Aleksandra Duda	Nr albumu: 130510
Data Science (grupa 1)	20.03.2022r.

SPRAWOZDANIE Z PROJEKTU 1

Metody wykorzystane w REST

Metody jakimi posłużono się przy budowaniu architektury REST to

Metoda HTTP	Opis
GET	Pobiera istniejące zasoby
POST	Tworzy nowy zasób
PUT	Aktualizuje w całości istniejący zasób
PATCH	Częściowo aktualizuje istniejący zasób
DELETE	Usuwa zasób

Należało zbudować odpowiednie endpoint'y – zasada ogólna jest taka, że używamy liczby mnogiej od danej encji do każdego z nich.

HTTP method	API endpoint	Description
GET	/users	Pobierz listę użytkowników
GET	/users/<id>	Pobierz jednego użytkownika
POST	/users	Stwórz nowego użytkownika
PATCH	/books/rent/:id	Częściowo uaktualnij tabelę z wypożyczeniami
DELETE	/users/<id>	Usuń użytkownika

W metodzie GET pobierającej jednego użytkownika <id> jest paramtrem, który pozwala nam sprawdzić użytkownika o podanym id.

Analogicznie działa to w przypadku pozostałych zaimplementowanych endpointów w projekcie, które zaimplementowano względem instrukcji od Prowadzącego zajęcia.

Nagłówki

Nagłówki (ang. headers) to dodatkowe źródła informacji do każdego odwoływania się do API. Prezentują meta dane związane z żądaniami i odpowiedziami do API.

Mamy

- **Accept –** tę informację musi zawrzeć klient w nagłówku, gdy jest wysyłane zapytanie do serwera. Klient informuje jakie dane jest w stanie odebrać od serwera. Zapewnia to mechanizm ochronny, aby serwer nie wysłał danych, które nie zostaną zrozumiane po stronie klienta
- **Content – type** – tę informację musi zawrzeć serwer w nagłówku odpowiedzi, kiedy są wysyłane dane do klienta. Serwer przez to oznajmia klientowi jakie dane wysyła w body.

Kody odpowiedzi na zapytanie

Aby komunikacja w REST była łatwiejsza i bardziej efektywna warto implementować kody odpowiedzi na zapytania. W przypadku wykonywania zapytań dostaniemy odpowiedni kod i opis, co da nam informację o tym chociażby, czy daną operację udało się zrealizować, czy też nie. Dodatkowy opis kodu pomoże w ewentualnej szybkiej identyfikacji, jeśli jakaś operacja się nie udała.

Ogólny opis tego jak działają poszczególne kody w projekcie:

Kod	Znaczenie
200	Zapytanie HTTP zakończone sukcesem
201	Zasób został stworzony z sukcesem
401	Zasób nie istnieje lub jest niepoprawny
400	Zapytanie nie może być zrealizowane przez użycie złego syntaxu, poprzez zbyt duży rozmiar albo inny błąd ze strony klienta.
404	Zasobu nie znaleziono

Napisano o nich ogólnie, ale działają w zależności od róznych endpointów analogicznie.

Możemy też tworzyć własne kody odpowiedzi, natomiast należy pamiętać, żeby trzymać się przy tworzeniu kodów określonych reguł, żeby to było w pełni przydatne i efektywne.

Oprócz kodów odpowiedzi, które mieliśmy zaimplementować w ramach zajęć, które zostały ujęte w *rest-biblioteka-swagger.yaml* dodano kilka dodatkowych.

W momencie kiedy ktoś chciałby zwrócić książkę, która już została zwrócona (return_date nie jest NULL) nadpisałoby to rzeczwysitą datę zwrócenia (tę pierwszą) na aktualną datę.

```
  returned_book = cur.execute("select return_date from tbl_rentals where
bookid_fk=(?) AND userid_fk=(?) and return_date is not
null",(id,user,)).fetchall()
    if returned_book:
```

```
        return app.response_class(response = 'Książka została już zwrócona',
                                  status=410,
                                  mimetype='application/json')
```

Obsłużono też dodatkową sytuację kiedy użytkownik ma wypożyczone egzemplarze, ale akurat nie ma akurat tego konkretneto który ktoś usiłuje zwrócić (np. pracownik systemu mógłby pomylić i próbować zwracać książkę osobie, która jej nawet nie wypożyczyła):

```
else:
        #if user has rented books check if he/she rented book with this id
        rented_books = cur.execute("select * from tbl_rentals where
bookid_fk=(?) and userid_fk=(?)",(id,user,)).fetchall()
        if not rented_books:
            return app.response_class(response = 'Użytkownik nie ma aktualnie
wypożyczego tego konkretnego egzemplarzu',
                                      status=409,
                                      mimetype='application/json')
```

Baza danych

Manipulacja danymi była możliwa dzięki uprzedniemu skorzystaniu z bazy danych, którą można było stworzyć na podstawie pliku dołączonego przez Prowadzącego, po drobnych zmianach w syntaksie. Skorzystano w pythonie z modułu sqlite w celu zarządzania bazą.

Aby tworzyć, przeszukiwać, czy usuwać zasoby w bazie danych przy pomocy endpointów należało zastosować zapytania do bazy danych.

- **Select** – wyświetla żądane dane z bazy danych

  SELECT * FROM tbl_books;

- **Delete –** usuwanie konkretnych danych do bazy danych

  DELETE FROM *tbl_books* WHERE *id=(?)*;

- **Insert –** wstawianie nowych danych do bazy danych

  INSERT INTO *tbl_users* (name)
  VALUES (*?*);

- **Update –** aktualizacja konkretnych danych w bazie danych
  UPDATE *tbl_books* SET quantity = quantity+1 WHERE id=(?);

Dla pozostałych encji zapytania prezentują się analogicznie.

Po każdym wypożyczeniu, czy zwróceniu książki również zadbano o to, aby aktualizowała się liczba dostępnych książek.

Przykład przy wypożyczniu:

```
update_book = cur.execute("update tbl_books set quantity=quantity-1 Where
id=?",(id,)).fetchall()
```