Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Vědomostní soutěž z dat na Wikipedii

Plzeň 2017 Kružej Martin

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 28. června 2017

Kružej Martin

Abstract

The goal of this thesis is to study and compare various technologies for storing and managing semi-structured data and to show an example of using this data. The SPARQL query language is used in the thesis to acquire Wikipedia data from DBpedia project. A system of questions and answers is created from the data. It is used as basis of a knowledge contest. The thesis designs principles of the contest which is playable in web application. The application is developed using modern technologies such as AngularJS, Node.js, Express and MongoDB.

Abstrakt

Cílem této práce je prostudování a porovnání různých technologií ukládání a správy semi-strukturovaných dat a demonstrace využití těchto dat. Využíváme data z projektu DBpedia a přistupujeme k nim pomocí SPARQL dotazovacího jazyka. Z těchto dat je vytvořen systém otázek a odpovědi sloužící pro vědomostní soutěž více hráčů, jejíž principy v práci navrhujeme. Tuto soutěž je možno si zahrát ve webové aplikaci, kterou v této práci navrhujeme a realizujeme. Aplikace je vytvořena za použití moderních technologií jako jsou AngularJS, Node.js, Express a MongoDB.

Obsah

1 Úvod

Pojmy data, jejich ukládání a správa, si prošly poslední dobou velkými změnami. Dříve převážně relační data s pevně danou strukturou jsou dnes dynamičtější s žádnou nebo často se měnící strukturou. Tradiční databáze nastupujícímu trendu nestačily, což vedlo ke vzniku novému způsobu práce s těmito daty, jež jsou v práci označované jako semi-strukturovaná data.

Práce se bude zabývat způsoby správy těchto dat. Představíme různé formáty pro ukládání semi-strukturovaných dat a porovnáme možnosti jejich zpracování. Ukážeme potenciální hodnotu dat a způsob jejich využití na vědomostní soutěži pro více hráčů.

Vybrali jsme Wikipedii a její data. Práce ukáže, jak k těmto datům přistoupit, jak je uložit a jak s nimi pracovat. Dále práce navrhne principy vědomostní soutěže, její různé části, pravidla, typy otázek a jejich vyhodnocení. Hlavním bodem práce je webová aplikace, jež spojí všechny předchozí body do jednoho funkčního celku.

2 Uspořádání dat

Dle míry uspořádání dělíme data následujícím způsobem [5, 13, 26]:

2.1 Strukturovaná data

Strukturovaná data jsou data, jež byla zorganizována do takové struktury, aby její položky byly snadno dostupné, dále organizovatelné a přístupné [5]. Jejich datový model určuje, jak budou data zpracována, uložena, jak k nim bude přistupováno, jaké datové typy tato data podporují (číslo, měna, datum...) a také jakékoli omezení dat (počet znaků, povolené znaky...)[13]. Lze se setkat s definicí, že jde o jakékoli data uložena v SQL, tedy relační, databázi, nebo obecně do databáze s řádky a sloupci. Taková data mají relační klíč a je velice snadné je ukládat do předem navržených polí [26].

2.2 Nestrukturovaná data

Nestrukturovaná data jsou opakem strukturovaných. Nejsou organizována do struktury či formátu, který by usnadnil přístup a zacházení s nimi.[5] Jedná se převážně o čistý text a multimediální data. Nestrukturovaná data se obecně nehodí do žádné z běžných relačních databází. Jako příklady takových dat je možno uvést: volný text, data sociálních médií, multimediální data (fotografie a video) a reprezentace signálů (satelitní snímky, data radarů, sonarů)[26].

2.3 Semi-strukturovaná data

Semi-strukturovaná data leží někde mezi strukturovanými a nestrukturovanými. Od strukturovaných se liší tím, že nemají striktní datový model [13] a nejsou organizována do komplexního systému, který by umožňoval sofistikovaný přístup a analýzu. Od nestrukturovaných se liší tak, že je s nimi spojena určitá informace, která umožňuje jednotlivým položkám dat být uložena v databázích a adresována [5]. Tyto informace mohou být metadata, tagy nebo jiné značky [26].

Hranice mezi semi-strukturovanými a nestrukturovanými je poměrně tenká. Na následujících příkladech bude ukázána jejich spojitost.

Čistý textový dokument napsaný např. v poznámkovém bloku je nestrukturovaný. Přidáním nějakých metadat či klíčových slov pro reprezentaci jeho obsahu se z něj stane semi-strukturovaný dokument. [5] Samotná fotografie či jiná grafika je nestrukturovaná, přidáním klíčových slov popisující obsah tohoto multimédia opět mluvíme o semi-strukturovaných datech [13].

3 Formát semi-strukturovaných dat

Semi-strukturovaná data se nejčastěji ukládají do formátů CSV, XML a JSON [26].

3.1 CSV

CSV (comma-separated values, hodnoty oddělená čárkami) je textový formát pro výměnu dat mezi aplikacemi. [2] Na každém řádku je několik položek oddělených čárkami, jednotlivé záznamy jsou odděleny systémem EOL znaků, tedy jeden záznam na řádek. Soubory tohoto formátu jsou snadno modifikovatelné použitím běžných tabulkových procesorů, např. Microsoft Excel [3].

Jako výhody je možno uvést jednoduchost na zpracování, dobrou čitelnost člověkem i strojem a snadné programování parseru dat uložených v tomto formátu. Mezi nevýhody patří limita funkcionality, skoro neexistující dokumentace a podpora silného datového typování [4](nekontroluje datové typy). Pro CSV nebyl nikdy vytvořen definitivní všeobecně přijatá specifikace formátu [2].

Příklad CSV dokumentu může vypadat takto:

```
1 street,city,state,owner
2 3526 HIGH ST,SACRAMENTO,CA,Josh MacIntosh
3 1889 COLD SPRINGS RD,PLACERVILLE,CA,Margaret Junior
4 5733 ANGELINA AVE, CARMICHAEL,CA,Christopher Strange
5 1813 AVENIDIA MARTINA, ROSEVILLE,CA,Patrick Jane
6 201 KIRKLAND CT, LINCOLN,CA,Theresa Lisbon
7 5 BISHOPGATE CT, SACRAMENTO,CA,Kimball Cho
8 3882 YELLOWSTONE LN,EL DORADO HILLS,CA,Wayne Rigsby
9 7756 TIGERWOODS DR,SACRAMENTO,CA,Thomas Rudy
```

3.2 XML

XML (Extensible Markup Language, rozšířitelný značkovací jazyk) je značkovací jazyk pro popis dat. Data popsána v XML jsou sebe-popisující či sebe-definující, to znamená, že struktura dat je vložena mezi samotná data. Následkem je, že není nutné připravovat nějakou strukturu pro uložení dat,

jelikož jsou data již dynamicky chápána v XML, tedy stačí data uložit ve stejné struktuře.

Základem XML je element, který je definován tagy. Každý element má začínající a ukončovací tag. Elementy mohou být vnořeny do ostatních elementů. Každý XML dokument má tzv. kořenový element, v němž jsou všechny ostatní elementy uloženy. Právě díky schopnosti vnořovat elementy XML podporuje hierarchické struktury [28].

Za výhody XML jsou považovány: čitelnost člověkem i počítačem, podpora Unicode kódování, mezinárodní standard W3C, striktní syntaxe, jednoduché a efektivní parsovací algoritmy, hierarchická struktura a nezávislost na platformě.

XML je často vyčítána přílišná upovídanost ve srovnání s alternativními formáty. Při kladení velkého důrazu na hierarchický model je možno namítnout, že XML model je stále limitován ve srovnání s relačním modelem či objektově orientovaným grafem [29].

Příklad XML dokumentu může vypadat takto:

```
<?xml version="1.0" encoding="UTF-8"?>
<messages>
  <note id="501">
    <to>Thomas</to>
    <from>Jani</from>
    <heading>Reminder</heading>
    <body>Don't forget me this weekend!</body>
  </note>
  <note id="502">
    <to>Jani</to>
    <from>Thomas</from>
    <heading>Re: Reminder</heading>
    <body>I will not!</body>
  </note>
</messages>
```

3.3 JSON

JSON (JavaScript Object Notation, JavaScriptový objektový zápis) je standardizovaný textový formát pro výměnu dat. Je založen na podmnožině programovacího jazyka JavaScript.

JSON je postaven na dvou strukturách:

- objekt – neuspořádaná množina párů klíč-hodnota. Jednotlivé páry odděleny čárkami, objekt uvozen složitými závorkami.

- pole – seřazená kolekce hodnot. Jednotlivé hodnoty odděleny čárkami [8].

Je dobře čitelný člověkem i strojem, nezávislý na platformě a nabízí široký okruh implementací. Díky tomu, že vychází s JavaScriptu, je ideální volbou pro asynchronní webové aplikace [9], především tedy ty postavené na AJAX, kdy je kladen velký důraz na načítání dat asynchronně a rychle, někde za oponou tak, aby se nezdržovalo načítání stránky [22].

Jednoduchý JSON dokument může vypadat takto:

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

3.4 Trojice

Trojice narozdíl od ostatních probraných formátů definují kromě samotných dat také vztahy mezi daty. Jde o sekvence tří termínů, označovaných jako subjekt, predikát, objekt, kde na každou řádku připadá jedna sekvence. Jde tedy o jednoduchý formát čistého textu, jehož jednotlivé záznamy odpovídají jednotlivým řádkám [14].

Trojice jsou používány pro reprezentaci RDF (Resource Definition Framework). RDF je standard pro kódování metadat a znalostí v Sémantickém webu [36]. Jde o způsob, jak rozložit libovolnou znalost na malé konkrétní části a přiřadit jim nějaký význam. RDF je jednoduchá metoda natolik, aby mohla vyjádřit jakoukoli vědomost a zároveň strukturovaná natolik, aby s daty mohli počítače pracovat.

K tomuto standardu se váží Trojice, které jednotlivé znalosti reprezentují. Tím se vracíme k termínům Trojic – subjekt, predikát, objekt – kde subjekt a objekt jsou názvy dvou věcí reálného světa a predikát je název vztahu mezi nimi, často ve formě slovesa.

Znalost se také může označovat jako fakt. V tabulce 3.1 je vyjádření čtyř faktů pomocí Trojic.

Všechny tři termíny Trojic jsou vždy jednoduché názvy ať již konkrétních věcí (*byt*) nebo abstraktních konceptů (*má*). Vždy buď odkazují na nějakou

Tabulka 3.1: Příklad Trojice

Subjekt	**Predikát**	**Objekt**
osoba_1	má	byt_1
osoba_2	má	byt_2
byt_1	má	počítač_1
byt_1	má	postel_1
byt_2	má	počítač_2
byt_2	je v	Plzni

věc, nebo označují skutečné věci ve světě. Věci, které označují, jsou nazývány jako zdroje, uzly nebo také entity. V příkladu z tabulky by *byt* označoval nějakou skutečnou entitu v reálném světě.

Predikát je vždy vztah mezi dvěma věcmi. Pořadí subjektu a predikátu je v RDF velmi důležité (RDF je orientovaný graf), když si představíme Trojici byt–má–postel a postel–má–byt, jsou na první pohled rozdílné [36].

3.5 Ostatní formáty

Ukládání semi-strukturovaných dat není omezeno pouze na uvedené textové formáty. Z dalších významných formátů je možno vybrat: BERT, YAML, MessagePack, BSON, Avro, Protocol Buffers a další [12].

4 Správa semi-strukturovaných dat

Správa dat velice často znamená uložit data do nějaké databáze. Jak bylo řečeno v první kapitole, strukturovaná data jsou často spojována s relačními databázemi. Tyto databáze byly po dlouhou dobu prominentní technologií, jak ukládat data a jak zajistit konzistentní přístup k datům několika klienty najednou. V průběhu let se sice objevily alternativy jako objektové databáze nebo XML databáze, ale tyto technologie nikdy nezískaly větší část trhu. Spíše byly absorbovány do relačních databázových systémů, tedy např. relační databáze začala podporovat ukládání dat do formátu XML.

Relační databáze tedy představovaly myšlenku, že jedno řešení stačí všemu. Tato myšlenka byla v nedávných letech zpochybněna, což mělo za následek vytvoření velkého množství alternativních databází. Těmto databázím se říká NoSQL [35].

4.1 NoSQL hnutí

V roce 2009 byl tento termín použit pro název konference zastánců nonrelačních databází [18] a znamená „Not only SQL“. Blogger a zaměstnanec RackSpace Eric Evans, jež je považován za toho, kdo tento termín proslavil, tvrdil, že NoSQL hnutí se snaží najít alternativu k vyřešení problému, na které se relační databáze nehodí [25].

Magazín Computerworld o konferenci konající se v San Francisku napsal: „Zastánci NoSQL se sešli, aby vymysleli, jak svrhnout tyranii pomalých a drahých relačních databázích ve prospěch efektivnějších a levnějších možností ukládání dat“ [21]. Nově vznikající společnosti skutečně začali pracovat bez Oraclu nebo MySQL, dvou nejpopulárnějších relačních databázích, a vytvořili své vlastní datová úložiště inspirována databázemi Amazon Dynamo a BigTable od Googlu [35].

Důvodů pro NoSQL hnutí byly vzrůstající problémy a omezení relačních databází, především jejich datový model a horizontální škálovatelnost přes několik serverů a velké množství dat. Tyto problémy byly zvýrazňovány vzrůstajícími trendy:

- Exponenciální růst množství dat generován uživateli i systémy a koncentrace těchto dat na distribuovaných systémech jako jsou Amazon,

Google a jiné cloud systémy.

- Vzrůstající závislost a komplexita dat kvůli růstu Internetu, Webu 2.0, sociálním sítím a standardizovanému přístupu k datům z velkého množství různých systémů [25].

4.2 Charakteristika NoSQL

Termín NoSQL vyzdvihuje fakt, že v těchto databázích SQL dotazování není hlavní prioritou. Tento termín se používá tedy pro mnoho různorodých databázových systémů, které nevycházejí z relačního modelu, a velice často jsou navrženy na jeden konkrétní problém. Jako základní charakteristika NoSQL databází se uvádí [20, 31]:

- jednoduché a flexibilní non-relační datové modely. NoSQL nabízí flexibilní schéma dat, nebo dokonce model bez schématu. Jsou navrženy tak, aby bylo možno do nich uložit velké množství různých dat,
- horizontální škálovatelnost přes množství serverů,
- vysoká dostupnost dat,
- BASE systémy (Basically Available, Soft state, Eventually Consistent).[20] Více v kapitole 4.4.

4.3 CAP teorém

S CAP teorémem přišel v roce 2000 Eric Brewer. [15] Tento teorém je dnes adoptovanými velkými webovými společnostmi jako například Amazon a také v NoSQL komunitě. CAP znamená následující [35]:

- **Konzistence (Consistency)** – Všichni klienti vidí stejnou verzi dat i po aktualizaci těchto dat.
- **Dostupnost (Availability)** – Všichni klienti po požadavku obdrží nějakou odpověď, i kdyby nějaká část systému nebyla funkční.
- **Odolnost k přerušení (Partition Tolerance)** – Systém pracuje stále stejně i při síťových problémech [25, 35].

Brewer tvrdí [15], že v distribuovaném systému je možno dosáhnout vždy pouze dvou z těchto tří charakteristik. Tento poznatek je důležitý pro následující kapitolu.

4.4 ACID a BASE

Relační databáze je založena sadě vlastností nazývaných ACID. Tato zkratka znamená:

- **Atomicita (Atomicity)** – buď se provedou všechny akce, nebo žádná (tedy nenastane změna dokud nenastane commit). Uživatel musí vždy vědět, v jakém je databáze stavu.
- **Konzistence (Consisteny)** – Jakákoliv transakce zachovává databázi v konzistentním stavu.
- **Izolovanost (Isolation)** – Události jedné transakce jsou ukryty před ostatními transakcemi probíhajícími paralelně.
- **Trvalost (Durability)** – Po provedení transkace a commitu výsledků do databáze systém garantuje, že tato data zůstanou zachována při selhání systému [37].

Dle CAP-teorému je možno si vybrat pouze dvě vlastnosti z konzistence, dostupnosti a partition tolerance. V dnešním světě ohromného množství rychle se měnících dat se společnosti spíše přiklání k dostupnosti a stabilitě, než ke konzistenci. Navíc je nutno počítat se škálováním a pro ACID jsou všechny tyto vlastnosti velice těžké dosáhnout, proto se NoSQL přiklonilo k přístupu zvaného BASE (Basically Available, Soft state, Eventually Consistent). Tyto tři vlastnosti nejsou nikde velice dobře definované, ale dají se popsat takto: Aplikace funguje v podstatě pořád, nemusí být pořád konzistentní, ale v nějakém známem stavu eventuálně bude [35]. NoSQL databáze tedy upustili od nároků na konzistenci a dosáhli místo toho na lepší dostupnost a partition tolerance [25].

4.5 NoSQL databáze

NoSQL databáze byly vytvářeny internetovými společnostmi za účelem splnit jejich specifické požadavky ohledně škálování, výkonu a údržby. K těmto problémům přistupovaly různě a k jejich odstranění využívaly jiných vlastností databází. I proto může být rozdělení NoSQL databází problematické. Je několik přístupů k tomu, jak tyto databáze klasifikovat, z nichž v této práci bude použito rozdělení podle datového modelu [35].

4.5.1 Key-value databáze

Key-value databáze ukládá data jako blob (binary large object - datový typ blíže nespecifikovaných binárních dat). Nejsou zde omezení na strukturu těchto dat, databázi tato struktura ani nezajímá. Kromě velikosti úložného prostoru zde není jakékoli omezení uložení dat. K jednotlivým blobům je přistupováno přes klíč. Ten je reprezentován libovolným řetězcem, např. jméno souboru, URI nebo hash [1, 31] .

Vzhledem k volnosti datového schématu je dotazování možné pouze pomocí klíče, ne hodnot. Navíc se musejí klíče přesně shodovat [25].

Považuje se za nejjednodušší databázi z hlediska Api perspektivy. Tyto databáze nemají obecně dotazovací jazyk typu SQL, jelikož si vystačí s jednoduchými příkazy get, put a delete pro získání hodnoty klíče, vložení hodnoty ke klíči a smazání klíče [1, 31].

V tabulce 4.1 je ukázka srovnání relační databáze, reprezentované Oracle, a key-value databází Riak [31].

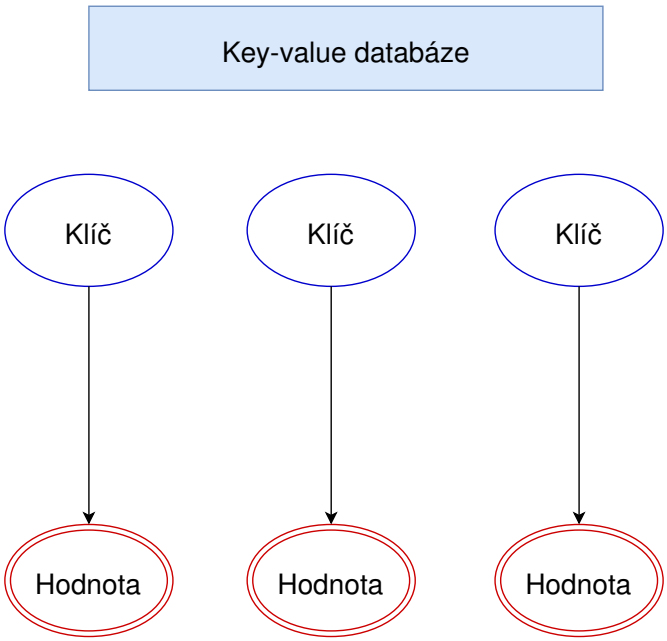
Tabulka 4.1: Sorvnání Oracle a Riak

Oracle	**Riak**
databázová instance	riak cluster
tabulka	bucket
řádka	key-value
id řádky	key

Key-value databáze jsou obecně vhodné ve chvíli, kdy aplikace přistupuje k datům jako celek přes unikátní klíč. Jako příklady je možno uvést: ukládání informací o session, konfigurace uživatelských profilů, data nákupního košíku. Ve všech těchto příkladech je k datům přistupováno přes jednoznačný identifikátor a vždy jako celek [20, 25, 31].

Key-value databáze jsou naopak nevhodné pro vyjádření vztahů mezi daty, jelikož tyto vztahy nelze vyjádřit v databázi, ale musí se o ně postarat aplikace. Problém také nastává při operacích nad větším množstvím položek, jelikož je k datům přistupováno pouze přes jeden klíč, odpovědnost znovu přechází na klientskou aplikaci. Další limitací je, pokud je požadováno dotazování podle nějakého atributu dat, toto není v key-value databázích možné [20, 31].

Mezi další používané key-value databáze patří: Redis, Memcached, OrientDB, Aerospike [7].



Obrázek 4.1: Key-value databáze

4.5.2 Dokumentové databáze

Na dokumentové databáze je možno nahlížet jako na key-value databáze s rozdílem, že hodnota není blob, ale je kompletně transparentní [20]. Právě dokument je hlavní koncept těchto databází. Dokumentem může být XML, JSON, BSON a další [25]. Jde o sebe-popisující hierarchické datové struktury skládající se z map, kolekcí či skalárních hodnot [31].

V tabulce 4.2 je srovnání relační databáze Oracle s dokumentovou databází MongoDB [31].

Tabulka 4.2: Srovnání Oracle a MongoDB

Oracle	**MongoDB**
databázová instance	MongoDB instance
tabulka	kolekce
řádka	dokument
id řádky	_id

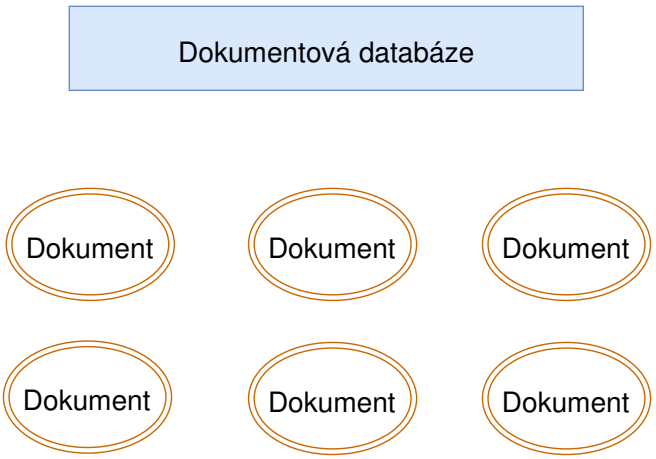
Dokumenty se mohou navzájem lišit v názvech atributů i jejich typech a přesto patřit do stejné kolekce. Jednotlivé dokumenty stejné kolekce mohou mít atributy navíc, a pokud je nemají, tak je nemusejí uvádět, v kontrastu s relačním pojetím a vyplňování null u nepřítomných atributů. Vytváření nových atributů je možné bez nutnosti znovu definovat nebo měnit existující

dokument [31].

Dokumentové databáze jsou velice vhodné pro aplikace pracující s daty, které mohou být interpretovány jako dokument, tedy blogy nebo systémy pro správu obsahu [20, 25, 31]. Dalšími příklady může být ukládání logovacích událostí, webové nebo real-time analýzy, ukládání dat E-Komerčních aplikací [31].

Limitace dokumentových databází jsou podobné těm, které mají key-value databáze. Není zde zabudovaná podpora pro vyjádření vztahů mezi jednotlivými dokumenty a transakční operace nad více dokumenty je problematická [20, 31].

Jako další příklad dokumentových databází je možno uvést: Amazon DynamoDB, Couchbase, CouchDB [7].



Obrázek 4.2: Dokumentová databáze

4.5.3 Sloupcové databáze

Sloupcové databáze ukládají data s klíči mapovaných na hodnoty a tyto hodnoty seskupeny do několika column-families, kde každá z těchto rodin je mapa dat [31]. Více bude vysvětleno dále.

V tabulce 4.3 je uvedeno srovnání relační databáze Oracle a sloupcové databáze Cassandra [31].

Data jsou ukládána do column families jako řádky, které mají mnoho sloupců, jež jsou asociovány s řádkovým klíčem. Column families jsou skupiny příbuzných dat, ke kterým je často přistupováno společně. Základní datovou jednotkou je sloupec. Tento sloupec se skládá ze jména a hodnoty, kde jméno se chová jako klíč. Každý z těchto párů key-value je jeden sloupec a je k němu přidána časová značka. Tato značka slouží k vyřešení konfliktu v zápisech, pomáhá při vypořádání se starými daty apod. Řádka je kolekce

Tabulka 4.3: Srovnání Oracle a Cassandra

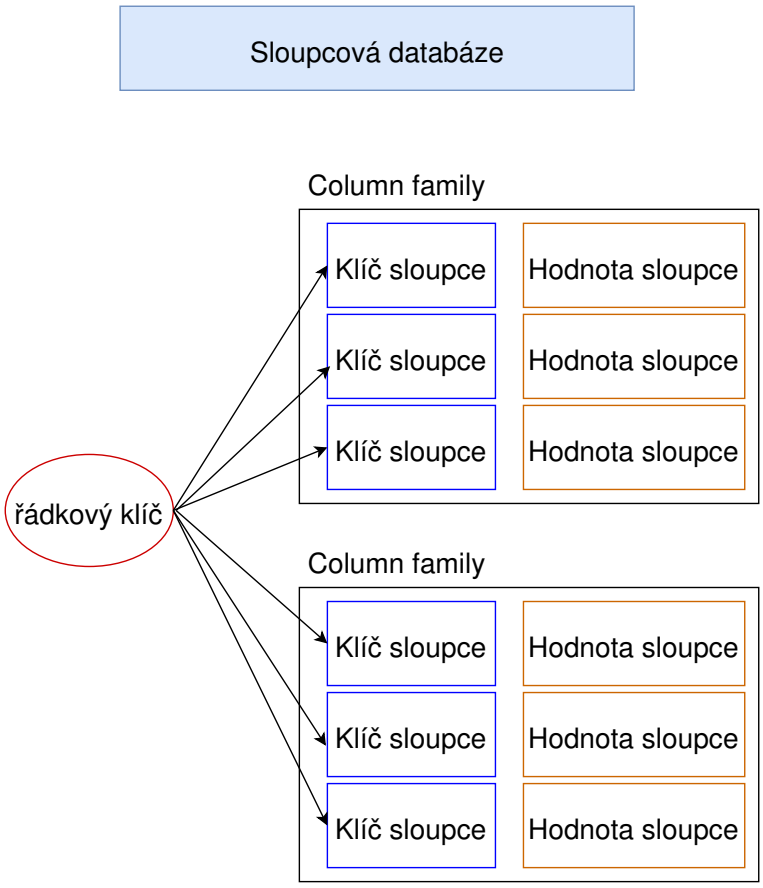
Oracle	**Cassandra**
databázová instance	cluster
databáze	keyspace
tabulka	column family
řádka	řádka
sloupec (stejný pro všechny řádky)	sloupec(různý pro různé řádky)

sloupců spojených s klíčem. Různé řádky nemusejí mít stejné sloupce. Nové sloupce se mohou do řádky přidávat bez nutnosti měnit ostatní řádky. Kolekce podobných řádků vytváří column family [31].

Sloupcové databáze se hodí na událostní logování, systémy pro správu obsahu a blogové platformy stejně, jako dokumentové databáze [31]. Vzhledem ke své schopnosti zvládnout velmi vysoký počet operací zápisu jsou vhodné pro webové analýzy [20]. Díky časové značce jsou dobře použitelné pro kontroly vypršení platnosti [31].

Sloupcové databáze sdílí nevýhody s předchozími NoSQL databázovými typy – chybí podpora reprezentace vztahů mezi daty a problematika transakcí nad více položkami [20].

Z dalších sloupcových databází je možno uvést: HBase, Accumulo, HyperTable [7].



Obrázek 4.3: Sloupcová databáze

4.5.4 Grafové databáze

Grafové databáze se zaměřují na to, co ostatním uvedeným databázovým modelům chybělo – vztahy mezi daty. Dosáhnou tím použitím grafových struktur. Data jsou uložena jako uzly a vztahy mezi nimi jsou vyjádřeny pomocí hran. Tyto vztahy mohou být jednoduché ale také velice komplexní, na což jsou grafové databáze zaměřeny. Neexistuje omezení na počet vztahů, které uzel může mít, ani jakého typu tento vztah může být.

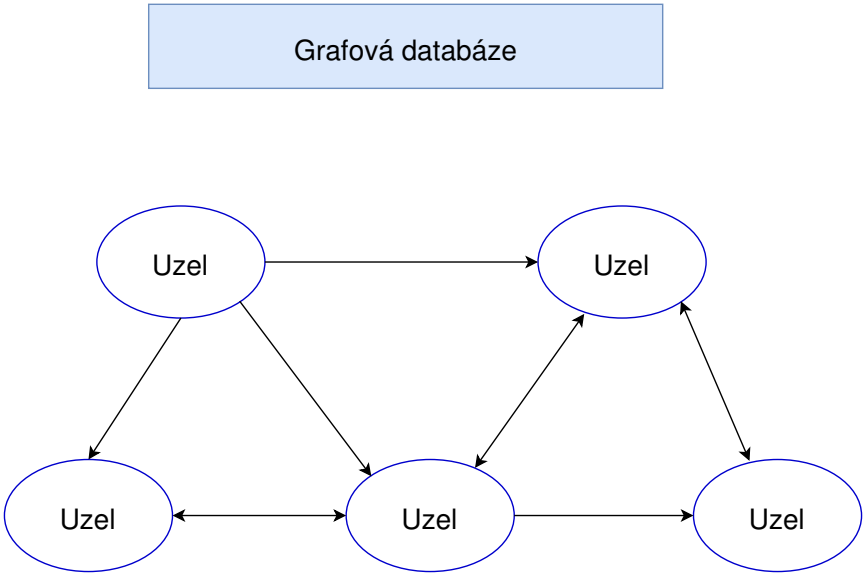
Každý uzel může mít různé vlastnosti pro popis jeho dat. Díky vztahům umožňují grafové databáze data jednou uložit a poté je interpretovat různými způsoby. Přes vztahy také probíhá dotazování, kdy databáze prochází uzly a jejich vztahy a formuluje odpověď na dotaz. Dotazování v grafových databázích se často označuje jako procházení grafu [31].

Z uvedených databázových typů jsou nejvíce zaměřeny také na vizuální reprezentaci dat a jsou tedy více přátelské pro člověka v tomto směru, než ostatní databáze [25].

Díky procházení grafem jsou komplexní operace nad více položkami dat mnohem jednodušší a rychlejší než u ostatních NoSQL databázích, ale také u relačních databázích [31].

Používání těchto databází je vhodné zejména v okamžicích, kdy je kladen větší důraz na vztahy než samotná data. Takovým příkladem jsou např. aplikace založené na lokaci, doporučovací systémy nebo komplexní síťové aplikace, tedy sociální, informativní, technické nebo biologické sítě [20]. Nevhodné použití grafových databázích může nastat při potřebě aktualizovat vlastnosti uzlů. Tato operace je komplikovaná a grafové NoSQL řešení nemusí být v tomto případě ideální [31].

Z nejpoužívanějších grafových databází je možno uvést: Neo4j, Titan, Giraph, AllegroGraph, InfiniteGraph [7].



Obrázek 4.4: Grafová databáze

4.5.5 RDF databáze

RDF databáze (Resource Description Framework) , nebo také databáze Trojic (Triple Store) (viz kapitola 3.4), je databáze pro ukládání a manipulaci s Trojicemi, tedy sekvencí řetězců mající význam subjekt-predikát-objekt [30].

Původně vymyšleno pro popis metadat zdrojů v informatice, dnes RDF databáze plní více obecnou úlohu a jsou často zmiňovány společně se sémantickým webem. Data těchto databází jsou reprezentována pomocí Trojic ve formátu subjekt-predikát-objekt.

RDF mají jistou podobnost s grafovými databázemi v případě, že interpretujeme predikát jako spojení mezi subjektem a objektem (tedy hranu mezi dvěma uzly). RDF ale nabízí specifické metody, které jdou dále než ty od grafových databází. Jako příklad lze uvést SPARQL, jenž je často spojován s Trojicemi a sémantickým webem, a většina RDF databází ho podporuje.

SPARQL je dotazovací jazyk podobný SQL s tím rozdílem, že by navržen speciálně pro RDF grafy. SPARQL dotaz, ať již jednoduchý či jakkoliv komplexní, se skládá ze tří vzorů, nazývaných *základní grafové vzory*, jež připomínají Trojice, ale na rozdíl od nich mohou být proměnné. Za použití těchto vzorů funguje vyhledávání v RDF grafu pomocí SPARQL jazyka, který za tyto vzory dosazuje Trojice a jejich sorvnáváním vytváří odpověď na zadaný dotaz [38].

Představme si, že máme databázi knih. Názvy všech těchto knih bychom získali jednoduchým SPARQL dotazem:

```
1 SELECT ?title WHERE
2 { ?book <http://purl.org/dc/elements/1.1/title> ?title }
```

Nejpoužívanější RDF databáze jsou Virtuoso, Jena, Algebraix a AllegroGraph [7].

4.5.6 Search engines

Search engines jsou specifickým druhem NoSQL databází, jelikož jsou dedikovány vyhledávání dat. Důraz se klade na rychlost, škálovatelnost, fulltextové vyhledávání a podporu komplexních dotazů.

Nejpoužívanějším zástupcem Search engines je Elasticsearch [7].

Je postaven nad Apache Lucene, což je výkonnostní fulltextová knihovna napsaná v Javě [10]. Kromě nabízených výhod v oblasti výkonu, minimálních nároků a velmi rychlého indexování, jsou s Lucene spojovány i problémy. Jde o knihovnu v Javě, a tedy vyžaduje i aplikaci v Javě pro přístup k ní, navíc vyžaduje velké znalosti pro používání, jelikož je velice komplexní. Elasticsearch používá Lucene pro indexaci a vyhledávání, ale schovává její komplexitu za jednoduchou REST Api (Representational State Transfer – architektura a přístup ke komunikaci používaný ve vývoji webových služeb) [27].

Elasticsearch navíc nabízí analýzu dat, distributivní vyhledávání v reálném čase v dokumentové databázi, kde jsou všechna pole indexovaná, škálovatelnost na stovky serverů a schopnost pracovat s obrovským množstvím dat (až petabajty) [16, 23].

Dalšími používanými mezi search engines jsou Solr, Splunk a Sphinx [7].

4.5.7 Multi-model databáze

V některých zdrojích se lze dočíst o tzv. multi-modelových databázích. Tyto databáze nepodporují jediný datový model, ale hned několik. Mezi nejpoužívanější multi-modelové databáze patří: MarkLogic, OrientDB, Apache Drill, Virtuoso [7, 11].

5 Návrh vědomostní soutěže

5.1 Principy soutěže

Základním principem vědomostní soutěže je položení otázky a očekávání odpovědi. Při zodpovězení otázky hráčem je odpověď vyhodnocena a hráč obeznámen se svým výsledkem. V případě úspěchu je odměněn nějakým postupem nebo bodovým ohodnocením, neúspěch obecně znamená penalizaci.

Při aplikaci na hru více hráčů je nutné přidat element soutěživosti. Vybrali jsme způsob, kdy jsou správné odpovědi bodově ohodnoceny a hra ukončena vítězstvím hráče s nejvyšším počtem bodů. Tento způsob je limitován ve své jednoduchosti, jelikož hráči postrádají interakci, což je jedna z hlavních myšlenek této práce. Věříme, že hráč je potěšen, když může svým rozhodnutím změnit soupeřovo strategii, např. mu vnutit otázku, která mu bude dělat problémy. S těmito principy byl vytvořen návrh na vědomostní soutěž v této práci.

Ještě předtím, než budou principy soutěže vysvětleny, je nutno si představit některé pojmy, které se v následujících odstavcích používají.

- **Otázka typu K** – tato otázka vychází z klasické představy vědomostní otázky. Je to otázka, která hráč nabízí možnosti, z nichž jedna je správná.
- **Otázka typu S** – tato otázka poskytuje hráči fakt a nabízí odpovědi ANO/NE. Hráč je předpokládán odpovědět, jestli si myslí, že je fakt správný či nesprávný.
- **Otázka typu N** – tato otázka očekává od hráče číselnou odpověď. Nenabízí žádné možnosti. Hráč je předpokládán odpovědět číslem, které odpovídá odpovědi nebo je alespoň blízko, tedy blíže než odpovědi protihráče.
- **Otázka typu W** – tato otázka vyžaduje od hráče doplnit odpověď sám. Nenabízí žádné možnosti.

Tyto typy otázek jsou použity ve vědomostní soutěži implementované v této práci. Následuje popis samotné soutěže.

5.2 Rozdělení hry

Soutěž je rozdělena na rozstřel a hlavní hru. Pokud není uvedeno jinak, otázky jsou společné pro všechny hráče a na otázky se váže časové omezení, které hráč má na zodpovězení otázky. Pokud nestihne odpovědět včas, je jeho odpověď automaticky brána jako špatná.

5.2.1 Rozstřel

Všichni hráči postupně odpovídají na otázky typu K. Za správné odpovědi jsou bodově ohodnoceni v závislosti na rychlosti, s jakou na otázku odpověděli. Za nesprávně zodpovězenou otázku nedostávají žádné body.

Účelem rozstřelu je hráče rozehrát a odměnit je body, které potom použijí v hlavní hře.

5.2.2 Hlavní hra

Hráči začínají hlavní hru s body, které nahráli v rozstřelu. Pro získání dalších bodů musejí vyhrát v některých z následujících kol, na které je hra rozdělena. Hráč, který ztratí všechny své body, nebo nenahrál žádný bod v rozstřelu, končí hru jako poražený. Hráč s nejvyšším počtem bodů na konci kol vyhrává hru.

5.2.3 Kolo

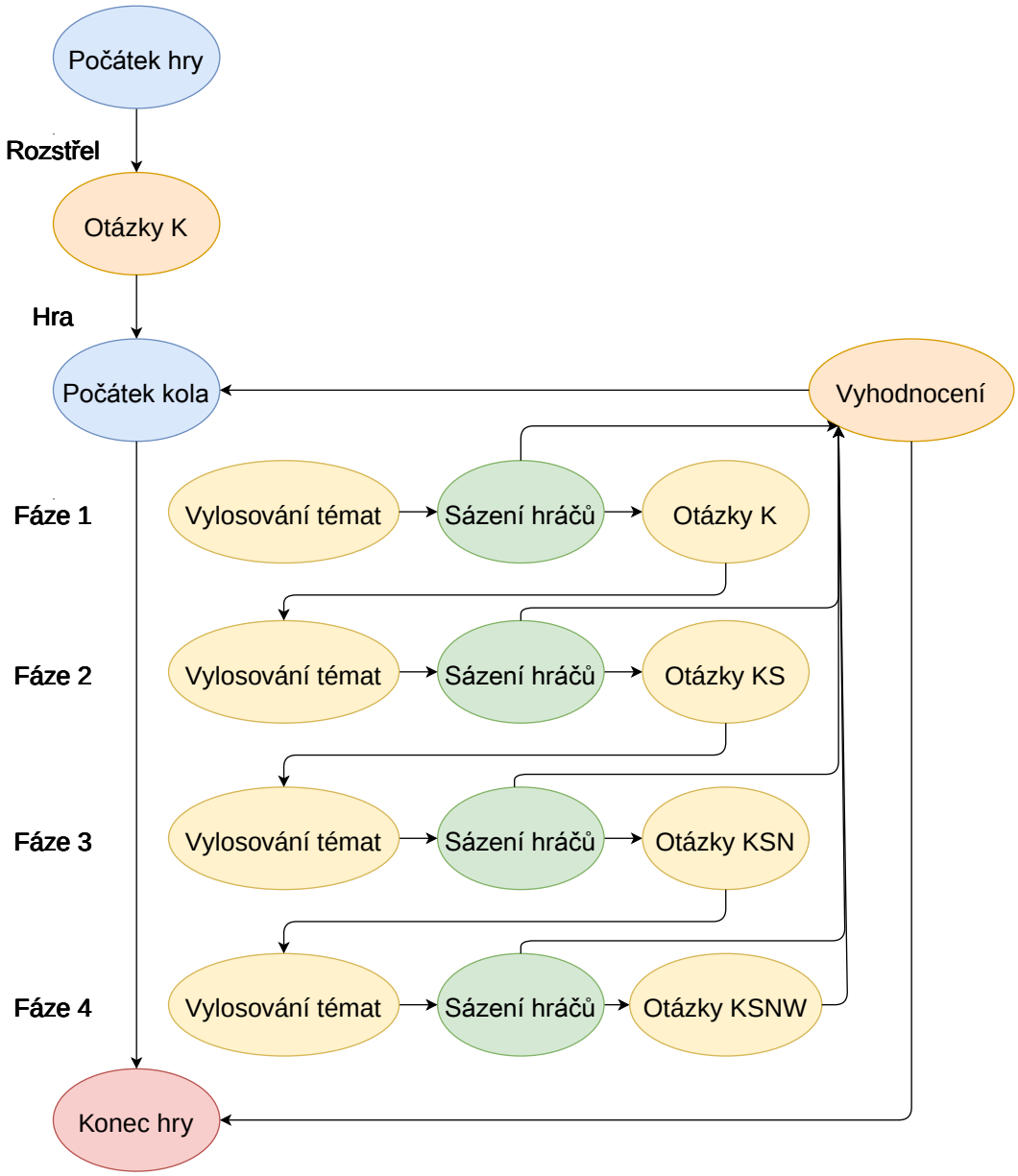
Každé kolo začíná vylosováním náhodných kategorií (témat) otázek. Postupně je každý hráč vyzván ke svému tahu, kdy má na výběr ze dvou možností. Pokud chce vybrané kolo hrát, musí vsadit určitou část svých bodů, za což dostane možnost vybrat některá z nabízených témat. Jím vybraná témata získají určité procento šance, že otázky v kole budou právě z těchto témat. Hráč má tedy příležitost tímto způsobem ovlivnit otázky kola ke svému prospěchu.

Pokud více než jeden hráč vsadili své body a vybrali svá témata, začíná fáze kola. Každé kolo obsahuje až čtyři fáze. V každé z těchto fází jsou hráčům pokládány otázky určitého typu. Za správné odpovědi získávají hráči žetony. Hráč s nejvyšším počtem žetonů na konci kola získává vsazené body všech hráčů. Každá fáze používá jiné téma, použitá témata jsou ze hry vyřazena. Témata jsou vždy náhodně vybrána z těch témat, které měli hráči možnost ovlivňovat na začátku fáze. Hráčům jsou témata neznámá dokud nezačne fáze. Mezi fázemi mají hráči možnost zkontrolovat stavy protivníků, tedy jejich žetony, a rozhodnout se, jak budou pokračovat. Pro pokračování

do další fáze musejí hráči vsadit další body, druhou možností je ukončit kolo, čímž ztratí body, které do hry vložili. Fáze se od sebe liší v pokládaných typech otázkách.

- **1. fáze** – pokládány otázky typu K. Každá správná odpověď v časovém limitu získává hráči žeton.
- **2. fáze** – pokládány otázky typu K a S. Každá správná odpověď v časovém limitu získává hráči žeton.
- **3. fáze** – pokládány otázky typu K, S a N. Odpovědi na otázky typu K a S získávají hráči žeton, otázky typu N odměňují pouze jednoho správného a nejrychlejšího hráče.
- **4. fáze** – pokládány otázky typu K, S, N a W. Pouze nejrychlejší a správná odpověď získává hráči žeton.

Pokud na konci čtyř fází mají někteří z hráčů stejný počet žetonů, rozhoduje super otázka náhodného typu, za kterou získá žeton pouze nejrychlejší správná odpověď. Vítěz kola získává vsazené body hráčů a začíná další kolo soutěže.



Obrázek 5.1: Diagram hry

6 Data Wikipedie

Cílem práce je vytvořit aplikaci vědomostní soutěže, která využívá data z Wikipedie, tedy je nutné tato data získat. Naštěstí pro nás byl tento problém již vyřešen projektem DBpedia [6].

6.1 Projekt DBpedia

Projekt DBpedia si dal za cíl extrahovat veškerá data z Wikipedie a zpřístupnit tyto informace na webu. Data DBpedia jsou uložená ve formátu Trojic a jsou volně dostupná ke stažení pro sestavení vlastní databáze anebo je možno přistoupit k těmto datům přes online přístup projektu DBpedia.

6.1.1 Online přístup

Online přístup k DBpedia datům je umožněn přes SPARQL endpoint za použití multi-model databáze OpenLink Virtuoso [32]. Za použití SPARQL dotazovacího jazyka je zde možné pokládat dotazy na DBpedia Virtuoso server a dostat odpovědi. Tento SPARQL endpoint má ale omezení pro uživatele, jmenovitě omezení maximálního počtu řádků na výsledek dotazu a maximální čas na vykonání dotazu. Tato omezení mají za následek méně komplexní dotazy a nemožnost dostat všechny odpovědi na dotaz v případě, že jich je více než maximální povolené množství [33].

Tato omezení limitují možnosti dotazování a tedy vytváření otázek pro aplikaci. Řešením by bylo sestavení vlastní databáze.

6.1.2 Vlastní databáze

Pro sestavení vlastní databáze jsou potřeba DBPedia data, jež jsou stažitelná ze stránek projektu DBpedia (tzv. data sety), a samotná databáze. Jak bylo řečeno, tato data jsou Trojice, a tedy náš výběr je omezen na NoSQL. Nelze samozřejmě tvrdit, že klasické relační databáze by nemohly Trojice uložit a pracovat s nimi, ale bylo by nutné velice komplexního řešení, kdy každý subjekt, predikát a objekt by musely mít vlastní tabulky a přes složité JOIN operace by se získávaly jednotlivé informace.

V předešlých kapitolách byly probrány různé NoSQL řešení. Nyní budou srovnány dle schopnosti pracovat s Trojicemi.

Key-value databáze jsou založeny na vyhledávání dat pomocí unikátního klíče. Už z tohoto pohledu je jasné, že pro Trojice v neupravené formě nejsou vhodné.

Sloupcové databáze se specializují na velké množství dat a jsou jakousi nadstavbou key-value databázích. Tyto databáze nebyly navrženy pro data typu Trojic a vyjadřování vztahů mezi jednotlivými záznamy by bylo složité.

Grafové databáze jsou podobné RDF v tom, že také popisují druh grafu. Grafové databáze kladou velký důraz na typ grafu (neorientovaný, orientovaný, ohodnocený, hypergrafy...), kdežto RDF záznam je vždy pouze orientovaná hrana mezi dvěma uzly. Grafové databáze používají vlastní dotazovací jazyk, na rozdíl od RDF SPARQLu. V neposlední řadě grafové databáze kladou velký důraz na uzel a jeho vlastnosti, tedy něco, co RDF vyjadřuje dalšími záznamy subjekt-predikát-objekt. RDF a grafové databáze jsou tedy oba druhem grafu, ale řeší stejný problém různě a síla grafových databází tkví v jiné oblasti, než je pro Trojice důležité. Tedy i přes společné rysy nejsou grafové databáze pro Trojice tím nejvhodnějším řešením z nabízených.

Dokumentové databáze jsou postavené na dokumentu, který velice často bývá JSON, XML nebo podobný textový formát, nikoliv Trojice. Největší zástupce dokumentových databází, MongoDB, si s Trojicemi v neupravené formě neporadí. Bylo by nutné převést Trojice do jednoho z textových formátů a s nimi poté dál v dokumentové databázi pracovat. V případě MongoDB je ještě možnost připravit další databázi, tentokrát AllegroGraph, což je grafová databáze s podporou RDF. MongoDB a AllegroGraph mají implementovány rozšíření, které jim umožňují spolupracovat ve smyslu, že je možno přes AllegroGraph posílat MongoDB dotazy jazyka SPARQL a umožnit tak MongoDB pracovat s RDF daty, i když je nativně nepodporuje. Toto řešení má zřejmou nevýhodu nutnosti existencí dvou databází a jejich vzájemnou spolupráci a konfiguraci [19].

Vzhledem k existenci RDF databází, které jsou specializované na Trojice, se tyto databáze jeví jako rozumná volba. DBPedia sama nabízí návod na sestavení databáze za použití Virtuoso [34].

Výhody jsou samozřejmě nativní podpora Trojic a dotazovacího jazyka SPARQL, nejpoužívanějšího dotazovacího jazyka sémantického webu. RDF však nejsou naší poslední volbou. Projekt na katedře KIV Západočeské univerzity v Plzni [17] se zabýval indexací Trojic data setů DBpedie do search enginu Elasticsearch. Kromě DBPedia dat ve formátu Trojic v databázi Virtuoso máme také možnost pracovat s Wikipedia daty ve formátu JSON v search enginu Elasticsearch.

Na tyto dvě možnosti se podíváme blíže.

6.2 Srovnání Virtuoso a Elasticsearch

Požadavky na databázový systém v naší aplikaci není pouze na uložení datových setů z DBpedie, ale také pro ukládání samotných otázek a dalších možných databázových požadavků aplikace. I to hraje roli při srovnání databázových řešení.

Tabulka 6.1: Srovnání Elasticsearch a Virtuoso

	Elasticsearch	**Virtuoso**
Databázový model:	Search engine	Multi-model
License:	Open Source	Open Source
Rok vydání:	2010	1998
Nynější verze:	5.2.2, Únor 2017	7.2.4, Duben 2016
Jazyk implementace:	Java	C
Operační systém:	Všechny OS s Java VM	Linux, OS X, Solaris, Windows po další konfiguraci
API:	Java API, REST HTTP/JSON API	HTTP API, OLE DB, WebDAV, ADO.NET, IBDC, OBDC
SQL:	ne	ano
Instalace:	velice jednoduchá	záleží na systému
Popularita:	vysoká, stoupající	nízká, stagnující
Přidání dat:	indexace	SPARQL funkce
Formát dat:	JSON	Trojice

Z tabulky je možno vyčíst několik zajímavých údajů. Obě databáze mají velice rozdílné databázové modely vyplývající z rozdílných požadavků. Virtuoso se svým multi-modelem poskytuje komplexní řešení správy dat a obsahuje podporu pro RDF. Elasticsearch je ve svém jádru vyhledávacím enginem a tedy toto databázové řešení nabízí uložení čehokoliv, co se dá zaindexovat, a klade velký důraz na rychlost vyhledávání.

Při mnoho společných rysech se liší zásadně ve způsobu přidávání dat. Virtuoso pracuje přes svůj dotazovací jazyk SPARQL a jeho funkce, zatímco u Elasticsearch je nutno tzv. *mapování indexu* předtím, než je možno vkládat data. Tuto indexaci již vyřešil projekt katedry KIV [17] a jak bylo řečeno DBpedia nabízí návod na instalaci Virtuoso i s daty.

Pro naši aplikaci zásadní rozdíl tedy bude v rozdílném způsobu dotazování a tedy získávání dat z databáze. Je tedy nutné srovnat dotazy Elastic-

search a SPARQL.

6.3 Srovnání dotazování Elasticsearch a SPARQL

Pro srovnání byl použit Online přístup projektu DBpedia a plugin Sense na běžícím serveru s Elasticsearch . Nejdříve srovnáme několik jednoduchých dotazů, na kterých srovnáme syntaxi, čitelnost a jednoduchost zápisu.

6.3.1 Dotaz 1

Všechny videohry od vývojáře Supermassive Games.

SPARQL

```
SELECT DISTINCT ?title WHERE {
     ?game a dbo:VideoGame  .
     ?game foaf:name ?title .
     ?game dbo:developer ?developer .
     ?game dbo:developer db:Supermassive_Games
}
```

Elasticsearch

```
GET documents/VideoGame/_search
{"query":{"match":{"developer":"Suppermasive Games"}}}
```

6.3.2 Dotaz 2

Všechny státy, kde žije více obyvatel, než 100 000.

SPARQL

```
SELECT DISTINCT ?population ?name WHERE {
     ?country a dbo:Country .
     ?country foaf:name ?name .
     ?country dbo:populationTotal ?population .
FILTER (?population > 100000)
}
```

Elasticsearch

```
GET documents/Country/_search
{"query":{"range":{"population-total":{"from":100000}}}}
```

6.3.3 Dotaz 3

Všechny bitvy, jejich název a výsledek, kde velel Napoleon Bonaparte a zvítězil.

SPARQL

```
SELECT DISTINCT ?title ?result WHERE {
 ?battle a dbo:MilitaryConflict .
 ?battle dbo:commander db:Napoleon .
 ?battle foaf:name ?title .
 ?battle dbp:result ?result .
FILTER(bif:contains(?title, "Battle"))
FILTER(bif:contains(?result, "'French victory*'"))
}
```

Elasticsearch

```
GET documents/MilitaryConflict/_search
{"query":{"bool":{"must":[{"match":{"commander":"Napoleon"}},
{"match":{"result":"French victory"}}]}}}
```

6.3.4 Dotaz 4

Všechny hry a země jejich původu, které byly vytvořeny na platformu Playstation 4, byly vydány po datu 1.1. 2015 a země jejich původu platí Eurem.

SPARQL

```
SELECT DISTINCT ?gameName ?countryName WHERE {
      ?game a dbo:VideoGame .
      ?platform dbo:computingPlatform dbr:PlayStation_4 .
      ?game dbo:releaseDate ?release .
      ?game dbo:developer ?developer .
      ?developer dbo:locationCountry ?country .
      ?country dbo:currency dbr:Euro .
```

```
      ?country foaf:name ?countryName .
      ?game foaf:name ?gameName .
FILTER (?release > "2015-01-01"^^xsd:date)
}
```

Elasticsearch

```
nelze jedním dotazem
```

6.3.5 Dotaz 5

Všechny státy, které mají více než 100 000 obyvatel, jejich hlavní města mají více než 1 000 000 obyvatel a leží v Evropě.

SPARQL

```
SELECT DISTINCT ?population ?name WHERE {
     ?country a dbo:Country .
     ?country foaf:name ?name .
     ?country dbo:populationTotal ?population .
     ?country dbo:capital ?capital .
     ?capital dbo:populationTotal ?populationCapital .
     ?country dct:subject dbc:Countries_in_Europe .
FILTER (?populationCapital > 1000000)
FILTER (?population > 100000)
}
```

Elasticsearch

```
nelze jedním dotazem
```

6.4 Závěr srovnání

Na jednoduchých dotazech (1 – 3) vidíme, že Elasticsearch je daleko méně upovídaný. U Elasticsearch stačí díky mapování indexů pouze srovnávat ty záznamy, kde určitý záznam odpovídá našim hledaným parametrům, na rozdíl od SPARQL, kde je nutné postupně definovat hledané záznamy a až poté s nimi pracovat.

Síla SPARQLu tkví v tom, že při dotazování na určitý záznam v databázi nám umožňuje zároveň dotazovat další záznamy v relaci s naším hledaným záznamem. Tyto komplexní dotazy (4 a 5) toto ukazují. Díky této vlastnosti

je možné vytvořit komplexnější otázky, nezaložené pouze na vlastnostech jednoho záznamu v databázi, ale i dalších záznam v relaci s ním.

Elasticsearch nabízí velice rychlé a full–textové vyhledávání, ale ani jedna z těchto vlastností není natolik významná, aby se vyrovnala komplexní možnostem dotazování SPARQLu. Rychlost na našem počtu dat je zhruba srovnatelná a full–textové vyhledávání využito není. Elasticsearch dále zamezuje použití relačního přístupu a tedy omezuje možnosti dotazování a tedy i otázek.

Závěr tohoto srovnání tedy je, že použití Trojic a SPARQLu je výhodnější z hlediska naší aplikace, než použití jiných databázových technik. Zbývá rozhodnout, jestli bude použit vlastní Virtuoso server nebo Online přístup projektu DBpedia.

Vraťme se k Online přístupu a jeho omezení a rozhodněme, jestli jsou pro nás natolik limitující, aby zabránila použití Online přístupu v naší aplikaci.

Počet připojení za vteřinu

DBPedia uvádí omezení na počet připojení za sekundu na jejich SPARQL endpoint, ale neuvádí přesný počet. Cílem DBpedie je tímto omezením zamezit přístupu špatně napsaným dotazům a robotům. Při zvýšené opatrnosti se dá omezení snadno vyhnout.

Počet vrácených řádků

Omezení na počet vrácených řádků/záznamů je na první pohled problematický. Toto číslo je uvedeno jako 2000 maximálních záznamů, jež DBpedia vrací. Při praktickém dotazování na otázky však nebudeme očekávat takto vysoký počet odpovědí a pokud ano, SPARQL nabízí tzv. *offsety* a *limity* pro posun záznamů a vybrání jiných.

Časové omezení dotazu

Omezení, kolik času SPARQL endpoint jednotlivým dotazům věnuje. Naše dotazy sice mohou být náročné, nikoliv ale natolik, aby si vyžádali tolik času, kolik je na endpointu vymezeno. Toto omezení by na naší aplikaci nemělo mít žádný vliv.

Tato omezení se tedy mohou dotknout naší aplikace, ale jsou zde metody, jak se jim vyhnout a je tedy možné SPARQL endpoint použít jako databázi DBpedia a tedy Wikipedia dat.

7 Webová aplikace

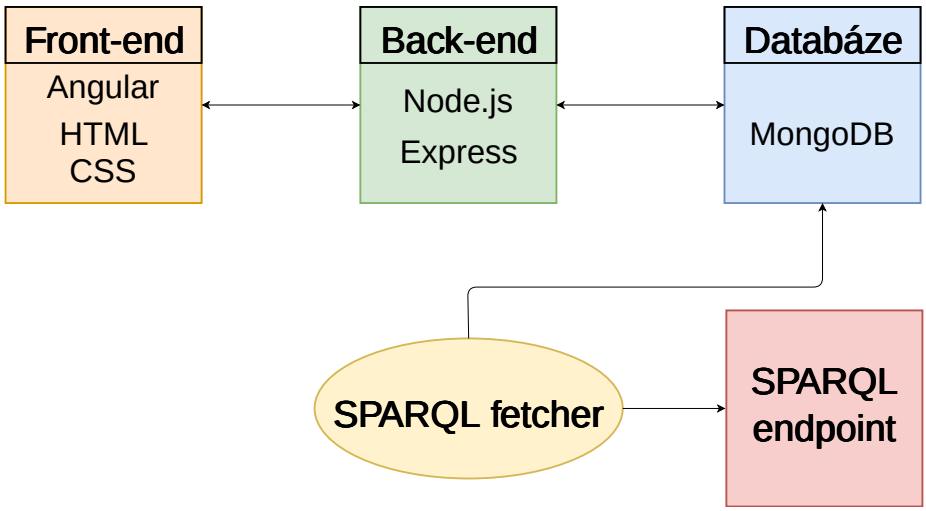
Pro vytvoření webové aplikace použijeme moderní *web stack* postavený na Javascriptu – **MEAN stack**. Tato zkratka je tvořena použitými technologiemi, jde o následující: **M**ongoDB, **E**xpress, **A**ngular a **N**ode.js [24].

MongoDB již bylo v práci zmíněno jako NoSQL databáze a databázovou roli splňuje i v tomto *stacku*. Angular je JavaScriptový framework, který umožňuje vytvořit dynamické SPA (*single-page application*). Ve *stacku* tedy jednoduše řečeno zodpovídá za to, co uživatel vidí. Express je framework, který pohání JavaScript v back-end části aplikace a zodpovídá za REST API. Express běží v prostředí Node.js.

Z popisu vyplývá, že jak front-end, tak back-end jsou napsány v Javascriptu, tedy není nutnost žádného dalšího skriptovacího jazyka. Výhoda MongoDB jako databáze je JSON formát souborů, který JavaScriptu velice vyhovuje. V neposlední řadě Node.js umožňuje asynchronní volání funkcí, a tedy při čekání na výsledek funkce je možno dělat něco jiného.

Vzhledem k závěru předešlé kapitole bude aplikace využívat pro získání otázek z dat Wikipedie SPARQL endpoint projektu DBpedia. Konfigurace jednotlivých otázek budou obsahovat SPARQL dotazy pro získání odpovědí. Tyto dotazy budou spouštěny na SPARQL endpoint a výsledky ukládány do databáze. Tím se databáze naplní otázkami, které budou použity při soutěži. Proto je součástí aplikace skript tzv. `SPARQL Fetcher`. Napsaný v JavaScriptu a spustitelný pod Node.js jeho úkolem je připojení na SPARQL Endpoint, položení dotazů pro získání otázek a následně jejich uložení v databázi.

Diagram aplikace je uveden na obrázku 7.1. Kromě zmíněních technologií je ve front-endu vyskytují navíc CSS a HTML, které neodmyslitelně k webovým aplikacím patří.



Obrázek 7.1: Diagram webové aplikace

Následuje popis jednotlivých částí aplikace. Je třeba doplnit, že aplikace byla vyvíjena na Windows. Také je třeba zmínit, že instalace balíčků a knihoven probíhala přes nástroj `npm`, který umožňuje jednoduchými příkazy knihovny a všechny jejich závislosti nainstalovat.

7.1 Databáze

Pro komunikaci mezi serverem a MongoDB databází je třeba přes `npm` nainstalovat knihovnu `mongoose`. Tato knihovna vyřizuje veškeré požadavky, které na databázi máme. Nabízí tzv. `modely`, struktury obsahující funkce pro vkládání, úpravu a získávání dokumentů z databáze, a definici struktury dokumentu, tzv. `schéma`.

Toto schéma definuje strukturu dokumentu, jeho parametry a typy těchto parametrů. Veškerá práce s databází vychází z těchto modelů.

Modely používané v aplikaci jsou následující:

- `user` – model uživatele
- `question` – model otázky
- `game` – model hry
- `round` – model jednotlivých kol hry

7.2 SPARQL Fetcher skript

Skript napsaný v JavaScript a běžící pod Node.js slouží k získání otázek a jejich uložení do databáze. Nelze jej spustit v rámci aplikace (tedy přes REST požadavek - volání funkce webového serveru), ale spouští se izolovaně. Skript nahrává z konfiguračního souboru otázky a posílá jejich nastavené SPARQL dotazy na SPARQL endpoint. Výsledné otázky ukládá do databáze.

Na tabulce 7.1 je uvedena konfigurační struktura otázky.

Tabulka 7.1: Struktura otázky

Parametr	**Možné hodnoty**
type	classic, statement, numeric nebo word
difficulty	1, 2 nebo 3
category	geography, literature, sport, games, history nebo politics
statement	otázka nebo tvrzení
rightAnswer	SPARQL dotaz na správné odpovědi
wrongAnswer	SPARQL dotaz na špatné odpovědi nebo prázdné

Typy otázek odpovídají otázkám definovaným v kapitole **Vědomostní soutěž** dle svých počátečních písmen, tedy typům C, S, N a W.

Příklad konfigurace otázky je uveden v tabulce 7.2.

Schéma modelu otázek je v databázi stejné s rozdílem, že *rightAnswer* a *wrongAnswer* již neobsahují SPARQL dotazy, ale jejich výsledek (tabulka).

Tabulka 7.2: Příklad otázky

Parametr	**Hodnota**
type	classic
difficulty	1
category	politics
statement	Who is the current Prime minister of Czech Republic?
rightAnswer	select ?query WHERE { db:Czech_Republic dbo:leader ?leader. ?leader rdfs:label ?query . ?leader dbp:office dbr:Prime_Minister_of_the_Czech_Republic . filter langMatches(lang(?query),"en")}
wrongAnswer	sselect ?query WHERE { ?person dct:subject dbc:Prime_Ministers_of_the_Czech_Republic . ?person a dbo:Person . ?person rdfs:label ?query . filter langMatches(lang(?query),"en")}

Tabulka 7.3: Příklad výsledků SPARQL dotazu

Parametr	**Hodnota**
rightAnswer	Bohuslav Sobotka
wrongAnswer	Bohuslav Sobotka, Petr Nečas, Stanislav Gross, Josef Tošovský, Petr Pithart, Jiří Paroubek, Miloš Zeman, Vladimír Špidla, Jan Fischer, Václav Klaus, Jiří Rusnok, Mirek Topolánek

Na tabulce 7.11 jsou uvedeny počty otázek jednotlivých typů a kategorií a potom celkový součet všech otázek.

7.3 Back-end

Back-end aplikace má na starosti několik věcí - vytvoření webového serveru, obsluhu REST požadavků a databáze. Vytvoření serveru probíhá přes knihovnu `express` a obsluhu databáze zajišťuje `mongoose`. Kromě obsluhy hry se přes REST požadavky řeší ještě přihlášení a registrace uživatelů.

Pracuje se s modelem uživatele, jehož schéma je ukázáno v tabulce 7.4.

Tabulka 7.4: Struktura uživatele

Parametr	**Popis**
username	uživatelské jméno
password	heslo

Uživatel tedy bude mít své uživatelské jméno a heslo a jeho autentizace bude spočívat ve srovnání obou parametrů. Je třeba si samozřejmě uvědomit, že ukládat heslo v *plain* formě není žádoucí. K tomu slouží tzv. *hash* funkce. Abychom nemuseli psát vlastní, necháme se o to postarat knihovnu `passport-local-mongoose`. Ta se automaticky postará o *hashování* (jednocestná transformace hesla na zdánlivě nesmyslný řetězec) a *solení* hesla (přidání posloupnosti náhodných čísel k *hash* funkci).

Databázová část je nyní hotova. Stačí přidat *session*, abychom uživatele udrželi přihlášeného. K tomu slouží knihovny `passport` a `passport-local`.

Hlavní úlohou back-endu je obsluha hry, o čemž bude řečeno více v kapitole 7.5.

7.4 Front-end

Front-end aplikace se stará o uživatelské rozhraní, komunikuje s uživatelem a přijímá jeho vstupy. Statické stránky jsou napsány v HTML a CSS, zatímco dynamická část aplikace je psána v JavaScriptu. Framework Angular umožňuje aplikaci rozdělit na tři části – datová část, zobrazovací část a řídící část, nazýváno také jako **MVC** - model, view, controller.

Jednotlivé stránky front-endu jsou uvedeny v tabulce 7.5.

Tabulka 7.5: Front-end stránky

Název	**Popis**
home	domovská stránka zobrazující základní informace o aplikaci (viz A.1)
rules	obsahuje pravidla hry pro rychlé seznámení se s hrou (viz A.2)
login	obsahuje formulář pro přihlášení (viz A.3)
register	obsahuje formulář pro registraci (viz A.4)
gameCreation	obsahuje formuláře pro vytvoření a připojení do hry (viz A.5)
game	obsahuje veškerý design ke hře (viz A.6)

Formuláře pro přihlášení a odhlášení jsou odesílány na server, který uživatele autorizuje, jak bylo uvedeno v předchozí kapitole. Front-end již jen zobrazuje výsledek. Pohledy `home` a `rules` jsou statické stránky a nevyžadují žádnou další pozornost.

Hlavní úlohou front-endu je zobrazování a hraní hry, o čemž bude řečeno v další kapitole.

7.5 Hra

Obsluha hry je rozdělena mezi všemi třemi částmi aplikace. Front-end hru zobrazuje, přijímá vstupy uživatele, které posílá serveru, jež načítá a ukládá informace o hře do databáze a vyhodnocuje je.

Nejdříve je třeba se podívat na databázovou část hry, tedy jaké informace a v jakém formátu jsou ukládány.

7.5.1 Herní schémata

Hra využívá dvou schémat - **game** a **round**, jejich schéma je uvedeno v tabulkách 7.6 a 7.7.

Každá hra má na začátku svého vytvoření serverem přiřazeno *id*. Toto *id* je předáno klientům, kteří ho poté vždy posílají s každým požadavkem

na server. slouží k jednoznačné identifikaci hry. Stav hry (*state*) bude více vysvětlen v další kapitole.

Schéma ukládá body obou hráčů (*aP1Score*, *aP2Score*), jejich sázky (*aP1Bet*, *aP2Bet*) a počet žetonů (*aP1Ticket*, *aP2Ticket*). Obtížnost hry (*difficulty*) je používána při získávání otázek z databáze. Kategorie otázek k dispozici (*availableCategories*) jsou ta témata otázek, jež ještě nebyly vybrány při sázení. Na začátku kola jsou k dispozici vždy všechny kategorie. Aktuální téma (*currentCategory*) je právě hraná kategorie v jednotlivých fázích. Témata pro sázení (*betCategories*) jsou 4 kategorie, jež byly vybrány a nabídnuty hráčům při sázení, aby z nich vybraly jednu, kterou budou hrát. Téma vsazené hráčem 1 (*aP1Categories*) jsou kategorie pro sázení, jejichž procentuální šance na vybrání byly změněny hráčem 1 při sázení.

Každé kolo má také přiřazeno *id*, kde toto *id* odpovídá hře, ke které se fáze váže. Fáze vždy ukládá počet otázek, na které jednotlivý hráči odpověděli, a jejich úspěšnost, tedy počet žetonů. Dále je ukládán počet otázek, na který musejí hráči v rámci dané fáze odpovědět, samotné otázky, na které odpovídají a jejich odpovědi.

S těmito parametry je server schopen obsluhovat a řídit hru. Zbývá vysvětlit, na jakém principu hra běží, tedy co jsou to stavy hry.

Tabulka 7.6: Struktura hry

Parametr	**Popis**
id	id hry
state	aktuální stav hry
aP1Bet	aktuální sázka hráče 1
aP2Bet	aktuální sázka hráče 2
aP1Score	aktuální skóre hráče 1
aP2Score	aktuální skóre hráče 2
aP1Ticket	aktuální počet žetonů hráče 1
aP2Ticket	aktuální počet žetonů hráče 2
difficulty	obtížnost hry
availableCategories	kategorie otázek k dispozici
currentCategory	aktuální kategorie
aP1Categories	kategorie vsazené hráčem 1
betCategories	kategorie pro sázení

Tabulka 7.7: Struktura fáze

Parametr	Popis
id	id kola / hry
aP1Q	počet zodpovězených otázek hráčem 1
aP2Q	počet zodpovězených otázek hráčem 2
aP1T	počet získaných žetonů hráčem 1
aP2T	počet získaných žetonů hráčem 2
number	počet otázek k zodpovězení
questions	jednotlivé otázky k zodpovězení
aP1Answers	odpovědi na otázky hráče 1
aP2Answers	odpovědi na otázky hráče 2

7.5.2 Stavy

Hra je řízena pomocí stavů. Jednotlivé stavy určují, co se má právě při hře dít. Tyto stavy jsou k dispozici serveru, který poté jednotlivé informace předává front-endu. Stavy se dají rozdělit na tři části – stavy fází, stavy sázení a stavy konce hry.

Stavy fází jsou ty stavy, jež řídí pokládání otázek ve fázi. Jejich parametry jsou uvedeny v tabulce 7.8. Parametr *todo* je ve fázových stavech vždy *play*, stejně jako *playingPlayer* je vždy 2 (hrají oba).

Tabulka 7.8: Stav fáze

Parametr	Popis	Příklad hodnoty
stateID	id stavu	1
desc	popis stavu	Playing - Warmup
todo	co se má při stavu dít	play
number	počet otázek	10
type	typ otázek	classic
playingPlayer	který hráč hraje v tomto stavu	2

Stavy sázení jsou ty stavy, jež řídí sázení hráčů. Pro jednotlivé sázky jsou tyto stavy vždy dva – jeden pro sázení každého hráče. Jejich parametry jsou uvedeny v tabulce 7.9. Parametr *todo* je v sázejících stavech vždy *bet*, *playingPlayer* může být 0 (sází hráč 1) nebo 1 (sází hráč 2). Typ sázení nabývá hodnot *full* u sázení na začátku kola (hráči vybírají výšku sázky) a *partial* (hráči pouze pokračují s předešlou sázkou).

Stavy konce hry uvozují konec hry a již se z nich do dalšího stavu nikdy nepřechází. Jejich parametry jsou uvedeny v tabulce 7.10. Parametr

Tabulka 7.9: Stav sázení

Parametr	Popis	Příklad hodnoty
stateID	id stavu	2
desc	popis stavu	Betting - Player 1
todo	co se má při stavu dít	bet
todoType	typ sázení	full
playingPlayer	který hráč hraje v tomto stavu	0

todo může být i *draw* v zácném případě, kdy hráči nenahráli žádné body v rozstřelu a hra skončila remízou.

Tabulka 7.10: Stav konce hry

Parametr	Popis	Příklad hodnoty
stateID	id stavu	14
desc	popis stavu	Game over - Player 1 won
todo	co se má při stavu dík	end
playingPlayer	který hráč hraje v tomto stavu	0

7.5.3 Začátek hry

Start hry vždy začíná ve front-endu, který poskytuje stránku *Game creation*, kde jsou dva formuláře - jeden pro vytvoření nové hry a druhý pro připojení do hry (viz A.5).

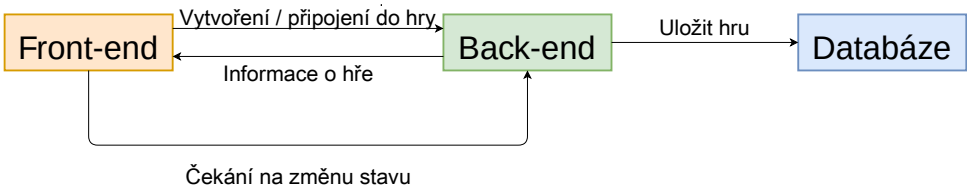
Při vytváření hry má možnost uživatel zvolit obtížnost hry. Při zaslání žádosti o novou hru je tento dotaz poslán do back-endu, jež ho dále zpracovává. Hře je přiřazeno *id*, uživatelem zvolená obtížnost a je uložena do databáze. Server potom vrací následující parametry:

- **id** – přiřazené id hry
- **player** – jaký hráč uživatel bude. Zakládající hráč vždy dostane 0 jako hráč 1.
- **desc** – popis stavu
- **stateID** – id aktuálního stavu

Na začátku hry je používaný stav vždy první stav, který se liší od ostatních stavů z předešlé kapitoly v tom, že obsahuje pouze *id* a *desc*, jelikož tento stav je pouze čekáním na dalšího hráče a tedy se nic při něm neděje.

V této chvíli hra čeká na připojení druhého hráče (viz A.6). Ten se připojí pomocí *id* hry přes zmíněný formulář připojení do hry. Na žádost o připojení mu server odpoví stejnými informacemi, které dostal hráč 1. Změnou bude, že parametr *player* bude mít hodnotu 1.

Diagram začátku hry je možno vidět na obrázku 7.2.



Obrázek 7.2: Diagram začátku hry

Stavová architektura aplikace vyžaduje od front-endu, aby periodicky kontroloval, jestli nedošlo ke změně stavu. Po vytvoření hry je v krátkém intervalu vyvolávána funkce na srovnání uloženého stavu a stavu serveru. Dokud nenastane změna, tak front-end nic nedělá. Po změně stavu si front-end vyžádá informace o novém stavu, na jejichž základě rozhodne, co bude dělat dál. V případě vytvoření hry nastane změna stavu ve chvíli, kdy se připojí druhý hráč. Tehdy server změní stávající stav na další stav v pořadí, tedy rozstřel. Na rozstřelu bude ukázán princip pokládání otázek a komunikace mezi serverem a front-endem.

7.5.4 Pokládání otázek

Při změně stavu front-end požádá o informace o hře a novém stavu.

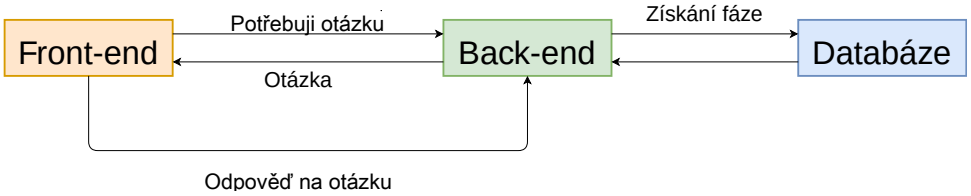
Informace o hře, jež se posílají, jsou skóre hráčů, počet žetonů a jejich sázky (viz A.13).

Informace o novém stavu, jež se posílají, jsou kromě *id* nového stavu také jeho popis (*desc*), co se má dít při stavu (*todo*) a kdo je na řadě (*playingPlayer*). V případě rozstřelu je parametr *todo* hodnoty *play* a *playingPlayer* hodnoty 2.

Po získání těchto informací je front-end připraven pokládat otázky (viz A.11). Tyto otázky získává ze serveru, který je vždy na začátku herního stavu připravuje do schématu *round*. Front-end poté požádá o číslo otázky a server mu jí ze schématu vrací. Tato otázka vždy obsahuje všechny parametry, ukázané v tabulce 7.1.

Při zodpovězení otázky uživatelem je jeho odpověď poslána serveru k vyhodnocení. Při rozstřelu, fázi 1,2 a částečně 3 není tato odpověď ukládána a místo toho jen vyhodnocena její správnost. Výjimku tvoří otázky typu N

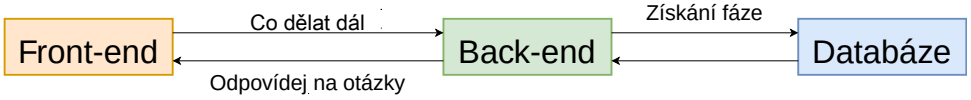
ve fázi 3 a otázky všech typů fáze 4, kdy může být jen jeden hráč správně a proto jsou jejich odpovědi ukládány před vyhodnocením. Kvůli tomu funguje poslání otázky serveru na stejném principu, jako čekání na nový stav, kdy je periodicky posílán požadavek na server o vyhodnocení otázky, dokud front-end nedostane odpověď (uvedeno na obrázku 7.3).



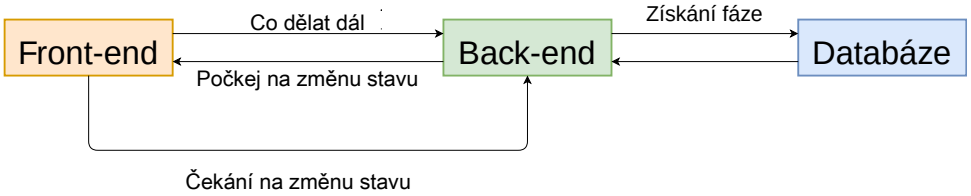
Obrázek 7.3: Diagram získání otázky

Server při vyhodnocení otázek typu C, S a W porovnává hráčovu odpověď se správnými odpověďmi. U otázek N rozhoduje vzdálenost ke správné odpovědi. Otázky pokládané ve čtvrté fázi jsou nejdříve uloženy a poté porovnávány. Nejdříve rozhoduje, který z hráčů byl správně / blíže, pokud odpověděli oba správně nebo jsou oba stejně blízko správné odpovědi u otázek typu N, rozhoduje, jak rychle odpověděli. V takovém případě získává žeton ten, kdo byl rychlejší. Pokud hráči odpověděli stejně rychle, dostávají žetony oba dva (rychlost hráčů se měří podle parametru *progress*, který postupně navyšován po určitém intervalu). Po vyhodnocení server vždy aktualizuje probíhající fázi.

Po dostání vyhodnocení otázky front-end zavolá server znovu s cílem zjistit, jaký je další postup. Server na tuto žádost odpoví dle toho, jestli hráč odpověděl na všechny otázky kola (srovnání parametrů *aP1Q* nebo *aP2Q* s *number* definovaným stavem). Pokud je třeba odpovídal na další otázku, front-end o ní požádá. Pokud je konec pokládání otázek, opět je vyvolávána periodicky funkce čekající na změnu stavu. Uvedeno na obrázcích 7.4 a 7.5).



Obrázek 7.4: Diagram zjištění dalšího postupu 1



Obrázek 7.5: Diagram zjištění dalšího postupu 2

Změna stavu nastane tehdy, až oba hráči odpoví na daný počet otázek. V případě rozstřelu je dalším stavem stav sázení před první fází.

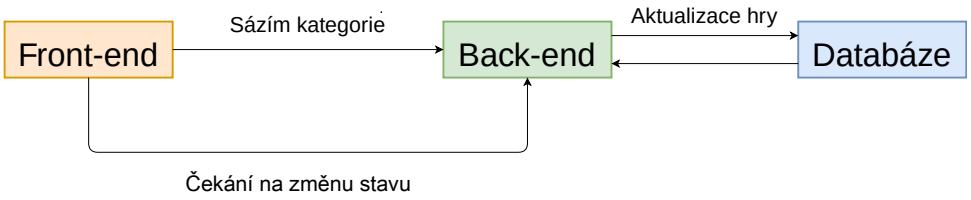
7.5.5 Sázení

Při přechodu na stav sázení server front-endu opět poskytne nové informace popsané v předchozí kapitole a přidá k tomu dva nové parametry a to *todoType*, jenž nabývá hodnot *full* nebo *partial* a *betCategories*, což jsou 4 vybrané kategorie pro sázení. Tyto kategorie jsou vybírány serverem při přechodu na nový stav. Podle parametru *playingPlayer* se rozhodne, kdo je nyní na řadě a ten sází.

Sázení je umožněno tlačítky s různými hodnotami sázky, jež se dají kombinovat (viz A.8). Hráč nemůže vsadit více, než je čtvrtina jeho celkového skóre. Místo sázení může také kolo ukončit. Pokud šlo o sázku na začátku kola, nepřijde o žádné body, v průběhu kola je již ztrácí ve prospěch druhého hráče. Po potvrzení sázky je hráč vpuštěn do výběru témat (viz A.9), kde může ovlivnit procentuální šanci na vybrání tématu v další fázi. Tyto kategorie jsou poslány serveru, jenž poté kombinuje témata obou hráčů a vybere jedno z nich, jež bude kategorií další fáze (parametr *currentCategory*). Po skončení sázky prvního hráče je stav posunut na sázení druhého hráče. Po jeho sázení server vybere téma další fáze, aktualizuje schéma **round** a daná fáze začíná s vybraným tématem otázek. Typy a počet těchto otázek definují jednotlivé stavy.

Sázení dalších fází je typu *partial* a neobsahuje část vybírání sázky, jelikož při pokračování do dalších fází je automaticky vsazena částka, jež hráč vsadil na začátku kola (viz A.10).

Sázení je ukázáno na obrázku 7.6.



Obrázek 7.6: Diagram sázení

7.5.6 Konec hry

Při ukončení sázky jedním z hráčů, na konci čtvrté fáze a také po skončení rozstřelu je vyhodnocení konce hry. Konec hry nastává tehdy, pokud jeden z hráčů již nemůže nadále sázet. Tehdy je stav změněn na stav konce hry v závislosti na tom, kdo vyhrál. Hra poté již nadále nepokračuje (viz A.12).

Ve vzácných případech, kdy ani jeden z hráčů neodpoví na žádnou otázku z rozstřelu správně, nastává remíza.

Tabulka 7.11: Počty otázek pro jednotlivé kategorie a typy

Kategorie	**Typ otázky**	**Počet**
Geography	classic	36
	statement	3
	numeric	28
	word	36
Politics	classic	36
	statement	3
	numeric	3
	word	26
History	classic	28
	statement	3
	numeric	3
	word	3
Literature	classic	7
	statement	3
	numeric	9
	word	3
Sport	classic	6
	statement	3
	numeric	3
	word	3
Movies	classic	21
	statement	3
	numeric	27
	word	21
Games	classic	8
	statement	3
	numeric	4
	word	3
	celkem	334

8 Testování

Aplikace byla testována sedmi uživateli, kteří obdrželi scénář a dotazník, jež po dokončení testování vyplnili.

8.1 Scénář

1. Přečtěte si pravidla soutěže.
2. Zaregistrujte se a přihlaste se.
3. Vytvořte hru s normální obtížností a počkejte na oponenta.
4. Postupně odpovídejte na všechny otázky.
5. Vsaďte část vašich bodů. Resetujte svojí možnost a vsaďte jinou část vašich bodů. Potvrďte.
6. Vyberte dvě kategorie a změňte jejich procentuální šance. Resetujte a udělejte to samé pro zbylé dvě kategorie. Potvrďte.
7. Po konci čtvrtého kola zjistěte, jak kolo dopadlo.

8.2 Dotazník

1. Přišla vám pravidla jasná a srozumitelná?
2. Připadá vám herní design jasný a srozumitelný? Věděli jste, kam máte psát a co máte klikat?
3. Připadali vám otázky nesrozumitelné nebo špatně zadané?
4. Připadalo vám, že máte na odpověď příliš málo času?
5. Změnili byste nějak, jaký vliv mají vaše body na sázející kategorie ?

8.3 Výsledky testování podle scénáře

1. Pravidla soutěže neměl nikdo z testerů problém najít. Všichni úspěšně navigovali menu aplikace a pravidla našli snadno.

2. Na registraci již není v menu odkaz a tak některým testerům nebylo zpočátku jasné, na jakou stránku se přepnout. Někteří si vzpomněli, že viděli odkaz na registraci na hlavní stránce, jiní se na registraci dostali přes stránku loginu. Se samotnou registrací a přihlášením problém nebyl.

3. Přes menu se testeři v pořádku dostali na správnou stránku. Nikdo neměl problém s vybráním obtížnosti a startem hry.

4. Všichni testeři bez problémů zodpověděli otázky rozstřelu.

5. Sázející část nedělala nikomu problém. Někteří si nevšimli, že jsou jejich body omezeny a divili se, že jim hra nechce dovolit sázet víc, ale tento problém byl osvětlen.

6. Sázející kategorie nedělaly nikomu problém.

7. Na konci čtvrtého kola byli někteří testeři nejistí, když je hra vrátila zpátky na začátek kola, aniž by jim cokoliv oznámila. Všichni si postupně všimli informačního panelu hry a pochopili, co jim říká. Někteří si tohoto panelu všimli až v tento okamžik.

8.4 Výsledky dotazníku

1. Testeři se shodli, že jim byla pravidla jasná a srozumitelná.

2. Herní design nedělal nikomu problémy. Někteří se vyslovili, že po stisku tlačítka *Enter* by předpokládali, že se jim odpověď na otázku uzná, bez nutnosti klikání na tlačítko *Answer*. Toto se týkalo otázek typu N a W. Také jim bylo na obtíž, že museli při zadání otázky klikat do textového pole pro psaní odpovědi.

3. Žádný tester nenahlásil, že by neporozuměl zadání otázky.

4. Většina testerů nahlásila, že na otázky neměli dostatek času, především u otázek typu N a W.

5. Drtivá většina testerů byla překvapena, jak malým počtem procent s kategoriemi hýbali. Další se divili, že jejich sázka neměla na procenta žádný vliv.

8.5 Závěr testování

Po shromáždění výsledků testování byly provedeny v aplikaci menší změny. Čas na zodpovězení otázky byl zvýšen o 3 vteřiny u všech typů otázek. Vsazené body přímo určují výši procent, jimiž hráč ovlivňuje témata fází. Tato procenta jsou nyní vypočítána jako **sázka hráče / 100 * 2**. Tato procenta nemohou jít pod 1 % a nad 100 %. Poslední malou změnou je přidání reakce na stisk klávesy *Enter* u otázek typu N a W a přidání automatického přepnutí do textového pole u některých formulářů a hlavně u odpovídání na otázky typu N a W.

8.6 Kritické zhodnocení

Byly porovnány přístupy k ukládání a zpracování semi-strukturovaných dat. Vybraný přístupem je SPARQL jazyk, který umožňuje složité dotazy. Tento jazyk, fungující na bázi Trojic, umožňoval využít při vytváření dotazů vztahy mezi daty, což by ostatní přístupy neumožnily. Použití SPARQL endpointu ušetřilo práci na vlastním Virtuoso nebo jiném serveru podporující RDF a jeho konfiguraci a naplnění daty projektu DBpedia.

Hra plně demonstruje využití dat pro vědomostní soutěž více hráčů. Testování prokázalo její funkčnost a atraktivitu, i když ve hře chybí několik původních myšlenek. Původním plánem bylo vytvoření uživatelského profilu. Hráč by měl k dispozici své statistiky, kolik zápasů hrál, vyhrál, jeho nejvyšší skóre. Dále by měl svoji úroveň, kterou by získával průběžně za hraní zápasů, které by ho odměňovaly určitými zkušenostními body. Úroveň by odemykala další možnosti profilu nebo konfiguraci her. Součástí profilu by také byl hráčův konfigurovatelný avatar. Další možností by bylo přidání funkce přátelství a chatu.

Konfigurace her by nabízela mnohem více možností. Při vytvoření hry by hráč měl možnost ovlivnit některé herní aspekty. Jedním z nich by byl počet hráčů, se kterým navržené principy počítají a rozšířit by se musela pouze implementace. Hráč by měl možnost určit, se kterými tématy se bude hrát. Toto nastavení nemá smysl při malém počtu témat. Další konfigurací by bylo nastavení časového limitu na zodpovězení otázky a na sázející část (implementováno bez limitu).

9 Závěr

V práci byly popsány formáty semi-strukturovaných dat a způsoby jejich ukládání. Bylo ukázáno, jak je možné pracovat s daty Wikipedie pomocí SPARQL dotazovacího jazyka. Tato data byla použita k vytvoření otázek a odpovědí pro vědomostní soutěž, jejíž principy práce navrhla a implementovala jako webovou aplikaci poskytující soutěž pro více hráčů.

Nápady, které se do práce nedostaly, byla správa uživatelských profilů (nejvyšší skóre, oblíbená témata, úroveň, avatar, přátelé), obtížnost hry (implementováno, ale značně omezeno vzhledem k počtu otázek) a další nastavení hry při jejím vytvoření (povolená témata, konfigurace časového limitu, větší počet hráčů). I přes tyto nenaplněné požadavky webová aplikace poskytuje funkční vědomostní soutěž pro dva hráče, což testování prokázalo. Navržená rozšíření by však dále mohly zvýšit atraktivitu hry.

Literatura

[1] *What is a Key-Value Store?* [online]. [cit. 2016/17/12]. Dostupné z: http://www.aerospike.com/what-is-a-key-value-store//.

[2] *What is CSV?* [online]. . [cit. 2016/17/12]. Dostupné z: http://super-csv.github.io/super-csv/csv_specification.html.

[3] *What is CSV file?* [online]. . [cit. 2016/17/12]. Dostupné z: https://www.csvreader.com/csv_format.php.

[4] *CSV, Comma Separated Values (RFC 4180)* [online]. . [cit. 2016/17/12]. Dostupné z: http://www.digitalpreservation.gov/formats/fdd/fdd000323.shtml.

[5] *DEFINITION semi-structured data* [online]. 2014. [cit. 2016/17/12]. Dostupné z: http://whatis.techtarget.com/definition/semi-structured-data.

[6] *DBPedia* [online]. [cit. 2017/24/3]. Dostupné z: http://wiki.dbpedia.org/.

[7] *DB-Engines Ranking* [online]. [cit. 2016/17/12]. Dostupné z: http://db-engines.com/en/ranking.

[8] *Introducing JSON* [online]. . [cit. 2016/17/12]. Dostupné z: http://www.json.org/.

[9] *An Introduction to JavaScript Object Notation (JSON) in JavaScript and .NET* [online]. . [cit. 2016/17/12]. Dostupné z: https://msdn.microsoft.com/en-us/library/bb299886.aspx.

[10] *Apache Lucene Core* [online]. [cit. 2017/17/2]. Dostupné z: https://lucene.apache.org/core/.

[11] *Neither fish nor fowl: the rise of multi-model databases* [online]. [cit. 2017/17/2]. Dostupné z: https://blogs.the451group.com/information_management/2013/02/08/neither-fish-nor-fowl/.

[12] *Binary Serialization Tour Guide* [online]. [cit. 2016/17/12]. Dostupné z: https://spin.atomicobject.com/2011/11/23/binary-serialization-tour-guide/.

[13] Beal, V. *Structured data* [online]. [cit. 2016/17/12]. Dostupné z: http://www.webopedia.com/TERM/S/structured_data.html.

[14] Beckett, D. *RDF 1.1 N-Triples* [online]. 2014. [cit. 2017/17/2]. Dostupné z: https://www.w3.org/TR/n-triples/.

[15] Brewer, E. A. *Towards Robust Distributed Systems* [online]. 2000. [cit. 2016/17/12]. Dostupné z: https://people.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf.

[16] Clinton Gormley, Z. T. *Elasticsearch: The Definitive Guide* [online]. 2014. [cit. 2017/17/2]. Dostupné z: https://www.elastic.co/guide/en/elasticsearch/guide/current/intro.html.

[17] linea, N. Semi-structured data processing with visualization support: Data parser and indexer documentation. May 2016, s. 4.

[18] Evans, E. *NOSQL 2009* [online]. 2009. [cit. 2016/17/12]. Dostupné z: http://blog.sym-link.com/2009/05/12/nosql_2009.html.

[19] Franz, I. *Interfacing with MongoDB* [online]. [cit. 2017/24/3]. Dostupné z: http://franz.com/agraph/support/documentation/current/mongo-interface.html.

[20] Grolinger, K. et al. Data management in cloud environments: NoSQL and NewSQL data stores. *Journal of Cloud Computing: Advances, Systems and Applications*. s. 41. ISSN 2192-113X. doi: 10.1186/2192-113X-2-22. Dostupné z: http://journalofcloudcomputing.springeropen.com/articles/10.1186/2192-113X-2-22.

[21] Lai, E. *No to SQL? Anti-database movement gains steam* [online]. Computerworld, 2009. [cit. 2016/17/12]. Dostupné z: http://www.computerworld.com/article/2526317/database-administration/no-to-sql--anti-database-movement-gains-steam.html.

[22] Lengstorf, J. *JSON: What It Is, How It Works, & How to Use It* [online]. [cit. 2016/17/12]. Dostupné z: https://www.copterlabs.com/json-what-it-is-how-it-works-how-to-use-it/.

[23] Malik, J. *Getting started with ElasticSearch* [online]. 2013. [cit. 2017/17/2]. Dostupné z: https://www.javacodegeeks.com/2013/04/getting-started-with-elasticsearch.html.

[24] MongoDB. *Introducing MEAN Stack* [online]. 2017. [cit. 2017/22/5]. Dostupné z: https://www.mongodb.com/blog/post/the-modern-application-stack-part-1-introducing-the-mean-stack.

[25] Moniruzzaman, A. B. M. – Hossain, S. A. NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and

Comparison. *International Journal of Database Theory and Application.* August 2013, 6, 4, s. 14. ISSN 2005-4270. Dostupné z: https://arxiv.org/abs/1307.0191.

[26] Ronk, J. *Structured, semi structured and unstructured data* [online]. 2014. [cit. 2016/17/12]. Dostupné z: https://jeremyronk.wordpress.com/2014/09/01/structured-semi-structured-and-unstructured-data/.

[27] Rouse, M. *REST (representational state transfer)* [online]. . [cit. 2017/17/2]. Dostupné z: http://searchmicroservices.techtarget.com/definition/REST-representational-state-transfer.

[28] Rouse, M. *XML (Extensible Markup Language* [online]. . [cit. 2016/17/12]. Dostupné z: http://searchsoa.techtarget.com/definition/XML.

[29] Rouse, M. *Advantages & Disadvantages Of XML* [online]. . [cit. 2016/17/12]. Dostupné z: http://www.techmynd.com/advantages-disadvantages-of-xml/.

[30] Rusher, J. *Triple Store* [online]. [cit. 2017/17/2]. Dostupné z: https://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/rusher.html.

[31] Sadalage, P. – Fowler, M. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence.* Addison-Wesley, 2013. ISBN 0-321-82662-0.

[32] Software, O. *Virtuoso SPARQL Query Editor* [online]. . [cit. 2017/24/3]. Dostupné z: http://dbpedia.org/sparql.

[33] Software, O. *Re: DBpedia: limit of triples* [online]. 2015. [cit. 2017/24/3]. Dostupné z: http://lists.w3.org/Archives/Public/public-lod/2011Aug/0028.html.

[34] Software, O. *Getting Started with DBpedia via preloaded and preconfigured Amazon EC2 AMIs for Virtuoso Cluster Edition* [online]. . [cit. 2017/24/3]. Dostupné z: https://www.openlinksw.com/dataspace/doc/dav/wiki/Main/VirtAWSDBpedia351C.

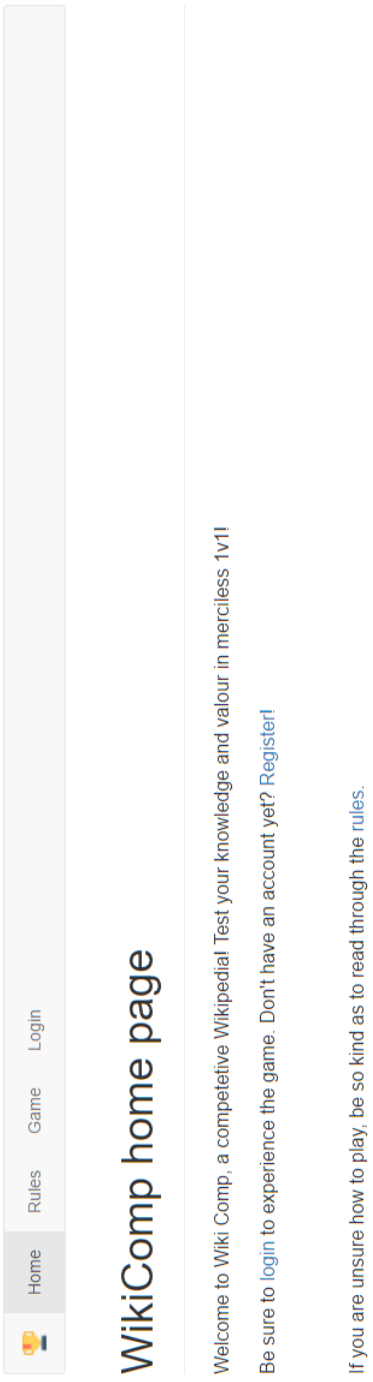
[35] Strauch, C. *NoSQL Databases* [online]. 2012. [cit. 2016/17/12]. Dostupné z: http://www.christof-strauch.de/nosqldbs.pdf.

[36] Tauberer, J. *What is RDF and what is it good for?* [online]. 2008. [cit. 2017/17/2]. Dostupné z: https://github.com/JoshData/rdfabout/blob/gh-pages/intro-to-rdf.md.

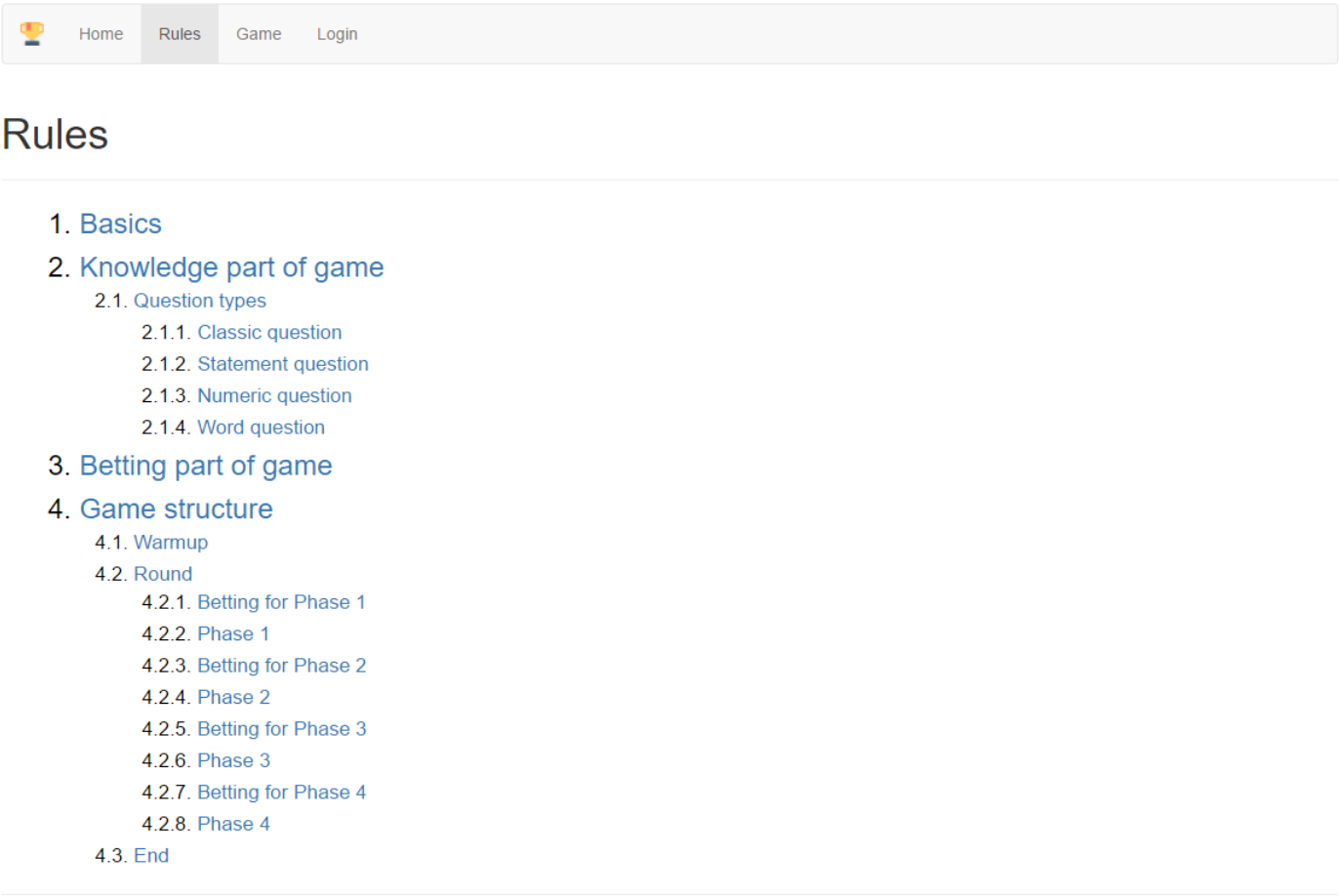
[37] Theo Haerder, A. R. Principles of transaction-oriented database recovery. *ACM Computing Surveys (CSUR)*. December 1983, 15, 4, s. 287–317. doi: 10.1145/289.291. Dostupné z: `http://dl.acm.org/citation.cfm?doid=289.291`.

[38] W3C. *SPARQL Query Language for RDFs* [online]. 2008. [cit. 2017/22/5]. Dostupné z: `https://www.w3.org/TR/rdf-sparql-query/`.

A Screenshoty



Obrázek A.1: Home



Obrázek A.2: Rules

Obrázek A.3: Login

Obrázek A.4: Register

Obrázek A.5: Game Creation

Obrázek A.6: Čekání na hru

Obrázek A.7: Rozstřel

Obrázek A.8: Sázení - vybírání sázky

Obrázek A.9: Sázení - vybírání témat

Obrázek A.10: Sázení - pokračování sázky

Obrázek A.11: Fáze hry

Obrázek A.12: Konec hry

Obrázek A.13: Panel informace

History

Index ▼	Type	Category	Statement	Answer	Result
22	classic	history	Battle of Bibracte was part of which conflict?	Gallic Wars	Your answer was right!
21	numeric	history	In which year was founded the Roman Republic?	756	Your answer was right, but your opponent was faster!
20	classic	history	Battle of Veii was part of which conflict?	First Punic War	Your answer was wrong!
19	numeric	history	In which year was founded the Roman Kingdom?	-	You didn't answer in time!
18	classic	history	Battle of Ilipa was part of which conflict?	Second Punic War	Your answer was right!
17	classic	politics	Who is the current Italian Minister of Education?	Letizia Moratti	Your answer was wrong!
16	classic	politics	What is the currency of Sweden?	Swedish krona	Your answer was right!
15	classic	politics	Who is the current Prime minister of Spain?	Mariano Rajoy	Your answer was right!
14	classic	politics	What is the currency of Turkey?	Turkish lira	Your answer was right!
13	classic	literature	Who is the author of Tarzan?	Edgar Rice Burroughs	Your answer was right!
12	classic	literature	Who is the author of Les Misérables?	Victor Hugo	Your answer was right!
11	classic	literature	Who is the author of The Silmarillion?	J. R. R. Tolkien	Your answer was right!
10	classic	movies	Who directed Ant-Man?	Peyton Reed	Your answer was right!
9	classic	history	Battle of Zama was part of which conflict?	Second Punic War	Your answer was right!

Obrázek A.14: Panel historie