# RESTFUL WEB SERVICES

Teaching Faculty: Umur INAN

Prepared by Umur INAN

## RESTFUL WEB SERVICES

- REST = Representational State Transfer

- REST is an architectural style consisting of a coordinated set of architectural constraints

- First described in 2000 by Roy Fielding in his doctoral dissertation at UC Irvine

- RESTful is typically used to refer to web services implementing a REST architecture

## Restful web services

- Alternative to other distributed-computing specifications such as SOAP
- Simple HTTP client/server mechanism to exchange data
- Everything – the UNIVERSE is available through a URI
- Utilizes HTTP GET/POST/PUT/DELETE operations

## Architectural principles

- **Uniform Interface**
    - Individual resources are identified in requests, i.e.,using URIs in web-based REST systems.   Resource, URI, HTTP methods  (CRUD)
- **Client-Server**
    - Separation of concerns. A uniform interface separates clients from servers.
- **Stateless**
    - The client-server communication is further constrained by no client context being stored on the server between requests.

## Architectural principles

- **Cacheable**
  - Clients can cache responses.
- **Layered System**
  - A client cannot necessarily tell whether it is connected directly to the end-server, or to an intermediary along the way.
- **Code** on **demand** (optional)
  - REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

## Restful Api http methods

| Http Method | Description |
|---|---|
| GET | To retrieve a resource from the server. |
| POST | To create a resource on the server. |
| PUT | To change the state of a resource or to update it. |
| DELETE | To remove or delete a resource from the server. |

## IDEMPOTENT + SAFE

- Idempotent: An identical request can be made once or several times in a row with the same effect while leaving the server in the same state.

- Safe: An HTTP method is safe if it doesn't alter the state of the server. In other words, a method is safe if it leads to a read-only operation.

## Restful Api http methods

| Http Method | safe | idempotent |
|---|---|---|
| GET | Yes | Yes |
| POST | No | No |
| PUT | No | Yes |
| DELETE | No | Yes |

## Versioning

- Media type versioning
  - GitHub
- (Custom) header versioning
  - Microsoft
- URI versioning
  - Twitter
- Request Parameter versioning
  - Amazon

## Media type versioning

```
GET api/helloworld HTTP/1.1
host: localhost
accept: text/plain;v=1.0
```

Returns the result from API version 1.0

```
GET api/helloworld HTTP/1.1
host: localhost
accept: text/plain;v=2.0
```

Returns the result from API version 2.0

## Header Versioning

- It uses HTTP headers to specify the desired version.
- It uses the “Accept” header for content negotiation or uses a custom header (for example, “APIVER” to indicate a version)

```
Accept: application/vnd.example.v1+json
APIVER:v1
```

## URI Versioning

- Using the URI is the most straightforward approach (and most commonly used as well) though it does violate the principle that a URI should refer to a unique resource.

http://api.example.com/v1

http://apiv1.example.com

## Request parameter Versioning

- This approach makes versioning as part of the request parameter, either make one as required query parameter or provide a default version to catch all if this is an optional parameter.

https://api.abc.com/products?**version=v1**

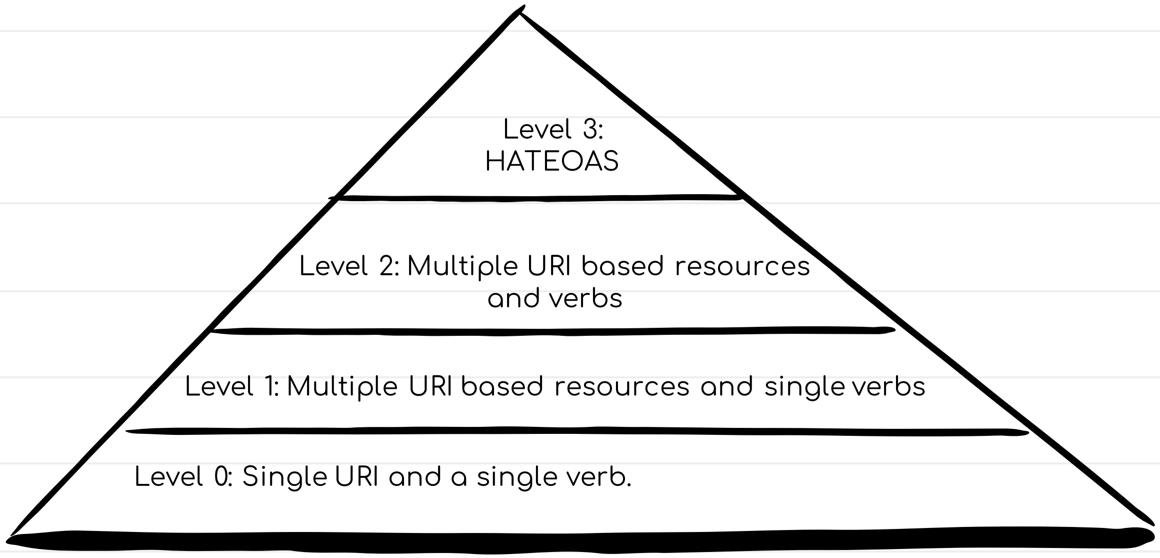
## Richardson Maturity Model

- Leonard Richardson analyzed a hundred different web service designs and divided these designs into four categories.
- These categories are based on how much the web services are REST compliant.
- This model of division of REST services to identify their maturity level – is called Richardson Maturity Model.

## Richardson maturity model

- Richardson used three main factors to decide the maturity of a service:
    - URI
    - HTTP Methods
    - HATEOAS

## MATURITY LEVELS


Level 3:
HATEOAS
Level 2: Multiple URI based resources and verbs
Level 1: Multiple URI based resources and single verbs
Level 0: Single URI and a single verb.

## STATELESS

- It means that the necessary state to handle the request is contained within the request itself and server would not store anything related to the session.
- In REST, the client must include all information for the server to fulfill the request whether as a part of query params, headers or URI.
- Statelessness enables greater availability since the server does not have to maintain, update or communicate that session state.
- There is a drawback when the client need to send too much data to the server, so it reduces the scope of network optimization and requires more bandwidth.

## Client-Server

- REST application should have a client-server architecture.
- A Client is requesting resources and are not concerned with data storage, which remains internal to each server, and server is someone who holds the resources and are not concerned with the user interface or user state.
- They can evolve independently. Client doesn't need to know anything about business logic and server doesn't need to know anything about frontend UI.

## Spring annotations

- Each annotation is meant to handle respective incoming request method type. @GetMapping is used to handle GET type of request method, @PostMapping is used to handle POST type of request method, etc.
    - @GetMapping
    - @PostMapping
    - @PutMapping
    - @DeleteMapping

## @REQUEST MAPPING

- It is used to configure the mapping of web requests.
- It can be applied to class-level and/or method-level in a controller.
- The class-level annotation maps a specific request path or pattern onto a controller.

## @RequestParam

- It is used to bind a web request parameter to the parameter of the handler method.
- The required element of @RequestParam defines whether the parameter value is required or not.
- The value element of @RequestParam can be omitted if the request param and handler method parameter names are same.

## @PATHVARIABLE

- It indicates that a method parameter should be bound to a URI template variable.

- If the method parameter is Map<String, String> then the map is populated with all path variable names and values.

## ResponseEntity<T>

- It is an extension of HttpEntity that adds an HttpStatus status code.
- It represents the whole HTTP response: status code, headers, and body.
- It is used to configure the HTTP response.

## Status codes

| Http Method | path | Status code | description |
|---|---|---|---|
| GET | /api/users | 200 (OK) | All users are fetched. |
| GET | /api/users/{id} | 200 (OK) | One user is fetched. |
| POST | /api/users | 201 (CREATED) | A new user is created. |
| PUT | /api/users/{id} | 200 (OK) | User is updated. |
| DELETE | /api/users/{id} | 204 | User is deleted. |
| ANY | Any??? | 403 | Authorization Error. |

## Best practices

- Never use CRUD function names in URIs
- Use hyphens (-) to improve the readability of URIs
- Do not use underscores ( _ )
- Use lowercase letters in URIs
- Do not use file extensions
- Use query parameters to filter URI collection

## Main points

- REST is defined by architectural constraints. It is able to access information through the ubiquitous URI. Everything on the web is available through a URI.
- Likewise, everything in creation is known through understanding and experience of the Unified Field of Consciousness.
- The Spring framework makes the transition to RESTful web services smooth though the flexible & adaptable design of Spring MVC controllers.
- The Structure of Life is flexible & adaptable.