RESTFUL API & SPRING BOOT

Intro REST API and Getting Started with **Spring Boot**

RESTFUL API

Before REST, What is API ?

What is REST ?

REST stands fo **RE**presentational **S**tate **T**ransfer.

In **REST based architecture** everything is a **resource**.

It means when a RESTful API is called, the server willl transfer to the client a representation of the state of the requested resource.

RESTFUL API ≠ HTPP API

REST architecture and HTTP 1.1 protocol **are independent of each other** , but the HTTP 1.1 protocol was built to be the ideal protocol to follow the principles and constraints of REST

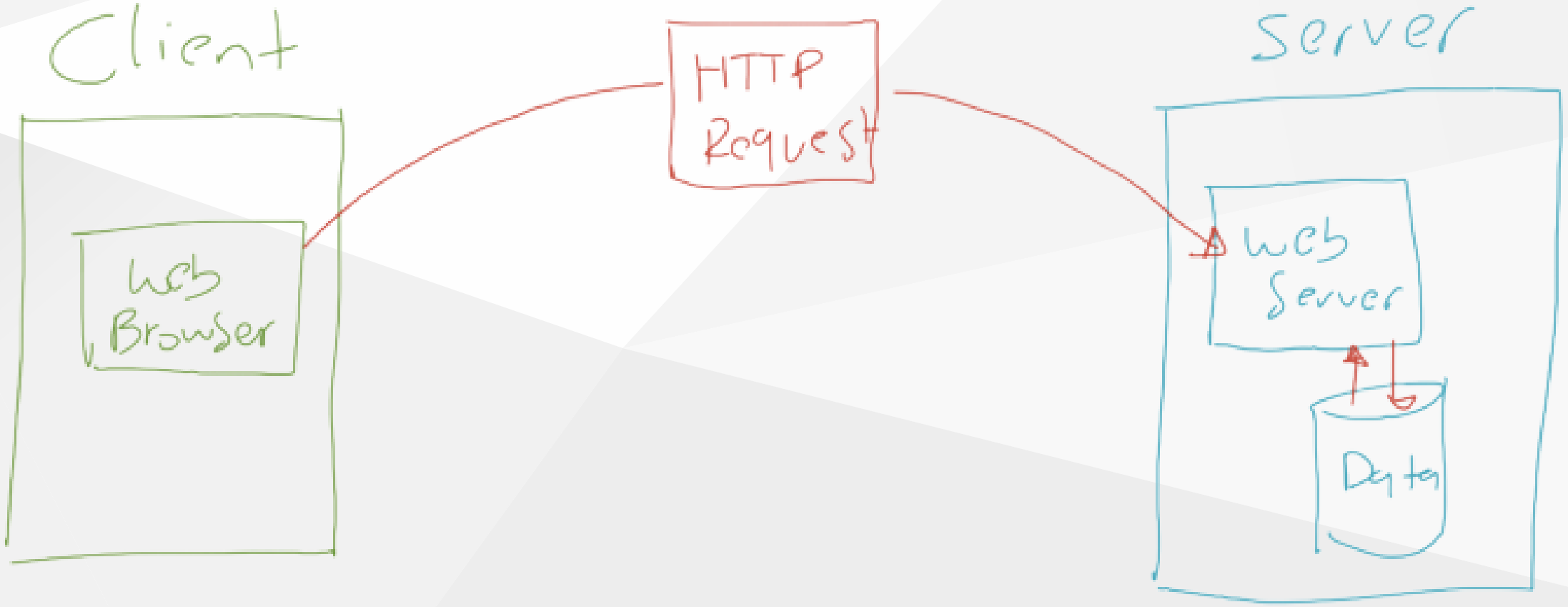
HTTP

HTTP stands for **H**yper**t**ext **T**ransfer **P**rotocol

HTPP is **a protocol** that transmite hypermedia document, lika HTML, JavaScript, CSS, Image, Audio. Video etc.

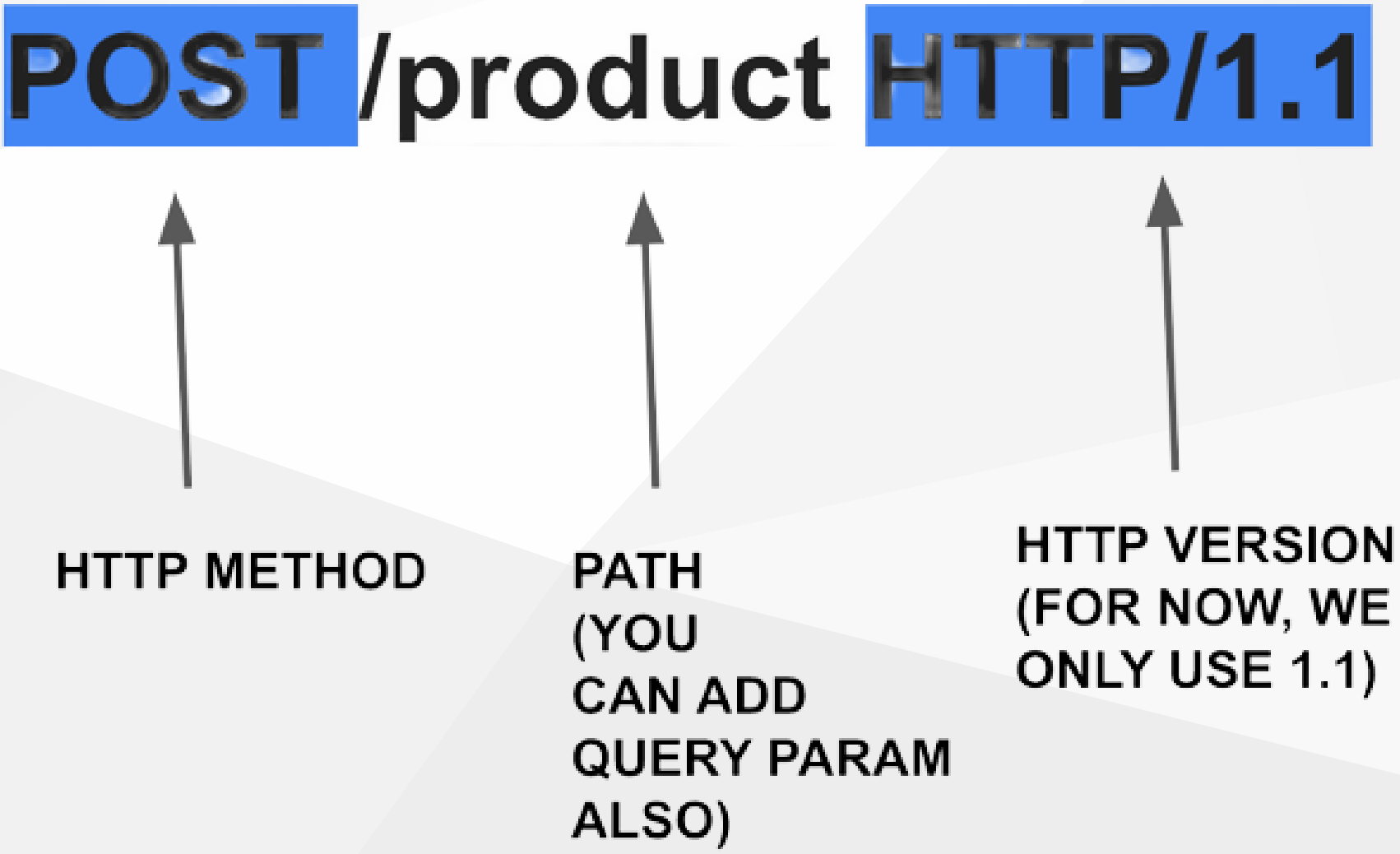
HTTP in earlier is designed for communicating between Web Browser dan Web Server, but now we use it in many applications.

HTPP Flow

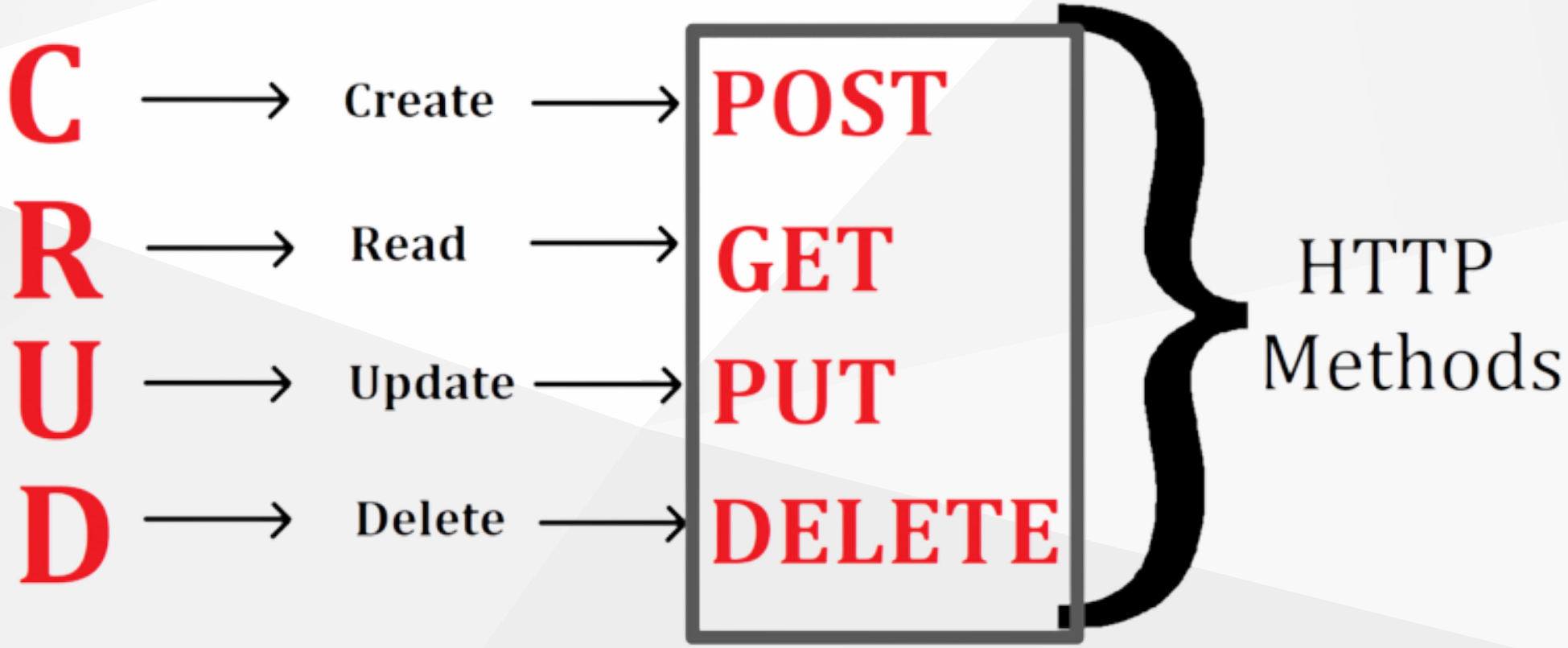


HTTP Request Example

Request Line

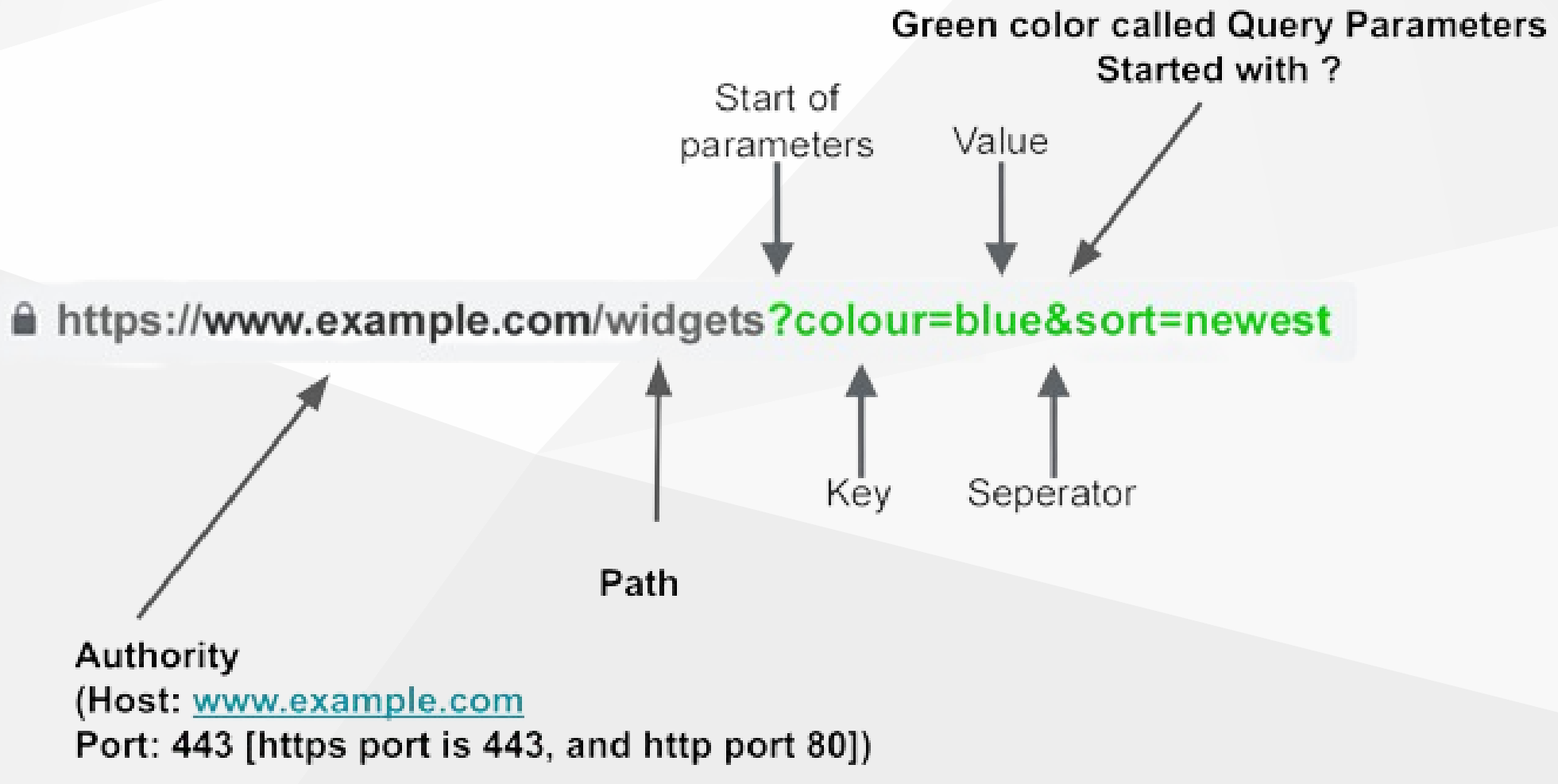


HTTP Common Method



HTTP Other Standard Method

URL

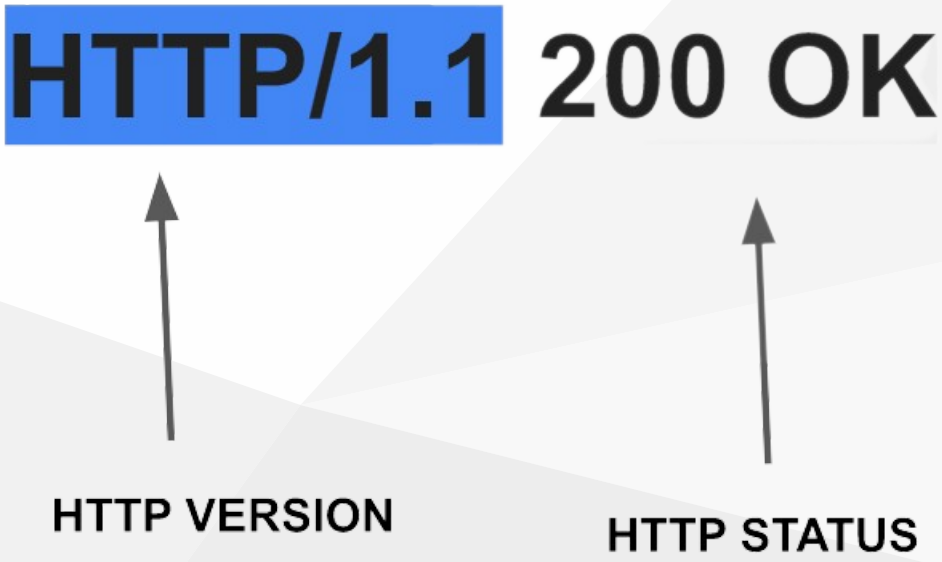

Green color called Query Parameters
Started with ?
Start of parameters
Value
https://www.example.com/widgets?colour=blue&sort=newest
Key
Seperator
Path
Authority
(Host: www.example.com
Port: 443 [https port is 443, and http port 80])

HTTP Headers (Example)

HTTP Header	Description
`Host`	Authority on URL (must be exist in HTTP 1.1)
`Content-Type`	Data type of HTTP Body
`Authorization`	Credential for authentication
`Accept`	Data type that client want to accept

HTTP Response Example

HTTP Response Status Line



List Of HTTP Status Codes

100 Continue
101 Switching Protocols
103 Early Hints
200 OK
201 Created
202 Accepted
203 Non-Authoritative Information
204 No Content
205 Reset Content
206 Partial Content
300 Multiple Choices
301 Moved Permanently
302 Found
303 See Other
304 Not Modified
307 Temporary Redirect
308 Permanent Redirect

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
409 Conflict
410 Gone
411 Length Required
412 Precondition Failed
413 Payload Too Large
414 URI Too Long
415 Unsupported Media Type
416 Range Not Satisfiable
417 Expectation Failed
418 I'm a teapot

422 Unprocessable Entity
425 Too Early
426 Upgrade Required
428 Precondition Required
429 Too Many Requests
431 Request Header Fields Too Large
451 Unavailable For Legal Reasons
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

HTTP Status Code

1XX
INFORMATIONAL

2XX
SUCCESS

3XX
REDIRECTION

4XX
CLIENT ERROR

5XX
SERVER ERROR

REST Path URL Restriction

`/user/1/balance` (balance owned by user id 1)
`/balance/1` (balance with id 1)
Remember, everything in REST is **resource, RPC -> function method only POST or GET**

Purpose	Method	Incorrect	Correct
Retrieves a list of users	GET	/getAllCars	/users
Create a new user	POST	/createUser	/users
Delete a user	DELETE	/deleteUser	/users/10
Get balance of user	GET	/getUserBalance	/users/11/balance

Before Spring Boot

Introduce: Build Automation Tools

Build Automation Tools is application tool that help you manage java application.

In Java, there are **Maven** (`pom.xml`) and **Gradle** (`Groovy / Kotlin`).
What they do?

1. Automate manage your dependencies
2. Automate manage your build application to binary
3. Automate manage your testing
4. Etc.

Common Lifecycle in Maven

Simple, we can call lifecycle = task

- `validate`: check if all information necessary for the build is available
- `compile`: compile the source code
- `test-compile`: compile the test source code
- `test`: run unit tests
- `package`: package compiled source code into the distributable format (jar, war, ...)
- `integration-test`: process and deploy the package if needed to run integration tests
- `install`: install the package to a local repository
- `deploy`: copy the package to the remote repository

Common CLI in Spring Boot

`mvn spring-boot:run` -> to run spring boot

`mvn clean install` -> to clean and install all related local dependencies

`mvn clean verify` -> to clean and verify all test related local dependencies

SPRING BOOT

What is Spring Boot ?

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can **"just run"**.

We take an **opinionated** view of the Spring platform and third-party libraries so you can **get started with minimum fuss**. Most Spring Boot applications need minimal Spring configuration.

Type Framework

Opinionated -> Complete (Web Server, Controller, Data Access)

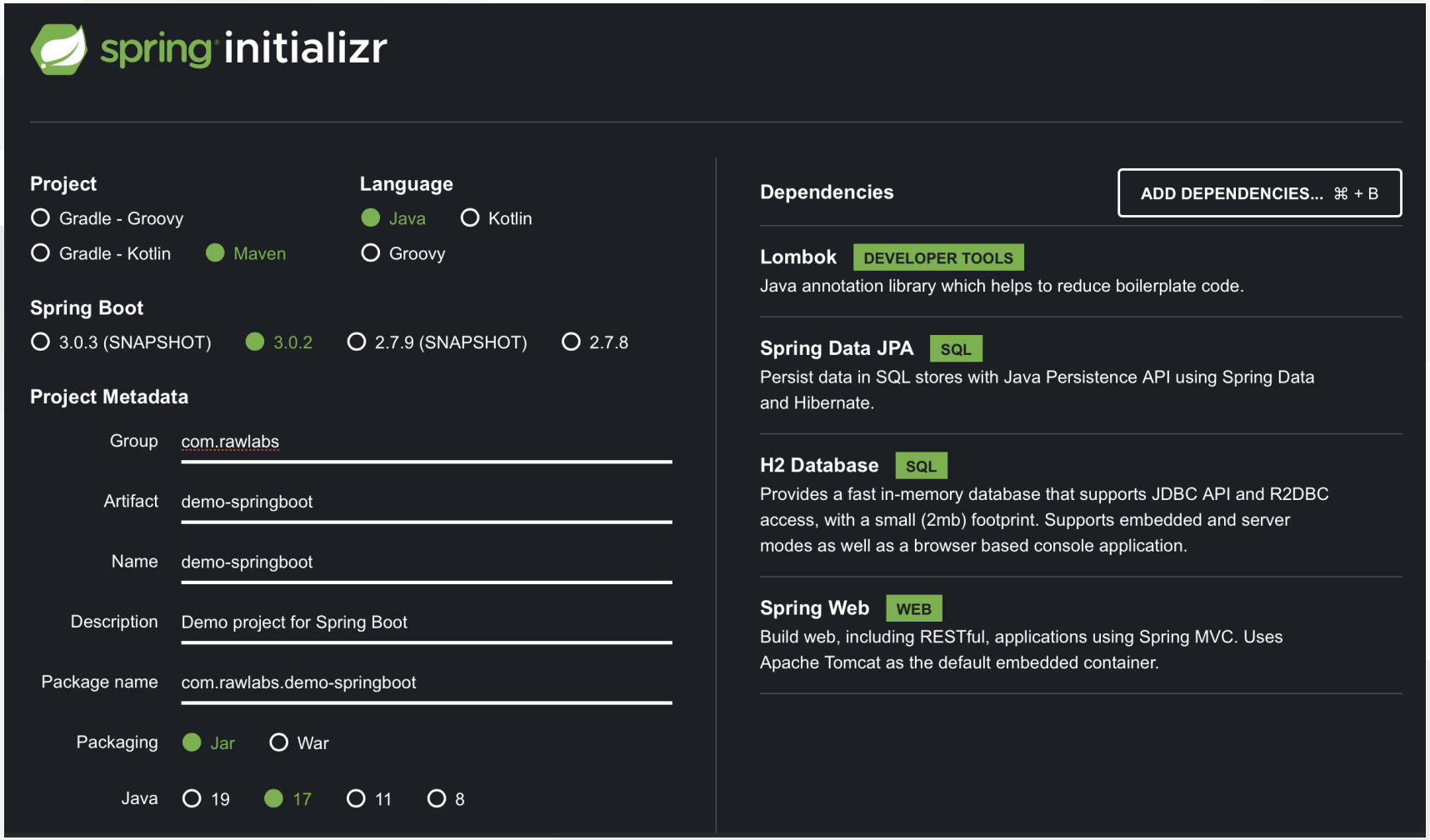
Non Opinionated -> Modular (Web Server)

Why Spring Boot?

You can choose Spring Boot because of the features and benefits it offers as given here:

- It provides a flexible way to configure Java Beans, XML configurations, and Database Transactions.
- It provides a powerful batch processing and manages REST endpoints.
- In Spring Boot, everything is auto configured; no manual configurations are needed.
- It offers annotation-based spring application
- Eases dependency management
- It includes Embedded Servlet Container

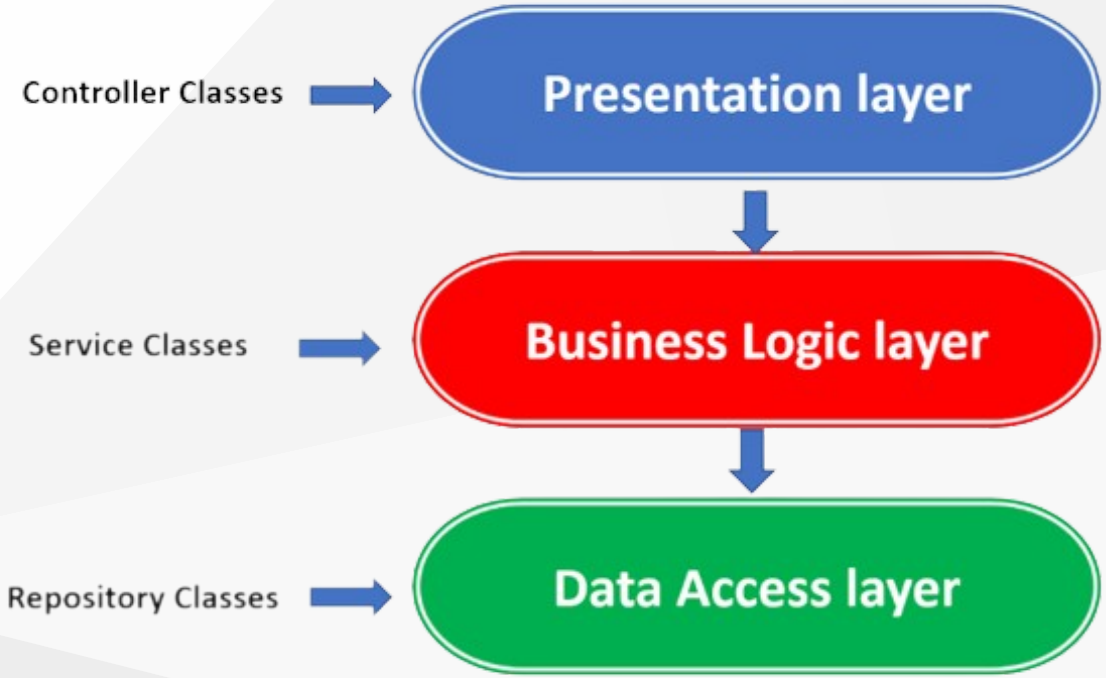
Getting Started with Spring Boot



Spring Boot
Architecture

Note : Get used to making it from the **bottom layer**.

Create the model or entity that represent the table of database and then create the **data access layer**.



Spring Boot Architecture

The `SpringApplication` class provides a convenient way to **bootstrap** a Spring application that will be started from a `main()` method. In many situations you can just delegate to the static `SpringApplication.run` method:

```
@SpringBootApplication
public class MyApplication {
public static void main(String[] args) {
    SpringApplication.run(MyApplication.class, args);
    }
}
```

Project Structure

```
.
└── com.rawlabs.demospringboot/
    ├── controller/          # Presentation layer
    │   └── ...
    ├── domain/
    │   ├── dao/             # Representation of Table
    │   │   └── ...
    │   └── dto/             # POJO Class
    │       └── ...
    ├── repository/          # Data access layer -> to database
    │   └── ...
    ├── service/             # Business logic layer
    │   └── ...
    └── DemoSpringbootApplication.java
```

Data Access Object (DAO)

```java
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "book")
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(name = "title", nullable = false)
    private String title;

    @Column(name = "price", nullable = false)
    private Integer price;

}
```

Annotations

Annotation	Description
`@Data`	Lombok setter and getter generator
`@Builder`	Lombok builder class
`@NoArgsConstructor`	Lombok generate no arguments constructor
`@AllArgsConstructor`	Lombok generate all arguments constructor
`@Entity`	Annotate that class being used for entity
`@Table`	Represent the table on database
`@Id`	Represent ID primary key
`@GeneratedValue`	Generate value automatically
`@Column`	Represent column name on database

Data Transfer Object (DTO)

```java
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class BookDto {

    private String title;

    private Integer price;

}
```

Annotations

Annotation	Description
`@Data`	Lombok setter and getter generator
`@Builder`	Lombok builder class
`@NoArgsConstructor`	Lombok generate no arguments constructor
`@AllArgsConstructor`	Lombok generate all arguments constructor

Repository - Data Access Layer

```
@Repository
public interface ProductRepository extends JpaRepository<Book, Long> {

}
```

The **annotations**

Annotation	Description
`@Repository`	Annotate that class being used for data access layer

Service - Bussiness Logic Layer

```java
@Service
public class BookService {
    private final BookRepository bookRepository;

    @Autowired
    public BookService(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    public Book save(BookDto request) {
        Book book = Book.builder()
                .title(request.getTitle())
                .price(request.getPrice())
                .build();
        return bookRepository.save(book);
    }

    public List<Book> getBooks() {
        return bookRepository.findAll();
    }
}
```

Annotations

Annotation	Description
`@Service`	Service class
`@Autowired`	Enabling spring boot to inject the object dependency implicitly

Controller - Presentation Layer

It comprises of all the logic related to User Interface (at this context is **HTTP Request** and **HTTP Response**).

```
@RestController
@RequestMapping("/v1/book")
public class BookController {
    private final BookService bookService;

    @Autowired
    public BookController(BookService bookService) {
        this.bookService = bookService;
    }

    @GetMapping(value = "", produces = MediaType.APPLICATION_JSON_VALUE)
    public List<Book> getBooks() {
        return bookService.getBooks();
    }

}
```

Annotations

Annotation	Description
`@RestController`	Controller class
`@RequestMapping`	Path naming
`@GetMapping`	`GET` Method
`@PostMapping`	`POST` Method
`@PutMapping`	`PUT` Method
`@DeleteMapping`	`DELETE` Method

JPA and Hibernate

Object-Relational Mapping **(ORM)** is the process of converting Java objects to database tables. In other words, this allows us to interact with a relational database without any SQL. **The Java Persistence API (JPA) is a specification that defines how to persist data in Java applications**. The primary focus of JPA is the ORM layer.

Hibernate is one of the most popular Java ORM frameworks in use today. Its first release was almost twenty years ago, and still has excellent community support and regular releases. Additionally, **Hibernate is a standard implementation of the JPA specification**, with a few additional features that are specific to Hibernate. Let's take a look at some core features of JPA and Hibernate.

JPA Dependecies

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
</dependency>
```

Spring JPA and Datasource Properties

```
spring.datasource.url=jdbc:postgresql://localhost:5432/demo
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.hibernate.ddl-auto-update
spring.jpa.show-sql=true
```


THANK YOU