5/3/2023

Endpointy knižničnej databázy

Dokumentácia zadania 5

Maroš Bednár
SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Maroš Bednár
116822

Obsah

Popis zadania

V rámci nového repozitára je potrebné implementovať API pre navrhnutú databázu zo zadania 4. Definovanie API sa nachádza v rámci yml súboru dostupného na https://github.com/FIIT-Databases/api-assignment. Tento súbor je možné otvoriť napr. https://stoplight.io. V rámci API nie je riešená žiadna autentifikácia požiadaviek.

V rámci jednotlivých endpointov je potrebné kontrolovať formát vstupov podľa definovania v API a v prípade zistenia nesprávneho formátu je potrebné vrátiť odpoveď s HTTP response code 400. V rámci implementácie je potrebné uskutočniť automatické migrácie tj. v prípade, že neexistuje štruktúra (relačná schéma) Vašej databázy v rámci servera, tak je vytvorená. Treba kontrolovať pri tom ako sa spúšťa Váš aplikačný server. Pri každom odovzdaní do testera sa Vám vytvára nová databáza na serveri.

Spustenie tejto migrácie je nevyhnutné pre možnosti overenia a vyhodnotenia funkčnosti Vášho riešenia. V prípade, že Vaše riešenie nedokáže uskutočniť túto začiatočnú migráciu, tak je Vaše riešenie považované za neakceptované.

Pre overenie funkčnosti Vášho implementovaného API je tiež nevyhnutné, aby Vám fungovali endpointy typu POST a GET. Pre možnosť otestovania zadania je možné použiť https://testerdbs.fiit.stuba.sk. V prípade metódy POST sa nachádza v rámci popisu aj ID záznamu. Toto ID záznamu sa v normálnych prípadoch negeneruje na strane klienta, ale na strane servera. V rámci zadania je ho potrebné generovať na strane klienta kvôli možnostiam testovania správnosti Vášho API. V praxi je ho nutné generovať na strane servera.

V rámci zadania 5 je možné počas implementácie používať aj **ORM**.

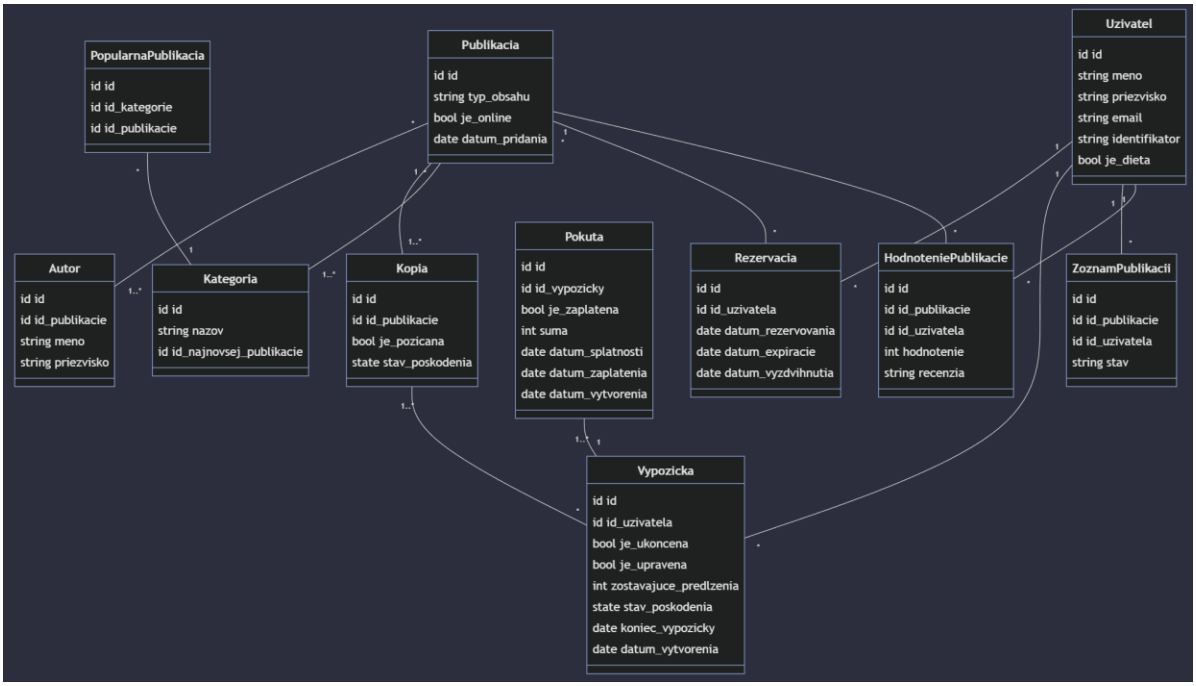
Okrem implementovanie samotného API je potrebné vyhotoviť dokumentáciu k Vášmu riešeniu, ktoré bude obsahovať:

- použité SQL dopyty pre jednotlivé endpointy s ich popisom.
- zmeny v návrhu DB oproti zadaniu 4.

Dokumentácia môže byť realizovaná ako PDF alebo markdown dokumentácia s tým, že sa bude nachádzať v AIS odovzdaní a aj v samotnom github repozitári.

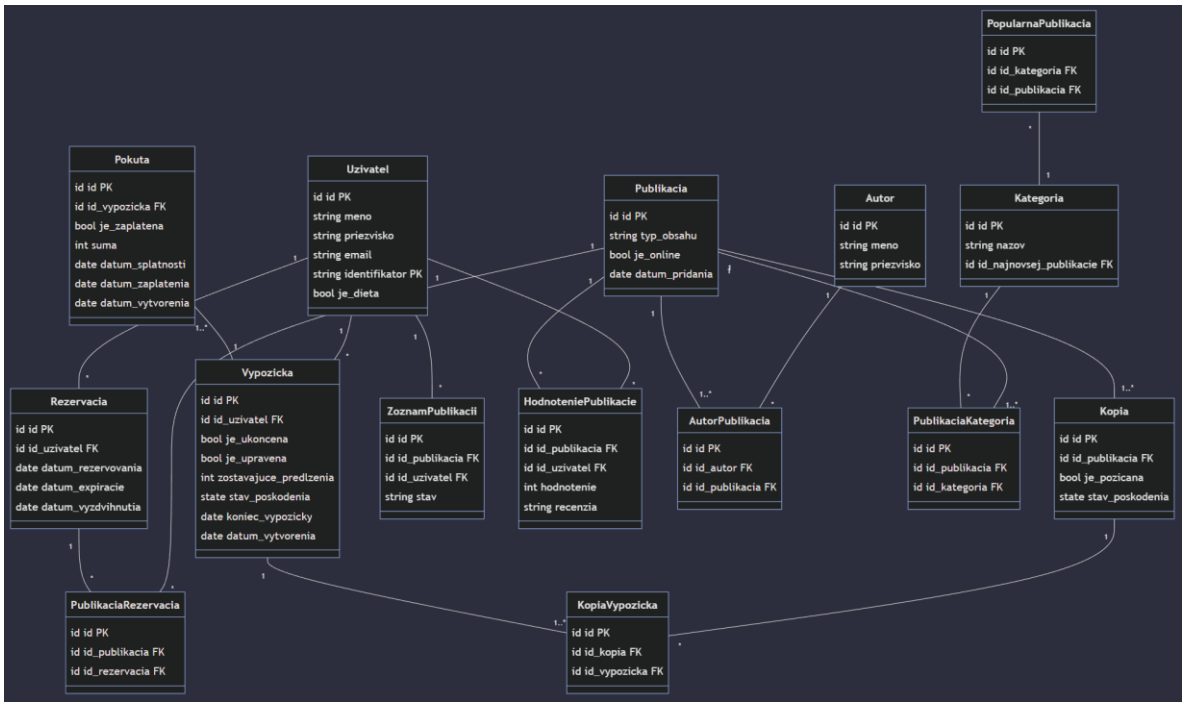
Diagramy

ER diagram



Obrázok 1 Entitno-relačný diagram

Relačný model



Obrázok 2 Relačný dátový model

Všeobecný opis kódu

Nasledujúci popis platí pre všetky použité endpointy.

V zadaní bol per priehľadnosť použitý pattern **MVC**. Vstupným bodom do ku endpointu je router.js. V tomto mieste sa aplikácia rozhodne, ktorú URL má spracovať. Následne, keď zistí, akú URL dopytujeme, pošle logiku do patričného kontroléra. **Kontrolér** je miesto, kde sa spracuje konkrétna služba „**service**“ a odpoveď z nej sa v tomto kontroléri zabalí a odošle naspäť na klienta.

Konkrétna logika služby je teda riadená v súbore končiacom „service.js“. Sem sa rozbalí celý request body, overia sa dáta, ktoré prišli z klienta a ak sú validné, spracuje samotnu logiku na databáze. Na databázu sa napojí skrze súbory končiace „DAO.js“

DAO je skratka pre Data Access Object. V tomto mieste prebieha samotný zápis do databázy. Zápis prebieha pomocou **ORM** služby poskytovanej v balíku **Knex.js**. Zápis je väčšinou jednoduchý, pretože väčšina logiky sa vykoná už v service časti a preto sa tu uskutočňujú iba operácie ako insert, delete, get a update.

Použité SQL dopyty

Users

/users/:userId

Metóda: GET

Výsledok: Vráti užívateľa.

Popis:

```
async getAuthor(id) {
    const [author] = await db('authors').where({ id });
    return author;
}
```

- Vráti jedného používateľa, podľa zadaného ID.

/users

Metóda: POST

Výsledok: Vytvorí nového používateľa.

Popis:

```
async createAuthor(name, surname, id) {
    const [author] = await db('authors')
        .insert({
            name: name,
            surname: surname,
            id: id,
        })
        .returning('*');

    return author;
}
```

- Vytvorí používateľa s parametrami name, surname a id a následne vráti jeho objekt.

/users/:userId

Metóda: PATCH

Výsledok: Updatnutý používateľ.

Popis:

```
async updateAuthor(id, data) {
    const [author] = await db('authors').where({ id }).update(data).returning('*');
    return author;
}
```

- Nájde používateľa podľa zadaného id a updatne mu určitú hodnotu, ktorú nastavíme v parametri data

/users/:userId

Metóda: DELETE

Výsledok: Zmazaný používateľ aj s jeho rezerváciami a pôžičkami.

Popis:

```
async deleteAuthor(id) {
    await db('authors').where({ id }).del();
}
```

- Zmaže používateľa podľa zadaného id a kaskádovite premaže aj rezervácie a pôžičky

Publications

/publications/:publicationId

Metóda: GET

Výsledok: Vráti publikáciu s jej autormi a kategóriami

Popis:

```
async getPublication(id) {
    const [publication] = await db('publications').where({ id });
    return publication;
}
```

```
async getAuthorsByPublicationId(id) {
    const publicationAuthors = await db('publications_authors').where({ publication_id: id });
    return publicationAuthors;
}
```

```
async getCategoriesByPublicationId(id) {
    const publicationCategories = await db('publications_categories').where({ publication_id: id });
    return publicationCategories;
}
```

- Vráti všetky údaje publikácie aj s jej autormi a publikáciami.

/publications

Metóda: POST

Výsledok: Vytvorí publikáciu s jej autormi a kategóriami

Popis:

```
async createPublication(id, title) {
    const [publication] = await db('publications')
        .insert({
            id,
            title,
        })
        .returning('*');
    return publication;
}
```

```
async createPublicationAuthor(publication_id, author_id) {
    const { v4: uuidv4 } = require('uuid');
    const id = uuidv4();

    const [publication_author] = await db('publications_authors')
        .insert({
            id,
            publication_id,
            author_id,
        })
        .returning('*');
    return publication_author;
}
```

```
async createPublicationCategory(publication_id, category_id) {
    const { v4: uuidv4 } = require('uuid');
    const id = uuidv4();

    const [publicationcategory] = await db('publications_categories')
        .insert({
            id,
            publication_id,
            category_id,
        })
        .returning('*');
    return publicationcategory;
}
```

- Vytvorí publikáciu s jej parametrami id a title
- Vytvorí spojenie medzi autormi a publikáciami
- Vytvorí spojenie medzi kategóriami a publikáciami

/publications/:publicationId

Metóda: PATCH

Výsledok: Updatne publikáciu

Popis:

```
async updatePublication(id, item) {
    const [publication] = await db('publications').where({ id }).update(item).returning('*');
    return publication;
}
```

```
async updatePublicationAuthor(id, name) {
    const [publicationAuthor] = await db('publications_authors')
        .where({ id })
        .update({
            name,
        })
        .returning('*');
    return publicationAuthor;
}
```

- Updatne dáta publikácie ako sú titul, autori alebo kategórie.

/publications/:publicationId

Metóda: DELETE

Výsledok: Zmaže publikáciu

Popis:

```
async deletePublication(id) {
    await db('publications').where({ id }).del();
}
```

```
async deletePublicationAuthors(publication_id) {
    await db('publications_authors').where({ publication_id }).del();
}
```

```
async deletePublicationCategories(publication_id) {
    await db('publications_categories').where({ publication_id }).del();
}
```

- Zmaže publikáciu aj so spojeniami autorov a kategórií

Categories

/categories/:categoryId

Metóda: GET

Výsledok: Vráti kategóriu

Popis:

```
async getCategory(id) {
    const [category] = await db('categories').where({ id });
    return category;
}
```

- Vráti kategóriu podľa zadaného id.

/categories

Metóda: POST

Výsledok: Vytvorí novú kategóriu

Popis:

```
async createCategory(id, name) {
    const [category] = await db('categories')
        .insert({
            id,
            name,
        })
        .returning('*');
    return category;
}
```

- Vytvorí novú kategóriu podľa zadaných parametrov id a name a nakoniec vráti jej objekt.

/categories/:categoryId

Metóda: PATCH

Výsledok: Updatne kategóriu

Popis:

```
async updateCategory(id, name) {
    const [category] = await db('categories')
        .where({ id })
        .update({
            name,
        })
        .returning('*');
    return category;
}
```

- Updatne meno kategórie podľa parametra name a ako výsledok vráti nový objekt.

/categories/:categoryId

Metóda: DELETE

Výsledok: Zmaže kategóriu

Popis:

```
async deleteCategory(id) {
    await db('categories').where({ id }).del();
    return;
}
```

- Zmaže kategóriu podľa zadaného id.

Cards

/cards/: cardId

Metóda: GET

Výsledok: Vráti kartu

Popis:

```
async getCard(id) {
    const [card] = await db('cards').where({ id });
    return card;
}
```

- Vráti kartu podľa hľadaného id.

/ cards

Metóda: POST

Výsledok: Vytvorí kartu

Popis:

```
async createCard(id, user_id, magstripe, status) {
    const [card] = await db('cards')
        .insert({
            id,
            user_id,
            magstripe,
            status,
        })
        .returning('*');
    return card;
}
```

- Vytvorí novú kartu podľa parametrov id, user_id, magstripe a status a ako výsledok vráti vytvorený objekt

/ cards /: cardId

Metóda: PATCH

Výsledok: Updatne kartu

Popis:

```
async updateCard(id, item) {
    const [card] = await db('cards').where({ id }).update(item).returning('*');
    return card;
}
```

- Updatne dáta karty podľa zadanej hodnoty item a ako výsledok vráti upravený objekt.

/ cards /: cardId

Metóda: DELETE

Výsledok: Zmaže kartu

Popis:

```
async deleteCard(id) {
    await db('cards').where({ id }).del();
}
```

- Zmaže kartu podľa daného id.

Instances

/instances/: instanceId

Metóda: GET

Výsledok: Vráti kópiu

Popis:

```
async getInstanceById(id) {
    const [instance] = await db('instances').where({ id });
    return instance;
}
```

- Vráti kópiu knihy podľa zadaného id.

/ instances

Metóda: POST

Výsledok: Vytvorí kópiu

Popis:

```
async createInstance(id, type, publisher, year, status, publication_id) {
    const [instance] = await db('instances')
        .insert({
            id,
            type,
            publisher,
            year,
            status,
            publication_id,
        })
        .returning('*');
    return instance;
}
```

- Vytvorí nový objekt kópie podľa parametrov id, type, publisher, year, status, publication_id.

/ instances /: instanceId

Metóda: PATCH

Výsledok: Updatne kópiu

Popis:

```
async updateInstance(id, item) {
    const [instance] = await db('instances').where({ id }).update(item).returning('*');
    return instance;
}
```

- Updatne dáta zadanej kópie podľa zadaného parametra item.

/ instances /: instanceId

Metóda: DELETE

Výsledok: Zmaže kópiu

Popis:

```
async deleteInstance(id) {
    await db('instances').where({ id }).del();
}
```

- Zmaže kópiu podľa zadaného id.

Rentals

/rentals/: rentalId

Metóda: GET

Výsledok: Vráti pôžičku

Popis:

```
async getRentalById(id) {
    const [rental] = await db('rentals').where({ id });
    return rental;
}
```

- Vráti pôžičku podľa zadaného id.

/ rentals

Metóda: POST

Výsledok: Vytvorí pôžičku

Popis:

```
async createRental(id, user_id, publication_instance_id, duration, start_date, end_date, status) {
    const [rental] = await db('rentals')
        .insert({
            id,
            user_id,
            publication_instance_id,
            duration,
            start_date,
            end_date,
            status,
        })
        .returning('*');

    return rental;
}
```

- Vytvorí novú pôžičku podľa zadaných parametrov a nakoniec vráti vytvorený objekt.

/ rentals /: rentalId

Metóda: PATCH

Výsledok: Updatne pôžičku

Popis:

```
async updateRental(id, item) {
    const [rental] = await db('rentals').where({ id }).update(item).returning('*');
    return rental;
}
```

- Updatne pôžičku kde sa zhoduje id. Updatne hodnotu, ktorá je zadaná v parametri item. Ako výsledok vráti updatnutý objekt.

/ rentals /: rentalId

Metóda: DELETE

Výsledok: Zmaže pôžičku

Popis:

```
async deleteRental(id) {
    await db('rentals').where({ id }).del();
}
```

- Zmaže pôžičku podľa zadaného id.

Reservations

/reservations/: reservationId

Metóda: GET

Výsledok: Vráti rezerváciu

Popis:

```
async getReservationById(id) {
    const [reservation] = await db('reservations').where({ id });
    return reservation;
}
```

- Vráti objekt rezervácie podľa zadaného id.

/ reservations

Metóda: POST

Výsledok: Vytvorí rezerváciu

Popis:

```
async createReservation(id, user_id, publication_id) {
    const [reservation] = await db('reservations')
        .insert({
            id,
            user_id,
            publication_id,
        })
        .returning('*');
    return reservation;
}
```

- Vytvorí novú rezerváciu podľa zadaného id, user_id a publication_id. Ako výsledok vráti nový objekt.

/ reservations /: reservationId

Metóda: PATCH

Výsledok: Updatne rezerváciu

Popis:

```
async updateReservation(id, item) {
    const [reservation] = await db('reservations').where({ id }).update(item).returning('*');
    return reservation;
}
```

- Updatne dáta rezervácie podľa zadaného id a ako výsledok vráti upravený objekt.

/ reservations /: reservationId

Metóda: DELETE

Výsledok: Zmaže rezerváciu

Popis:

```
async deleteReservation(id) {
    await db('reservations').where({ id }).del();
}
```

- Zmaže rezerváciu podľa zadaného id.

Authors

/authors/: authorId

Metóda: GET

Výsledok: Vráti autora

Popis:

```
async getAuthor(id) {
    const [author] = await db('authors').where({ id });
    return author;
}
```

- Vráti objekt autora podľa zadaného id.

/ authors

Metóda: POST

Výsledok: Vytvorí autora

Popis:

```
async createAuthor(name, surname, id) {
    const [author] = await db('authors')
        .insert({
            name: name,
            surname: surname,
            id: id,
        })
        .returning('*');

    return author;
}
```

- Vytvorí nový objekt autora podľa zadaných parametrov name, surname a id. Ako výsledok vráti nový objekt.

/ authors /: authorId

Metóda: DELETE

Výsledok: Zmaže autora

Popis:

```
async deleteAuthor(id) {
    await db('authors').where({ id }).del();
}
```

- Zmaže autora podľa zadaného id.

Zmeny v návrhu DB oproti zadaniu 4

V zadaní 5 a zadaní 4 nie sú žiadne rozdiely v implementácii. Realizácia je rovnaká ako samotný návrh.

Záver

Dokumentácia slúži ako návrh databázy, ktorú budeme v blízkej doby vytvárať. Slúži na to, aby sme si vedeli predstaviť problematiku a taktiež vedeli, čo všetko nás v ďalšom zadaní čaká. Naviac sme si precvičili tvorbu diagramov. Konkrétne Entitno-relačného diagramu, ktorý slúži na opis vzťahov medzi entitami. A relačný model, ktorý slúži na finálne vytvorenie databázy. Obsahuje presné hodnoty, ktoré budeme používať. Je rozšírený o primárne a cudzie kľúče a rozbíja vzťahy many-many na jednoduchšie.