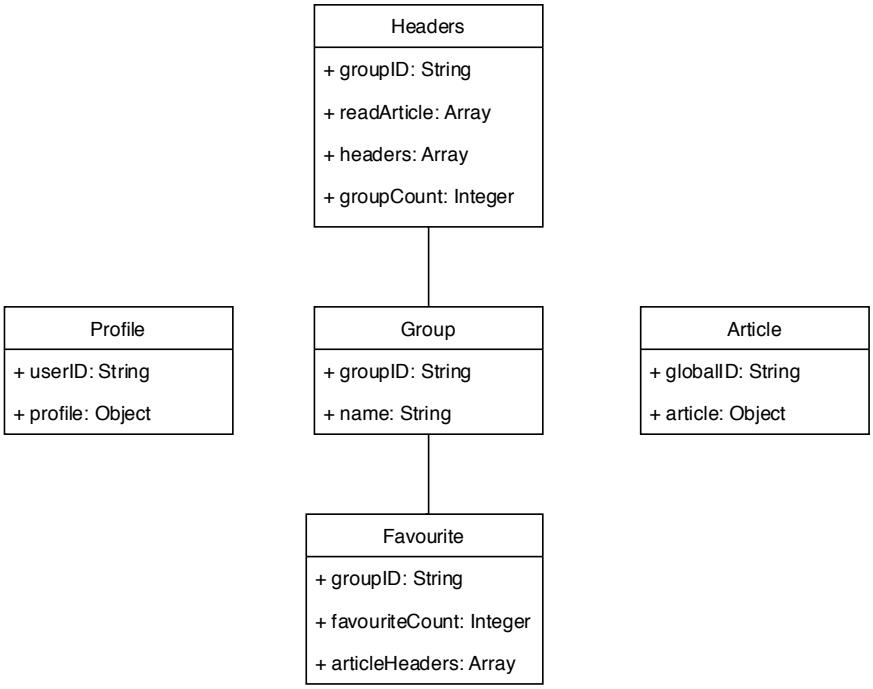
5.2.6 Klientský databázový model

Nyní když je naše aplikace schopna pracovat i v offline režimu díky ukládání statických souborů do mezi paměti, je třeba se zamyslet jakým způsobem uchovávat na klientské straně data. Momentálně by uživatel viděl pouze základní kostra aplikace, bez dalších dat, které by zajišťovaly následné fungování aplikace. Z tohoto důvodu bylo nutné použít technologii, která nám umožní ukládat aplikační data, včetně binárních souborů na straně klienta. Počáteční myšlenkou bylo použití základní *Web Storage*, funkcionality, kterou již nabízí všechny moderní webové prohlížeče, bohužel od tohoto řešení bylo na základě testování, shledáno jako nedostatečné. Zejména kvůli omezené velikosti, nemožností ukládání binárních dát a delšímu přístupovému času. Vyšší přístupový čas, protože *Web Storage,* stejně jako klientská aplikace, běží na hlavním JavaScriptovém vlákně prohlížeče, navzájem tedy soupeří o systémové zdroje. Jako ideální, a nakonec i finální řešení byla použita poměrně mladá *NoSQL* databáze schopná ukládat i objemná strukturovaná data v podobě JSON objektu, včetně objemných binárních souborů. To vše v poměrně krátkém přístupovém čase a bez krádeže systémových zdrojů hlavní klientské aplikace díky použití *service-workerů*. Veškeré práce nad databází tedy v ideálním případě probíhá na vlastních procesním vlákně.

Veškerá data uložená v klientské databázi jsou indexovaná pomocí unikátního klíče, díky tomu následně dochází k získání dat. Pro úspěšné ukládání dat je nejprve potřeba databázovému enginu říci, jaká je struktura dat, v našem případě se jedná o kolekce, nad kterými bude pracovat. Bylo tedy zapotřebí provést analýzu veškerých dat, se kterými bude aplikace pracovat a eliminovat ty, které se nehodí pro ukládání do klientského úložiště, případně by pouze zbytečně zabírala místo a zvyšovala přístupový čas k ostatním datům.

První kolekce dat se nazývá ***Headers***. Jako klíč použit název skupiny, za níž se schovává jeden obrovský JavaScript objekt, který obsahuje pole globálních klíčů všech přečtených článku v dané skupině. Dále obsahuje číselnou hodnotu všech článku dané skupin a v neposlední řadě pole hlaviček všech stažených článků. Nejsou tedy stahovány všechny články, ale pouze takové množství, která si uživatel libovolně navolí v nastavení aplikace. Počáteční hodnota stažených článků pro každou skupinu je 100. Dále je potřeba ukládat skupiny, které si uživatel přidal mezi sledované. K tomu je určena kolekce ***Group***. Zde je obdobně jako u kolekce předchozí jako klíč použit název dané skupiny. Hodnota je totožná s klíčem. K ukládání oblíbených článků uživatele slouží kolekce ***Favourite***. Klíčem je opět název diskuzní skupiny, do které článek spadá. Obsahem pro každou skupinu je objekt obsahující číselnou reprezentaci počtu článků a pole samotných hlaviček. Aby bylo uživateli umožněno číst například oblíbené články i offline existuje kolekce ***Article***. V níž je jako klíč použit globálně unikátní identifikátor článku. Datová struktura kolekce obsahuje objekt reprezentující stromovou strukturu všech článků, jež na sebe odkazují. Jedná se tedy o kompletní vlákno jednoho hlavního článku. Pro potřeby zobrazení uživatelských nastaveních byla vytvořena kolekce ***Profile***, jež jako klíč používá unikátní identifikátor uživatele. Identifikátor odkazuje na jednoduchý JavaScript objekt obsahující veškerá uživatelská data. Příkladem je podpis, zdali je povolen offline režimu nebo preferované téma uživatelského rozhraní.

Model datových struktur datových kolekcí graficky znázorněn v diagramu níže.



Obrázek 9. Diagram kolekcí klientské databáze

5.2.7 Ukládání a práce s offline daty

Aby byla umožněna efektivní práce nad lokálně uloženými daty a daty vzdáleného poskytovatele, bylo zapotřebí vytvořit mechanismus, jež bude umět pracovat v závislosti na různorodých faktorech. Výčet faktorů, se kterými je nutné počítat je vypsán v seznamu níže.

1. Připojení k internetu bude k dispozici
2. Připojení k internetu není k dispozici
3. Je povolen offline režim
4. Offline režim je zakázán

V počátcích vývoje produktu bylo zřejmé, že se nejspíše bude jednat o složitou implementaci. Opak se nakonec stal pravdou. To za pomoci dvou hlavních funkcionalit, jež nám nabízí samotná knihovna React od verze 16.8.

Aby bylo možné zjistit, zdali si uživatele přeje použít offline mód a povolit tak ukládání dat do lokální klientské databáze, bylo nutné použít rozhraní, jež umožní tyto informace uložit a být globálně přístupné. Toho je opět dosaženo použitím React *Context API*, a *React Hooks*. Konkrétně byla využita funkce *useContext*, jež umožňuje přímý přístup k námi požadované informaci, a to v jakékoliv funkcionální

komponentě naší aplikace. Je tedy možné k daným informacím ihned přistoupit kdekoliv, kde by mohl požadavek na získání vzdálených dat vzniknout.

Pro zjištění, zdali je k dispozici připojení k internetu implementoval autor vlastní *Hook*, pracovně nazvaný *useIsOffline*. Ten periodicky v reálném čase za pomocí http požadavku, v případě této práce serverová aplikace dané kontrolní rozhraní vystavuje, kontroluje, zdali je připojení k dispozici. Tato funkcionalita byla vyčleněna do samotného modulu a je v rámci *MIT* licence nabízena k volnému šíření přes portál s balíčky *npm*.

Finální implementace již obsahuje pouze jednoduchý strom podmínek, který dle daných proměnných hodnot kontroluje, jaká funkcionalita je požadována. Pokud je připojení k dispozici, tak je za každých okolnosti vyslán požadavek pro získání dat ze serveru. Je-li v té samé podmínkové větvi zjištěno, že je povolen offline režim, jsou nově získaná data uložena do lokální databáze k obnově jejího obsahu. V případě opačném je tato činnost vynechána. Pokud by nastala situace, že připojení k internetu není k dispozici nebo vzdálený server neodpovídá, jsou použita data z databáze lokální. Stalo by se, že bude lokální databáze prázdná, například z důvodu deaktivace offline režimu, je navrácena výchozí předem nadefinovaná odpověď. Uživatel by na tuto skutečnost měl být upozorněn pomocí notifikace.

5.3 Serverová aplikace

Pro serverovou aplikaci byly použitý nástroje definované v kapitole **4.3**, **4.4** a **4.6**. Pro zopakování byl použit Node.js framework *Koa* ideálně běžící na dedikovaném serverovém stroji, tato část výsledné aplikace slouží zejména jako most mezi komunikací klienta a Usenet serveru. V neposlední řadě také jako hlavní spojovací prvek uživatelem a databázovým úložištěm *SQLite*, které slouží pro ukládání uživatelských meta-dat, to z důvodu podpory synchronizace napříč uživatelovi zařízeními.

5.3.1 NNTP knihovna

Již dle počáteční analýzy bylo rozhodnuto, že serverová aplikace bude zastávat funkci prostředníka komunikace mezi Usenet serverem a klientským zařízením. Z tohoto důvodu vznikla nutnost použití takového softwarového

rozhraní, jež bude danou část obstarávat. Jako nejjednodušší řešení by bylo použití nativních součástí prostředí Node.js. Bohužel, na rozdíl, například od programovacího jazyka Python, který nativní rozhraní pro komunikaci pomocí síťového protokolu NNTP implementuje, Node.js, bohužel, jakožto relativně mladá technologie podobné rozhraní nenabízí. Z tohoto důvodu byla autorem práce věnována značná část úsilí zkoušení a testování rozdílných volně šiřitelných JavaScript NNTP knihoven.

Jako první byla použita knihovna *node-nntp*, ta bohužel nesplňovala ani základní požadavky na funkčnost, například chybějící implementace příkazu **OVER**. Značnou nevýhodou byla také absence alespoň základních jednotkových testů, z dlouhodobého hlediska se tedy nejednalo o vhodnou volbu. Následující je knihovna s názvem *nitpin*, jež ačkoliv silně převyšovala základní funkční požadavky, například dekódování binárních příloh zakódované pomocí yEnc. Naneštěstí knihovna využívá pro návrat asynchronních dat zastaralého a nepřehledného způsobu za použití takzvaných *callback* funkcí. Tato skutečnost neplnila autorovu představu o příjemné a přehledné práci s asynchronním kódem. Třetí a poslední byla, z této třetice nejmladší knihovna s názvem *newsie*. Předností této knihovny je nejen kompletní pokrytí jednotkovými testy, všechny napsané dle specifikacích definovaných v RFC dokumentech, ale také použití moderního *Promise* a *Async/Await* přístupu pro práci s asynchronním aplikačním kódem. Nespornou konkurenční výhodou byl také kód napsaný pomocí programovacího jazyka TypeScript. Ten z pohledu této práce zaručuje zejména předvídatelnost při práci nad nabízenými rozhraními. Ačkoliv se bezesporu jednalo o jednu z nejlepších implementací komunikace pomocí protokolu NNTP, autorovými požadavky bylo, vyjma základních funkčnosti, také jednoduché a snadno udržovatelné API. Knihovna *newsie* sice dané požadavky splňovala v plném rozsahu, ale nedostatkem byla nepřítomnost podpůrných funkcionalit, například požadované skládání článků do stromové struktury, nebo podpora článku ve velice populárním MIME standardu. Tyto podpůrné funkcionality bohužel nepodporovala žádná z výše uvedených knihoven, což bylo důvodem, proč bylo přistoupeno k implementaci vlastního řešení.

Základním stavebním kamenem vlastní knihovny bylo použití nativních prostředků, které nám Node.js nabízí, jedná se o balíček *stream*. s jehož pomocí dokážeme číst data z Usenet serveru pomocí proudů dat. Nedochází tak k přetěžování sítě přenášením jednoho objemného souboru dat, ale pouze malých částí, které následně složíme dohromady. Aby bylo možné pracovat s proudy dat, které Usenet zasílá, vznikly dvě podpůrné třídy, které s těmito proudy umí pracovat. První ***SingleLineStream*** je použita v případě, když je očekávaná odpověď Usenet serveru jednořádková. Třídou druhou je ***MultiLineStream***, jež pracuje s víceřádkovými daty. Obě třídy jsou potomkem nativní Node.js *Stream* třídy. Druhým balíčkem je *net*, díky němuž dostáváme přístup k nativní třídě *Socket*, s její pomocí je nám umožněno navázání síťové komunikace s Usenet serverem. Komunikace nadále probíhá zasíláním definovaných příkazů mezi sebou. Pro příjem dat již stačí pomocí klíčové metody *pipe*, jež třída *Socket* nabízí, zřetězit námi vytvořenou instanci dané třídy s instancemi našich tříd pracují s proudy.

5.3.1.1 Kódování binárních příloh

Protože počáteční testovací implementace knihovny umožňovala zasílání článku pouze jako jednoduchý textový řetězec, bylo zásadní doimplementovat rozhraní umožňující odesílání jak textové části, tak jedné nebo více binární přílohy. Ačkoliv byla nalezena řada RFC dokumentů, které se danou problematikou zabývají, v žádném nebylo srozumitelným způsobem implicitně napsáno, jak řešit kódování příloh při odesílání na Usenet server. Samozřejmostí je takové řešení, které podporuje největší možná většina, v době psaní práce dostupných Usenet server implementací. Dalším problémem bylo zjištění způsobu, jak binární přílohu k odesílanému článku přikládat. Konečným řešením bylo otestování více možných způsobů a následným vybráním toho jež přináší největším praktický přínos.

Prvním použitým kódováním bylo takzvané *Uuencode*, jež bylo na většině RFC dokumentech nejčastěji skloňováno. Počátky implementace spočívaly v reverzním inženýrství nativních NNTP implementací jiných programovacích jazyků, například v již zmíněném jazyce Python, to vše z jednoho prostého důvodu, zjištění, jakým způsobem ostatní jazyky zakódované binární soubory přikládají k uživatelem definovanému textovému obsahu článku. Díky zkoumání zdrojových

kódů nakonec autor práce zjistil, že pro připojení binárních příloh stačí za textový obsah článku použít kombinaci byte-kóde znaků CR a LF. Za tuto kombinaci pak následně stačí vložit již zakódované binární soubory. Bohužel i přes nově nabyté zkušenosti, se použití *Uuencode*, z důvodů celkové zastaralosti a slabé podpory mezi novějšími Usenety, ukázalo pouze jako slepá ulička.

Druhým a opět neúspěšným pokusem bylo použití novějšího způsobu kódování označovaného jako *yEnc*. Naneštěstí vzhledem k absenci kvalitních kódovacích a dekódovacích knihoven by bylo zapotřebí přijít s vlastní implementací, pro nedostatek času a zkušeností bylo nakonec opuštěno i od *yEnc* kódování. Jako poslední pokus, který se nakonec ukázal být nejvhodnějším, bylo odesílání článků včetně binárních příloh pomocí v dnešní době nejvíce používaného internetového standardu MIME.

MIME je zkratka pro Multi-Purpose Mail Extensions. Jak již název napovídá, jedná se o rozšíření internetové emailové protokolu, jež dovoluje uživatelům výměnu nespočtu druhů datových souborů v rámci sítě internet. Příkladem mohou být fotografie, audio záznamy nebo video záznamy. Ačkoliv byl MIME vytvořen zejména pro použití v rámci emailového protokolu SMTP, typy obsahu definované práce pomocí MIME standardu jsou důležité i v ostatních komunikačních protokolech, například HTTP nebo námi používaný protokol NNTP [21].

Jednoduché implementaci odesílání článku v MIME standardu bylo docíleno použití moderní volně šiřitelné knihovny *mailparser*, konkrétně za pomoci jedné z jejích metod *simpleParser*. Implementačně bylo zapotřebí akorát doplnit definici nadstandardních hlaviček specifických pro protokol NNTP. Výsledkem byla fungující implementace zajišťující jak odesílání čistého textu, tak binárních příloh. To vše v moderním MIME standardu. Podporu odesílání článků v MIME standardu v době psaní práce žádná z volně šiřitelných Node.js NNTP knihoven nenabízí.

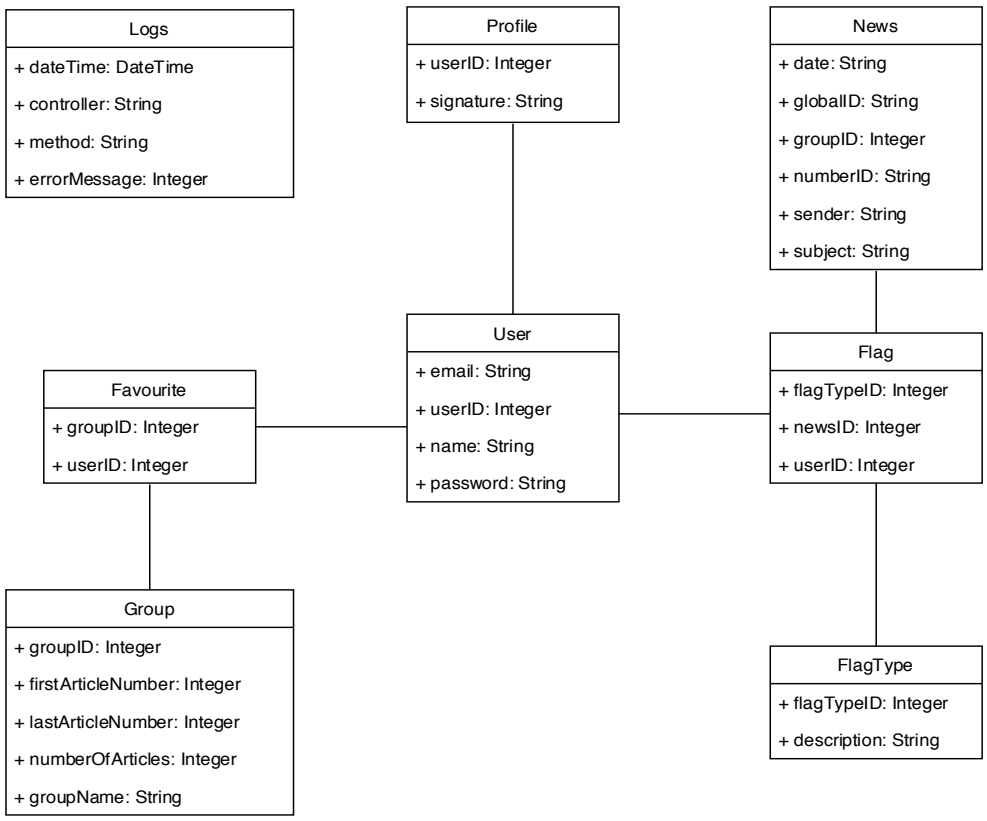
5.3.2 Použité modely

Pro uložení veškerých dat na straně serverové aplikace slouží souborové relační databázové *SQLite* uložiště. Veškeré tabulky v databázi jsou v serverové aplikaci reprezentovány jako navzájem závislá dvojice schéma a model. Schéma je jednoznačná definice entity, definována pomocí databázového jazyka

SQL, definující atributy tabulky, jejich typy a vztahy mezi jednotlivými entitami. Model následně definuje množinu služeb, jež může samotný program nad samotnou entitou využívat.

Základním modelovým prvkem je model s názvem ***User***, ten definuje samotného uživatele klientské aplikace. Na něm přímo závisí model ***Profile***, jež obsahuje veškeré uživatelsky potřebné dodatečné informace. Model ***Favourite***, definuje množinu oblíbených diskuzních skupin uživatele. Data pro její správné použití zajištuje model ***Group*** obsahující veškeré potřebné informace o všech dostupných diskuzních skupinách daného Usenetu. Model ***Flag***, dává naopak do spojitosti další model ***FlagType***. Ten obsahuje informace o příznacích, které může uživatel libovolným článkům přiřadit. Hlavičky daných článků jsou definovány a průběžně ukládány za pomoci modelu ***News***.

Veškeré atributy, datové typy a závislosti výše zmíněných modelů, jsou znázorněny na níže uvedeném UML diagramu.



Obrázek 10. Diagram použitých databázových schémat

5.3.2.1 Logování

Pro účely logování byl vytvořen samostatný model ***Logs***, zobrazen k diagramu výše. Tento model není na rozdíl od zbytku závislý na ostatních a funguje jako samostatná entita, jejíž účelem je jen a pouze sběr veškerých běhových chyb. Model poskytuje pouze základní rozhraní pro čtení a zápis. Chyby ukládané v tomto modelu mohou nastat, ať už z důvodu programátorem neošetřené programové výjimky nebo vinou mimo aplikačních vlivů, například nedostupný Usenet server případně chyba některé z použitých knihoven. Do logu by měli být zapisovány pouze relevantní informace o daném problému, jež mohou, být dále analyzovány a pomoci při hledání řešení vzniklých problémů, nebo vytyčení slabých míst v aplikaci. Informace ukládané pro potřeby této práce jsou, *dateTime*, označována jako časová stopa serveru v době výskytu chyby, *controller*, místo vzniku chyby, *method*, operace, která způsobila vyvolání programové výjimky a samotný textový popis vzniklé chyby, vytvořený prostředím Node.js, *errorMessage*. Tento textový popis není námi žádným způsobem upravován.

5.3.3 Autorizace uživatele

Aby bylo docíleno správně fungující synchronizace uživatelských preferencí napříč zařízeními, byla na straně serverové aplikace implementována autorizační funkcionalita. K Autorizaci uživatele je využíván protokol oauth2. Samotná autorizace je již prací volně dostupné knihovny s názvem *passport*, k samotné autorizaci uživatele a získání jeho identifikačních údajů, například uživatelské jméno či emailová adresa, je využito nezávislých poskytovatelů protokolu oauth2. Pro účely této práce bylo zvoleno řešení poskytované známou a důvěryhodnou společností Google. Vlastní aplikační kód, následně slouží pouze k uložení údajů o uživateli do lokální serverové databáze.

Pro následné prokazování přihlášeného uživatele je využit systém známý jako *Session*. Tento způsob spoléhá na to, že každý požadavek z klientské aplikace bude obsahovat hlavičku s identifikátor platné *Session*, díky tomuto identifikátoru je následně server schopen zjistit od jakého uživatele požadavek přišel a poskytnou mu pouze přístup k takovým zdrojům, ke kterým má povolen přístup. Momentálně je v aplikaci pouze jedna uživatelská role, všichni uživatelé proto mají přístup

ke stejným zdrojům. O všechny operace jako je kontrola platného *Session* identifikátoru nebo získání a uložení údajů přihlášeného uživatele se stará výše zmíněna knihovna *passport*. V době psaní práce je zmíněná knihovna jakýmsi standardem autorizace uživatele v Node.js aplikacích. Pro použití je zapotřebí pouze vytvoření konfiguračního souboru a definice potřebných parametrů. V našem případě byla potřebná zejména definice poskytovatele autorizace. Za tímto účelem byla použita pomocná knihovna *passport-google-oauth20*. Jedná se o neoficiální knihovnou nevytvořenou společností Google, ta jako potřebné konfigurační parametry vyžaduje následující základní informace. Prvním parametrem je identifikátor a tajný identifikátor aplikace, který bude službu využívat. Všechny potřebné údaje jsou přístupně po vytvoření vývojářského Google účtu a registraci aplikace do služby oauth2.

5.4 Zabalení a nasazení na produkčním prostředí

Nasazení obou aplikací na produkční prostředí lze dvěma způsoby. Preferovaným způsobem je použití veřejně dostupného Docker obrazu aplikace a nasazení provést bez nutnosti přidaného softwaru a větších znalostí. V případě potřeby větší konfigurace, lze nasazení provést pomocí samotného běhového prostředí Node.js. V obou případech je nutné při nasazení dbát na přítomnost konfiguračního souboru *.env* v kořenovém adresáři, ve kterém se aplikace nachází. Veškeré příkazy jsou následně spouštěny pomocí příkazové řádky.

Pro správnou funkčnost aplikace bez připojení k internetu je třeba použít zabezpečený protokol **HTTPS**.

5.4.1 Konfigurační soubor .env

Je konfigurační soubor slouží k definici konfiguračních proměnných, které by se mohly v různých prostředích nebo implementacích lišit. Protože účelem této práce bylo vytvořit aplikaci, kterou bude možné nasadit na jakémkoliv prostředí ať už v prostředí firemním nebo soukromém. Bylo dbáno na to, aby konfigurace byla v obou případech jednoduchá a zvládl ji i začátečník v oboru. Soubor sám o sobě tedy pro správný a bezproblémový chod serverové aplikace musí bez výjimek obsahovat

definici všech konfiguračně povinných parametrů. Seznam všech proměnných a jejich význam je popsán v tabulce níže.

Název	Popis	Výchozí hodnota
PORT	Systémový port, na kterém bude serverová aplikace naslouchat na příchozí klientské požadavky.	3333
AUTH_TYPE	Textová definice oauth2 autentifikačního poskytovatele, v základní verzi aplikace je k dispozici pouze google poskytovatel	‘google‘
CLIENT_ID	Textový řetězec, sloužící k identifikaci serverové aplikace používající oauth2 služby	null
CLIENT_SECRET	Hash generovaný textový řetězec, reprezentující daného klienta, v rámci této aplikace by se měl nacházet pouze v *.env* souboru, aby nemohl být zneužit	null
COOKIE_KEY	Klíč sloužící k identifikace cookies, při vytváření a příjmu klientských požadavcích	‘cookie‘
HASH_KEY	Klíč sloužící ke generování hash textového řetězce při vytváření hesla uživatele.	null
NNTP_IP	IP adresa Usenet poskytovatele	null
NNTP_PORT	Port, na kterém daný Usenet poskytovatel naslouchá	119

*Tabulka 2. Definice proměnných souboru **.env***

5.4.2 Kompilace pomocí Node.JS

Prvním způsobem je spuštění aplikaci přímo v samotném prostředí Node.js. Základním předpokladem je přítomnost běhového prostředí Node.js verze 11.9.0 nebo novější a balíčkový manažer *npm* verze 6.8.0 nebo vyšší. Systémové požadavky aplikace jsou dány požadavky daného prostředí. Veškeré příkazy potřebné ke spuštění aplikace v produkčním módu jsou definovány v základním konfiguračním souboru *package.json*, seznam příkazů a jejich účel jsou popsány v kapitole **5.4.4**.

Pro úspěšné spuštění aplikace je nutné použít množinu následujících příkazů v dané sekvenci.

```
1. npm run install:server
2. npm run install:client
3. npm run build
4. npm run create:database
5. npm run start
```

Příkaz 10. Kompilace a spuštění aplikace pomocí prostředí Node.js

5.4.3 Spuštění pomocí Docker obrazu

Druhým a preferovaným způsobem je použití Docker obrazu, volně přístupného pomocí služby na ukládání obrazů zvané *DockeHub*. Pro tento způsob je potřeba mít nainstalované pouze prostředí Docker ve verzi 2.0.0.3 nebo novější. Systémové požadavky jsou dány prostředím Docker. Dodavatel služby na svých webových stránkách uvádí systémové požadavky pro většinu operačních systémů. Příkladem mohou být požadavky operačního systému CentOS [22]. Stažení a spuštění samotného obrazu je postupně popsáno pomocí příkazů níže.

```
1. docker pull nomiadam/lunews
2. docker run --name lunews --env-file=<Cesta k souboru .env> -p 443:<Port
   definovaný v souboru .env> -d nomiadam/lunews
```

Příkaz 11. Kompilace a spuštění aplikace pomocí nástroje Docker

Následně pomocí konzolového příkazu `docker ps` můžeme ověřit úspěšnost spuštění aplikace.

5.4.4 Povolené příkazy

Součástí všech aplikací běžících na platformě Node.js je skupina, pomocných a konfiguračních skriptů. Jejich účelem je usnadnit používání často se opakujících činností, ať už se jedná o zapnutí jednotkových testů, či produkční kompilace výsledné aplikace.

Je důležité zmínit, že všechny skripty jsou určené k použití v příkazové řádce. Každý příkaz musí splňovat formát uvedený níže.

```
npm run <název příkazu>
```

Příkaz 12. Požadovaný formát npm příkazů

Seznam všech dostupných příkazů naší aplikace a jejich stručný popis je znázorněn v tabulce níže.

Start	Start celé aplikace
Server	Spuštění vývojářské verze serverové části aplikace, s podporou restartování aplikace v závislosti na změnách
client	Spuštění klientské skriptu, jež zapíná vývojářský mód klientské aplikace s podporou *hot-reloading* funkcionality
install:server	Stažení potřebných závislostí pro serverou část aplikace
install:client	Stažení potřebných závislostí pro klientskou část aplikace
create:database	Spuštění konfigurační skriptu pro inicializaci základních datových struktur databázové souboru
build:server	Kompilace zdrojového kódu serverové části aplikace z vývojářské TypeScript syntaxe do produkční JavaScript notace
build:client	Kompilace zdrojové kódu klientské aplikace do produkční verze
dev	Paralelní spuštění příkazů *server* a *client*
build	Paralelní spuštění příkazů *build:client* a *build:server*

test	Spuštění jednorázové kontroly veškerých jednotkových testů v rámci celé aplikace
test:watch	Spuštění kontroly všech jednotkových testů s funkcionalitou znovuspuštění testů v případě detekce změn ve zdrojovém kódu

Tabulka 3. Dostupné NPM příkazy

6 Výsledky a testování

Obsahem testování byla kontrola, zda byl splněn základní požadavek a klientská aplikace je použitelná na většině v dnešních době používaných mobilních zařízeních. Mimo jiné, protože podstatnou částí této práce je komunikace klientské aplikace se serverovým rozhraním, byla otestována rychlost rozhraní, která budou nejvíce zatěžována.

6.1 Podpora mobilních zařízeních

Při testování podpory mobilních zařízeních bylo hleděno zejména zdali je k dispozici použitelné uživatelské rozhraní, bez omezení funkcionalit aplikace. Pro účel testu, bylo použito jak testovací prostředí webového prohlížeče Google Chrome, tak dvě reálná mobilní zařízení. Při testování v prostředí Google Chrome byl test proveden pro různé velikosti mobilních obrazovek.

Seznam virtuálních zařízeních, pro které byl test proveden je uveden v tabulce níže.

Rozlišení obrazovky	Virtuální zařízení
320x568	iPhone 5/SE
360x640	Samsung Galaxy S5
375x667	iPhone 6/7/8
375x812	iPhone X/Xs
411x731	Google Pixel 2

Tabulka 4. Testovaná mobilní zařízení

U žádného z výše uvedeného zařízení nebyla zjištěno žádná odchylka vůči požadovanému standardu.

Pro test s reálnými zařízeními byly použity mobilní telefony značky Apple, konkrétně modely iPhone SE a iPhone X. V obou případech byl výsledek testu ekvivalentní s totožným zařízením virtuálním.

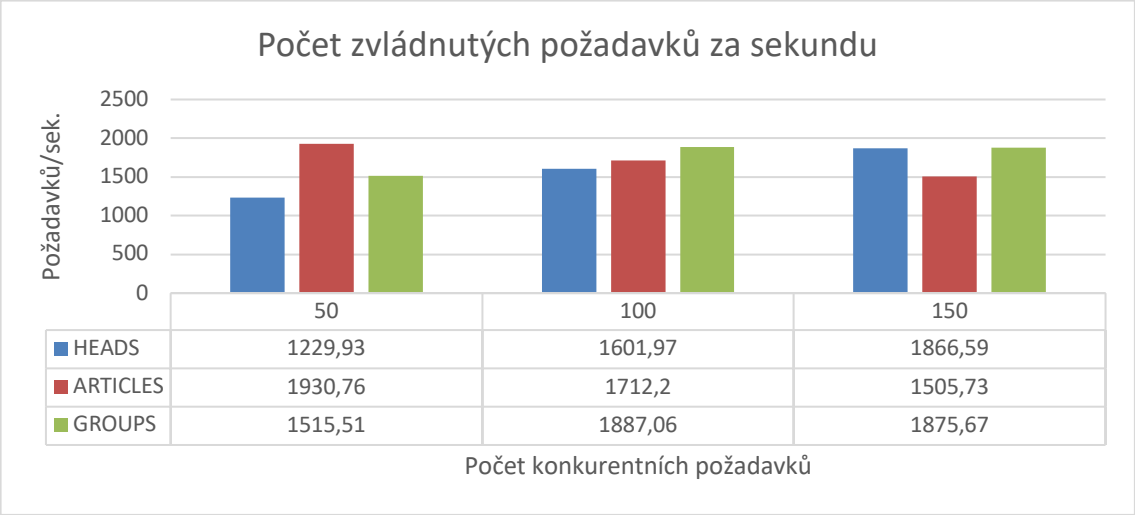
Díky tomuto testu bylo ověřeno, že aplikaci je možno bez jakýchkoliv omezení používat jak na desktop zařízeních, tak na zařízeních mobilních.

6.2 Rychlost zpracování požadavku

Jak bylo zmíněno v úvodu této kapitoly, stěžejní část praktické části této práce je komunikace klientské a serverové aplikace. Je třeba brát v potaz, že svou roli hraje nejen rychlost serverové aplikace a doba přístupu k lokální databázi, ale především odezva samotného Usenet poskytovatele. K otestování zvolených rozhraní byl použit nástroj s názvem *Apache Benchmark* [22].

Pro testování byla zvolena tři rozhraní, u kterých se v budoucnu očekává největší zatížení. První rozhraní **HEADS**, slouží k získání prvních sta hlaviček vybrané diskuzní skupiny. Rozhraní **ARTICLES** slouží k získání stromové struktury náhodného článku obdobně zvolené diskuzní skupiny. Poslední rozhraní **GROUPS** slouží k získání všech dostupných diskuzních skupin daného Usenet poskytovatele.

V následujícím grafu je zobrazeno, množství požadavků za sekundu, které jsou testovaná rozhraní schopna obsloužit.

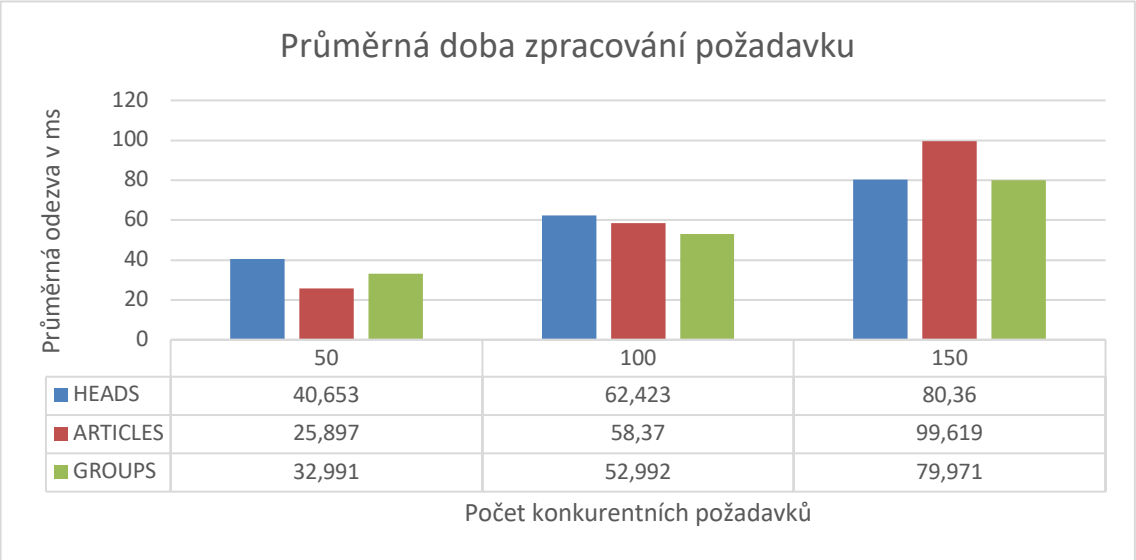


Graf 1. Počet zvládnutých požadavků za sekundu

Z grafu je zřetelné, že nejvíce rovnoměrné množství zvládnutých požadavků je při stech paralelních požadavcích, naopak při stoupajícím množství se začínají objevovat značné rozdíly.

Dalším klíčovým kritériem je průměrná doba zpracování požadavku. V našem případě vše, co trvá nad 100ms, je považováno za nadměrné čekání.

V grafu níže je vidět, že i při největší zátěži, tedy sto padesáti paralelních požadavcích za sekundu, žádná z hodnot mezní hranici nepřekračuje. Je obdobně zřetelné, že při větším počtu požadavků, nebude možné rychlou odezvu zaručit.



Graf 2. Průměrná doba zpracování požadavku

7 Závěr

Výsledkem práce je plně funkční webová aplikace s názvem LuNews. Produkt umožnuje plný uživatelský zážitek při práci s Usenety. LuNews oproti konkurenčním produktům vyniká zejména použitím moderních technologických postupů a doplňků. V době psaní práce se jedná o jedinou moderní Usenet čtečku použitelnou v prostředí webového prohlížeče. Příkladem funkcionalit, které žádná z konkurenčních webových aplikací nenabízí, může být například synchronizace napříč zařízeními, offline režim nebo podpora více platforem. Aplikace jako taková se v mnoha ohledech vyrovná nativním aplikacím, nebo je dokonce překoná. Navíc se dle dostupných informací jedná o první Usenet čtečku, napsanou kompletně pouze pomocí programovací jazyka JavaScript.

Hlavní výzvou praktické části této práce byla zejména práce se samotným protokolem NNTP. Problémem se ukázala být zejména chybějící novodobá dokumentace, nebo malá vývojářská komunita. V případě problému se tak nebylo na koho obrátit. Naopak zcela pozitivním bylo bližší seznámení s RFC dokumenty. S těmi již autor dokáže pohodlně pracovat a efektivně pomocí nich analyzovat problémové domény.

Při tvorbě práce samotné si autor vyzkoušel nejen softwarovou analýzu, ale také například identifikaci požadavků budoucích klientů, nebo celkový návrh kompletní softwarové aplikace. V neposlední řadě také implementaci projektu střední velikosti. Na druhou stranu aplikace trpí řadou nepříjemných neduhů. Hlavním problémem se z autorova pohledu jeví ne příliš vzhledné uživatelské rozhraní. Výzvou bylo také implementace offline režimu, který ačkoliv z pohledu uživatele funkční, nesplňuje všechny autorovi požadavky.

Kromě projektu samotného, v rámci této práce vznikly také tři nezávislé volně šiřitelné knihovny. Dané knihovny jsou dostupné na portálu s balíčky *npm*. Samotný výsledná aplikace je volně dostupná, v licenci MIT, na adrese, verzovacího portálu *github*, *https://github.com/NomiAdam/lunews*. Projekt je tedy možné ho libovolně pře použít či dále upravovat.

8 Seznam použité literatury

[1] *Usenet* [online]. 2019 [vid. 2019-04-17]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Usenet&oldid=889940787

[2] LAPSLEY, P. a B. KANTOR. *RFC 977* [online]. 1986 [vid. 2019-04-17]. Dostupné z: https://tools.ietf.org/html/rfc977

[3] *Network News Transfer Protocol* [online]. 2019 [vid. 2019-04-17]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Network_News_Transfer_Protocol&oldid=16954498

[4] BARBER <SOB@ACADEM.COM>, Stan. *RFC 2980* [online]. 2000 [vid. 2019-04-17]. Dostupné z: https://tools.ietf.org/html/rfc2980

[5] FEATHER, C. *RFC 3977* [online]. 2006 [vid. 2019-04-17]. Dostupné z: https://tools.ietf.org/html/rfc3977

[6] MURCHISON, Kenneth a Jeffrey M. VINOCUR. *RFC 4643* [online]. 2006 [vid. 2019-04-17]. Dostupné z: https://tools.ietf.org/html/rfc4643

[7] MURCHISON, Kenneth a Jeffrey M. VINOCUR. *RFC 4644* [online]. 2006 [vid. 2019-04-17]. Dostupné z: https://tools.ietf.org/html/rfc4644

[8] ELIE <JULIEN@TRIGOFACILE.COM>, Julien. *RFC 6048* [online]. 2010 [vid. 2019-04-17]. Dostupné z: https://tools.ietf.org/html/rfc6048

[9] MERHAUT, Filip. *Implementace klienta NNTP* [online]. B.m., 2006 [vid. 2019-04-17]. Bakalářská práce. Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky. Dostupné z: https://theses.cz/id/9a41qr/

[10] HORTON, M. R. a R. ADAMS. RFC 1036 [online]. 1987 [vid. 2019-04-17]. ISSN 2070-1721. Dostupné z: https://www.rfc-editor.org/info/rfc1036

[11] *Mozilla Thunderbird* [online]. 2019 [vid. 2019-04-17]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Mozilla_Thunderbird&oldid=885696603

[12] JavaScript. *MDN Web Docs* [online]. 2005 [vid. 2019-04-17]. Dostupné z: https://developer.mozilla.org/bm/docs/Web/JavaScript

[13] ASTON, Ben. A brief history of JavaScript. *Ben Aston* [online]. 1. duben 2015 [vid. 2019-04-17]. Dostupné z: https://medium.com/@benastontweet/lesson-1a-the-history-of-javascript-8c1ce3bffb17

[14] *React* [online]. 2019 [vid. 2019-04-17]. Dostupné z: https://reactjs.org/index.html

[15] FOUNDATION, Node js. Nodejs. *Node.js* [online]. 2015 [vid. 2019-04-17]. Dostupné z: https://nodejs.org/en/about/

[16] FOUNDATION, Node js. The Node.js Event Loop, Timers, and process.nextTick(). *Node.js* [online]. 2015 [vid. 2019-04-17]. Dostupné z: https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/

[17] FOUNDATION, Node js. Overview of Blocking vs Non-Blocking. *Node.js* [online]. 2015 [vid. 2019-04-17]. Dostupné z: https://nodejs.org/en/docs/guides/blocking-vs-non-blocking/

[18] *Docker (software)* [online]. 2019 [vid. 2019-04-17]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Docker_(software)&oldid=890503436

[19] *About SQLite* [online]. 2000 [vid. 2019-04-17]. Dostupné z: https://www.sqlite.org/about.html

[20] GAUNT, Matt. Service Workers: an Introduction | Web Fundamentals. *Google Developers* [online]. 2019 [vid. 2019-04-17]. Dostupné z: https://developers.google.com/web/fundamentals/primers/service-workers/

[21] *MIME* [online]. 2019 [vid. 2019-04-17]. Dostupné z: https://en.wikipedia.org/w/index.php?title=MIME&oldid=891254387

[22] *ab - Apache HTTP server benchmarking tool - Apache HTTP Server Version 2.4* [online]. [vid. 2019-04-25]. Dostupné z: https://httpd.apache.org/docs/2.4/programs/ab.html

9 Přílohy

Adresářová struktura na přiloženém CD

- src
    - Zdrojové soubory serverové aplikace
- client
    - config
        - Konfigurační soubory klienstké aplikace
    - src
        - Zdrojové soubory klientské aplikace
    - public
        - favicon.ico
        - index.html
        - logo.png
        - manifest.json
    - scripts
        - Spouštěcí skripty klientské aplikace
    - jestConfig.js
    - package.json
- tsconfig.json
- tslint.json
- .dockerignore
- .gitignore
- DockerFile
- LICENSE
- package.json
- README.md

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2018/2019

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Kvasnička Adam	Urbanice 51, Urbanice	I1600566

TÉMA ČESKY:

Moderní klient pro Usenet news

TÉMA ANGLICKY:

Modern Usenet News Client

VEDOUCÍ PRÁCE:

Ing. Pavel Kříž, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl: Navrhnout a implementovat moderní klient pro čtení o odesílání příspěvků v systému Usetnet News pomocí protokolu NNTP. Klient bude implementován v Reactu jako progresivní webová aplikací využívající lokální databázi ve webovém prohlížeči.

Osnova:
1. Úvod
2. Protokol NNTP
3. Rešerše existujících klientů
4. Nástroje
5. Návrh a implementace řešení
6. Testování
7. Závěr

SEZNAM DOPORUČENÉ LITERATURY:

https://tools.ietf.org/html/rfc977
https://tools.ietf.org/html/rfc3977
https://reactjs.org/

Podpis studenta: ..... **Datum:** 10. 10. 18

Podpis vedoucího práce: ..... **Datum:** 10. 10. 18