Module 2-11

Web Services GET

Anatomy of a URL

Here is a URL that uses an IP number:

https://**127.0.0.1**:**3000**

- **protocol:** others - http, ftp
- **ip address:** This is the unique address of a machine on a network.
- **port:** Number allocated for a specific type of service.

Anatomy of a URL

Here is another URL that uses hostnames, this is certainly easier to remember than a bunch of numbers.

https://**skynet**.**wecomeinpeace.com**

- host name: A physical name assigned to your machine.
- **domain name**: Defines a specific “region of control” on the internet, also, .com is referred to as the top-level domain name.

The above URL is an example of a fully qualified domain name (FQDN).

DNS

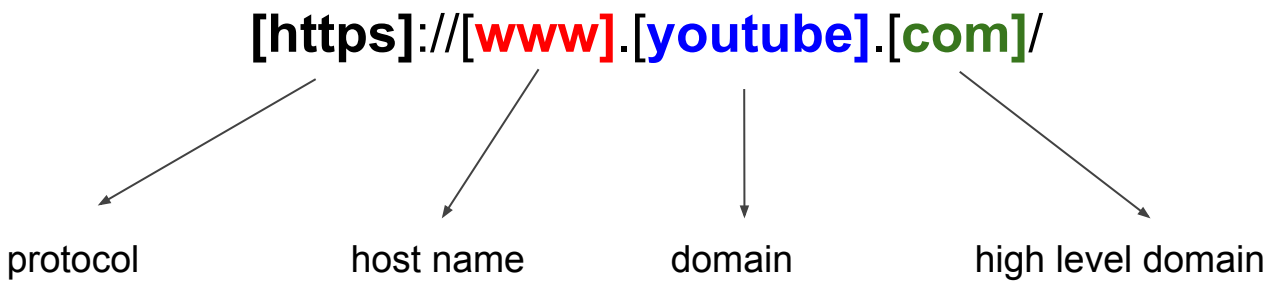
- **DNS** is an acronym for Domain Name System.

- A DNS server is responsible for converting a URL containing human readable domain names (second example) to one containing an IP address (first example).

WWW

Because I’m sure you’ve wondered…

- On a URL the appearance of www has no bearing on the means by which we are communicating with another machine on the network (the protocol is still http or https).
- www is simply a hostname:



What is an API?

- A set of functions and/or procedures designed to interact with an external system.
- Modern cloud architecture relies heavily on API’s.
- *Consuming an API* means interacting with an API’s code to produce a desired result.

API's as a source of data

- We have explored various ways of obtaining data, starting from having Java read a text file, to building a sophisticated relational database like PostgreSQL.
- API's could potentially be yet another source of data for other applications to consume.

Request Types to an API

Recall that a REST controller can be configured to handle various types of requests. Let's review them:

- **GET**: Ideally suited to retrieve all the records from a REST endpoint.
- **GET (with path variable)**: We can configure path variables (i.e. puppy/1) to retrieve a single record of data.
- **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.

Our focus today will be on GETs, in particular how we consume them in Java.

Possible Responses from API

Once a request is made, the REST server can respond with specific status codes:

- **200**: All's well, the request was successful.
- **4XX**: The client (you or your application) has not structured the request correctly. Common examples of these are 400 Bad Request and 401 Unauthorized Request.
- **5XX**: The server has encountered some kind of error. The most common of these is the 500 Internal Server Error message.

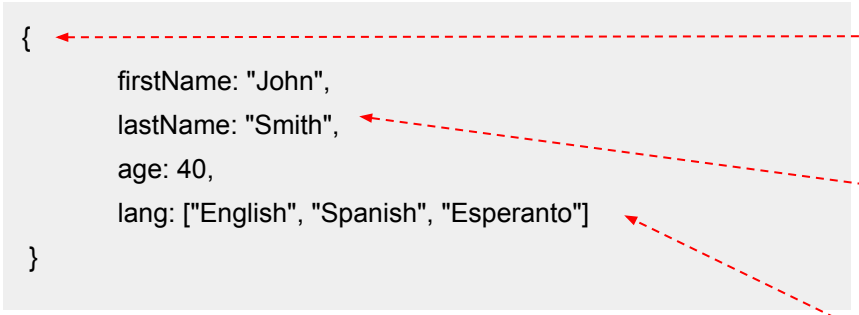
JSON

JSON (JavaScript Object Notation)

- The data used to interact with the API must be in JSON format.
- JSON is simple! Only three rules:
  - Objects in JSON’s are delimited by curly braces { … }
  - Arrays in JSON’s are delimited by brackets [...]
  - Data is listed in key-value pairs (key : value)
- The categories above can be mixed and matched to create complex descriptions of data.

JSON Example

Here is an example of JSON data:



- The object itself is enclosed with a set of curly braces.
- Each property of the object is listed as a key value pair.
- An array is enclosed with square brackets.

Postman

- Postman is a common tool used to interact with API’s.
- It allows us to test our understanding of an API before we implement API code.

Let’s see some examples of JSON through Postman

Making a GET Request through Java using getForObject

The RestTemplate class provides the means with which we can make a request to an API. Here is an example call:

```
String API_BASE_URL = "http://helpful-site/v1/api/data";
RestTemplate restTemplate = new RestTemplate();
MyObj myobj = restTemplate.getForObject(API_BASE_URL, MyObj.class);
```

Note that we can specify the return type of the API call with the second parameter (MyObj.class). Alternatively, if you are getting an array of objects back, we can write the following:

```
MyObj [ ] myobj = restTemplate.getForObject(API_BASE_URL, MyObj[ ].class);
```

Getting Results Back

- If no exception was encountered, then you've made the request successfully.

- The deserialized object contains the request's data and can thus be interrogated using the class' getters.

Making a GET Request through Java using getForEntity

- There is a family of methods used to make API calls, and you've just seen getForObject.
- Another method is getForEntity which returns a more generalized return response object as opposed to an object of a specific class:

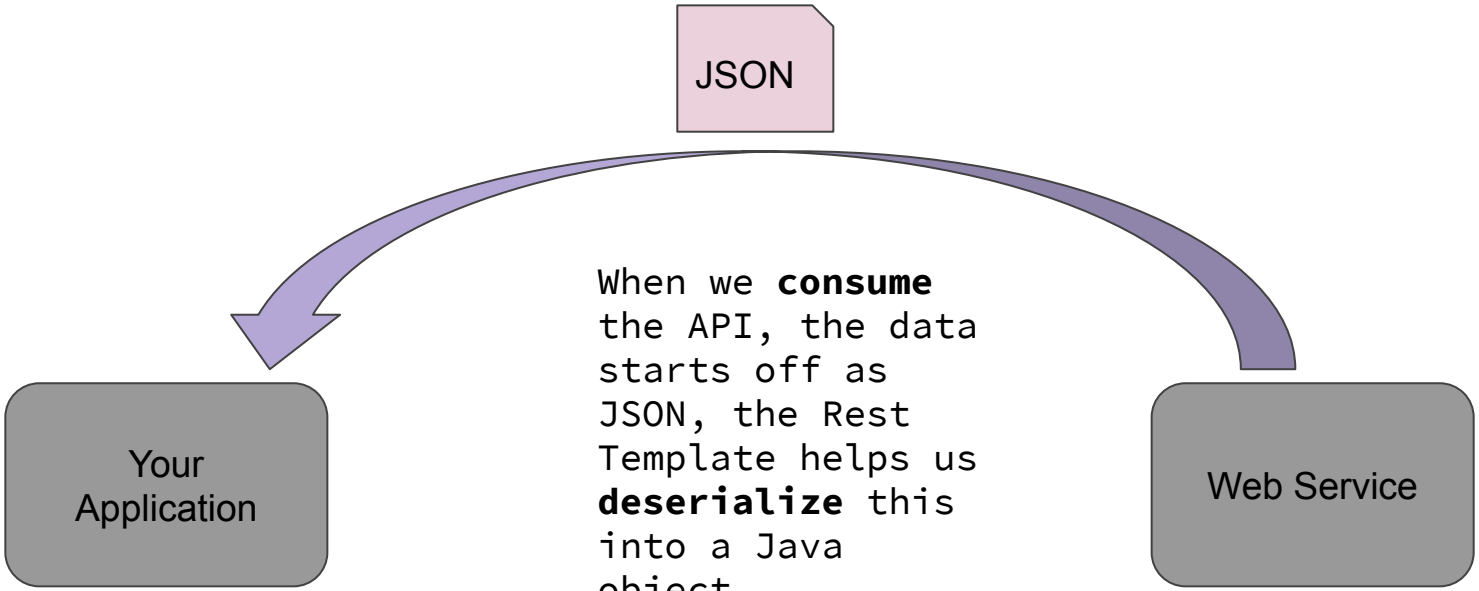
```
RestTemplate restTemplate = new RestTemplate();
ResponseEntity response = restTemplate.getForEntity(...)
```

Serialize vs Deserialize

- When we convert our data from a JSON string into an object, we are **deserializing**.

- We won’t cover this today, but the opposite of this is to **serialize**, which converts the object into a byte stream.

Serialize vs Deserialize



When we **consume** the API, the data starts off as JSON, the Rest Template helps us **deserialize** this into a Java object.

JSON Server

- For the demo today and the homework we will be simulating an external API using a tool called JSON Server.

Let’s write some GET requests!