FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Implementace webového portálu pro sběratelskou karetní hru
Student:	Bc. Pavel Špecht
Vedoucí:	Ing. Jiří Novák, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je implementovat webový portál pro hráče sběratelské karetní hry The Lord of the Rings TCG. Vytvořte systém, kam bude hráč schopen uložit svou kolekci karet. Karty budou odpovídat reálným objektům, proto je zaznamenán i jazyk a stav karty. Herní balíček bude vytvořen za pomoci filtrace karet. Filtrovat půjde na základě textového řetězce a parametrů karet. Systém na základě historie vkládání navrhne do balíčku další karty a bude kontrolovat validitu balíčku. Karty z kolekce lze vyměňovat s ostatními hráči. Uživatelé mezi sebou budou moci komunikovat v oddělených konverzacích i v obecném vláknu.

- Popište použité technologie a klíčové prvky
- Shrňte pravidla a mechaniky dané karetní hry
- Získejte a zpracujte data z existujících wiki stránek
- Navrhněte strukturu databáze, hierarchii systému a rozložení webového portálu
- Porovnejte načítání dat pomocí ElasticSearch indexu a pomocí dotazu z databáze, relevanci filtrování a efektivitu přenosu dat
- Otestujte pomocí Unit testů

Elektronicky schválil/a Ing. Jaroslav Kuchař, Ph.D. dne 10. prosince 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Implementace webového portálu pro sběratelskou karetní hru

Bc. Pavel Špecht

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Novák, Ph.D.

4. května 2022

Poděkování

Rád bych poděkoval vedoucímu práce, Ing. Jiřímu Novákovi, Ph.D., za rychlou odezvu, ochotu, podporu, shovívavost a užitečné rady při vypracování práce. Rodina, přátelé a hlavně přítelkyně (slovo hlavně jsem byl i tentokrát přinucen napsat) mi byli velkou oporou.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 4. května 2022 . . . . . .

Odkaz na tuto práci

Špecht, Pavel. *Implementace webového portálu pro sběratelskou karetní hru.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Hlavní téma práce spočívá ve vytvoření webové aplikace pro nadšence do sběratelské karetní hry LOTR TCG. Po prvotním nastínění problému se představí teoretické koncepty a technologie, které výrazně urychlí a zlepší práci se systémem. Po získání informací o kartách pomocí dolování dat z webového zdroje byl vytvořen databázový model. Po naplnění databáze zpracovanými daty byl nakonfigurován Elasticsearch index. Aby spolu uživatelé systému mohli komunikovat v reálném čase, využila se technologie WebSocket. Pro snížení velikosti přenesených dat a zrychlení načítání stránek bylo výchozí nastavení politiky pro práci s dočasnou pamětí ve frameworku Symfony modifikováno. V části vytváření herních balíčků má uživatel k dispozici doporučovací nástroj založený na filtrování se zaměřením na obsah z metadat a informací o kartách. Kromě dočasné paměti na úrovni HTTP dotazů a odpovědí se řešilo i ukládání výsledků náročných výpočetních operací na serveru. V závěrečné části práce byl ověřen přínos použití Elasticsearch technologie oproti databázovému řešení. Jaké skutečné výsledky přinesla aplikace vlastní politiky pro dočasnou paměť? Vrací doporučovací nástroj relevantní výsledky podle aktuálního stavu herního balíčku? Naměřené hodnoty indikují splnění všech vytyčených cílů. Funkčnost systému podporuje budoucí použití v uzavřené komunitě hráčů a sběratelů.

Klíčová slova webová aplikace, Elasticsearch, WebSocket, dolování dat z webového zdroje, politika dočasné paměti

Abstract

The main topic of this thesis is considered as a creation of a web application for enthusiasts of the LOTR trading card game. Before the real application, theoretical concepts and principles are presented. They will significantly improve the system performance and reduce the loading time. After card information is gathered, rows consisting of parsed data are inserted into the database. The Elasticsearch index was configured afterwards. To enable real-time communication for system users, the WebSocket technology was used. As it is not based on the HTTP request-response model, the server does not need to wait for the client to send a message first. Default Symfony framework caching policy settings was made remarkably. However, when there is a need for change, it can be quite difficult to adjust. To reduce a transfer data amount, a custom configuration was set. An application of a recommender system can be found in a deck building part of the system. It is based on content filtering using card parameters and metadata. In the end, the Elasticsearch solution was compared with the plain relational database solution. Did changes matter in cache policy? Are results of the recommender system relevant to the current deck situation? Measured values indicate successful fulfillment of set objectives. The functional system implies a future usage in a closed group of players and collectors of this game.

Keywords web application, Elasticsearch, WebSocket, web data mining, caching policy

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Představení projektu

Majitelé sběratelských karet se dělí na dva hlavní tábory. Prvním jsou hráči. Na pravidelně se opakujících turnajích využívají karty ke hře. Obecně určená pravidla si často ještě sami modifikují kvůli nevybalancovaným mechanikám. Druhým táborem jsou sběratelé. Na rozdíl od hráčů berou velmi vážně stav karet a jejich vizuální nedostatky. Záměr sbírání může být čistě osobní, kdy se sběrateli líbí předloha či zpracování karet, a od začátku není účelem sbírku jako celek následně prodat se ziskem. Jiní berou tyto předměty pouze jako formu investice, kdy se karty v podstatě neliší od uměleckých předmětů či starých aut. Někteří sbírají pouze určité národy/druhy (podobné barvám u mariášových karet), jiní pouze určité série či foily (zpravidla cenná a vzácná povrchová úprava karty).

Obě skupiny potřebují mít dobrý přehled nad vlastní kolekcí. Kolekce lze ukládat do levných papírových pořadačů, avšak existují mnohem nákladnější varianty v podobě speciálních šanonů, ve kterých jsou karty uloženy v několika ochranných vrstvách. Někteří sběratelé si také jednotlivé počty karet ukládají pomocí tabulkového softwaru z důvodu přenositelnosti a rychlosti kontroly.

Motivace a současný stav

Existuje mnoho nástrojů pro správu sběratelských karet. Jsou však většinou implementovány obecně, což často znemožňuje efektivní filtrování a plně neuspokojuje potřeby uživatele. Pro sběratelskou karetní hru, pro kterou je vytvořen tento projekt, není k dispozici mnoho nástrojů. Hlavním důvodem je fakt, že karty byly vydávány mezi lety 2001 a 2007. Aktivní distribuce ze strany výrobce a držitele licence zanikla již dávno. Fanouškovská základna není tak početná jako je tomu u slavnějších konkurentů. Z toho důvodu nebyla velká tendence vytvořit specificky zaměřený nástroj. Aktuálně existují

dvě neoficiální fanoušky spravované stránky. První stránka funguje jako encyklopedie s přidruženým fórem, kde lze prohlížet všechny existující karty. Některé herní mechaniky jsou zde detailněji popsány včetně většiny karet, které se na danou herní mechaniku zaměřují. Druhá stránka umožňuje stavbu balíčků a online herní variantu s velmi nepřívětivým uživatelským prostředím.

Nákup karet zajišťuje několik zbylých internetových obchodů či velké mezinárodní aukční portály. Výměna karet probíhá osobně na turnajích organizovaných lokální komunitou. Mezinárodní transakce mezi členy skupin na sociálních sítích vyžadují nutnou porci důvěry. Tento projekt se snaží propojit většinu zmíněných funkcionalit v jednom systému.

Porovnání funkcionalit projektu a existující konkurence			
Řešení	LOTR TCG Wiki	GEMP-LOTR	Projekt
Prohlížení existujících karet	Ano	Ano	Ano
Fulltextové vyhledávání	Omezeně	Ano	Ano
Filtrování karet podle parametrů	Ne	Ano	Ano
Tvorba herního balíčku	Ne	Ano	Ano
Komunikace mezi uživateli	Ne	Ano	Ano
Ukládání kolekce	Ne	Ne	Ano
Výměna karet	Ne	Ne	Ano
Online herní varianta	Ne	Ano	V plánu
Uživatelská přívětivost	Spíše ne	Minimální	Určitě ano[1]

Tabulka 0.1: Porovnání funkcionalit projektu a existující konkurenci

Herní mechaniky

Princip zvolené karetní hry je unikátní. Každý z hráčů disponuje skupinou kladných postav a také skupinou záporáků. Kladná skupina se pohybuje po herní mapě, tvořené z devíti lokací, a snaží se dojít na konec. Soupeřova skupina záporáků se snaží jí v tom zabránit. Hlavní postava kladné skupiny má speciální dovednost a pokud je zneškodněn, jeho majitel prohrává. Každý z hráčů má tři možné způsoby, jak nad svým oponentem vyhrát – dorazit se svou kladnou skupinou na konec mapy, zneškodnit cizí hlavní postavu nebo pomocí speciální techniky snížit na dno „morálku“ hlavní postavy. Každé kolo se skládá ze sedmi částí, ve kterých jsou přesně dané akce, které každý z hráčů může provádět. Od vykládání kladných a záporných karet, přes lukostřelbu a souboj, až po přeskupení. Každou kartu jednoznačně identifikuje její kód, umístěný v pravém dolním rohu. Existuje několik typů karet, významně omezující a určující, jak lze s kartou nakládat.

Personální důvody

První karty získal autor ve školní družině na základní škole, kdy mu po nešťastné události jeden z přítomných hráčů věnoval své vodou polité karty. Po několika letech aktivního sbírání nastal útlum. Na vysoké škole autor opět karty objevil, začal znovu aktivně sbírat a přidal se i k hráčské komunitě. Po vypuknutí epidemie nastal příliv nových, často znovuobjevených, hráčů. Negativní důsledek, který trvá do současnosti, spočívá v růstu cen karet, výrazně limitující zejména hráče, protože v nové vlně přišli zejména sběratelé, kteří si chtějí zkompletovat své dávno zapomenuté sbírky. Tento fenomén se netýká pouze této hry, ale nastává ve většině velkých sběratelských her.

Teoretická část

V teoretické části se čtenář seznámí s technologiemi, díky kterým mohl projekt vzniknout. Nejprve je zmíněno dolování dat z webu, kde jsou vysvětleny funkční postupy i etické otázky této techniky. Chaotické a nekontrolované procházení a sbírání dat z webové stránky může negativně ovlivnit poskytovatele služby. Následně je nutné získaná data zpracovat do podoby, aby s ní dokázal pracovat navržený doménový model. Aby projekce a selekce uložených dat trvala co nejmenší čas, využívá se často Elasticsearch. Před ukázkou použití je popsán princip této technologie.

Konfigurace v použitém jazyce a frameworku vyžaduje určitá pravidla. Knihovny zprostředkující komunikaci mají omezené rozhraní. Podrobnosti lze najít v této podkapitole. Během výměny karet je často klíčová komunikace mezi oběma stranami. Součástí systému musí být technologie pro posílání

[1]Toto vychází z čistě subjektivního názoru autora

zpráv ideálně v reálném čase. Princip WebSocket na úrovni protokolů a implementace v daném frameworku je popsán v další části. Doporučovací systém, známý hlavně z komerčního světa internetových obchodů, lze použít i na vytváření herních balíčků. Na základě existujících karet systém nabízí další vhodné kandidáty. Jak tato technika funguje, příklady funkcí a technik vysvětlí podkapitola o podobnosti výsledků.

Následně je popsán model MVC, správné vývojové techniky a zásady. Existuje přes tři a půl tisíce různých karet. Každá má vlastní obrázek. Při znovunačtení stránky se musí často zbytečně načítat již dříve zobrazená data. Jak maximálně využít dočasnou paměť na straně klienta a jak minimalizovat datový přenos objasní poslední podkapitola.

Praktická část

Po seznámení s teoretickými principy a zásadami je představena aplikace zmíněných technik na reálném projektu. Diskutovány jsou problémy a nedostatky, které se vyskytly během implementace. Informace, získané pomocí dolování dat z webu, často obsahovaly chyby, které bylo nutné ručně řešit. Struktura se u jednotlivých typů karet dost lišila. Zpracování dat a vytváření vkládacích příkazů do databáze provázela série oprav.

Databázový model bylo nutné správně navrhnout na základě požadavků na systém a logických omezení. Některé složitější vazby již nelze navrhnout pomocnými knihovnami. Následuje popis implementované struktury od entit přes repozitáře, fasády, controller až po zobrazení v šablonách. Diskutují se použité technologie, jejich kladné stránky a nevýhody. Zda systém funguje správně hlídají integrační a unit testy.

Testování a závěr

Nad fungujícím systémem je provedeno několik typů měření, které mají za cíl zhodnotit přínos použitých technologií. Pozitivní výsledky jsou podstatné pro splnění vytyčených cílů. Na závěr jsou prezentovány budoucí cíle, zhodnocení výsledků a splnění cílů práce.

Cíle práce

Praktická část obsahuje pět cílů. Prvním cílem je implementovat funkční systém pro správu karet. Uživatel může vkládat, editovat a mazat karty ze své kolekce. Ze všech dostupných karet lze pomocí aplikace filtrů vybrat konkrétní jedince. Ze svých i abstraktních karet lze vytvořit herní balíček. V části systému Tržiště je uživatelům umožněno vyměňovat karty mezi sebou za stanovených podmínek. Jedná se tedy o požadavek na zajištění požadovaných funkcionalit.

Druhý cíl bude splněn, pokud stejný výsledek vrátí dotaz pomocí Elasticsearch technologie a původního relačního databázového dotazu. Čas vyhodnocení a vrácení výsledku bude v případě technologie Elasticsearch stejný nebo rychlejší než druhá zmíněná varianta. Pokud je uživatelům umožněno komunikovat v oddělených konverzacích či obecném vláknu, je splněn třetí cíl projektu. Komunikace musí probíhat v reálném čase bez nutného znovunačtení stránky za pomocí technologie WebSocket.

Posílání mnoha obrázků a ostatních načítaných materiálů je velmi datově náročné. Obzvlášť pokud komunikace mezi klientem a serverem probíhá pomocí mobilní sítě, může uživatel negativně pocítit absenci špatně zvolené paměťové politiky. Čtvrtým cílem je proto dosažení stavu systému, ve kterém se v případě znovunačtení stejného zdroje pošle ze serveru na klienta výrazně méně prostředků, protože klient již většinou informací disponuje z minulého dotazu. Poslední pátý cíl je splněn, pokud našeptávač, v části věnované tvorbě herních balíčků, vrací relevantní položky na základě vypočteného skóre pomocí zvoleného algoritmu. Z textu je jasná návaznost praktické části na té teoretické. Bez popsání technologií a jejich aplikace ve zvoleném frameworku by nebylo možné splnit cíle v praktické části.

Kapitola 1

Získání dat z webového zdroje

S rostoucím množstvím webového obsahu je důležité mít automatizované postupy, které dokáží získat z ohromného množství nestrukturovaných dat důležité informace. Automatické nástroje, sbírající informace o konkurenci, se mohou stát klíčovým faktorem úspěchu v globálním světě.

1.1 Dolování dat

Podmnožina obecného dolování dat se zaměřuje na získání informací z webového zdroje. Na webových stránkách se nalézají vzory pro nestrukturovaná data, aby se s nimi dále mohlo užitečně nakládat. Rozeznávají se tři základní kategorie této činnosti. Dolování struktury se pokouší získat jasný obrázek o tom, jak je na daný web odkazováno. Zkoumá, jak jsou jednotlivé stránky navzájem propojené či kam se web odkazuje. Mezi nejznámější algoritmy založené na analýze odkazů patří PageRank, Weighted PageRank a Hypertext Induced Topic Search [1].

Další kategorií je analýza použití či chování na webu. Z logovacích souborů, obsahujících informace o uživateli nebo pořadí navštívených stránek lze usoudit vzorce chování, které se mohou stát neocenitelnou zpětnou vazbou pro marketingové oddělení. Na základě získaných údajů (časů příchodu na stránku, IP adres, cookies) se může cílit reklama na různé věkové kategorie a regiony [2].

Třetí kategorií je získání obsahu z webu. Pokud chce mít obchodník přehled o cenách konkurence, může pomocí webového nástroje pravidelně kontrolovat ceny položek na jejich webových stránkách. Nástroje nemusí oklamat změna adresy, kde se položka nachází. Lze navrhnout celý proces průchodu webem od zadání názvu položky a filtrování, až po získání konkrétní hodnoty zboží. Kromě textu se dají automatizovaně získávat i multimédia [3].

1.2 Scraping a crawling

Oba zmíněné termíny spolu úzce souvisí. Za crawlera se označuje v libovolném programovacím jazyce vytvořený robot procházející webové stránky. Funguje tak, že webovou stránku analyzuje, vyhledá odkazy, uloží je pro pozdější použití a pokračuje přes odkazy na další stránky. Automaticky zavrhuje již navštívené stránky a většinou ignoruje odkazy mířící mimo aktuální web [4].

Crawlerem shromážděné odkazy využívá scraper. Scraper pomocí předem daných instrukcí ze stránky získá požadované informace. Pohybovat po stránce se může přes HTML prvky, regulární výrazy nebo selektory pro jednoznačnou identifikaci cíle [5].

Obrázek 1.1: Průběh získávání a zpracování dat z webového zdroje[41]

1.3 Zásady a etické aspekty

I když dolování dat z webového zdroje není přímo nelegální, mělo by se řídit některými pravidly, které umožní kooperaci obou zúčastněných stran. Přistoupíme-li na cizí webovou stránku za účelem získání dat prostřednictvím robota, měli bychom oznámit účel pobytu na stránce v hlavičce dotazu v poli User Agent. Z této hodnoty zjistí stránka verzi prohlížeče, operační systém a další informace. Aktivita sbírání dat by neměla negativně ovlivnit server. Z toho důvodu by se měla brát v potaz frekvence dotazů, aby nebyl server zbytečně moc zatížen. Dostali bychom se totiž skoro na úroveň DDoS útočníka. Omezení a pravidla navštívení webu jsou k dispozici v souboru robots.txt. Soubor kromě povolení a zákazů může robotovi i radit, kde hledání dat nemá smysl, protože je tam například vyžadována autentizace. V případě existence veřejného rozhraní vytvořeného pro sdílení informací, vždy bychom jej měli využít. Data získaná z dolování by měla být použita pro vytvoření nové přidané informace, aby nebyla data jednoduše duplikovaná [6, 7].

1.4 Technologie na sběr dat

Mezi nejoblíbenější nástroje ke sběru webového obsahu patří automatické prohlížečová rozšíření. Fungují na principu zaznamenání makra podle volby uživatele na stránce, které následně opakují. Užitečnou pomůckou je opakování chování uživatele v rámci průchodu webem, takže není nutné nic ručně zadávat. U počítačových programů typu Octoparse uživatel pomocí diagramových nástrojů popíše, co má robot na stránce provádět. Selekce dat k uložení probíhá pomocí dialogového okna, nebo je možné napsat selektory definující cestu k položce na stránce ručně. Po provedení příkazu se data uloží do vybraného formátu. Typické je použití kancelářského tabulkového softwaru. K získání dat ovšem není nutné použít cizí program, ale například v programovacím jazyce Python je sestavení vlastního scraping crawleru poměrně jednoduché. Díky paletě dostupných knihoven může být takový úkon zaznamenán na pěti řádcích kódu [8].

```
import requests
from bs4 import BeautifulSoup

page = requests.get("https://stranka-s-daty.com/konkretniStranka")
soup = BeautifulSoup(page.content, "html.parser")
```

Výpis kódu 1.1: Ukázka jednoduchého crawleru v jazyce Python

Kapitola 2

Architektonický vzor MVC

Architektonické vzory se navzájem liší svým přístupem ke komunikaci mezi logickými komponentami aplikace. Určují, jak má vypadat struktura aplikace ve smyslu průchodu dotazu od uživatele k datům a zpět. Protože použitá technologie Symfony se řadí mezi MVC frameworky, bude princip tohoto vzoru detailněji popsán.

2.1 Princip

Tento architektonický vzor se skládá ze tří komponent. První z nich, Model, se stará o uložení a správu dat. Na základě požadavku předává data ze zdroje, obvykle z relační databáze, ve strukturované objektové podobě. Druhou komponentou je View. Má za úkol zobrazovat připravená data uživateli. Její součástí je vše, s čím uživatel přijde do kontaktu. Požadavek vytvořený ve View je zpracován Controllerem. Tato třetí komponenta má roli zodpovědného prostředníka, který na základě dotazů z Modelu získá požadované informace a předá je do View. Nemusí se již starat o to, jak data vypadají, to má na starosti Model [9, 10].

2.2 Výhody a nevýhody

Díky přesně přiděleným úkolům mohou být jednotlivé komponenty vyvíjeny nezávisle na sobě. Díky tomu může být vývoj takové aplikace mnohem rychlejší. Důležité je jasně specifikovat rozhraní pro komunikaci mezi komponentami. Této vlastnosti se říká Loosely coupled. V případě nutnosti nahrazení jedné z komponent, například přepracování uživatelského rozhraní, není nutné pozměnit celou aplikaci. Díky oddělenému poli působnosti stačí vyměnit jednu komponentu a ostatní zůstanou nedotčené. Testovací procedury mohou být prováděny nezávisle na ostatních komponentách. Srozumitelné rozhraní ne-

musí být dobré pouze k výměně zastaralých komponent, ale umožňuje používat více různých uživatelských zobrazení zároveň [11, 12].

Nevýhodu lze vidět v robustnosti návrhu, který může být zbytečně složitý pro malé aplikace. Jednotlivé komponenty jsou navzájem nezávislé, proto je však nutné vytvořit mnoho nadbytečné logiky při průchodu aplikací [11, 12].

2.3 Struktura Symfony projektu

Jakožto MVC framework Symfony obsahuje všechny tři potřebné komponenty. Proces začíná u souborů obsahující popis mapování zdrojových dat do objektů. Takové třídy se nazývají entity. Ke každé entitě je přiřazen výchozí repozitář. Pomocí repozitáře se dotazuje na datový zdroj, který většinou vrátí výsledek v přidružené entitě. Je-li nutné na získaná data aplikovat business logiku, takové akce se provádí ve fasádách. Krajní část pomyslné modelové komponenty za pomoci oddělených služeb provádí operace s entitami a připravuje finální podobu dat [13].

Jakmile jsou data připravena, posílají se do controlleru. V controlleru by se data neměla nijak modifikovat, pouze nasměrovat do šablon. Hlavní funkcí této části spočívá ve zpracování dotazů a posílání odpovědí. Zároveň se zde kontroluje oprávnění uživatele provádět danou akci [13].

Twig je výchozí šablonovací systém pro Symfony. Z controlleru posílaná data se v šabloně mapují na HTML prvky. Zpřístupněné jsou základní programovací struktury jako podmínky a cykly. Šablony lze do sebe libolně vnořovat, díky čemuž se omezí redundance a zefektivní práce. Po provedení všech úkonů v šabloně vznikne plnohodnotný dokument, který je odeslán jako odpověď klientovi [14].

```
<!DOCTYPE html>
<html>
    <head>
        <title>Ukázka šablony</title>
    </head>
    <body>
        <h1>{{ page_title }}</h1>
        {% if app.user %}
            Hello there, general {{ app.user.uername }}!
        {% endif %}
    </body>
</html>
```

Výpis kódu 2.1: Ukázka jednoduché šablony Twig

Kapitola 3

Elasticsearch

Vyhledávání a filtrace dat na základě složitějších dotazů může u relačních databází trvat i více než několik set milisekund. Přítomnost vzoru v dlouhém řetězci je ideální příklad, na který většinou nestačí SQL syntax LIKE %vzor%. Relačně databázové řešení by vyžadovalo mohutné následné zpracování, které je u vyhledávání v reálném čase nežádoucí. Elasticsearch se od svého vzniku v roce 2010 dostal do čela nejpoužívanějších vyhledávacích nástrojů na světě [15].

3.1 Popis technologie

V Javě vytvořený vyhledávací nástroj podporuje klienty vytvořené ve většině nejpoužívanějších programovacích jazycích. Jeho jádrem je index, do kterého se pomocí mapování určitého schématu uloží data, která mohou pocházet z více zdrojů. Každý index může obsahovat více dokumentů, což je základní jednotka v systému. Data jsou reprezentovaná v JSON objektové notaci. Na základě vlastních či výchozích filtrů si uživatel definuje analyzátory, pomocí kterých se data předzpracují, aby se v nich mohlo snadněji vyhledávat. Schéma ovšem nemusí být předem určeno a systém ho vyvodí sám na základě dodaných dat. Podobá se tak částečně dokumentovým NoSQL databázím typu MongoDB. Interakce probíhá přes API umožňující základní operace s daty, včetně analýzy. Na rozdíl od nich, ale musí být zaručeno, že se nebudou mísit datové typy v jednotlivých polích (dá se přirovnat ke sloupci tabulky v relačních databázích) [16].

Instance Elasticsearch procesu se nazývá node. Nody se shlukují do clusterů, kde se podle funkce dělí na řídící nebo datové. Nody v rámci jednoho clusteru se podílí na jeho indexačních a vyhledávacích činnostech. Tato struktura procesů umožňuje vybudovat dobře replikovatelný a škálovatelný systém. Aby mohlo jeden index sdílet více serverů, rozdělí se index na tzv shardy. Díky paralelizaci se dosahuje rychlejších časů vyhledávání [16, 17].

3.2 Tvorba indexu

Index je vytvořen pomocí PUT požadavku. Kromě nastavení práce v rámci shardu (rozdělení indexu mezi více serverů), se specifikuje mapování data Při dynamické variantě systém přiřadí datový typ a základní nastavení pro pole sám. Pokud se uživatel rozhodne pro explicitní konfiguraci, nastavení se uplatní na základě jeho pokynů. Mezi základní parametry u mapování patří analyzér a ukládané položky [17].

Analyzér slouží k určení toho, jak se mají data v jednotlivých "sloupcích" předzpracovat. Obvykle se používají různé druhy stemmingu (nalezení kmene slova), odstranění stop slov (nadbytečná slova neobohacující zbytek textu o žádnou přidanou informaci), převod na malá či velká písmena či případně základní tvar slova při skloňování. Úpravy mají za cíl zpřesnit relevanci výsledku, který by mohl být například skloňováním slov negativně ovlivněn [18].

```
PUT my-index-000001
{
  "settings": {
    "analysis": {
      "analyzer": {
        "my_analyzer": {
          "tokenizer": "my_tokenizer"
        }
      },
      "tokenizer": {
        "my_tokenizer": {
          "type": "edge_ngram",
          "min_gram": 2,
          "max_gram": 10,
          "token_chars": [
            "letter",
            "digit"
          ]
        }
      }
    }
  }
}
```

Obrázek 3.1: Syntaxe pro definování vlastního analyzátoru pro Elasticsearch index[42]

U ukládaných položek je nejdůležitější uvést datový typ. I když může být typ keyword i text uplatněn na stejný řetězec, liší se přístupem při vyhledávání. Zatímco keyword vyžaduje sto procentní shodu se vzorem, text nalézá využití při hledání podřetězce nebo u našeptávače ve vyhledávači. Na každou položku může být uplatněn jiný analyzátor. Zároveň se může lišit analyzátor použití při tvorbě indexu a při vyhledávání v reálném čase.

3.3 Druhy a struktura dotazu

Na základě vytvořeného indexu a jeho struktury se uživatel může dotazovat na uložená data. Jednotlivé dotazy se mohou vnořovat do sebe a vytvořit tak libovolné logické struktury. Je-li položka typu nested, znamená to, že se jedná o objekt, který obsahuje vlastní položky. Pokud je nutné filtrovat položky zmíněného objektu, používá se speciální vnořený dotaz. Booleovská logika je aplikovatelná pomocí bool dotazu, který simuluje chování formule s klauzulemi AND a OR [17].

3.4 Knihovny pro integraci se Symfony

Na výběr je několik knihoven, které se navzájem liší hlavně mírou volnosti v tom, jaké druhy dotazů lze na Elasticsearch index posílat. Obecně jsou velmi špatně zdokumentované. Kromě stručných pokynů pro instalaci obsahují jednotlivé návody jen málo ukázkových příkladů. Knihovna FOSElasticaBundle zaujme svou jednoduchostí a silnou integrací. Možnosti tvorby dotazů jsou však omezené například u řazení dat. Velkou volnost nabízející oficiální klient vyžaduje pokročilé znalosti a postrádá uživatelskou přívětivost. Střední cestu mezi oběma zmíněnými knihovnami poskytují Elastica a Elastically [19].

Vlastní index, vytvořený pomocí knihovny FOSElasticaBundle, se definuje v souboru fos_elastica.yaml umístěného mezi ostatními soubory konfigurací ve složce config. Na obrázku níže lze vidět hlavní nastavitelné části indexu article. V kategorii settings si uživatel může definovat vlastní tokenizátory či analyzátory, které se dají následně aplikovat na položky v kategorii properties. Pokud je u položky specifikována vlastnost boost, znamená to, že se při výpočtu skóre daná položka násobně prioritizuje. Kategorie persistence zaznamenává uložení a služby starající se o načítání vstupních dat [20].

```
fos_elastica:
    indexes:
        article:
            settings:
                index:
                    analysis:
                        analyzer:
                            my_analyzer:
                                type: snowball
                                language: English
            persistence:
                driver: orm
                model: Acme\DemoBundle\Entity\Article
                provider: ~
                finder: ~
            properties:
                title: { boost: 10, analyzer: my_analyzer }
                tags:
                categoryIds:
```

Obrázek 3.2: Struktura konfiguračního souboru pro balíček FOSElasticaBundle v Symfony[43]

Kapitola 4

WebSocket

Při návrhu webové aplikace se může vývojář dostat do situace, kdy je princip klasického HTTP protokolu na škodu. Funguje totiž na principu Request - Response. Po dotazu ze strany klienta následuje odpověď serveru, přičemž nelze vytvořit obecné full-duplex spojení (server bez předchozího dotazu nemůže poslat zprávu klientovi). Existuje ovšem alternativa, která full-duplex komunikaci umožňuje [21, 22].

4.1 Popis a princip

Pojmem WebSocket se označuje skupina standardů - WebSocket rozhraní a WebSocket protokol s rozšířeními. Rozhraní pro komunikaci přes WebSocket protokol je intuitivní a jednoduché. Pouhé čtyři funkce a inicializace stačí ve webovém prohlížeči k plnohodnotné konfiguraci klienta. Rozlišují se dvě schémata. Kromě nešifrované varianty ws existuje i šifrovaná wss na principu TCP s TLS standardem zabezpečení komunikace. Velkou výhodou je automatická CORS politika. Spojení lze vytvořit pouze k URI adrese, která obsahuje jedno ze zmíněných schémat. Spojení vytvořené mezi klientem a serverem, narozdíl od HTTP, trvá do ukončení komunikace jednou ze stran. Obvykle však dříve ukončí spojení výchozí nastavení webového serveru [21, 22].

Vytvoření spojení vzniká přes upgrade z HTTP. Klient, který chce začít komunikovat, vyšle na server dotaz s několika povinnými hlavičkami. Prvním z nich je Connection s hodnotou Upgrade. Signalizuje, že klient provedl žádost o změnu protokolu, avšak potřebuje mít spojení stále otevřené. Obvyklé nastavení této hlavičky ovlivňuje, zda se po odpovědi ze strany serveru uzavře spojení či nikoliv. Další hlavička Upgrade určuje, jakým jiným protokolem by klient rád pokračoval v komunikaci. Zbylé hlavičky jsou určeny pro specifikování verze WebSocketu a k předání tajného klíče.

Po odeslání dotazu čeká klient na odpověď od serveru. V případě kladného přijetí obsahuje odpověď návratový kód 101, reprezentující změnu protokolu.

```
GET ws://example.com:8181/ HTTP/1.1
Host: localhost:8181
Connection: Upgrade
Pragma: no-cache
Cache-Control: no-cache
Upgrade: websocket
Sec-WebSocket-Version: 13
Sec-WebSocket-Key: q4xkcO32u266gldTuKaSOw==
```

Obrázek 4.1: Detail požadavku na použití WebSocketu [46]

Hlavičky jsou podobné jako je tomu v případě dotazu. Obě strany, klient i server, si uchovávají vygenerovaný tajný klíč druhé strany. Po přijetí odpovědi klientem je spojení zahájeno.

Data z jednotlivých zpráv se posílají ve formě rámců. Velikost hlavičky se pohybuje mezi dvěma a čtrnácti byty v každém rámci. Pro představu HTTP/1 dotaz obsahuje pět set až osm set bytů dodatečných informací. Kvůli kompresi se tato hodnota u HTTP/2 zmenšila. Dokáže se snížit až na hodnotu osmi bytů. Spojení mezi klientem a serverem se uzavře po odeslání ukončovacího rámce [22, 21, 23].

4.2 Použití

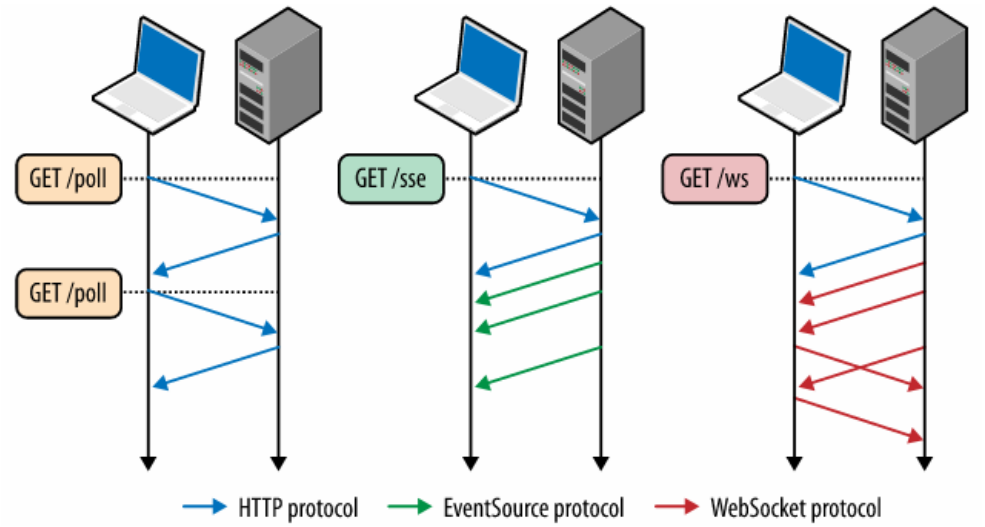
Po objasnění principu WebSocketu nastává otázka, kdy se vyplatí ho použít. Hlavní výhoda se nalézá v obousměrné komunikaci, kdy může server posílat data klientovi, aniž by byl nucen čekat na dotaz z druhé strany. Pro jakoukoliv aplikaci, která vyžaduje komunikaci v reálném čase, se hodí použití WebSocketu. Prvním příkladem je hra v prohlížeči. Pokud je zásadní odezva mezi klientem a serverem, musel by se klient neustále ptát serveru na aktuální situaci. Díky tomu by byl server zahlcen požadavky. Obzvlášť, pokud by se jednalo o hru pro více hráčů. Pravděpodobně by docházelo k výrazné redundanci dat.

Dalším příkladem může být chatovací aplikace. Zprávy od ostatních by si uživatel zobrazil až po znovunačtení stránky. Posledním příkladem může být aplikace z burzovního prostředí. Podobně jako u herní aplikace, rychlost odezvy na změnu trhu může být markantní [22, 21].

4.3 Alternativy

Jak problém obousměrné komunikace v reálném čase vyřešit bez použití WebSocketu? Existuje několik alternativ. Prvním z nich je Polling. Jedná se o techniku, která nepotřebuje jinou technologii než HTTP. Obousměrná komunikace je simulována několika způsoby. Opakování dotazů klientem v přesně definované intervalu umožní serveru informovat klienta o změnách. Čím menší

je interval, tím víc zpráv je nutné si po kanálu vyměnit. Z toho vyplývá datová náročnost, hrozba zahlcení serveru a mnoho zbytečných dotazů, protože server pravděpodobně nebude mít žádné nové zprávy k dispozici [23].



Obrázek 4.2: Vizualizace komunikace pomocí WebSocketu a jeho alternativ [45]

Long Polling se od předchozího příkladu liší v několika částech. Klient se nedotazuje v pravidelných intervalech serveru, ale server vyčkává s odpovědí do poslední chvíli než by nastal timeout. V případě absence nové informace v časovém limitu oznámí tuto skutečnost server klientovi, který ihned odešle další dotaz.

Ačkoliv pomocí Server Sent Events nelze dosáhnout obousměrné komunikace jako v případě WebSocketu, umožňuje serveru poslat klientovi několik zpráv za sebou. Jedná se vlastně o vytvoření jednosměrného proudu od serveru klientovi. Oproti pollingu, kde kvůli zabezpečení komunikace a nutnosti jak dotazu klienta, tak odpovědi serveru, trvá získání dat násobně kratší dobu. V případě vysoké redundance dat je rozdíl mezi pollingem a Server Sent Events nejvíce markantní [24, 23].

4.4 Integrace v Symfony

Pro práci s technologií WebSocket ve frameworku Symfony se využívá knihovna Ratchet. Server tvoří čtyři metody vyžadované pomocí rozhraní. Metody určují, jak se má server zachovat v případě vytvoření spojení, příchozí zprávy, ukončení spojení a chyby. Nová spojení je vhodné ukládat do libovolné struktury, aby příchozí zpráva mohla být doručena svému příjemci [25].

```php
<?php
namespace MyApp;
use Ratchet\MessageComponentInterface;
use Ratchet\ConnectionInterface;

class Chat implements MessageComponentInterface {
    public function onOpen(ConnectionInterface $conn) {
    }

    public function onMessage(ConnectionInterface $from, $msg) {
    }

    public function onClose(ConnectionInterface $conn) {
    }

    public function onError(ConnectionInterface $conn, \Exception $e) {
    }
}
```

Obrázek 4.3: Ukázka jednoduchého rozhraní pro komunikaci pomocí WebSocket v Symfony [47]

Kapitola 5

Dočasná paměť

V případě opakovaného přístupu uživatele na stránku může načtení zdrojů trvat razantně méně času oproti prvnímu pokusu. Můžou za to pravděpodobně různé druhy dočasných pamětí. Všechny moderní druhy prohlížečů umožňují použití dočasné paměti jako prostředku pro načtení již v minulosti načtených informací.

5.1 Druhy dočasné paměti

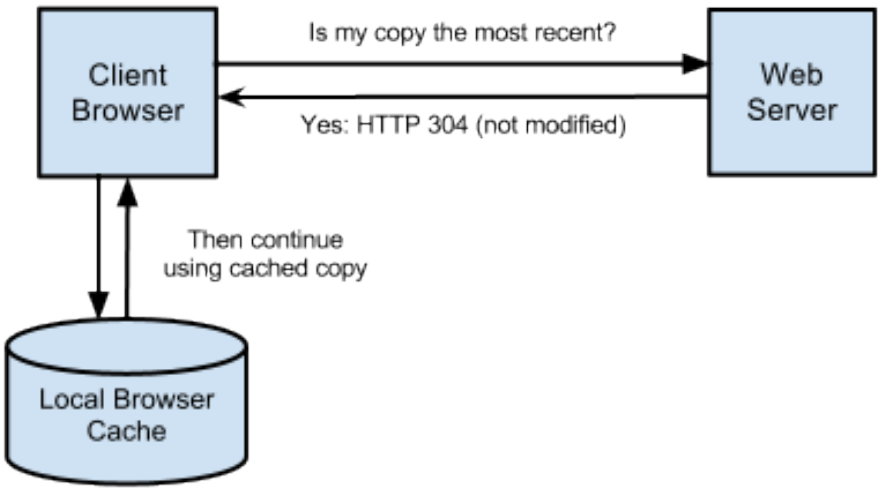
Dočasné ukládání výsledku probíhá na několika úrovních. První úroveň se nachází přímo na serveru a slouží jako úložiště mezivýsledků a náročných operací. Paměti se dělí do oddělených skupin, tzv. pools, které mohou mít rozdílné typy ukládání dat a nastavení. V rámci skupiny se jednotlivé položky rozlišují pomocí unikátních klíčů. Aby se mohlo smazat více spolu příbuzných klíčů naráz, lze položkám přidat značky a adresovat operace přes ně.

Další úrovně souvisí s ukládáním odpovědi ze serveru. Patří sem prohlížečová, proxy a výstupní (gateway či reverse proxy) dočasná paměť. Nejblíže uživateli je umístěna ta prohlížečová. Slouží jako mezipaměť například při zpětném pohybu v prohlížených stránkách. Použití dočasné paměti se zde rozhoduje na základě čerstvosti zdroje, která je většinou porovnávaná jednou za relaci. Proxy slouží na podobném principu ve velkém měřítku. Tyto dočasné paměti jsou sdíleny mezi více uživateli a vrací přednačtený zdroj, který si mohl vyžádat jiný z uživatelů. V případě zabezpečeného dotazu pomocí HTTPS se sdílená paměť nepoužívá. Na rozdíl od sdílených dočasných pamětí se ty výstupní nacházejí blízko webového serveru. Většinou se používají i pro lepší škálovatelnost a dostupnost zdroje [26, 27, 28].

5.2 Princip

Na základě definovaných pravidel se správa dočasné paměti rozhodne, zda je možné uživateli vrátit odpověď, která byla na základě politiky dočasné paměti uložena pro budoucí použití. Při nesprávně použité dočasné paměti hrozí, že budou uživateli zobrazeny neaktuální informace. Po odeslání požadavku na adresu od uživatele se zkontroluje čerstvost a validita zdroje. Čerstvost spočívá ve specifikování časového údaje, ve kterém je zdroj považován za bezpečně použitelný. Ovlivnit se dá pomocí hlavičky Cache-Control: max-age nebo Expires. Při odpovědi ze serveru může být tato hodnota specifikována buď jako počet sekund nebo datum do vypršení čerstvosti zdroje [29].

Validita v dočasné paměti funguje na základě zjištění modifikace zdroje. Může být určena vygenerovaným řetězcem nebo podobně jako čerstvost datem. Hlavička dotazu, obsahující řádek s časovým údajem (If-Modified-Since) či identifikátorem (If-None-Match), se porovná s časem poslední změny/identifikátorem zdroje. Pokud je zjištěna neaktuálnost, je vrácena nová verze zdroje. Ideální je použití kombinovaného přístupu, kdy se nastaví kratší čas čerstvosti v kombinaci s validační podmínkou na změnu zdroje [30].



Obrázek 5.1: Schéma validačního přístupu u dočasné paměti [48]

5.3 Dočasná paměť v Symfony

Práce s dočasnou pamětí v Symfony se dá rozdělit na dvě hlavní kategorie. První pracuje na úrovni HTTP dotazů a odpovědí, kde se specifikuje čerstvost a validita zdroje. Druhá slouží spíše pro uložení výsledků náročnějších výpočtů uvnitř PHP funkcí [31].

5.3.1 Aplikační dočasná paměť

Opakuje-li se často výpočetně náročná operace, jejíž výsledek z nějakého důvodu nelze uložit trvale, vyplatí se uložit výsledek do aplikační či systémové dočasné paměti. Systémová paměť slouží pro anotace, serializace a validace. Aplikační paměť je určena pro běžné použití v kódu např. ve fasádě. Položky v paměti jsou v tzv. poolech [31].

5.3.2 HTTP dočasná paměť

Dle výše zmíněných principů lze vložit mezi hlavičky odpovědi pravidla o dočasné paměti. Zpřístupnění těchto možností je silně doporučeno provádět pouze u bezpečných metod jako je GET, HEAD a OPTIONS. Jakmile se má upravovat stav objektu, není vhodné dočasnou paměť používat. Pokud se zobrazovaná stránka skládá ze statické a dynamické části, existuje způsob, jak pravidla o dočasné paměti rozdělit. Technologie pro více nastavení zároveň se jmenuje ESI (Edge Side Includes). Pomocí syntaxe v šabloně render_esi vložíme blok, který může mít například rozdílnou čerstvost než zbytek stránky [32, 33].

```
{# templates/static/about.html.twig #}

{# you can use a controller reference #}
{{ render_esi(controller('App\\Controller\\NewsController::latest', { 'maxPerPage': 5 })) }}

{# ... or a URL #}
{{ render_esi(url('latest_news', { 'maxPerPage': 5 })) }}
```

Obrázek 5.2: ESI technologie v Twig šabloně [49]

5.4 Miniatury v Symfony

Jednou z hlavních věcí, které se uživateli nevyplatí načítat znovu a znovu, jsou obrázky. Na tyto prvky existuje v rámci frameworku výchozí politika s dočasnou pamětí, takže není potřeba nic nastavovat. Problém může být s nastavením webového serveru, kde v některých případech musí být explicitně nastavena expirace médií a jiných statických zdrojů. Pokud chceme ale zobrazit například na mobilním zařízení jinou kvalitu obrázku než na běžném desktopu, musela by být přítomna dvojice stejných obrázků. Knihovna LiipImagineBundle umí za běhu vytvářet různě modifikované verze obrázků (různá velikost, ořez, apod.) či provádět post-processing [34].

Kapitola 6

Doporučovací systémy

Od uživatele jakéhokoliv systému se nemůže očekávat kompletní znalost obsahu, který je mu zobrazován. Ať už se jedná o výběr filmů na online streamovací platformě, články na internetové encyklopedii nebo nabídku obchodu se žárovkami. Pomocí doporučení dalšího obsahu můžeme uživatele přimět, aby předčasně neopustil systém. Když mu zobrazovaný produkt z nějakého důvodu nevyhovuje, trochu odlišný výrobek může být ideální [35].

6.1 Princip

I když se jednotlivé doporučovací systémy liší, obecný princip mají stejný. Na základě získaných informací, ať už z historie chování uživatele nebo z vlastností porovnávaných položek, systém pomocí zvoleného algoritmu vypočítá podobnost ostatních položek a nabídne uživateli x nejrelevantnějších výsledků. Proces výběru výsledků se dá shrnout do tří částí. V první části se snaží systém nashromáždit co nejvíce relevantních informací. Informace obvykle získává z historie uživatele nebo metadat o produktech. Tyto informace ovšem nemusí s aktuálním uživatelem vůbec souviset a může se jednat o aktuální globální trend, který určuje relevanci položek. Během druhé fáze systém aplikuje algoritmus na získaná data, ze kterých ve třetí fázi určí výsledek. Zásadním požadavkem na systém je relevance a čas, za který dokáže podobné položky nalézt [35, 36].

6.2 Hlavní typy doporučovacích systémů

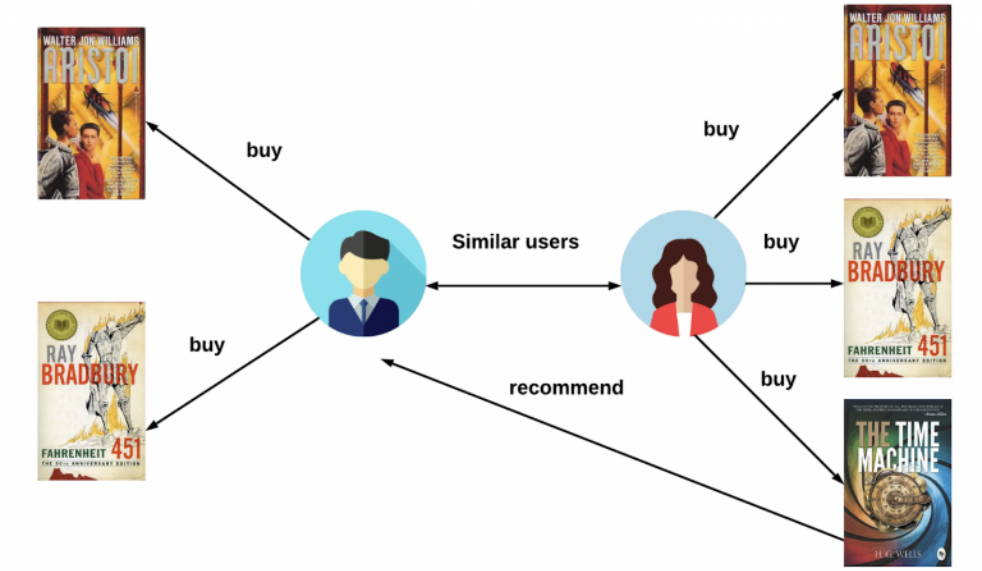
6.2.1 Zaměření na globální trendy

Pro mnoho uživatelů může být klíčové, jak je daný produkt oblíbený nebo kupovaný ostatními uživateli. Dává smysl, že pokud si daný druh mobilního telefonu koupilo mnoho lidí, někteří tak učinili na základě doporučení od ostat-

ních majitelů. Tato informace však nemusí být vůbec relevantní. Na rozdíl od jiných typů doporučovacích systémů netrpí z počátečního nedostatku informací. Výsledek však ztrácí na relevanci. Díky globálnímu principu se systém nedokáže zaměřit na potřeby specifického uživatele [35, 38].

6.2.2 Kolaborativní filtrování

Doporučení vhodných kandidátů je prováděno na základě úvahy, že osoby s podobnými preferencemi a chováním na webu mají větší pravděpodobnost sdílení společným zájmů. Pokud uživatelé například viděli podobné skupiny filmů ve videotéce, systém doporučí filmy prvního uživatele tomu druhému. Pro tento typ systému je klíčová historie chování uživatelů. Po čistém startu nejsou k dispozici žádní podobní uživatelé, a proto doporučení nemůže být příliš relevantní [37, 38, 39].



Obrázek 6.1: Schéma kolaborativního filtrování [40]

6.2.3 Filtrování se zaměřením na obsah

Tento přístup nepotřebuje znát podobně se chovající uživatele. Na základě parametrů položek z uživatelovi historie určí podobné položky k doporučení. Díky doporučení na základě podobnosti produktů uživatel ztrácí diverzitu výsledku. Aby systém fungoval, je nutné mít dobře kategorizovatelné a hodnotitelné položky. Například filmové snímky nebo hudební nahrávky zmíněné vlastnosti postrádají [37, 38, 39].

6.2.4 Hybridní systémy

Kvůli nedostatkům jednotlivých systémů se jejich algoritmy často spojují, aby byl výsledný hybrid připraven i na rizikové scénáře. Neexistuje-li dostatečně velká skupina podobných uživatelů, mohou být pro začátek více relevantní metadata položek [37, 38, 39].

6.3 Řešení krajních případů

Pokud na stránku dorazí zákazník bez předchozí historie, vhodným řešením je mu doporučit globálně populární či parametry podobné položky. Tomuto scénáři se říká podle automobilové terminologie start za studena. Pokud uživatel časem mění své chování, nehodí se mu nabízet stále stejné položky, což je častým problémem filtrování se zaměřením na obsah. Místo toho lze do výsledku umístit náhodně zvolené předměty, které se mohou uživateli zalíbit. Neměl by však nabýt dojmu, že doporučení nedává smysl [38, 39].

6.4 Kvalita výsledku

Pomocí různých druhů měření lze zkoumat kvalitu doporučeného obsahu. Mezi hlavní metriky patří přesnost a výtěžnost. Zatímco přesnost určuje poměr správných typů k těm špatným, výtěžnost se zaměřuje na nalezená správná řešení vzhledem k těm, které se měly ve výsledku objevit [35].

Kapitola 7

Návrh systému a databáze

Po nabytí potřebných vědomostí bylo možné začít vytvářet webovou aplikaci. Před získáním dat se musí připravit struktura databáze, do které se data následně nahrají. Kvůli testování se vytvořilo jak standardní dotazování do relační databáze, tak i to na Elasticsearch index. Tvorba systému probíhala napříč vrstvami kontinuálně, kdy se připravila funkčnost jedné části systému. Nebyly tudíž naráz vytvořeny všechny controllery, uživatelská rozhraní a služby. Pouze základní nastavení pro mapování entit bylo vytvořeno současně.

7.1 Použité technologie

Aplikace byla vytvořena v programovacím a často zavrhovaném jazyce PHP (verze 8.0.17), konkrétně v již výše avizovaném frameworku Symfony (verze 5.4.6). Pro zpracování stylů byla využita čistě JavaScriptová knihovna WebpackEncore. Umožňuje pre-processing stylů a skriptů do jednotlivých souborů. Urychlení dotazování zajišťuje Elasticsearch (verze 7.10.1). Kde se na serveru nepoužívá na dotazování Elasticsearch, zajišťuje načtení dat databázový systém SQLite (verze 3.34). Reálná komunikace by nebyla možná bez WebSocket technologie (verze 13). Získání dat z webového zdroje proběhlo za použití aplikace Octoparse. Její intuitivita, rychlost a free trial verze velmi usnadnila celý proces. Použité technologie byly zvoleny na základě předchozích zkušeností autora (kromě Octoparse). Protože obrázky karet obsahují licencovaný materiál, byly pomocí Batch Image Manipulation Pluginu pro program GIMP modifikovány výraznou pixelizací.

7.2 Hlavní logické části aplikace

Podle plánu se webová aplikace skládá ze čtyř hlavním částí. Pokud si chce uživatel rychle prohlédnout seznam všech existujících karet, během několika

přesměrování a akcí by měl v rozumném čase nalézt libovolnou kartu na základě dostupných filtrů či fulltextového vyhledávání. Výsledek dotazu jakožto kombinace filtrů se zobrazí v uživatelsky přívětivé obrazové variantě nebo v seznamu, který nebude tak intuitivní, ale dá se v něm podle identifikátoru karty bez problémů orientovat. První část bude tedy fungovat jako internetová encyklopedie existujících karet - wiki.

Druhou částí bude uživatelská kolekce. Správa vlastní sbírky ulehčí práci i čas, protože jednoduše označí karty, které uživatel ještě potřebuje získat, protože mu chybí do herních balíčků nebo jako položka do sbírky. Sekce bude umět to stejné co encyklopedie - na základě filtrování zobrazit požadované karty - ovšem přidá k nim možnost správy reálné instance karty, která zachycuje stav karty. Hráč si bude moci nahrát vlastní verzi přední i zadní strany, aby případný kupec věděl, jaké jsou přesné nedostatky potencionálního zboží.

Na základě uložených karet může hráč v sekci Tvorba balíčku vytvářet herní balíčky. Protože ne každý disponuje všemi kartami, mohou být použity i abstraktní verze, ze kterých se vytvoří jednoduše rozpoznatelné alternativy k herním kartám. Součástí tvorby balíčku je i našeptávač, který na základě již vložených karet doporučí tři nejvhodnější na základě vypočteného skóre.

Poslední částí je tržiště. Sběratelské karetní hry by bylo nesmírně náročné sbírat bez pomoci ostatních. Pokud by chtěl sběratel v této hře sesbírat všechny foil karty (vzácná blýskavá varianta karty) z první série, musel by v balíčcích nalézt 365 různých karet. Pravděpodobnost nalezení náhodného foilu v balíčku je 1:6. Originální pořizovací cena balíčku v dobách aktivní distribuce se pohybovala kolem 110 korun. Jenom nalézt 365 foilů by bylo finančně náročné, nemluvě o tom, že by se karty mohly opakovat. Proto je více než doporučeno samotnými distributory, aby si lidé karty mezi sebou vyměňovali. Jednáte-li o obchodu s několika různými lidmi z celého světa, může se jednoduše stát, že ztratíte pojem o tom, jaké karty máte na výměnu k dispozici, které jste již úspěšně vyměnil a které stále na vaši péči stále čekají. Tržiště umožní kupovat, prodávat a vyměňovat karty. Nefiguruje jako zprostředkovatel platby na oficiální úrovni, pouze se ujímá role prostředníka pro obchod.

Zájemce do nového obchodu přidá žádané karty a zkusí navrhnout své na výměnu. Pokud nevlastní žádné ze žádaných protistrany, může navrhnout nákup za peníze v libovolné měně. Po odeslání žádosti může protistrana přijít s upravenou nabídkou, pokud se jí například nelíbí navrhovaná cena či poměr karet. Jakmile jsou obě strany spokojeny, po potvrzení výměny karty automaticky změní majitele a už je na uživatelích, aby si karty fyzicky poslali a provedli případné finanční transakce v online nebo offline podobě.

Hlavní přednost systému oproti konkurenci se skrývá v manipulaci s kolekcí. Uživatelské karty jsou skutečně uloženy v databázi a mohou být vyměněny. Na rozdíl od čistě virtuálních karetních her není cílem, aby uživatel postupně získával neexistující karty, ale aby reflektoval skutečný stav své sbírky. Plánovaná online herní varianta přidá systému úplně nový rozměr.

7.3 Analýza a návrh databáze

Před samotným vytvořením databáze bylo nutné zhodnotit, jak se systém bude používat. Pomyslným "srdcem" databáze se dá označit tabulka Card. Z této tabulky se odvíjí většina funkcionalit, jelikož hlavní činností aplikace pravděpodobně bude provádět základní operace s uživatelskou kolekcí, včetně prohlížení všech dostupných karet. Aby si hráč mohl spravovat svou kolekci karet, bylo nutné oddělit abstraktní verzi karty, popisující vlastnosti karty po stránce herní, od fyzické kopie v hráčově sbírce.

7.3.1 Uložení vlastností karet

Jedním z důležitých rozhodnutí bylo určit umístění parametrů a vlastností karet. Herní karty jsou různě vzácné. Zatímco běžné karty se v náhodném balíčku o jedenácti kartách vyskytovaly šestkrát a méně běžné čtyřikrát, v balíčku nebyla nikdy více než jedna vzácná. Vzácné karty jsou obecně silnější, ale získat je ve větším počtu logicky není snadné. Parametr vzácnosti nabývá jednu ze čtrnácti různých hodnot, z toho šest existuje jen u malého počtu karet. V případě uložení hodnoty přímo do tabulky Card by došlo k velké redundanci dat. Jelikož se jedná o jeden z parametrů, podle kterého budou karty často filtrovány, není nejlepším řešením výsledek omezovat podle rovnajících se textových řetězců. Místo toho byla i za konzultace s vedoucím práce zvolena varianta číselníkových pomocných tabulek, kdy je hodnota v tomto případě vzácnost v tabulce Card reprezentována ve formě identifikátoru řádku nově vzniklé tabulky Rarity. Přímá varianta by zajistila jednodušší schéma databáze. Nad takto duplikovanými hodnotami by se vyplatilo vytvořit index vhodný pro řádky s malou kardinalitou. Příkladem může být třeba bitmapový index.

7.3.2 Srdce databázového návrhu

Stejně jako v uvedeném příkladu se vzácností karet se problém vyřešil u jazyku, série, pečeti a kultury. Skutečné uživatelské karty jsou uloženy v tabulce RealCard. Pomocí cizího klíče je připojen konkrétní uživatel, kterému karta patří. Aby mohl uživatel nahrát vlastní obrázek karty, tabulka je doplněna o odkazy na uložení nahraných obrázků. Rovněž se zde nachází cizí klíč tabulky Condition značící stav konkrétní karty. Tabulka Card je ve vztahu s RealCard pomocí vazby One-To-Many. Jeden řádek z tabulky Card je vzorem pro více záznamů v tabulce RealCard. Každý řádek v tabulce RealCard náleží právě jednomu záznamu v tabulce Card.

7.3.3 Card Type

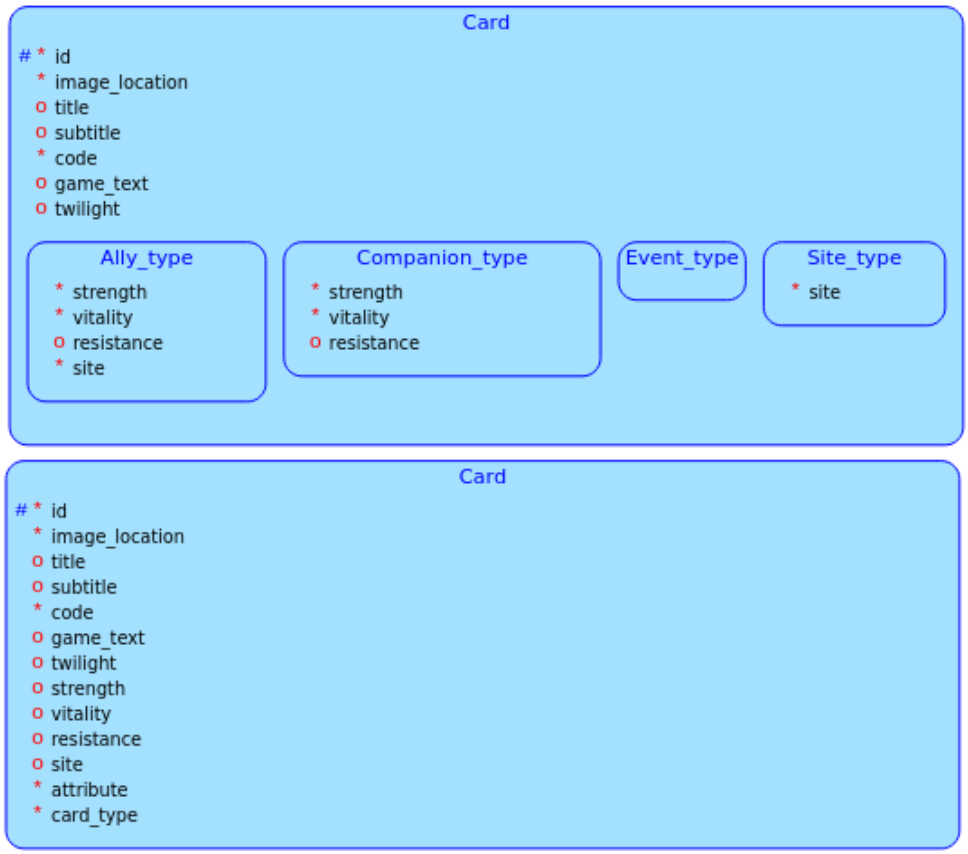
Kromě umístění parametrů bylo důležité rozhodnout, jak se naloží s typy karet. Každá karta má právě jeden specifický typ. Parametry karty se razantně liší v závislosti na typu karty. Postava disponuje cenou vyložení, útokem, životy a pečetí, zato země obsahuje pořadí na mapě a twilight číslo (herní měna produkovaná kombinací hodných postav a aktuální země na mapě, kterou spotřebovává soupeřova armáda záporáků). Na základě typu karty bude umožněno filtrování, ale její skutečný potenciál bude v budoucím rozšíření systému o online herní variantu. Podle definovaných požadavků se jevilo jako správné řešení použití entitního ISA vztahu (k nadřazenému typu existuje několik závislých podtypů).

Existují tři varianty aplikace této logické struktury pro databázové použití. V prvním případě se pro každý podtyp vytvoří nová tabulka, zatímco nadtyp vlastní tabulku nemá. Druhou možností je kombinace, kdy je mezi tabulkou nadtypu a podtypu přidána vazba. Třetí možnost zachovává pouze tabulku pro nadtyp, která bude uchovávat všechny možné atributy podtypů, ovšem některé z nich budou mít hodnotu NULL. Vítěznou realizací se stala třetí možnost, protože se parametry podtypů často překrývají, není jich celkově mnoho a existují společné parametry pro všechny podtypy. Kombinovaná varianta byla zamítnuta kvůli tomu, že často prováděnou operací v systému bude filtrování podle typu a některých atributů. Vyplatí se data držet v jedné tabulce, kde se jednotlivé typy rozeznávají podle rozdělovacího sloupce.

7.3.4 Herní balíček

Herní balíček se skládá z vlastní karet uživatele. Pokud uživatel dané karty nevlastní, ale má v úmyslu si je pořídit, ocenil by vizualizaci, které karty mu do reálného balíčku stále chybí. Proto bylo rozhodnuto, že balíčky půjde tvořit i z abstraktních karet. Tabulka RealCard byla rozšířena o sloupec is_real, aby byly rozlišeny karty určené pouze do balíčku. Dochází tak k paradoxní situaci, že v tabulce s reálnými kartami budou i falešné (označení Real Fake je termín často zobrazovaný v současné seriálové a filmové tvorbě). Jednoduchá struktura tabulky Deck obsahuje název balíčku a cizí klíč tabulky User, která reprezentuje uživatele v systému. Mezi tabulkami Deck a RealCard existuje M:N vazba dekomponovaná přes vazební tabulku bez přidaných atributů.

Jednotlivé série, ve kterých karty vycházely, patří do různých herních bloků/formátů. Jedná se o limitování karet, které lze v daném formátu hrát. Pro jednotlivé formáty existuje i specifický banlist (seznam zakázaných nevybalancovaných karet). Aby měl uživatel přehled, na jaké herní formáty může balíček uplatnit, měla by mu být zobrazen seznam kompatibilních formátů. Kdyby se tato informace získávala z databáze, musel by se seznam po každém načtení stránky provádět složitý výpočet na základě obsažených karet v balíčku. Tato informace je totiž přítomna i na přehledu balíčků, ne pouze



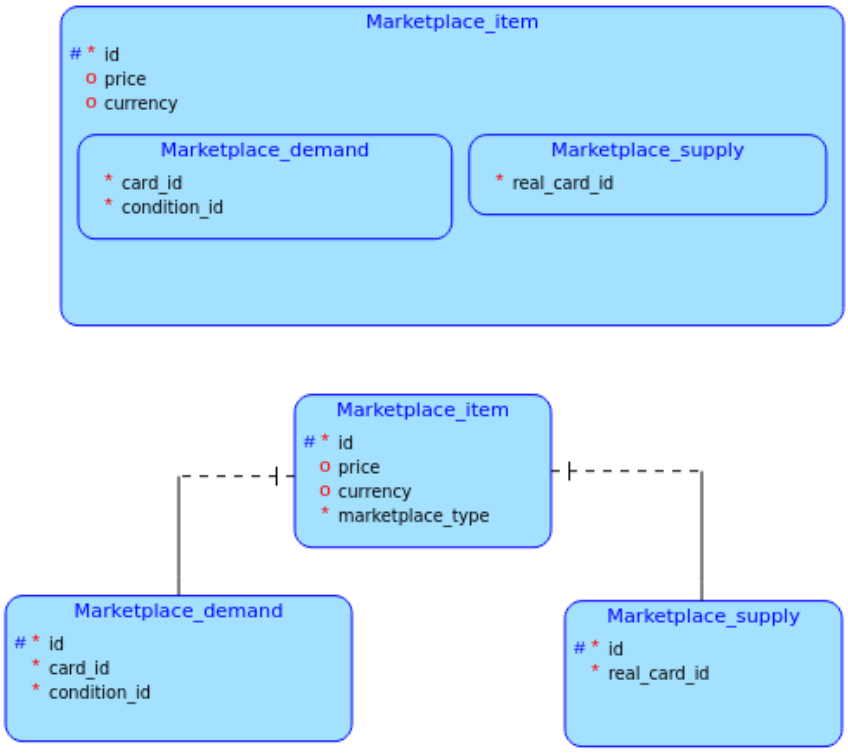
Obrázek 7.1: Dekompozice ISA entitního vztahu do jedné tabulky

při jejich vytváření. V obou případech je nutné prověřit seznam po každé změně balíčku.

7.3.5 Tržiště

Na tržišti hráči mohou nabízet své karty a shánět nové. V případě absence karet, které potřebuje protistrana existuje možnost nabídnout finanční ekvivalent. Obchod funguje ve třech krocích. V prvním si hráč na tržišti vybere o hráče karty, které chce. Poté si může vyhledat, které karty shání prodejce. Vlastní-li hráč dané karty v požadované kvalitě, může je přidat na druhou stranu obchodu. Při vkládání karet na tržiště si hráč rozmyslí, zda chce karty pouze měnit, nebo požaduje finanční obnos. Rozdíl mezi kartami, který je nutný finančně dorovnat, je zobrazen na dané straně obchodu. Jakmile je s obchodem hráč spokojen, odešle žádost o výměnu/prodej druhé straně. Ta může obchod odmítnout, zaslat protinávrh, nebo obchod potvrdit.

Položka na tržišti je reflektovaná v entitě MarketplaceItem. Může se jednat buď o nabídku nebo poptávku. Situace je podobná jako u karetního typu. Protože zde jednotlivé slabé typy obsahují cizí klíče, které druhý nemá, v tomto případě se realizovalo vytvoření tabulek pro subtypy a ponechání společných atributů v tabulce nadtypu. Poptávka obsahuje cizí klíč informující o stavu, ve kterém hráč kartu shání. Nabídka obsahuje odkaz na reálnou kartu prodejce.



Obrázek 7.2: Dekompozice ISA entitního vztahu na rodič-potomek

Tabulka TradingProposal představuje nadřazenou entitu určenou pro jednorázovou nabídku mezi dvěma hráči. Každý návrh má několik poptávek a nabídek, autora a příjemce. Zároveň je zaznamenáno, kdo by měl zaplatit případnou dodatečnou částku ve vybrané měně. Pokud se příjemci nabídky některý aspekt návrhu nelíbí a odešle protinávrh, vytvoří se nový záznam v tabulce TradingProposal, avšak existuje vztah mezi aktuální verzí návrhu a návrhem předchozím.

Jakmile je obchod finalizován, uloží se informace o prodané kartě do tabulky Deal. Tato na první pohled zbytečná operace má svůj důvod. Když do hry přijde nový člen, nemá pravděpodobně povědomí o hodnotě jednotli-

vých karet. Může se tak stát obětí potenciálního podvodníka, který využije neznalosti nováčka. Z tabulky Deal se rychle získá průměrná cena, za kterou byla karta koupena. Samozřejmě systém není bez chyb, ale případné odhalení pachatelů bude vyžadovat jen několik databázových dotazů. Bez existence oddělené tabulky pro finalizované transakce by bylo nutné provádět dotazy nad tabulkou s velkým počtem záznamů.

7.3.6 Uložení zpráv

Aby spolu mohli uživatelé komunikovat, bylo nutné vytvořit promyslet ukládání zpráv do databáze. Zvolilo se řešení se zaznamenaným autorem a příjemcem zprávy ve formě cizích klíčů z tabulky User. Komplikovat situaci s nadřazenou entitou konverzace by bylo zbytečné. Dále jsou důležitou součástí entity parametry, kam se ukládá obsah zprávy a status o přečtení zprávy.

Kapitola 8

Získání a příprava dat

Jakmile byl model připraven, který se ovšem za pochodu ještě mírně změnil, a spuštěny databázové migrace, mohly se jednotlivé tabulky naplnit daty. Jak jednotlivé obrázky karet[44], tak informace o nich byly získány z online encyklopedie provozované fanoušky této hry.

8.1 Analýza a získání dat z webového zdroje

Jednoduchá struktura online encyklopedie výrazně ulehčila vytvoření plánu na získání dat. Z globálního indexu, kde jsou karty seřazené podle jednoznačného identifikátoru v podobě karty, je po kliknutí na jméno karty návštěvník přesměrován na detail karty. Mezi stránkami s detaily o kartách existuje přímá sekvenční cesta ve formě šipek. Počátečním místem se stala detailní stránka o první kartě na seznamu. Po získání dat na stránce se pomocí zmíněného odkazu přesměrovalo na další kartu a proces se opakoval, dokud byl na stránce daný odkaz přítomný. Jeho absence indikovala návštěvu poslední karty a indikátor k ukončení procesu.

I když plán získání dat nebyl příliš komplikovaný, rozhodl se autor použít existující program sloužící k dolování webového obsahu Octoparse. Hlavní výhodou programu je uživatelská přívětivost a intuitivita. Bezplatná omezená varianta programu bohatě stačila pro účely práce. Po provedení definovaných úkonů se data uložila v uživatelem specifikované podobě. Jako první byly staženy obrazové materiály. Podle zmíněného procesu se začátkem na prvním detailu karty byl získán obsah html prvku. Obrázek byl pojmenován tvůrci encyklopedie jako složenina série, ve které karta existuje, reprezentovaná dvojciferným číslem a číslem karty v série reprezentovanou trojciferným číslem. Obrázky byly uloženy v kompresním formátu JPEG.

Po získání obrazového materiálu následoval podobný proces s informacemi o kartách. Zde se objevil velký problém. Na základě typu karty byly na stránce přítomny mnohdy úplně rozdílné informace. Řešením se ukázalo

```
<font size="5">
  <b>
    The Lord of the Rings TCG Wiki:
    <a href="/wiki/lotr00009?do=backlink" title="Backlinks">Phial of Galadriel (P) (0P9)</a>
  </b>
 </font>
</center>
<div class="page">
  <!--wikipage start-->
  <h1> ··· </h1>
  <div class="level1">
    <div class="wrap_dbshell plugin_wrap">
      <div class="wrap_center plugin_wrap" style="width:990px;">
        <div class="wrap_center plugin_wrap" style="width:240px;"> ··· </div>
        <div class="wrap_db wrap_rightalign plugin_wrap" style="width:500px;">
          <p>
            <span class="curid">
              <a class="media" href="/wiki/lotr00009" title="lotr00009">
                <img class="media" src="/wiki/_media/cards:lotr00009.jpg" title="Phial of Galadriel (P) (0P9)
                Galadriel (P) (0P9) Card Image">
              </a>
            </span>
          </p>
        </div>
        <div class="wrap_db2 wrap_leftalign plugin_wrap" style="width:480px;">
          <div class="table sectionedit1">
            <table class="inline">
              <tbody>
                <tr class="row0 roweven">
                  <td class="col0">
                    <a class="wikilink1" href="/wiki/set" title="set">Set</a>
                    :
                  </td>
                  <td class="col1">
                    <a class="wikilink2" href="/wiki/promotional" title="promotional" rel="nofollow">Promotion
                  </td>
                </tr>
```

Obrázek 8.1: Detail zdrojového kódu webového zdroje

do sloupců ve výsledném tabulkovém procesoru uložit všechny řádky tabulky přítomné na stránce detailu karty. Protože některé důležité parametry karet byly obsažené pouze v indexovém seznamu, poslední proces se cyklil na každém řádku seznamu, odkud získal jednoznačný identifikátor karty a k němu potřebné informace. Jak údaje z detailů karet, tak ze seznamu, byly uloženy do tabulkového procesoru.

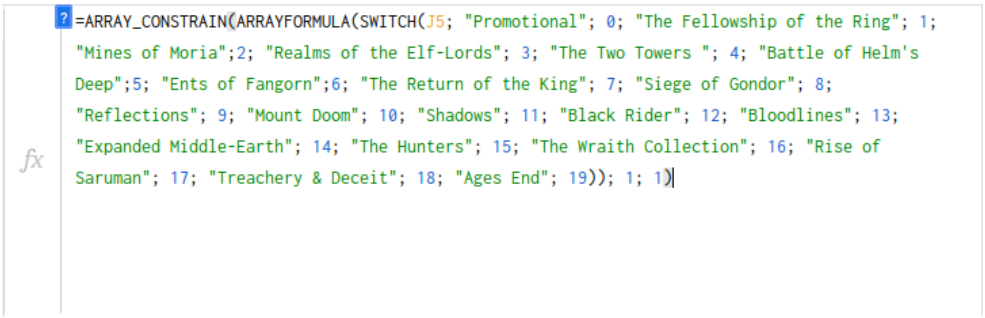
8.2 Zpracování dat a příprava skriptů

Z encyklopedie získané obrázky se přesunuly do struktury projektu. Vytvoření vkládacích příkazů, vytvořených ze získaných informací o kartách, vyžadovalo větší množství operací. Protože se parametry v tabulce nenacházely ve stejném sloupci, musel se najít vzor, jak sloupce seřadit. Naštěstí se struktura informace ve sloupcích skládala z názvu vlastnosti karty a její hodnoty spojené dvojtečkou. Po aplikování funkce, rozdělující text na dvě části podle dvojtečky, se do volného sloupce pomocí podmíněného příkazu dokázala nasměrovat požadovaná vlastnost. Po aplikaci funkce na všechny existující vlastnosti byla dosaženo integrity vlastností ve sloupcích. Ze série a čísla karty v sérii, které byly získány z jednoznačného identifikátoru karty, byl vytvořen odkaz na obrazovou podobu karty.

```
=IF(ISERROR(FIND("Lore"; AI3));IF(ISERROR(FIND("Lore"; AK3));IF(ISERROR(FIND("Lore";
AM3));IF(ISERROR(FIND("Lore"; AO3));IF(ISERROR(FIND("Lore"; AQ3));IF(ISERROR(FIND("Lore";
AS3));IF(ISERROR(FIND("Lore"; AU3));IF(ISERROR(FIND("Lore"; AW3));IF(ISERROR(FIND("Lore";
AY3));IF(ISERROR(FIND("Lore"; BA3));IF(ISERROR(FIND("Lore"; BC3));"null"; BD3); BB3); AZ3);
AX3); AV3); AT3); AR3); AP3); AN3); AL3); AJ3)
```

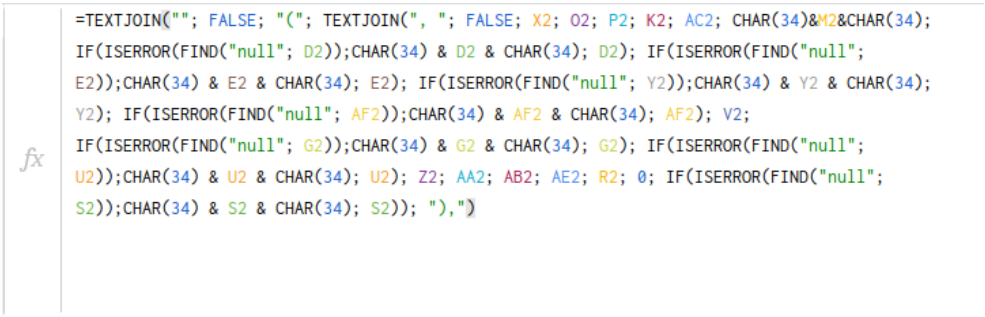
Obrázek 8.2: Funkce u zpracování dolovaných informací v tabulkovém softwaru

Bylo nutné změnit parametry karty obsažené v tabulce Cards jako cizí klíče cizí tabulky za dané numerické identifikátory. Z důvodu nízkého počtu různých hodnot v ostatních pomocných tabulkách byly informace do nich vkládány ručně. Příslušné textové hodnoty získané dolování z webového zdroje se pomocí funkce zaměnily za numerický ekvivalent.



Obrázek 8.3: Číselník v tabulkovém softwaru při zpracování dolovaných informací

Z kombinace hodnot ve sloupcích byl vytvořen vkládací skript reprezentující jednu z tří a půl tisíce karet. Popsaná metoda přípravy skriptů nebyla pravděpodobně nejlepší, ale výsledku bylo dosaženo. Bylo nutné ručně upravit některé informace, avšak chybovost byla v poměru k celkovému počtu řádků tabulky zanedbatelná.



Obrázek 8.4: Funkce pro vytvoření finálního vkládacího skriptu

Kapitola 9

Integrování Elasticsearch indexu

Zvolená knihovna FOSElasticaBundle integruje Elasticsearch do frameworku Symfony. Díky svému přímočarému konfigurování umožňuje v krátkém čase nastavit index nahrazující dotazovaní z relační databáze.

9.1 Instalace a konfigurace

Po zahrnutí composerem mezi požadované knihovny a instalaci se vygeneroval konfigurační soubor v adresáři packages. V něm byly definovány čtyři vlastní indexy. Nejdůležitější a nejpoužívanější card vychází ze stejně pojmenované entity. V indexu jsou uchovány všechny parametry, které se používají pro filtrování karet.

V kontrolním panelu umístěný filtr karet vyhledává nejčastěji parametry uložené v číselníkových pomocných tabulkách. Pro uložení parametrů cizí entity mající s "domácí" entitou nějaký vztah se používá typ nested. U parametru označeného jako nested se dále definují jeho vlastní parametry. Takto vnořených struktur může být mnoho, dotazování se bude ale logicky zpomalovat s každou přidanou úrovní. Jelikož k filtraci stačí znát identifikátor dané entity, vytvoří se v entitě Card pomocný getter, který získá identifikátor z cizí entity. V případě absence entity se vrací null. V tomto konkrétním případě není nutné zbytečně používat syntaxi s nested typem. Použití typu keyword pro identifikátor vychází z faktu, že není nutné ani žádoucí porovnávat část hodnoty, vždy jen celek.

Pro vyhledávání na základě textu obsaženého na kartě bylo zapotřebí vytvořit vlastní analyzátory. Tyto nástroje slouží k předzpracování textu. Dle zvolených parametrů například odstraní z textu interpunkce, vše převedou na malá písmena, apod. Jelikož by se mělo vyhledávání textu spustit již po napsání dvou znaků, musí se slova z parametrů rozdělit na podřetězce pomocí filtru edge_ngram. Definuje se u něj nejmenší a největší velikost podřetězce. Funkci našeptávače by zastal i běžný typ text, ale toto řešení by selhalo na

hledání slova umístěného uprostřed věty. Hledání textu probíhá ve čtyřech různých parametrech, na které jsou aplikované rozdílné analyzátory. Mezi použité filtry patří redukce duplicit, odstranění slov bez přidaného významu, změna na malá písmena, rozdělení na podřetězce a transformace apostrof. Knihovna umožňuje použít rozdílný analyzér při tvorbě indexu a při hledání v reálném čase. Kdyby se měly filtry aplikovat až při hledání, trvalo by to mnohem více času.

```
settings:
    index:
        analysis:
            analyzer:
                code_analyzer:
                    type: custom
                    tokenizer: classic
                    filter: [ lowercase, code_ngram ]
                title_analyzer:
                    type: custom
                    tokenizer: classic
                    filter: [ lowercase, custom_ngram, cus_unique, apostrophe ]
                game_text_analyzer:
                    type: custom
                    tokenizer: classic
                    filter: [ stop, lowercase, custom_ngram, cus_unique, apostrophe ]
            filter:
                code_ngram:
                    type: edge_ngram
                    min_gram: 1
                    max_gram: 6
                custom_ngram:
                    type: edge_ngram
                    min_gram: 1
                    max_gram: 10
                cus_unique:
                    type: unique
                    only_on_same_position: true

properties:
    id: { type: keyword }
    title: { type: text, analyzer: title_analyzer, search_analyzer: standard }
```

Obrázek 9.1: Vlastní konfigurace analyzátoru u Elasticsearch indexu

Další index se stará o uchování informací o kartách uživatelů. Pro zachování možnosti filtrace uživatelských karet podle parametrů abstraktních předloh se v podstatě zkopírovaly parametry indexu cards jako nested parametr v indexu real_cards. Jelikož funkce na pozadí, starající se o aktualizaci

indexu, berou v potaz změny parametrů pouze na první úrovni, při změně v entitě Cards by mohla vzniknout nekonzistentnost. Variantou by bylo nejprve získat abstraktní karty vyhovující aplikovaným filtrům a poté omezit uživatelovy karty podle získané množiny. Výhodnějším řešením bude získat výsledek ihned po provedení dotazu za cenu duplikace dat. V tomto případě jde především o rychlost.

Další dva indexy se týkají položek na tržišti. U poptávky lze postupovat stejně jako u uživatelských karet. V případě nabídky by již hrozila nekonzistence, pokud by se uživatel rozhodl kartu z nabídky pozměnit. V indexu by byla stále viditelná v původní podobě, protože by se data v indexu automaticky neaktualizovala. Pomocí logiky poslouchající na změnu entity RealCard se uměle zavolá změna poptávky a tím se aktualizuje hodnota v indexu.

9.2 Tvorba dotazů

Pro práci s obsahem knihovny FOSElasticaBundle byla vyčleněna speciální služba nazvaná ElasticService. V konstruktoru bylo nutné specifikovat tzv. finder. Jedná se o určení nástroje, který se dotazuje na konkrétní index a převádí výsledek na databázové objekty. Konkrétní druh finderu se musí specifikovat v argumentu deklarace služby v souboru services.yaml. Požadavek na index je strukturován do víceúrovňových dotazů (queries). Pro určení priority se používá booleovská logika, kde výraz je must ekvivalentem AND a should se dá chápat jako OR. Každý požadavek má páteřní dotaz, na který se připojují konkrétní omezení dodaná z filtrů.

V controlleru zpracované filtrovací požadavky jsou předány službě ve formě asociovaného pole. Neobsahuje-li požadavek určitou konkrétní limitaci, není ani její název obsažen jako klíč asociativního pole. Postupně se zjišťuje přítomnost klíčů z filtru. Při kladné odezvě se vytvoří poddotaz a do něj se umístí příslušný počet vyhledávacích dotazů podle počtu specifikovaných hodnot filtrem. Jelikož je ve filtru použita knihovna Select2Entity, kdy se uživateli zobrazí název položky, návratová hodnota obsahuje identifikátor a při zpracování se vrátí celá entita, musí se z entity získat ještě její identifikátor pomocí jednoduchého getteru.

Situace je trochu rozdílná u fulltextového vyhledávání v některých parametrech. Aby se mohlo vyhledávat ve více parametrech naráz, tak se používá multi dotaz. Funkcionalita se liší podle specifikovaného typu. Text se může porovnávat s každým parametrem zvlášť a vybere se nejlepší dosažené skóre. V jiném případě se dosažená skóre sčítají. Nejvíc se ale osvědčil typ Cross fields, kde se virtuálně sloučí všechny parametry a porovnává se pouze jednou.

Výsledek je vrácen v pořadí podle určeného způsobu řazení. Kvůli velkému množství hodnot se nevyhnutelně muselo použít stránkování. Maximální počet položek na stránce může uživatel nastavit podle své preference.

```php
$rootQuery = new Query();

if(!empty($filters)){
    $filterQuery = new BoolQuery();
    if(array_key_exists( key: "textQuery", $filters)){
        $textQuery = new MultiMatch();
        $textQuery->setType( type: MultiMatch::TYPE_CROSS_FIELDS);
        $textQuery->setOperator( operator: MultiMatch::OPERATOR_AND);
        $textQuery->setQuery($filters['textQuery']);
        $textQuery->setFields(['title', 'subtitle', 'gameText', 'code']);
        $filterQuery->addMust($textQuery);
    }
    if(array_key_exists( key: "culture", $filters) and !empty($filters['culture'])){
        $cultureBool = new BoolQuery();
        foreach ($filters['culture'] as $culture){
            $cultureBool->addShould(new MatchQuery( field: 'cultureId', $culture->getId()));
        }
        $filterQuery->addMust($cultureBool);
    }
```

Obrázek 9.2: Tvorba dotazu pro Elasticsearch pomocí balíčku FOSElastica-Bundle

Kapitola 10

Komunikace pomocí WebSocket

Uživatelé musí mít možnost komunikovat v reálném čase napříč celou aplikací. Dostupné je globální vlákno, kde jsou zprávy viditelné všem ostatním, a oddělené konverzace. V případě zájmu o soukromou komunikaci vyhledá uživatel jméno příjemce a přidá si ho mezi své kontakty. Po kliknutí na konkrétní jméno uživatele se objeví společná historie komunikace a pole pro novou zprávu.

Knihovna cboden/ratchet slouží k integraci WebSocket technologie ve frameworku Symfony. Popis technologie a výhody oproti pollingu jsou k dispozici v teoretické části práce v kapitole WebSocket.

10.1 Instalace a konfigurace

Po instalaci knihovny se na rozdíl od knihovny FOSElasticaBundle nevytvoří žádné nové konfigurační soubory. Nejprve se musí vytvořit služba, kde se budou uchovávat aktuálně připojení uživatelé a zpracovávat příchozí zprávy. Za strukturu pro aktuálně existující spojení se použilo obyčejné asociativní pole s klíčem podle identifikátoru uživatele. V základní šabloně base je v případě aktivního uživatele vložena šablona pro určená pro komunikaci mezi uživateli. Při vývoji se použila nezabezpečená variant ws, v produkčním prostředí by bylo nutné použít zabezpečenou šifrovanou variantu wss. Server se spouští pomocí příkazu, který spustí nový server a předá mu službu pro správu připojení a zpráv. Kromě adresy, na které server poslouchá se specifikuje i použitý port. I když by mohl server poslouchat na specifické adrese, reálně se jedinou relevantní informaci považuje číslo portu.

Připojení je vytvořeno na adrese obsahující aktuálně připojeného uživatele. Identifikátor uživatele v adrese slouží při otevření spojení jako klíč pro asociativní pole. Logicky se nabízí získat aktuálně připojeného uživatele pomocí Security. Jelikož však musí metoda volaná při novém spojení obsahovat pouze parametry definované rozhraním, nelze uměle vložit Security jako vstupní pa-

rametr. Nepřipadá v úvahu ani předá parametru z příkazu, protože v té době nemusí být uživatel přihlášen. Cesta, jak získat uživatele jiným způsobem by byla příliš náročná.

```
protected function execute(InputInterface $input, OutputInterface $output)
{
    $server = IoServer::factory(
        new HttpServer(
            new WsServer(
                new MessageService($this->em, $this->userRepository, $this->messageFactory)
            )
        ),
        port: 8888
    );
    $output->writeln( messages: "Starting server on " . $server->socket->getAddress());
    $server->run();
    return 0;
}
```

Obrázek 10.1: Metoda na spuštění WebSocket serveru

10.2 Použití

Jakmile uživatel stiskne tlačítko pro poslání zprávy, socket na pozadí připraví zprávu a odešle ji na server. Ve funkci zaznamenávající příchozí zprávu se z obsahu zprávy identifikuje příjemce a odesílatele. Má-li příjemce speciální hodnotu global, znamená to, že se jedná o zprávu do globálního vlákna. V ostatních případech je příjemce i odesílatel rozpoznán podle identifikátoru. Existuje-li aktivní spojení příjemce, nastaví se status zprávy na přečtená. Zpráva se po uložení do databáze odešle na obě strany. Bez potvrzení ze serveru ani odesílatel v historii komunikace neuvidí svou zprávu. Jedná se o pojistku v případě chyby zpracování na serveru, aby nedošlo k nekonzistenci a nedorozumnění mezi příjemcem a odesílatelem zprávy. V případě opuštění stránky se z asociativního pole odebere spojení mezi serverem a uživatelem. Nastane-li chyba, spojení je násilně ukončeno ze strany serveru.

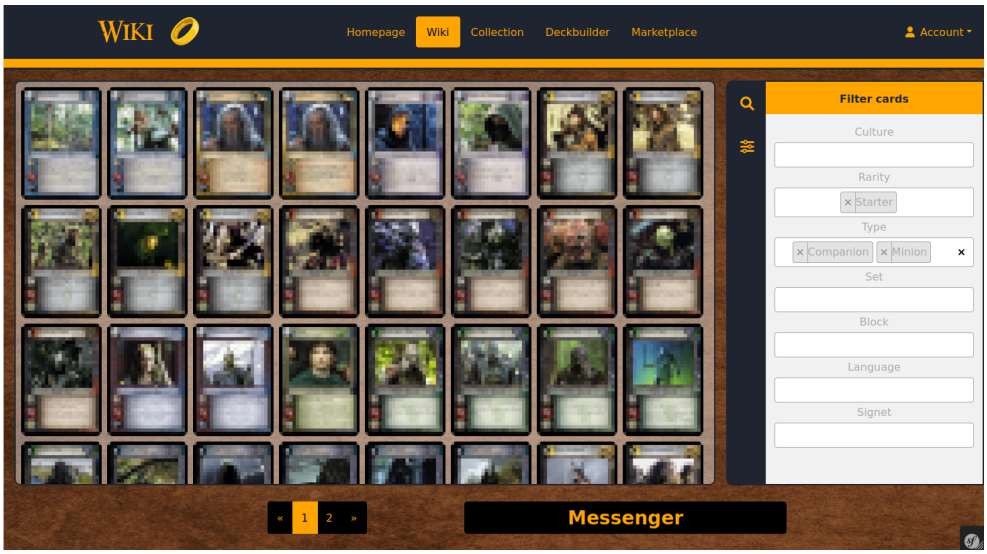
```
$message = $this->messageFactory->createNew();
$message->setAuthor($author);
$message->setContent($msgJson->content);

if($msgJson->receiverId == 'global'){
    foreach ($this->connections as $connection){
        $connection->send(json_encode($msgJson));
    }
    $message->setStatus( status: Message::MESSAGE_SEEN);
} elseif($receiver instanceof User) {
    $message->setReceiver($receiver);
    $from->send(json_encode($msgJson));
    if(array_key_exists($msgJson->receiverId, $this->connections)) {
        $this->connections[$msgJson->receiverId]->send(json_encode($msgJson));
        $message->setStatus( status: Message::MESSAGE_SEEN);
    }
} else {
    $this->onClose($from);
    return;
}
$this->em->persist($message);
$this->em->flush();
```

Obrázek 10.2: Zpracování zpráv z WebSocketu na serveru

Kapitola 11

Wiki a kolekce

Nejdůležitější část, na které přímo i nepřímo závisí zbytek systému, se stará o správu abstraktních a konkrétních uživatelských karet. Wiki představuje omezenější verzi, pro nepřihlášeného uživatele, kde je možné prohlížet a filtrovat seznam všech dostupných karet. Kolekce umožňuje provádět všechny operace dostupné ve wiki, avšak jako bonus přidává možnost přidání, editaci a odebrání karet z/do soukromé sbírky daného uživatele.



Obrázek 11.1: Zobrazení Wiki sekce s aplikovaným filtrem

11.1 Backend

Veškeré dění v controlleru, určeného pro dotazy z wiki části systému, se točí kolem funkce filterAction. Aby bylo možné karty na stránce filtrovat, vytváří se formulář s názvem CardQueryType. Parametrů jsou typu Select2EntityType, zpostředkované knihovnou Select2Entity. Nastavuje se například zobrazovaná a návratová hodnota, reprezentující třída a minimální délka vstupu. Velkou předností tohoto typu je možnost použití autodoplňování, protože uživatel nemusí znát všechny dostupné možnosti. Ze specifikované cesty pro autodoplňování přichází odpověď ve formátu JSON. Tato funkce využívá předpřipravenou službu v rámci knihovny Select2Entity, nutné je pouze definovat funkci v controlleru a třídě předat dotaz a druh formuláře. Jediná výjimka se vyskytuje v limitaci zobrazených typů karet. Typy jsou určené pomocí rozdělovacího sloupce, avšak postrádají vlastní tabulky. Entita Cards obsahuje výčet všech existujících typů ve formě pole. Toto pole je předáno do formuláře prostřednictvím nabídky pro výčtový typ.

Pokud se formulář vyhodnotí jako odeslaný, provede se zpracování položek do formy asociativního pole. V opačném případě zůstane pole prázdné. Po vyhodnocení formuláře se zavolá interní služba zajišťující komunikaci s Elasticsearch indexem. Do služby se předává pole s parametry filtrů, aktuální stránka a počet položek na stránce. Tím je dosaženo stránkování. Výsledky dotazu na Elasticsearch index jsou pomocí nástroje finder automaticky převedeny do objektové verze. Toto zobrazení již dokáže šablona vykreslit do podoby HTML prvků.

11.2 Frontend

Při přístupu uživatele na do wiki části portálu se zavolá výchozí funkce indexAction, jejíž jediným úkolem je vrátit vykreslenou základní šablonu. Tato základní šablona dědí z obecné šablony specifikující výchozí rozvržení aplikace a funkce platné v celé aplikaci včetně například WebSocket komunikace na straně klienta. Výchozí šablonovací nástroj Twig umožňuje kód shlukovat do bloků, které se v případě dědění mohou či nemusí lišit. Vytvořená víceúrovňová struktura výrazně redukuje velikost kódu při zachování plné funkčnosti.

Hlavička se skládá z nadpisu, navigačního seznamu, bloku pro zobrazení oznámení a uživatelského boxu. Hlavní blok main obsahuje sekci pro zobrazení obrázků či seznamu karet a na straně umístěný vysouvací kontrolní panel. V kontrolním panelu je umístěn filtr na limitování zobrazených karet, možnosti pro zobrazení a v případě kolekci i bloky pro přidání a editace karet do/z kolekce. Přepínání režimů mezi seznamem a obrázky karet funguje na principu změny viditelnosti. Z controlleru jsou k dispozici obě varianty naráz. Při změně tedy nedochází ke zpomalení čekáním na AJAX dotaz.

```
->add( child: 'type',  type: ChoiceType::class, [
    'choice_loader' => new CallbackChoiceLoader(function(){
        return Card::TYPES;
    }),
    'multiple' => true,
    'expanded' => false
])
->add( child: 'set',  type: Select2EntityType::class, [
    'multiple' => true,
    'remote_route' => 'wiki_query_autocomplete',
    'class' => CardSet::class,
    'placeholder' => 'Vybrat sérii',
    'delay' => 250,
    'minimum_input_length' => 0,
    'language' => 'en',
    'primary_key' => 'id',
    'text_property' => 'name',
    'property' => 'name'
])
```

Obrázek 11.2: Filtrovací formulář ve wiki sekci

Na principu změny viditelnosti fungují i okna v kontrolním panelu. Ten je v defaultním případě v zúženém režimu a po kliknutí se zvětší do definované velikosti pomocí přidání třídy. Po změně v některém z filtrovacích nastavení se odešle AJAX požadavek na pozadí, který obsah filtračního formuláře pozmění na textový řetězec a odešle jako data POST dotazu na server. Aktualizovaná data jsou vrácena vykreslená na šabloně obsahující pouze vnitřní strukturu sekce. Nedochází tedy k aktualizaci celé stránky. Pod hlavním blokem umožňuje změnu stránky paginační lišta. Vedle ní se nachází blok pro komunikaci s ostatními uživateli rovněž schovaný ve výchozí poloze.

```twig
{% extends 'base.html.twig' %}

{% block title %}Wiki{% endblock %}

{% block headerTitle %}
    <h1>Wiki</h1>
{% endblock %}

{% block navItems %}
    <li class="nav-item"><a href="{{ path('index') }}" class="nav-link">Homepage</a></li>
    <li class="nav-item"><a class="nav-link active">Wiki</a></li>
    <li class="nav-item"><a href="{{ path('collection_index') }}" class="nav-link">Collection</a></li>
    <li class="nav-item"><a href="{{ path('deckbuilder_index') }}" class="nav-link">Deckbuilder</a></li>
    <li class="nav-item"><a href="{{ path('marketplace_index') }}" class="nav-link">Marketplace</a></li>
{% endblock %}

{% block body %}
    <section class="wiki-display">{{ render_esi(controller('App\\Controller\\WikiController::filterAction')) }}</section>
{% endblock %}

{% block filterHeader %}
```

Obrázek 11.3: Twig šablona s použitými bloky na indexu Wiki sekce

Kapitola 12

Tvorba balíčků

Že je okamžité stavění herních balíčků z reálných karet nepraktické vám potvrdí každý hráč. Myšlenkové pochody se mohou v průběhu tvorby měnit a fyzická manipulace s kartami zdržuje. Schopnost rychle vložit karty do balíčku a v případě mylného rozhodnutí je stejně rychle odstranit patří mezi klíčové vlastnosti, které by měl nástroj pro tvorbu balíčků splňovat.

12.1 Správa balíčku

Při vstupu do části systému věnované tvorbě balíčků je uživateli zobrazen seznam jeho již existujících balíčků. Na výběr má vytvoření nového, editaci nebo smazání existujícího balíčku. Ke každému balíčku je uveden seznam herních formátů, ve kterých lze balíček použít. Tato informace se ukládá do databáze při každém uložení balíčku. Rozhraní pro vytvoření i editaci neobsahuje žádné viditelné rozdíly. Hlavní část stránky obsahuje několik sekcí, které stejně jako sekce v kolekci pracují na získání viditelnosti. V pravé části je umístěn kontrolní panel, kde uživatel nalezne opět správu zobrazení, validační nástroj a filtraci karet podle zvoleného režimu zobrazení. V levé části je umístěn seznam reprezentující aktuální stav balíčku. Do databáze se stav uloží až po kliknutí na tlačítko uložit. Uložení po každé změně v balíčku by bylo lehce realizovatelné, avšak nešlo by zkoušet změnit již existující balíček, protože by původní verze nebyla dohledatelná.

Vkládat karty do balíčku lze několika způsoby. Vlastní-li hráč karty, které chce přidat, může si je s pomocí filtrů nalézt ve vlastní kolekci a přidat je do balíčku kliknutím na akční panel umístěný na obrázku karty, případně stisknutím tlačítka v seznamu karet. Pokud hráč požadovanou kartu nevlastní, může ji vyfiltrovat ve wiki sekci a stejným postupem umístit do balíčku. Na pozadí to funguje tak, že se do skrytého seznamu přidá identifikátor karty a po uložení balíčku se vytvoří speciální verze reálné karty lehce rozeznatelná od té běžné na základě parametru. Třetí možností je použití doporučovacího

nástroje. Více se o něm můžete dozvědět v jiné podkapitole.

```
        {% for realCard in group %}
            {% if loop.first == true %}
                <li data-card="{{ realCard.card.id }}"
                    data-realcard="{{ realCardArray|join(',') }}"
                    data-fakecard="{{ fakeCardArray|join(',') }}">
                    <span>{{ realCard.card.code }}</span>
                    <span>
                        {{ realCard.card.title }}
                        {% if realCard.card.subtitle is not null %},
                            {{ realCard.card.subtitle|length > 10 ? realCard.card.subtitle|slice(0,5) ~
                        {% endif %}
                    </span>
                    <span><span>{{ loop.length }}</span>x</span>
                </li>
            {% endif %}
        {% endfor %}
    {% endfor %}
    {{ form_widget(form.cards) }}
</ul>
<ul id="abstractCards-fields-list" class="d-none"
    data-prototype="{{ form_widget(form.abstractCards.vars.prototype)|e }}"
    data-widget-tags="{{ '<li></li>'|e }}"
    data-widget-counter="{{ form.abstractCards|length }}">
    {% for abstractCardsField in form.abstractCards %}
        <li>
            {{ form_errors(abstractCardsField) }}
            {{ form_widget(abstractCardsField) }}
        </li>
    {% endfor %}
</ul>
```

Obrázek 12.1: Twig šablona zachycující zobrazení uloženého balíčku s připraveným skrytým sloupcem pro nové karty

Odebrání karty z balíčku umožňuje několik způsobů. Po kliknutí na řádek s danou kartou se sníží kvantita dané abstraktní karty v balíčku. Pokud se klikne v akčním panelu na symbol pro odebrání, odebere se karta podle toho, zda se jedná o abstraktní nebo kartu z kolekce. Karty se odebírají ve specifickém pořadí. Jako první se ze seznamu odeberou abstraktní karty, které se na začátku editace v balíčku nenacházely - nejsou momentálně uloženy v databázi. Jako druhé se odeberou abstraktní karty, které mají svůj záznam v tabulce. Rozeznat se dají podle HTML data parametru umístěného v řádku karty v balíčku. Až při absenci abstraktních karet se odeberou karty obsažené v kolekci. Toto pořadí lze ručně změnit odebráním karty cestou přes akční panel na kartě.

12.2 Validace balíčku

Balíček musí dodržovat jistá pravidla, aby mohl být použit v souboji proti jiným hráčům. Mezi základní pravidla patří dodržení rovnosti hodných a zlých karet v balíčku. V balíčku nesmí chybět určité herní postavy, herní mapa se musí skládat přesně ze sedmi zemí a všechny použité země musí mít jiné číslo. Aby hráč na žádné z uvedených pravidel nezapomněl, zobrazuje se mu v kontrolním panelu infopanel o validitě balíčku. Při každém vložení a odebrání karty z balíčku (bez nutnosti uložení) se na pozadí pomocí AJAX požadavku zkontroluje validita balíčku. Ze seznamu karet se podle parametru data-card získají konkrétní druhy karet a z přilehlého textového řetězce i kvantifikátor pro danou kartu. Server po přijetí AJAX požadavku prověří všechny možné situace podle dodaných dat. Všechny indikátory pro nevyhovující typy dat ve výchozím stavu počítají se špatným scénářem a až teprve po důkazu validity se nastaví na správnou hodnotu.

```php
$validationMessages = [];
$validationCriterias = [];
$validationCriterias['ringbearerFlag'] = [true, "Chybí Ring-bearer"];
$validationCriterias['oneRingFlag'] = [true, "Chybí One Ring"];
$validationCriterias['moreThanOneRing'] = [false, "Balíček má víc než jeden One Ring"];
$validationCriterias['siteCounterFlag'] = [true, "Balíček nemá přesně 9 zemí"];
$validationCriterias['siteBadFlag'] = [false, "Špatné složení zemí"];
$validationCriterias['differentNumberFlag'] = [false, "Rozdílný počet Free people a Shadow karet"];

$freeCounter = 0;
$shadowCounter = 0;

$sites = [];

foreach ($cards as $card) {
    if ($card instanceof SiteType) {
        $sites[] = $card;
    } elseif ($card instanceof RingType) {
        if ($validationCriterias['oneRingFlag'][0]) {
            $validationCriterias['oneRingFlag'][0] = false;
        } else {
            $validationCriterias['moreThanOneRing'][0] = true;
        }
    } else {
        if ($card instanceof CompanionType and $validationCriterias['ringbearerFlag'][0] == true
            and str_contains(strtolower($card->getGameText()), 'ring-bearer')) {
            $validationCriterias['ringbearerFlag'][0] = false;
```

Obrázek 12.2: Proces validace balíčku s definovanými indikátory

Součástí validátoru je i zkouška, pro jaké formáty je balíček vhodný. Herní formát v základní formě limituje série, ve které karta vyšla. Obtížnější omezení se skrývá v jednotlivých zakázaných kartách. Každý blok obsahuje informaci o tom, které karty se v něm nesmí použít. V cyklu se zkontroluje počet a druh

karet. Postupně se vylučují formáty podle sérií a druhů karet, až zůstanou jen ty vhodné. Informace je zobrazena spolu s chybami ve validaci v kontrolním panelu.

12.3 Doporučovací nástroj

Zejména noví hráči ocení pomocný nástroj pro stavbu balíčků. Funguje na principu doporučovacího systému, kdy se ze vstupních dat získají hodnoty pro skórovací funkci. Funkce přiřadí každému potenciálnímu výsledku skóre, podle kterého je celkový výsledek seřazen. V případě stavby balíčku se jedná o nápovědu tří karet podle parametrů použitých karet.

Při zpracování dat ke kartám se identifikovaly herní mechaniky využívající danou kartu. Seznam všech existujících vlastností a mechanik, které karty mohou obsahovat, se nachází v entitě CardKeyword. Každá karta může mít několik různých vlastností a mechanik. Ze vstupních dat jsou tyto hodnoty extrahované a jejich výskyt je vydělen celkovým počtem všech hodnot. Protože se ve hře vyplatí mít v balíčku pouze malý počet různých kultur, vzniklá hodnota se zkombinuje s ohodnocením pro kulturu podle zastoupení v balíčku. Roli hraje i zastoupení podle sérií karet. Karty se podle výsledného skóre seřadí a první tři výsledky se vrátí v odpovědi. Pokud se uživateli karty nebudou líbit, může požádat o následující tři karty výsledku. Hodnota se samozřejmě nepočítá znovu, je uložena v dočasné paměti pro pozdější využití.

```
    $cardSet = $c->getCardSet();
    if($cardSet instanceof CardSet and array_key_exists($cardSet->getId(), $cardSets)){
        $cardSetScore = 1 * $cardSets[$cardSet->getId()]['score'];
    }

    $score = $cultureScore + $keywordScore + $cardSetScore;
    $scoringCards[] = ['score' => $score, 'card' => $c];
}

usort( &array: $scoringCards, function($a, $b) {
    return $a['score'] < $b['score'];
});
```

Obrázek 12.3: Vypočítání finálního skóre v doporučovacím nástroji a následné seřazení podle relevance

Kapitola 13

Tržiště

Z důvodu konce distribuce nových balených produktů se již dají sehnat karty pouze neoficiální cestou. Jedná se o mezinárodní fóra na sociálních sítích, komunitami organizované turnaje nebo aukční portály. Zabudovaná funkcionalita na výměnu karet v části systému Marketplace zjednodušuje nakládání s kartami hned z několik důvodů. Bez použití výměny přes aplikaci by komunikace s druhou stranou pravděpodobně probíhala na jiném médiu. Po dokončení transakce, ať už finanční či karetní, by musel uživatel ručně nahrát karty do systému pomocí návodu v kolekci. Mezi hlavní výhody tržiště patří automatický přesun karet po dokončeném obchodě. Z dokončených obchodů se navíc získává průměrná cena karty, která pomůže novým uživatelům s cenovou orientací.

13.1 Popis a zobrazení tržiště

Tržiště se podobně jako zbytek systému skládá z několika sekcí pracujících s viditelností. Vpravo umístěný kontrolní panel umožňuje přepínat režimy zobrazení podle požadované funkcionality. Pokud uživatele zajímají nové karty do sbírky, vybere v kontrolním panelu v oddělení zobrazení režim Nákup. Pokud ho zajímá poptávka po kartách ostatních, zvolí opačný režim Prodej. Tyto dva režimy lze dodatečně rozlišit podle toho, jestli chce uživatel vidět nabídky a poptávky ostatních, či jeho položky umístěné na tržišti. Z kontrolního panelu se rovněž přidávají a upravují existující karty na tržišti. Nutné je zvolit konkrétní kartu pomocí stisknutí příslušného pole v akčním panelu umístěném na kartě.

Vlastní-li uživatel karty z poptávky v požadované kvalitě, na akčním panelu vybere danou kartu z kolekce a přidá kartu do obchodu na svou stranu výměny. Rozhodne-li se uživatel přidat cizí karty do nového obchodu, stačí stisknout pole v akčním panelu na kartě podobně jako u vkládání a editaci vlastních karet. Daná karta se ihned přidá do obchodu mezi uživatelem a maji-

telem karty. V jednom obchodu může být mnoho karet, musí být samozřejmě pouze od jednoho jiného uživatele. Pokusí-li se uživatel přidat kartu od jiného uživatele, vytvoří se další obchod. V současné chvíli nelze mít současně dva nedokončené a nezamítnuté obchody s jedním uživatelem. Na přehled obchodu se uživatel přesměruje pomocí odkazu v navigační liště, která se nachází v hlavičce.

13.2 Průběh obchodu

Obchody jsou v přehledu rozdělené podle stavu, ve kterém se nacházejí. Výchozí stav si lze představit jako košík v internetovém obchodě. Uživatel vybírá karty na obě strany obchodu. Každá karta může mít své finanční ohodnocení. Při vkládání do návrhu obchodu se přepočítává finanční rozdíl, který by měla jedna ze stran zaplatit té druhé. Částku lze ručně změnit oběma uživateli.

Po rozhodnutí o finalizaci návrhu se po stisknutí tlačítka Odeslat ke schválení předá návrh druhé straně. V případě odlišné představy protistrany může dojít k protinávrhu. Druhá strana může rovněž návrh odmítnout. Momentálně se neřeší spamming obchodníka s opakovaně zamítavými návrhy. Pokud je návrh akceptován, karty jsou automaticky převedeny mezi uživateli. Předpokládá se, že komunikace mezi oběma stranami bude probíhat přes komunikační nástroj dostupný všem přihlášeným uživatelům.

Kapitola 14

Testy

Pro testování ve frameworku Symfony se nejčastěji používá balíček test-pack integrující obecnou PHP knihovnu PHPUnit. Po instalaci se vytvoří adresář tests. V rámci testování byly používány dva typy testů. Prvním typem jsou Unit testy, které se zaměřují na malou část kódu. Druhým typem jsou složitější integrační testy, které už často vyžadují součinnost více částí aplikace.

14.1 Unit Testy

Unit testy jsou rozeznatelné podle dědičnosti z abstraktní třídy TestCase. Jedná se o nejjednodušší testy zaměření na testování algoritmů a jiných funkcionalit. Nelze jednoduše využívat ostatní služby v aplikaci. Pomocí assertů se kontroluje, zda testovaná část kódu vrací správné výsledky.

V aplikaci se Unit testy využily na kontrolu validátoru herních balíčků. Testovala se obecná funkčnost jednotlivých validačních identifikátorů a následně i aplikace na specifický scénář. Další testovanou částí kódu bylo filtrování herních formátů podle karet obsazených v balíčku. Formáty se omezují podle série, ve kterých jednotlivé karty vycházely. Protože některé karty dosahovaly enormního úspěchu a nebyly vybalancované, jednotlivé formáty mohou úplně zakázat nebo omezit počet kopií dané karty v balíčku.

Po vytvoření formátů a karet následovalo postupné přidávání omezujících kritérií a série assertů. Protože složitější algoritmy se nachází hlavně v části tvorby herního balíčku a tržiště, netestovala se část se správou karet. Z tržiště se testovalo vytvoření protinabídky či správné zobrazení popisku.

14.2 Integrační testy

Hlavním rozdílem mezi Integračními a Unit testy je dostupnost kontejneru a díky tomu možnost volání služeb napříč aplikací. Každá testovací funkce startuje s novou instancí kernelu, aby nedocházelo k ovlivnění dat z minulými

```
$inArray = $this->createSpecificDeck();

$cards = new ArrayCollection($inArray[0]);
$blocks = new ArrayCollection($inArray[1]);

$suitableBlocks = $this->getSuitableBlocks($cards, $blocks);
$this->assertEquals( expected: 2, sizeof($suitableBlocks));
$this->assertEquals($blocks[0], $suitableBlocks[0]);
$this->assertEquals($blocks[1], $suitableBlocks[1]);

$limited2 = new LimitedCardForBlock();
$limited2->setIsForbidden( isForbidden: false);
$cards[1]->addLimitedBlock($limited2);
$blocks[1]->addLimitedCard($limited2);

$suitableBlocks = $this->getSuitableBlocks($cards, $blocks);
$this->assertEquals( expected: 2, sizeof($suitableBlocks));
$this->assertEquals($blocks[0], $suitableBlocks[0]);
$this->assertEquals($blocks[1], $suitableBlocks[1]);
```

Obrázek 14.1: Testování funkcionality uvnitř třídy za pomoci Unit testu

testy. Integrační test musí dědit ze třídy KernelTestCase, aby mu byl kernel a kontejner zpřístupněn.

Integrační test byl použit například na ověření, zda se správně vytvoří kopie karet při úspěšně uzavřeném obchodu na tržišti. Z kontejneru se získá továrna na vytvoření kopie karty na základě dodaného nového majitele a vzorové karty. Originální karta se následně nastaví na skrytou. Na konec se provede kontrola, zda se obě karty shodují. Dalším testem je kontrola náhodně vygenerovaného řetězce. Z kontejneru se dvakrát zavolá generující služba a výsledky se porovnají. Teoreticky by mohlo dojít ke shodě při pravděpodobnosti 26^15 kvůli délce řetězce.

Kapitola 15

Cachování a náhledové obrázky

Hlavním cílem aplikace politiky dočasné paměti je bezesporu dosažení co nejmenšího objemu dat, které se ke klientovi ze serveru musí během komunikace přenést. Framework Symfony ve svém výchozím nastavení podstatnou část řeší za uživatele. Avšak, pokud uživatel vyžaduje dodatečné modifikace, dalo mnoho úsilí přivést framework k poslušnosti.

15.1 Statické stránky

Systém obsahuje několik statických stránek. Jedná se například o stránku s často kladenými otázkami nebo základní informace o systému. Ze dvou zmíněných technik pro nastavení politiky pro práci s dočasnou pamětí se zde hodí ta expirující. Nepředpokládá se, že se bude měnit jejich obsah v řádu dní. Jelikož je jejich obsah pro všechny uživatele totožný, může se využít veřejné nastavení pro dočasnou paměť. Za vhodný časový údaj pro expiraci u obrázků, stylů, apod. může být považován i časový interval v řádu několika měsíců i rok.

V případě změny se jednoduše nahraje nová verze, která klienta přinutí si stáhnout aktualizovaná data. U samotné stránky takový časový údaj nastavit nelze, protože změna názvu šablony už není tak praktická jako u stylů, kde je tato funkce často automatizovaná. Jako ideální se ukazuje nastavená doba v řádu několika hodin.

```
$response = $this->render( view: 'static/faq.html.twig', []);
$response->headers->set( key: AbstractSessionListener::NO_AUTO_CACHE_CONTROL_HEADER, values: 'true');
$response->setPublic();
$response->setMaxAge( value: 1440);
$response->headers->addCacheControlDirective('must-revalidate');
return $response;
```

Obrázek 15.1: Expirační přístup u dočasné paměti

15.2 Wiki a kolekce

V části správy kolekce a procházení existujících karet uživatel načítá velké množství obrázků. Použitá knihovna LiipImagineBundle umožňuje pomocí filtrů snížit velikost obrázku či vytvořit náhledové obrázky. Vytvoření takto upravených kopií může probíhat na pozadí z příkazové řádky, nebo až za reálného běhu systému, jakmile o dotyčný obrázek požádá klient. Do specifikovaného adresáře se tyto kopie postupně ukládají.

Protože se stránky skládají z často měnící se části se zobrazením obrázků a statické části s navigací a filtrem, použila se technika ESI, která umožní rozdělit stránku na jednotlivé fragmenty s odlišnými politikami pro dočasnou paměť. Zatímco statická část může mít expirační obecné nastavení, tak v dynamické části je obsah personalizovaný. Zda se musí přenést nová data závisí na změně v bloku. Obsah se pomocí algoritmu převede na E-Tag identifikátor, který se následně porovnává s identifikátorem předaným z požadavku klienta. V případě shody může klient použít uložená data, jinak se mu pošle aktualizovaná verze. Pro přepsání výchozího Symfony nastavení musí odpověď serveru obsahovat hlavičku signalizující vlastní nastavení politiky dočasné paměti.

I když se při použití Elasticsearch indexu dosahuje rychlejších časů vyhledávání než u databázového řešení, v případě dotaz se musí provést, i když již byl předtím vykonán. Proto bylo dotazování na index obohaceno o interní dočasnou paměť vhodnou pro mezivýsledky, apod. Aby nemohlo dojít k záměně dvou uložených hodnot, musí být klíč jedinečný. V případě wiki části se klíč skládá ze zakódované informace o filtračním parametrech, konkrétní stránce, počtu položek na stránce a indikátoru případného použití pro stavbu herního balíčku. U zakódování entit bylo nutné si dát pozor na to, aby byl objekt správně převeden na textový řetězec. Může se v entitě specifikovat obecně známá metoda toString nebo pro tento případ více vyhovující metoda pro JSON serializaci. Pro dosažení unikátní hodnoty v serializaci postačil identifikátor. Interní dočasná paměť uchová vypočítanou hodnotu, v tomto případě objekt se stránkováním. Kvůli nevyhovující serializaci stránkovacího objektu se uvnitř funkce vykreslila konečná šablona, protože nenásledovala žádná další logika. Položce v paměti se přiřadila expirační doba a v případě kolekce i štítek.

```
$response = new Response($cards);
$response->headers->set( key: AbstractSessionListener::NO_AUTO_CACHE_CONTROL_HEADER, values: 'true');
$response->setEtag(md5($response->getContent()));
$response->setPrivate();
$response->headers->addCacheControlDirective('no-cache');
$response->isNotModified($request);
return $response;
```

Obrázek 15.2: Validační přístup u dočasné paměti

Štítkem se může uvolnit více paměťových záznamů naráz. Štítek v kolekci byl zvolen jako kombinace statického řetězce a uživatelského identifikátoru. Klíč pro paměť totiž nemůže obsahovat informace o vnitřní struktuře, pouze o limitujících parametrech. Změní-li se některá z položek v hráčské kolekci, hodnota v dočasné paměti by do doby expirace vracela původní hodnotu. Jakmile se tyto položky v paměti označí štítkem, může se u každé operace s uživatelskými kartami (vytvoření, změna, smazání) uvolnit větší skupina položek naráz.

15.3 Tvorba herních balíčků a tržiště

Největší uplatnění dostala interní dočasná paměť v této sekci. Vypočítání hodnoty pro skórovací funkci v doporučovacím systému patří mezi náročnější operace. Jakmile se uživateli nelíbí první tři záznamy s nejvyšším skóre, celý algoritmus by se musel spustit znovu pro další výsledky. Jako klíč pro identifikaci byla použita zakódovaná hodnota serializovaných karet z balíčku. Odpovědi serveru se následně ukládají s validační politikou, kdy může být vrácená hodnota použita i pro ostatní uživatele, protože výsledek není personalizovaný, ale založený na obecných kartách v balíčku. Jedna z direktiv dočasné paměti určuje způsob nakládání s uloženou hodnotou. V tomto případě se nastavil typ no-cache. Znamená to, že identifikátor E-Tag musí být vždy porovnán, aby vždy bylo dosaženo aktuality výsledku. Podobné řešení se použilo i u validace herního balíčku. Zde se ukládaly jak hodnoty validačních identifikátorů na základě karet v balíčku, tak vhodné herní formáty s posouzením karetních sérií a zakázaných karet.

Interní dočasná paměť byla v sekci tržiště použita u načítání poptávek a nabídek uživatelů. Podobně jako v případě kolekce se využily identifikační štítky, aby v případě změny v datech bylo možné okamžitě zneplatnit hodnoty v dočasné paměti. Protože velká část metod v controlleru vrací pouze JSON odpověď typu úspěch/neúspěch, nebylo tolik případů, kde dočasnou paměť použít. Jakékoliv filtrační formuláře napříč systémem byly nastaveny expiračním typem politiky s veřejným nastavením, protože se všem uživatelům zobrazují stejně.

```php
$key = 'md' . md5(json_encode($filters, flags: JSON_FORCE_OBJECT, depth: 5))
    . ';' . md5(json_encode($user, flags: JSON_FORCE_OBJECT, depth: 5)) . ';' . $page . ';' . $perPage . ';' . $userFlag;
$cards = $cache->get($key, function(ItemInterface $item) use ($filters, $user, $page, $perPage, $userFlag) {
    $cards = $this->elasticService->findMarketplaceDemandCards($filters, $user, $page, $perPage);
    $twigOptions = ['demands' => $cards, 'userFlag' => $userFlag];
    if(!array_key_exists( key: 'userFlag', $filters)) {
        $cardsForDemand = [];
        /** @var MarketplaceDemand $demand */
        foreach ($cards->getCurrentPageResults() as $demand) {
            $ownSupply = $user->getMarketplaceSuppliesByCard($demand->getCard());
            if(sizeof($ownSupply) > 0){
                $cardsForDemand[$demand->getId()] = $ownSupply;
            }
        }
        $twigOptions['cardsForDemand'] = $cardsForDemand;
    }
    $result = $this->renderView( view: 'marketplace/demands.html.twig', $twigOptions);
    $item->expiresAfter( time: 600);
    $item->tag(['demands', 'supplies']);
    return $result;
});
```

Obrázek 15.3: Interní dočasná paměť pro uložení mezivýsledků

Kapitola 16

Měření a porovnání

Pro dosažení některých z vytyčených cílů bylo potřeba měřit směrodatné veličiny. U dočasné paměti se jedná o načtení stránky, kdy se měří přenesená velikost dat a čas. Měření probíhalo v několika pokusech a do tabulky byla zanesena průměrná hodnota. U porovnání rychlosti načítání dat se měřil čas od zadání požadavku po získání dat na serveru. Původně se měl na měření času použít Stopwatch Bundle, ovšem po několika testech se zjistilo, že nejmenší měřitelný čas je desetina milisekundy. Přesnějším nástrojem na měření je např. volně dostupná knihovna hrtime. Výhoda Stopwatch Bundle by spočívala v současném měření času a použitých prostředků. Některé z cílů vyžadující obecně fungující systém nelze zaznamenat hodnotou, pouze konstatovat funkčnost.

16.1 Dočasná paměť

Jaké úspory času a zdrojů dosahuje vlastní konfigurace politiky dočasné paměti se testovalo pomocí dvou veličin ve třech různých testech. V prvním testu se načetla stránka s vymazanou interní pamětí, aby se ověřilo, kolik času navíc si aplikace interní dočasné paměti vyžádá. V druhém testu se opakovaně přistupovalo na stejnou stránku a během třetího se procházelo mezi stránkami výsledku. První měření mělo odhalit, zda se vyplatí na často opakující se místa v kódu umístit interní dočasnou paměť. Po prvním načtení stránky po smazání paměti si interní paměť pro sebe vyžádala o cca 10-15 ms více, ovšem v dalších dvou testech byly hodnoty více než dvojnásobně menší oproti klasické variantě.

Druhé měření porovnávalo hodnoty výchozí Symfony politiky s dočasnou pamětí na úrovni odpovědí serveru klientovi (HTTP caching). Vlastní nastavení nepřineslo žádné snížení přeneseného objemu dat s výjimkou posledního testu. Při procházení již načtených stránek nebyl poslán ze serveru jediný KB nových informací, vše pocházelo z dočasné paměti.

Testování aplikační dočasné paměti			
Použití dočasné paměti	První načtení stránky	Opakovaný přístup na stejnou stránku	Opakovaný přístup u stránkování
Bez aplikační paměti	560 ms	44 ms	62 ms
S aplikační pamětí	572 ms	19 ms	25 ms

Tabulka 16.1: Porovnání průměrných dosažených časů při použití aplikační dočasné paměti a bez ní

Testování vlastní dočasné paměti na úrovni klient-server			
Průměrná přenesená velikost dat	První načtení stránky	Opakovaný přístup na stejnou stránku	Opakovaný přístup u stránkování
Symfony výchozí řešení	6,47 MB	184 KB	110 KB
Vlastní cache politika	6,47 MB	184 KB	0 KB

Tabulka 16.2: Porovnání průměrné přenesené velikosti dat při výchozím a vlastním nastavení dočasné paměti

16.2 Porovnání databázového a Elasticsearch řešení

Často používané části byly z databázového řešení předělány na využití Elasticsearch indexu. Jak velké časové výhody se ovšem dosáhlo? Měření probíhalo na serveru, kdy se měřil interval od dotazování po vrácení výsledku. Výsledné rozdíly se ukázaly jako opravdu markantní. Zatímco databázové řešení při filtrování karet dosahovalo hodnot v řádu desítek milisekund, dotaz přes Elasticsearch index se ve většině případů dostal pod desetinu milisekundy. Výsledky se ve všech případech rovnaly. Za zmínku stojí fakt, že jakmile bylo u parametru více hodnot než jedna, výsledný čas se kvůli disjunkci mírně zvýšil. Roli ovšem hrálo i to, kolik položek splňovalo danou hodnotu kritéria.

Porovnání databázového a Elasticsearch řešení			
Popis filtru	Databázový dotaz [ms]	Elaticsearch dotaz [ms]	Stejné výsledky
Kultura Dwarven + text Elf	53.136	0.081	Ano
Typ Follower/Artifact + text Theoden	51.298	0.101	Ano
Jazyk Tengwar	48.749	0.065	Ano
Rarita Vzácná + set 2. + text Maneuver	72.058	0.069	Ano
Rarita Starter + set 1. + jazyk Angličtina	125.725	0.06	Ano
Kultura Gondor + vzácnost Rare + set 2.	78.397	0.081	Ano
Text Burden + kultura Gollum + rarita Rare + jazyk Tengwar	79.539	0.090	Ano
Text Arwen + kultura Elven + rarita Rare + pečeť Frodo	76.775	0.097	Ano
Rarita Rare/Common/Uncommon + type Condition/Minion/Ally-/Condition	94.935	0.090	Ano
Kultura Gandalf/Elven + typ Artifact	55.705	0.095	Ano

Tabulka 16.3: Porovnání databázového a Elasticsearch řešení u filtrování ve wiki sekci

16.3 Doporučovací nástroj

Relevanci doporučovacího nástroje lze ověřit porovnáním vypočítaného skóre u jednotlivých karet s obsahem balíčku. V několika scénářích se ověřilo, zda se skóre vypočítává správně. Ve všech scénářích bylo dosaženo příznivých výsledků.

16.3.1 První scénář - lukostřelci

V balíčku se momentálně nachází jedenáct karet, kde převládá jedna kultura. Tématem balíčku je jednoznačně lukostřelba, protože ji využívá každá karta. Výsledek jasně ovládla dominantní kultura z balíčku. Všechny tři nejlepší výsledky obsahují nejdůležitější klíčová slova podle skóre.

Tabulka hodnot prvního scénáře doporučovacího nástroje		
Kultura	Počet výskytů	Skóre
Elven	10	0.909
Gondor	1	0.091
Série karty	Počet výskytů	Skóre
1	5	0.455
5	4	0.364
3, 10	1	0.091
Klíčové slovo	Počet výskytů	Skóre
Archer	11	0.208
Archery	10	0.189
Fellowship, Fellowship archery total	9	0.17
Elf	2	0.038
Home, Exert, Tale, Discard, Skirmish, Strength +	1	0.0189

Tabulka 16.4: Nasbírané hodnoty z balíčku v prvním scénáři měření relevance doporučovacího nástroje

Tabulka hodnot u karet s nejlepším skore u prvního scénáře			
Kultura	Série	Klíčová slova	Skóre
Elven	1	Archer, Elf, Archery, Fellowship, Exert, Skip the archery phase, Fellowship archery total	3.178
Elven	1	Archer, Archery, Fellowship, Tale, Exert, Fellowship archery total	3.16
Elven	5	Archer, Archery, Fellowship, Exert, Fellowship archery total	3.05

Tabulka 16.5: První tři karty podle vypočítaného skóre v prvním scénáři u měření relevance doporučovacího nástroje

16.3.2 Druhý scénář - rozdílné kultury, stejná série, společná mechanika

Sérii ve druhém scénáři sdílely všechny karty z balíčku. Absenci karetní ekvivalence podle kultury způsobilo pravidlo pro různé hodnoty v balíčku. Na základě zvolení stejné série bylo očekávatelné, že se objeví i ve výsledku. Dalším společným prvkem karet byla konkrétní herní mechanika. Zátěž nositele prstenu přidává první a třetí karta výsledku, druhá naopak odebírá.

16.3.3 Třetí scénář - různé kultury, různé série, společná mechanika

U třetího scénáře bylo prověřeno, zda plně se ve výsledku objeví karty plně definované použitou mechanikou. Žádná karta s jinou nesdílela kulturu a sérii. Elfskou kulturu doplňoval přízrak mezi třemi nejlepšími výsledky s danou společnou mechaniku.

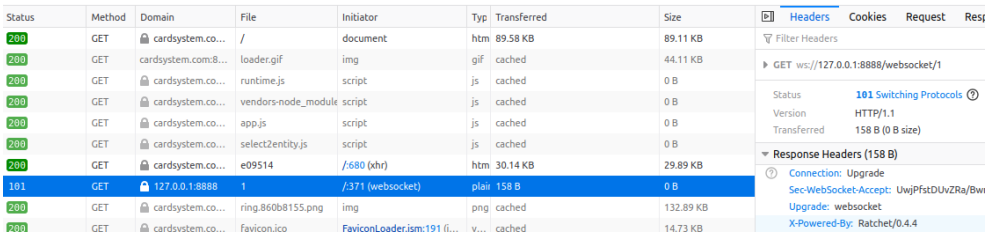
16.4 WebSocket

Činnost WebSocketu lze jednoduše ověřit. Přihlášenému uživateli se v konzoli prohlížeče po načtení stránky obsahující komunikační blok zobrazí věta WebSocket active, která se volá po navázání spojení se serverem. V přehledu požadavků a dotazů lze vidět dotaz uživatele na adresu websocket{id_uživatele}. Server v odpovědi specifikuje status 101 značící změnu protokolu komunikace a danou technologii v hlavičce Upgrade: websocket.

Tabulka hodnot druhého scénáře doporučovacího nástroje		
Kultura	Počet výskytů	Skóre
Twilight, Gandalf, Elven, Shire, Sauron, Moria	1	0.091
Série karty	Počet výskytů	Skóre
1	6	1
Klíčové slovo	Počet výskytů	Skóre
Ring-bearer, Exert	4	0.16
Remove a burden, Skirmish, Response, Add a burden	2	0.08
Tale, Spell, Strength +, Shadow, Draw, Discard, Draw deck, Assign, Fellowship	1	0.04

Tabulka 16.6: Nasbírané hodnoty z balíčku v druhém scénáři měření relevance doporučovacího nástroje

Po odeslání zprávy není automaticky nová zpráva přidána do konverzace odesílateli, avšak nejprve se zpráva validuje serverem a následně odešle na obě strany komunikace. Jakmile odesílatel uvidí svou zprávu v konverzaci, může si být jistý, že je zpráva uložená v databázi. Každá zpráva kromě konkrétního textu obsahuje i informaci o vytvoření a autorovi.



Obrázek 16.1: Přehled komunikace mezi klientem a serverem při změně protokolu na WebSocket

Tabulka hodnot u karet s nejlepším skore u druhého scénáře			
Kultura	Série	Klíčová slova	Skóre
Twilight	1	Ring-bearer, Skirmish, Response, Exert, Wins, Exert the Ring-bearer, Add a burden	1.893
Elven	1	Archer, Archery, Fellowship, Tale, Exert, Fellowship archery total	1.853
Elven	5	Archer, Archery, Fellowship, Exert, Fellowship archery total	1.773

Tabulka 16.7: První tři karty podle vypočítaného skóre ve druhém scénáři u měření relevance doporučovacího nástroje

Tabulka hodnot třetího scénáře doporučovacího nástroje		
Kultura	Počet výskytů	Skóre
Shire, Moria, Elven, Rohan, Gandalf, Gollum, Raider	1	0.143
Série karty	Počet výskytů	Skóre
1, 2, 3, 4, 5, 6, 7	1	0.143
Klíčové slovo	Počet výskytů	Skóre
Heal	7	0.206
Exert, Skirmish	4	0.118
Fellowship	2	0.059
Ring-bearer, Hobbit, Orc, Elf, Home, Forest, Mount, Unhasty, Ent, Assignment, Discard, Assign, Response, Wins, Man, Easterling, Remove a burden	1	0.029

Tabulka 16.8: Nasbírané hodnoty z balíčku v třetím scénáři měření relevance doporučovacího nástroje

Tabulka hodnot u karet s nejlepším skore u třetího scénáře			
Kultura	Série	Klíčová slova	Skóre
Twilight	6	Heal, Shirmish, Exert, Discard, Draw deck, Shadow	0.96
Shire	7	Regroup, Exert, Easterling, Discard, Move Limit, Prevent this, Skirmish, Heal	0.929
Twilight	4	Unhasty, Exert, Fellowship, Discard, Heal	0.929

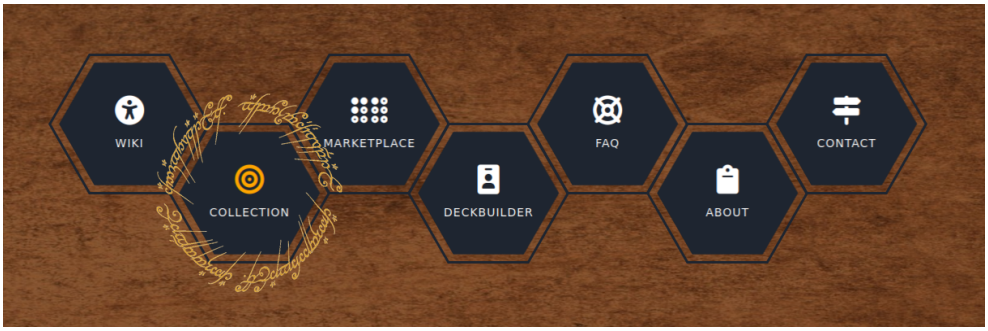
Tabulka 16.9: První tři karty podle vypočítaného skóre ve třetím scénáři u měření relevance doporučovacího nástroje

Kapitola 17

Závěr a zhodnocení

Aby byl naplněn první z cílů práce, bylo nutné vytvořit systém, kde uživatel může uložit svou kolekci, prohlížet všechny dostupné karty, stavět herní balíčky a vyměňovat karty s ostatními hráči. Všech požadovaných funkcionalit bylo docíleno. Z webového zdroje se ze získaných dat vytvořily databázové záznamy reprezentující jednotlivé abstraktní karty. Uživatel si jejich konkrétní instance může přidávat do kolekce. Následně jsou tyto instance k dispozici napříč celým systémem. Ze svých a abstraktních karet vytváří uživatel herní balíčky, kde dochází k validaci a přiřazení ke konkrétním herním formátům. Na tržišti může uživatel měnit, nakupovat a prodávat karty. Systém neumožňuje finanční transakce, slouží pouze jako prostředek pro komunikaci a jako nástroj k výměně karet. Po finalizovaném obchodu jsou karty automaticky přidány do kolekce protistraně. Mezi pozitiva u tohoto cíle autor považuje vyřešení přidávání karet do herního balíčku na základě typu karty a fakt, že po dokončeném obchodu s kartou může nový majitel ihned nakládat. Za nedostatek se dá považovat absence zrušení finalizovaného obchodu například kvůli nedodaným kartám nebo nezaplacení příslušného finančního obnosu jednou ze stran. Následky se musí řešit přes administrátora systému, který by provedl patřičný zásah v datech.

Druhý cíl vyžaduje funkčnost použití Elasticsearch indexu a jeho dostatečnou výkonnost v porovnání s databázovým ekvivalentním řešením. Elasticsearch index byl úspěšně aplikován na místa, kde bude docházet k častým požadavkům na uložená data. Po každé změně v datech se index automaticky aktualizuje, aby nedocházelo k vrácení neaktuálních dat. Na základě měření lze konstatovat, že bylo dosaženo rychlejších časů při použití Elasticsearch technologie a požadavek na výkonnost byl tedy splněn. Úspěšné zapracování našeptávače a filtrovatelných parametrů do limitace výsledku při dotazu na index autor považuje za velký úspěch. Rovněž použití vlastních analyzátorů při tvorbě indexu.



Obrázek 17.1: Hlavní grafické menu

Třetí cíl byl splněn díky implementaci komunikace mezi uživateli pomocí WebSocket technologie. Přihlášení uživatelé mohou posílat a číst zprávy v reálném čase. Lze využít globální komunikační vlákno nebo oddělené konverzace. Mezi nedostatky patří absence indikace příchozí zprávy a možnost vytvoření skupinové konverzace. Za pozitiva lze označit načítaní předchozích zpráv a přidávání nových kontaktů.

Aplikace vlastní politiky pro práci s dočasnou pamětí na více úrovních výrazně zrychlila práci se systémem. Nejlepších výsledků dosáhla interní paměť, která snížila čas načtení stránky o více než polovinu. Kvůli obecně dobré politice Symfony na úrovni ukládání dotazů a obrázků nepřinesla v této oblasti vlastní konfigurace výrazné výhody. Pouze načtení již navštívené stránky při stránkování se rapidně zrychlilo. Avšak v celkovém měřítku vlastní konfigurace dosáhla výborných výsledků a čtvrtý cíl může být označen za splněný.

Doporučovací systém v sekci tvorby vlastního herního balíčku fungující jako pomocník při výběru karet vrací relevantní výsledky. Na základě předzpracovaných metadat v kombinaci s parametry na jednotlivých kartách se utvořila skórovací funkce. Vypočítané skóre seřazených karet reflektuje vstupní data v podobě již existujícího balíčku. Předzpracovaná metadata v podobě klíčových slov získaná z textů na kartě by potřebovala do budoucna revizi, ovšem momentálně plně dostačují pro správnou funkci našeptávače.

Na základě splnění všech vytyčených cílů lze konstatovat, že práce splnila očekávání. Pro budoucí nasazení v praxi bude nutné jednotlivé služby spojit v celek kontejnerovou službou. Protože mezi autorovy silné stránky nepatří tvorba frontendu, bude nutné optimalizovat webovou aplikaci pro mobilní zařízení, což je asi největší současný nedostatek práce. Díky stále se rozšiřující komunitě dané karetní hry najde systém uplatnění. Díky možnosti použití překladů může být do budoucna zařazeno více dostupných jazyků. Velkou plánovanou změnou do budoucna bude vytvoření online herní varianty. Návrh databáze je na to z velké míry připraven. Protože jsou pravidla poměrně náročná, už od začátku se neplánovalo zařadit tuto část do současného stavu

Obrázek 17.2: Použitá varianta karty a alternativní plné odstranění licencovaného obsahu a úprava karty pro zachování původního vzhledu

práce. Protože karty obsahují licencovaný filmový materiál, pro následné použití bylo nutné karty upravit, aby splňovaly platnou legislativu. Pomocí grafického editoru byly modifikovány vysokou pixelizací. Alternativně by bylo možné úplně filmový záběr z karty odstranit a nahradit jednotným pozadím. Mohl by se také použít takový filtr, aby byl licencovaný obsah dostatečně pozměněn, avšak karta by si zachovala alespoň částečný vzhled.

Literatura

[1] JUST, J. *A Short Survey of Web Data Mining*. In: WDS'13 Proceedings of Contributed Papers, Part I, 2013, s. 59-62 [cit. 2022-04-07]. ISBN 978-80-7378-250-4. Dostupné také z: https://www.mff.cuni.cz/veda/konference/wds/proc/pdf13/WDS13_109_i2_Just.pdf

[2] Java T Point. *Data Mining- World Wide Web*. In: Blogger [online]. [cit. 2022-04-07]. Dostupné z: https://www.javatpoint.com/data-mining-world-wide-web.

[3] GINNI. *What are the types of Web Mining?*. Tutorialspoint. In: Blogger [online]. 2022-02-15 [cit. 2022-04-07]. Dostupné z: https://www.tutorialspoint.com/what-are-the-types-of-web-mining.

[4] Target Internet. *What Is Data Scraping And How Can You Use It?*. In: Blogger [online]. [cit. 2022-04-07]. Dostupné z: https://www.targetinternet.com/what-is-data-scraping-and-how-can-you-use-it/.

[5] KENNY, Colm. *What Is Web Scraping?*. Zyte. In: Blogger [online]. [cit. 2022-04-07]. Dostupné z: https://www.zyte.com/learn/what-is-web-scraping/.

[6] DENSMORE, James. *Ethics in Web Scraping*. Towards Data Science. In: Blogger [online]. 2017-07-23 [cit. 2022-04-07]. Dostupné z: https://towardsdatascience.com/ethics-in-web-scraping-b96b18136f01.

[7] JAMI. *A Guide to Ethical Web Scraping*. Empirical Data. In: Blogger [online]. 2019-12-02 [cit. 2022-04-07]. Dostupné z: https://www.empiricaldata.org/dataladyblog/a-guide-to-ethical-web-scraping.

[8] WILLIAMS, Janet. *Scrape the Web like a Master with these Technologies*. Prompt Cloud. In: Blogger [online]. 2016-08-19 [cit. 2022-04-07].

Dostupné z: https://www.promptcloud.com/blog/technologies-for-web-scraping/.

[9] DOHERTY, Erin. *MVC Architecture in 5 minutes: a tutorial for beginners*. Educative. In: Blogger [online]. 2020-04-11 [cit. 2022-04-17]. Dostupné z: https://www.educative.io/blog/mvc-tutorial.

[10] MARTIN, Matthew. *MVC Framework Tutorial for Beginners: What is, Architecture & Example*. GURU99. In: Blogger [online]. 2022-02-17 [cit. 2022-04-17]. Dostupné z: https://www.guru99.com/mvc-tutorial.html.

[11] KANJILAL, Joydip. *Comparing the MVC MVP and MVVM Design Patterns*. Developer.com. In: Blogger [online]. 2021-12-03 [cit. 2022-04-08]. Dostupné z: https://www.developer.com/design/mvc-vs-mvp-vs-mvvm-design-patterns/.

[12] BHUI, Joydip. *Understand MVC Architecture in 5 mins*. Crio.do. In: Blogger [online]. 2022-3-22 [cit. 2022-04-08]. Dostupné z: https://www.crio.do/blog/understand-mvc-architecture/.

[13] Symfony SAS. *Controller*. Symfony.com [online]. [cit. 2020-04-17]. Dostupné z: https://symfony.com/doc/current/controller.html.

[14] Symfony SAS. *Creating and Using Templates*. Symfony.com [online]. [cit. 2020-04-17]. Dostupné z: https://symfony.com/doc/current/templates.html.

[15] DB-Engines *DB-Engines Ranking of Search Engines*. solidIT consulting & software development gmbh [online]. [cit. 2022-04-17]. Dostupné z: https://db-engines.com/en/ranking/search+engine.

[16] Sematext *Elasticsearch Tutorial*. Sematext.com [online]. [cit. 2022-04-17]. Dostupné z: https://sematext.com/guides/elasticsearch/.

[17] Elasticsearch B.V. *Index modules*. [online]. [cit. 2022-04-17]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/current/index-modules.html.

[18] VESELÝ, Luděk *Seriál Elasticsearch: 4. Fulltextové vyhledávání v češtině*. Sematext.com [online]. 2017-09-28 [cit. 2022-04-17]. Dostupné z: https://www.ludekvesely.cz/serial-elasticsearch-4-fulltextove-vyhledavani-v-cestine/.

[19] DAMIEN, Alexandre *Elasticsearch the right way in Symfony*. JoliCode [online]. 2021-01-01 [cit. 2022-04-17]. Dostupné z: https://jolicode.com/blog/elasticsearch-the-right-way-in-symfony.

[20] FOSElasticaBundle *setup.md*. [online]. [cit. 2022-04-17]. Dostupné z: https://github.com/FriendsOfSymfony/FOSElasticaBundle/blob/master/doc/setup.md.

[21] Wallarm *A simple explanation of what a WebSocket is*. [online]. In: Blogger [cit. 2022-04-18]. Dostupné z: https://www.wallarm.com/what/a-simple-explanation-of-what-a-websocket-is.

[22] SOOKOCHEFF, Kevin *How Do Websockets Work?*. In: Blogger [online]. 2019-04-04 [cit. 2022-04-18]. Dostupné z: https://sookocheff.com/post/networking/how-do-websockets-work/.

[23] GRIGORIK, Ilya *WebSocket, Browser APIs and Protocols, Chapter 17*. ©2013. In: Blogger [online]. [cit. 2022-04-18]. Dostupné z: https://hpbn.co/websocket/.

[24] BABEL, Nicolas *What is Server-Sent Events?*. Axway Blog. In: Blogger [online]. 2017-09-27 [cit. 2022-04-18]. Dostupné z: https://blog.axway.com/api-streaming/server-sent-events.

[25] Ratchet *Creating Your First Application*. [online]. [cit. 2022-04-18]. Dostupné z: http://socketo.me/docs/hello-world.

[26] Heroku Dev Center *Increasing Application Performance with HTTP Cache Headers*. ©2022 Salesforce.com [online]. 2022-03-09 [cit. 2022-04-18]. Dostupné z: https://devcenter.heroku.com/articles/increasing-application-performance-with-http-cache-headers.

[27] GRIGORIK, Ilya, Posnick, Jeff *Prevent unnecessary network requests with the HTTP Cache*. Web.dev [online]. 2020-04-17 [cit. 2022-04-18]. Dostupné z: https://web.dev/http-cache/.

[28] NOTTINGHAM, Mark *Caching Tutorial for Web Authors and Webmasters*. Mnot.net ©1998-2013 [online]. 2017-10-25 [cit. 2022-04-18]. Dostupné z: https://www.mnot.net/cache_docs/.

[29] JACQUEMIN, Léo *An in-depth introduction to HTTP caching: Cache-Control & Vary*. FreeCodeCamp [online]. 2019-10-24 [cit. 2022-04-18]. Dostupné z: https://www.freecodecamp.org/news/an-in-depth-introduction-to-http-caching-cache-control-vary.

[30] TOMAYKO, Ryan *Things Caches Do*. Tomayko.com [online]. 2008-11-16 [cit. 2022-04-18]. Dostupné z: https://tomayko.com/blog/2008/things-caches-do.

[31] Symfony SAS. *Cache*. Symfony.com [online]. [cit. 2020-04-18]. Dostupné z: https://symfony.com/doc/current/cache.html#creating-a-cache-chain.

[32] Symfony SAS. *Working with Edge Side Includes*. Symfony.com [online]. [cit. 2020-04-18]. Dostupné z: https://symfony.com/doc/current/http_cache/esi.html.

[33] Symfony SAS. *HTTP Cache*. Symfony.com [online]. [cit. 2020-04-18]. Dostupné z: https://symfony.com/doc/current/http_cache.html#http-caching-and-user-sessions.

[34] Symfony SAS. *LiipImagineBundle*. Symfony.com [online]. [cit. 2020-04-18]. Dostupné z: https://symfony.com/doc/current/LiipImagineBundle/index.html.

[35] DWIVEDI, Rohit. *What Are Recommendation Systems in Machine Learning?*. AnalyticSteps [online]. 2020-04-16 [cit. 2020-04-19]. Dostupné z: https://www.analyticssteps.com/blogs/what-are-recommendation-systems-machine-learning.

[36] ThingsSolver. *Introduction to recommender systems*. [online]. [cit. 2020-04-19]. Dostupné z: https://thingsolver.com/introduction-to-recommender-systems/.

[37] SKOPAL, Tomáš *Personalized search and social context*. Searching the Web and Multimedia Databases, ©2020 [online]. [cit. 2020-04-19]. Dostupné z: https://moodle-vyuka.cvut.cz/pluginfile.php/493967/course/section/76282/BIVWM_lecture07.pdf.

[38] ISINKAYE, F.O., FOLAJIMI, Y.O., OJOKAH, B.A *Recommendation systems: Principles, methods and evaluation*. Egyptian Informatics Journal, s. 261-273, 2015-10-20 [online]. [cit. 2020-04-19]. Dostupné z: https://www.sciencedirect.com/science/article/pii/S1110866515000341.

[39] Crossing Minds. *What are today's top recommendation engine algorithms?*. itnext.io [online]. 2020-03-09 [cit. 2020-04-19]. Dostupné z: https://itnext.io/what-are-the-top-recommendation-engine-algorithms-used-nowadays-646f588ce639.

[40] Obrázek ze serveru ThingsSolver. [online]. [cit. 2020-04-19]. Dostupné z: https://thingsolver.com/wp-content/uploads/ph3-768x461.png.

[41] Obrázek ze serveru DataOx. [online]. [cit. 2020-04-07]. Dostupné z: https://data-ox.com/wp-content/uploads/2020/06/Web-scraping-process-at-data-ox.svg.

[42] Obrázek ze serveru https://www.elastic.co. [online]. [cit. 2020-04-17]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/reference/8.2/analyzer.html.

[43] Obrázek z manuálových stránek FOSElasticaBundle. [online]. [cit. 2020-04-17]. Dostupné z: https://github.com/FriendsOfSymfony/FOSElasticaBundle/blob/master/doc/usage.md.

[44] Veškeré obrazové materiály karet pochází ze serveru LOTR TCG Wiki. [online]. [cit. 2020-02-05]. Dostupné z: https://lotrtcgwiki.com/wiki/start.

[45] Obrázek ze serveru oreilly.com [online]. [cit. 2020-04-17]. Dostupné z: https://www.oreilly.com/library/view/high-performance-browser/9781449344757/images/hpbn_1702.png.

[46] Obrázek ze serveru sookocheff.com [online]. [cit. 2020-04-18]. Dostupné z: https://sookocheff.com/post/networking/how-do-websockets-work/.

[47] Obrázek ze serveru socketo.me [online]. [cit. 2020-04-18]. Dostupné z: http://socketo.me/docs/hello-world.

[48] Obrázek ze serveru heroku.com [online]. [cit. 2020-04-18]. Dostupné z: https://devcenter0.assets.heroku.com/article-images/782-imported-1443570285-782-imported-1443554754-56-original.jpg.

[49] Obrázek ze serveru symfony.com [online]. [cit. 2020-04-18]. Dostupné z: https://symfony.com/doc/current/http_cache/esi.html.

Příloha **A**

Seznam zkratek a pojmů

HTTP
Hypertext Transfer Protocol

LOTR
The Lord of the Rings

TCG
Trading Card Game

MVC
Model-View-Controller

HTML
Hypertext Markup Language

SQL
Structured Query Language

JSON
JavaScript Object Notation

API
Application Programming Interface

CORS
Cross-origin resource sharing

TLS
Transport Layer Security

ESI Edge Side Include

AJAX
Asynchronous JavaScript and XML

Příloha **B**

Obsah přiloženého DVD

- `readme.txt` ..... Popis obsahu média
- `instalace.txt` ..... Seznam instrukcí pro instalaci
- `card_system.zip` ..... Zdrojové soubory projektu
- `prace.pdf` ..... Diplomová práce ve formátu PDF
- `prace.zip` ..... Zdrojové kódy k diplomové práci ve formátu TEX