|  |  |  |  |
| --- | --- | --- | --- |
|  | Politechnika Bydgoska im. J. J. Śniadeckich  Wydział Telekomunikacji,  Informatyki i Elektrotechniki  **Zakład Systemów Teleinformatycznych** | |  |
| **Przedmiot** | Skryptowe języki programowania | | |
| **Prowadzący** | mgr inż. Martyna Tarczewska | | |
| **Temat** | FastAPI | | |
| **Student** | Paweł Jońca | | |
| **Nr lab.** | 9, 3/4 | **Data wykonania** | 13.01.2025r |
| **Ocena** |  | **Data oddania spr.** | 13.01.2025r |

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

app = FastAPI()  
  
*# Definicja klasy Mountains  
class* Mountains(BaseModel):  
 name: str  
 height: float *# Wysokość w metrach* location: str *# Lokalizacja* mountain\_id: int | *None* = *None  
  
# Prowizoryczna "baza danych" dla obiektów Mountains*database: List[Mountains] = []

Zad 1

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

*# Endpoint: Zwraca wszystkie elementy z bazy danych*@app.get("/mountains")  
*def* get\_all\_mountains() -> List[Mountains]:  
 *return* database  
  
*# Endpoint: Zwraca element o wybranym ID*@app.get("/mountains/{mountain\_id}")  
*def* get\_mountain(mountain\_id: int) -> Mountains:  
 mountains = [x *for* x *in* database *if* x.mountain\_id == mountain\_id]  
 *if not* mountains:  
 *raise* HTTPException(status\_code=404, detail="Mountain not found")  
 *return* mountains[0]

Zad 2

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

*# Dodaje element do bazy danych*@app.post("/mountains/{mountain\_id}")  
*def* add\_mountain(mountain\_id: int, mountain: Mountains) -> Mountains:  
 mountain.mountain\_id = mountain\_id  
 database.append(mountain)  
 *return* mountain

Zad 3

Obraz zawierający tekst, zrzut ekranu, Czcionka, oprogramowanie

Opis wygenerowany automatycznie

*# Endpoint: Aktualizuje element o podanym ID*@app.put("/mountains/{mountain\_id}")  
*def* update\_mountain(mountain\_id: int, mountain: Mountains) -> Mountains:  
 *# Szukaj góry w bazie danych  
 for* i, existing\_mountain *in* enumerate(database):  
 *if* existing\_mountain.mountain\_id == mountain\_id:  
 *# Zaktualizuj dane góry* updated\_mountain = mountain.copy(update={"mountain\_id": mountain\_id})  
 database[i] = updated\_mountain  
 *return* updated\_mountain  
 *# Jeśli góra nie istnieje, zwróć błąd 404  
 raise* HTTPException(status\_code=404, detail="Mountain not found")

Zad 4

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

*# Endpoint: Usuwa element o podanym ID*@app.delete("/mountains")  
*def* delete\_mountain(mountain\_id: int) -> dict:  
 *for* i, existing\_mountain *in* enumerate(database):  
 *if* existing\_mountain.mountain\_id == mountain\_id:  
 deleted\_mountain = database.pop(i)  
 *return* {"message": "Mountain deleted successfully", "deleted\_mountain": deleted\_mountain}  
 *raise* HTTPException(status\_code=404, detail="Mountain not found")

Zad 5

Podgląd na wszystkie dodane endpointy w Swagger UI.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Obraz zawierający zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne, tekst

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Strona internetowa

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Ikona komputerowa

Opis wygenerowany automatycznie

Zad 6.

Stosowanie funkcji asynchronicznych w FastAPI jest dobrą praktyką

* Zmniejsza zużycie zasobów.
* Poprawia skalowalność i wydajność.
* Umożliwia bardziej responsywne działanie aplikacji.
* Wspiera nowoczesne podejście do programowania backendowego.

Dzięki tym zaletom, funkcje async są szczególnie zalecane w aplikacjach wymagających obsługi dużej liczby zapytań lub operacji I/O.

Wnioski

W trakcie ćwiczenia utworzyłem obiekt przy użyciu biblioteki Pydantic, co ułatwiło zarządzanie danymi w aplikacji zbudowanej w FastAPI. Dodałem różne endpointy (GET, POST, PUT, DELETE), co pozwoliło na realizację operacji CRUD i efektywną pracę z listą danych. Dzięki zastosowaniu funkcji asynchronicznych aplikacja działa wydajniej, szczególnie podczas obsługi wielu zapytań jednocześnie.

\*Zadanie dodatkowe\*

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

*from* typing *import* Optional  
*from* pydantic *import* BaseModel, Field  
  
*class* Item(BaseModel):  
 name: str = Field(..., title="Name of the item", description="The name of the item being created", max\_length=50)  
 description: Optional[str] = Field(*None*, title="Description of the item", description="A detailed description of the item", max\_length=300)  
 price: float = Field(..., title="Price of the item", description="The price must be a positive value", gt=0)  
 tax: Optional[float] = Field(*None*, title="Tax applied to the item", description="Optional tax percentage for the item", ge=0)  
  
*# Przykładowe użycie w FastAPI  
from* fastapi *import* FastAPI  
  
app = FastAPI()  
  
@app.post("/items/")  
*async def* create\_item(item: Item):  
 *return* item

**Wyjaśnienie pól:**

1. **name**
   * **Typ**: str
   * **Opis**: Pole wymagane (... oznacza brak wartości domyślnej), które reprezentuje nazwę tworzonego przedmiotu.
   * **Dodatkowe informacje**:
     + title: "Name of the item" – krótki tytuł widoczny w dokumentacji.
     + description: "The name of the item being created" – szczegółowy opis pola.
     + max\_length: 50 – maksymalna długość tekstu.
2. **description**
   * **Typ**: Optional[str] (pole opcjonalne, może być None).
   * **Opis**: Szczegółowy opis przedmiotu, który użytkownik może dodać.
   * **Dodatkowe informacje**:
     + title: "Description of the item" – krótki tytuł w dokumentacji.
     + description: "A detailed description of the item" – szczegółowy opis pola.
     + max\_length: 300 – maksymalna długość tekstu.
3. **price**
   * **Typ**: float
   * **Opis**: Pole wymagane, które reprezentuje cenę przedmiotu. Musi być wartością dodatnią.
   * **Dodatkowe informacje**:
     + title: "Price of the item" – krótki tytuł w dokumentacji.
     + description: "The price must be a positive value" – szczegółowy opis pola.
     + gt: 0 – wartość musi być większa niż 0.
4. **tax**
   * **Typ**: Optional[float] (pole opcjonalne, może być None).
   * **Opis**: Wartość podatku, która może być zastosowana do przedmiotu. Jeśli nie podano, domyślnie None.
   * **Dodatkowe informacje**:
     + title: "Tax applied to the item" – krótki tytuł w dokumentacji.
     + description: "Optional tax percentage for the item" – szczegółowy opis pola.
     + ge: 0 – wartość musi być większa lub równa 0.

Cały kod do zadań 1-4

*from* typing *import* List, Optional  
*from* fastapi *import* FastAPI, HTTPException, Query  
*from* pydantic *import* BaseModel  
*import* uvicorn  
  
app = FastAPI()  
  
*# Definicja klasy Mountains  
class* Mountains(BaseModel):  
 name: str  
 height: float *# Wysokość w metrach* location: str *# Lokalizacja* mountain\_id: Optional[int] = *None  
  
# Prowizoryczna "baza danych" dla obiektów Mountains*database: List[Mountains] = []  
  
*# Endpoint: Zwraca wszystkie elementy z bazy danych*@app.get("/mountains")  
*def* get\_all\_mountains() -> List[Mountains]:  
 *return* database  
  
*# Endpoint: Zwraca element o wybranym ID*@app.get("/mountains/{mountain\_id}")  
*def* get\_mountain(mountain\_id: int) -> Mountains:  
 mountains = [x *for* x *in* database *if* x.mountain\_id == mountain\_id]  
 *if not* mountains:  
 *raise* HTTPException(status\_code=404, detail="Mountain not found")  
 *return* mountains[0]  
  
*# Dodaje element do bazy danych*@app.post("/mountains")  
*def* add\_mountain(mountain: Mountains) -> Mountains:  
 new\_id = len(database) + 1 *# Proste generowanie ID* mountain.mountain\_id = new\_id  
 database.append(mountain)  
 *return* mountain  
  
*# Endpoint: Aktualizuje element o podanym ID*@app.put("/mountains/{mountain\_id}")  
*def* update\_mountain(mountain\_id: int, mountain: Mountains) -> Mountains:  
 *for* i, existing\_mountain *in* enumerate(database):  
 *if* existing\_mountain.mountain\_id == mountain\_id:  
 updated\_mountain = mountain.copy(update={"mountain\_id": mountain\_id})  
 database[i] = updated\_mountain  
 *return* updated\_mountain  
 *raise* HTTPException(status\_code=404, detail="Mountain not found")  
  
*# Endpoint: Usuwa element o podanym ID*@app.delete("/mountains")  
*def* delete\_mountain(mountain\_id: int) -> dict:  
 *for* i, existing\_mountain *in* enumerate(database):  
 *if* existing\_mountain.mountain\_id == mountain\_id:  
 deleted\_mountain = database.pop(i)  
 *return* {"message": "Mountain deleted successfully", "deleted\_mountain": deleted\_mountain}  
 *raise* HTTPException(status\_code=404, detail="Mountain not found")  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 uvicorn.run("main:app", host="127.0.0.1", port=8000, reload=*True*)