What does REST stand for?
REST stands for the REpresentational State Transfer, which uses the HTTP protocol to send data from the client to the server, e.g. a book in the server can be delivered to the client using JSON or XML. However, if you are not familiar with REST, I suggest you to first check out the REST API design and development to better understand it.

What is a resource?
A resource is how data is represented in the REST architecture. By exposing entities as the resource, it allows a client to read, write, modify, and create resources using HTTP methods, for example, GET, POST, PUT, DELETE, etc.

What are safe REST operations?
REST API uses HTTP methods to perform operations. Some of the HTTP operations, which doesn't modify the resource at the server, are known as safe operations, including GET and HEAD. On the other hand, PUT, POST, and DELETE are unsafe, because they modify the resource on the server.

What are idempotent operations? Why is idempotency important?
There are some HTTP methods — like GET — that produce the same response no matter how many times you use them, sending multiple GET request to the same URI will result in same response without any side-effect. Hence, this is known as idempotent.

On the other hand, the POST is not idempotent, because if you send multiple POST request, it will result in multiple resource creation on the server, but, again, PUT is idempotent, if you are using it to update the resource.

Even multiple PUT requests can be used to update a resource on a server and will give the same end result. You can take a HTTP Fundamentals course by Pluralsight to learn more about idempotent methods of HTTP protocol and HTTP in general.

Is REST scalable and/or interoperable?
Yes, REST is scalable and interoperable. It doesn't mandate a specific choice of technology either at client or server end. You can use Java, C++, Python, or JavaScript to create RESTful web services and consume them at the client end.

What are the advantages of the RestTemplate? (answer)
The RestTemplate class is an implementation of the Template method pattern in the Spring framework. Similar to other popular template classes, like the JdbcTemplate or JmsTempalte, it also simplifies the interaction with RESTful web services on the client side. You can use it to consume a RESTful web servicer very easily, as shown in this RestTemplate example.

Which HTTP methods does REST use?
REST can use any HTTP methods, but the most popular ones are GET for retrieving a resource, POST for creating a resource, PUt for updating resource, and DELETE for removing a resource from the server.

What is an HttpMessageConverter in Spring REST?
An HttpMessageConverter is a strategy interface that specifies a converter that can convert from and to HTTP requests and responses. Spring REST uses this interface to convert HTTP responses to various formats, for example, JSON or XML.

Each HttpMessageConverter implementation has one or several MIME Types associated with it. Spring uses the "Accept" header to determine the content type that the client is expecting.

It will then try to find a registered HTTPMessageConverter that is capable of handling that specific content-type and use it to convert the response into that format before sending it to the client. If you are new to Spring MVC, see this Spring 5: Beginner to Guru resource to learn the basics.

How to create a custom implementation of the HttpMessageConverter to support a new type of request/responses? (answer)
You just need to create an implementation of the AbstractHttpMessageConverter and register it using the WebMvcConfigurerAdapter#extendMessageConverters() method with the classes that generate a new type of request/response.

Is REST normally stateless? (answer)
Yes, REST API should be stateless, because it is based on HTTP, which is also stateless. A request in REST API should contain all the details required to process it. It should not rely on previous or next requests or some data maintained at the server end, like sessions. The REST specification puts a constraint to make it stateless, and you should keep that in mind while designing your REST API.

What does @RequestMapping annotation do? (answer)
The @RequestMapping annotation is used to map web requests to Spring Controller methods. You can map a request based upon HTTP methods, e.g. GET, POST, and various other parameters.

For example, if you are developing a RESTful web service using Spring, then you can use, produce, and consume property along with media type annotations to indicate that this method is only used to produce or consume JSON, as shown below:

```
@RequestMapping (method = RequestMethod.POST, consumes="application/json")
public Book save(@RequestBody Book aBook) {
  return bookRepository.save(aBook);
}
```

Similarly, you can create other handler methods to produce JSON or XML. If you are not familiar with these annotations, then I suggest you join this Spring MVC For Beginners course on Udemy to learn the basics.

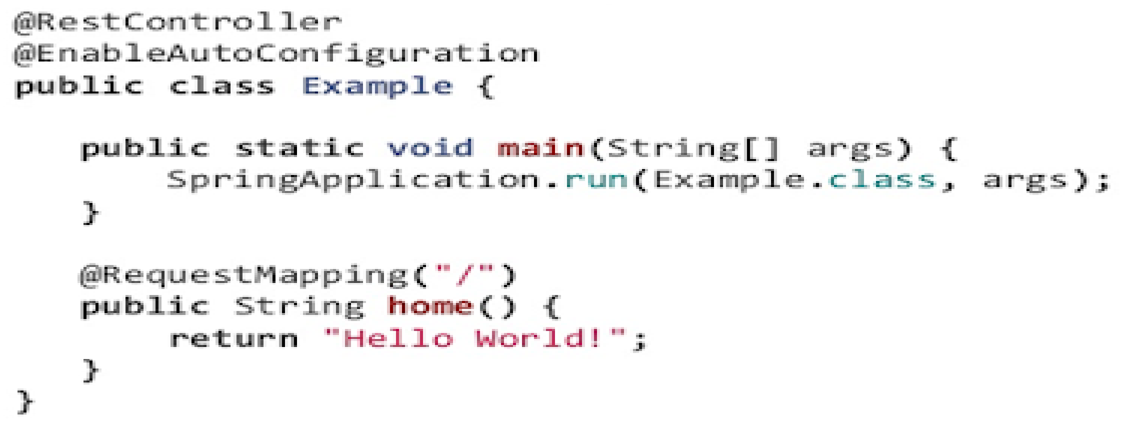
Is @Controller a stereotype? Is @RestController a stereotype? (answer)
Yes, both @Controller and @RestController are stereotypes. The @Controller is actually a specialization of Spring's @Component stereotype annotation. This means that the class annotated with the @Controller will also be automatically detected by the Spring container, as part of the container's component scanning process.

And, the @RestController is a specialization of the @Controller for the RESTful web service. It not only combines the @ResponseBody and @Controller annotations, but it also gives more meaning to your controller class to clearly indicate that it deals with RESTful requests.

Your Spring Framework may also use this annotation to provide some more useful features related to REST API development in future.

What is the difference between @Controller and @RestController? (answer)
There are many differences between the @Controller and @RestController annotations, as discussed in my earlier article (see the answer for more!), but the most important one is that with the @RestController you get the @ResponseBody annotation automatically, which means you don't need to separately annotate your handler methods with the @ResponseBody annotation.

This makes the development of RESTful web services easier using Spring. You can see here to learn more about Spring Boot and how it can help you to create Spring MVC based web applications.



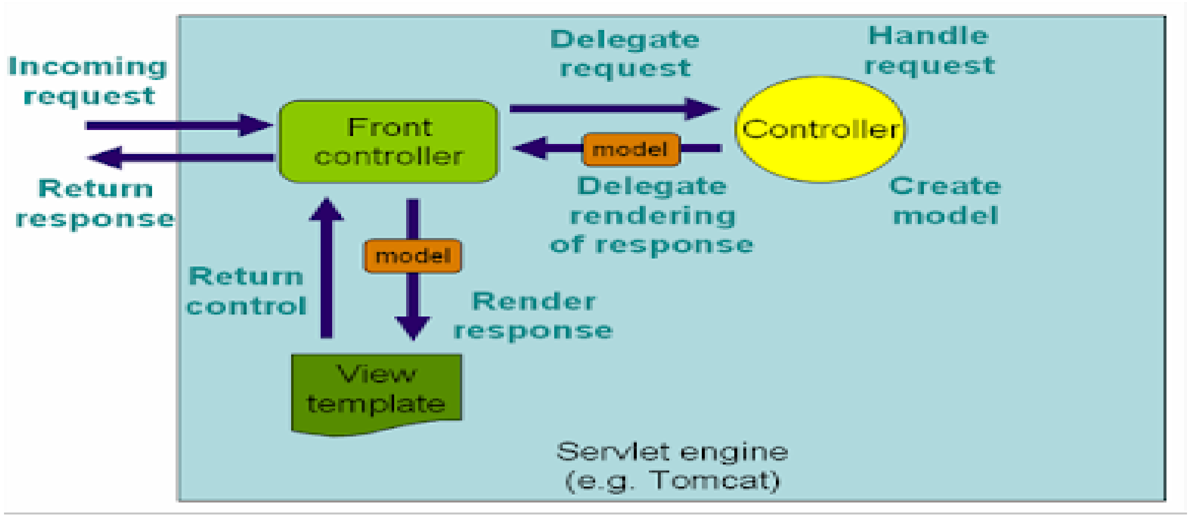
```
@RestController
@EnableAutoConfiguration
public class Example {

    public static void main(String[] args) {
        SpringApplication.run(Example.class, args);
    }

    @RequestMapping("/")
    public String home() {
        return "Hello World!";
    }
}
```

When do you need @ResponseBody annotation in Spring MVC? (answer)
The @ResponseBody annotation can be put on a method to indicate that the return type should be written directly to the HTTP response body (and not placed in a Model, or interpreted as a view name).

For example:
```
@RequestMapping(path = "/hello", method = RequestMethod.PUT)
@ResponseBody
public String helloWorld() {
  return "Hello World";
}
```

Alternatively, you can also use the @RestController annotation instead of the @Controller annotation. This will remove the need for using @ResponseBody because, as discussed in the previous answer, it comes automatically with the @RestController annotation.

What does @PathVariable do in Spring MVC? Why it's useful in REST with Spring? (answer)
This is one of the useful annotations from Spring MVC that allows you to read values from the URI, like query parameter. It's particularly useful in case of creating RESTful web service using Spring, because, in REST, resource identifiers are part of the URI. This question is normally asked by experienced Spring MVC developers with 4 to 6 years of experience.



What is the HTTP status return code for a successful DELETE statement? (answer)
There is no strict rule about what status code your REST API should return to after a successful DELETE. It can return 200 Ok or 204 No Content.
In general, if the DELETE operation is successful, the response body is empty, return 204. If the DELETE request is successful and the response body is NOT empty, return 200.

What does CRUD mean? (answer)
CRUD is a short form of Create, Read, Update, and Delete. In REST API, the POST is used to create a resource, GET is used to read a resource, PUT is used to updated a resource, and DELETE is used to remove a resource from the server.

Where do you need @EnableWebMVC? (answer)
The @EnableWebMvc annotation is required to enable Spring MVC when Java configuration is used to configure Spring MVC instead of XML. It is equivalent to **<mvc: annotation-driven>** in an XML configuration.
It enables support for the @Controller-annotated classes that use @RequestMapping to map incoming requests to handler methods that are not already familiar with Spring's support for Java configuration.

When do you need @ResponseStatus annotation in Spring MVC? (answer)
This is a good question for 3 to 5 years as an experienced Spring developer. The @ResponseStatus annotation is required during error handling in Spring MVC and REST. Normally, when an error or exception is thrown at the server side, the web server returns a blanket HTTP status code 500 — Internal server error.

This may work for a human user but not for REST clients. You need to send them the proper status code, like 404, if the resource is not found. That's where you can use the @ResponseStatus annotation, which allows you to send custom HTTP status codes along with proper error message in case of an exception.

In order to use it, you can create custom exceptions and annotate them using the @ResponseStatus annotation and proper HTTP status code and reason.

When such exceptions are thrown from the controller's handler methods and not handled anywhere else, then the appropriate HTTP response with the proper HTTP status code is sent to the client.

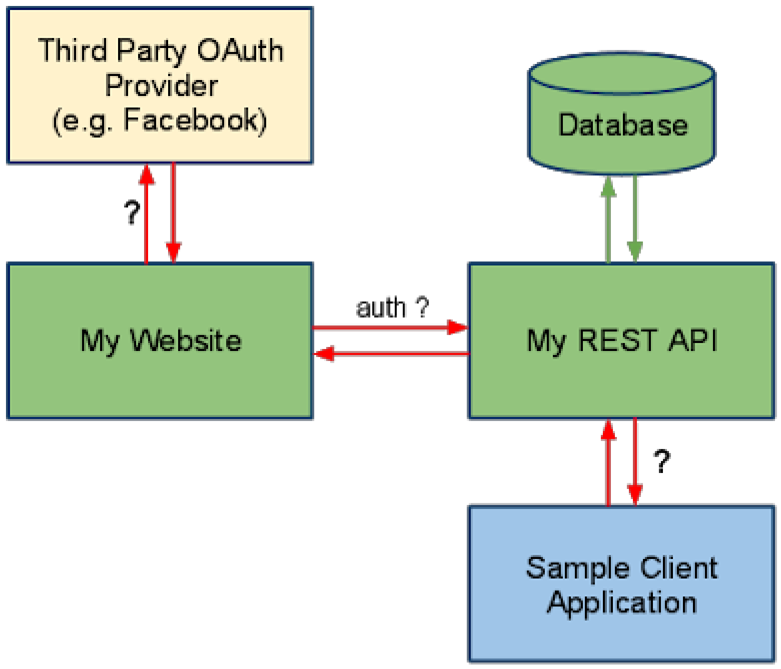
For example, if you are writing a RESTful web service for a library that provides book information, then you can use @ResponseStatus to create an exception that returns the HTTP response code 404 when a book is not found instead of the Internal Server Error (500), as shown below:

```
@ResponseStatus(value=HttpStatus.NOT_FOUND, reason="No such Book")  // 404
 public class BookNotFoundException extends RuntimeException {

    // ...

 }
```

If this exception is thrown from any handler method, then the HTTP error code 404 with the reason "No such Book" will be returned to the client.

Is REST secure? What can you do to secure it? (answer)
This question is mostly asked by experienced Java programmers with about 2 to 5 years of experience with both REST and Spring. Security is a broad term; it could mean security of message, which is provided by encryption or access restriction that are provided using authentication and authorization. REST is normally not secure, but you can secure it by using Spring Security.

At the very least, you can enable the HTTP basic authentication by using HTTP in your Spring Security configuration file. Similarly, you can expose your REST API using HTTPS, if the underlying server supports HTTPS.


Third Party OAuth Provider (e.g. Facebook)
Database
?
My Website
auth ?
My REST API
?
Sample Client Application

Does REST work with transport layer security (TLS)? (answer)
Transport Layer Security (TLS) is used for secure communication between the client and server. It is the successor of SSL (Secure Socket Layer). Since HTTPS can work with both SSL and TLS, REST can also work with TLS.

Actually, in REST, it is up to the server to implement security protocols. The same RESTful web service can be accessed using HTTP and HTTPS, if the server supports SSL.
If you are using Tomcat, you can learn more about how to enable SSL in Tomcat.