



TRANSITION
TECHNOLOGIES
MANAGED
SERVICES

GROUP
TRANSITION
TECHNOLOGIES

REST

KRZYSZTOF ŁOPUCKI • 2018

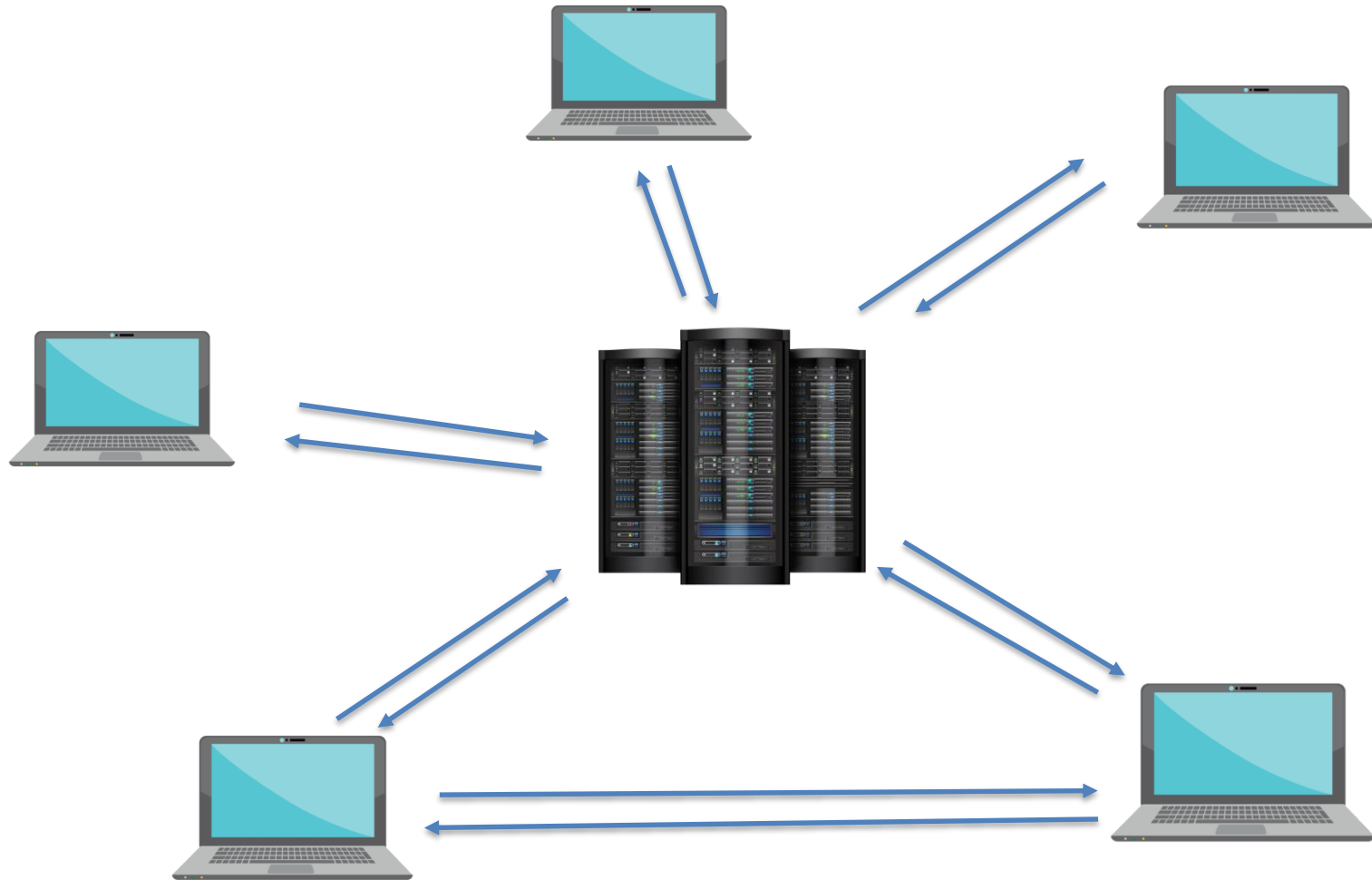
TECHNOLOGIES THAT GUIDE INDUSTRY

REST

REpresentational State Transfer

REST

- Architektura służąca do komunikowania się komputerów w sieci.
- Ułatwia programowanie rozproszonych aplikacji.
- Udostępnia przyjemne w obsłudze API.



- Działają w sieci i w ten sposób komunikują się z innymi komputerami.
- Rozpraszanie aplikacji na wiele węzłów.
- Środowisko mikroservisów.
- Z góry ustalony format danych.

Budowa rządania (request) oraz odpowiedzi (response)

- Metod HTTP, które definiują jakiego rodzaju operacje będą wykonywane.
- Nagłówek, który pozwoli klientowi przekazać dodatkowe informacje do zapytania.
- Ścieżkę do odpowiedniego zasobu.
- Opcjonalną wiadomość przechowującą dane w ciele rządania (ang. body).

Metody HTTP

- GET – najprostsza i najczęściej używana metoda, służy do pobierania zasobów. Standardowo przeglądarki obsługują ten typ metod do pobrania zasobów którymi zwykle są witryny internetowe.
- POST – tworzony jest nowy zasób.
- PUT – aktualizacja pojedynczego zasobu (np. na podstawie id).
- DELETE – usuwanie pojedynczego zasobu (np. na podstawie id).

Parametry nagłówka ACCEPT

- Obrazy – image/png, image/jpeg, image/gif
- Muzyka – audio/wav, image/mpeg
- Filmy – video/mp4, video/ogg
- Aplikacyjne – application/json, application/pdf, application/xml, application/ctet-stream

Kody odpowiedzi serwera

- 200 (OK) – odpowiedź serwera o powodzeniu zrealizowania rządania.
- 201 (CREATED) – odpowiedź serwera informująca o udanym utworzeniu zasobu.
- 204 (NO CONTENT) – odpowiedź o udanym wykonaniu rządania ale ciało odpowiedzi będzie puste.
- 400 (BAD REQUEST) – zasób nie może być pobrany z uwagi na złą składnię rządania lub inny błąd po stronie serwera.
- 401 – klient nie jest zalogowany.
- 403 (FORBIDDEN) – klient nie ma uprawnień do pobrania konkretnego zasobu.
- 404 (NOT FOUND) – nie można znaleźć zasobu. Prawdopodobnie został usunięty albo nie istnieje z jakiegoś innego powodu.
- 500 (INTERNAL SERVER ERROR) – wiadomość o nieudanym stworzeniu odpowiedzi przez serwer. Zwykle informuje nas o błędnym przetworzeniu informacji po stronie serwera.

Kody odpowiednie do sytuacji

- GET — zwraca 200 (OK)
- POST — zwraca 201 (CREATED)
- PUT — zwraca 200 (OK)
- DELETE — zwraca 204 (NO CONTENT)

Przykłady serwer

Pobieranie zasobów

```
@Controller
@RequestMapping("/users")
public class UserController {

    @GetMapping(produces = APPLICATION_JSON_VALUE)
    public List<User> users() {
        return users;
    }
}
```

Method	Request URL
GET	▼ http://localhost:8080/users





```
Array[3]
-0: {
  "id": 1,
  "firstName": "Darek",
  "lastName": "Wos",
  "age": 44,
  "gender": "MALE"
},
-1: {
  "id": 3,
  "firstName": "Mariusz",
  "lastName": "Kowalski",
  "age": 34,
  "gender": "MALE"
},
-2: {
  "id": 4,
  "firstName": "Mariusz",
  "lastName": "Kowalski",
  "age": 34,
  "gender": "MALE"
}
```

Pobieranie jednego zasobu

Method [Request URL](#)
GET ☐ <http://localhost:8080/users/3>

Parameters ☐

200 OK 7.80 ms

```
{
  "id": 3,
  "firstName": "Mariusz",
  "lastName": "Kowalski",
  "age": 34,
  "gender": "MALE"
}
```

Wysyłanie zasobów

```
@PostMapping(consumes = APPLICATION_JSON_VALUE)
public User save(@RequestBody User user) {
    user.setId(Long.valueOf(users.size() + 1));
    users.add(user);
    return user;
}
```

```
{
  "id": 4,
  "firstName": "Teresa",
  "lastName": "Kłos",
  "age": 34,
  "gender": "FEMALE"
}
```

Method	Request URL
POST	http://localhost:8080/users

Parameters ^

Headers	Body
---------	------

Body content type	Editor view
application/json	Raw input

FORMAT JSON MINIFY JSON

```
{
  "firstName": "Teresa",
  "lastName": "Kłos",
  "age": "34",
  "gender": "FEMALE"
}
```

Aktualizacja zasobów

```
@PostMapping(consumes = APPLICATION_JSON_VALUE)
public User update(@RequestBody User user) {
    User oldUser = getUser(user.getId());
    updateUser(oldUser, user);
    return user;
}
```

```
{
  "id": 1,
  "firstName": "Marek",
  "lastName": "Wolski",
  "age": 4,
  "gender": "MALE"
}
```

Method	Request URL
PUT	http://localhost:8080/users
Parameters ^	
Headers	
Body	
Body content type	Editor view
application/json	Raw input
FORMAT JSON MINIFY JSON	
<pre>{ "id": "1", "firstName": "Marek", "lastName": "Wolski", "age": "4", "gender": "MALE" }</pre>	

Przykłady klient

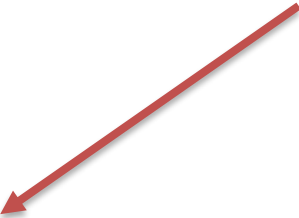
Spring RestTeplate

- Api do wysyłania zapytań restowych
- Stworzone w ramach spring
- Elastyczne w obsłudze

- Wysyłanie danych

```
RestTemplate restTemplate = new RestTemplate();  
HttpEntity<User> request = new HttpEntity<>(user);  
ResponseEntity<User> response = restTemplate  
    .exchange(USERS_URL, POST, request, User.class);  
  
User responseBody = response.getBody();
```

- Aktualizacja danych



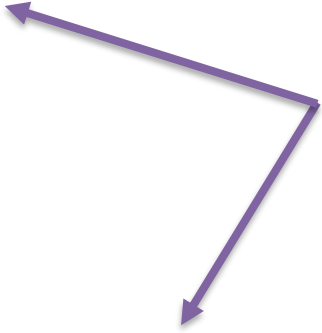
```
User updatedInstance = new User(1, "Darek", "Wos", 44, MALE);
HttpEntity<User> requestUpdate = new HttpEntity<>(updatedInstance);
RestTemplate restTemplate = new RestTemplate();
restTemplate.exchange(
    USERS_URL,
    HttpMethod.PUT,
    requestUpdate,
    Void.class
);
```

- Pobieranie jednego zasobu w postaci JSON

```
RestTemplate restTemplate = new RestTemplate();  
ResponseEntity<String> response  
    = restTemplate.getForEntity(USERS_URL + "/1", String.class);
```

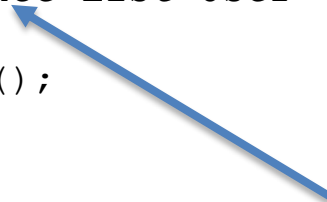
- Pobieranie jednego zasobu w postaci obiektu

```
RestTemplate restTemplate = new RestTemplate();  
User response = restTemplate.getForObject(USERS_URL + "/1", User.class);
```



- Pobieranie kolekcji danych

```
RestTemplate restTemplate = new RestTemplate();
ResponseEntity<List<User>> response = restTemplate.exchange(
    USERS_URL,
    GET,
    null,
    new ParameterizedTypeReference<List<User>>() {
    });
List<User> users = response.getBody();
```



- Usuwanie zasobów

```
String entityUrl = USERS_URL + "/2";  
RestTemplate restTemplate = new RestTemplate();  
restTemplate.delete(entityUrl);
```

Repozytorium

<https://github.com/klopucki/integracja>

Zadanie

Zbuduj aplikację dla serwisu samochodowego
Wymagania:

- Zaprojektowanie API aplikacji klienckiej i serwerowej.
- Implementacja aplikacji serwerowej oraz klienckiej dla stworzonego projektu API.
- Nowi klienci mają mieć możliwość zapisania na wizytę podając wstępną przyczynę wizyty (co w samochodzie nie działa).
- Aplikacja powinna przechowywać informacje o wykonanych naprawach.



TRANSITION
TECHNOLOGIES
MANAGED
SERVICES

GROUP
TRANSITION
TECHNOLOGIES

DZIĘKUJĘ

Krzysztof Łopucki

Krzysztof.Lopucki@ttms.pl