PDB - projekt

Jakub Málek
Miroslava Škutová

3. 12. 2023

1 Úvod

Aplikace vytvořená v projektu je CMS systém, který slouží ke správě článků, tagů a uživatelů. Uživatelé mají možnost články číst, tvořit a hodnotit pomocí "like" nebo k nim přidat komentář. Počet komentářů pro jednotlivé články je omezený (konkrétní omezení se nastavuje pro každý článek zvlášť). Uživatelé mohou mít přiřazené regiony, a díky tomu je možné sledovat trendy článků v jednotlivých oblastech. Každý uživatel má také svůj vlastní přizpůsobený feed podle článků, které nejčastěji čte nebo lajkuje.

Systém poskytuje operace:

1. CRUD operace pro
   - Kategorie
   - Články
   - Uživatele
   - Komentáře
2. Operace vložení a smazání pro
   - Liky
   - Zobrazení
   - Regiony
3. Trendy (populární články, nejvíce přečtení a like dohromady) za určité období (hodina, den, týden, měsíc, rok)
   - V určité oblasti
   - Globálně (bez regionu)
4. Generování feedu pro uživatele (nepřečtené, seřazeno dle času a maximálně prolínajících se tagů zájmů uživatele a článku)
5. Zjištění, zda uživatel článek zobrazil
6. Zjištění, zda uživatel dal článku like
7. Počet liků článku
8. Počet komentářů článku
9. Počet zobrazení článku

2 Návrh databáze

2.1 Relační část

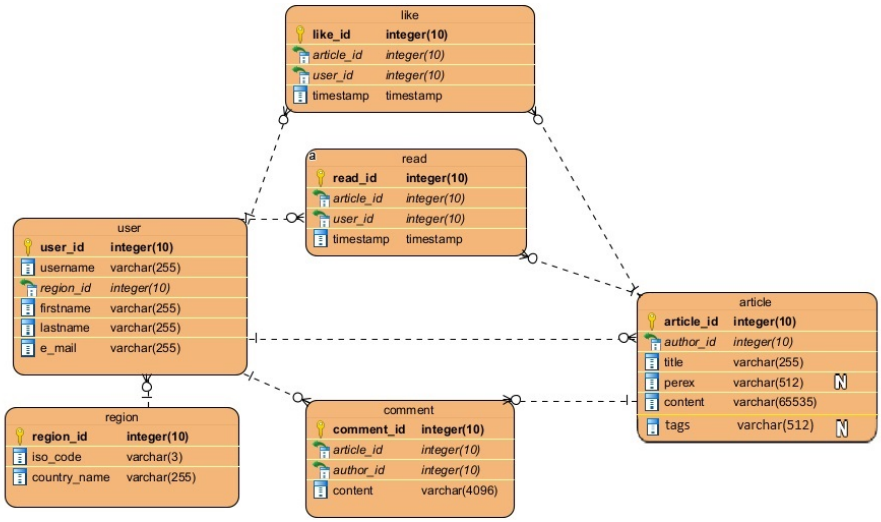

Figure 1: ERD diagram aplikace

2.2 Nerelační část

Nerelační část je optimalizovaná pro čtení. Počet lajků a přečtení je uložen přímo u článku, abychom se vyhnuli složitému dotazování u každého zobrazení článku. Interakce (čtení a like) jsou uloženy v jedné tabulce a jsou rozlišeny indexovaným polem "typ". Interakce také přímo ukládají region, ze kterého byly provedeny, aby byl jednodušší výpočet feedů a trendů.

articles

- `_id: ObjectId()`
- `author_id: ObjectId()`
- `timestamp: Date`
- `title: String`
- `perex: String`
- `content: String`
- `tags: [String]`
- `like_count: Int` // Sum z interactions type == 0
- `read_count: Int` // Sum interactions.type == 1

users

- `_id: ObjectId()`
- `username: String`
- `region_id: ObjectId()`
- `firstname: String`

- `lastname: String`
- `e_mail: String`

region

- `_id: ObjectId()`
- `iso_code: String`
- `country_name: String`

interactions

- `_id: ObjectId()`
- `timestamp: Date`
- `type: Int` // like nebo read
- `user_id: ObjectId()` // composite index user - article nebo user a article, používá se jen ve spojením s userem
- `article_id: ObjectId()`
- `region_id: ObjectId()` // Kvůli joinům (like.user.region), index
- `tags: [String]` // Joiny (article.tags)

comments

- `_id: ObjectId()`
- `timestamp: Date`
- `user_id: ObjectId()`
- `article_id: ObjectId()`
- `text: String`

2.3 Popis architektury

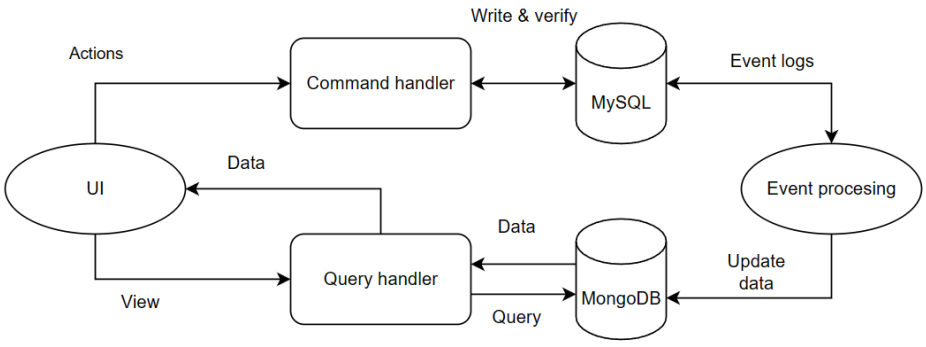

Figure 2: Architektura systému

Systém je podle CQRS principu rozdělen na dvě databáze, SQL a NoSQL. SQL v našem případě představuje source of truth, tedy databázi, ve které jsou v každém okamžiku data pravdivá. Slouží pro zápis a ověření, že danou akci můžeme provést. Akce zpracovává command handler, což je kontroler. Data jsou ukládaná v nejvíce normalizované formě.

Součástí aplikace je broker (RabbitMQ). Při operacích nad SQL databází se do brokeru vloží zpráva, která obsahuje typ operace a samotnout entitu. Na pozadí běží několik listenerů (pro každou entitu jeden), které čekají na přijetí zpráv a patřičně aktualizují NoSQL databázi, která slouží ke čtení. Zde jsou data zde uložená v jiné podobě, optimalizované pro čtení. Čtení zpracovává query handler, další kontroler(y). Tyto dotazy jsou směřovány na NoSQL databázi.

UI označuje rozhraní, které vydává příkazy. V normálním případě by to bylo uživatelské rozhraní, v rámci projektu řešíme pouze terminálovou aplikaci. Komunikace probíhá přes REST api.

3 Použité technologie

- **Python**: Backendový server je napsán v jazyce Python, což je vhodný jazyk pro vývoj webových aplikací.
- **Flask**: Framework Flask byl zvolen pro jednoduchost a flexibilitu při vytváření RESTful API.
- **Databáze**: Jako SQL zapisovací databázi jsme vybrali MySQL. Čtecí NoSQL databází je MongoDB, dokumentová databáze ukládající data ve formátu JSON (BSON). ID v této databázi je generováno kontrolerem podle entit v MySQL databázi, což umožňuje rychlý přístup a zároveň zajistí, že nikdy neztratíme ID daného dokumentu.
- **Broker** Message broker RabbitMQ slouží pro synchronizaci čtecí a zápisové databáze. Přeposílá zprávy generované při zápisu, úpravě nebo mazání entit v MySQL databázi do kontrolerů aktualizujících NoSQL databázi.

4 Architektura serveru

Architektura serveru je rozdělena do tří hlavních částí:

1. **Routování**: Tato část zahrnuje definici API cest a mapování těchto cest na odpovídající kontrolery.
2. **Kontrolery**: Kontrolery obsahují obslužné funkce, které zpracovávají požadavky klientů na základě URL cesty. Každý kontroler obsahuje metody pro různé operace, jako je čtení, zápis, aktualizace a mazání dat.
3. **Modely**: Modely představují datový model aplikace a zahrnují definice datových tříd, které se používají pro práci s daty v databázi nebo jiném úložišti.

5 Struktura adresářů

Struktura adresářů pro projekt může vypadat následovně:

- app - adresář obsahující vše potřebné k provozu BE serveru
  - src - adresář obsahující zdrojové kódy
- tests - adresář obsahující testy systému
- docker-compose.yaml - soubor popisující docker container
- README.md - Popis projektu
- postman_PDB.json - export z aplikace Postman, kde probíhalo původní zkoušení funkčnosti jednotlivých endpointů.
- doc.pdf - tento soubor s dokumentací projektu

6 Testy

Spuštění testů:
Je potřeba být v adresáři tests

```
cd tests
```

Před první spuštění je potřeba stáhnout závislosti pomocí

```
npm install
```

Samotné spuštění testů probíhá pomocí příkazu

```
npm test
```

Jednotkové testy

Možnost získání všech uživatelů / článků / regionů

- **Vstupy:**
  - Poslat požadavek na získání uživatelů / článků / regionů
- **Očekávaný výstup:**
  - Pole struktur uživatelů / článků / regionů
- **Výsledek:**
  - Endpoint na získání uživatelů / článků / regionů je funkční a vrací validní data

Vytvoření a získání uživatele / regionu a smazání

- **Vstupy:**
  - Poslat požadavek na vytvoření uživatele / regionu s správnými informacemi.
  - Poslat požadavek na získání tohoto uživatele / regionu podle ID.
- **Očekávané výstupy:**
  - Získání uživatele / regionu by mělo vrátit data, která byla při vytvoření zadána.
  - Data v odpovědi by měla odpovídat vstupům.
- **Výsledek:**
  - Uživatel / Region byl vytvořen a následně úspěšně získán a poté smazán.

Vytvoření uživatele s duplicitním username a emailem

- **Vstupy:**
  - Poslat požadavek na vytvoření uživatele s uživatelským jménem a e-mailem, které již existují v systému.
- **Očekávaný výstup:**
  - Požadavek na vytvoření uživatele by měl skončit chybou, protože username a email musí být unikátní.
- **Výsledek:**
  - Nelze vytvořit duplicitního uživatele

Vytvoření uživatele a článku a následné smazání

- **Vstupy:**
  - Poslat požadavek na vytvoření uživatele s uživatelským jménem a e-mailem a následně vytvořit článek, který bude mít autora vytvořeného uživatele.
- **Očekávaný výstup:**
  - Požadavek na vytvoření uživatele a článku by neměl selhat.
- **Výsledek:**
  - Lze vytvořit článek s přiřazeným autorem

Vytvoření článku a více komentářů než je povoleno

- **Vstupy:**
  - Poslat požadavek na vytvoření uživatele s uživatelským jménem a e-mailem a následně vytvořit článek, který bude mít autora vytvořeného uživatele. Poté poslat 11 komentářů, kde poslední komentář skončí chybou protože je nad limit počtu komentářů k jednomu článku
- **Očekávaný výstup:**
  - Požadavek na vytvoření 11. komentáře by měl selhat kvůli omezení maximálního počtu komentářů.
- **Výsledek:**
  - Lze vytvořit článek s omezeným počtem komentářů

Vytvoření uživatele, příspěvku s tagy, přiřazenými liky, a přečtením a poté získání trendů.

- **Vstupy:**
  - Poslat požadavek na vytvoření uživatelů, následně vytvořit články, které budou mít různé tagy a různá přečtení a liky od různých uživatelů. Následně zavolat získaní trendů pro určité období, které vrátí nejčtenější a nejvíce lajkované příspěvky.
- **Očekávaný výstup:**
  - Příspěvky s nevíce liky a přečtením a seřazené sestupně.
- **Výsledek:**
  - Získání trendů funguje.

7 Nasazení

```
git clone https://github.com/malekjakub69/pdb-project
cd pdb-project
docker-compose up
```

Aplikace je připravena k použití. Webserver s API běží na portu 5123 (localhost:5123), k dispozici jsou tyto endpointy:

MySQL / Zápis

- **POST** {url}/api/mysql/user - Vytvořit nového uživatele
- **DELETE** {url}/api/mysql/user/{USER_ID} - Smazat uživatele
- **POST** {url}/api/mysql/article - Vytvořit nový článek
- **DELETE** {url}/api/mysql/article/{ARTICLE_ID} - Smazat článek
- **POST** {url}/api/mysql/read - Označit článek jako přečtený
- **POST** {url}/api/mysql/comment - Přidat komentář
- **POST** {url}/api/mysql/like - Lajknout článek
- **POST** {url}/api/mysql/unlike - Odlajknout článek
- **POST** {url}/api/mysql/region - Vytvořit nový region
- **DELETE** {url}/api/mysql/region/{REGION_ID} - Smazat region

MongoDB / Čtení

- **GET** {url}/test - Testovat připojení k MongoDB
- **GET** {url}/api/user_feed/{USER_ID} - Generovat feed pro uživatele
- **GET** {url}/api/trends/{TIMEFRAME (hour — day — week — month — year)}/{?REGION_ID} - Trendy podle časového rámce (a volitelně regionu)
- **GET** {url}/api/user/{USER_ID} - Údaje o uživateli
- **GET** {url}/api/users - Kolekce uživatelů
- **GET** {url}/api/interactions/{ARTICLE_ID}/{USER_ID} - Zjistit, zda uživatel lajkl a přečetl článek
- **GET** {url}/api/interactions/like/{ARTICLE_ID}/{USER_ID} - Zjistit, zda uživatel lajkl článek
- **GET** {url}/api/interactions/read/{ARTICLE_ID}/{USER_ID} - Zjistit, zda uživatel přečetl článek
- **GET** {url}/api/article/{ARTICLE_ID} - Údaje o článku
- **GET** {url}/api/articles - Kolekce článků
- **GET** {url}/api/comments/{ARTICLE_ID} - Komentáře k článku
- **GET** {url}/api/comment/{COMMENT_ID} - Detail komentáře
- **GET** {url}/api/region/{REGION_ID} - Detail regionu
- **GET** {url}/api/regions - Kolekce regionů