Laboratorul 04: Conectarea cu o baza de date SQL

Pana acum, laboratoarele au avut o nota teoretica accentuata. In acest laborator insa, vom realiza un Rest API complet, care opereaza pe date adevarate, din care lipseste doar sistemul de autentificare.

Scurta introducere teoretica: Async/Await

In javascript exista doua tipuri de operatii:

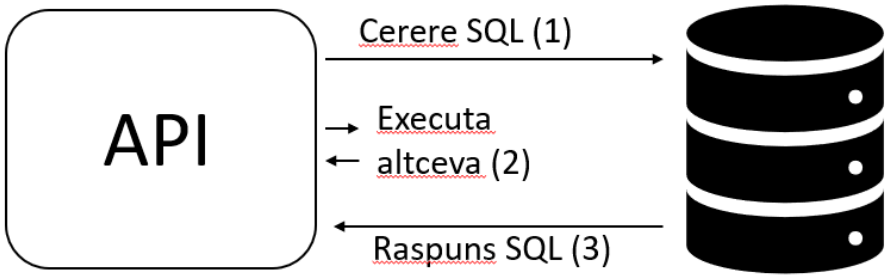
- sincrone, care se executa instant
- asincrone, care returneaza rezultatul la un moment de tip viitor

Implicit, operatiile intensive din punct de vedere computational sau I/O sunt **asincrone**, pentru a nu bloca firul de executie.

Asadar, interactiunea cu o baza de date este un exemplu de operatie asincrona.

Flow-ul poate fi dedus in felul urmator:

1. lanseaza cerere SQL (asincron)
2. executa alte operatii mult mai putin costisitoare dpdv computational
3. raspuns de la cererea SQL (in viitor)



Fara un mecanism clar de gestiune a operatiilor asincrone, codul poate deveni ilogic sau greu de scris si inteles.

O data cu introducerea **async/await** in EcmaScript 8, lucrurile s-au usurat considerabil:

- Orice functie marcata cu **async** este considerata functie asincrona si returneaza o **promisiune**
- Promisiunile pot fi asteptate folosind **await**
- Await poate fi folosit doar intr-o functie async

Asadar, in cadrul exemplului de mai sus, din moment ce lansarea cererii catre baza de date implica o promisiune, aceasta poate fi asteptata cu await, pentru ca rezultatul sa poata fi ulterior folosit.

```
const getData = async () => { return 'Ana'; }

(async() => {
    const result = getData();
    console.log(result); // Promisiune in derulare
})();

(async() => {
    const result = await getData();
    console.log(result); // Ana
})();
```

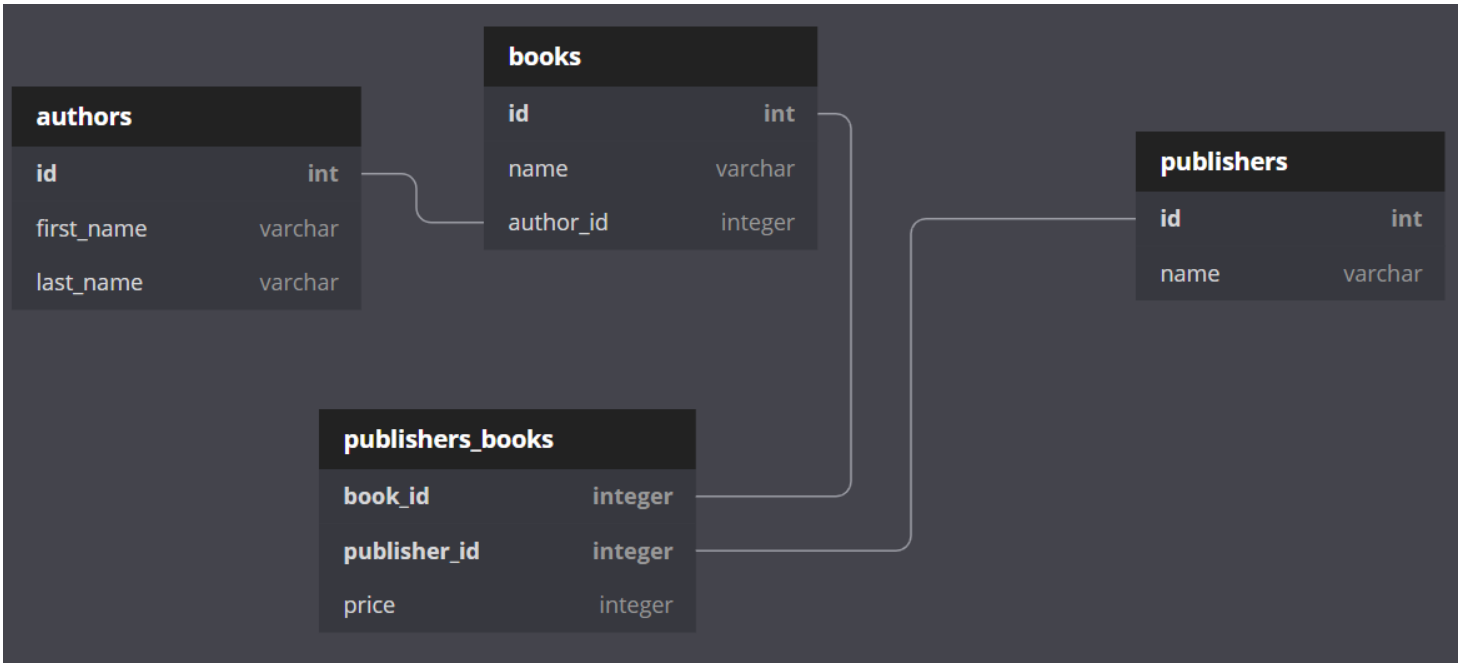
Configuratia Laboratorului

In urma realizarii laboratorului, se va obtine un REST Api ce ofera suport pentru o librarie online de carti. Api-ul se conecteaza la o baza de date PostgreSQL [https://www.postgresql.org/] ce ruleaza intr-un container de Docker [https://www.docker.com/].

Va trebui sa aveti instalat atat Docker, cat si Docker Compose [https://docs.docker.com/compose/install/]

Structura Datelor

Structura bazei de date este urmatoarea:



- Tabela **authors** retine autorii din sistem
- Tabela **books** retine cartile din sistem. Fiecare carte are cate un autor
- Tabela **publishers** retine editurile din sistem
- Tabela **publishers_books** este tabela de jonctiune pentru legaturile many-to-many dintre carti si edituri (o editura poate avea mai multe carti, o carte poate avea mai multe edituri)

Structura Api-ului

Api-ul trebuie sa ofere suport pentru manipularea datelor in sistem. Aveti deja implementat, in schelet, operatiile CRUD de baza pentru autori. Totusi, dorim sa obtinem functionalitate completa.

Pe langa rutele CRUD de baza, este nevoie si de rute **extinse**. Un exemplu de ruta extinsa este urmatorul:

- **GET /authors/:id/books** trebuie sa returneze toate cartile pentru autorul cu id-ul :id

Mediul de lucru

Baza de date si utilitarul de interactiune cu baza de date, PG Admin [https://www.pgadmin.org/], vor rula in Docker. In cadrul scheletului de laborator exista un fisier **docker-compose.yml** care face usoara pornirea acestor doua servicii.

```
version: "3.7"

services:

  pgadmin:
    image: dpage/pgadmin4
    ports:
      - "30001:80"
    environment:
      PGADMIN_DEFAULT_EMAIL: test@test.com
      PGADMIN_DEFAULT_PASSWORD: test

  db:
    image: postgres
    environment:
      POSTGRES_USER: dbuser
```

```
      POSTGRES_PASSWORD: dbpass
      POSTGRES_DB: bookstore
      TZ: Europe/Bucharest
      PGTZ: Europe/Bucharest
    ports:
      - "5432:5432"
    volumes:
      - lab4_pgdb:/var/lib/postgresql/data
      - ./init_db.sql:/docker-entrypoint-initdb.d/init_db.sql

volumes:
  lab4_pgdb:
```

Pentru a porni rularea celor doua containere, trebuie sa lansati comanda:

```
docker-compose up
```

in folderul in care se afla fisierul docker-compose.yml

Dupa pornirea celor doua containere, daca veti intra, in browser, pe **localhost:30001**, veti putea sa utilizati PGAdmin cu credentialele test@test.com si test

Scheletul de laborator

Scheletul de laborator [https://gitlab.com/tehnologiiweb/lab4] urmareste structura unui REST Api modern, bazat pe modelul arhitectural Clean. Sunt implementate urmatoarele:

- conexiunea cu baza de date - **Infrastructure/PostgreSQL/index.js**
- clasa de erori custom pentru - **WebApp/Models/ServerError.js**
- controllerele pentru autori - **WebApp/Controllers/AuthorsController.js**
- handler pentru raspuns - **WebApp/Filters/ResponseFilter.js**
- accesul la tabela Authors din baza de date - **Infrastructure/PostgreSQL/Repository/AuthorsRepository.js**
- fisierul **start.js**

Variabile de Mediu

Gestiunea variabilelor de mediu este realizata utilizand Dotenv [https://www.npmjs.com/package/dotenv] la rularea proiectului:

```
"scripts": {
    "start": "node -r dotenv/config src/start.js",
    "start-dev": "nodemon -r dotenv/config src/start.js"
  },
```

Variabilele de mediu sunt citite din fisierul **.env** atunci cand se executa comanda **npm run start**.

Gestionarea Erorilor

Gestionarea erorilor se realizeaza in maniera **centralizata**, folosind pachetul express-async-errors [https://www.npmjs.com/package/express-async-errors]. Toate erorile aruncate sunt captate automat si transmise catre **middleware-ul** creat special pentru tratarea erorilor, din start.js

```
// in cadrul start.js
app.use((err, req, res, next) => {
    if (err) {
        console.error(err);
        let status = 500;
        let message = 'Something Bad Happened';
        if (err instanceof ServerError) {
            message = err.Message;
            status = err.StatusCode;
        }
        return next(createError(status, message));
    }
});
```

Observati cum, in cadrul middleware-ului centralizat, se verifica daca eroarea primita este de tipul **ServerError**. Astfel, decidem daca eroarea este generata de noi, deci poate fi trimisa catre utilizator. Altfel, eroarea este ascunsa si se trimite un mesaj generic (e.g. Something bad happened)

Async/Await si Express

Pentru a folosi **async/await** in express, este nevoie ca middleware-urile din cadrul rutelor sa fie marcate ca functii **async**.

```
Router.get('/', async (req, res) => {

    const authors = await AuthorsDataAccess.getAllAsync();

    ResponseFilter.setResponseDetails(res, 200, authors);
});
```

Bineinteles, metoda **getAllAsync** este si ea, la randul ei, **async** si din acest motiv, ea poate fi asteptata cu **await**

Este buna practica sa sufixati toate metodele asincron cu "Async"

Exercitii

Plecand de la scheletul de laborator, implementati functionalitatile pentru Books si Publishers si ruta extinsa pentru Authors:

(4p) Pentru Books:

- GET /books → va returna toate cartile. Pentru fiecare carte se va preciza numele cartii si id-ul acesteia
- GET /books/:id → va returna id-ul si numele cartii cu id-ul :id, impreuna cu id-ul, numele si prenumele autorului si id-ul si numele editurilor
- POST /books → va returna cartea adaugata
- PUT /books/:id → va returna cartea modificata
- DELETE /books/:id → nu va returna nimic, 204

(4p) Pentru Publishers:

- GET /publishers → va returna toate editurile. Pentru fiecare editura se va preciza numele editurii si id-ul acesteia
- GET /publishers/:id → va returna id-ul si numele editurii cu id-ul :id, impreuna cu toate cartile din editura respectiva. Pentru fiecare carte se va preciza id-ul si numele cartii, impreuna cu id-ul si numele si prenumele autorului.
- POST /publishers → va returna editura adaugata
- PUT /publishers/:id → va returna editura modificata
- DELETE /publishers/:id → nu va returna nimic, 204

(1p) Pentru BooksPublishers:

- POST /books/:id/publishers → adauga la o carte un publisher si pretul asociat
- PUT /books/:bookId/publishers/:publisherId → modifica pretul asociat unei relatii book-publisher
- DELETE/books/:bookId/publishers/:publisherId → sterge pretul asociat si publisher-ul de la o carte

(1p) Pentru Authors

- GET /authors/:id/books → va returna id-ul si numele cartilor pentru autorul cu id-ul :id, impreuna cu id-ul si numele editurii/editurilor pentru fiecare carte