Class 4 – Spring Boot Microservice Enterprise Applications with Java

Contents

Review

- Class3
- Q&A

Break

Introduction to REST

HTTP Overview

Before we can talk about REST you need to be familiar with the HTTP protocol[1]. In the 7[2] layer OSI network model, HTTP sits on top in the application layer.

The mechanism is that the **user-agent** makes a **request** to the server and receives a **response**. This is what your grandma uses when checking her email so it is ubiquitous.

When using a browser (which is a user-agent) you are generally using the GET method and sometimes POST. Most commonly used methods:

- GET
- POST
- PUT
- DELETE

PUT and DELETE are **idempodent** per the protocol meaning that multiple invocations of the exact same call should have the same effect as a single call. This is an important concept, however theory vs practice might be different.

These methods map nicely to CRUD operations!

HTTP protocol is **stateless** – server may maintain a "session" however the cookies/session id are sent by the user agent on *every* request to the server.

URI[3] - is a string of characters that identify a web resource

- http://en.wikipedia.org/wiki/URI#Examples_of_URI_references ("http" specifies the 'scheme' name, "en.wikipedia.org" is the 'authority', "/wiki/URI" the 'path' pointing to this article, and "#Examples_of_URI_references" is a 'fragment' pointing to this section.)
- http://example.org/absolute/URI/with/absolute/path/to/resource.txt
- //example.org/scheme-relative/URI/with/absolute/path/to/resource.txt
- /relative/URI/with/absolute/path/to/resource.txt
- relative/path/to/resource.txt
- ../../../resource.txt
- ./resource.txt#frag01

[1] http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[2] http://en.wikipedia.org/wiki/OSI_model
[3] http://en.wikipedia.org/wiki/Uniform_resource_identifier

HTTP Response – There is an http response code[4] defined in the RFC and corresponds to a success/error/other condition. Some of the important ones:

Code	Status Name	Detail
200	OK	General success GET an entity corresponding to the requested resource is sent in the response; HEAD the entity-header fields corresponding to the requested resource are sent in the response without any message-body; POST an entity describing or containing the result of the action;
201	Created	New resource created
302	Found	User agent redirection
401	Unauthorized	Credentials not provided
403	Forbidden	Credentials invalid
404	Not Found	Resource not found
409	Conflict	Invalid state of resource
500	Internal Server Error	Unexpected server error

Headers – these are name-value pairs included on the request and response

Query params – these are name-value pairs typically included in a GET request

Body – this a the message on the request and response

The HTTP protocol typically operates on Port 80 and HTTPS over 443. Generally firewalls are already preconfigured for these.

You can capture HTTP network traffic from your browser using plugins in Chrome/Firefox or a standalone application like Fiddler. Wire Shark shows traffic at all OSI layers but likely overkill for our purpose.

- Chrome – Console (hit F12)
- Firefox – Live Headers, Firebug plugins

HTTP2

For our purposes HTTP1.1 is sufficient.

API = Application Programming Interface

APIs creates the ability for someone else to use your application so their ease of use is very important. For example you use the Java SDK everyday – Sun put a lot of thought into that design!

[4] http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

When developing internet applications that must communicate with each other we have look at various methods (see Class 1). REST APIs are the latest iteration.

Why REST?

First proposed in a PhD thesis by Fielding and REST = Representation State Transfer[5]

- Scalability – not just performance but also can you scale to meet all the features you have to support
- Generality – REST is independent of underlying technology. Applications in multiple languages like Java, Ruby, C# etc can communicate with each other
- Independence – a complex application maybe composed of many independent REST services/components. For example take a look at Netflix's Hystrix[6]
- Latency and Caching – allows to improve application performance
- Security – Use HTTP security (SSL and other mechanisms)
- Encapsulation – the APIs allow abstraction levels and data may/may not be exposed as necessary

Your REST APIs may not follow the HTTP protocol to the letter – which is ok. Practical implementation vs strict REST is debatable[7]. I am not a "rest-afarian." You can pick your own side!

JSON = JavaScript Object Notation[8]

XML is a bit verbose and JSON offers a simpler alternative. XML is also perfectly acceptable though probably not as fashionable!

From JSON.org

JSON is built on two structures:

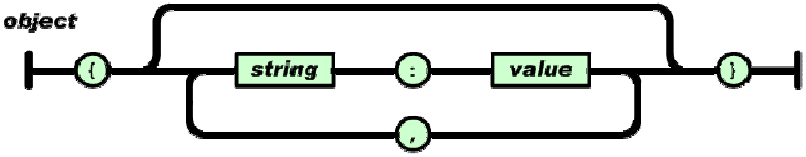
- A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array
- An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence

These are universal data structures. Virtually all modern programming languages support them in one form or another. It makes sense that a data format that is interchangeable with programming languages also be based on these structures.

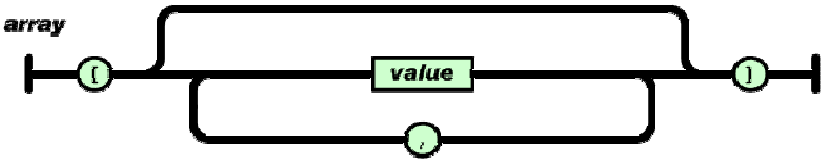
In JSON, they take on these forms:

[5] http://en.wikipedia.org/wiki/Representational_state_transfer
[6] https://github.com/Netflix/Hystrix
[7] https://blog.apigee.com/detail/api_design_are_you_a_rest-afarian_or_a_rest_pragmatist
[8] http://www.json.org/

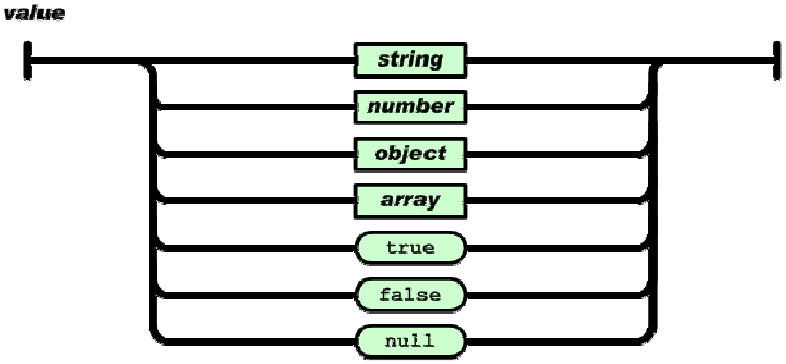
- An *object* is an unordered set of name/value pairs. An object begins with { (left brace) and ends with } (right brace). Each name is followed by : (colon) and the name/value pairs are separated by , (comma)



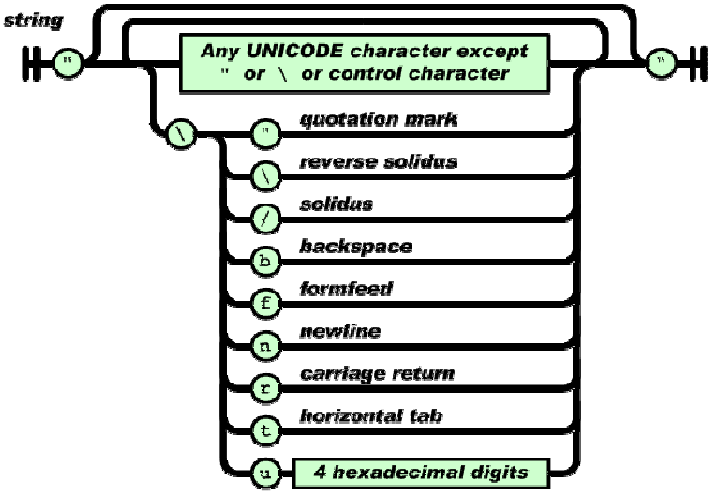
- An *array* is an ordered collection of values. An array begins with [(left bracket) and ends with] (right bracket). Values are separated by , (comma)



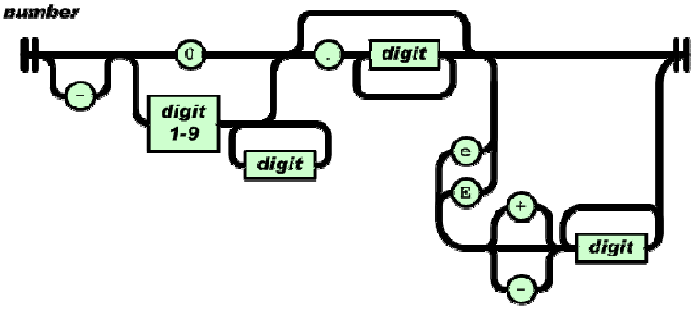
- A *value* can be a string in double quotes, or a number, or true or false or null, or an object or an array. These structures can be nested



- A *string* is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. A string is very much like a C or Java string



- A number is very much like a C or Java number, except that the octal and hexadecimal formats are not used.



Example

```
{
    "firstName": "John",
    "lastName": "Doe",
    "age": 25,
    "address": {
        "streetAddress": "21 Main Street",
        "city": "Mountain View",
        "state": "CA",
        "postalCode": 94043
    },
    "phoneNumbers": [
        {
            "type": "home",
            "number": "408 555-1234"
        },
        {
            "type": "fax",
            "number": "408 555-4567"
        }
    ]
}
```

More examples: http://json.org/example.html

Break

REST+JSON APIs

In REST we create “resources” that can then acted on through HTTP. Resources have “properties” and “behaviors” can be applied to these resources. This can be thought of analogous to Objects and their attributes and methods for those objects.

We need to present these as HTTP URIs and there are two fundamental types:

- Collections – multiple resources or container representation. Eg: /users
- Instances – a particular instance. Eg: /users/123

Behaviors - ***generally*** these crud behaviors map to the HTTP methods

- POST – create a resource. Eg:

  POST /users
  {
  ….user representation
  }
  Response: 201 Created
  Location http….. /users/123
  Body { … json }
- GET – get a collection or instance as in above examples
  Request GET /users/123
  Response: 200 OK
  Body {… json }
- PUT – modify a resource
  PUT /users/123
  {
  ….user representation for what you want to update - “sparse” update
  }
  Response: 200 OK
  Body { …json }
- DELETE – delete
  DELETE /users/123
  Response: 200 OK
  Body – could be blank

Request header: Accept – When sending the request the user-agent tells the server how it is sending its request. For example “accept:application/json” means you are sending JSON. These may be custom defined.

Response header: Content-Type – In the response the server tells the user-agent how it is responding. Again could be custom defined.

Sub-resources: Generally resources have sub-resources. Making resources top-level or sub-resources is a subjective design choice. For example: /users/123/checking-accounts/4567

Conventions

- Attribute Names – use camelCase (its JavaScript object notation after all)

- Dates – use ISO-8601 (and for time zone use Z which is UTC)

Error Response

Returning meaningful error responses is important on an API. Usually a HTTP 409 is returned, along with an error body. For example:

HTTP 409 Conflict

```
{
   "status":409,
   "errorCode":"OOPS",
   "message":"something bad happened",
   "debug":"some debug string, maybe in dev environment only"
}
```

Take a look at how other public APIs do error reporting.

API Versioning

While we will not be versioning APIs for our small app it is a significant concern in large applications. Over time your APIs will change and new versions created may not be backwards compatible so clients using your application will need to know which version of the API to use. This version is typically sent in one of two ways:

- On the URL for eg: /v1/users/123 (more popular way)
- As a header – this be either a custom header for eg:X-Version-Foo=1 or on the accept header for eg: application/json+foo;application,v=1 (custom)

Dates

How to handle dates is a common question. Take a look at ISO 8601[9] which is the standard how dates are represented. In Java there is the SimpleDateFormat class which can help you do the conversions.

Date:	**2017-04-23**
Combined date and time in UTC:	**2017-04-23T21:36:34+00:00** **2017-04-23T21:36:34Z**

Example

Let us look at a couple public examples:

Twitter: https://dev.twitter.com/docs/api/1.1

Intuit QuickBooks: https://developer.intuit.com/

[9] https://en.wikipedia.org/wiki/ISO_8601

Example – TBTF Bank

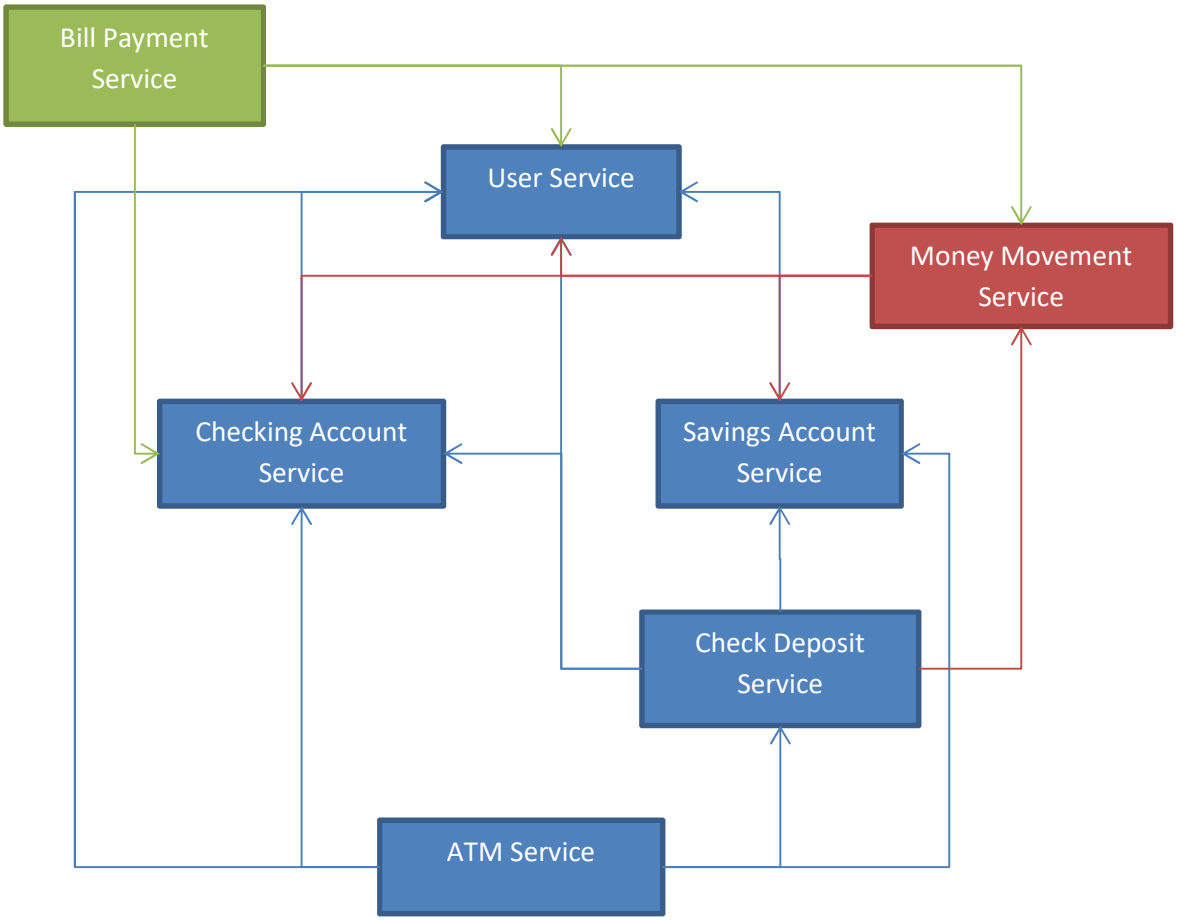
Continuing with our Too Big to Fail Bank, recall we had defined some services. In brief:

1. User Service – manages the user identity
2. Checking Account Service – manages checking accounts
3. Saving Account Service
4. Check Deposit Service – knows how to deposit checks, presumably given an image can read it, process funds and deposit them
5. ATM Service – helps operations at an ATM and uses other services to accomplish this

These were by no means all the services you would need in a bank. For example, depositing a check requires moving money:

- From different account in the same bank or from a different bank
- It takes multiple days so likely there are some background operations that take many days to complete
- If the check is successfully processed then the hold on the deposit is removed
- Also the money movement may involve removing funds from your account if you wrote a check
- Insufficient funds may cause some errors!
- Moving money is not limited to checks alone. For example direct deposit of your paycheck or online bill payments
- Testing external services may have new challenges

Let us expand our view:



How could we modify our implementation?

Refactoring code: we will be modifying the example TBFT bank code in coming weeks and you will be modifying your assignment code as well. It is ok if you have to change code, refactoring is a good thing!

REST APIs

Error Reporting

Errors are reported with a JSON response body. The error response includes the following fields in its JSON body

JSON Field name	Detail
status	The HTTP status code
code	One of the codes from the table below
message	A message to help you debug the problem. This message is for developer debugging only and is not localized
debug	This is a detailed error message, sometimes

	including a stack track. This is returned only in the test environment and not in production

Known error codes

Code	Detail
INVALID_FIELD	For a required field either the value is not provided or it does not meet the conditions
MISSING_DATA	Some required values are missing

*Codes could be numeric if you prefer. I like a string as it is easier to understand and simple Java enum

REST APIs

Method and endpoint: **POST /users**

Details: Create a new user. This will be used in the bank when a customer is signing up.

Request

JSON Field name	Conditions	Detail
user.firstName	Required, max 45 chars	The user's first name
user.lastName	Required, max 45 chars	The user's last name
user.pin	Required, max 10 chars	A secret number

Success Response
HTTP Status 201
Location header for new entity

JSON Field name	Detail
user.id	Id of newly created user
user.firstName	The first name
user.lastName	The last name

Errors

HTTP	Code	Detail
409	INVALID_FIELD	Missing or invalid value for required field

Examples

1. Create a user

Request

POST /users

```
{
   "user":{
      "firstName":"John",
      "lastName":"Doe",
      "pin":"1234"
   }
}
```

Response: 201

Location: /users/62

```
{
   "user":{
      "id":62,
      "firstName":"John",
      "lastName":"Doe"
   }
}
```

2. Error in first name

Request

POST /users

```
{
   "user":{
      "lastName":"Doe",
      "pin":"1234"
   }
}
```

Response: 409

```
{
   "error":{
      "status":409,
      "code":"INVALID_FIELD",
      "message":"missing or invalid value firstName",
      "debug":"some exception detail"
   }
}
```

GET /users

Get users by some criteria. This will be by call center agents or in the bank to lookup users

Request Query Params – atleast one parameter must be provided

Query Param	Condition	Details
firstName	Optional, min 3 chars, max 45 chars	Partial search is performed on

		firstName, case insensitive
lastName	Optional, min 3 chars, max 45 chars	Partial search is performed on lastName, case insensitive

Success Response

HTTP Status 200

JSON Fields	Details
users	JSON Array. Each value in the array is a user. Refer to POST user response for user entity

Errors

HTTP	Code	Detail
409	MISSING_DATA	Required parameters are missing

Examples

1. Search by last name

Request

GET /users?lastName=doe

Response: 200

```
[
   {
      "id":789,
      "firstName":"John",
      "lastName":"Doe"
   },
   {
      "id":8987,
      "firstName":"Jane",
      "lastName":"Doe"
   }
]
```

1. Nothing found

Request

GET /users?lastName=hello

Response: 200

```
[]
```

These are a few samples but it hopefully gives you an idea.

Homework for Class 4

Download MySQL as appropriate for your platform. For Windows download the Installer. You only need the two components listed below. Choose custom install.

Note: depending on your install option the Workbench may get installed along with the server and you will not need to download it separately.

- MySQL Community Server: http://dev.mysql.com/downloads/mysql/
- MySQL Workbench: http://dev.mysql.com/downloads/tools/

The latest stable version (GA or Generally Available version) is ok.