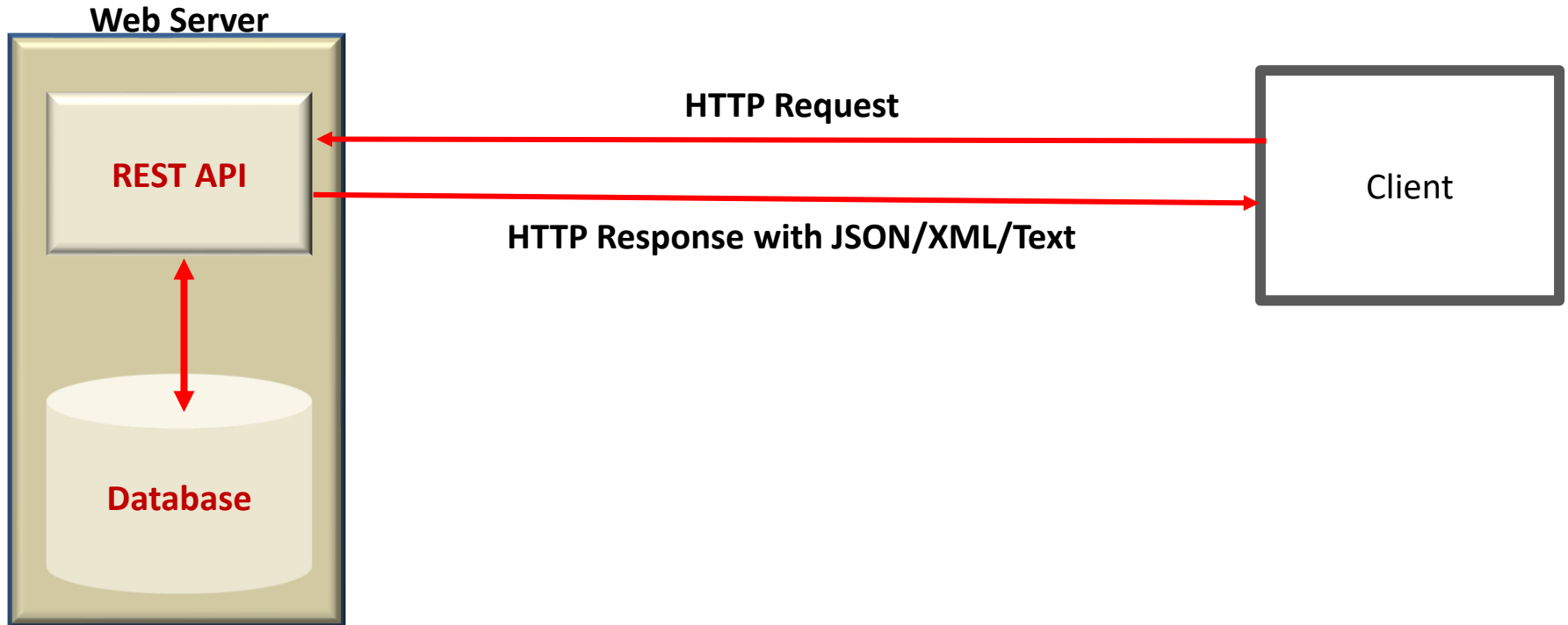


What is REST

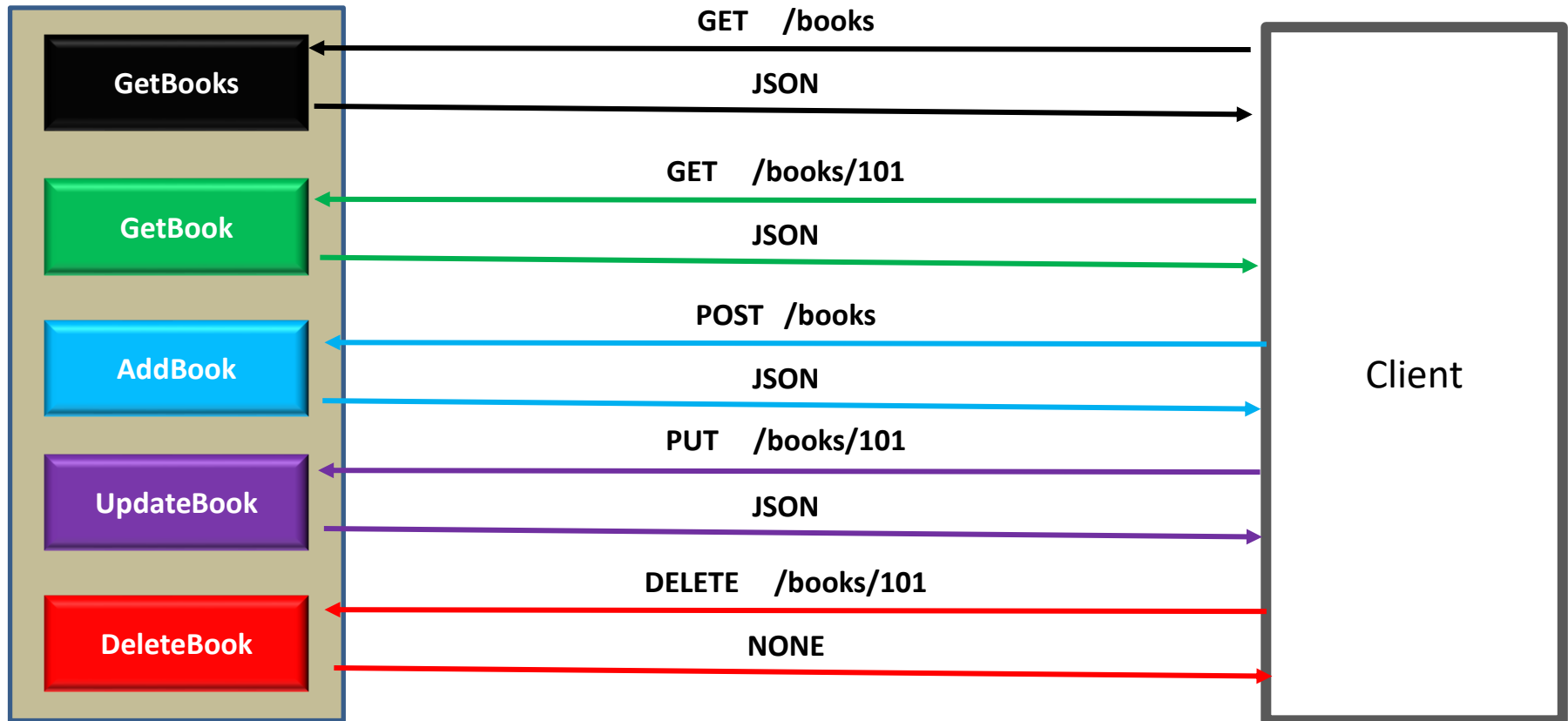


- ❑ A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services.
- ❑ REST stands for representational state transfer and was created by computer scientist Roy Fielding.
- ❑ When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint. This information, or representation, is delivered in one of several formats via HTTP
- ❑ REST is Stateless, meaning no client information is stored between get requests and each request is separate and unconnected.

REST API



REST API Methods vs. Http Methods



Maven Dependencies



```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!-- To load application automatically after change -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <optional>true</optional>
</dependency>
<
```

Spring Boot will automatically add static web resources located within any of the following directories, which are either in classpath or in root directory of context.

- ☐ **/META-INF/resources/**
- ☐ **/resources/**
- ☐ **/static/**
- ☐ **/public/**

- ❑ It is possible to create a RESTful Web Service using `@RestController` and `@RequestMapping` annotations.
- ❑ Spring uses Jackson JSON library to automatically convert Java classes to JSON.
- ❑ Building a rest controller is similar to MVC controller except that it sends JSON and not HTML.
- ❑ Annotation `@RestController` marks the class as a controller where every method returns a domain object instead of a view. It's shorthand for `@Controller` and `@ResponseBody` rolled together.

- ❑ A URI Template is a URI-like string, containing one or more variable names
Variables are enclosed in {}
- ❑ We can use `@PathVariable` annotation on a method argument to bind it to the value of a URI template variable

```
@GetMapping("/books/{id}")  
public String findBook(@PathVariable("id") String bookId)  
{  
}  
  
@GetMapping(value="/books/{id}/chapters/{chno}")  
public String findChapter(@PathVariable("id") String id,  
                          @PathVariable("chno") String chno)  
{  
}
```

@RequestParam



- ❑ Annotation which indicates that a method parameter should be bound to a web request parameter.
- ❑ If the method parameter is `Map<String, String>` or `MultiValueMap<String, String>` and a parameter name is not specified, then the map parameter is populated with all request parameter names and values.
- ❑ Optional attributes - `defaultValue`, `name`, `required`
- ❑ When `defaultValue` is provided, it implicitly sets `required` to false.

```
@GetMapping("/books")
public String findBook(@RequestParam("id") String bookId)
{ }
```

```
@GetMapping("/chapter")
public String findChapter(
    @RequestParam(name="id") int id,
    @RequestParam(name="chno", required=false, defaultValue="1")
    int chno)
{ }
```


Spring HTTP Message Converters



- ☐ Message converters convert object to the required format.
- ☐ These message converters need extra libraries like Jackson JSON to be in classpath to work.

- ❑ If the client's request has **Accept** header set to **application/json** then if Jackson JSON library is in the application's classpath, the object returned from the handler method is given to **MappingJacksonHttpMessageConverter** for conversion into a JSON representation to be returned to the client.
- ❑ On the other hand, if the request header indicates that the client prefers **text/xml**, then **Jaxb2RootElementHttpMessageConverter** is tasked with producing an XML response to the client.

Request Mapping Annotations



These are HTTP method specific shortcut variants of `@RequestMapping`:

- ✓ `@GetMapping`
- ✓ `@PostMapping`
- ✓ `@PutMapping`
- ✓ `@DeleteMapping`

Request Mapping Annotations



```
@RestController
@RequestMapping("/persons")
class PersonController {
    @GetMapping("/{id}")
    public Person getPerson(@PathVariable Long id) {
        // ...
    }

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public void add(@RequestBody Person person) {
        // ...
    }
}
```

RestController Example



```
@RestController
@RequestMapping(value = "/api/jobs")
public class JobsRestController {
    @Autowired
    private JobRepository jobRepo;
```

Http Response Codes



Code	Meaning
200	Ok
201	Created
204	No Content
400	Bad Request
401	Unauthorized (unauthenticated)
403	Forbidden
404	Not Found
405	Method not allowed
500	Internal server error

ResponseStatusException



- ❑ When a method in REST API throws this exception, it indicates some error.
- ❑ We can create an instance of it providing an ***HttpStatus*** and optionally a *reason* for error.

```
ResponseStatusException(HttpStatus status)
```

```
ResponseStatusException  
    (HttpStatus status, String reason)
```

```
ResponseStatusException  
    (HttpStatus status, String reason, Throwable cause )
```

```
new ResponseStatusException  
    (HttpStatus.NOT_FOUND, "Product Id Not Found");
```

HttpStatus Constants



ACCEPTED	202 Accepted.
BAD_REQUEST	400 Bad Request.
CREATED	201 Created.
FORBIDDEN	403 Forbidden.
INTERNAL_SERVER_ERROR	500 Internal Server Error.
METHOD_NOT_ALLOWED	405 Method Not Allowed.
NO_CONTENT	204 No Content.
NOT_FOUND	404 Not Found.
OK	200 OK.
UNAUTHORIZED	401 Unauthorized.

Enabling CORS



- ❑ Cross-Origin Resource Sharing (CORS) allows clients such as JavaScript to consume data via REST APIs.
- ❑ Often, the host that serves the JS (client) (e.g. client.com) is different from the host that serves the data (REST API) (e.g. api.server.com).
- ❑ In such a case, CORS enables cross-domain communication.
- ❑ **@CrossOrigin** annotation is used to enable CORS for a method or all methods of a Class.
- ❑ If **@CrossOrigin** is used with class then CORS is enabled for all methods in the class.
- ❑ **@CrossOrigin** defaults are:
 - ✓ All origins are allowed.
 - ✓ The HTTP methods allowed are those specified in the *@RequestMapping* annotation of the method
 - ✓ The time that the preflight response is cached (*maxAge*) is 30 minutes.

@CrossOrigin Annotation



- ❑ Annotation for permitting cross-origin requests on specific handler classes and/or handler methods.
- ❑ When applied at the class level, the same @CrossOrigin configuration is applied to all the @RequestMapping methods. If the @CrossOrigin annotation is specified at both the class level and the method level, Spring will derive the CORS configuration by combining attributes from both annotations.

String allowCredentials

Whether the browser should send credentials, such as cookies along with cross domain requests, to the annotated endpoint.

String[] allowedHeaders

The list of request headers that are permitted in actual requests, possibly "*" to allow all headers.

long maxAge

The maximum age (in seconds) of the cache duration for preflight responses.

RequestMethod[] methods

The list of supported HTTP request methods.

String[] origins

A list of origins for which cross-origin requests are allowed.

@CrossOrigin Example



```
@CrossOrigin(origins = "http://localhost:9999")
@GetMapping("/products")
public List<Product> getProducts()
{
}
```

```
@CrossOrigin(origins = "http://example.com", maxAge = 3600)
@RestController
@RequestMapping("/account")
public class AccountController {
}
```

```
@CrossOrigin
    (origins = {"http://abc.com", "http://xyz.com" },
     methods = { RequestMethod.GET, RequestMethod.POST })
public List<Product> getProducts() {}
```