

Débuter pas à pas avec Wicket et Eclipse

par Philippe Sénéchal ([Accueil](#))

Date de publication : 27 Novembre 2008

Dernière mise à jour :

Wicket est un framework Java utilisé pour développer des applications web. Ce tutoriel s'adresse à tous ceux qui souhaitent découvrir le développement d'applications web à l'aide de ce framework sans connaissance préalable de JEE.

I - Préambule.....	3
II - Installation d'un projet Wicket.....	3
II-A - Installation de Maven 2.....	3
II-B - Installation de Wicked Shell.....	4
II-C - Utilisation du QuickStart.....	4
II-D - Ouverture du projet avec Eclipse.....	5
III - Installation d'un conteneur de servlet.....	9
III-A - Installation de Jetty.....	9
III-B - Installation de Tomcat.....	10
IV - Principes de Wicket.....	14
IV-A - Le fichier de configuration.....	14
IV-B - La classe d'initialisation.....	15
IV-C - Une page web avec wicket.....	15
V - Personnalisation de notre application.....	15
V-A - Création d'un gabarit.....	16
V-B - Personnalisation de l'accueil.....	17
V-C - Mise en page avec le principe dénommé Markup Inheritance.....	19
V-D - Titre des pages.....	22
V-E - Utilisation d'un FeedbackPanel.....	22
VI - Session.....	24
VI-A - L'utilisateur.....	24
VI-B - La session.....	25
VI-C - L'accès à la session.....	26
VI-D - La navigation.....	27
VI-E - Les liens.....	29
VII - Conclusion.....	32
VIII - Ressources.....	32
IX - Remerciements.....	32

I - Préambule

La complexité et la multiplicité des modèles de programmation que l'on peut utiliser dans JEE créent une étape difficile à franchir quand on veut aborder ce domaine après un apprentissage du langage java. Wicket représente à mon avis une possibilité de découvrir les notions de base de la programmation d'application web en se servant quasi exclusivement de ses connaissances en java et en Html. Ce tutoriel s'applique à suivre cette démarche avec une approche pas à pas de la mise en oeuvre de wicket et de ses mécanismes de base .

II - Installation d'un projet Wicket

Une application web Java est en général contenue dans un fichier war (Web Application aRchive) et présente la structure suivante :


répertoire racine


```


├─+META-INF
├─+WEB-INF
│   └─+classes : les servlet (classes traitant les requêtes HTTP) et autres classes
│   └─+lib : les librairies utilisées par l'application (fichiers .jar)
│   └─web.xml : le descripteur de déploiement de l'application
└─ressources (pages html, images, fichiers css...)
    
```

Nous allons nous servir d'Eclipse Ganymede version JEE pour créer notre projet d'application web utilisant wicket. Il va donc falloir créer cette structure, et référencer les librairies nécessitées par notre application. On peut le faire simplement directement à partir d'Eclipse selon la méthode décrite ici : **création d'une application wicket avec eclipse**

Wicket a toutefois prévu une méthode de construction de projet appelée QuickStart que l'on peut trouver sur son site à l'adresse suivante : <http://wicket.apache.org/quickstart.html>.

Cette méthode utilise un outil de  build appelée Maven2, pour construire ce projet. Il nous faudra donc tout d'abord télécharger et installer Maven 2.

Il va falloir ensuite utiliser la  **ligne de commande** avec l'ordre généré sur le site par le quickstart. On peut le faire directement à partir d'eclipse si on a installé le plugin Wicked Shell qui permet d'utiliser la ligne de commande de Windows avec Eclipse.

 **Attention à ne pas confondre Wicked et Wicket !**

II-A - Installation de Maven 2

On peut télécharger Maven 2 sur cette page : <http://maven.apache.org/download.html#Installation>.

Il faut ensuite décompresser l'archive dans le répertoire de votre choix: ici ce sera : `E:\Programmation`. Maven se trouve alors dans le répertoire: `E:\Programmation\apache-maven-2.0.9`.

Il reste a positionner les variables d'environnement `M2_HOME` et `M2`.

Pour accéder aux variables d'environnement, sous Windows, il faut à partir du menu *démarrer*, cliquer sur *panneau de configuration*, puis sur *Système*, sur l'*onglet avancé*, puis enfin sur le bouton *Variables d'environnement*.

Dans les variables utilisateurs, il faut cliquer sur *nouveau*, et remplir les cases comme suit dans notre cas (à adapter selon votre configuration):

- Nom de la variable : `M2_HOME` Valeur de la variable : `E:\Programmation\apache-maven-2.0.9`
- Nom de la variable : `M2` Valeur de la variable : `%M2_HOME%\bin`
- Nom de la variable : `JAVA_HOME` Valeur de la variable : `C:\Program Files\Java\jdk1.6.0_01` (si elle n'existe pas déjà)
- Mettre à jour la variable : `PATH` en rajoutant la valeur : `%M2%;%Path%`

Sous Linux, il faudra écrire dans la console : `# gedit .profile` sous *gnome*, ou `# kate .profile` sous *KDE*, puis ajouter les lignes suivantes :

Dans la console Linux

```
export M2_HOME=/home/user/apache-maven-2.0.9
export M2=$M2_HOME/bin
export JAVA_HOME=/usr/lib/java/jdk1.6.0_01/
PATH="$M2:$PATH"
```

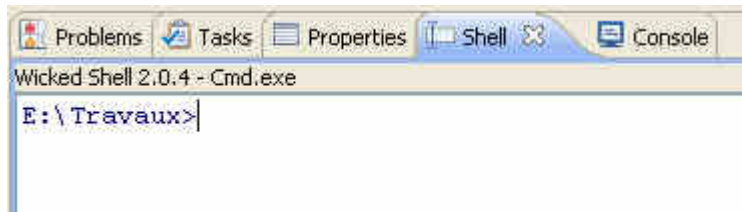
Il existe un plugin maven pour eclipse (m2eclipse) que l'on peut installer à partir de l'update d'éclipse à cette adresse : <http://m2eclipse.sonatype.org/update/>.

II-B - Installation de Wicked Shell

On peut installer Wicked Shell via l'update d'eclipse en suivant les instructions de cette page : <http://www.wickedshell.net/download.html>

Dans le menu *préférence* d'Eclipse, il suffit de cocher la case *Use workspace as startup directory* pour que la ligne de commande pointe sur le répertoire de travail d'Eclipse. Ici, c'est le dossier `E:\Travaux`.

On affiche ensuite la fenêtre de Wicked Shell à l'aide du menu *Show view*:



On peut alors tester l'installation de Maven avec la commande : `mvn --version`, qui si tout est bien installé devrait afficher le numéro de version de Maven.

II-C - Utilisation du QuickStart

On va remplir les 3 lignes suivantes de la page *Quickstart* du site Wicket:

- *GroupId* qui correspondra au package java (ici on l'appellera : `com.tuto`)
- *ArtifactId* qui correspondra au dossier contenant le projet (ici : `project`)
- *Version* qui correspond à la version de Wicket que l'on veut utiliser. (ici : `1.4-SNAPSHOT`)

On obtient alors la ligne de commande suivante que l'on va copier:

```
mvn archetype:create -DarchetypeGroupId=org.apache.wicket -DarchetypeArtifactId=wicket-archetype-quickstart -DarchetypeVersion=1.4-SNAPSHOT -DgroupId=tuto -DartifactId=project -DremoteRepositories=http://wicketstuff.org/maven/repository/
```

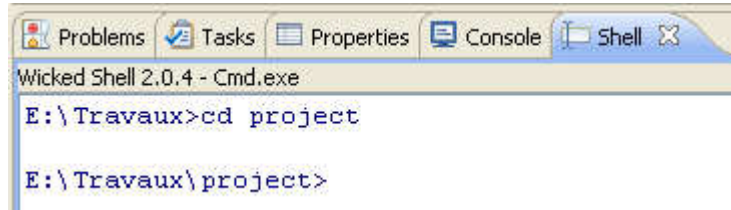
On va ensuite coller cette commande sur la fenêtre de wicked Shell :



En tapant sur *Entrée*, le projet se construit alors dans le dossier : `E:\Travaux\project`.

II-D - Ouverture du projet avec Eclipse

Il va maintenant falloir travailler avec ce projet dans Eclipse J2EE. Pour se placer dans le répertoire de notre projet, on tape la commande suivante : `cd project` dans la fenêtre Wicked Shell.

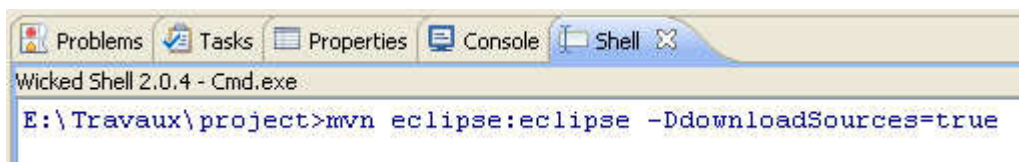


```

Wicked Shell 2.0.4 - Cmd.exe
E:\Travaux>cd project

E:\Travaux\project>
    
```

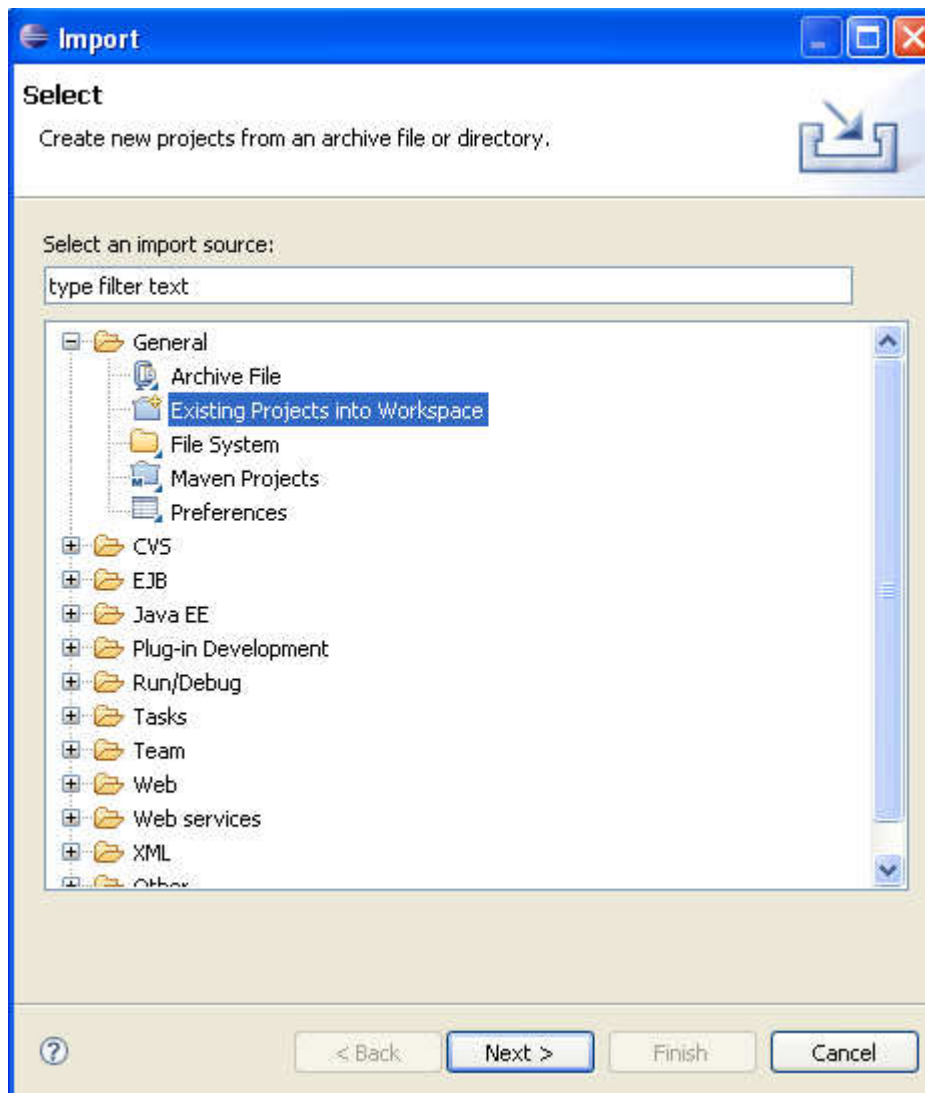
Puis on colle la commande suivante qui permettra l'utilisation du projet par Eclipse : `mvn eclipse:eclipse -DdownloadSources=true`



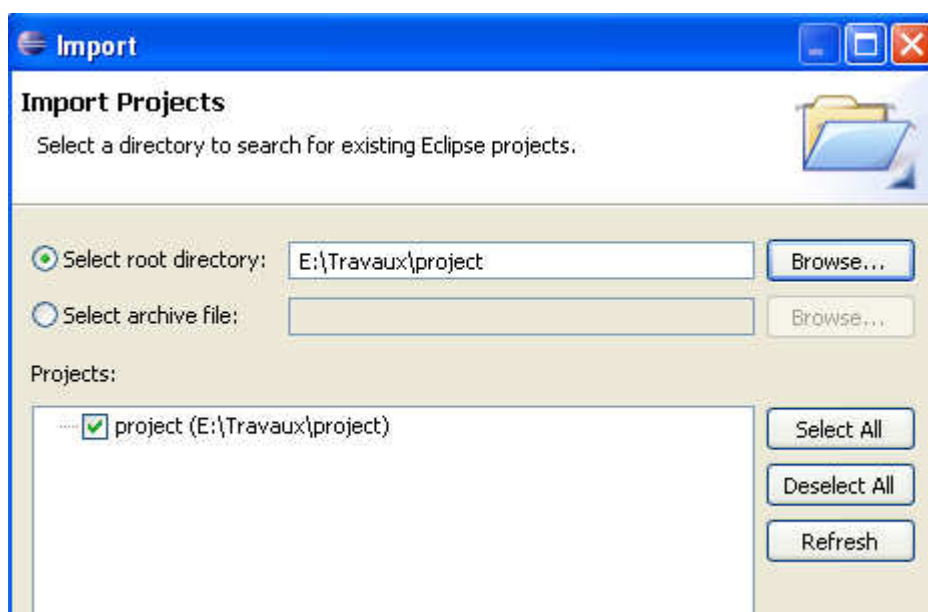
```

Wicked Shell 2.0.4 - Cmd.exe
E:\Travaux\project>mvn eclipse:eclipse -DdownloadSources=true
    
```

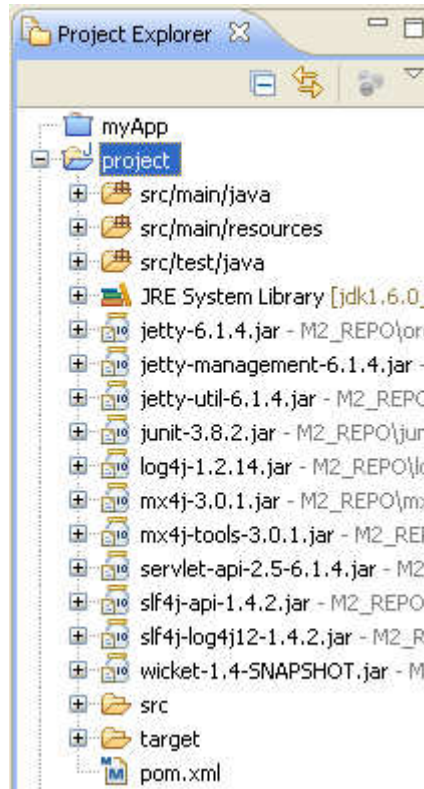
Il faut ensuite ouvrir le menu *File* d'Eclipse puis cliquer sur *Importer* . Dans la fenêtre *Import*, cliquer sur le noeud *General*, puis sur *Existing Project in Workspace* :



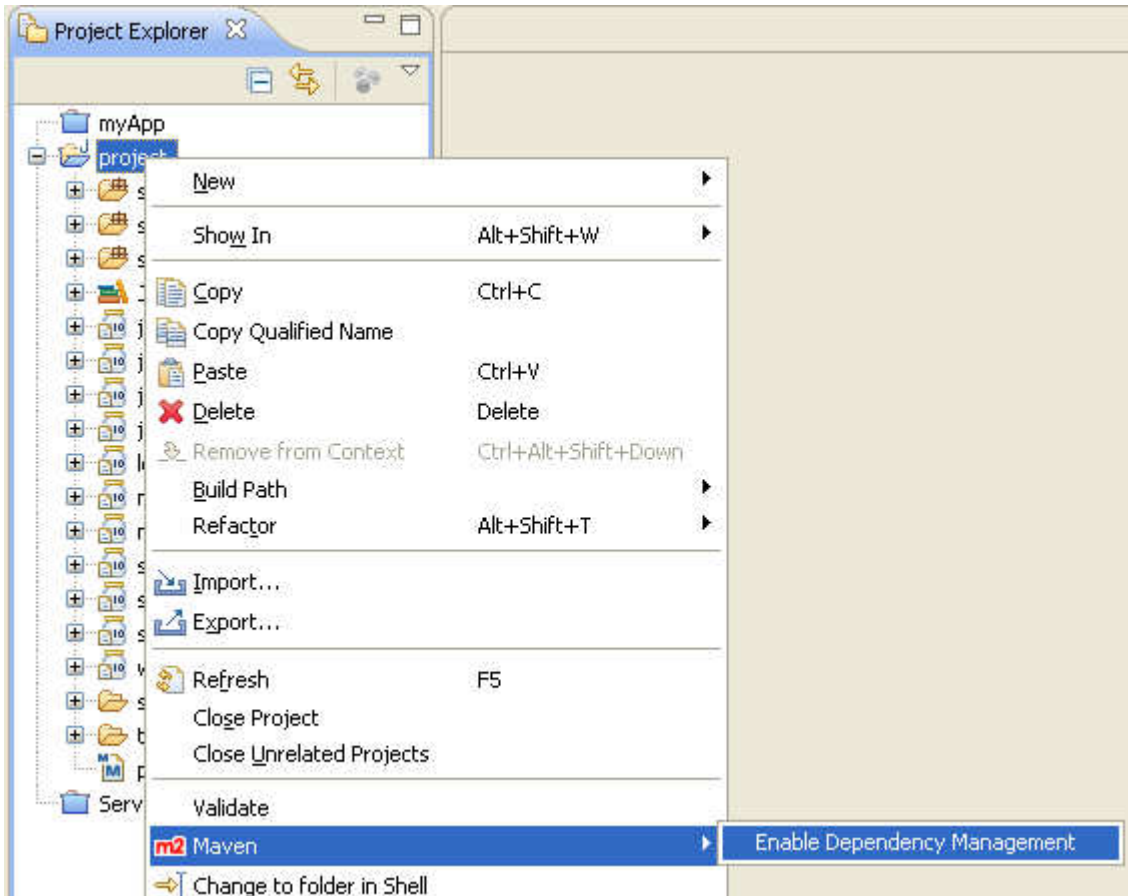
Puis sélectionner le répertoire du projet avant de fermer la fenêtre :



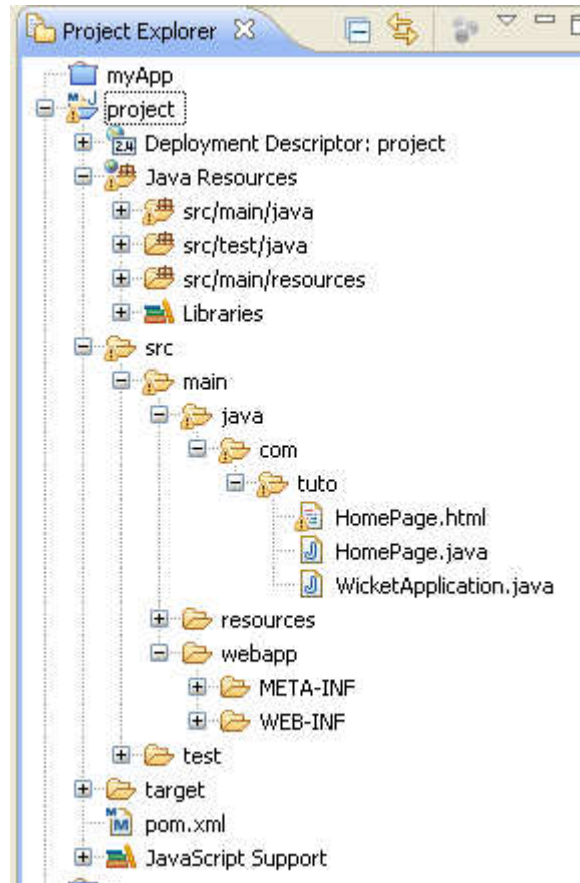
Notre projet est maintenant intégré à Eclipse, et en cliquant sur son nom dans l'onglet *Project Explorer*, on peut visualiser les différents répertoires et fichiers qui le compose.




Il reste ensuite à intégrer les dépendances en cliquant sur *Enable dependency management* comme suit :



Notre projet a alors cette structure :



Le projet est maintenant prêt à l'emploi sous Eclipse, mais pour déployer le projet Wicket sur internet, il nous faut un conteneur de  **servlet**, c'est à dire une application permettant d'exécuter des applications web java sur un serveur web.

III - Installation d'un conteneur de servlet

Je vais ici détailler l'installation de deux conteneurs de servlet : Jetty et Tomcat.

III-A - Installation de Jetty

Jetty est un serveur open source 100% Java. Il est déjà présent dans l'installation que l'on vient de faire avec le QuickStart, puisque l'on trouve ses librairies dans les dépendances maven.

On peut l'utiliser sans autres préparations en tapant dans la fenêtre de Wicked Shell : `mvn jetty:run`. Si tout s'est bien passé, on peut alors visualiser dans notre navigateur la page suivante de notre application Wicket à cette adresse : <http://localhost:8080/project/>



Wicket Quickstart Archetype Homepage


If you see this message wicket is properly configured and running

Jetty a un autre avantage : le hot deployment; c'est à dire que le moteur de servlet va vérifier (selon un intervalle de temps que l'on va définir) si les sources ont été modifiées, et redéployer l'application aussitôt, sans que l'on ait à l'arrêter et à le redémarrer. Pour cela, il faudra rajouter deux lignes dans le fichier de configuration `pom.xml` du projet. Pour pouvoir l'arrêter à partir de la ligne de commande en tapant `mvn jetty:stop`, il va aussi nous falloir rajouter 2 lignes dans ce même fichier.

Ces modifications sont surlignées ici en bleu :



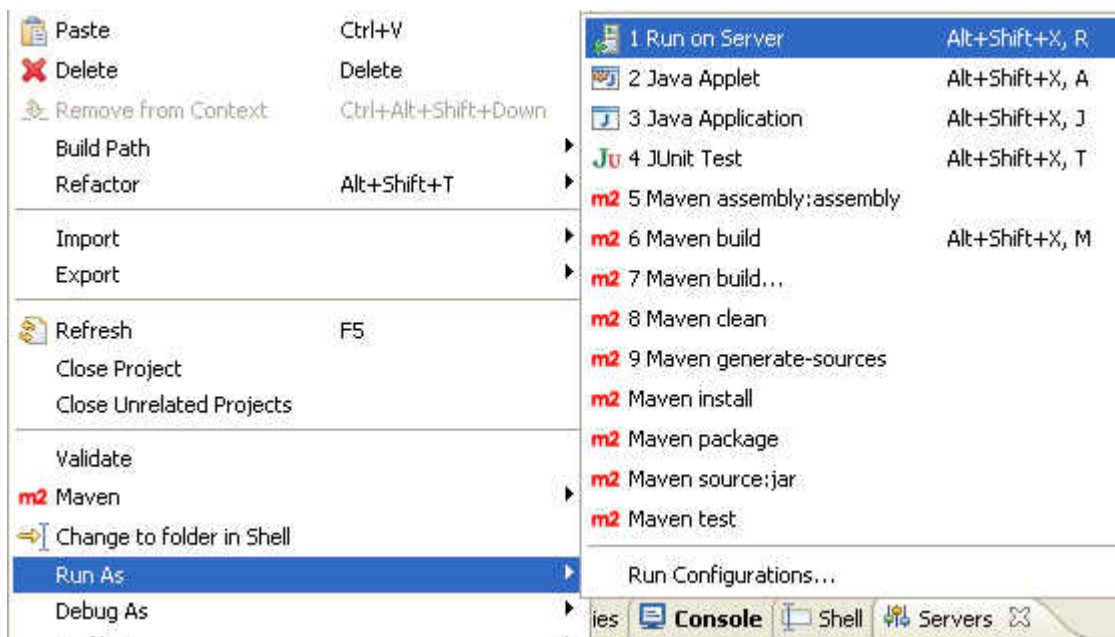
Mais à côté de ces avantages, l'utilisation de Jetty dans la configuration Maven a 2 inconvénients à mon avis : la nécessité d'utiliser la ligne de commande, et l'absence de prévisualisation du résultat dans Eclipse. Pour pallier à cela, on peut installer Jetty selon la méthode utilisée ci-dessous pour l'installation de Tomcat.

 Quand jetty est démarré avec la commande `mvn jetty:run`, l'invite de commande qui pourrait permettre de stopper le serveur n'est plus active. Pour pallier à cela, il peut être utile de lancer la commande `start` dans la fenêtre Wicked shell, avant de démarrer le serveur, ce qui permet d'ouvrir une autre fenêtre de commande à partir de laquelle on pourra arrêter le serveur.

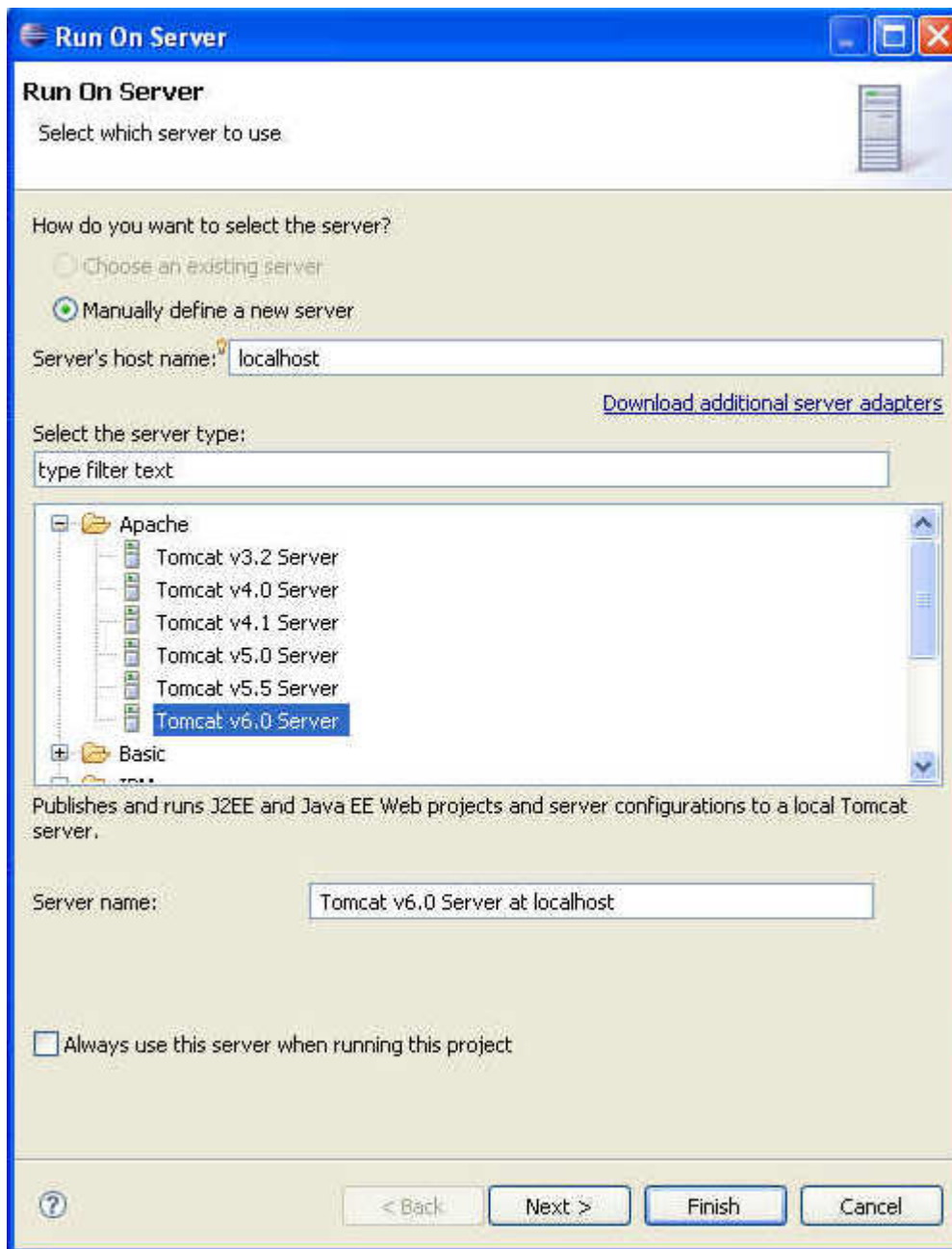
III-B - Installation de Tomcat

Tomcat est le conteneur de servlet open source java probablement le plus utilisé. Il peut être téléchargé dans sa version 6 à cette adresse : <http://tomcat.apache.org/download-60.cgi>

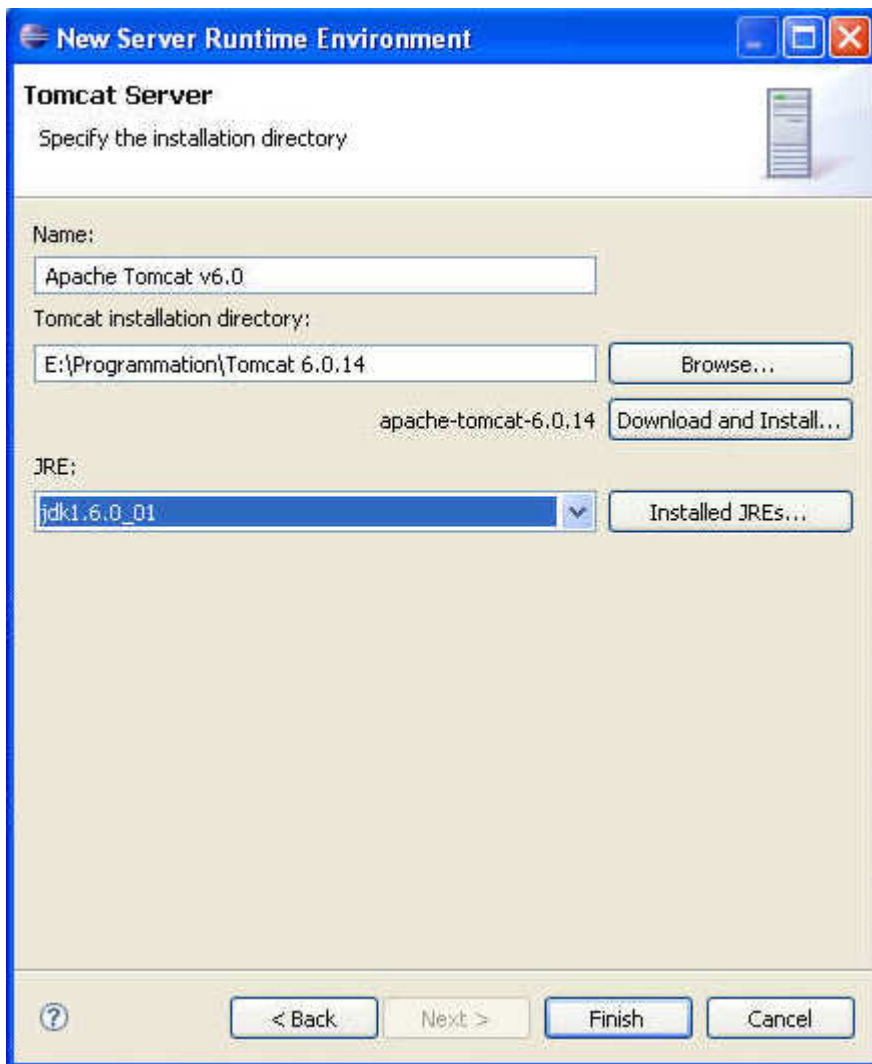
Il s'agit d'un fichier compressé que l'on décompressera dans le dossier de notre choix : ici ce sera : `E:/Programmation`. Pour l'utiliser, on va faire un clic droit sur le dossier racine de notre projet dans l'explorateur de projet, puis cliquer sur `Run as`, puis sur `Run on server` comme suit :



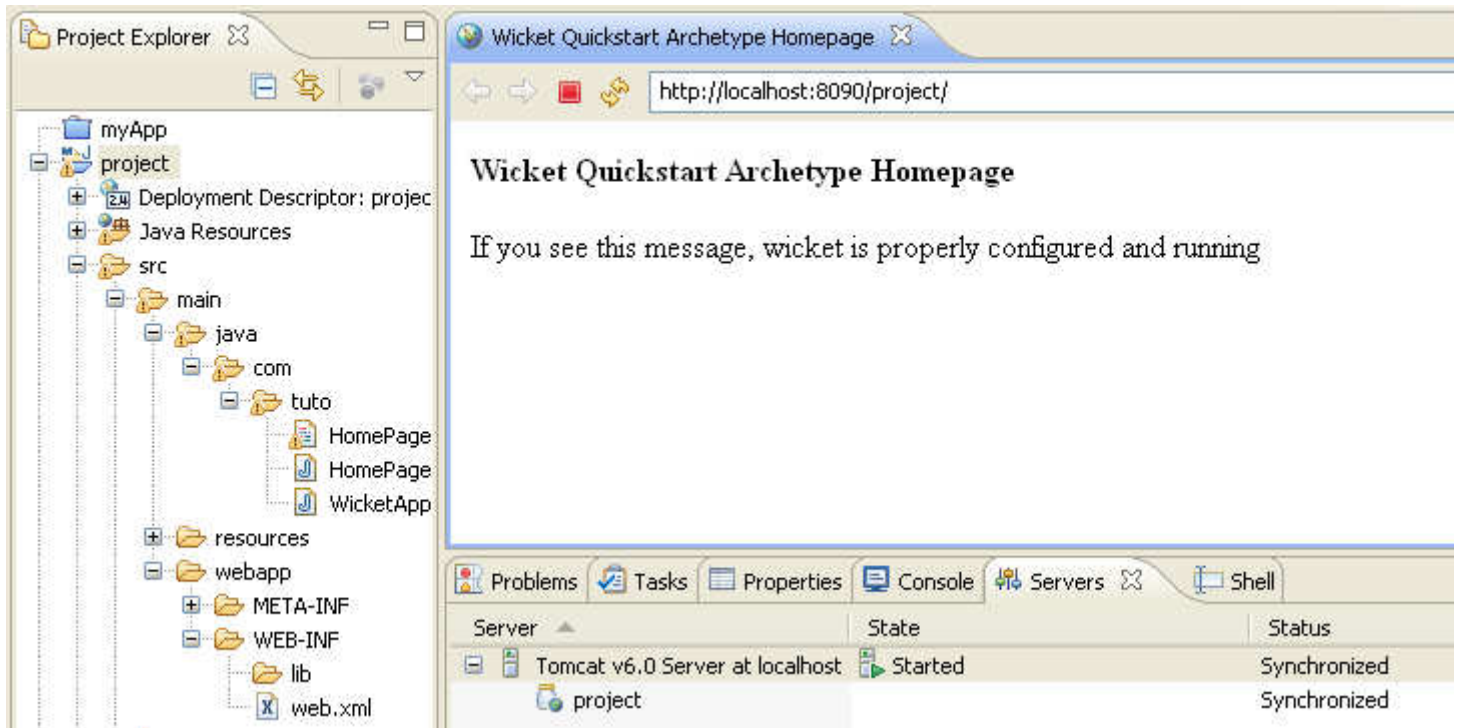
On se retrouve alors sur cette fenêtre, où l'on choisira *Tomcat v6.0 Server*.



A la fenêtre suivante, on renseigne le chemin de Tomcat et le jdk ou jre utilisé :



Tomcat va alors charger notre projet et l'afficher dans le navigateur interne d'Eclipse sur le port 8090 de localhost. La fenêtre *Servers* d'Eclipse située ici sous le navigateur d'Eclipse permet de démarrer et d'arrêter Tomcat à volonté.



Après ce préambule qui nous a permis d'installer et de faire fonctionner le projet Wicket du Quickstart, nous allons maintenant nous intéresser à son contenu et aux principes de Wicket.

IV - Principes de Wicket

IV-A - Le fichier de configuration

Comme nous l'avons vu plus haut, on trouve pour toute application web un fichier `web.xml` dans le dossier `web-inf`: c'est le descripteur de déploiement de l'application, et il contient les informations de configuration de celle-ci dans le conteneur de servlet. Sans entrer dans les détails, ce fichier informe ici le conteneur de servlet qu'il faut utiliser un filtre wicket (la classe `WicketFilter`), et définit la classe d'initialisation de l'application : ici elle se nomme `WicketApplication`.

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4">

  <display-name>project</display-name>
  <filter>
    <filter-name>wicket.project</filter-name>
    <filter-class>org.apache.wicket.protocol.http.WicketFilter</filter-class>
    <init-param>
      <param-name>applicationClassName</param-name>
      <param-value>com.tuto.WicketApplication</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>wicket.project</filter-name>
    <url-pattern>*</url-pattern>
  </filter-mapping>

</web-app>
```

IV-B - La classe d'initialisation

Il s'agit de la classe `WicketApplication` comme nous l'a appris le fichier `web.xml`. Cette classe étend la classe `WebApplication`, et sa méthode `getHomePage()` retourne la classe servant à afficher la page d'accueil de notre application nommée ici : `HomePage`. La redéfinition des nombreuses méthodes de la classe `WebApplication` permet aussi de configurer l'application.

IV-C - Une page web avec wicket

Toute page wicket est constituée de deux éléments :

- une classe qui doit hériter de la classe `WebPage` du paquet wicket (ou d' une classe dérivée elle même de `WebPage`),
- un fichier html portant le même nom que la classe. Ce fichier pourra utiliser des balises et attributs wicket servant à identifier les composants java (<http://cwiki.apache.org/WICKET/wickets-xhtml-tags.html>).

Ces deux éléments doivent être situés dans le même dossier. (Ce dernier point peut être modifié de différentes façons comme on peut le voir ici : <http://cwiki.apache.org/WICKET/control-where-html-files-are-loaded-from.html>).

On retrouve donc dans nos sources les deux fichiers : `HomePage.java` et `HomePage.html`.

La page `html` est classique, hormis la ligne suivante :

extrait de `HomePage.html`

```
<span wicket:id="message">message will be here</span>
```

On constate deux choses :

- dans la balise `span`, il y a un attribut `wicket:id` intitulé `message`
- et à l'affichage, ce n'est pas la phrase `message will be here` qui apparaît, mais `If you see this message, wicket is properly configured and running`.

Si l'on regarde maintenant notre fichier `HomePage.java` on retrouve ces deux éléments comme paramètres de la classe `Label`. Dans les imports on constate que cette classe provient du paquet wicket.

extrait de `HomePage.java`

```
add(new Label("message", "If you see this message, wicket is properly configured and running"));
```

On en déduit donc qu'à la clé `message`, est associée la valeur `If you see this message, wicket is properly configured and running`, et que c'est cette valeur qui sera utilisée à la place de celle contenue dans la balise `span`. De cette manière, on peut utiliser un attribut `wicket:id` pour chaque composant que l'on veut rajouter sur nos pages.

On a là les principes essentiels sur lequel le framework est construit : l'utilisation du namespace wicket dans les balises du code html, avec dans la classe java correspondant au fichier html du même nom, les composants java permettant l'affichage et la manipulation des données de l'application.

V - Personnalisation de notre application

On a vu comment installer et déployer une application Wicket, et comment s'articulent les fichiers composant notre application. On va maintenant personnaliser celle-ci, pour pénétrer un peu plus dans le mécanisme de fonctionnement de Wicket. Pour ce faire on va rajouter, comme dans toute application web classique, un peu de css pour habiller l'application.

V-A - Création d'un gabarit

On va donner à nos pages une structure classique en ajoutant dans le `body` de la page `html` :

- une en-tête (`id="en tete"`) comprenant un bandeau avec un logo (`id="bandeau"`) et un menu horizontal (`id="menu"`),
- un corps (`id="corps"`)
- et un pied de page avec un autre logo (`id="pied"`).
- Pour coller le pied de page en bas de la page, on va rajouter un `id="wrapper"` autour de l'en-tête et du corps.

On rajoute un lien vers le fichier `css` dans la balise `Head`, et on modifie le titre.

Pour utiliser les fichiers `css` et les images, il faut les télécharger à l'aide des liens ci-dessous, et les copier dans le dossier `webapp`

- [base.css](#)
- [logo.png](#)
- [logoFoot.gif](#)

On place notre `label` dans le `<div id="corps">` et on modifie le message associé dans le code de `HomePage.java`. (En mettant par exemple : `Bienvenue sur notre page d'accueil`).

On obtient alors ce code que l'on va utiliser pour le fichier `HomePage.html`

HomePage.html

```
<html xmlns:wicket="http://wicket.apache.org/">
  <head>
    <title>Accueil</title>
    <link rel="stylesheet" type="text/css" href="base.css" />
  </head>
  <body>
    <div id="wrapper">
      <div id="en tete">
        <div id="bandeau">
          
        </div>
        <div id="menu">
        </div>
      </div>
      <div id="corps">
        <span wicket:id="message"></span>
      </div>
    </div>
    <div id="pied">
      
    </div>
  </body>
</html>
```

Ce qui affichera une page comme celle-ci:

WICKET WORLD

Bienvenue sur notre page d'accueil



V-B - Personnalisation de l'accueil

Notre application est maintenant plus avenante, mais elle reste impersonnelle. Nous allons donc demander à l'utilisateur son prénom afin de pouvoir délivrer un message personnalisé sur notre page d'accueil.

Pour cela, nous allons dans une nouvelle page *html* que l'on va nommer [Login.html](#), utiliser un formulaire composé d'une classique balise *Form*, avec une question, une zone de texte et un bouton, auxquels on va rajouter des attributs *wicket:id*.

Login.html

```
<html xmlns:wicket="http://wicket.apache.org/">
<head>
</head>
<body>
<form wicket:id="loginForm">
<h3>Quel est votre prénom ? </h3>
<input type="text" wicket:id="login">
<input type="submit" value="OK">
</form>
</body>
</html>
```

A toute page *html* doit correspondre comme nous l'avons vu un fichier *java* du même nom. Ce fichier [Login.java](#) va contenir le traitement de l'information.

Logiquement avec wicket, à la balise *form* correspond une classe *Form*. Cette classe comprend une méthode [onSubmit\(\)](#) dans laquelle on va traiter la réponse à l'appui du bouton **OK** du formulaire. La balise *Text* du formulaire sera elle représentée par un classique *TextField* que l'on ajoutera à notre instance de *Form*. L'instanciation de ce *TextField* comporte deux paramètres :

- La chaîne correspondant à l'attribut *wicket:id* que l'on a utilisé au niveau de la balise *Text*;
- Et une instance de la classe *Model*. Il est prématuré d'approfondir à ce niveau cette notion de modèle qui justifie à elle seule un tutoriel. Cela a été fait il y a peu de temps par **Jawher Moussa** : [Exploration des modèles de Wicket](#). Je me contenterai donc ici de vous orienter vers son article pour comprendre et

approfondir cette notion qui est fondamentale dans l'utilisation de wicket, après avoir intégré les principes de base de wicket.

Login.java

```
package com.tuto;

import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.form.TextField;
import org.apache.wicket.model.Model;

public class Login {

    public Login() {
        super();
        final TextField loginText = new TextField("login", new Model());
        Form form = new Form("loginForm") {
            @Override
            protected void onSubmit() {
                System.out.println(loginText.getValue());
            }
        };
        form.add(loginText);
        add(form);
    }
}
```

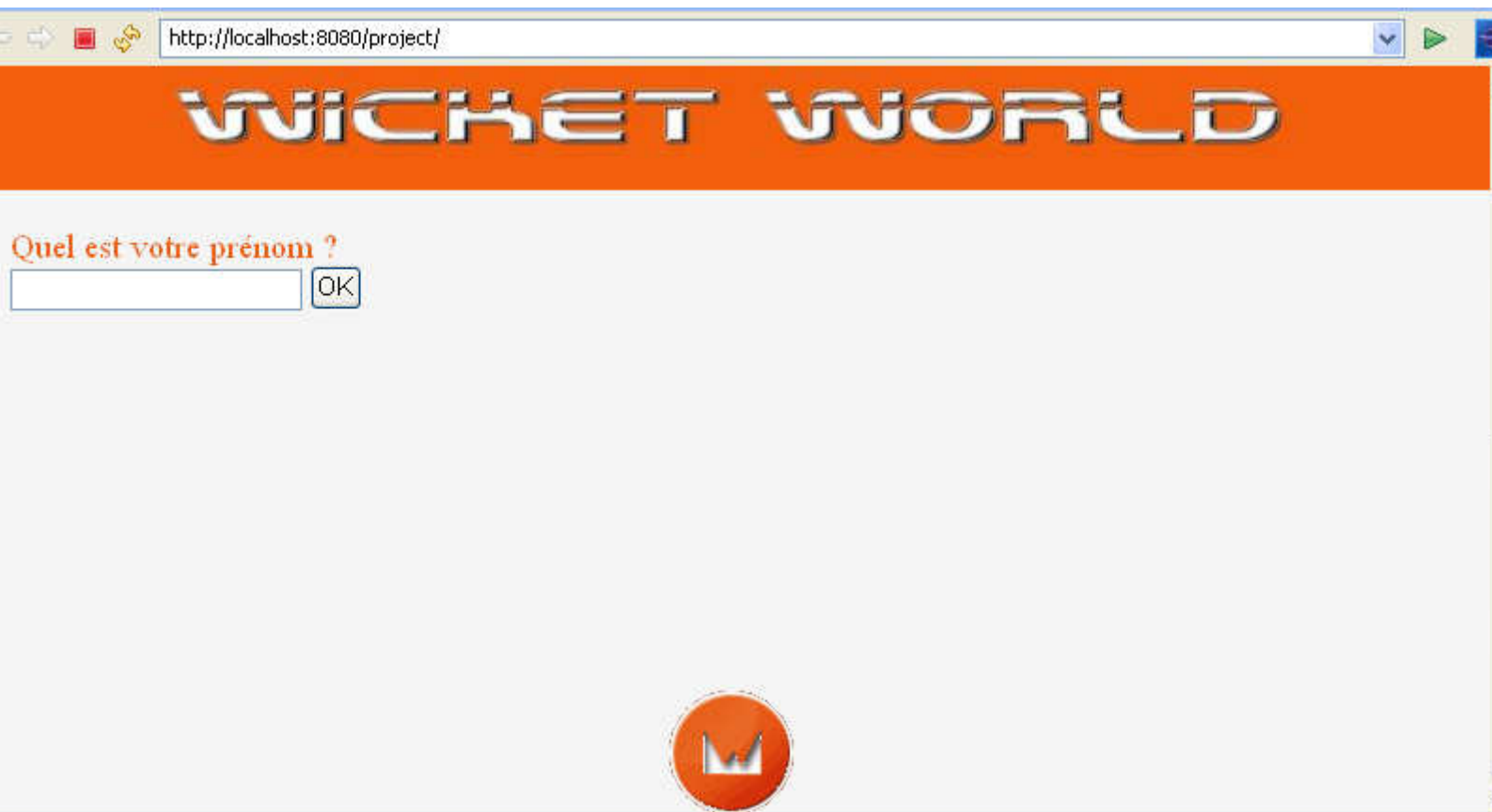
Pour l'instant, l'appui sur le bouton OK se contente d'écrire sur la console le prénom tapé dans la zone de texte et récupéré par la méthode `getValue()` du `TextField`. Cette nouvelle page `html` doit par ailleurs devenir notre page d'accès à l'application. Pour cela, il nous faut modifier la classe retournée par la méthode `getHomePage()` du fichier `WicketApplication.java`, en remplaçant `HomePage.class` par `Login.class`

Pour donner l'identité visuelle de notre application à cette page, il nous suffit d'utiliser le code de `HomePage.html` et d'inclure le formulaire dans le `<div id="corps">`

Login.html

```
<html xmlns:wicket="http://wicket.apache.org/">
  <head>
    <link rel="stylesheet" type="text/css" href="base.css" />
  </head>
  <body>
    <div id="wrapper">
      <div id="en_tete">
        <div id="bandeau">
          
        </div>
        <div id="menu">
        </div>
      </div>
      <div id="corps">
        <form wicket:id="loginForm">
          <h3>Quel est votre prénom ? </h3>
          <input type="text" wicket:id="login">
          <input type="submit" value="OK">
        </form>
      </div>
    </div>
    <div id="pied">
      
    </div>
  </body>
</html>
```

Notre application aura alors l'aspect suivant dans le navigateur interne d'Eclipse :



On se rend compte qu'il y a là une redondance de code. Wicket pour simplifier cela utilise une astuce nommée *Markup Inheritance*, qui met en jeu des mécanismes d'héritage.

V-C - Mise en page avec le principe dénommé Markup Inheritance

Pour mettre en oeuvre ce principe, nous allons copier le code *html* commun aux deux pages dans une nouvelle page *html* que nous appellerons *BasePage.html*. Le *Markup Inheritance* utilise une balise `<wicket:child/>` dans le corps de cette *BasePage.html* en remplacement du code propre à chaque page. Celui-ci sera alors développé dans des pages "filles" ou il sera inclu entre des balises `<wicket:extend>` et `</wicket:extend>`. La classe *BasePage.java* correspondante sera elle vierge, puisque la page *html* ne contient pas de composants.

BasePage.java

```
package com.tuto;

import org.apache.wicket.markup.html.WebPage;

public class BasePage extends WebPage{

    public BasePage() {
    }
}
```

BasePage.html

```
<html xmlns:wicket="http://wicket.apache.org/">
  <head>
    <link rel="stylesheet" type="text/css" href="base.css" />
  </head>
  <body>
    <div id="wrapper">
      <div id="en_tete">
        <div id="bandeau">
```

BasePage.html

```

        
    </div>
    <div id="menu">
    </div>
</div>
<div id="corps">
    <wicket:child/>
</div>
</div>
<div id="pied">
    
</div>
</body>
</html>

```

Il reste ensuite :

- à dériver de la classe `BasePage.java` les classes correspondant aux pages de notre application (`public class HomePage extends BasePage` et `public class Login extends BasePage`),
- et à inclure dans les pages "filles", le code *html* propre à chaque page entre les balises `<wicket:extend>` et `</wicket:extend>`. Ces pages pourront d'ailleurs être prévisualisées directement dans un browser sans qu'il soit nécessaire d'invoquer la page parente.

HomePage.html

```

<html xmlns:wicket="http://wicket.apache.org/">
<body>
    <wicket:extend>
        <span wicket:id="message"></span>
    </wicket:extend>
</body>
</html>

```

Login.html

```

<html xmlns:wicket="http://wicket.apache.org/">
<body>
    <wicket:extend>
        <form wicket:id="loginForm">
            <h3>Quel est votre prénom ? </h3>
            <input type="text" wicket:id="login">
            <input type="submit" value="OK">
        </form>
    </wicket:extend>
</body>
</html>

```

```
apache.org/">  
  
ext/css" href="base.css" />
```

HomePage.html

```
<html xmlns:wicket="http://www.apache.org/licenses/LICENSE-2.0">  
  <body>  
    <wicket:ext class="HomePage" />  
    <span wicket:id="title" />  
  </wicket:ext>  
  </body>  
</html>
```

Login.html

```
<html xmlns:wicket="http://www.apache.org/licenses/LICENSE-2.0">  
  <body>  
    <wicket:ext class="Login" />  
    <form wicket:id="loginForm">  
      <h3>Que pensez-vous de ce site ?</h3>  
      <input type="text" />  
      <input type="submit" value="Envoyer" />  
    </form>  
    <wicket:ext class="Footer" />  
  </body>  
</html>
```

V-D - Titre des pages

On retrouve l'utilisation du même principe au niveau de la partie *head* des pages, avec des balises `<wicket:head>` et `</wicket:head>`. On peut s'en servir pour ajouter un titre à chaque fichier.

extrait de BasePage.html

```
<html xmlns:wicket="http://wicket.apache.org/">
  <head>
    <wicket:head>
      <link rel="stylesheet" type="text/css" href="base.css" />
    </wicket:head>
  </head>
  <body>
    ...
  </body>
</html>
```

extrait de HomePage.html

```
<html xmlns:wicket="http://wicket.apache.org/">
  <wicket:head>
    <title>Accueil</title>
  </wicket:head>
  <body>
    ...
  </body>
</html>
```

extrait de Login.html

```
<html xmlns:wicket="http://wicket.apache.org/">
  <wicket:head>
    <title>Identification</title>
  </wicket:head>
  <body>
    ...
  </body>
</html>
```

V-E - Utilisation d'un FeedbackPanel

On peut améliorer notre formulaire en utilisant la méthode `setRequired(true)` du `TextField`, qui empêche la validation d'une entrée nulle au niveau du formulaire. On peut aussi ajouter à notre page un composant `FeedbackPanel`, qui comme son nom le laisse deviner va nous permettre d'afficher divers retours provenant des composants de la page. Plusieurs méthodes de la classe `Component` peuvent être utilisées à cet effet en fonction du type de message qui doit être retourné : `info(..)` `debug(..)` `error(..)`. Sa mise en oeuvre est toute simple: il suffit d'ajouter dans les deux fichiers sources les lignes suivantes :

extrait de Login.html

```
<html xmlns:wicket="http://wicket.apache.org/">
  ...
  <wicket:extend>
    <span wicket:id="feedback"></span>
    <!--form ...>
      < code du Form>
    </form-->
    <br/>
  </wicket:extend>
</html>
```

extrait de Login.java

```
package com.tuto;

// imports ...

public class Login extends BasePage{

    public Login() {
        ...
        add(form);
        loginText.setRequired(true);
        add(new FeedbackPanel("feedback"));
    }
}
```

Il suffit de valider le formulaire avec une zone de texte vide pour tester l'usage du `FeedbackPanel`.

WICKET WORLD

- Le champ 'login' est obligatoire.

Quel est votre prénom ?

De la même façon, à titre d'exemple, on peut ajouter un `feedbackpanel` à notre page d'accueil, avec un contenu d'information.

extrait de HomePage.html

```
...
    <wicket:extend>
        <h2 wicket:id="message"></h2><br/>
        <div id="feedbackPanel">
            <span wicket:id="feedback"></span>
        </div>
    </wicket:extend>
...
```

extrait de HomePage.java

```
...
public class HomePage extends BasePage {
```

extrait de HomePage.java

```
public HomePage () {
    super();
    add(new Label("message", "Bienvenue sur notre page d'accueil "));
    add(new FeedbackPanel("feedback"));
    String date = DateFormat.getDateInstance().format(new Date());
    String time = DateFormat.getTimeInstance(DateFormat.SHORT).format(new Date());
    info("La page a été ouverte le " + date + " à " + time);
}
}
```

WICKET WORLD

Bienvenue sur notre page d'accueil

- La page a été ouverte le 8 oct. 2008 à 00:25



Nous avons maintenant deux pages web, qui sont pour l'instant indépendantes l'une de l'autre. Le chapitre suivant va développer la notion de session qui permet d'englober ces deux pages dans une entité, et la navigation qui permet de passer de l'une à l'autre.

VI - Session

Le protocole HTTP est un protocole dit sans état, c'est à dire qu'il traite chaque requête indépendamment les unes des autres. Or, s'adressant potentiellement à plusieurs utilisateurs, une web application va nécessiter un contexte propre à chacun englobant la succession des pages renvoyées à ces utilisateurs, et l'interactivité générée par ces pages. On appelle ce contexte Session. C'est au conteneur de servlet que revient la gestion de ces Sessions. Il est nécessaire dans la programmation d'une web application de pouvoir accéder à la session. Nous allons montrer ici comment y parvenir avec wicket en utilisant le prénom que l'on vient de demander dans la page d'accueil comme un champ de la la session, ce qui nous permet alors de l'utiliser dans toutes nos pages. Il va donc falloir créer une classe User pour pouvoir utiliser cette donnée.

VI-A - L'utilisateur

Notre classe **User** va tout simplement comporter un champ **String** avec ses accesseurs, et un constructeur surchargé.

User.java

```
package com.tuto;

public class User{

    private String name;

    public User(){
    }

    public User(String s){
        this.name=s;
    }

    public String getName() {
        return name;
    }

    public void setName(final String name) {
        this.name = name;
    }

}
```

VI-B - La session

Pour pouvoir utiliser notre `User` comme champ de la session, il va falloir redéfinir dans notre classe `WicketApplication`, la méthode `newSession(Request, Response)` de la classe `Application` (la classe parente de `WebApplication`). Cette méthode retourne un Objet de la classe abstraite `Session`, parente de `WebSession`. On va donc créer une classe que l'on va nommer `UserSession`, qui va dériver de `WebSession`, dans laquelle on va utiliser un champ `User` avec ses accesseurs. Notre utilisateur appartiendra alors à la session et non à une `WebPage`.

WicketApplication.java

```
package com.tuto;

import org.apache.wicket.Request;
import org.apache.wicket.Response;
import org.apache.wicket.Session;
import org.apache.wicket.protocol.http.WebApplication;
import org.apache.wicket.protocol.http.WebSession;

public class WicketApplication extends WebApplication{

    public WicketApplication(){
    }

    @Override
    public Session newSession(Request request, Response response){
        return new UserSession(request);
    }

    @Override
    public Class getHomePage(){
        return Login.class;
    }

}
```

UserSession.java

```
package com.tuto;

import org.apache.wicket.Request;
```

UserSession.java

```
import org.apache.wicket.protocol.http.WebSession;

public final class UserSession extends WebSession{

    private User user;

    protected UserSession(Request request) {
        super(request);
    }

    public User getUser(){
        return user;
    }

    public void setUser(final User user){
        this.user = user;
    }
}
```

VI-C - L'accès à la session

Pour accéder à la session, il faut utiliser la méthode `getSession()` de la classes `Component` du package `Wicket`. Cette méthode retourne une instance de la classe `WebSession`; il va donc falloir faire un cast pour obtenir une instance de notre classe `UserSession`. Pour simplifier le code, nous allons créer une méthode `getMySession()` au niveau de la classe `BasePage` pour faire ce travail. Cette méthode sera ensuite accessible à partir des pages dérivées de `BasePage`.

extrait de BasePage.java

```
...
public UserSession getMySession(){
    return (UserSession)getSession();
}
...
```

De cette façon, on va pouvoir personnaliser avec le prénom de l'utilisateur le message d'accueil au niveau de notre `HomePage` :

extrait de HomePage.java

```
...
public HomePage() {
    super();
    User nom = getMySession().getUser();
    add(new Label("message", "Bienvenue sur notre page d'accueil "+nom.getName()));
}
...
```

Chaque session comporte un identifiant unique généré par la servlet que l'on peut visualiser très simplement à l'aide de la méthode `getId()` de notre instance d'`UserSession`. Pour compléter notre application, on va l'ajouter au `FeedbackPanel` de `HomePage` avec l'instruction : `info("Vodre id de session = "+ getMySession().getId());`.

```
package com.tuto;

import java.text.DateFormat;
import java.util.Date;
import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.panel.FeedbackPanel;

public class HomePage extends BasePage {

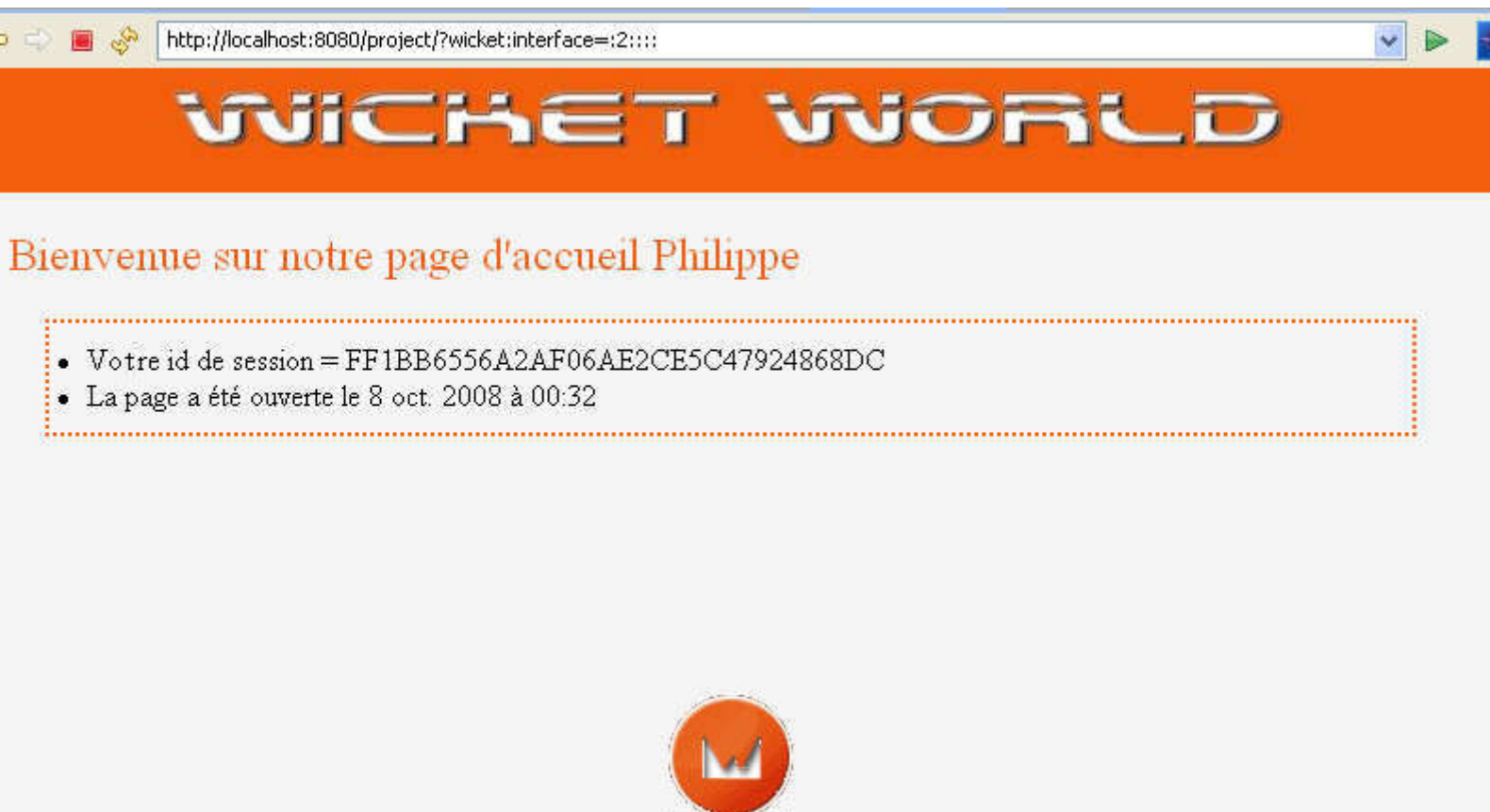
    public HomePage() {
```

```

super();
User nom = getSession().getUser();
add(new Label("message", "Bienvenue sur notre page d'accueil "+nom.getName()));
add(new FeedbackPanel("feedback"));
info("Votre id de session = " + getSession().getId());
String date = DateFormat.getDateInstance().format(new Date());
String time = DateFormat.getTimeInstance(DateFormat.SHORT).format(
    new Date());
info("La page a été ouverte le " + date + " à " + time);
}
}

```

Ce qui nous donnera :



VI-D - La navigation

Il va maintenant falloir créer un lien entre nos deux pages. Une fois identifié, l'utilisateur devra être dirigé vers la page d'accueil. On utilisera alors tout naturellement dans la méthode `onSubmit()` du fichier `Login.java` la méthode `setResponsePage(new HomePage())` de la classe `Component` pour créer une instance de la page d'accueil.

Login.java

```

package com.tuto;

import org.apache.wicket.markup.html.form.Form;
import org.apache.wicket.markup.html.form.TextField;
import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.apache.wicket.model.Model;

public class Login extends BasePage{

    public Login() {
        super();
    }
}

```

Login.java

```
final TextField loginText = new TextField("login", new Model());
loginText.setRequired(true);
Form form = new Form("loginForm") {
    @Override
    protected void onSubmit() {
        getSession().setUser(new User(loginText.getValue()));
        setResponsePage(new HomePage());
    }
};
form.add(loginText);
add(form);
add(new FeedbackPanel("feedback"));
}
```

Maintenant notre application peut diffuser un message d'accueil avec le prénom qu'on lui aura indiqué. Une fois sur la page d'accueil, la barre d'adresse contient l'adresse suivante : <http://localhost:8080/project/?wicket:interface=2:::> Copiez la dans le presse papier et éteignez le serveur. Relancez le serveur et collez l'adresse dans la barre d'adresse; vous aurez alors le message :

Page Expired

The page you requested has expired.

[Return to home page](#)

Il s'agit d'une page dite *stateful page* c'est à dire liée à la session, qui ne pourra pas être ajoutée aux favoris, puisque logiquement, elle ne devra être accessible qu'après s'être identifié. On l'oppose aux pages *stateless pages* c'est à dire indépendantes de la session qui pourront être marquées dans les favoris et appelées directement à partir de ce lien.

Pour illustrer ces notions, nous allons créer une page *stateless*. Nous allons l'appeler `InfoPage.html`, et lui faire afficher des informations concernant le navigateur de l'utilisateur pour lui donner un contenu.

infoPage.html

```
<html xmlns:wicket="http://wicket.apache.org/">
<wicket:head>
<title>Informations</title>
</wicket:head>
<body>
<wicket:extend>
<h2 wicket:id="infoPage"></h2>
<div id="feedbackPanel">
<span wicket:id="feedback"></span>
</div>
</wicket:extend>
</body>
</html>
```

InfoPage.java

```
package com.tuto;

import org.apache.wicket.markup.html.basic.Label;
import org.apache.wicket.markup.html.panel.FeedbackPanel;
import org.apache.wicket.protocol.http.ClientProperties;
import org.apache.wicket.protocol.http.request.WebClientInfo;

public class InfoPage extends BasePage {

    public InfoPage() {
```

InfoPage.java

```

super();
add(new Label("infoPage", "Page d'information"));
add(new FeedbackPanel("feedback"));
getApplication().getRequestCycleSettings().setGatherExtendedBrowserInfo(true);
WebClientInfo w = (WebClientInfo) getWebRequestCycle().getClientInfo();
ClientProperties cp = w.getProperties();
info("Votre navigateur : "+cp.getNavigatorUserAgent());
info("Votre adresse IP : "+cp.getRemoteAddress());
if (cp.isJavaEnabled()){
    info("Java est activé");
}else{
    info("java est désactivé");
}
if(cp.isCookiesEnabled()){
    info("Les cookies sont acceptés");
}else{
    info("Les cookies ne sont pas acceptés");
}
}
    
```

Pour voir cette nouvelle page, il va falloir utiliser un mécanisme de navigation que l'on va baser sur des liens .

VI-E - Les liens

On va placer ces liens dans un menu dont on a déjà déterminé la place lors de la création du gabarit de la page vierge : c'est à dire dans le `<div id="menu">` de notre `BasePage.html`, puisque les liens devront pouvoir être utilisés à partir de chaque page.

On va disposer les liens dans une liste, que l'on affichera selon une disposition horizontale grâce au fichier css. Ils seront affichés dans une balise `html` classique ``, avec à nouveau un attribut `wicket:id` qui permet de référencer le lien dans le fichier `java` correspondant.

BasePage.html

```

<html xmlns:wicket="http://wicket.apache.org/">
<head>
<link rel="stylesheet" type="text/css" href="base.css" />
</head>
<body>
<div id="wrapper">
<div id="en_tete">
<div id="bandeau">
<a href="http://wicket.apache.org/"></a>
</div>
<div id="menu">
<ul>
<li><a href="#" wicket:id="homePage">Accueil</a></li>
<li><a href="#" wicket:id="infoPage">Page 1</a></li>
</ul>
</div>
</div>
</div>
<div id="corps">
<wicket:child/>
</div>
</div>
<div id="pied">

</div>
</body>
</html>
    
```

Avec wicket, les liens utilisent la classe `Link` et les classes dérivées. Ces classes ont une méthode `onClick()` dans laquelle on peut fournir un traitement et utiliser des méthodes `setResponsePage(...)` pour indiquer la page à afficher.

Dans notre fichier `BasePage.java` :

- on utilisera pour le lien vers la page d'accueil, la classe `Link`. Dans la méthode `onClick()` on utilisera une méthode de la forme `setResponsePage(Page p)`. La page créée ne pourra pas être utilisée dans les favoris: c'est une *stateful* page (page maintenue en mémoire côté serveur); le lien sera maintenu le temps de la session, et aura la forme : `http://localhost:8080/project/?wicket:interface=:X:::`, X étant incrémenté à chaque changement de page
- on utilisera pour le lien vers la page d'information, la classe `BookmarkablePageLink` dérivée de la classe `Link`. Cette classe crée une page sans état (*stateless* page); la page pourra alors être utilisée dans les favoris. L'adresse de la page utilise les éléments : `wicket:bookmarkablePage=:` suivi du chemin de la classe dans le paquet. Ici on aura donc : `http://localhost:8080/project/?wicket:bookmarkablePage=:com.tuto.InfoPage`
Les méthodes `setResponsePage(Class c)` et `setResponsePage(Class c, PageParameters pp)` créent également des pages de ce type.

BasePage.java

```
package com.tuto;

import org.apache.wicket.PageParameters;
import org.apache.wicket.markup.html.WebPage;
import org.apache.wicket.markup.html.link.Link;

public class BasePage extends WebPage{

    public BasePage(final PageParameters pageParameter) {
        add(new Link("homePage") {
            public void onClick() {
                if(getMySession().getUser() != null) {
                    setResponsePage(new HomePage());
                }else{
                    setResponsePage(new Login());
                    info("Vous devez vous identifier");
                }
            }
        });
        add(new BookmarkablePageLink("infoPage", InfoPage.class));
    }

    public UserSession getMySession() {
        return (UserSession)getSession();
    }
}
```

Nous avons maintenant un embryon d'application web fonctionnel, dont la page d'identification se présente ainsi :



La page d'information s'affichera elle ainsi :



VII - Conclusion

Comme nous avons pu le voir, wicket qui est un framework java d'application web, fait la part belle à ce langage. Pour en approfondir l'étude, je conseillerai à nouveau de bien s'imprégner de l'utilisation des modèles, et d'étudier les exemples d'utilisation des composants. On peut aussi signaler la très récente sortie du livre *Wicket in action* à réserver aux anglophiles.

VIII - Ressources

- Ressources concernant les applications web java :
 - Comprendre l'environnement web**
 - Le Comportement des Navigateurs Internet**
 - Le Protocole HTTP**
 - La technologie des servlets**
 - Applications web : sécuriser la session utilisateur**
 - Tomcat et applications web**
 - Configuration de Tomcat**
 - Les bases du développement web MVC en Java**
- Ressources concernant wicket :
 - Le site de wicket**
 - Hands on wicket** : une très bonne vue d'ensemble de wicket
 - Redécouvrez le web avec Wicket**
 - Mise en oeuvre d'Apache Wicket**
 - Exploration des modèles de Wicket**
 - Introduction to the Wicket Web Framework** : utiliser wicket avec netbeans
 - Wicket : the first steps**

IX - Remerciements

Je remercie **djo.mos** pour l'aide qu'il m'a apporté pour l'élaboration de cet article, et **RomainVALERI** pour les corrections orthographiques.