

Module 2 Day 12 (Lecture 10)

HTTP Web Services Post

More Request Types

In the last lecture we saw GET's, which simply read the data. Today we will deal with request types that might potentially change the application's data permanently:

- **POST**: Ideally suited for inserting new data into the data source.
- **PUT**: Ideally suited for updating an existing record within a data source.
- **DELETE**: Ideally suited for removing an existing record from the data source.

For the POST & PUT requests we are converting an object to data

Implementing a POST

Suppose the documentation for the API specifies POST as well :

(POST) *http://localhost:3000/hotels/{id}/reservations*

```
String API_BASE_URL = "http://localhost:3000/"
RestTemplate restTemplate = new RestTemplate();
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);

// Where reservation is an object of type Reservation.
HttpEntity<Reservation> entity = new HttpEntity<>(reservation, headers);
restTemplate.postForObject(BASE_URL + "hotels/" + reservation.getHotelID() + "/reservations", entity,
Reservation.class);
```

Note that POST request have a body and header as well!

Let's implement the POST

Implementing an PUT

Suppose the API's documentation states that there is a PUT endpoint:
(PUT) *http://localhost:3000/reservations/{id}*

Using a REST template we can implement the following:

```
String API_BASE_URL = "http://localhost:3000/"
RestTemplate restTemplate = new RestTemplate();
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);

// Where reservation is an object of type Reservation.
HttpEntity<Reservation> entity = new HttpEntity<>(reservation, headers);
restTemplate.put(BASE_URL + "reservations/" + reservation.getId(), entity);
```

Implementing an PUT

Sometimes requests require that a body and a header be sent along as well. The `HttpEntity` object helps us capture these pieces of information:

```
HttpHeaders headers = new HttpHeaders();  
headers.setContentType(MediaType.APPLICATION_JSON);  
  
// Where reservation is an object of type Reservation.  
HttpEntity<Reservation> entity = new HttpEntity<>(reservation, headers);  
restTemplate.put(BASE_URL + "reservations/" + reservation.getId(), entity);
```

- Here we have a header consisting of an instance of the `HttpHeaders` class.
- We also have a body, which will just be an instance of a `Reservation` class.

Implementing a DELETE

Suppose the API's documentation states that there is a DELETE endpoint:
(DELETE) *http://localhost:3000/reservations/{id}*

Using a REST template we can implement the following:

```
String API_BASE_URL = "http://localhost:3000/"
RestTemplate restTemplate = new RestTemplate();

// Where id is an int:
restTemplate.delete(BASE_URL + "reservations/" + id);
```

Let's Create the PUTs & DELETEs
requests!

Exceptions and Error Handling

There are 2 exceptions to be aware of when dealing with APIs:

- **RestClientResponseException** - for when a status code other than a 2XX is returned.
- **ResourceAccessException** - for when there was a network issue that prevented a successful call.

Let's catch those exceptions