



WARSZTATY TESTOWANIE API CZĘŚĆ I



» Architekt / tester automatyzujący / trener

» 8+ lat doświadczenia w automatyzacji

» mtadla@sii.pl



WIOLETA
BUSINESS DEVELOPMENT
MANAGER



1

Aktualne stanowisko i doświadczenie
w testowaniu API oraz automatyzacji

2

Poziom programowania obiektowego Java

3

Znajomość REST / Postman / RestAssured

4

Czego oczekujesz od szkolenia?

Przerwy



Agenda

Moduł 1 – Podstawy REST

1. JSON
2. Backend vs frontend
3. Omówienie zasad komunikacji protokołu http
4. Czym jest REST?
5. Wprowadzenie pojęć związanych z REST – request, response, endpoint, resource, status code, header, parameters, body
6. Wprowadzenie do metod HTTP – get, post, put, patch, delete
7. Wprowadzenie do aplikacji Postman
8. Tworzenie zapytań z użyciem aplikacji Postman
9. Automatyzacja scenariusza end 2 end z użyciem Postman
10. Przygotowanie do modułu 2



PAULINA
RESOURCE MANAGER

Moduł 2 – Automatyzacja z RestAssured

1. Czym jest narzędzie RestAssured
2. Omówienie struktury given() when() then() w RestAssured
3. Przekazywanie parametrów i uzupełnianie headerów
4. Wysyłanie requestów GET, POST, DELETE, PATCH
5. Wypełnianie request body, request/response Specification
6. Wbudowanie logowanie w RestAssured
7. Walidacja response: statusCode, extract()
8. Automatyzacja testów API
9. Budowanie frameworka testów API
10. Dobre praktyki w automatyzacji testów API

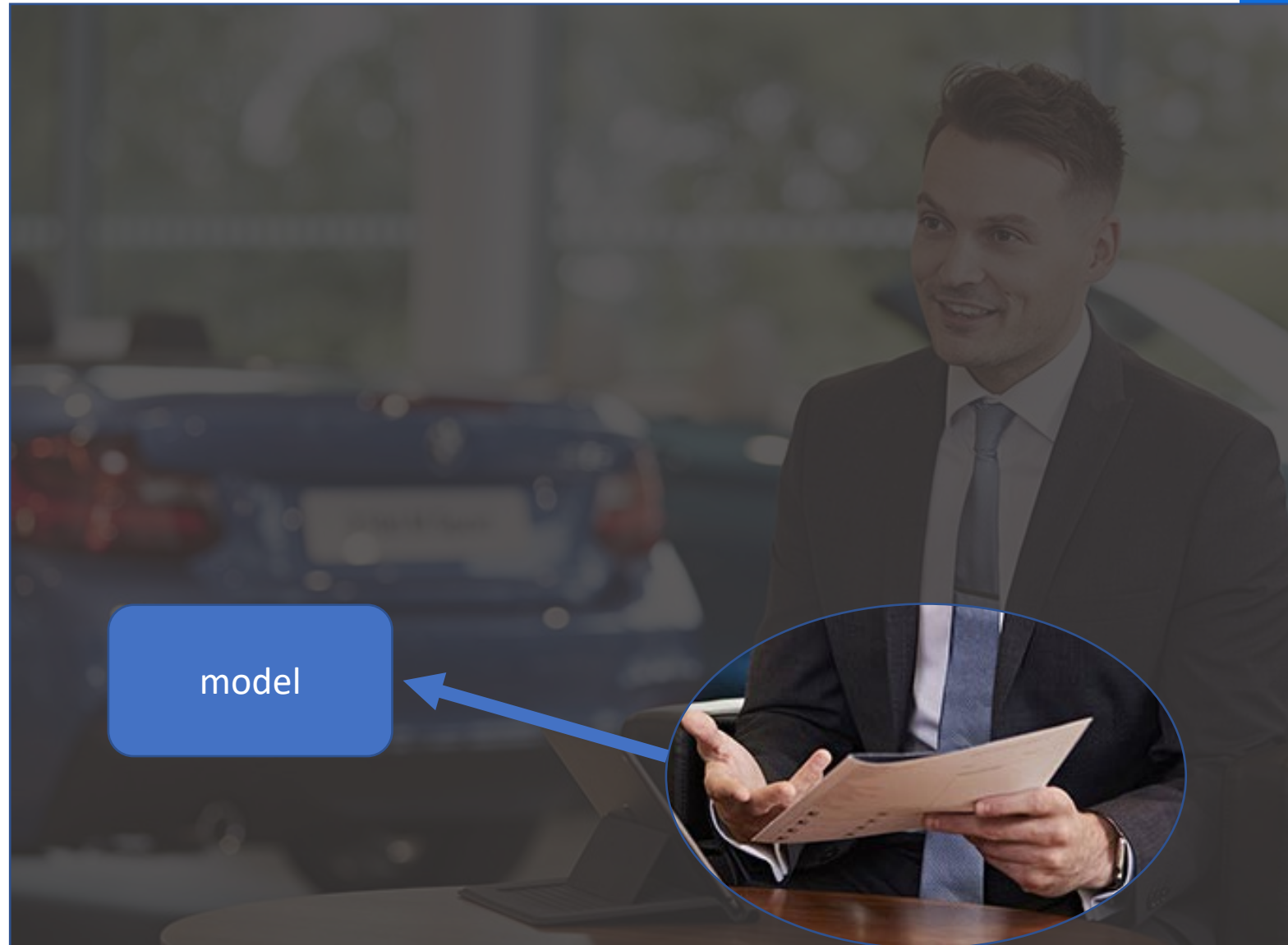


PAULINA
RESOURCE MANAGER

Klasa vs obiekt













Obiekt – fizyczna instancja klasy

Jedna klasa -> wiele obiektów



Obiekt – fizyczna instancja klasy

Jedna klasa -> wiele obiektów



VIN: QWEASDZXC123



VIN: ERTDFGCVB345



VIN: WERSDFXCV234



JSON

JavaScript Object Notation

- Lekki format wymiany danych
- Może przechowywać wyłącznie dane - nie ma funkcji ani metod
- Brak komentarzy
- Niezależny od żadnego języka
- Gdzie stosować?
- Jak wymawiać?

JSON

```
{  
  "firstName": "Mateusz",  
  "lastName": "Tadla"  
}
```

JS object

```
{  
  firstName: "Mateusz",  
  lastName: "Tadla"  
}
```

JSON vs obiekt w JavaScript



JSON – obiekt {}


```
{  
  "firstName": "Mateusz",  
  "lastName": "Tadla"  
}
```

```
{  
  "firstName": "Mateusz",  
  "lastName": "Tadla",  
  "age": 32
```

```
}
```

```
{  
  "firstName": "Mateusz",  
  "lastName": "Tadla",  
  "age": 32,  
  "isAutomationTester": true  
}
```

```
{  
  "firstName": "Mateusz",  
  "lastName": "Tadla",  
  "age": 32,  
  "isAutomationTester": true,  
  "address": {  
    "city": "New York",  
    "street": " Wall Street"  
  }  
}
```



```
{  
  "firstName": "Mateusz",  
  "lastName": "Tadla",  
  "age": 32,  
  "isAutomationTester": true,  
  "address": {  
    "city": "New York",  
    "street": " Wall Street"  
  },  
  "hobbies": ["Gym", "Squash"]  
}
```

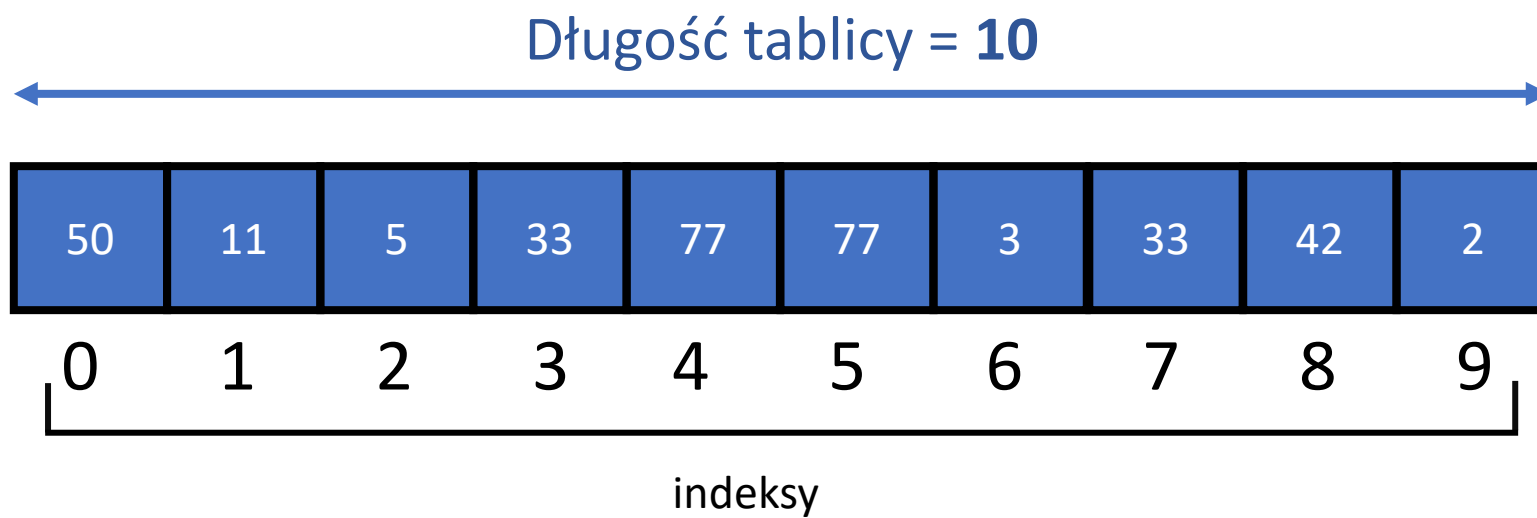
```
{  
  "firstName": "Mateusz",  
  "lastName": "Tadla",  
  "age": 32,  
  "isAutomationTester": true,  
  "address": {  
    "city": "New York",  
    "street": " Wall Street"  
  },  
  "hobbies": ["Gym", "Squash"],  
  "kids": null  
}
```



JSON – tablica []

```
[  
  {  
    "firstName": "Mateusz",  
    "lastName": "Tadla"  
  },  
  {  
    "firstName": "Adam",  
    "lastName": "Nowak"  
  },  
  {  
    "firstName": "Jan",  
    "lastName": "Kowalski"  
  }  
]
```

```
{
  "firstName": "Jan",
  "lastName": "Kowalski",
  "parents": [
    {
      "firstName": "Adam",
      "lastName": "Kowalski"
    },
    {
      "firstName": "Maria",
      "lastName": "Kowalska"
    }
  ]
}
```



Demo!

Wymagane dane w JSONie:

1. `title` (*tekst*)
2. `author` (*obiekt z `firstName` i `lastName`*)
3. `pages` (*liczba*)
4. `isAvaiaibleInPolish` (*wartość logiczna*)
5. `mainCharacterNames` (*tablica imion*)
6. `website` (*null jeżeli brak*)

Wymagane dane w JSONie:

1. **title** (*tekst*)
2. **director** (*obiekt z firstName i lastName*)
3. **genre** (*tekst*)
4. **year** (*liczba*)
5. **actors** (*lista 3 obiektów – każdy firstName i lastName*)

Wymagane dane w JSONie:

1. **movies** - *tablica obiektów, gdzie każdy obiekt powinien zawierać:*

1. **title** (*tekst*)

2. **director** (*obiekt z firstName i lastName*)

3. **genre** (*tekst*)

4. **year** (*liczba*)

5. **actors** (*tablica 3 obiektów – każdy firstName i lastName*)

JSONPath

- JSONPath to ~język zapytań, który w przypadku pracy z dużymi i złożonymi strukturami JSON, pozwala na dostęp do określonych fragmentów danych.
- Np. mamy 10 000 filmów w pliku JSON i chcemy wyszukać w JSONie wszystkie tytuły filmów
- JSONPath Online Evaluator:

<https://jsonpath.com/>



```
{  
  "store": {  
    "name": "Bookworld",  
    "book": [  
      {  
        "title": "The Doll",  
        "author": "Prus",  
        "price": 8.99  
      },  
      {  
        "title": "Hobbit",  
        "author": "Tolkien",  
        "price": 5.99  
      }  
    ]  
  }  
}
```

JSONPath:
\$

```
{  
  "store": {  
    "name": "Bookworld",  
    "book": [  
      {  
        "title": "The Doll",  
        "author": "Prus",  
        "price": 8.99  
      },  
      {  
        "title": "Hobbit",  
        "author": "Tolkien",  
        "price": 5.99  
      }  
    ]  
  }  
}
```

JSONPath:
\$.store

```
{  
  "store": {  
    "name": "Bookworld",  
    "book": [  
      {  
        "title": "The Doll",  
        "author": "Prus",  
        "price": 8.99  
      },  
      {  
        "title": "Hobbit",  
        "author": "Tolkien",  
        "price": 5.99  
      }  
    ]  
  }  
}
```

JSONPath:
`$.store.name`

```
{  
  "store": {  
    "name": "Bookworld",  
    "book": [  
      {  
        "title": "The Doll",  
        "author": "Prus",  
        "price": 8.99  
      },  
      {  
        "title": "Hobbit",  
        "author": "Tolkien",  
        "price": 5.99  
      }  
    ]  
  }  
}
```

JSONPath:
\$.store.book

```
{  
  "store": {  
    "name": "Bookworld",  
    "book": [  
      {  
        "title": "The Doll",  
        "author": "Prus",  
        "price": 8.99  
      },  
      {  
        "title": "Hobbit",  
        "author": "Tolkien",  
        "price": 5.99  
      }  
    ]  
  }  
}
```


JSONPath:
`$.store.book[0].title`

```
{  
  "store": {  
    "name": "Bookworld",  
    "book": [  
      {  
        "title": "The Doll",  
        "author": "Prus",  
        "price": 8.99  
      },  
      {  
        "title": "Hobbit",  
        "author": "Tolkien",  
        "price": 5.99  
      }  
    ]  
  }  
}
```

JSONPath:
`$.store.book[0]`

```
{  
  "store": {  
    "name": "Bookworld",  
    "book": [  
      {  
        "title": "The Doll",  
        "author": "Prus",  
        "price": 8.99  
      },  
      {  
        "title": "Hobbit",  
        "author": "Tolkien",  
        "price": 5.99  
      }  
    ]  
  }  
}
```

JSONPath:
`$.store.book[*].title`

```
{  
  "store": {  
    "name": "Bookworld",  
    "book": [  
      {  
        "title": "The Doll",  
        "author": "Prus",  
        "price": 8.99  
      },  
      {  
        "title": "Hobbit",  
        "author": "Tolkien",  
        "price": 5.99  
      }  
    ]  
  }  
}
```

JSONPath:
`$.store.book[?(@.price < 6)]`

```
{  
  "store": {  
    "name": "Bookworld",  
    "book": [  
      {  
        "title": "The Doll",  
        "author": "Prus",  
        "price": 8.99  
      },  
      {  
        "title": "Hobbit",  
        "author": "Tolkien",  
        "price": 5.99  
      }  
    ]  
  }  
}
```

JSONPath:

`$.store.book[?(@.author == "Tolkien")]`

```
{
  "store": {
    "name": "Bookworld",
    "book": [
      {
        "title": "The Doll",
        "author": "Prus",
        "price": 8.99
      },
      {
        "title": "Hobbit",
        "author": "Tolkien",
        "price": 5.99
      }
    ]
  }
}
```

JSONPath:

`$.store.book[?(@.author == "Tolkien" && @.price < 6)]`

```
{  
  "store": {  
    "name": "Bookworld",  
    "book": [  
      {  
        "title": "The Doll",  
        "author": "Prus",  
        "price": 8.99  
      },  
      {  
        "title": "Hobbit",  
        "author": "Tolkien",  
        "price": 5.99  
      }  
    ]  
  }  
}
```


DEMO

1. Jakie jest miasto, w którym znajduje się biblioteka?.
2. Pobierz wszystkie informacje o adresie biblioteki
3. Pobierz wszystkie tytuły książek dostępnych w bibliotece
4. Które książki zostały wypożyczone?
5. Pobierz wszystkie informacje o książkach napisanych przez J.R.R. Tolkiena
6. Podaj tytuły książek, które zostały napisane przed 1900 rokiem
7. Pobierz książki, które są z gatunku "Fantasy" i nie zostały wypożyczone
8. Pobierz rok publikacji książki o tytule "The Hobbit"



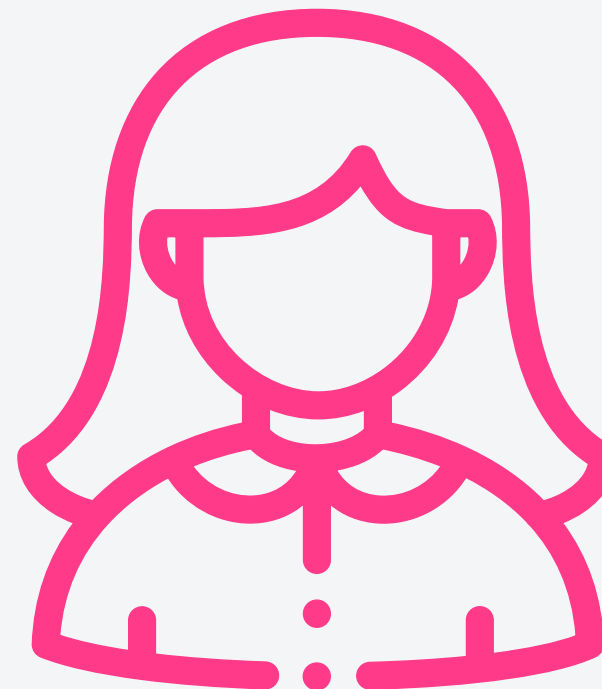
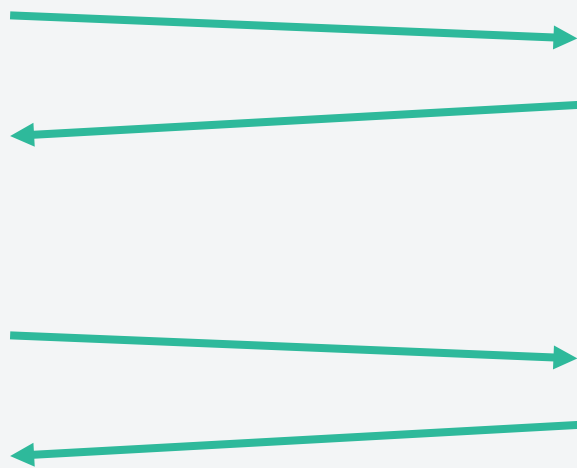
SII BYDGOSZCZ TEAM

Czym jest protokół?

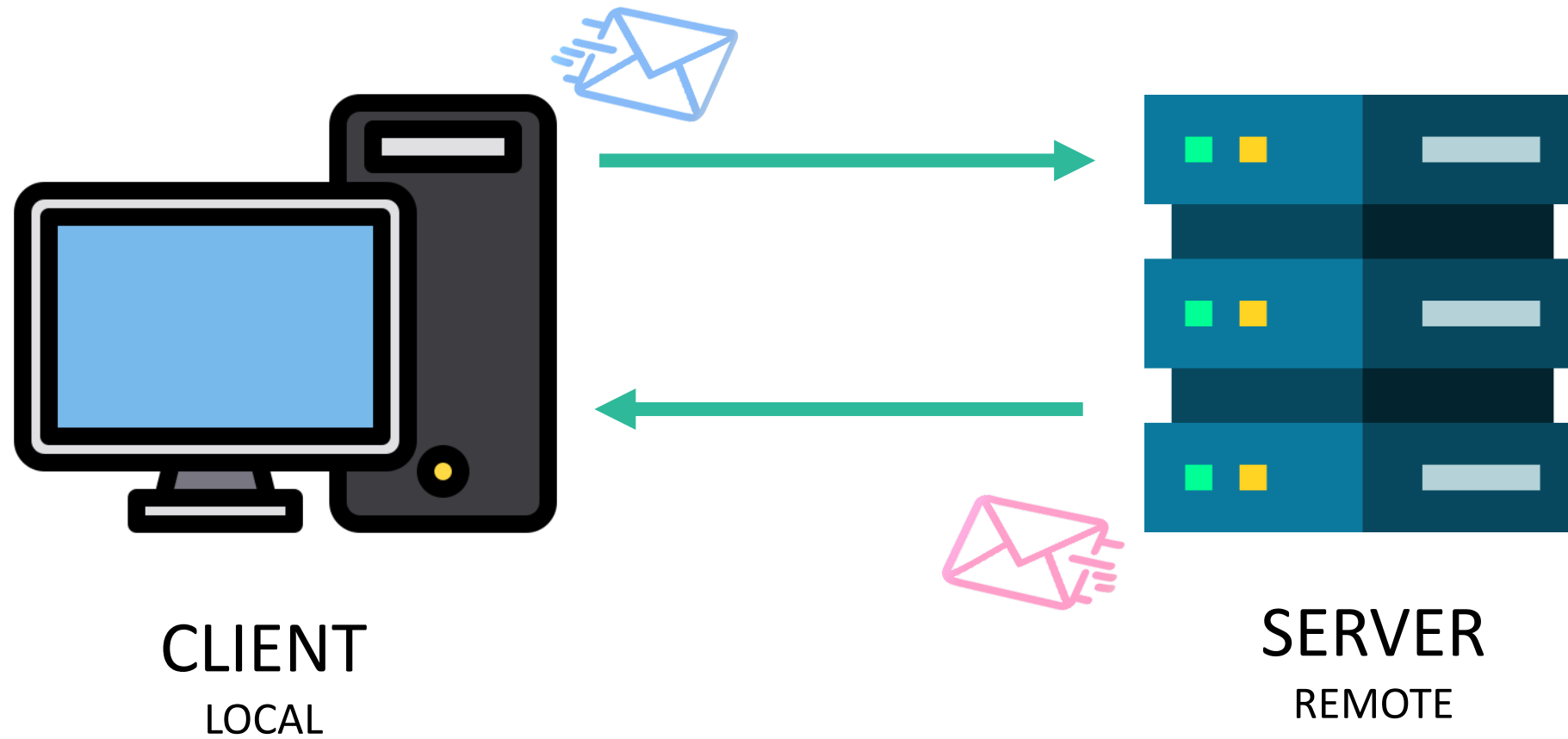


Czym jest protokół?









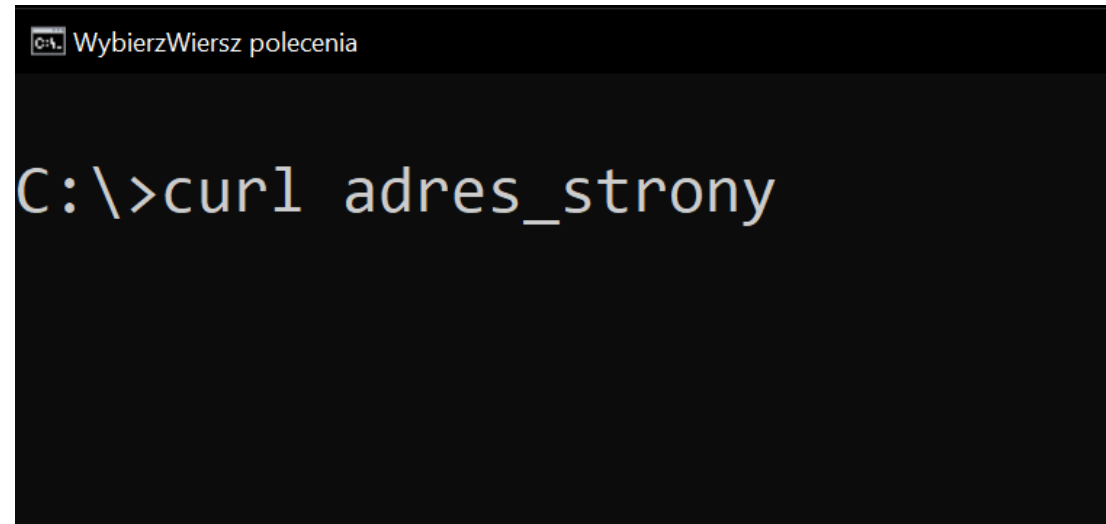


1. Bezpołączeniowość
2. Bezstanowość
3. Niezależność ze względu na typy danych

Pierwsze zapytanie do serwera

- Narzędzie Open Source umożliwiające wykonywanie odwołań do serwerów z poziomu cmd
- Wspiera szeroką gamę protokołów (w tym HTTP)
- Jest wbudowany w Windows 10 (od kompilacji 17063) i znajduje się w C:\Windows\System32





```
C:\>curl adres_strony
```

Pierwsze odwołanie do serwera przy pomocy cURL

Składnia	Objaśnienie
<code>curl <adres_url></code>	Wyświetlenie odpowiedzi serwera
<code>curl -I <adres_url></code>	Wyświetlenie samych nagłówków z odpowiedzi
<code>curl -i <adres_url></code>	Wyświetlenie odpowiedzi serwera razem z nagłówkami
<code>curl -L <adres_url></code>	Podążanie za przekierowaniami
<code>curl -u login:pass <adres_url></code>	Przekazanie danych do uwierzytelniania
<code>curl -X POST <adres_url></code>	Zmiana domyślnej metody 'GET' na inną (np. POST)
<code>curl -d "op1=wart1&op2=wart2,, <adres_url></code>	Przekazywanie parametrów do zapytania

Składnia	Objaśnienie
<code>curl <adres_url></code>	Wyświetlenie odpowiedzi serwera
<code>curl -I <adres_url></code>	Wyświetlenie samych nagłówków z odpowiedzi
<code>curl -i <adres_url></code>	Wyświetlenie odpowiedzi serwera razem z nagłówkami
<code>curl -L <adres_url></code>	Podążanie za przekierowaniami
<code>curl -u login:pass <adres_url></code>	Przekazanie danych do uwierzytelniania
<code>curl -X POST <adres_url></code>	Zmiana domyślnej metody 'GET' na inną (np. POST)
<code>curl -d "op1=wart1&op2=wart2,, <adres_url></code>	Przekazywanie parametrów do zapytania

Za pomocą polecenia: **cURL -I <adres_strony>**

1. Wykonaj zapytanie do adresu:
 - **http://wp.pl**
 - *Oczekiwany status: 301*
2. Następnie wykonaj ponownie polecenie do adresu, który przyszedł w nagłówku **Location**
 - Oczekiwany status: 200

Za pomocą polecenia: **cURL -i <adres_strony>**

1. Wykonaj zapytanie do adresu:

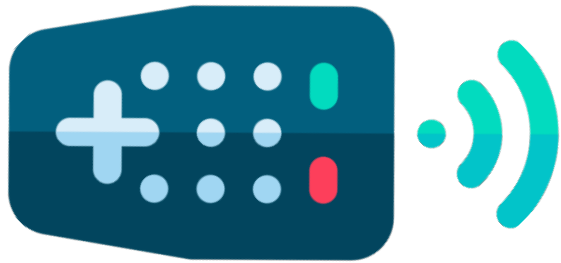
- **example.com**
- *Oczekiwany status: 200*



SII BYDGOSZCZ TEAM

Czym jest interfejs?









UI

User Interface

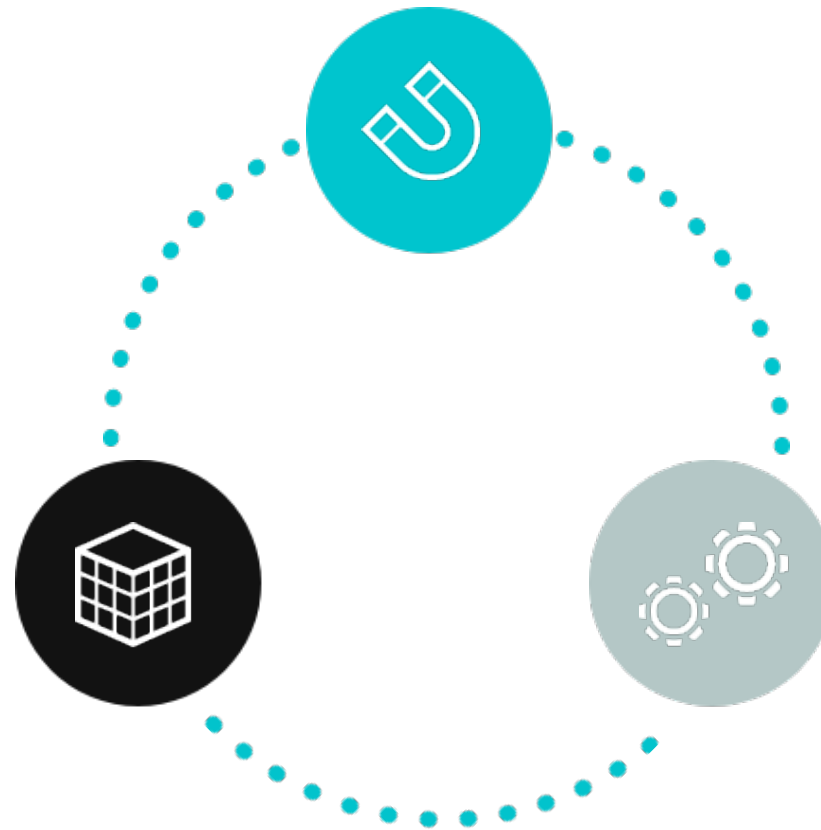






API

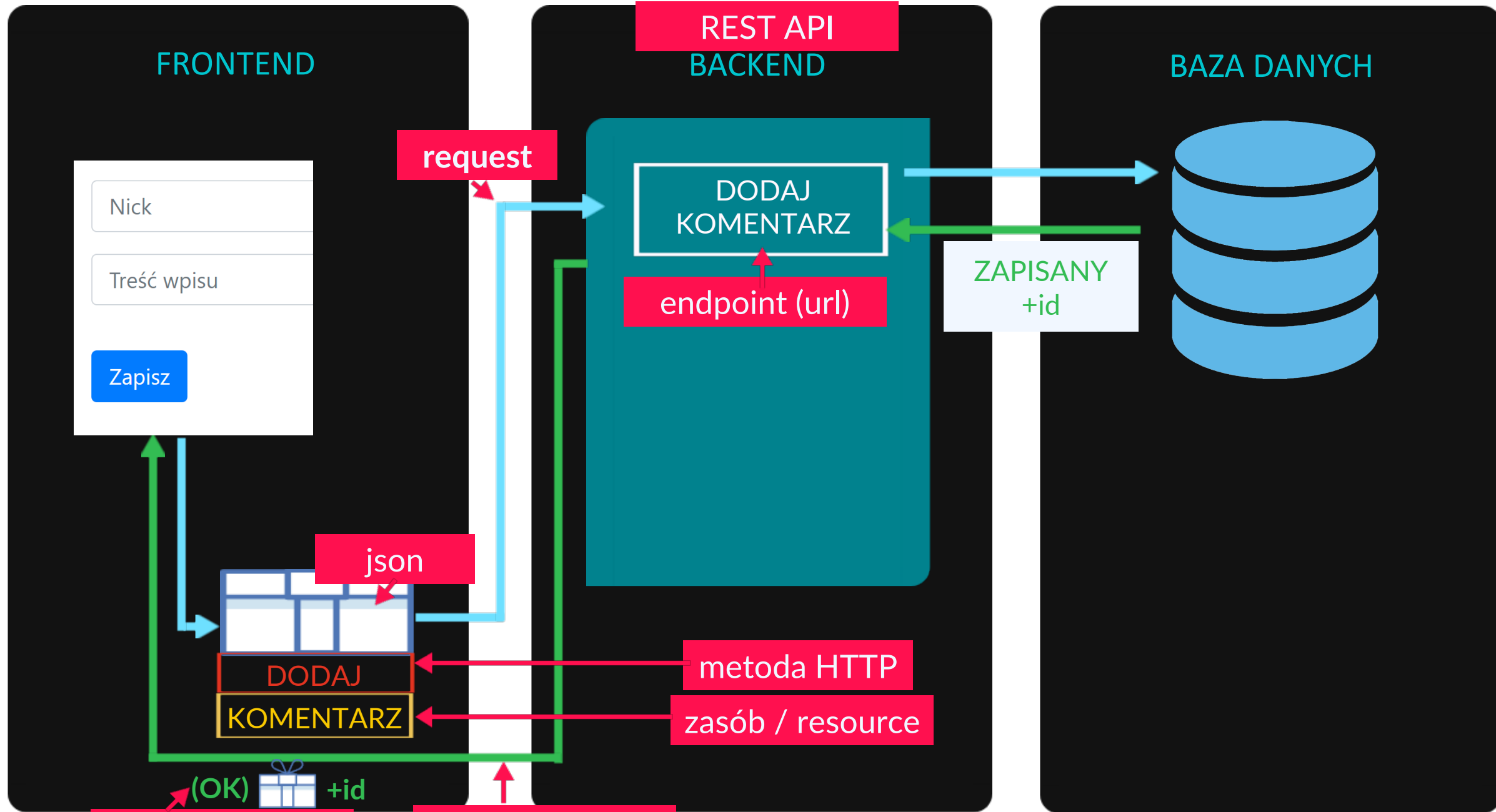
Application Programming Interface



A man with short brown hair and glasses, wearing a blue and white checkered shirt and large black headphones, is seated in a beige office chair. He is looking at a computer monitor. In the background, two other men are working at their desks. The office has large windows letting in natural light. Two blue mugs are on the desk in the foreground.

SII BYDGOSZCZ TEAM

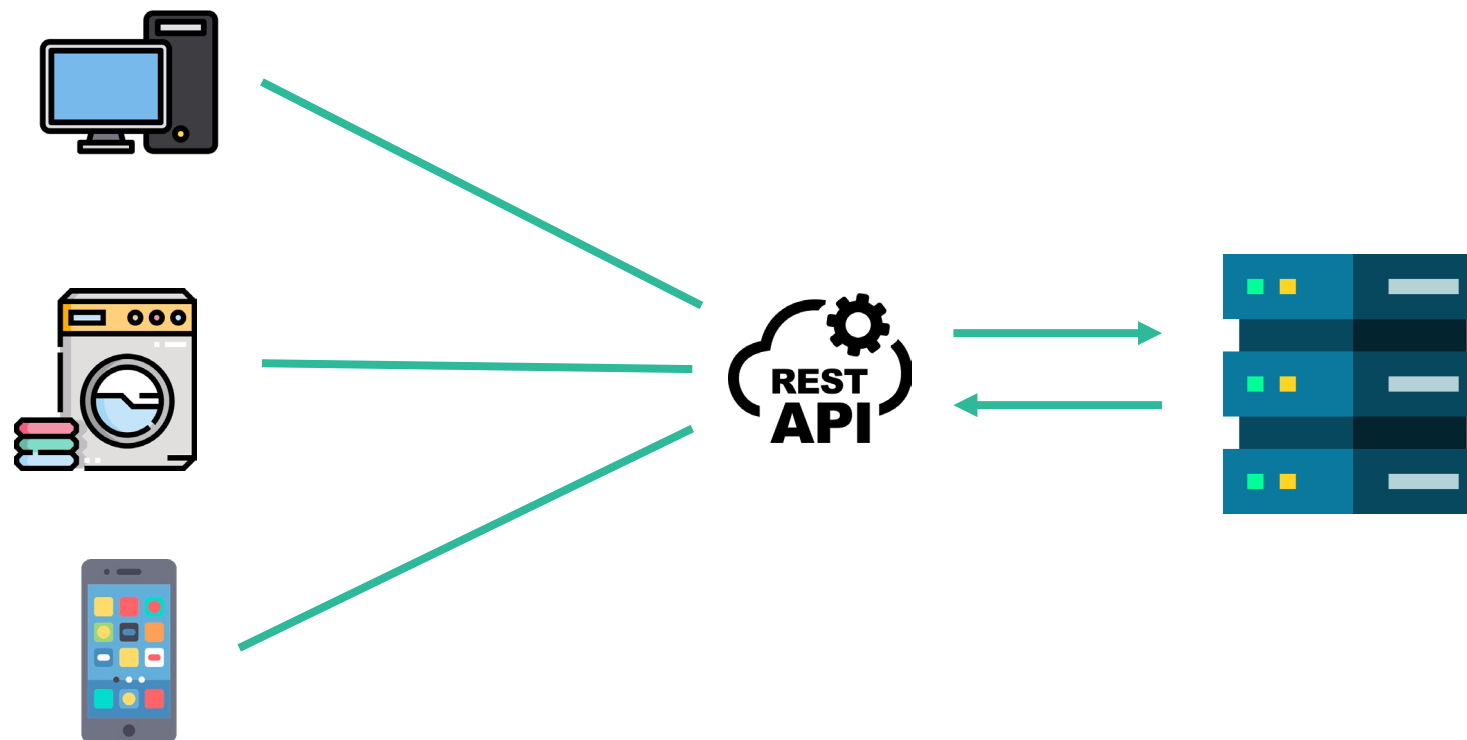
Czym jest backend



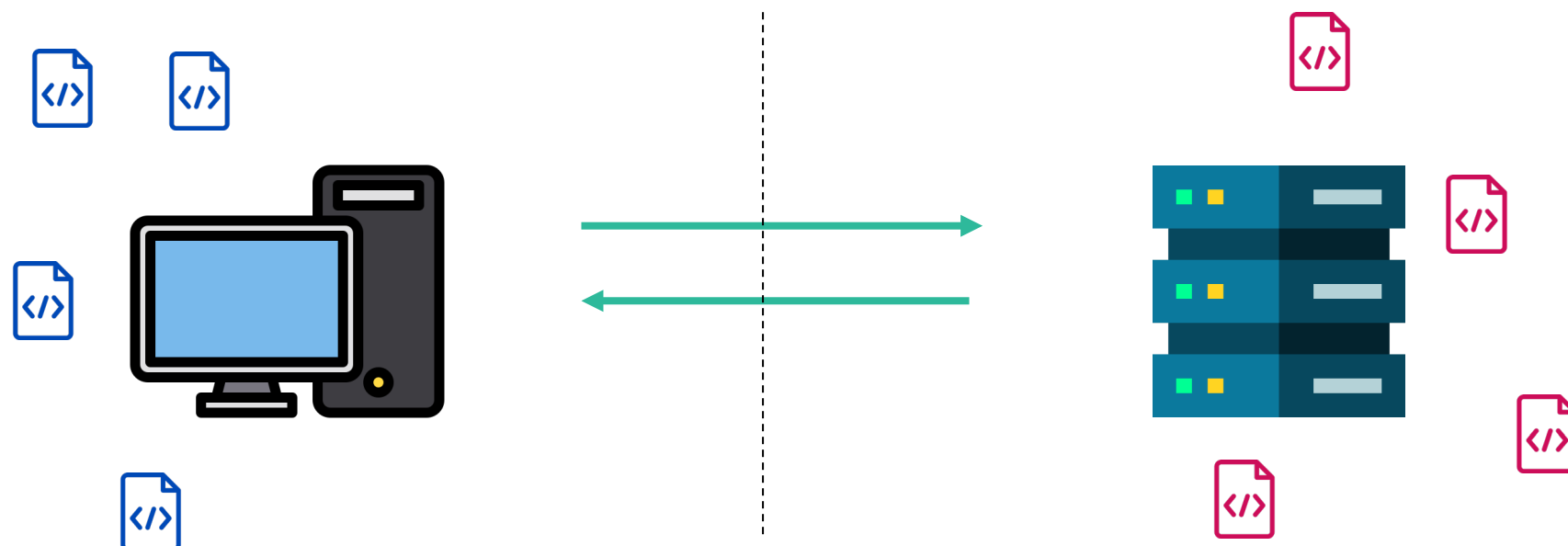
REST

REpresentational **S**tate **T**ransfer

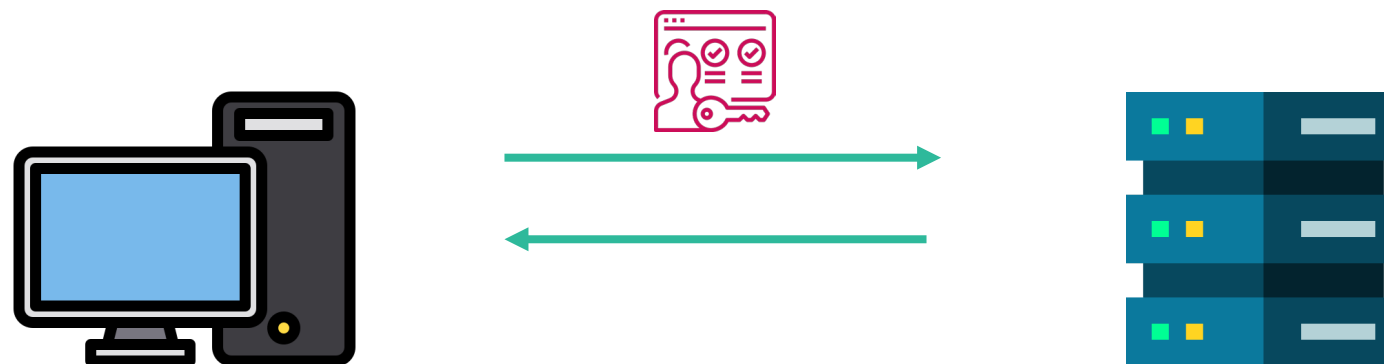
1. Uniform interface
2. Client-server
3. Stateless
4. Cacheable
5. Layered system
6. Code on demand (*opcjonalna*)



Interfejs REST API powinien mieć ustandaryzowany system komunikacji pomiędzy klientem a serwerem



Podział pomiędzy aplikacją działającą po stronie klienta i serwera



Każde zapytanie musi posiadać komplet informacji koniecznych do jego poprawnego zakończenia



API powinno wspierać cache'owanie danych w celu zwiększenia wydajności



System powinien być zaprojektowany w taki sposób aby klient wysyłający zapytanie mógł uzyskać odpowiedź bez konieczności posiadania wiedzy o tym co się dzieje po drugiej stronie



Przewiduje możliwość wysłania fragmentów kodu, które może być wykonany po stronie klienta



1. GET
2. POST
3. PUT
4. PATCH
5. DELETE



1xx - informacyjne

2xx - powodzenia

3xx - przekierowania

4xx – błędu po stronie klienta

5xx – błędu po stronie serwera



200 - ok

201 - created

204 – No content



301 - Moved Permanently



400 - Bad Request

401 - Unauthorized

403 - Forbidden

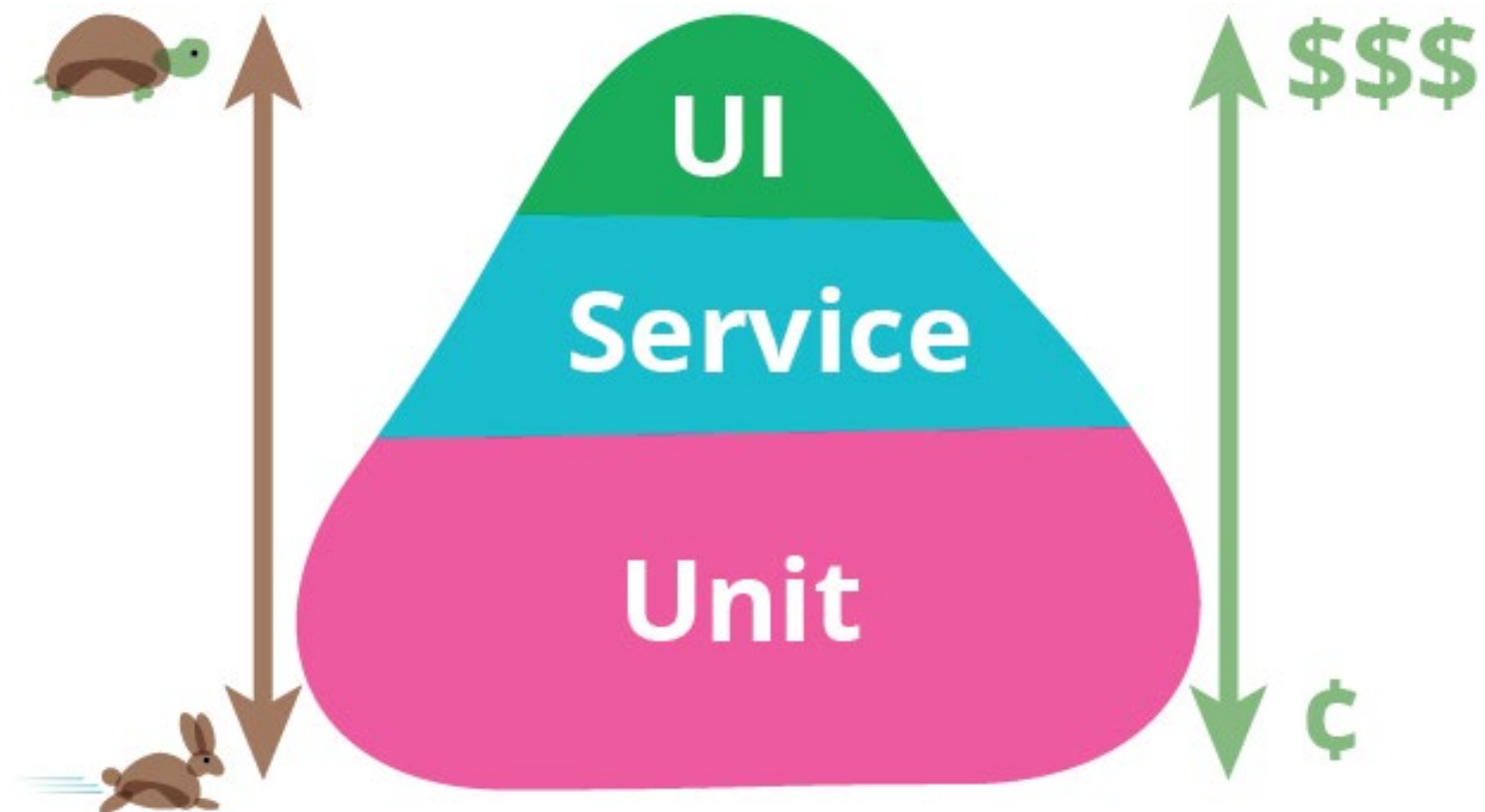
404 – Not Found

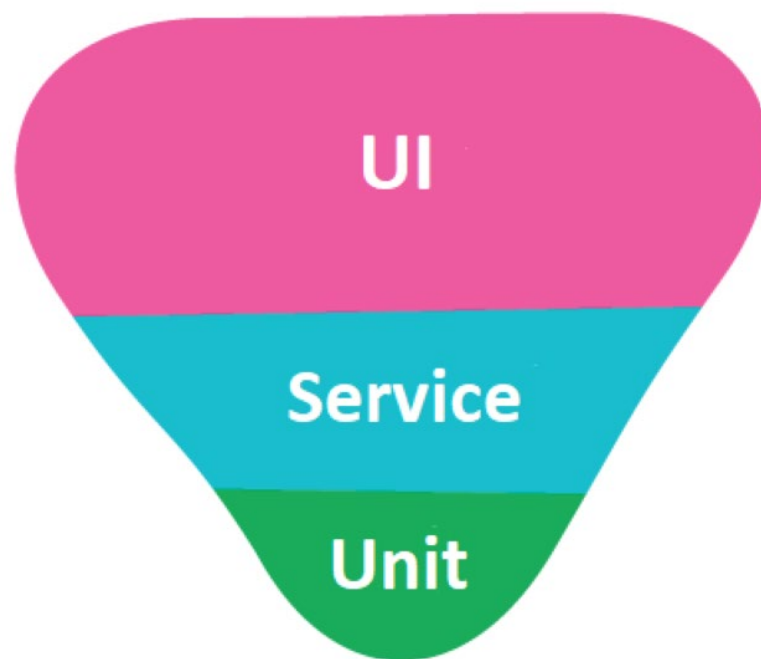
405 – Method Not Allowed



500 - Internal Server Error

Podział testów





Postman DEMO

Testy w aplikacji Postman

```
pm.test("Status code is 200", function () {  
    pm.response.to.have.status(200);  
});
```

Weryfikacja status code

```
pm.test("Status code name has string", function () {  
    pm.response.to.have.status("Created");  
});
```

Weryfikacja tekstu status code

```
pm.test("Body matches string", function () {  
    pm.expect(pm.response.text()).to.include("string_you_want_to_search");  
});
```

Sprawdzenie czy response zawiera tekst

```
pm.test("Content-Type is present", function () {  
    pm.response.to.have.header("Content-Type");  
});
```

Weryfikacja posiadania nagłówka


```
pm.test("Response time is less than 200ms", function () {  
    pm.expect(pm.response.responseTime).to.be.below(200);  
});
```

Weryfikacja czasu odpowiedzi

```
pm.test("Successful POST request", function () {  
    pm.expect(pm.response.code).to.be.oneOf([201, 202]);  
});
```

Weryfikacja czy status code jest jednym z tych podanych w tablicy

```
pm.test("Your test name", function () {  
    var jsonData = pm.response.json();  
    pm.expect(jsonData.value).to.eql(100);  
});
```

Sprawdzenie wartości JSONa z użyciem JSONPath

1. Stwórz zapytanie GET pobierające listę userów (/users)



SII BYDGOSZCZ TEAM

Rest assured



- Czym jest?
- Do czego służy?
- Składnia

REST-assured

Rest assured DEMO

Java HashMap

Typ_danych	Typ_danych
klucz	wartość
klucz	wartość
klucz	wartość
klucz	wartość

HashMap – jest to struktura danych w języku Java przechowująca dane typu **klucz-wartość**

String	Integer
Mark	34
Tom	23
Carl	23
Sara	41

HashMap – jest to struktura danych w języku Java przechowująca dane typu **klucz-wartość**

String	String
Poland	Warsaw
Germany	Berlin
France	Paris
Norway	Oslo

HashMap – jest to struktura danych w języku Java przechowująca dane typu **klucz-wartość**

Integer	String
345234	Mat
12347437	Isabel
12315	Carlos
1234	John

HashMap – jest to struktura danych w języku Java przechowująca dane typu **klucz-wartość**

```
HashMap<String, Integer> emplds = new HashMap<>();  
  
emplds.put("Michael", 3151213);  
emplds.put("Tom", 125122);  
emplds.put("James", 13123);
```

Przykład implementacji

DEMO!

1. Stwórz **HashMap<String, String>** przechowującą dane odnośnie **krajów i ich stolic**
2. Uzupełnij ją danymi dla krajów: Polska, Francja, Anglia, Niemcy
3. Wypisz wartość dla klucza o nazwie **"Polska"**
4. Wydrukuj całą mapę używając *System.out.println(nazwa_mapy);*
5. Dodaj do mapy klucz-wartość: „Polska” – „Kraków”
6. Wydrukuj ponownie całą mapę
7. Wypisz wartość dla klucza o nazwie **"Chorwacja"** (*nieistniejący klucz*)

1. Stwórz **HashMap<Integer, String>** przechowującą ID użytkowników oraz ich imiona
2. Uzupełnij ją 10 elementami typu klucz-wartość
3. Wypisz całą zawartość mapy
4. Wypisz imiona tych użytkowników, których ID jest parzyste (użyj pętli)

Przypomnienie: sprawdzenie czy x jest parzyste (podzielne przez 2):

```
int x = 3;  
if(x % 2 == 0){  
    // co jeżeli jest parzyste  
}
```


1. Stwórz klasę Person z polami String firstName, String lastName, int age
2. W klasie Person nadpisz metodę toString() aby zwracała wszystkie dane osoby***
3. Stwórz **HashMap<Integer, Person>** przechowującą ID oraz obiekt osoby
4. Utwórz 3 obiekty klasy Person i umieść je w mapie z unikalnymi ID
5. Wypisz całą zawartość mapy

*** - część zadania dla chętnych