REST

Hello!

Michał Nowakowski

lead software engineer @EPAM

michal@nowakowski.me.uk

REST
Definicja

- Representational State Transfer
- Zmiana stanu poprzez reprezentacje
- Wykorzystanie protokołu HTTP dla systemów rozproszonych

REST
Cechy

- Jednorodny interfejs
- Bezstanowa komunikacja
- Łatwość cacheowania
- Skalowalność

REST
Cechy

- Wykorzystanie metod HTTP
- Brak wymagań odnośnie formatu danych (najczęściej JSON)
- Bardzo łatwa integracja z JavaScript
- Brak standardowego wsparcia dla autentykacji i autoryzacji

REST

Metody HTTP jako operacje CRUD

- GET – Pobranie danych (**R**ETRIEVE)
- PUT – Modyfikacja danych (**U**PDATE)
- POST – Utworzenie nowego rekordu (**C**REATE)
- DELETE – Usunięcie danych (**D**ELETE)

REST
GET

- GET – Pobranie danych (**R**ETRIEVE)

```
GET http://127.0.0.1:8080/rest-server/user?id=1
```

```
{
  "name": "Adam",
  "surname": "Iksinski",
  "id": 1,
  "credentials": {
    "user": "adam"
  }
}
```

REST
POST

- POST – Utworzenie nowego rekordu (**C**REATE)

```
POST http://127.0.0.1:8080/rest-server/user
```

```
{
  "name": "Janek",
  "surname": "Kowalsky-Nowakosky",
  "credentials": {
    "user": "jan1",
    "password":"haslo"
  }
}
```

REST
PUT

- PUT – Modyfikacja danych (**U**PDATE)

```
PUT http://127.0.0.1:8080/rest-server/user
```

```
{
  "name": "Janek",
  "surname": "Kowalsky-Nowakosky",
  "id": 3,
  "credentials": {
    "user": "jan1",
    "password":"haslo"
  }
}
```

REST
DELETE

- DELETE – Usunięcie danych (**D**ELETE)
  `DELETE http://127.0.0.1:8080/rest-server/user?id=2`

REST
Parametry wywołania

- Parametry zapytania (ang. *query parameters*)
- Parametry ścieżki (ang. *path parameters*)
- Ciało zapytania (ang. *request body*)
- Nagłówki zapytania (ang. *request headers*)

REST
Query parameters

- Dodawane na koniec URLa
- `http://127.0.0.1:8080/rest-server/user?id=2`

```
@QueryParam("id") Integer id
```

REST
Path parameters

- Część ścieżki URL
- `http://127.0.0.1:8080/rest-server/hello/jan`

```
@Path("/hello/{name}")
```

```
@PathParam("name") String name
```

REST
Request body

- Najczęściej obiekt JSON wysyłany jako ciało zapytania HTTP (**nie GET !!!**)

```
@Consumes(MediaType.APPLICATION_JSON)

User user
```

REST
Request headers

- Nagłówki zapytania HTTP
- `My-Custom-Header: some value`

```
@HeaderParam("My-Custom-Header") String header
```

REST
Zadanie 1

- `$ git clone https://github.com/infoshareacademy/jjdd2-materialy-api-rest-soap.git`
- Zmodyfikuj metodę `UserService.sayHello()` tak, aby przyjmowała imię jako parametr ścieżki (ang. *path parameter*) i zwracała powitanie z imieniem

```
@PathParam("name") String name
```

REST
Zadanie 2

- Wstrzyknij kontekst `UriInfo`
- Zobacz, jakie informacje zawiera, zaloguj wybrane

```
@Context
private UriInfo uriInfo;
```

REST
Zadanie 3

- Dodaj nową metodę GET do klasy `UserService`, która sprawdzi w nagłówku zapytania wartość `user-agent`
- Dodaj wartość nagłówka do logu i zwróć do klienta jako tekst

```
@HeaderParam("user-agent") String agent
```

REST
Zadanie 4

- Dodaj metodę GET o ścieżce `/users`, która zwróci listę wszystkich użytkowników (patrz `UserStore`) jako JSON

```
@Produces(MediaType.APPLICATION_JSON)
```

- Metoda powinna zwracać odpowiedni kod (200 gdy lista zawiera dane i 204 gdy lista jest pusta)

```
return Response.noContent().build();
```

- Hasło użytkownika nie może być zwracane!!!

```
@JsonIgnore
```

REST
Zadanie 5

- Dodaj metodę GET o ścieżce `/user?id={id}`, która zwróci konkretnego użytkownika jako JSON

```
@QueryParam("id") Integer id
```

- Metoda powinna zwracać odpowiedni kod (200 gdy użytkownik istnieje i 204 gdy użytkownika z podanym id nie znaleziono)

REST
Zadanie 6

- Utwórz metodę GET o ścieżce `/login`, która zwróci formularz logowania (HTML). Formularz powinien umożliwiać podanie loginu i hasła i wysłanie danych metodą POST do ścieżki `/authenticate`

```
@FormParam("user") String user,
@FormParam("password") String password
```

- Utwórz metodę POST o ścieżce `/authenticate`. Metoda powinna sprawdzać hasło i zwracać odpowiedni kod HTTP.

```
return Response.status(Status.UNAUTHORIZED).build();
```

REST
Zadanie 7

- Utwórz drugą metodę POST do logowania, ale przyjmującą dane logowania (login, hasło) w formacie JSON - `/authenticate`

```
@Consumes(MediaType.APPLICATION_JSON)
```

```
@JsonProperty
```

- Co się stanie, gdy request body będzie w innym formacie?

REST
Zadanie 8

- Utwórz metodę POST `/user` dodającą nowego użytkownika.
- Powinna akceptować dane użytkownika w formacie JSON.
- Po prawidłowym dodaniu użytkownika, metoda powinna zwrócić wszystkich użytkowników w formacie JSON.

REST
Zadanie 9

- Utwórz metodę PUT `/user` modyfikującą istniejącego użytkownika.
- Powinna akceptować dane użytkownika w formacie JSON.
- Po prawidłowym zapisaniu zmian, metoda powinna zwrócić użytkownika w formacie JSON.
- Jeśli podany użytkownik nie istnieje, metoda powinna zwrócić odpowiedni kod HTTP.

REST
Zadanie 10

- Utwórz metodę DELETE `/user?id={id}` usuwającą istniejącego użytkownika.
- Metoda powinna przyjmować id użytkownika jako parametr URLa.
- Po prawidłowym usunięciu, metoda powinna zwrócić listę wszystkich użytkowników.
- Jeśli podany użytkownik nie istnieje, metoda powinna zwrócić odpowiedni kod HTTP.

Thanks!!

Q&A