



SPRING

A decorative wavy line in light blue and white, flowing vertically along the left side of the slide.

SPRING MVC

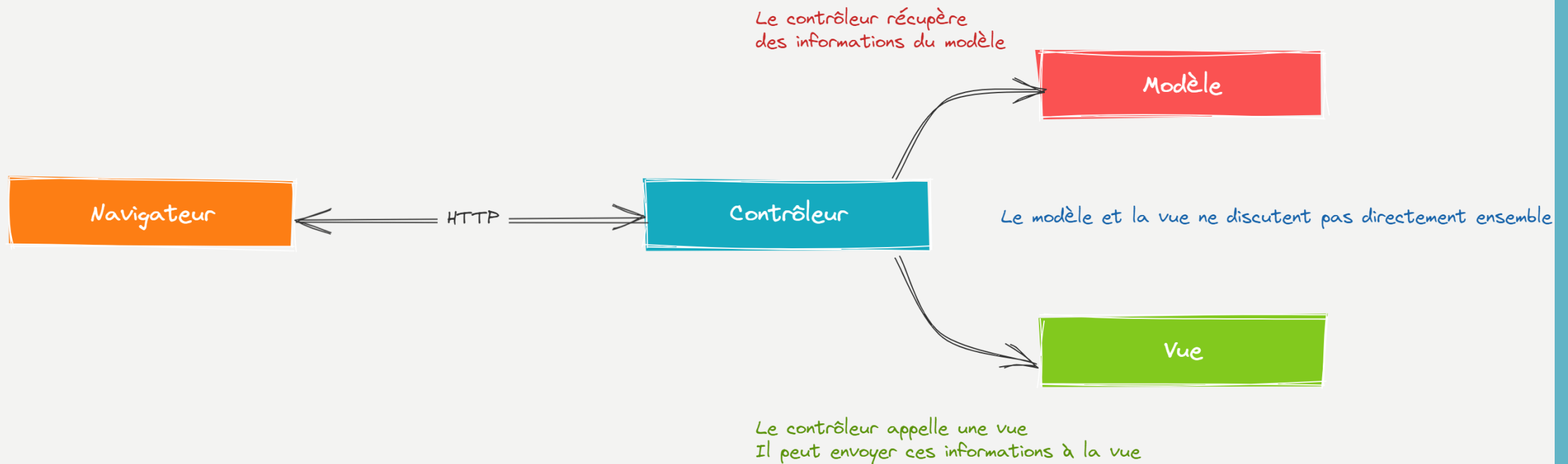
INTRODUCTION À
SPRING WEB MVC

PRÉSENTATION DE SPRING WEBMVC

- Une **Servlet** principale : `DispatcherServlet`
 - Délègue les requêtes à des contrôleurs (classes annotées `@Controller`)
 - Selon le point d'accès (la ressource URL)
- Un **controller**
 - Fabrique un modèle sous la forme d'une **Map** qui contient les éléments de la réponse
 - Clé / Valeur
 - Utilise une `View` pour afficher la vue (la page **HTML**)

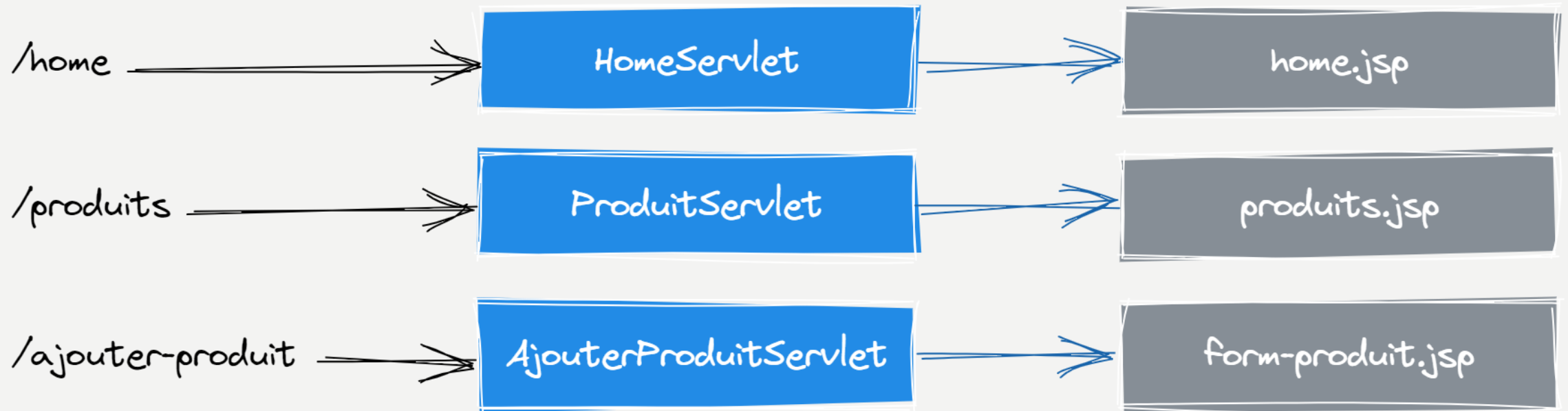
PRÉSENTATION DE SPRING WEBMVC

Application MVC - Modèle - Vue - Contrôleur



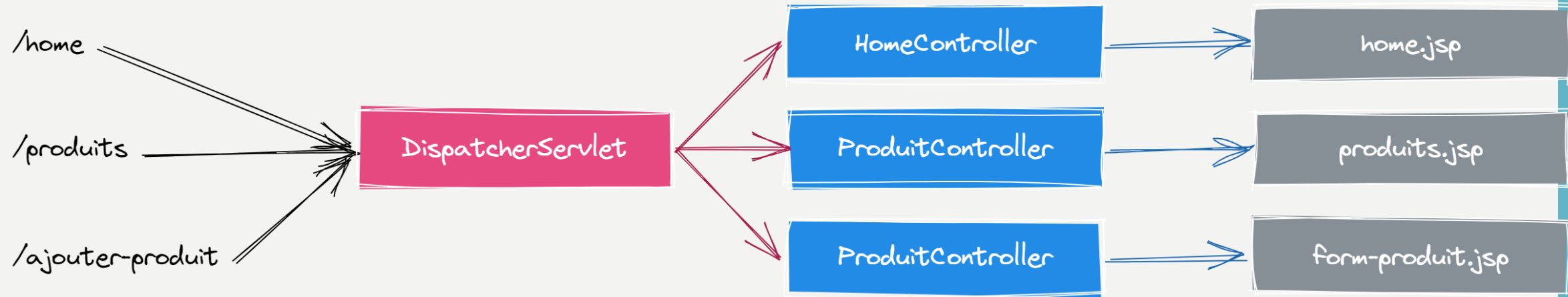
PRÉSENTATION DE SPRING WEBMVC

JEE Servlets



PRÉSENTATION DE SPRING WEBMVC

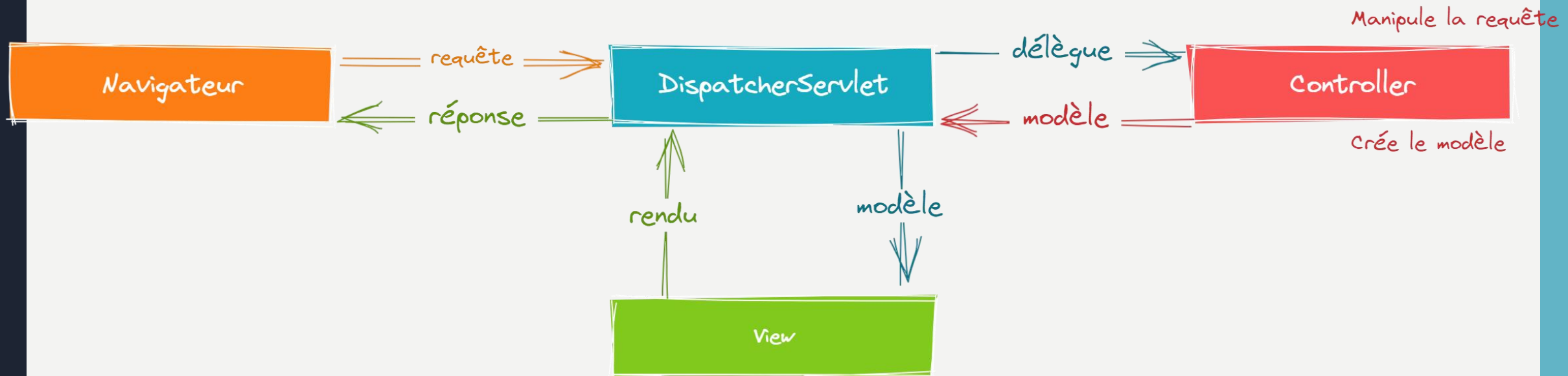
Spring WEB MVC



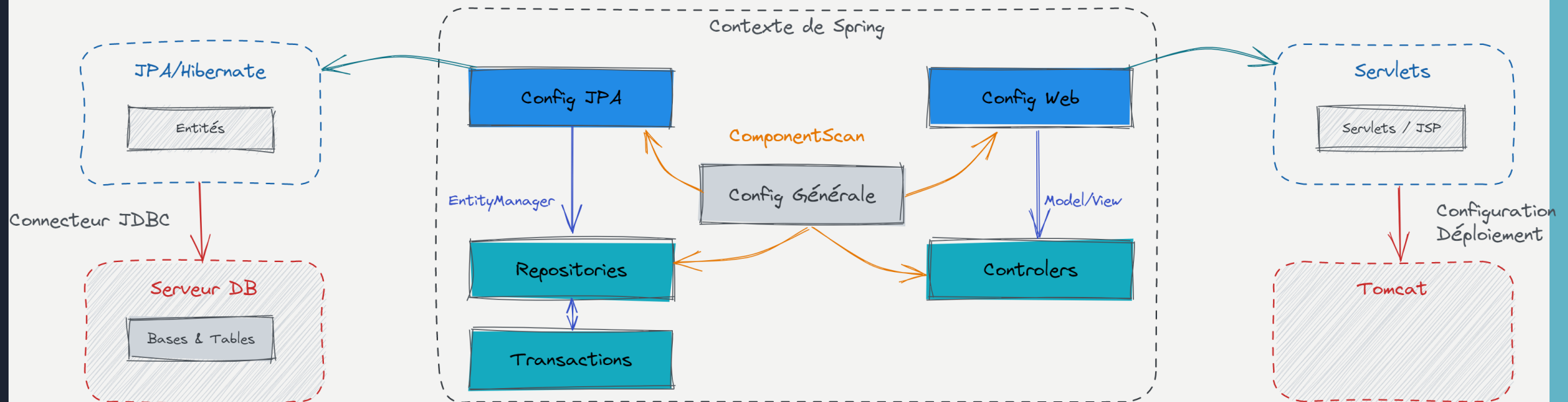
PRÉSENTATION DE SPRING WEBMVC

- La **Servlet DispatcherServlet** est mappée sur un pattern d'URL
 - Par exemple « / » (toutes les URL)
- Il n'y a plus nécessité des **Servlets JEE**
 - Mais si elles existent, leur mapping prend le pas sur le mapping des `@Controller` !
 - Tout est géré dans le contexte de **Spring**

PRÉSENTATION DE SPRING WEBMVC



PRÉSENTATION DE SPRING WEBMVC



A decorative wavy line in light blue and white, running vertically along the left side of the slide.

CONFIGURATION

MISE EN PLACE

CONFIGURATION

- Pour une utilisation dans un cadre Application Web MVC
 - Dépendance **spring-webmvc**
 - Déclaration de la **Servlet** `DispatcherServlet` dans *web.xml*

CONFIGURATION

```
<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

  <init-param>
    <param-name>contextClass</param-name>
    <param-value>org.springframework.web.context.support.AnnotationConfigWebApplicationContext</param-value>
  </init-param>

  <init-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>fr.formation.config.AppConfig</param-value>
  </init-param>

  <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>
```

CONFIGURATION

- Si on veut configurer le profile, on ajoute un paramètre dans le DispatcherServlet

```
<init-param>  
  <param-name>spring.profiles.active</param-name>  
  <param-value>dev</param-value>  
</init-param>
```

CONFIGURATION

- Dans le fichier de configuration de `DispatcherServlet`
 - Utiliser l'annotation `@EnableWebMvc` pour déléguer les requêtes aux contrôleurs
 - Cette annotation crée deux **beans**
 - `DefaultAnnotationHandlerMapping`
 - `AnnotationMethodHandlerAdapter`

```
@Configuration
@EnableWebMvc
public class WebConfig {

}
```

EXERCICE

- Créer un nouveau projet « eshop-web » (Maven)
- Implémenter et configurer Spring MVC
 - Dépendance **spring-webmvc**
- Déployer le projet sur le serveur
 - Il ne doit pas y avoir d'erreur au démarrage du serveur !

CONFIGURATION

- DispatcherServlet est mappée sur toutes les ressources ... (/)
 - Comment accéder aux ressources **CSS, JS, Images**, ... ?
 - Comment distribuer une ressource statique ?
- La classe de configuration doit implémenter WebMvcConfigurer
 - Vous devez surcharger la méthode addResourceHandlers()

```
@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/assets/**").addResourceLocations("/WEB-INF/assets/");
    }
}
```




CONTROLLER

LE CONTRÔLEUR SPRING WEBMVC

LE CONTROLLER

- Le contrôleur est utilisé pour traiter une requête
 - Classe **POJO** annotée de `@Controller`
- Le mécanisme est le suivant :
 - Mapping d'une URL sur une méthode d'un contrôleur
 - Injection possible
 - Des paramètres de la requête aux arguments de la méthode
 - Des objets du contexte de la requête aux arguments de la méthode
 - De tout **bean** géré par **Spring**
 - La méthode retourne une chaîne de caractères
 - Il s'agit du nom logique de la vue que **Spring** doit générer
 - **Spring** va utiliser le **bean** `ViewResolver` déclaré dans le fichier de configuration

LE CONTROLLER

Déclaration du contrôleur

Mapping de l'URL

```
@Controller
public class HomeController {
    @RequestMapping("/home")
    public String home() {
        return "/WEB-INF/views/home.jsp";
    }
}
```

Nom logique de la vue

Note : on peut rediriger en utilisant "redirect:url" (exemple : "redirect:/produits")

LE CONTROLLER

- La méthode mappée retourne une chaîne de caractères
 - Utilisée pour retourner le nom de la vue
 - Peut être utilisée pour rediriger l'utilisateur vers une autre adresse URL
 - Avec "redirect:/url"

```
@Controller
public class ProduitController {
    @RequestMapping("/ajouterProduit")
    public String addProduit() {
        return "redirect:/produits";
    }
}
```


LE CONTROLLER

- Il est possible de spécifier le type de commande **HTTP** (GET, POST, PUT, DELETE,...)
 - Par défaut, toutes les commandes **HTTP** sont mappées sur l'URL avec `@RequestMapping`

```
@RequestMapping(value = "/home", method = RequestMethod.GET)
public String home() {
    return "home";
}
```

Note : il existe aussi

`@GetMapping`
`@PostMapping`
`@PutMapping`
`@DeleteMapping`
...



Spécifie la commande HTTP

EXERCICE

- Créer un contrôleur HomeController
 - Méthode mappée sur « /home »
 - Affiche une **JSP** *home.jsp*
 - « Allô le monde ?! »



VIEWRESOLVER

RÉSOLUTION NOM DE VUE LOGIQUE

VIEWRESOLVER

- On peut paramétrer les vues dans la configuration avec un `ViewResolver`
 - Nos vues **JSP** sont dans le répertoire `"/WEB-INF/views/"`
 - Le nom des fichiers se terminent par `".jsp"`
 - Permettra de retourner `"home"` au lieu de `"/WEB-INF/views/home.jsp"` dans le contrôleur

```
@Bean
public ViewResolver viewResolver() {
    UrlBasedViewResolver viewResolver = new UrlBasedViewResolver();

    viewResolver.setViewClass(JstlView.class);
    viewResolver.setPrefix("/WEB-INF/views/");
    viewResolver.setSuffix(".jsp");

    return viewResolver;
}
```


EXERCICE

- Mettre en place la configuration du `ViewResolver`
 - Modifier le contrôleur en conséquence



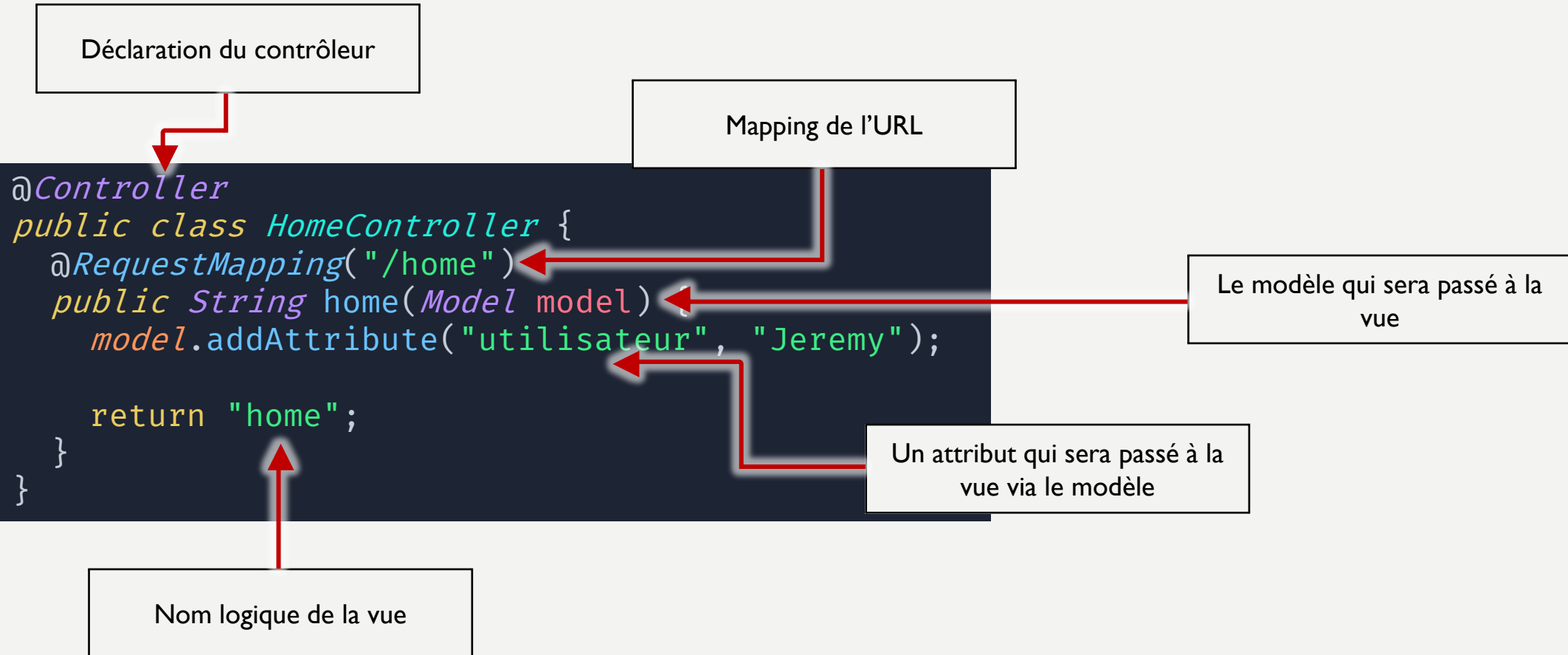
INJECTION

INJECTION SPRING

INJECTION

- On peut demander à **Spring**, pour la requête, de nous injecter des dépendances
 - Le modèle **Spring** utilisé du Contrôleur vers la Vue
 - Session de l'utilisateur
 - Paramètres de requête (simples ou complexes)
 - Variables de chemin
 - Toute autre information liée à la requête **HTTP**

INJECTION



Note : on peut rediriger en utilisant "redirect:url" (exemple : "redirect:/produits")

INJECTION

- Demander l'objet `HttpSession` (qui nous permettra de manipuler la session)
 - Principe d'injection de dépendance basé sur le type

```
@GetMapping("/home")  
public String home(HttpSession session, Model model) {  
    return "home";  
}
```



Demande l'objet **HttpSession**

EXERCICE

- Créer un contrôleur HomeController
 - Méthode mappée sur `/home`
 - N'accepte que du GET
 - Affiche une **JSP** `home.jsp`
 - « Allô le monde ?! »

A decorative wavy line in light blue and white, flowing from the top left towards the bottom left of the slide.

PARAMÈTRES

PARAMÈTRES DE REQUÊTE

PARAMÈTRES


- Avec les **Servlets**
 - Utilisation de l'objet `HttpServletRequest` pour récupérer un paramètre (`getParameter`)
- Avec **Spring**
 - Possible d'utiliser `HttpServletRequest`...
 - Mais préférable d'utiliser l'injection, avec l'annotation `@RequestParam` !
 - Convertit automatiquement la valeur selon le type de l'argument

PARAMÈTRES

`http://localhost:8080/mon-projet/home?username=Albert`

Injection du paramètre
"username"

```
@Controller
public class HomeController {
    @GetMapping("/home")
    public String home(@RequestParam String username, Model model) {
        model.addAttribute("utilisateur", username);
        return "home";
    }
}
```



NOTE : Si le nom du paramètre ("username") est identique au nom de l'argument, **Spring** fait le lien tout seul. Sinon :

```
@RequestParam(value = "username", required = false, defaultValue = "Jeremy") String name
```

PARAMÈTRES

- **Spring** est capable de convertir le paramètre selon le type de l'argument

```
@Controller
public class ProduitController {
    @GetMapping("/ajouterProduit")
    public String addProduit(@RequestParam String nom, @RequestParam float prix) {
        //...
    }
}
```

EXERCICE

- Modifier HomeController
 - La méthode attend un paramètre optionnel "username"
 - N'accepte que du GET
 - Affiche une **JSP** « home.jsp »
 - « Bonjour nomUtilisateur ! »
- Créer un contrôleur ProduitController
 - Méthode mappée sur « /produit » qui attend un paramètre obligatoire "id" (entier)
 - N'accepte que du GET
 - Rechercher le produit qui a l'ID
 - Affiche une **JSP** « produit.jsp »
 - « Voici le libellé du produit : libelle »

A decorative wavy line in light blue and white, flowing from the top left towards the bottom left of the slide.

PATHVARIABLE

VARIABLES DE CHEMIN

VARIABLES DE CHEMIN

- A l'instar des paramètres
 - Les variables de chemin sont des informations envoyées par le client
 - Sous une forme plus technique, directement incluse dans le chemin de l'URL
- Exemples avec paramètres
 - `http://localhost:8080/mon-projet/produit?id=42`
 - `http://localhost:8080/mon-projet/produit?id=42&nom=gopro-hero-6`
- Exemples avec variables de chemin
 - `http://localhost:8080/mon-projet/produit/42`
 - `http://localhost:8080/mon-projet/produit/42-gopro-hero-6`

VARIABLES DE CHEMIN

- Les variables de chemin est les paramètres peuvent s'utiliser en même temps
 - `http://localhost:8080/mon-projet/produit/42-gopro-hero-6?couleur=noire`

VARIABLES DE CHEMIN

- Comme pour les paramètres et leur annotation `@RequestParam`
 - Les variables de chemin ont leur annotation `@PathVariable` !

```
@GetMapping({ "/produit", "/produit/{id}" })  
public String home(@PathVariable int id, Model model) {  
    model.addAttribute("id", id);  
    return "produit";  
}
```

```
@PathVariable(value = "produitId", required = false)
```

EXERCICE

- Modifier `ProduitController`
 - Méthode mappée sur « `/produit` » qui attend une variable obligatoire "id" (entier)
 - N'accepte que du GET
 - Rechercher le produit qui a l'ID
 - Affiche une **JSP** `produit.jsp`
 - « Voici les informations du produit : id, libelle et prix »



DATA BINDING

DATA BINDING

DATA BINDING

- Des données issues d'un formulaire
 - Il faut mapper les propriétés de la classe une à une
 - `setNom(request.getParameter("nom"))`
 - `setPrix(request.getParameter("prix"))`
 - ... même chose en utilisant `@RequestParam`
- Le **Data Binding** va nous éviter tout ça !

DATA BINDING

- C'est **Spring** qui se chargera d'utiliser les setters de `Produit`, à notre place

```
@PostMapping("/produit/nouveau")
public String ajouterProduit(Produit produit, Model model) {
    System.out.println("Nom : " + produit.getNom());
    System.out.println("Prix : " + produit.getPrix());
    //...
    return "redirect:/produits";
}
```

Récupérer l'objet bindé via
l'attribut "user" de Model

`@ModelAttribute` permet de réinjecter dans le model la valeur de l'objet

DATA BINDING

```
<form method="POST">
  <div>
    <label for="nom">Nom</label>
    <input id="nom" name="nom" type="text" value="${ produit.nom }" />
  </div>

  <div>
    <label for="prix">Prix</label>
    <input id="prix" name="prix" type="number" value="${ produit.prix }" />
  </div>
</form>
```

Objet du modèle et la valeur
de sa propriété

Nom des attributs

Dans le cas où on utilise **Thymeleaf**, il faut que l'objet existe, ou ajouter « ? » après le nom de l'attribut, pour tester si la valeur existe

EXERCICE

- Modifier le CRUD « produit » en conséquence
 - Retirer les `@RequestParam`
 - Implémenter la modification d'un produit
 - Utiliser le même formulaire pour l'ajout et l'édition !

A decorative wavy line in light blue and white, running vertically along the left side of the slide.

PETITS PLUS

DES PLUS SUR LE CONTRÔLER

LE CONTROLLER

http://.../produit/liste	➔ produits.jsp
http://.../produit/gopro	➔ produit.jsp
http://.../produit/iphone	➔ produit.jsp

- Il est possible de mapper toutes les méthodes d'un contrôleur
 - En annotant le contrôleur de `@RequestMapping`

```
@Controller
@RequestMapping("/produit")
public class ProduitController {
    @GetMapping("/liste")
    public String getProduits() {
        return "produits";
    }

    @GetMapping("/{id}")
    public String getProduit(@PathVariable Integer id) {
        return "produit";
    }
}
```

PETITS PLUS

- Il est possible de modifier le status **HTTP** de réponse
 - En utilisant l'annotation `@ResponseStatus` sur la méthode

EXERCICE

- Commencer le CRUD de « produit » dans un même contrôleur `ProduitController`
 - Lister les produits
 - GET
 - Ajouter un produit
 - GET / POST
 - Supprimer un produit
 - GET

A decorative wavy line in light blue and white, running vertically along the left side of the slide.

EXCEPTION

GESTION DES EXCEPTIONS

EXCEPTIONS

- Un traitement peut lever une exception
 - On peut choisir de la traiter dans le code avec les blocs `try .. catch ...`
 - Ou bien d'afficher un message d'erreur **HTTP** à l'utilisateur (status erreur)
 - Créer une `RuntimeException` annotée de `@ResponseStatus`
 - Lever cette `RuntimeException` si nécessaire
 - Le status **HTTP** sera modifié et l'erreur envoyée à l'utilisateur

EXCEPTIONS

- Un traitement peut lever une exception
 - On peut choisir de la traiter dans le code avec les blocs *try .. catch ...*
 - Ou bien dédier un contrôleur spécifique à la gestion des exceptions
 - Annoté de `@ControllerAdvice`
 - Les méthodes d'interception seront annotées de `@ExceptionHandler` et éventuellement de `@ResponseStatus`

```
@ControllerAdvice
public class ExceptionHandlerController {
    @ExceptionHandler(Exception.class)
    public String handleException(Exception ex) {
        return "pageErreur";
    }
}
```

EXCEPTIONS

- On peut aussi gérer les ressources non trouvées (non mappées)
 - Ajouter dans web.xml le paramètre *throwExceptionIfNoHandlerFound* avec la valeur à *true*
 - Avoir une méthode qui intercepte l'exception *NoHandlerFoundException*

```
<init-param>
  <param-name>throwExceptionIfNoHandlerFound</param-name>
  <param-value>true</param-value>
</init-param>
```

```
@ControllerAdvice
public class ExceptionHandlerController {
    @ExceptionHandler(NoHandlerFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public String noHandlerFound() {
        return "pageErreur404";
    }
}
```

EXERCICE

- Mettre en place la gestion des exceptions
 - Page 404 personnalisée
 - Page d'erreur générale