Wprowadzenie teoretyczne

- Endpoint
- Kody http
- JSON
- Request Body / Params
- JWT Token

Flask Server

Instalacja potrzebnych paczek

Do zainstalowania wszystkich potrzebnych paczek użyjemy pipenv, który powinien zainstalować wszystkie dependencje.

```
py -m pipenv install
```

Schema (schema.sql)

- W pliku zdefiniowana jest prosta schema bazy danych, która pomoże nam w utrzymywaniu danych

Init Database (init_db.py)

W celu stworzenia bazy danych należy odpalić komendę:

```
py -m init_db.py
```

lub

```
python init_db.py
```

Jeżeli wszystko się powiodło powinniśmy zobaczyć plik **database.db**.

Dodatkowo możemy otworzyć bazę przy użyciu rozszerzenia Database Client.

- Otwieramy **Database Client**
- Wybieramy **SQLite**
- Szukamy ścieżki do pliku **database.db** i klikamy **Connect**
- Otwieramy tabelke **Posts**

Serwer (app.py)

Włączenie serwera:

```
py -m flask run
```

lub

```
flask run
```

Wywoływanie zapytań do serwera

Postman

- Pobierz program dostępny pod linkiem
- Zaimportuj plik **lab_04.postman_collection.json**
- Poeksperymentuj z zapytaniami i przeanalizuj odpowiedzi

Rest Client

- Pobierz rozszerzenie VS Code Rest Client
- Otworz plik **collection.http**
- Poeksperymentuj z zapytaniami i przeanalizuj odpowiedzi
- Porównaj z programem **Postman**

Debugowanie

VS Python Extension

- Dodaj plik **launch.json** do folderu **.vscode**
- Zainstaluj rozszerzenia do pythona
  - https://marketplace.visualstudio.com/items?itemName=ms-python.python
  - https://marketplace.visualstudio.com/items?itemName=ms-python.debugpy
- Otwórz sekcję **Run and Debug**
- Kliknij przycisk **Run**
- Dodaj **breakpoint**

Zadania [10 pkt]

Zadanie – Mikroserwisy w Node.js (Express + ORM)

Celem zajęć jest stworzenie trzech niezależnych serwisów (mikroserwisów) w **Node.js** Express, komunikujących się przez REST API. Przykładowa domena: **księgarnia internetowa**.

Każdy serwis powinien być uruchamiany na osobnym porcie (np. 3001, 3002, 3003).

Do obsługi bazy danych należy użyć ORM (np. **Sequelize**) oraz prostą bazę danych (np. **SQLite**).

Serwis 1: Obsługa książek (2.5 pkt)

Serwis przechowuje informacje o książkach dostępnych w księgarni.

Struktura tabeli `books`:

- `id` (PK, auto-increment)
- `title` (nazwa książki)
- `author`
- `year` (rok wydania)

Endpointy:

Metoda	Endpoint	Opis
GET	`/api/books`	Zwraca listę wszystkich książek
GET	`/api/books/:bookId`	Zwraca dane konkretnej książki
POST	`/api/books`	Dodaje nową książkę (`title`, `author`, `year`) i zwraca jej `id`
DELETE	`/api/books/:bookId`	Usuwa książkę o podanym `id`

Serwis 2: Obsługa zamówień (2.5 pkt)

Serwis odpowiada za tworzenie i zarządzanie zamówieniami użytkowników.

Struktura tabeli `orders`:

- `id`
- `userId`
- `bookId`
- `quantity`

Endpointy:

Metoda	Endpoint	Opis
GET	`/api/orders/:userId`	Zwraca listę zamówień użytkownika
POST	`/api/orders`	Dodaje zamówienie (`userId`, `bookId`, `quantity`) i zwraca `id` zamówienia. **Sprawdź, czy `bookId` istnieje** (należy wykorzystać serwis 1. Nie wykonywać bezpośrednio

Metoda	Endpoint	Opis
		zapytania do bazy!).
DELETE	`/api/orders/:orderId`	Usuwa zamówienie
PATCH	`/api/orders/:orderId`	Aktualizuje wybrane dane zamówienia (np. ilość)

Serwis 3: Obsługa użytkowników (2 pkt)

Serwis zarządza rejestracją i logowaniem użytkowników.

Struktura tabeli `users`:

- `id`
- `email`
- `password` (hasło przechowywać zaszyfrowane np. `bcrypt`)

Endpointy:

Metoda	Endpoint	Opis
POST	`/api/register`	Rejestracja nowego użytkownika (`email`, `password`). Zwraca `id` użytkownika.
POST	`/api/login`	Logowanie użytkownika (email + password). Zwraca **JWT token**.

Uwaga:
Endpointy modyfikujące dane (POST, DELETE, PATCH) w serwisach **Książki** i **Zamówienia** muszą wymagać poprawnego **JWT** (autoryzacja).

Dodatkowe wymagania

Zadanie	Punkty
Zabezpieczenie endpointów (JWT)	1 pkt
Kolekcja endpointów w Postman / HTTP Client	1 pkt
Użycie ORM (np. Sequelize)	1 pkt