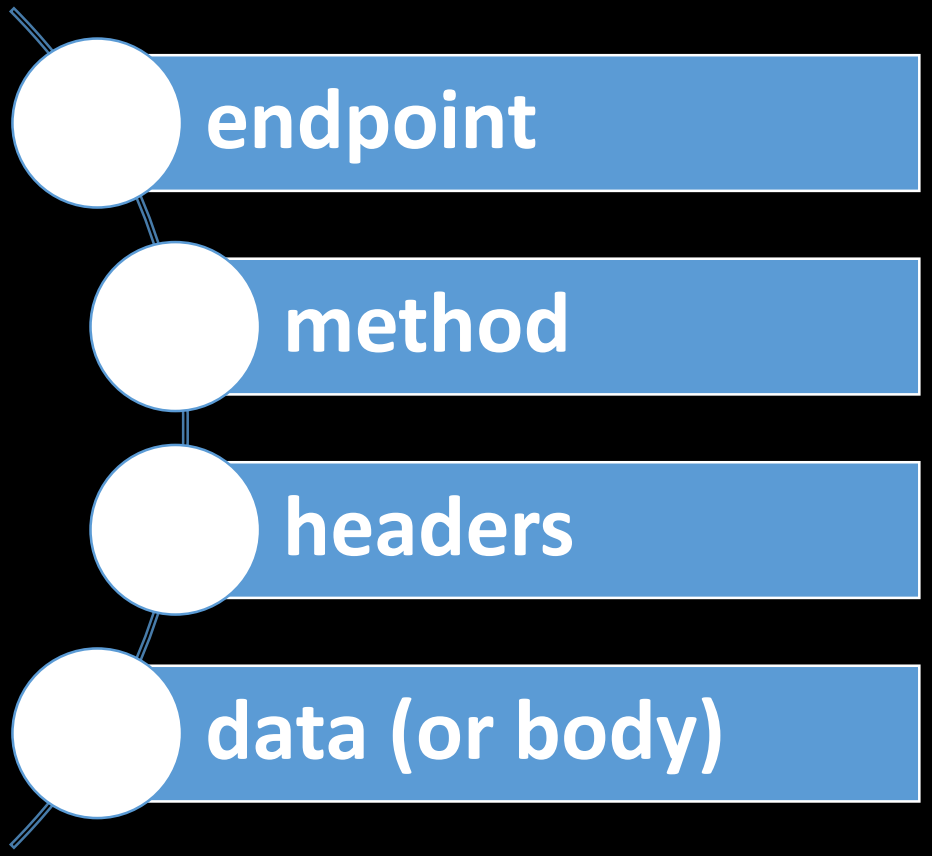
Spring REST

{ REST }



Representational state transfer (**REST**) is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called *RESTful* Web services, provide interoperability between computer systems on the Internet. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations.

REST is a set of principles that define how Web standards, such as HTTP and URIs, are supposed to be used

All our resources have unique identifiers (URL), which gives us an opportunity to logically formulate our requests and structure URLs available for requests
Universality. You can use the same server response data to return back to XML or JSON for software processing or to wrap in a beautiful design for people to see.
Each URL is called a **request** while the data sent back to you is called a **response**.

HTTP REST API METHODS

HTTP Method	Safe	Idempotent
GET	YES	YES
POST	NO	NO
PUT	NO	YES
DELETE	NO	YES
OPTIONS	YES	YES
HEAD	YES	YES

Here is the quick summary of each method and when to use them:

GET: This method is safe and idempotent. It is always used for retrieving the information and doesn't have any side effects.

POST: This method is neither safe nor idempotent. This method is most widely used for creating the resources.

PUT: This method is idempotent. That is the reason instead POST one can use this method for updating the resources. Avoid using POST for updating the resources.

DELETE: As the name implies, this method is used for deleting the resources. But, this method is not idempotent for all the requests.

OPTIONS: This method is not used for any resource manipulation. But, it is useful when client doesn't know the various methods supported for a resource, using this method client can retrieve the various representation of a resource.

HEAD: This method used for querying a resource in the server. It is very similar to GET method, but HEAD has to send request and get the response only in the header. As per HTTP specification, this method should not use body for request and response.

HTTP STATUS CODES

1XX Informational	
100	Continue
101	Switching Protocols
102	Processing

2XX Success	
200	OK
201	Created
202	Accepted
203	Non-authoritative Information
204	No Content
205	Reset Content
206	Partial Content
207	Multi-Status
208	Already Reported
226	IM Used

3XX Redirectional	
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
308	Permanent Redirect

4XX Client Error	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout

4XX Client Error Continued	
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Payload Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
418	I'm a teapot
421	Misdirected Request
422	Unprocessable Entity
423	Locked
424	Failed Dependency
426	Upgrade Required
428	Precondition Required
429	Too Many Requests
431	Request Header Fields Too Large
444	Connection Closed Without Response
451	Unavailable For Legal Reasons
499	Client Closed Request

5XX Server Error	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates
507	Insufficient Storage
508	Loop Detected
510	Not Extended
511	Network Authentication Required
599	Network Connect Timeout Error

HTTP defines various status codes for indicating different meaning to the client. Your REST API could effectively use all the available HTTP codes to help the client route the response accordingly. Here is the list of HTTP status codes for your reference

200 OK – This is response to successful GET, PUT, PATCH or DELETE. This code also be used for a POST that doesn't result in a creation.

201 Created – This status code is response to a POST that results in a creation.

204 No Content – This is a response to a successful request that won't be returning a body (like a DELETE request)

304 Not Modified – Use this status code when HTTP caching headers are in play

400 Bad Request – This status code indicates that the request is malformed, such as if the body does not parse

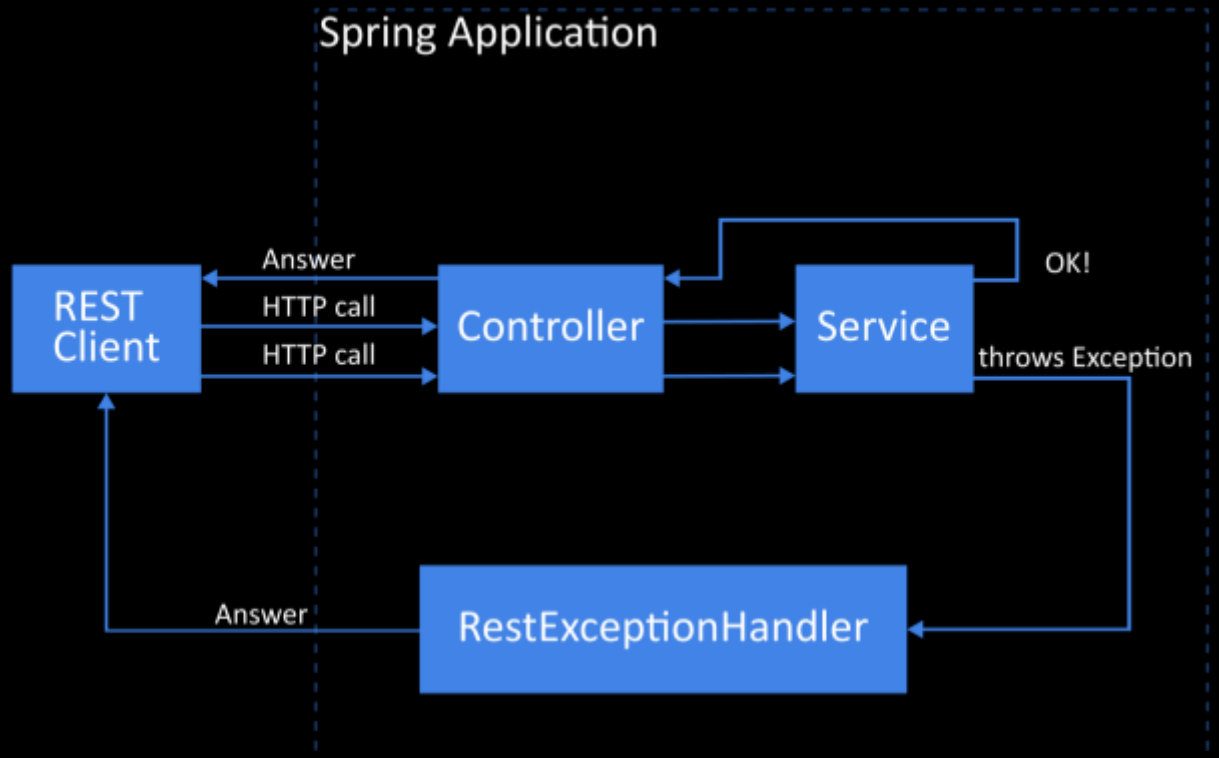
401 Unauthorized – When no or invalid authentication details are provided. Also useful to trigger an auth popup if the API is used from a browser

403 Forbidden – When authentication succeeded but authenticated user doesn't have access to the resource

404 Not Found – When a non-existent resource is requested

405 Method Not Allowed – When an HTTP method is being requested that isn't allowed for the authenticated user

SPRING BOOT ERROR HANDLING



SpringBoot provides a very powerful annotation called **@ControllerAdvide**. This annotation makes our life easy to handle all kinds of exceptions at a central place in our application. We don't need to catch any exception at each method or class separately instead you can just throw the exception from the method and then it will be caught under the central exception handler class annotated by @ControllerAdvide. Any class annotated with @ControllerAdvice will become a controller-advice class which will be responsible for handling exceptions. Under this class, we make use of annotations provided as @ExceptionHandler, @ModelAttribute, @InitBinder.

Exception handling methods annotated with **@ExceptionHandler** will catch the exception thrown by the declared class and we can perform various things whenever we come through the related type exceptions.

@RestController annotation

@RestController

The @RestController annotation was introduced in Spring 4.0 to simplify the creation of RESTful web services. It's a convenience annotation that combines @Controller and @ResponseBody – which eliminates the need to annotate every request handling method of the controller class with the @ResponseBody annotation.

Restful application

A RESTFul application follows the REST architectural style, which is used for designing networked applications. RESTful applications generate HTTP requests performing CRUD (Create/Read/Update/Delete) operations on resources. RESTFul applications typically return data in JSON or XML format.