Java Spring

RESTful

What is REST?

- REST stands for **RE**presentational **S**tate **T**ransfer and is an architectural style for designing distributed network applications
- More precisely, REST is an architectural style consisting of a coordinated set of architectural constraints applied to components, connectors, and data elements, within a distributed hypermedia system.

Client-Server

- Concerns should be separated between clients and servers. This enables client and server components to evolve independently and in turn allows the system to scale.

Stateless

- The communication between client and server should be stateless. The server need not remember the state of the client. Instead, clients must include all of the necessary information in the request so that server can understand and process it.

Layered System

- Multiple hierarchical layers such as gateways, firewalls, and proxies can exist between client and server. Layers can be added, modified, reordered, or removed transparently to improve scalability.

Cache

- Responses from the server must be declared as cacheable or noncacheable. This would allow the client or its intermediary components to cache responses and reuse them for later requests. This reduces the load on the server and helps improve the performance.

Uniform Interface

- All interactions between client, server, and intermediary components are based on the uniformity of their interfaces.
- This simplifies the overall architecture as components can evolve independently as long as they implement the agreed-on contract..

Code on demand

- Clients can extend their functionality by downloading and executing code on demand. Examples include JavaScript scripts, Java applets, Silverlight, and so on. This is an optional constraint.

URI

- Uniform Resource Identifier, or URI, for uniquely identifying resources. scheme:scheme-specific-part (http://zipcoder.io)

URI Templates

- Resource links can be dynamic
- http://blog.zipcoder.io/**year**/posts

Representation

- RESTful resources are abstract entities, which represent a snap shot of data at a point in time.

HTTP Method (GET)

- The GET method is used to retrieve information from the given server using a given URI. Requests using GET should only retrieve data and should have no other effect on the data.

Idempotency

- An operation is considered to be idempotent if it produces the same server state whether we apply it once or any number of times. HTTP methods such as GET, HEAD (which are also safe), PUT, and DELETE are considered to be idempotent, guaranteeing that clients can repeat a request and expect the same effect as making the request once.

HTTP Methods (POST)

- A POST request is used to send data to the server, for example, customer information, file upload, etc. using HTML forms.

HTTP Methods (HEAD)

- Same as GET, but transfers the status line and header section only.

HTTP Methods (PUT)

- Replaces all current representations of the target resource with the uploaded content.

HTTP Methods (**DELETE**)

- Removes all current representations of the target resource given by a URI.

HTTP Methods (**CONNECT**)

- Establishes a tunnel to the server identified by a given URI.

HTTP Methods (**OPTIONS**)

- Describes the communication options for the target resource.

HTTP Methods (**TRACE)**

- Performs a message loop-back test along the path to the target resource.

CRUD

- Create
- Read
- Update
- Delete

HTTP Status Codes

- 1xx Informational
  - 100 Continue
- 2xx Success
  - 200 OK
- 3xx Redirection
  - 300 Multiple Choices
- 4xx Client Error
  - 400 Bad Request
- 5xx Server Error
  - 500 Internal Server Error

HTTP Status Codes (1xx Informational)

- 100 Continue -
  This means that the server has received the request headers, and that the client should proceed to send the request body (in the case of a request for which a body needs to be sent; for example, a POST request).
- 101 Switching Protocols –
  This means the requester has asked the server to switch protocols and the server is acknowledging that it will do so
- 102 Processing -
  As a WebDAV request may contain many sub-requests involving file operations, it may take a long time to complete the request. This code indicates that the server has received and is processing the request, but no response is available yet

HTTP Status Codes (2xx Success)

- 200 OK
  Standard response for successful HTTP requests.
- 201 CREATED
  The request has been fulfilled and resulted in a new resource being created.
- 202 Accepted
  The request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place.

HTTP Status Codes (3xx Redirection)

- 300 Multiple Choices
  Indicates multiple options for the resource that the client may follow.
- 301 Moved Permanently
  This and all future requests should be directed to the given URI
- 308 Permanent Redirect
  The request, and all future requests should be repeated using another URI.

HTTP Status Codes (4xx Client Error)

- **400 Bad Request**
  The server cannot or will not process the request due to something that is perceived to be a client error
- **401 Unauthorized**
  Similar to *403 Forbidden*, but specifically for use when authentication is required and has failed or has not yet been provided.
- **404 Not Found**
  The requested resource could not be found but may be available again in the future. Subsequent requests by the client are permissible

WEB Socket

- A protocol providing full-duplex communication channels over a single TCP connection.

Servlet

- A **servlet** is a small Java program that runs within a Web server. **Servlets** receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol.

AOP

- Aspect-Oriented Programming (**AOP**) complements Object-Oriented Programming (OOP) by providing another way of thinking about program structure. The key unit of modularity in OOP is the class, whereas in**AOP** the unit of modularity is the aspect.

ORM

- Object-relational mapping (**ORM**) is a technique (a.k.a. design pattern) of accessing a relational database from an object-oriented language (Java, for example)

JMS

- **JMS** is a part of the Java Platform, Enterprise Edition, and is defined by a specification developed under the Java Community Process as JSR 914. It is a messaging standard that allows application components based on the Java Enterprise Edition (Java EE) to create, send, receive, and read messages.

Transaction

- In computer programming, a transaction usually means a sequence of information exchange and related work (such as database updating) that is treated as a unit for the purposes of satisfying a request and for ensuring database integrity.

BEAN

- A **bean** is an object that is instantiated, assembled, and otherwise managed by a **Spring** IoC container. These **beans** are created with the configuration metadata that you supply to the container

IOC

- In software engineering, **inversion of control** (**IoC**) describes a design in which custom-written portions of a computer program receive the flow of control from a generic, reusable library.

Dependency Injection (DI)

- **Dependency Injection** design pattern allows us to remove the hard-coded dependencies and make our application loosely coupled, extendable and maintainable.

Aspect Oriented Programming

- *Aspect-Oriented Programming* (AOP) complements Object-Oriented Programming (OOP) by providing another way of thinking about program structure. The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the *aspect*. Aspects enable the modularization of concerns such as transaction management that cut across multiple types and objects.

@RequestParam annotation

- The @RequestParam annotation is used to bind Servlet request parameters to handler/controller method parameters.

@RequestMapping

- @RequestMapping annotation is used to map a Web request to a handler class or handler method

```
@RequestMapping(value="/saveuser.html", method=RequestMethod.POST)
public String saveUser(@RequestParam String username, @RequestParam String password)
{
    // Save User logic
    return "success";
}
```