Python średnio zaawansowany

Dzień 3

Blok nr 2:

Akwizycja danych

AGENDA

- API
  - WebAPI
  - REST
- Typy błędów dla protokołu HTTP

API

Application **P**rogramming **I**nterface

API to zestaw ściśle określonych reguł w jaki programy komunikują się ze sobą.

W świecie rzeczywistym odpowiednikiem API może być pilot do telewizora lub pokrętło zamontowane w pralce służące do wyboru programu. Obydwa te przedmioty dostarczają ściśle określony **interfejs** (przyciski), które umożliwiają **komunikację** z urządzeniem.

Application **P**rogramming **I**nterface

API to zestaw:

- procedur (reguł),
- struktur danych,
- protokołów,
- usług,

określający w jaki sposób komponenty (a nawet systemy) powinny wchodzić we wzajemne interakcje.

Dobry interfejs API ułatwia tworzenie programów, zapewniając wszystkie elementy składowe.

Application **P**rogramming **I**nterface

Istnieje wiele różnych typów interfejsów API dla:

- systemów operacyjnych (dostęp do sprzętu, usług np. do schowka, uprawnienia, harmonogramy)
- usług sieciowych (stos IP)
- usług / aplikacji, np. baz danych
- aplikacji webowych

Ważne:

API ogranicza dostęp do pełnej funkcjonalności i szczegółów implementacyjnych systemów do dostępu przez wybrane interfejsy.

WebAPI

WebAPI

WebAPI – rodzaj implementacji API, wykorzystywany w komunikacji między klientem (najczęściej to przeglądarka) a serwerem webowym.

Serwer udostępnia usługi klientom przy wykorzystaniu protokołu **HTTP**, klienci są **konsumentami** tych usług.

WebAPI to określenie ogólne na komunikację przy pomocy HTTP, nie jest stylem architektury oprogramowania.

WebAPI

Usługi, udostępnione przez aplikacje webowe działające na serwerach, są identyfikowane przy pomocy URL (Uniform Resource Locator).

Prosty URL:

https://jsonplaceholder.typicode.com/posts

- protokół
- host
- ścieżka do zasobu

WebAPI

Złożony URL:

https://jan:kowalski@jsonplaceholder.typicode.com:8080/posts?id=10

- protokół (http, https)
- login
- hasło
- host (jednoznaczne wskazanie na serwer)
- port (80, 8080, 443, 8443)
- ścieżka do zasobu
- ścieżka wyszukiwania

WebAPI

- `GET /posts`

  lista zasobów spełniających żądanie

- `GET /posts/1`

  pojedynczy element spełniający żądanie, identyfikator jednoznacznie określa zasób

- `GET /posts?userId=1`

  wyszukiwanie zasobu spełniającego warunek atrybut = wartość

WebAPI

- `POST /posts`

  utworzenie nowego zasobu, dane przesyłane w ciele zapytania

- `PUT /posts/1`

  edycja pojedynczego elementu spełniającego żądanie, identyfikator jednoznacznie określa zasób

- `DELETE /posts/1`

  żądanie usunięcia pojedynczego elementu, identyfikator jednoznacznie określa zasób

Moduły Python

Do obsługi WebAPI będziemy korzystali z modułów:

- Requests: HTTP for Humans™ - obsługa żądań HTTP
- urllib – przetwarzanie URL

Dodatkowe moduły pomocnicze:

- logging – każda aplikacja potrzebuje rejestrować zdarzenia
- json – wsparcie dla JSONów

Praktyka: http_methods_examples/

REST

REST

Początkowo, zasoby w sieci były definiowane ogólnie, tylko jako dokumenty lub pliki identyfikowalne poprzez ich adresy URL.

Obecnie liczba dostępnych zasobów jest ogromna i nie ogranicza się tylko do tych, które dominowały na początku podejścia opartego o dostęp do zasobów zdalnych.

REST

REST (Representational State Transfer) – styl architektury oprogramowania, w którym każdy zasób powinien być dostępny przez URI.

URI – pojęcie szersze niż **URL**, nadrzędne do **URL; URL** jest przypadkiem **URI**

W świecie serwisów **REST**ful, zapytania o zasób pod żądanym URI zwracają odpowiedź w jednym z formatów: HTML, XML, JSON lub Plain Text.

REST

Odpowiedź może potwierdzać zmianę dokonaną na zasobie i zapewniać linki do innych powiązanych zasobów lub kolekcji. Przy wykorzystaniu HTTP dostępne operacje to GET, POST, PUT, DELETE, oraz pozostałe z grupy CRUD HTTP methods.

REST pozwala na złożoną interakcję pomiędzy serwisami bez ingerencji człowieka.

Praktyka: process_response.py

REST – praktyka

Powszechną praktyką jest ograniczanie dostępu do API:

- Ilość żądań/jednostka czasu
- tylko uprawnione podmioty (uwierzytelnienie -> autoryzacja -> konsumpcja)

Praktyka: headers_example.py

Wybrane kody błędów HTTP

Wybrane kody błędów HTTP

- Nieprawidłowe zapytanie: **400**
- Błąd uwierzytelnienia (nie dostarczono lub nieodpowiednie): **401**
- Błąd autoryzacji (brak uprawnień): **403**
- Niedozwolona metoda HTTP: **405**
- Przekroczenie czasu odpowiedzi: **408**
- "Jestem czajnikiem": **418**
- Zbyt duża liczba przekierowań: **429**
- Usługa niedostępna: **503**

Praktyka: exceptions_example.py

Dzięki!