

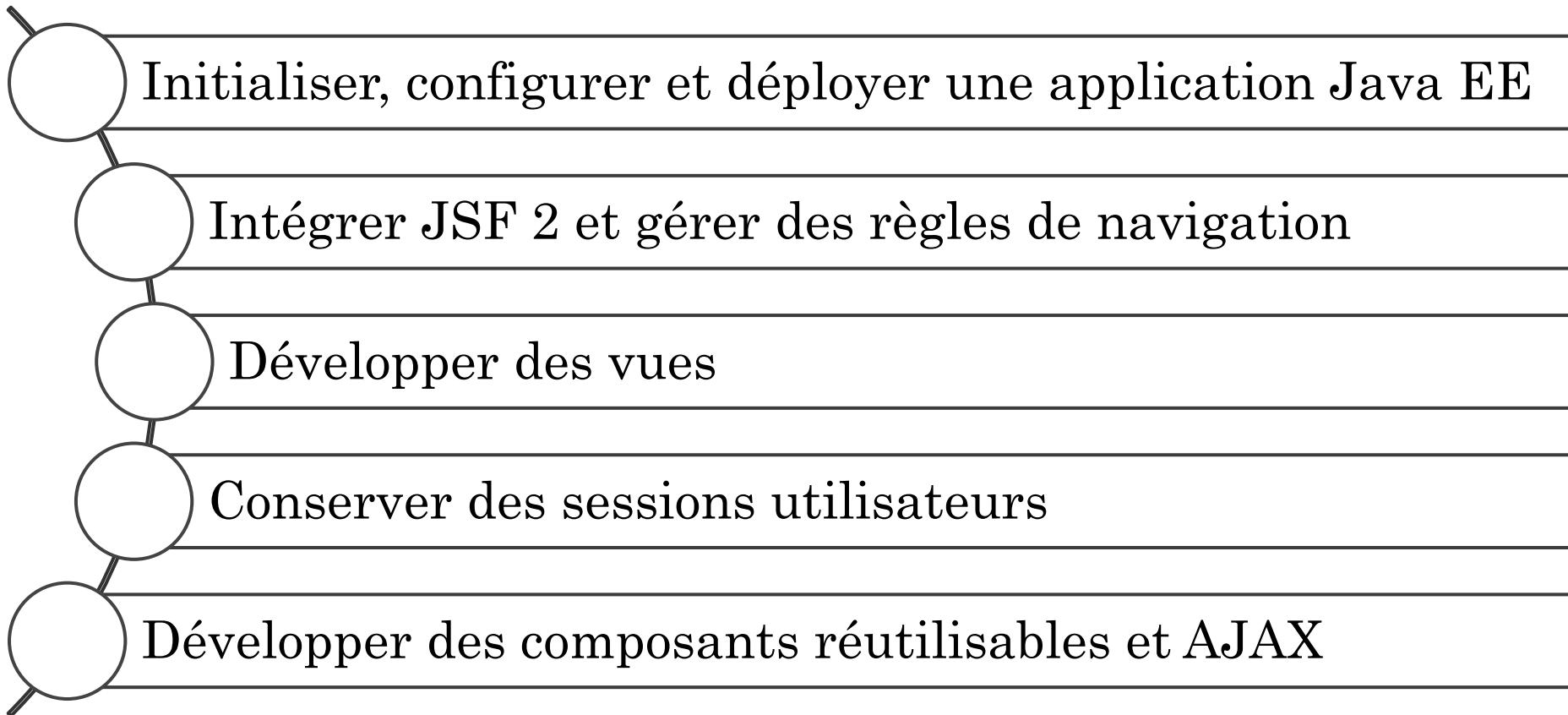
Java EE JSF et AJAX



**Omar Farouk
KOUGBADA**

Formateur et Ingénieur Informatique

Plan d'actions





Initialiser, configurer et déployer une application Java EE

Java & Java EE & JavaScript ?



JavaScript

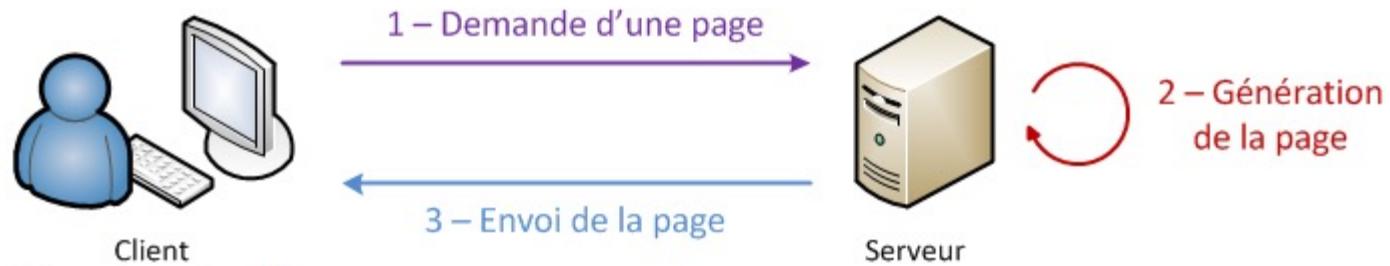


American Airlines

The American Airlines logo consists of the company name in a dark blue serif font, with a red, white, and blue stylized "A" icon to the right.

Comment ça marche

- **le client** : dans la plupart des cas, c'est le navigateur installé sur votre ordinateur.
- **le serveur** : c'est la machine sur laquelle le site est hébergé, où les fichiers sont stockés et les pages web générées.



1 - Le client saisit une URL



Client

2 – Le navigateur envoie une
requête HTTP au serveur



4 – Le serveur renvoie une
réponse HTTP au client

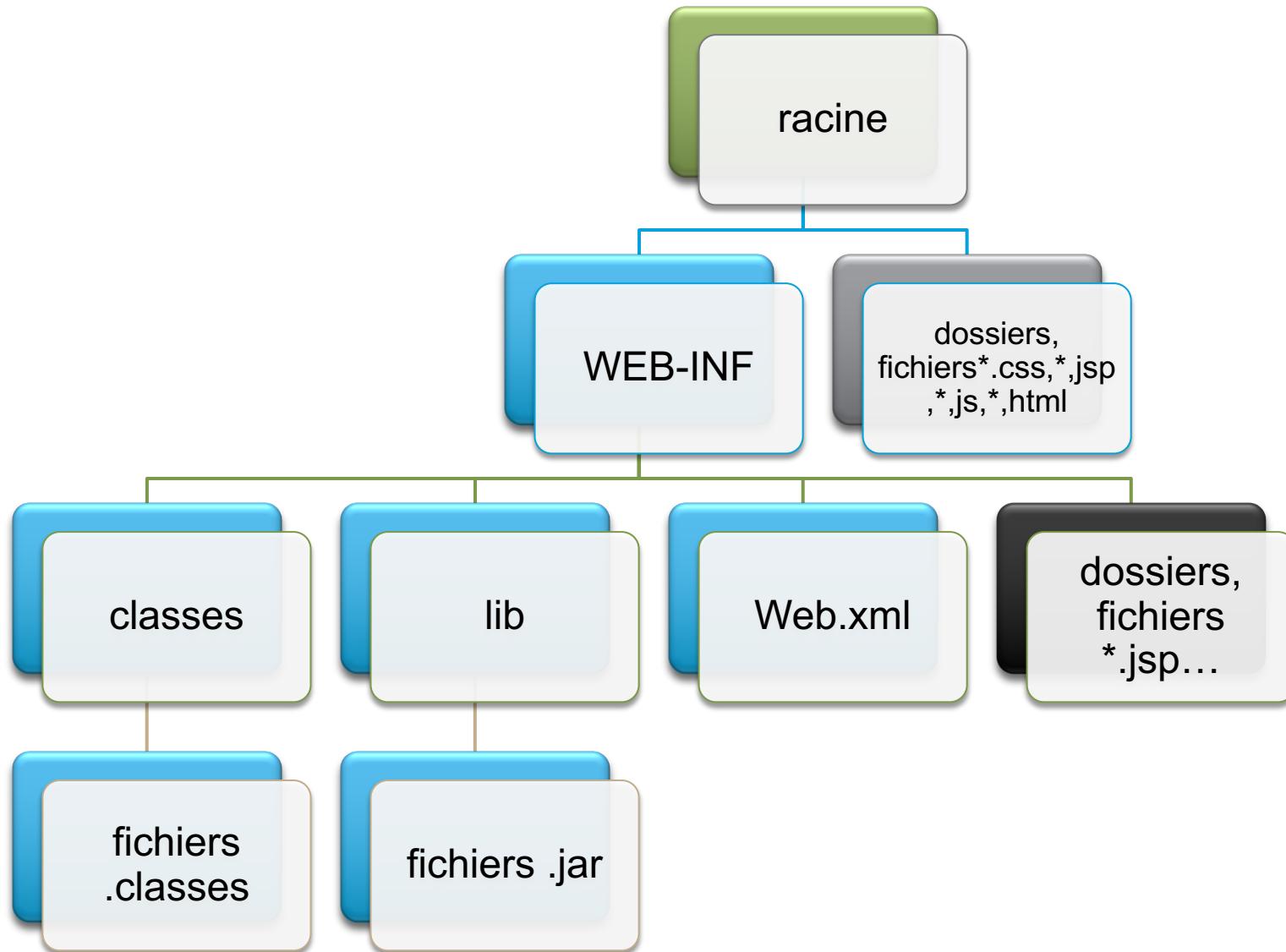


Serveur

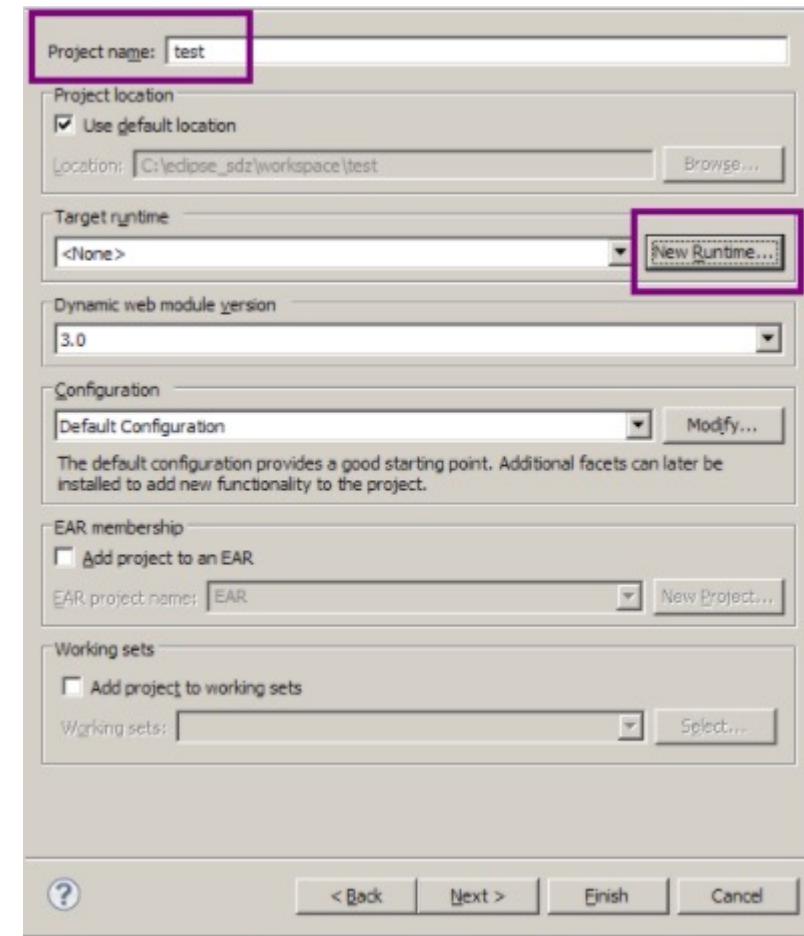
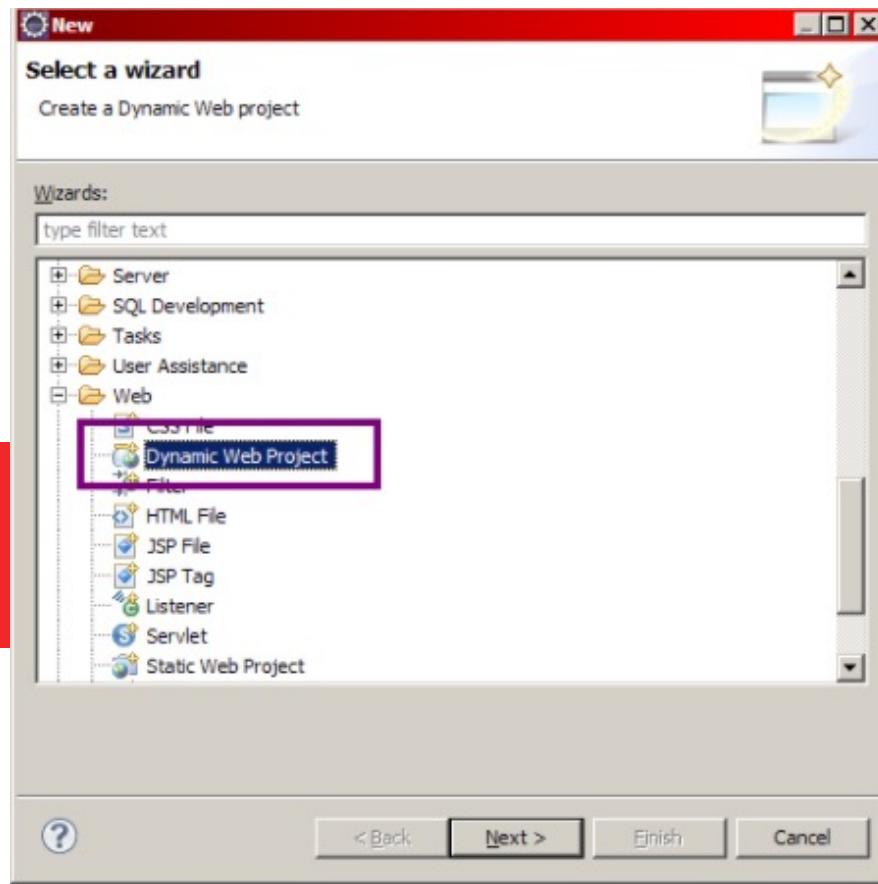
3 – Le serveur traite la requête et
génère la page web demandée



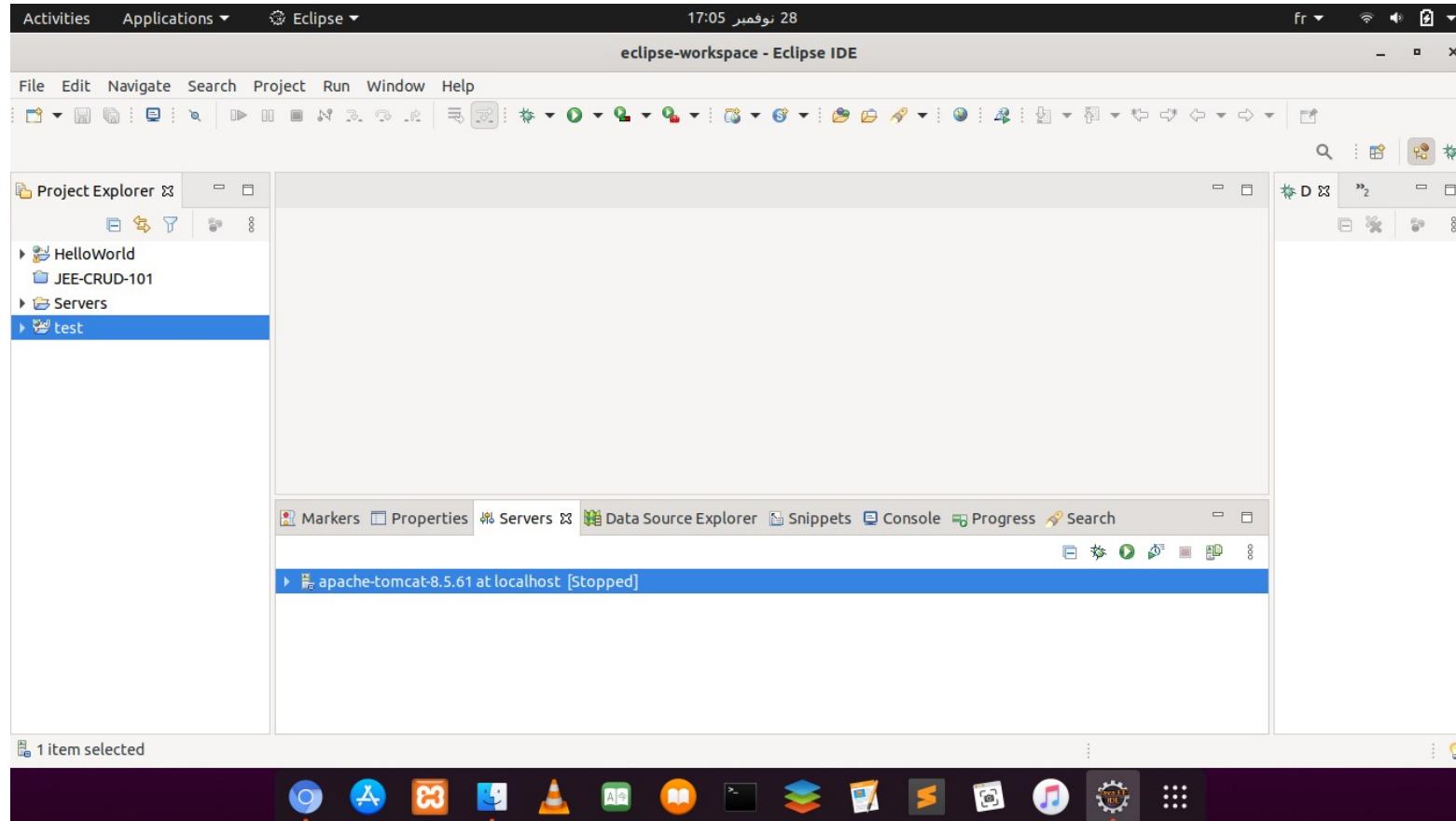
Structure d'une application Java EE



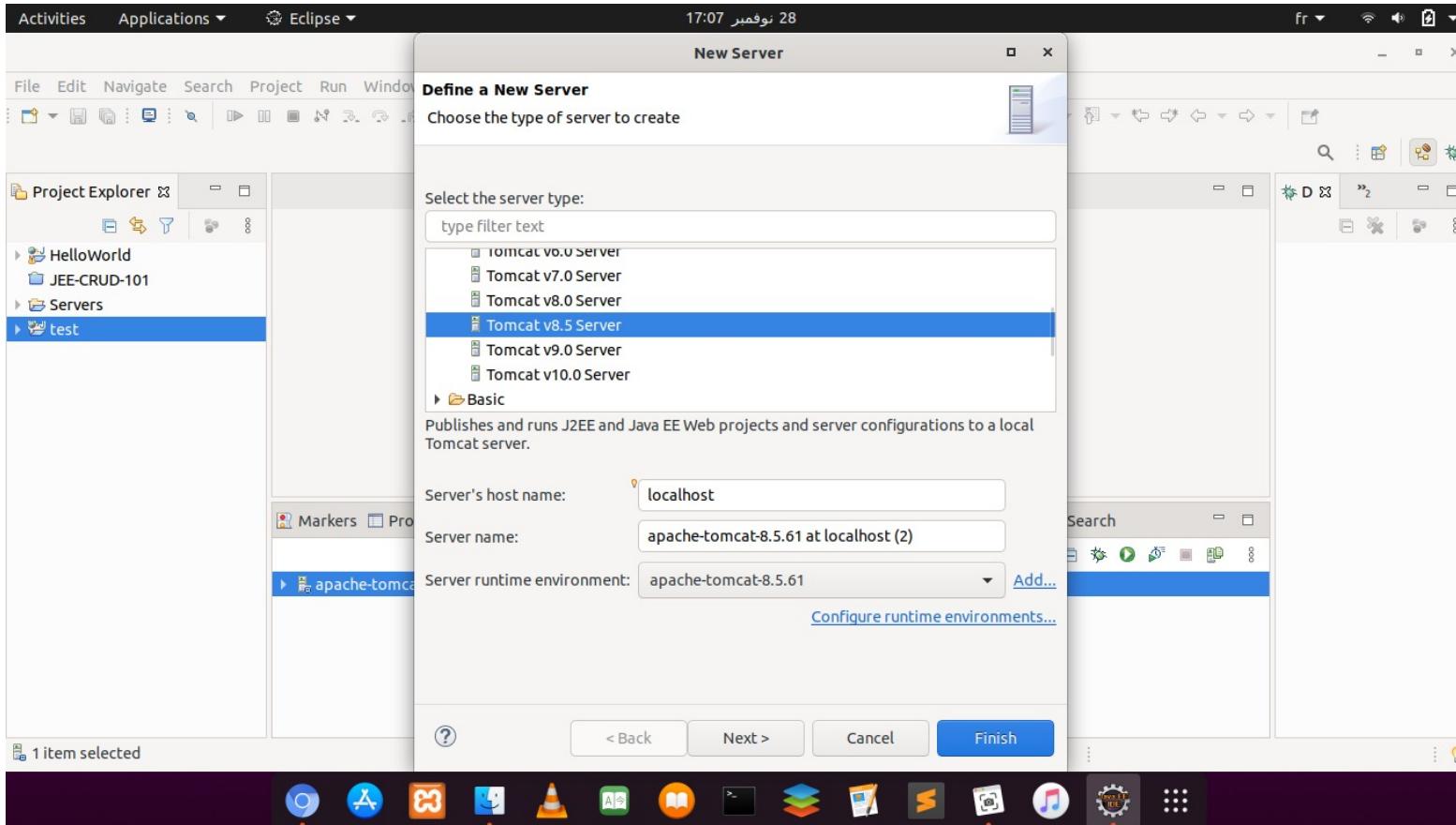
Création du projet web avec Eclipse



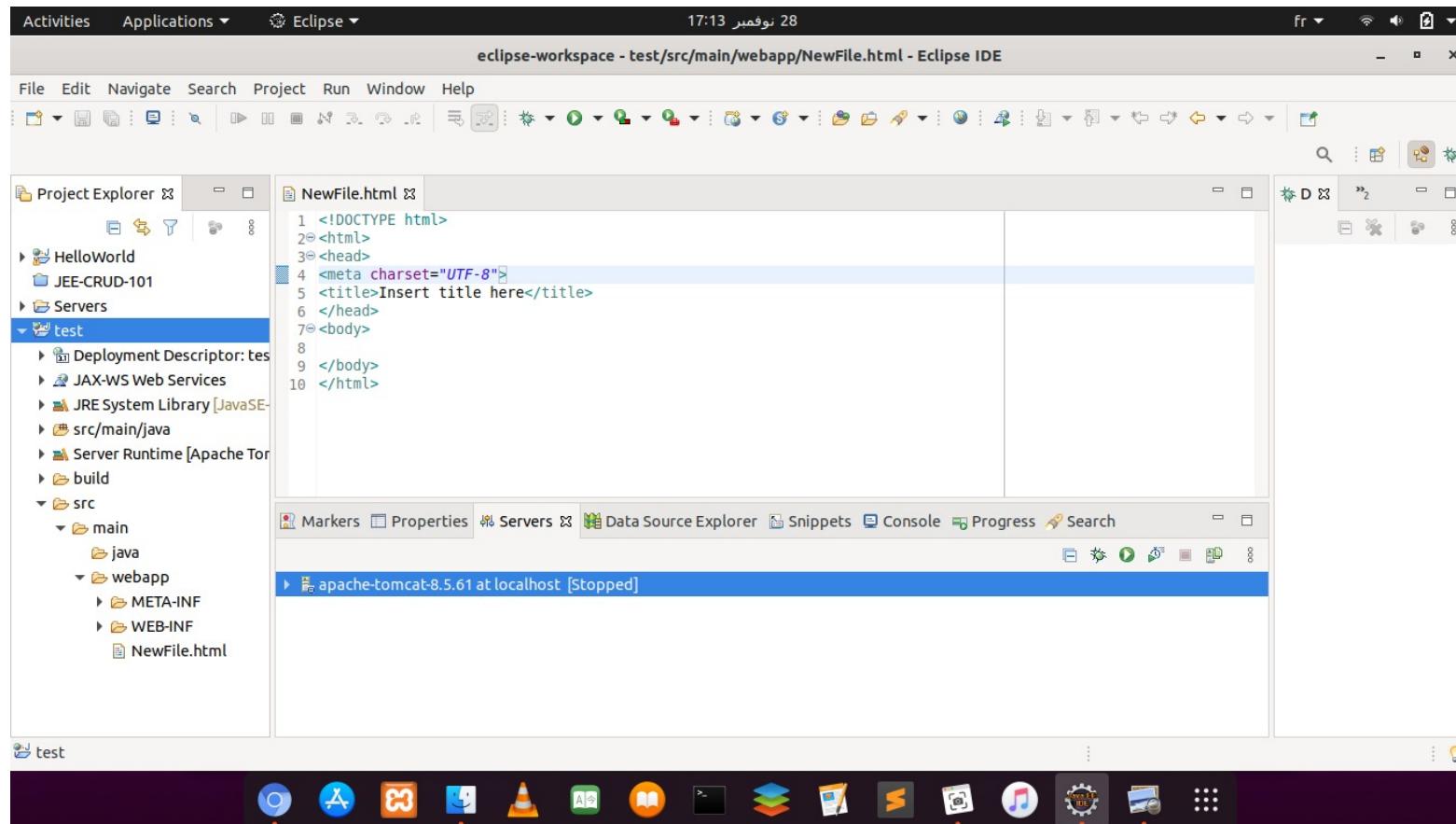
Création du projet web avec Eclipse



Création du projet web avec Eclipse



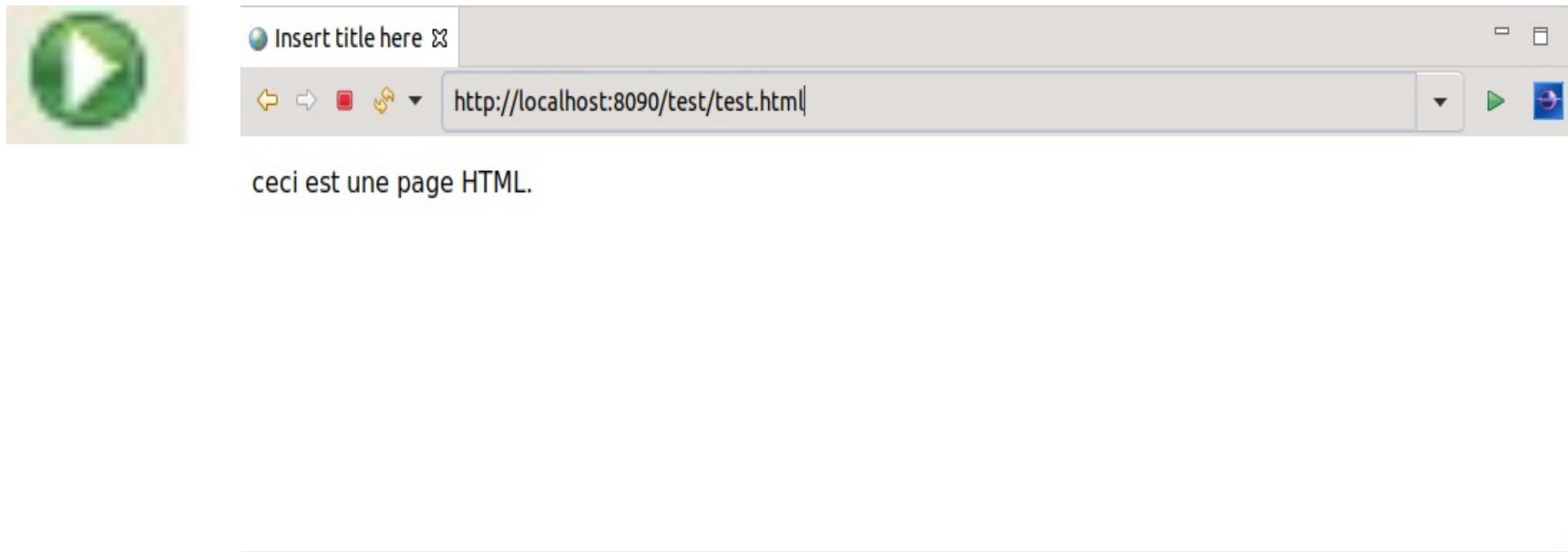
Création d'une page web



Création d'une page web

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Test</title>
    </head>
    <body>
        <p>Ceci est une page HTML.</p>
    </body>
</html>
```

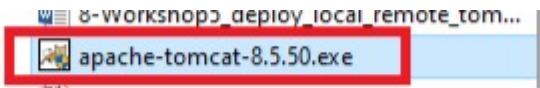
Création d'une page web



JAR ou WAR

- Un fichier JAR (Java archive) est un fichier ZIP utilisé pour distribuer un ensemble de classes Java.
- Un fichier WAR (pour Web application Archive) est un fichier JAR utilisé pour contenir un ensemble de JavaServer Pages, servlets, classes Java, fichiers XML, et des pages web statiques (HTML, JavaScript...),
 - le tout constituant une application web.

- Commençons par installer le serveur Tomcat local



- Vous pouvez éditer un nom d'utilisateur / mot de passe, un port pour votre serveur pendant l'étape d'installation.

Apache Tomcat/8.5.50



Recommended Reading:

- [Security Considerations How-To](#)
- [Manager Application How-To](#)
- [Clustering/Session Replication How-To](#)

Server Status

Manager App

Host Manager

Export



WAR Export

Export Web project to the local file system.

Web project: test

Destination: /home/seebro/test.war

Browse...

Target runtime

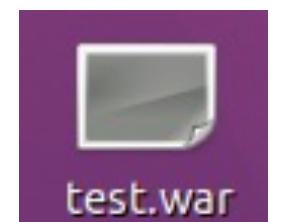
 Optimize for a specific server runtime

Apache Tomcat v8.5

 Export source files Overwrite existing file

Cancel

Finish



- Maintenant, téléchargeons le fichier war sur notre serveur Tomcat local

WAR file to deploy

Select WAR file to upload test.war

- Maintenant, notre projet fonctionne sous le port 8080

/test	None specified		true	0	<input type="button" value="Start"/> <input type="button" value="Stop"/> <input type="button" value="Reload"/> <input type="button" value="Undeploy"/>
					<input type="button" value="Expire sessions"/> with idle ≥ <input type="text" value="30"/> minutes

← → ⌂ ⓘ localhost:8080/test/test.html

ceci est une page HTML.





Intégrer JSF 2 et gérer des règles de navigation

JSF?

- Java Server Faces (JSF) est un framework basé sur Java destiné à simplifier l'intégration du développement des interfaces utilisateur Web



ebay



COSTCO
WHOLESALE

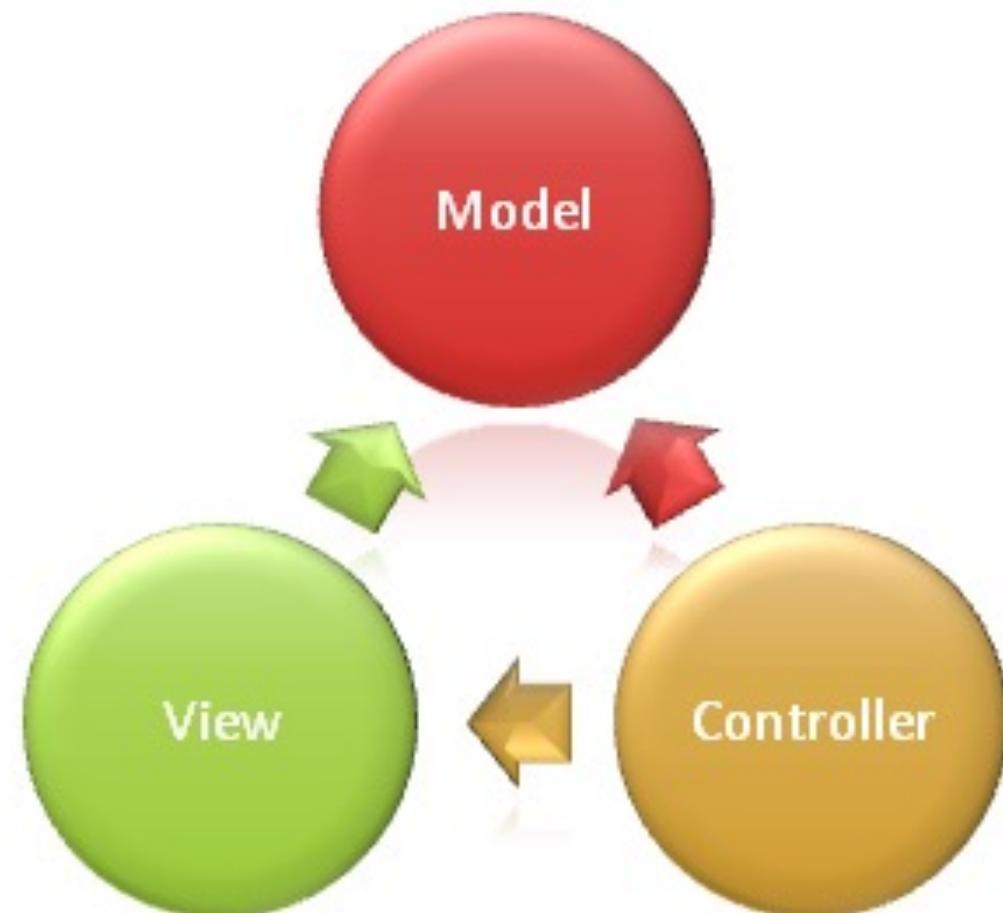


Lufthansa

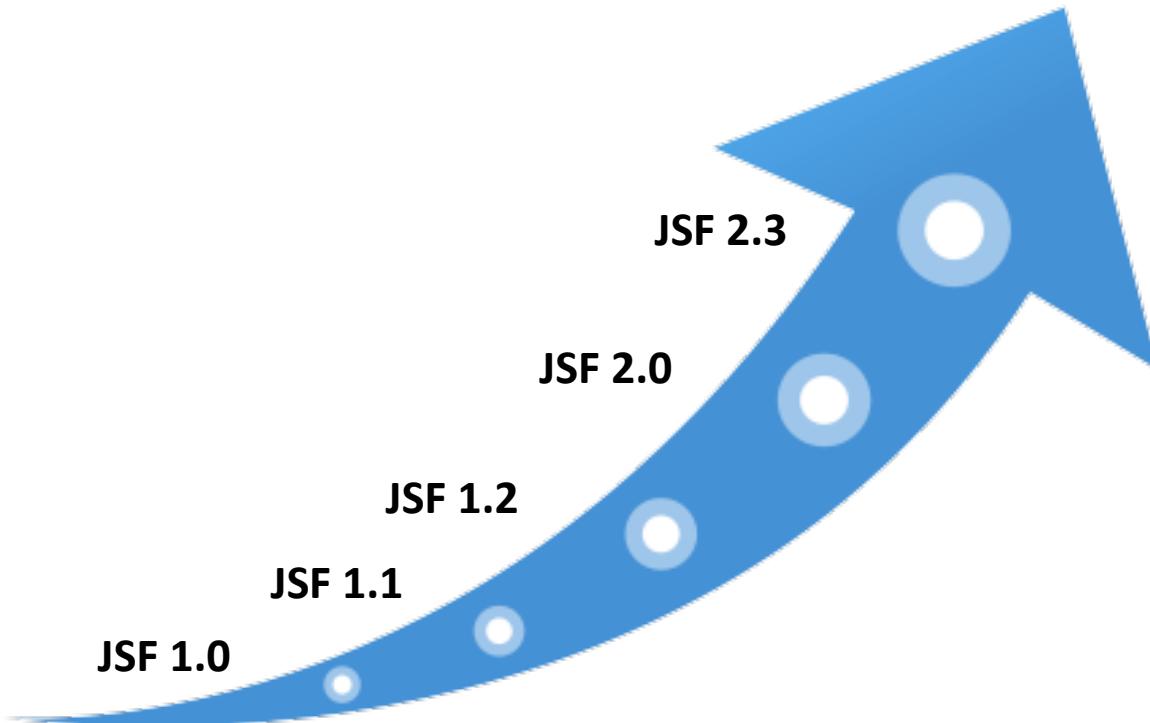
JSF: Principe

JavaServer Faces (JSF) est un framework web MVC qui simplifie la construction d'interfaces utilisateur (UI) pour les applications basées sur serveur à l'aide de composants d'interface utilisateur réutilisables dans une page.

Framework MVC ?



JSF : Historique



Structure d'une application JSF

- La vue est généralement assurée par des pages JSP ou par des pages XHTML
- Le modèle est assuré par des entités ou des JavaBeans
- le contrôleur, est décomposé en deux éléments :
 - 1) Une unique servlet mère servant de point d'entrée à toute requête, la FacesServlet ;
 - 2) Un JavaBean particulier, déclaré via une annotation et désigné par le terme managed-bean

Intégration de JSF avec Spring Boot



```
<dependency>
    <groupId>org.apache.myfaces.core</groupId>
    <artifactId>myfaces-impl</artifactId>
    <version>2.2.12</version>
</dependency>
<dependency>
    <groupId>org.apache.myfaces.core</groupId>
    <artifactId>myfaces-api</artifactId>
    <version>2.2.12</version>
</dependency>
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
</dependency>
```

```
<dependency>
    <groupId>org.ocpsoft.rewrite</groupId>
    <artifactId>rewrite-servlet</artifactId>
    <version>3.4.1.Final</version>
</dependency>
<dependency>
    <groupId>org.ocpsoft.rewrite</groupId>
    <artifactId>rewrite-integration-faces</artifactId>
    <version>3.4.1.Final</version>
</dependency>
<dependency>
    <groupId>org.ocpsoft.rewrite</groupId>
    <artifactId>rewrite-config-prettypfaces</artifactId>
    <version>3.4.1.Final</version>
</dependency>
<dependency>
    <groupId>org.primefaces</groupId>
    <artifactId>primefaces</artifactId>
    <version>6.1</version>
</dependency>
```

JSF - Balises de base

1	h:inputText Rend une entrée HTML de type="text", zone de texte.
2	h:inputSecret Rend une entrée HTML de type="password"
3	h:inputTextarea Affiche un champ de zone de texte HTML.
4	h:inputHidden Rend une input HTML de type="hidden".
5	h:selectBooleanCheckbox Affiche une seule case à cocher HTML.
6	h:selectManyCheckbox Affiche un groupe de cases à cocher HTML.
7	h:selectOneRadio Affiche un seul bouton radio HTML.

8	h:selectOneListbox Affiche une seule zone de liste HTML.
9	h:selectManyListbox Affiche une zone de liste multiple HTML.
10	h:selectOneMenu Affiche une zone de liste déroulante HTML.
11	h:outputText Rend un texte HTML.
12	h:outputFormat Rend un texte HTML. Il accepte des paramètres.
13	h:graphicImage Rend une image.
14	h:outputScript Inclut une feuille de style CSS dans l'ouput HTML.

Intégration de JSF avec Spring Boot

```
<build>
    <outputDirectory>src/main/webapp/WEB-INF/classes</outputDirectory>
    <!-- plugins... -->
</build>
```

Configuration JSF



A screenshot of a Java code editor showing the file `Application.java`. The code is annotated with line numbers from 10 to 38. It imports various Spring Framework and JSF annotations, including `Bean`, `ComponentScan`, `FacesServlet`, `DispatcherType`, and `EnumSet`. The class `Application` extends `SpringBootServletInitializer` and contains methods for `main`, `ServletRegistrationBean`, and `FilterRegistrationBean`. The `ServletRegistrationBean` method creates a `FacesServlet` instance and returns a `ServletRegistrationBean` for it. The `FilterRegistrationBean` method creates a `RewriteFilter` instance and sets its dispatcher types to `FORWARD`, `REQUEST`, `ASYNC`, and `ERROR`, then adds a URL pattern of `/*`.

```
10 import org.springframework.context.annotation.Bean;
11 import org.springframework.context.annotation.ComponentScan;
12
13 import javax.faces.webapp.FacesServlet;
14 import javax.servlet.DispatcherType;
15 import java.util.EnumSet;
16
17 @EnableAutoConfiguration
18 @ComponentScan({"com.auth0.samples.bootfaces"})
19 public class Application extends SpringBootServletInitializer {
20
21     public static void main(String[] args) {
22         SpringApplication.run(Application.class, args);
23     }
24
25     @Bean
26         public ServletRegistrationBean servletRegistrationBean() {
27             FacesServlet servlet = new FacesServlet();
28             return new ServletRegistrationBean(servlet, "*.jsf");
29
30     @Bean
31         public FilterRegistrationBean rewriteFilter() {
32             FilterRegistrationBean rwFilter = new FilterRegistrationBean(new RewriteFilter());
33             rwFilter.setDispatcherTypes(EnumSet.of(DispatcherType.FORWARD, DispatcherType.REQUEST,
34                                         DispatcherType.ASYNC, DispatcherType.ERROR));
35             rwFilter.addUrlPatterns("/*");
36         }
37     }
38 }
```

Création d'une application JSF sur Spring Boot

Product.java

```
1 import javax.persistence.Id;
2 import java.math.BigDecimal;
3
4 @Entity
5 public class Product {
6     @Id
7     @GeneratedValue(strategy = GenerationType.AUTO)
8     private Long id;
9
10    @Column
11    private String name;
12
13    @Column
14    private BigDecimal price;
15
16    protected Product() {
17    }
18    public Product(String name, BigDecimal price) {
19        this.name = name;
20        this.price = price;
21    }
22    public Long getId() {
23        return id;
24    }
25    public void setId(Long id) {
26        this.id = id;
27    }
28    public String getName() {
29        return name;
30    }
31    public void setName(String name) {
32        this.name = name;
33    }
34    public BigDecimal getPrice() {
35        return price;
36    }
37    public void setPrice(BigDecimal price) {
38}
```

application.properties

```
1 server.port = 8090
2 spring.jpa.hibernate.ddl-auto=update
3 spring.datasource.url=jdbc:mysql://localhost:3306/products
4 spring.datasource.username=root
5 spring.datasource.password=
```

```
package com.auth0.samples.bootfaces;

import org.springframework.data.jpa.repository.JpaRepository;

public interface ProductRepository extends JpaRepository<Product, Long> {
}
```

Construire l'interface JSF pour créer des produits

The image shows a Java IDE interface with two code editors side-by-side.

layout.xhtml

```
1 <!DOCTYPE html>
2 <html xmlns="http://www.w3.org/1999/xhtml"
3     xmlns:h="http://xmlns.jcp.org/jsf/html"
4     xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
5     xmlns:f="http://xmlns.jcp.org/jsf/core"
6     xmlns:p="http://primefaces.org/ui">
7 <f:view>
8     <h:head>
9         <meta charset="utf-8" />
10        <meta http-equiv="X-UA-Compatible" content="IE=edge" />
11        <meta name="viewport" content="width=device-width, initial-scale=1" />
12        <title>Product</title>
13    </h:head>
14    <h:body>
15        <div class="ui-g">
16            <div class="ui-g-12">
17                <p:toolbar>
18                    <f:facet name="left">
19                        <p:button href="/" value="List of Products" />
20                        <p:button href="/product" value="New Product" />
21                    </f:facet>
22                </p:toolbar>
23            </div>
24            <div class="ui-g-12">
25                <ui:insert name="content" />
26            </div>
27        </div>
28    </h:body>
29 </f:view>
30 </html>
```

ProductController.java

```
1 package com.auth0.samples.bootfaces;
2
3 import org.ocpsoft.rewrite.annotation.Join;
4 import org.ocpsoft.rewrite.el.ELBeanName;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.context.annotation.Scope;
7 import org.springframework.stereotype.Component;
8
9 @Scope(value = "session")
10 @Component(value = "productController")
11 @ELBeanName(value = "productController")
12 @Join(path = "/product", to = "/product-form.jsf")
13 public class ProductController {
14     @Autowired
15     private ProductRepository productRepository;
16
17     private Product product = new Product();
18
19     public String save() {
20         productRepository.save(product);
21         product = new Product();
22         return "/product-list.xhtml?faces-redirect=true";
23     }
24
25     public Product getProduct() {
26         return product;
27     }
28 }
```

product-form.xhtml

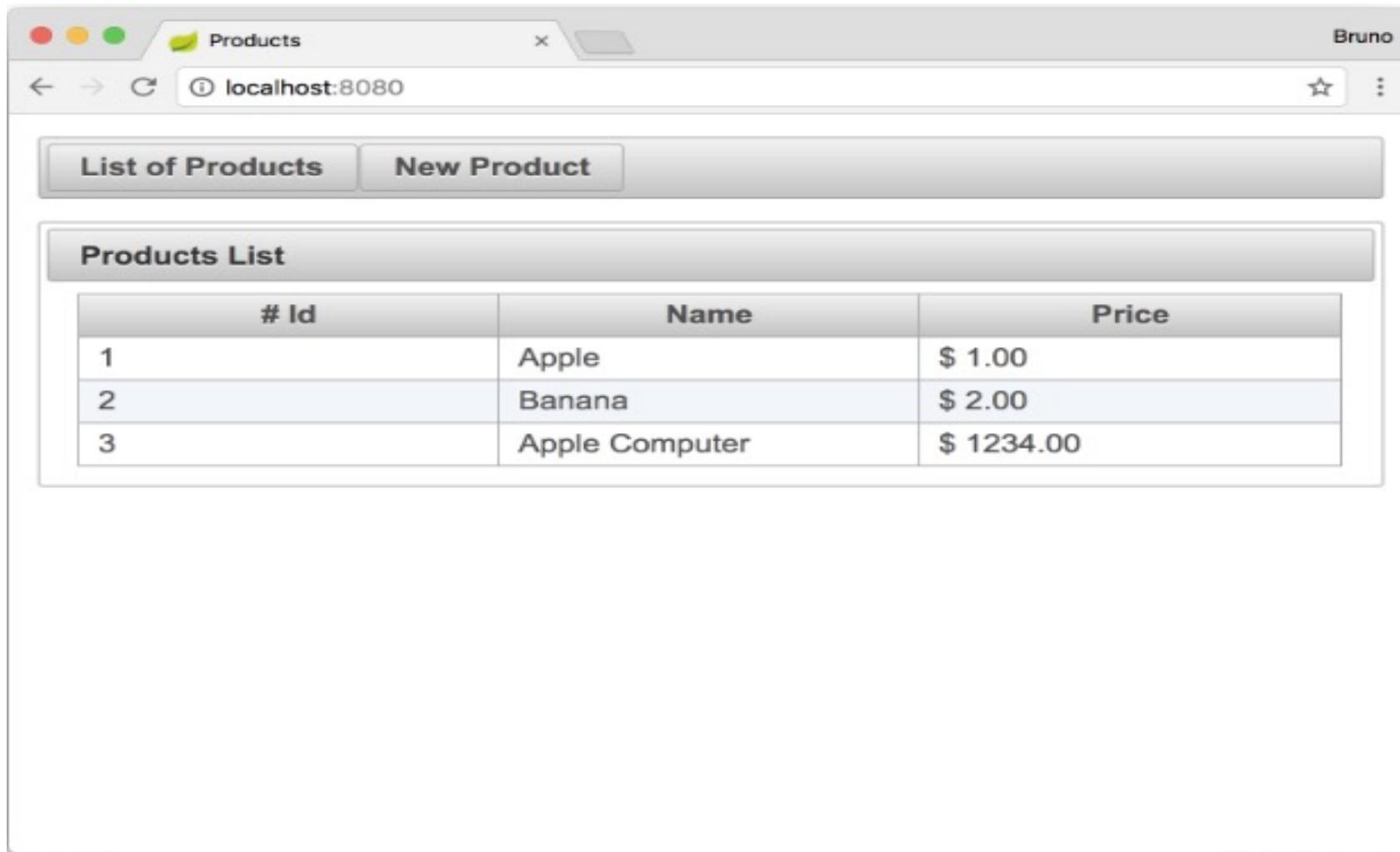
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4<html xmlns="http://www.w3.org/1999/xhtml"
5       xmlns:h="http://xmlns.jcp.org/jsf/html"
6       xmlns:ui="http://xmlns.jcp.org/jsf/facelets" xmlns:p="http://primefaces.org/ui">
7<ui:composition template="layout.xhtml">
8    <ui:define name="content">
9      <h:form id="productForm">
10        <p:panel header="Product Details">
11          <h:panelGrid columns="1">
12            <p:outputLabel for="name" value="Name: " />
13            <p:inputText id="name" value="#{productController.product.name}" />
14            <p:outputLabel for="price" value="Price: " />
15            <p:inputNumber id="price" value="#{productController.product.price}" />
16            <h:commandButton value="Save" action="#{productController.save}" />
17          </h:panelGrid>
18        </p:panel>
19      </h:form>
20    </ui:define>
21  </ui:composition>
22 </html>
```

Création de l'interface JSF pour la liste des produits

```
ProductListController.java ☐
1 import org.ocpsoft.rewrite.faces.annotation.Deferred;
2 import org.ocpsoft.rewrite.faces.annotation.IgnorePostback;
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.context.annotation.Scope;
5 import org.springframework.stereotype.Component;
6
7 import java.util.List;
8
9 @Scope (value = "session")
10 @Component (value = "productList")
11 @ELBeanName(value = "productlist")
12 @Join(path = "/", to = "/product-list.jsf")
13 public class ProductListController {
14     @Autowired
15     private ProductRepository productRepository;
16
17     private List<Product> products;
18
19     @Deferred
20     @RequestAction
21     @IgnorePostback
22     public void loadData() {
23         products = productRepository.findAll();
24     }
25
26     public List<Product> getProducts() {
27         return products;
28     }
29 }
```

```
product-list.xhtml ☐
1 <?xml version="1.0" encoding="UTF-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4 <html xmlns="http://www.w3.org/1999/xhtml"
5       xmlns:ui="http://xmlns.jcp.org/jsf/faces"
6       xmlns:h="http://xmlns.jcp.org/jsf/html"
7       xmlns:f="http://xmlns.jcp.org/jsf/core"
8       xmlns:p="http://primefaces.org/ui">
9 <ui:composition template="layout.xhtml">
10    <ui:define name="content">
11        <h:form id="form">
12            <p:panel header="Products List">
13                <p:dataTable id="table" var="product" value="#{productList.products}">
14                    <p:column>
15                        <f:facet name="header"># Id</f:facet>
16                        <h:outputText value="#{product.id}" />
17                    </p:column>
18
19                    <p:column>
20                        <f:facet name="header">Name</f:facet>
21                        <h:outputText value="#{product.name}" />
22                    </p:column>
23
24                    <p:column>
25                        <f:facet name="header">Price</f:facet>
26                        <h:outputText value="#{product.price}">
27                            <f:convertNumber type="currency" currencySymbol="$ " />
28                        </h:outputText>
29                    </p:column>
30                </p:dataTable>
31            </p:panel>
32        </h:form>
33    </ui:define>
34 </ui:composition>
35 </html>
```

Résultat

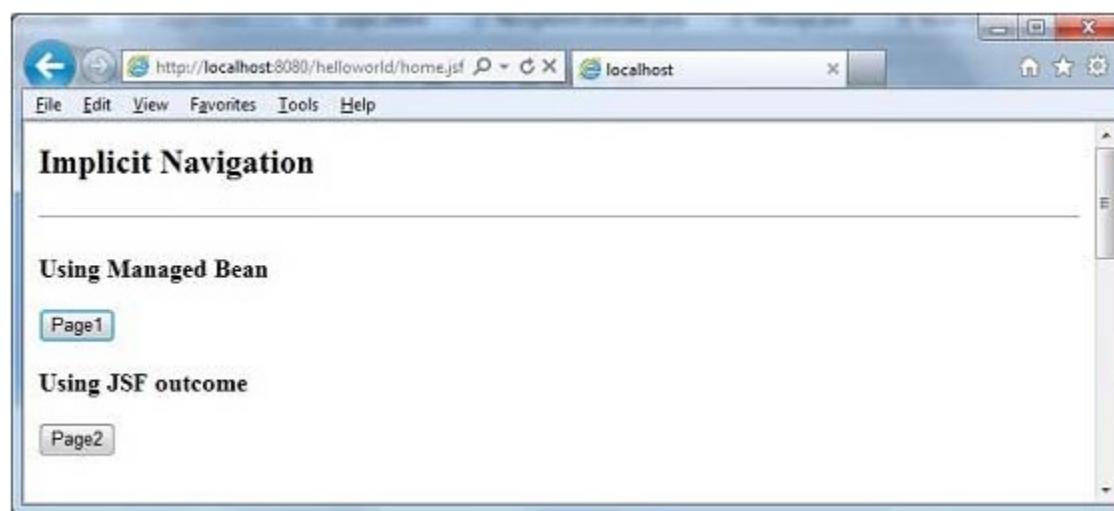


Les règles de navigation

- Les règles de navigation sont les règles fournies par JSF Framework qui décrivent quelle vue doit être affichée lorsqu'un bouton ou un lien est cliqué.
- Les règles de navigation peuvent être définies dans le fichier de configuration JSF nommé faces-config.xml. Ils peuvent être définis dans des beans gérés.
- Les règles de navigation peuvent contenir des conditions sur la base desquelles la vue résultante peut être affichée.

Les Types de règles de navigation: Navigation implicite

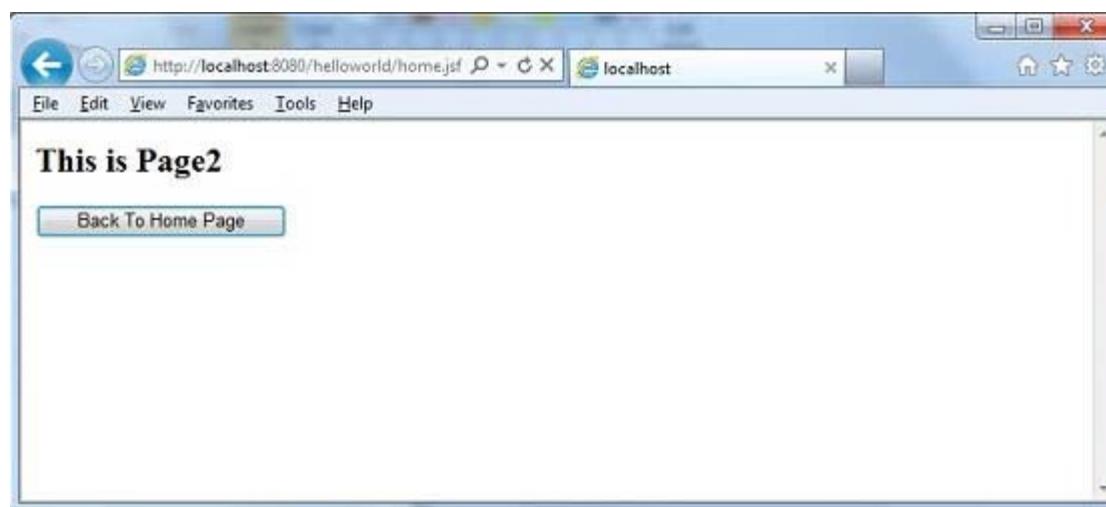
- JSF 2.0 fournit un mécanisme de résolution de page d'affichage automatique nommé navigation implicite . Dans ce cas, il vous suffit de mettre le nom de la vue dans l'attribut d'action et JSF recherchera automatiquement la page de vue correcte dans l'application.



Navigation automatique dans la page JSF

- Définissez le nom de la vue dans l'attribut d'action de tout composant d'interface utilisateur JSF.

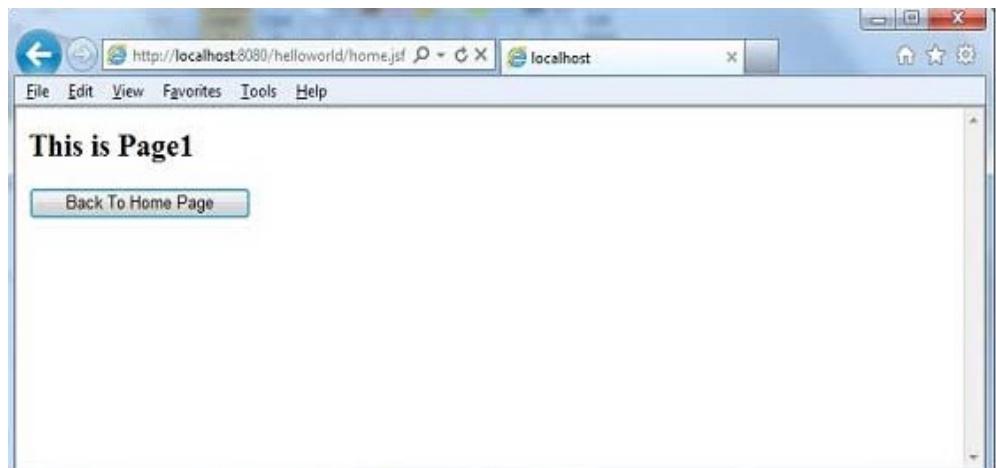
```
<h:form>
    <h3>Using JSF outcome</h3>
    <h:commandButton action = "page2" value = "Page2" />
</h:form>
```



Navigation automatique dans Managed Bean

- Définissez une méthode dans le bean pour renvoyer un nom de vue

```
@ManagedBean(name = "navigationController", eager = true)  
@RequestScoped  
  
public class NavigationController implements Serializable {  
    public String moveToPage1() {  
        return "page1";  
    }  
}
```



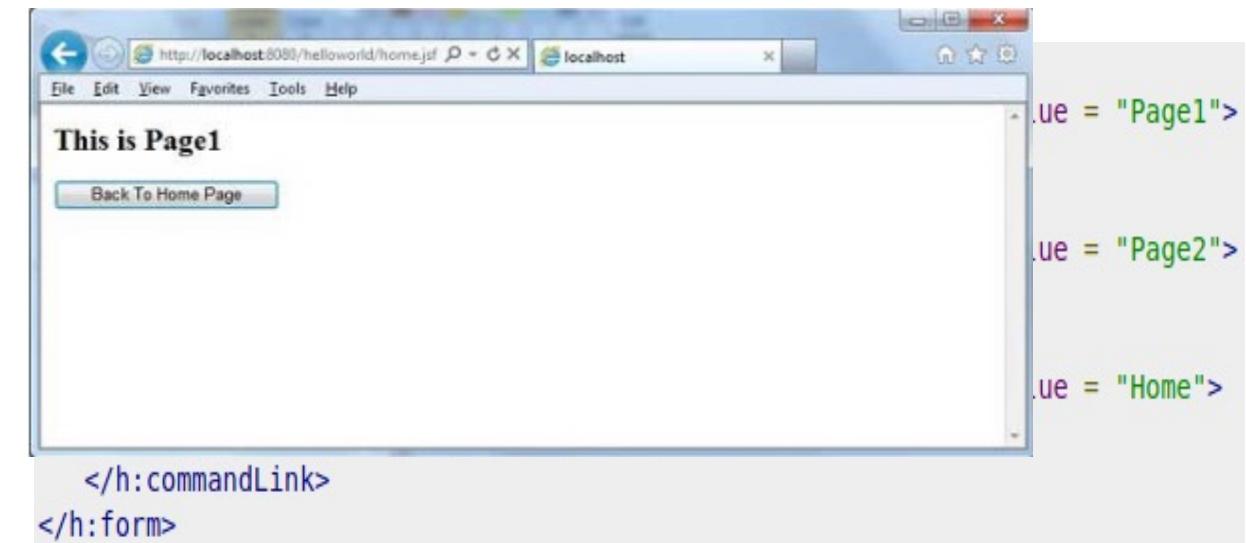
```
<h:form>  
    <h3> Using Managed Bean</h3>  
    <h:commandButton action = "#{navigationController.moveToPage1}"  
        value = "Page1" /glt;  
</h:form>
```

Navigation conditionnelle

```
@ManagedBean(name = "navigationController", eager = true)
@RequestScoped

public class NavigationController implements Serializable {
    //this managed property will read value from request parameter pageId
    @ManagedProperty(value = "#{param.pageId}")
    private String pageId;

    public String navigate() {
        if(pageId == null) {
            return "page1";
        } else if(pageId.equals("2")) {
            return "page2";
        } else {
            return "home";
        }
    }
}
```

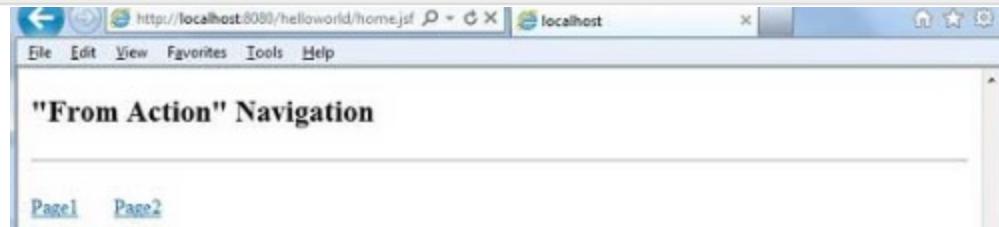
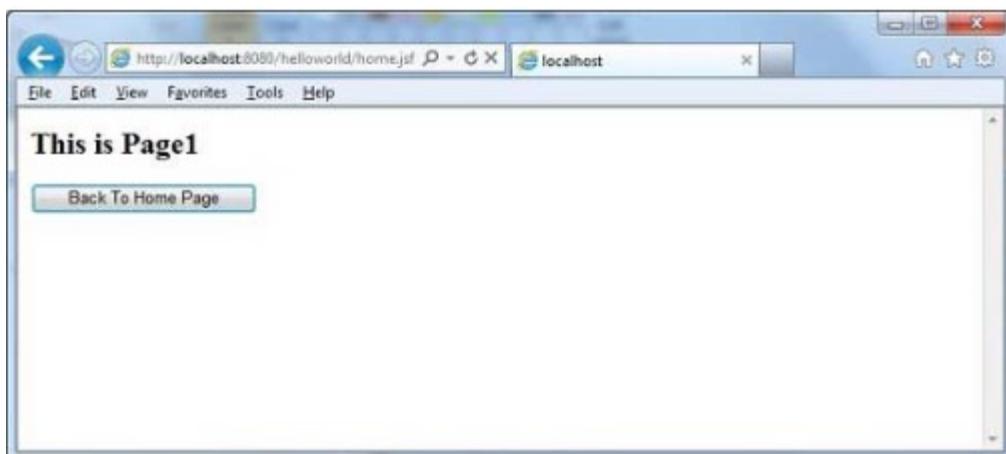


Résolution de la navigation basée sur from-action

- JSF fournit une option de résolution de navigation même si les différentes méthodes du managed bean renvoient le même nom de vue.

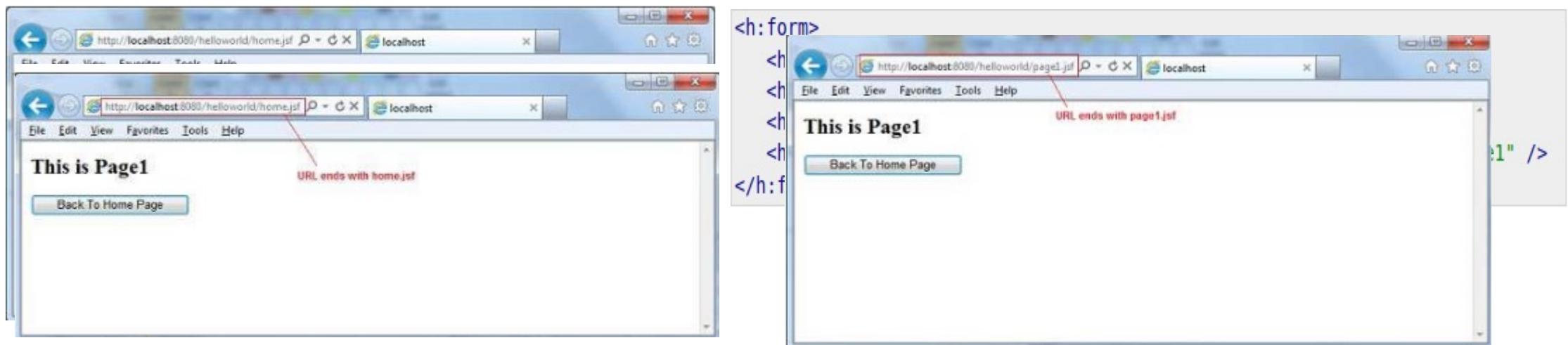
```
public String processPage1() {  
    return "page";  
}  
public String processPage2() {  
    return "page";  
}
```

```
<navigation-rule>  
    <from-view-id>home.xhtml</from-view-id>  
  
    <navigation-case>  
        <from-action>#{navigationController.processPage1}</from-action>  
        <from-outcome>page</from-outcome>  
        <to-view-id>page1.jsf</to-view-id>  
    </navigation-case>  
  
    <navigation-case>  
        <from-action>#{navigationController.processPage2}</from-action>  
        <from-outcome>page</from-outcome>  
        <to-view-id>page2.jsf</to-view-id>  
    </navigation-case>  
  
</navigation-rule>
```



Transférer vs Rediriger

- JSF par défaut effectue un transfert de page de serveur lors de la navigation vers une autre page et l'URL de l'application ne change pas.
- Pour activer la redirection de page, ajoutez faces-redirect=true à la fin du nom de la vue.

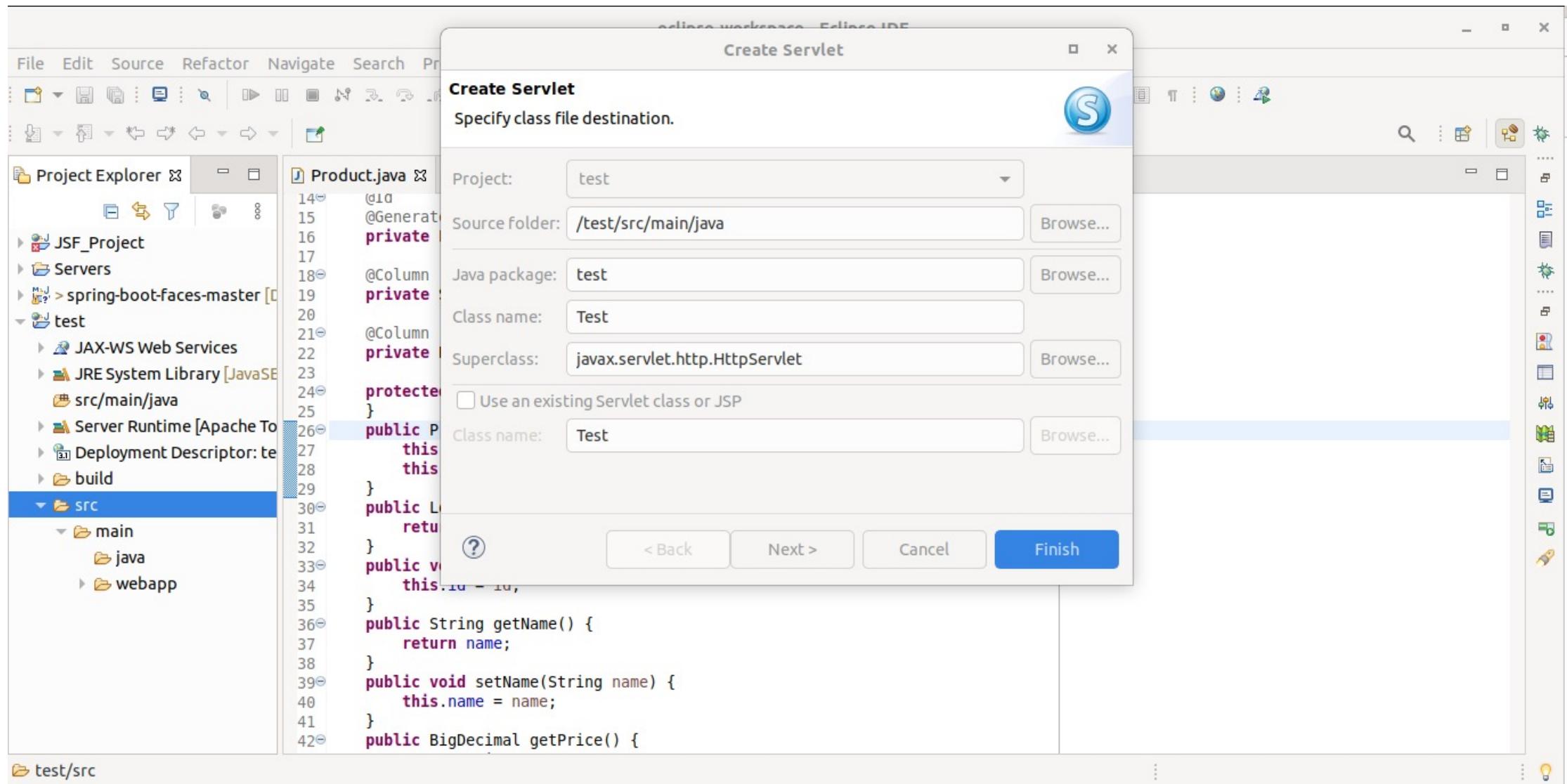




Développer des vues : JSP



Comprendre les Servlets et les JSPs



JSP : définition

- La JSP (Java Server Page) est une technologie Java qui permet la génération de pages web dynamiques.
- La technologie JSP permet de séparer la présentation sous forme de code HTML et les traitements écrits en Java sous la forme de JavaBeans ou de servlets.



Test.java

```
1 package test;
2
3④ import java.io.IOException;
4 import javax.servlet.ServletException;
5 import javax.servlet.annotation.WebServlet;
6 import javax.servlet.http.HttpServlet;
7 import javax.servlet.http.HttpServletRequest;
8 import javax.servlet.http.HttpServletResponse;
9
10
11 @WebServlet("/Test")
12 public class Test extends HttpServlet {
13     private static final long serialVersionUID = 1L;
14
15⑤     public Test() {
16         super();
17     }
18⑥     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
19         response.getWriter().append("Served at: ").append(request.getContextPath());
20     }
21
22
23⑦     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
24         doGet(request, response);
25     }
26
27 }
28
```

web.xml

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://JAVA.sun.com/xml/ns/javaee" xsi:schemaLocation="http:
3     <servlet>
4         <servlet-name>Test</servlet-name>
5         <servlet-class>com.test.servlets.Test</servlet-class>
6     </servlet>
7     <servlet-mapping>
8         <servlet-name>Test</servlet-name>
9         <url-pattern>/bonjour</url-pattern>
10    </servlet-mapping>
11 </web-app>
```



The screenshot shows an IDE interface with two tabs: "bonjour.jsp" and "Test.java".

bonjour.jsp:

```
1 <%@ page language="java" contentType="text/html; charset=UTF-8"
2     pageEncoding="UTF-8"%>
3 <!DOCTYPE html>
4<html>
5<head>
```

Test.java:

```
1 package com.test.servlet;
2
3 import java.io.IOException;
4
5 @WebServlet("/Test")
6 public class Test extends HttpServlet {
7     private static final long serialVersionUID = 1L;
8
9     public Test() {
10         super();
11     }
12
13     protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
14         response.getWriter().println("Bonjour !");
15     }
16
17     protected void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException {
18     }
19
20 }
21
22 }
```

The browser preview panel displays the output of the Java code. It shows the URL `http://localhost:8090/test/bonjour` and the resulting page content "Bonjour !".

JSP: Balises et directives

- <%-- --%> est un commentaire JSP
- <%! %> est une balise de déclaration d'une variable;
- <% %> est une balise d'inclusion de code Java quelconque....;
- <%= %> est une balise d'affichage.
- <%@ page import="java.util.Date" %> pour importer des packages Java
- <%@ taglib uri="NsyTagLib.tld" prefix="nsy" %> permet d'ajouter des tags libraries ou bibliothèques de balises
- <%@ include file="header.jsp" %> permet de composer des pages HTML à partir de plusieurs fragments





Conserver des sessions utilisateurs

Les Sessions

- 1. Qu'est-ce que la session ?
- 2. Gestion de session dans Java
- 3. Obtenir ou créer une session
- 4. Liaison de données à une session
- 5. Configurer le délai d'expiration de la session
- 6. Invalider une session

1. Qu'est-ce que la session

- Un moyen de conserver des données relatives à un visiteur sur toutes les pages de notre site.
- Ces données seront enregistrées sur le serveur
- Pour chaque session, on peut associer une infinité de variables de session de tout type (primitif, objet, tableau d'objets...)
- Ces variables seront détruites, si le visiteur se déconnecte, s'il dépasse une certaine durée...

2. Gestion de session dans Java

- En Java, un objet **HttpSession** représente la session d'un utilisateur particulier.
- Vous pouvez stocker des informations relatives à l'utilisateur dans une session sous forme de paires clé et valeur. L' interface HttpSession définit la méthode setAttribute(key, value) pour stocker une entrée clé-valeur et la méthode getAttribute(key) pour obtenir la valeur d'une clé spécifiée.
- Par défaut, Java utilise des cookies pour le suivi de session.
- Un cookie portant le nom JSESSIONID est stocké temporairement dans le navigateur Web. Il est utilisé pour identifier le même utilisateur à travers différentes requêtes,

3. Obtenir ou créer une session

- La méthode `getSession()` renvoie la session actuelle associée à la requête

```
1 | protected void doGet(HttpServletRequest request, HttpServletResponse response)
2 |   throws ServletException, IOException {
3 |     HttpSession session = request.getSession();
4 |
5 |     // work with the session...
6 | }
```

4. Liaison de données à une session

- Pour stocker une valeur dans une session, utilisez la méthode setAttribute(key, value) de l' objet HttpSession . Par exemple, l'instruction suivante stocke le nom de l'utilisateur :

```
1 | session.setAttribute("username", "Daniel Tran");
```

- Vous pouvez stocker n'importe quel type d'objet dans la session.

```
1 | List<Student> students = studentDao.getStudents();  
2 | session.setAttribute("listStudent", students);
```

5. Configurer le délai d'expiration de la session

- Vous pouvez définir le délai d'expiration de la session pour une application Web individuelle en modifiant son fichier de descripteur de déploiement Web (web.xml). Par exemple:

```
1 | <?xml version="1.0" encoding="UTF-8"?>
2 | <web-app...>
3 |
4 |   <session-config>
5 |     <session-timeout>15</session-timeout>
6 |   </session-config>
7 |
8 | </web-app>
```

- Vous pouvez définir une valeur de délai d'expiration pour une session individuelle par programmation comme ceci :

```
1 | session.setMaxInactiveInterval(300);
```

6. Invalider une session

- Par défaut, une session n'est détruite qu'une fois que l'utilisateur est resté inactif pendant un certain délai.
- Au cas où vous voudriez détruire une session individuelle immédiatement, appelez la méthode invalidate() comme ceci :
- ```
1 | session.invalidate();
```

# Les sessions

```
Personne personne = new Personne (100, "wick", "john");
```

- Pour créer une variable session

```
HttpSession session = request.getSession();
```

- Pour ajouter l'objet personne dans la session

```
session.setAttribute("perso", personne);
```

- Pour récupérer une donnée de session dans une Servlet

```
session.getAttribute("perso");
```

- Pour récupérer une donnée de session (dans une JSP)

```
<%
```

```
Personne p = (Personne) session.getAttribute("perso");
```

```
out.print(p);
```

```
%>
```

- Pour supprimer une valeur associée à une clé de la session, utilisez la méthode removeAttribute(key) :

```
session.removeAttribute("username");
```

READY FOR A  
QUIZ?

61



Développer des composants réutilisables et AJAX

# Qu'est-ce que le HTTP ?

- HTTP est un protocole qui nous permet d'envoyer des informations sur le Web. Par exemple, lorsque votre navigateur demande <http://www.googme.com>, il envoie en fait une requête HTTP et gère ensuite la réponse (la page d'accueil de google).
- Les requêtes HTTP se présentent sous différentes formes, utilisant différents verbes (GET, POST, PUT et DELETE sont les quatre plus courants). Le type de requête que fait votre navigateur est une requête GET. Une requête GET signifie généralement que nous aimerais que le serveur nous envoie des données.

# AJAX c'est quoi ?

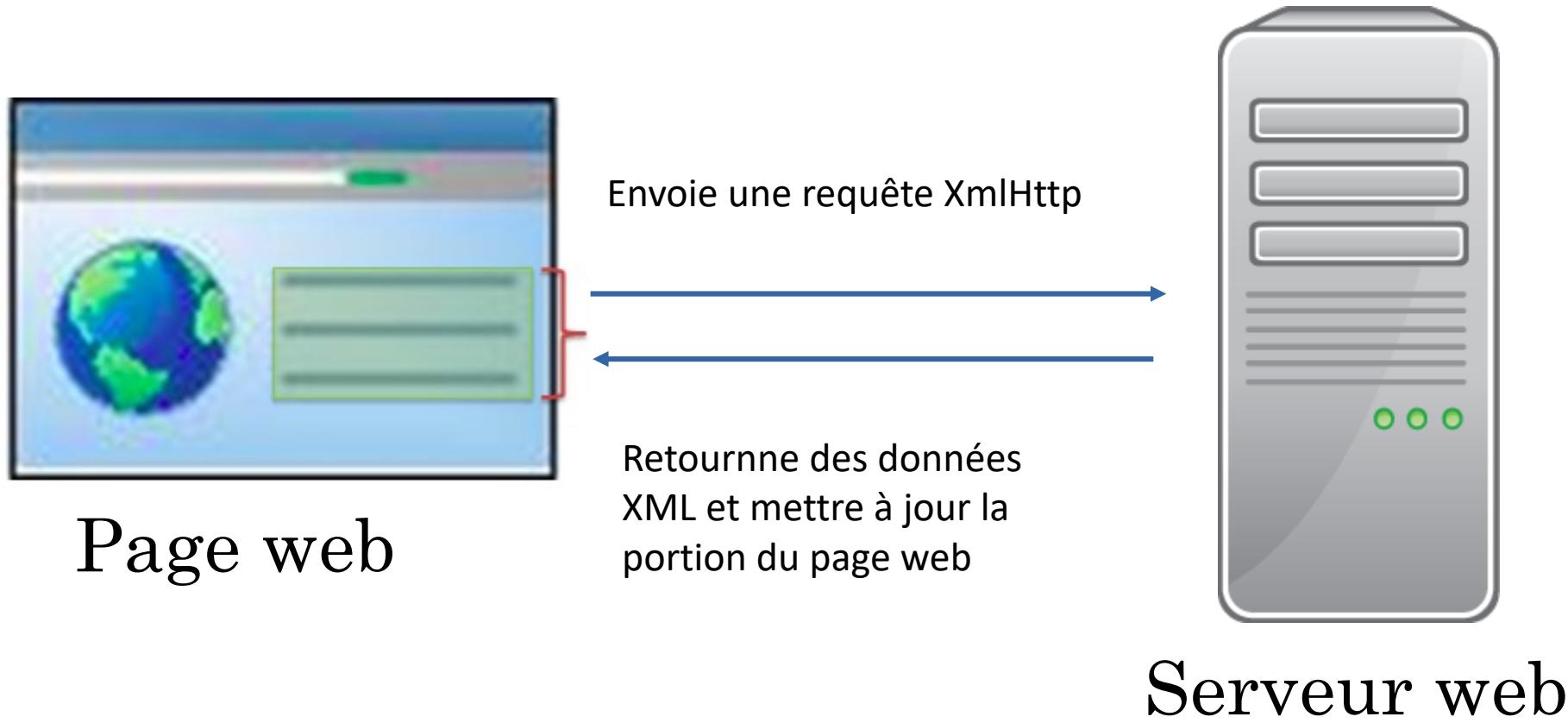
- AJAX est un acronyme qui désigne Asynchronous Javascript And XML (qui signifie littéralement Javascript et XML Asynchrones). Il ne s'agit pas d'une technologie en soi mais plutôt d'un ensemble de technologies qui permettent la mise à jour du contenu d'une page Web d'une manière rapide et sans chargement complet de celle ci,
  - AJAX s'appuie sur les technologies suivantes:
    - 1) Javascript et DOM pour le traitement.
    - 2) XML ou JSON pour l'extraction des données (le contenu à afficher).
    - 3) HTML et CSS pour la présentation.

# Ajax : Principes

- Communication entre une page et un serveur sans chargement de la page
- JavaScript en charge de la connexion entre la page et le serveur



# Fonctionnalité d'Ajax



# Utilité d'AJAX

- Parmi les applications les plus courantes d'AJAX on trouve:
- Rechargement d'une (ou plusieurs) zone de la page Web sans être obligé de recharger celle-ci en entier (ce qui est d'ailleurs l'essence d'AJAX).
- Suggestion automatique lors de la saisi d'un champ, comme c'est le cas quand vous saisissez des mots clés sur le moteur de recherche Google.
- Sauvegarde automatique d'un texte dans la base de données lors de la saisie, sans pour autant quitter le champs dans lequel on est entrain d'écrire.
- Upload d'un fichier sur le serveur tout en visualisant l'état de progression du chargement.
- Affichage d'un contenu supplémentaire quand on atteint le bas de la page avec la barre de défilement, comme c'est le cas sur Youtube.
- Exécution d'opérations telles que la suppression ou la modification suite à un glisser déposer sans quitter la vue en cours.

# Exécution d'une requête GET avec AJAX à l'aide de jQuery

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="utf-8">
5 <title>jQuery get() Demo</title>
6 <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
7 <script>
8 $(document).ready(function(){
9 $("button").click(function(){
10 $.get("date-time.php", function(data){
11 // Display the returned data in browser
12 $("#result").html(data);
13 });
14 });
15 });
16 </script>
17 </head>
18 <body>
19 <div id="result">
20 <h2>Content of the result DIV box will be replaced by the server date and time</h2>
21 </div>
22 <button type="button">Load Date and Time</button>
23 </body>
24 </html>
```

December 01, 2021 02:20:31 PM  
Date et heure de chargement

```
1 <?php
2 // Return current date and time from the server
3 echo date("F d, Y h:i:s A");
4 ?>
```

# Exécution d'une requête POST avec AJAX à l'aide de jQuery

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4 <meta charset="utf-8">
5 <title>jQuery post() Demo</title>
6 <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
7 </script>
8 $(document).ready(function(){
9 $("form").submit(function(event){
10 // Stop form from submitting normally
11 event.preventDefault();
12
13 /* Serialize the submitted form control values to be sent to the web server with
the request */
14 var formValues = $(this).serialize();
15
16 // Send the form data using post
17 $.post("display-comment.php", formValues, function(data){
18 // Display the returned data in browser
19 $("#result").html(data);
20 });
21 });
22 });
23 </script>
24 </head>
25 <body>
26 <form>
27 <label>Name: <input type="text" name="name"></label>
28 <label>Comment: <textarea cols="50" name="comment"></textarea></label>
29 <input type="submit" value="Send">
30 </form>
31 <div id="result"></div>
32 </body>
33 </html>
```

Nom: Fatma

rien

Commenter:

Envoyer

Salut Fatma. Votre commentaire a été reçu avec succès.  
Voici le commentaire que vous avez entré : rien

```
1 <?php
2 $name = htmlspecialchars($_POST["name"]);
3 $comment = htmlspecialchars($_POST["comment"]);
4 echo "Hi, $name. Your comment has been received successfully." . "";
5 echo "Here's the comment what you've entered: $comment";
6 ?>
```



