Consuming RESTful APIs
Part 2:
POST, PUT, DELETE

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

Important Buzz Word: Idempotent

You may be asked about whether PUT and POST are idempotent or what differentiates them.

- **`PUT`** is **idempotent**, meaning that calling it once or several times successively has the same effect (that is no side effect).
- **`POST`** is NOT idempotent - we usually use it for creating data. which means each call could wind up with a different effect (i.e. create a different set of data).

Let See
POST/PUT/DELETE
in Action...

Implementing POST Requests

Create HTTP Headers for POST

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

public Reservation addReservation(Reservation newReservation) {
      HttpHeaders headers = new HttpHeaders();
      headers.setContentType(MediaType.APPLICATION_JSON);

      HttpEntity<Reservation> entity = new HttpEntity<Reservation>(newReservation, headers);
      Reservation savedReservation = restTemplate.postForObject(API_BASE_URL + "reservations",
entity, Reservation.class);
      return savedReservation;
}
```

Implementing POST Requests

Create HTTP Headers for POST

Set the **content-type** for JSON

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

public Reservation addReservation(Reservation newReservation) {
      HttpHeaders headers = new HttpHeaders();
      headers.setContentType(MediaType.APPLICATION_JSON);

      HttpEntity<Reservation> entity = new HttpEntity<Reservation>(newReservation, headers);
      Reservation savedReservation = restTemplate.postForObject(API_BASE_URL + "reservations",
entity, Reservation.class);
      return savedReservation;
}
```

Implementing POST Requests

Create HTTP Headers for POST

Set the **`content-type`** for JSON

Create an **`HttpEntity`**, which allows us to combine headers and body

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

public Reservation addReservation(Reservation newReservation) {
        HttpHeaders headers = new HttpHeaders();
        headers.setContentType(MediaType.APPLICATION_JSON);

        HttpEntity<Reservation> entity = new HttpEntity<Reservation>(newReservation, headers);
        Reservation savedReservation = restTemplate.postForObject(API_BASE_URL + "reservations",
entity, Reservation.class);
        return savedReservation;
}
```

Implementing POST Requests

Create HTTP Headers for POST

Set the **`content-type`** for JSON

Create an **`HttpEntity`**, which allows us to combine headers and body

Call **`postForObject`** with the **`HttpEntity`** and class to post for (Reservation).

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

public Reservation addReservation(Reservation newReservation) {
      HttpHeaders headers = new HttpHeaders();
      headers.setContentType(MediaType.APPLICATION_JSON);

      HttpEntity<Reservation> entity = new HttpEntity<Reservation>(newReservation, headers);
      Reservation savedReservation = restTemplate.postForObject(API_BASE_URL + "reservations",
entity, Reservation.class);
      return savedReservation;
}
```

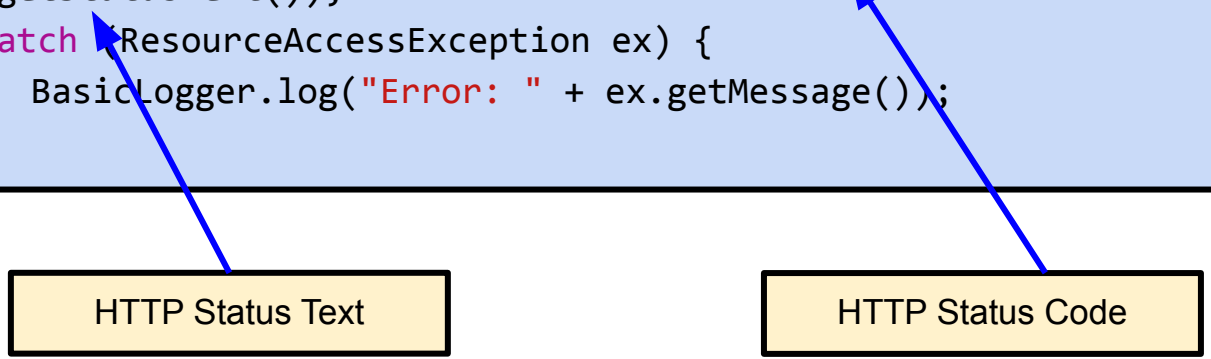
Exceptions and Error Handling

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

- **`RestClientResponseException`** - is thrown when a status code other than a 2XX is returned.
    - Can check status code via this Exception's **`getRawStatusCode()`** method
    - Can get text description of the status code (i.e. Not Found for 404) from this Exception's **`getStatusText()`** method
- **`ResourceAccessException`** - is thrown when there was a network issue that prevented a successful call.

Exception Handling Example

```
try {
     Reservation savedReservation = restTemplate.postForObject(API_BASE_URL +
          "reservations", entity, Reservation.class);
     return savedReservation;
} catch (RestClientResponseException ex) {
     BasicLogger.log("Error: " + ex.getRawStatusCode() + " " +
ex.getStatusText());
} catch (ResourceAccessException ex) {
     BasicLogger.log("Error: " + ex.getMessage());
}
```



Implementing PUT Requests

- PUT requests are similar to POST requests in that they usually have both headers and a payload contained in the message body.
    - We can write code for a PUT request much like our POST code but using the put method rather than postForObject method.

```
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.APPLICATION_JSON);

HttpEntity<Reservation> entity = new HttpEntity<Reservation>(updatedReservation, headers);

restTemplate.put(API_BASE_URL + "reservations/" + updatedReservation.getId(), entity);
```

Implementing DELETE Requests

- DELETE requests are similar to GET requests In that they have only headers and not a payload contained in the message body.
    - We can write code for a DELETE request much like our GET code but using the delete method rather than getForObject method.

```
restTemplate.delete(API_BASE_URL + "reservations/" + id);
```