



QU'EST CE QUE SPRING BOOT WEB

`Spring boot web` est un module de `Spring boot` qui permet de créer des applications web.

Sans lui, spring boot ne permet pas de créer des applications web.

DÉPENDANCES

Pour utiliser **Spring boot web**, il faut ajouter les dépendances suivantes dans le fichier **pom.xml**:

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

LES CONTROLLERS

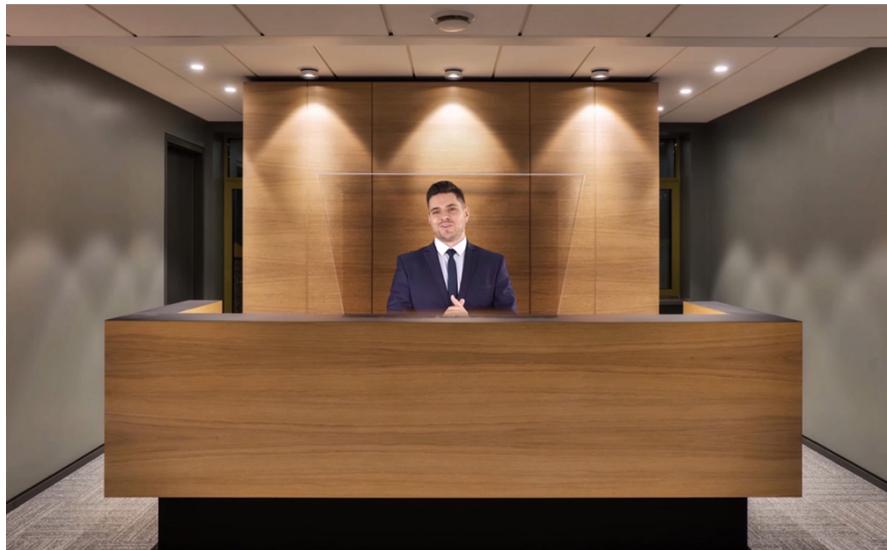
Les controllers sont des classes qui permettent de gérer les requêtes HTTP.

Elles ont les rôles suivants :

- Définir les routes
- Récupérer les informations de la requête
- Envoyer une réponse

SON RÔLE

Être l'accueil de l'application web.



CRÉATION D'UN CONTROLLER

Pour créer un controller, il faut créer une classe annoté par
`org.springframework.web.bind.annotation.RestController`.

Par convention, le nom de la classe doit se terminer par `Controller`.

```
@RestController
public class HelloController {
    // ...
}
```

@RESTCONTROLLER

`@RestController` est une annotation qui permet de définir une classe comme controller.

C'est une annotation qui hérite de `@Component`.

La classe doit être dans le même package (ou package enfant) que la classe principale.

@REQUESTMAPPING

Dans un controller, il est possible de définir les routes.

`@RequestMapping` est une annotation qui permet de définir une route.

Dans l'exemple suivant, la route est `/hello` retourne le message `Hello World`.

```
1 @RestController
2 public class HelloController {
3     @RequestMapping("/hello")
4     public String hello() {
5         return "Hello world";
6     }
7 }
```

SPÉCIFIER LE TYPE DE REQUÊTE

Une méthode annotée avec `@RequestMapping` peut être appelée pour n'importe `verbe`(GET, POST, PUT, ...).

Pour spécifier le type de requête, il faut utiliser les annotations de type `XXXMapping`:

- `GetMapping` pour les requêtes `GET`
- `PostMapping` pour les requêtes `POST`
- `PutMapping` pour les requêtes `PUT`
- `DeleteMapping` pour les requêtes `DELETE`

EXAMPLE

```
@RestController
public class HelloController {
    @GetMapping("/hello")
    public String hello() {
        return "Hello Getter";
    }

    @PostMapping("/hello")
    public String helloPost() {
        return "Hello Poster";
    }
}
```

@REQUESTMAPPING SUR UN CONTROLLER

`@RequestMapping` peut être utilisé sur un controller pour définir une route par défaut.

Dans l'exemple suivant, la route par défaut est `/parent`.

```
1 @RestController
2 @RequestMapping("/parent")
3 public class HelloController {
4     // Appelé pour GET /parent/hello
5     @GetMapping("/hello")
6     public String hello() {
7         return "Hello Getter";
8     }
9
10    // Appelé pour POST /parent/hello
11    @PostMapping("/hello")
12    public String helloPost() {
13        return "Hello Poster";
14    }
15 }
```

@REQUESTMAPPING SUR UN CONTROLLER

`@RequestMapping` peut être utilisé sur un controller pour définir une route par défaut.

Dans l'exemple suivant, la route par défaut est `/parent`.

```
1 @RestController
2 @RequestMapping("/parent")
3 public class HelloController {
4     // Appelé pour GET /parent/hello
5     @GetMapping("/hello")
6     public String hello() {
7         return "Hello Getter";
8     }
9
10    // Appelé pour POST /parent/hello
11    @PostMapping("/hello")
12    public String helloPost() {
13        return "Hello Poster";
14    }
15 }
```

11.1

@REQUESTMAPPING SUR UN CONTROLLER

`@RequestMapping` peut être utilisé sur un controller pour définir une route par défaut.

Dans l'exemple suivant, la route par défaut est `/parent`.

```
1 @RestController
2 @RequestMapping("/parent")
3 public class HelloController {
4     // Appelé pour GET /parent/hello
5     @GetMapping("/hello")
6     public String hello() {
7         return "Hello Getter";
8     }
9
10    // Appelé pour POST /parent/hello
11    @PostMapping("/hello")
12    public String helloPost() {
13        return "Hello Poster";
14    }
15 }
```

11.2

LES ANNOTATIONS DE PARAMÈTRES

`@RequestMapping` et `#XXXMapping` permettent d'exécuter une méthode en fonction de la route.

Pour récupérer des informations de la requête, il faut utiliser les annotations suivantes :

- `@PathVariable` pour récupérer une partie de la route
- `@RequestParam` pour récupérer un paramètre de la requête
- `@RequestBody` pour récupérer le corps de la requête

@PATHVARIABLE

`@PathVariable` permet de récupérer une partie de la route.

Dans l'exemple suivant, la route est `/hello/{name}`.

```
1 @RestController
2 public class HelloController {
3     /*
4      * Ici quand le client fait une requête GET sur /hello/John, le paramètre
5      * name sera égal à "John".
6      */
7     @GetMapping("/hello/{name}")
8     public String hello(@PathVariable String name) {
9         return "Hello " + name;
10    }
```

@PATHVARIABLE

`@PathVariable` permet de récupérer une partie de la route.

Dans l'exemple suivant, la route est `/hello/{name}`.

```
1 @RestController
2 public class HelloController {
3     /*
4      * Ici quand le client fait une requête GET sur /hello/John, le paramètre
5      * name sera égal à "John".
6      */
7     @GetMapping("/hello/{name}")
8     public String hello(@PathVariable String name) {
9         return "Hello " + name;
10    }
```

13.1

@REQUESTPARAM

`@RequestParam` permet de récupérer un paramètre de la requête.

Dans l'exemple suivant, la route est `/hello` et le paramètre `name` est récupéré.

```
1 @RestController
2 public class HelloController {
3     /*
4      * Ici quand le client fait une requête GET sur /hello?name=John, le pa
5      */
6     @GetMapping("/hello") // /hello?name=John
7     public String hello(@RequestParam String name) {
8         return "Hello " + name;
9     }
10 }
```

@REQUESTPARAM

`@RequestParam` permet de récupérer un paramètre de la requête.

Dans l'exemple suivant, la route est `/hello` et le paramètre `name` est récupéré.

```
1 @RestController
2 public class HelloController {
3     /*
4      * Ici quand le client fait une requête GET sur /hello?name=John, le pa.
5      */
6     @GetMapping("/hello") // /hello?name=John
7     public String hello(@RequestParam String name) {
8         return "Hello " + name;
9     }
10 }
```

14.1

@REQUESTPARAM AVEC VALEUR PAR DÉFAUT

`@RequestParam` demande que le paramètre soit présent dans la requête.

Pour définir une valeur par défaut, il faut utiliser l'attribut `defaultValue`.

```
1 @RestController
2 public class HelloController {
3     /*
4      * Ici quand le client fait une requête GET sur /hello, le paramètre name
5      */
6     @GetMapping("/hello")
7     public String hello(@RequestParam(defaultValue = "World") String name) {
8         return "Hello " + name; // = Hello World
9     }
10 }
```

@REQUESTPARAM AVEC VALEUR PAR DÉFAUT

`@RequestParam` demande que le paramètre soit présent dans la requête.

Pour définir une valeur par défaut, il faut utiliser l'attribut `defaultValue`.

```
1 @RestController
2 public class HelloController {
3     /*
4      * Ici quand le client fait une requête GET sur /hello, le paramètre name
5      */
6     @GetMapping("/hello")
7     public String hello(@RequestParam(defaultValue = "World") String name) {
8         return "Hello " + name; // = Hello World
9     }
10 }
```

15.1

@REQUESTPARAM AVEC VALEUR PAR DÉFAUT

`@RequestParam` demande que le paramètre soit présent dans la requête.

Pour définir une valeur par défaut, il faut utiliser l'attribut `defaultValue`.

```
1 @RestController
2 public class HelloController {
3     /*
4      * Ici quand le client fait une requête GET sur /hello, le paramètre name
5      */
6     @GetMapping("/hello")
7     public String hello(@RequestParam(defaultValue = "World") String name) {
8         return "Hello " + name; // = Hello World
9     }
10 }
```

15.2

@REQUESTPARAM OPTIONNEL

`@RequestParam` peut être utilisé sans valeur.

Dans ce cas, le paramètre est optionnel.

```
1 @RestController
2 public class HelloController {
3     /*
4      * Ici quand le client fait une requête GET sur /hello, le paramètre na
5      */
6     @GetMapping("/hello")
7     public String hello(@RequestParam(required = false) String name) {
8         return "Hello " + name; // = Hello null
9     }
10 }
```

@REQUESTPARAM OPTIONNEL

`@RequestParam` peut être utilisé sans valeur.

Dans ce cas, le paramètre est optionnel.

```
1 @RestController
2 public class HelloController {
3     /*
4      * Ici quand le client fait une requête GET sur /hello, le paramètre na
5      */
6     @GetMapping("/hello")
7     public String hello(@RequestParam(required = false) String name) {
8         return "Hello " + name; // = Hello null
9     }
10 }
```

16.1

@REQUESTPARAM OPTIONNEL

`@RequestParam` peut être utilisé sans valeur.

Dans ce cas, le paramètre est optionnel.

```
1 @RestController
2 public class HelloController {
3     /*
4      * Ici quand le client fait une requête GET sur /hello, le paramètre na
5      */
6     @GetMapping("/hello")
7     public String hello(@RequestParam(required = false) String name) {
8         return "Hello " + name; // = Hello null
9     }
10 }
```

16.2

@REQUESTBODY

`@RequestBody` permet de récupérer le corps de la requête.

Dans l'exemple suivant, la route est `/hello` et le corps de la requête est récupéré.

```
1 @RestController
2 public class HelloController {
3     /*
4      * Ici quand le client fait une requête POST sur /hello avec le corps :
5      */
6     @PostMapping("/hello")
7     public String hello(@RequestBody String name) {
8         return "Hello " + name;
9     }
10 }
```

@REQUESTBODY

`@RequestBody` permet de récupérer le corps de la requête.

Dans l'exemple suivant, la route est `/hello` et le corps de la requête est récupéré.

```
1 @RestController
2 public class HelloController {
3     /*
4      * Ici quand le client fait une requête POST sur /hello avec le corps ".
5      */
6     @PostMapping("/hello")
7     public String hello(@RequestBody String name) {
8         return "Hello " + name;
9     }
10 }
```

17.1

SÉRIALISATION ET DESERIALISATION



SÉRIALISATION

La **Sérialisation** est le processus de conversion d'un objet en une séquence de bits.

Les objets java sont représenté en mémoire de manière complexe.
Pour les envoyer au client, il faut les convertir en une séquence de bits.

EXEMPLE

```
class Personne {  
    private String name;  
    private int age;  
  
    public Personne(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    // Getters et Setters  
}
```

Une instance de cette classe est représentée en mémoire avec des références vers des objets.

EXEMPLE

```
Personne p = new Personne("John", 42);
```

peut être représenté en Json de la manière suivante :

```
{  
    "name": "John",  
    "age": 42  
}
```

Et le format **JSON** est une séquence de bits qui peut facilement être envoyé au client.

C'est la **Sérialisation** de l'instance de la classe **Personne** en **JSON**.

DESERIALISATION

La **Deserialisation** est le processus inverse de la **Sérialisation**.



DESERIALISATION

Il est nécessaire de déserialiser les objets pour les récupérer depuis le client.

Le plus souvent, on utilise le format **JSON** pour la deserialization.

EXEMPLE

```
{  
    "name": "John",  
    "age": 42  
}
```

peut être déserialisé en une instance de la classe **Personne**.

```
Personne personne = Deserialiseur.deserialise(json, Personne.class);  
personne.getName(); // = John  
personne.getAge(); // = 42
```

SÉRIALISATION ET DESERIALIZATION AVEC SPRING

Spring boot utilise Jackson pour la sérialisation et la deserialisation au format JSON.

Pour le bon fonctionnement de la sérialisation et de la deserialisation, il faut que:

- les classes soient `public`
- les getters et setters soient `public`
- Qu'il y ai un constructeur sans paramètre public
- Qu'il n'y ait pas de boucle dans les objets (Personne -> Personne -> Personne -> ...)

SÉRIALISATION JACKSON

Jackson ne sérialise que les méthodes **getter**.

```
class Personne {  
    private String name;  
    private int age;  
  
    public Personne(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    public String getName() {  
        return name;  
    }  
  
    public void setAge(int age) {  
        return age;  
    }  
}
```

sera sérialisé en **JSON** de la manière suivante :

```
{  
    "name": "John"  
}
```

DESERIALISATION JACKSON

Jackson ne désérialise que les méthodes **setter**.

```
{  
    "name": "John",  
    "age": 42  
}
```

Seul le champ **age** sera affecté lors de la désérialisation s'il n'y a pas de setter sur **name**.

SÉRIALISATION AVEC @REQUESTBODY

L'annotation `@RequestBody` permet de déserialiser le corps de la requête.

```
@RestController
public class HelloController {
    @PostMapping("/persons")
    public String hello(@RequestBody Person person) {
        return "Hello " + person.getName();
    }
}
```

DÉSÉRIALISATION DE LA RÉPONSE

Lors du retour d'une réponse, **Spring boot** sérialise automatiquement l'objet en **JSON**.

```
@RestController
public class HelloController {
    @GetMapping("/persons")
    public Person hello() {
        return new Person("John", 42);
    }
}
```

GESTION DES STATUS

Spring boot web permet de gérer les status de la réponse.

L'ANNOTATION @RESPONSESTATUS

L'annotation `@ResponseStatus` permet de définir le status de la réponse.

```
@RestController
public class HelloController {
    @GetMapping("/hello")
    @ResponseStatus(HttpStatus.OK)
    public String hello() {
        return "Hello World";
    }
}
```

L'ANNOTATION @RESPONSESTATUS

L'annotation `@ResponseStatus` prend en paramètre un objet `HttpStatus` qui permet de définir le status de la réponse.

L'ANNOTATION @RESPONSESTATUS SUR LES EXCEPTIONS

L'annotation `@ResponseStatus` peut être utilisée sur les exceptions.

Dans ce cas, si l'annotation est retournée, spring boot web retourne le status défini.

```
@RestController
public class HelloController {
    @GetMapping("/hello")
    public String hello() {
        throw new HelloException();
    }
}

@ResponseStatus(HttpStatus.NOT_FOUND)
class HelloException extends RuntimeException {
}
```

LES EXCEPTIONS RESPONSESTATUSEXCEPTION

Spring boot web fournit des exceptions `ResponseStatusException`.

Ces exceptions permettent de retourner un status et un message.

```
@RestController
public class HelloController {
    @GetMapping("/hello")
    public String hello() {
        throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Hello not found")
    }
}
```

EXERCICE 1

LES SERVICES

Pour créer un service, il faut utiliser l'annotation `@Service`.

`@Service` est équivalent à `@Component`.

```
@Service  
public class HelloService {  
    public String hello() {  
        return "Hello World";  
    }  
}
```

INJECTION DE DÉPENDANCES

Pour injecter un service dans un controller, utilisez le constructeur.

```
@RestController
public class HelloController {
    private final HelloService helloService;

    public HelloController(HelloService helloService) {
        this.helloService = helloService;
    }

    @GetMapping("/hello")
    public String hello() {
        return helloService.hello();
    }
}
```

LES REPOSITORIES

Pour créer un repository, il faut utiliser l'annotation `@Repository`.

`@Repository` est équivalent à `@Component`.

```
@Repository
public class TrucRepository {

    private final List<Truc> Trucs = new ArrayList<>();

    public List<Truc> findAll() {
        return Trucs;
    }

    public void save(Truc Truc) {
        trucs.add(Truc);
    }

    public void deleteAll() {
        trucs.clear();
    }
}
```

INJECTION DE DÉPENDANCES

Pour injecter un repository dans un service, utilisez le constructeur.

```
@Service
public class TrucService {
    private final TrucRepository trucRepository;

    public TrucService(TrucRepository TrucRepository) {
        this.trucRepository = TrucRepository;
    }

    public List<Truc> findAll() {
        return trucRepository.findAll();
    }

    public void save(Truc Truc) {
        trucRepository.save(Truc);
    }
}
```

EXERCICE 2

CONNEXION À UNE BASE DE DONNÉES