KurumsalJava.com
Java Enterprise Architecture

Web Framework Gökyüzünde Yeni Bir Yıldız: Wicket!

Özcan Acar
Bilgisayar Mühendisi
http://www.ozcanacar.com/

Giriş

Web tabanlı programlar oluşturmak için kullanabileceğimiz Java tabanlı birçok web framework bulunmaktadır. Bunlardan en popüler olanları:

- Struts
- JSF (Java Server Faces)
- Spring MVC
- WebWork

İsmi geçen frameworkler ile çalışma fırsatı buldum. MVC (Model View Controller) tasarım şablonu kullanılarak oluşturulan bu frameworkler (JSF harici – JSF'in komponent modeli mevcuttur), kurumsal web projeleri için gerekli tüm özeliklere sahiptirler.

Bu yazımda web frameworkleri arasında bir kıyaslama yer almayacak. Sizlere bu yazımda Wicket[1] ismini taşıyan ve Apache Software Foundation [2] bünyesinde geliştirilen bir web frameworkü tanıtmak istiyorum. Doğal olarak Struts, JSF ya da Spring MVC zaten işimizi görüyor, yeni bir web frameworke ne gerek var diyenler olacaktır. Uzun bir süredir ismi geçen web frameworkleri ile değişik projeler yapıyorum. Ama ne yazık ki düşündüğüm web programlama tarzını bu frameworkler ile oluşturmam mümkün olmadı. Nasıl bir model düşündüğümü sizlere aktarmak istiyorum. Bu aynı zamanda nasıl Wicket'i keşfettiğimin ve kullanmamın açıklamasıdır.

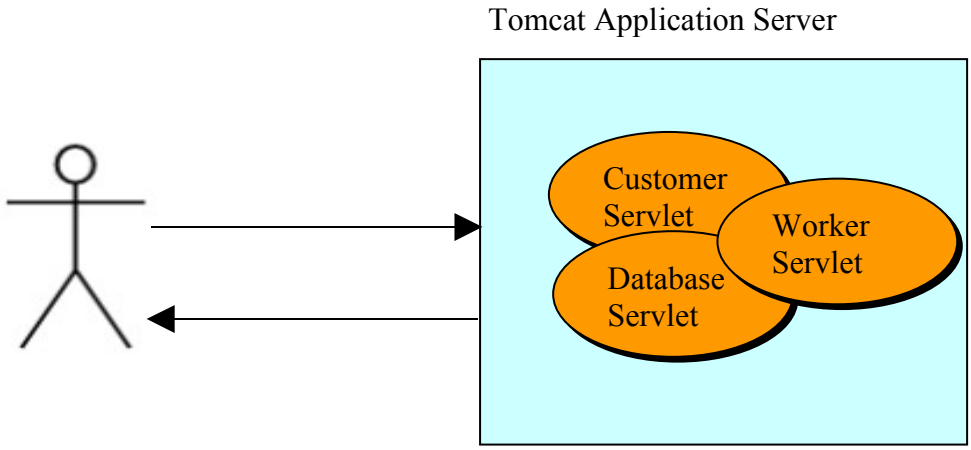
Benim web programcılığında yıllardan beri karşılaştığım bir sorun var. Genelde her web projesi için aynı Java kodu yazılır. Örneğin üyelerin login işlemini gerçekleştirmek için login modülü ya da kayıt işlemleri için kayıt modülü gereklidir. Bu modülleri komponent (başka bir yazılım sisteminde tekrar kullanılabilir yapıda olan program parçası) kılıfına sokmayı beceremediğim için her proje için aynı modülleri sil baştan programlamak zorunda kaldım. Aradığım, web programlama için bir komponent modeliydi. Sadece bir sefer login modülünü komponent olarak implemente ettikten sora, yeni bir projede bu komponenti bir Jar dosyası olarak, değişiklik yapmak zorunda kalmadan kullanabilmeliydim. Bu rüyam ne yazık ki ismi geçen frameworkler ile gerçek olmadı, çünkü bu web frameworklerinde gösterim katmanında yer alan HTML arayüzler Java kodundan ayrı olarak düşünülür ve programlanır. Gösterim katmanında yer alan HTML (JSP sayfaları) arayüzleri için özel programlama (daha sonra inceleyeceğimiz JSP ve JSTL tarzı) teknikleri geliştirilmiştir. Oluşturulan Java kodu bir web aplikasyonda WEB-INF/classes dizinindeyken JSP sayfaları WEB-INF ile aynı (root) dizin içindedir. Görüldüğü gibi gösterim katmanı ve işletim katmanını oluşturan dosyalar bile fiziksel olarak yayılma (deployment) esnasında birbirlerinden ayrı tutulmaktadırlar. Bu web için kullanılan grafikler (gif, jpeg) ve Style Sheet (css) dosyaları için de geçerlidir. Bu Java kodu ile JSP sayfalarının beraber ve tekrar kullanımını engellemektedir. Bu yüzden gerçek anlamda, adı geçen web frameworkler ile tekrar kullanılabilir komponentler oluşturmak mümkün değildir. Wicket ile bu mümkün! Nasıl olduğunu yakından inceleyeceğiz. Wicket'in sahip olduğu komponent modelini anlayabilmek için öncelikle Java dilinde yapılan web programcılığının tarihçesini incelememiz gerekiyor. Mevcut teknolojileri kıyaslama usulü ile Wicket'in var oluş nedenini anlamamız kolaylaşacaktır.

[1] Bakınız: http://wicket.apache.org/
[2] Bakınız: http://www,apache.org

Java'da Web Programcılığın Tarihçesi

Doksanlı yılların ortalarında benim web programcılığım Perl ve CGI ile tanışarak başladı. O zamanlar ne Java ne de Eclipse gibi güçlü bir yazılım aracı vardı :) Daha sonra 2000 senedinde bu konuda bir kitap [3] çalışmam oldu. Üniversite yıllarında Java ile tanıştım. Java, ilk versiyonlarında sadece nesneye yönelik (object oriented) programlama yapılan bir dildi. Java'nın kısa sürede popüler olmasının sebeplerinin birisi, Java ile yazılan programların web tarayıcısı (browser) içinde çalışabiliyor olmalarıydı. Applet olarak bilinen bu programlar ile zengin arayüzler (GUI – graphical user interface) oluşturmak mümkün. Ne yazık ki ilk Java sürümleri ile hazırlanan Appletler çok yavaş olduklarından günümüze dek süregelen ve Java'nın yavaş bir platform olduğu kanaati oluştu. Java'yı bilmeyenler için Appletlerin yavaş olmaları, Java'nın yavaş olduğu anlamına geliyor. Bu kanaati değiştirmek artık mümkün değil sanırım. Ama Java ile neler yapılabileceğini bilen biliyor :)

Java'nın ihtiva ettiği, Appletler haricinde web tabanlı programlar oluşturulabilmek için kullanılan ilk teknoloji Servlet[4] teknolojisidir. Servlet teknolojisi ile kelime hazinemize aplikasyon server (application server – örneğin Tomcat) terimi yerleşti. Servlet olarak hazırlanmış bir program aplikasyon serveri olarak bilinen bir çalışma ortamına ihtiyaç duymaktadır. Servlet tabanlı bir programın aplikasyon serverinde çalışır hale getirilmesi işlemine deployment (yayılma) adı verilmektir.



Resim 1 Servlet sınıfları application server içinde çalışır.

Bir Java sınıfı ***javax.servlet.http.HttpServlet*** sınıfını genişleterek bir Servlet haline gelir. Bir sonraki Servlet örneğinde **service()** metodu işlem görmekte ve ekranda Merhaba Dünya yazısı yer almaktadır.

[3] Bakınız: http://www.pusula.com/2/index.php?option=com_pusula&func=detail&Itemid=34&id=69
[4] Bakınız: http://java.sun.com/products/servlet/

```java
package com.kurumsaljava.com.sample.servlet;

import java.io.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet extends HttpServlet
{
	private static final long serialVersionUID = 1L;

	public void init(ServletConfig config) throws ServletException
	{
		super.init(config);
	}

	public void destroy()
	{
		super.destroy();
	}


	public void service(HttpServletRequest request,
			HttpServletResponse response)
			throws ServletException, IOException {

		response.setContentType("text/html");
		ServletOutputStream output = response.getOutputStream();
		output.print("<h1>Merhaba Dünya");
	}

	public void doGet(HttpServletRequest request,
			HttpServletResponse response)
			throws ServletException, IOException {

		service(request, response);
	}

	public void doPost(HttpServletRequest request,
			HttpServletResponse response)
			throws ServletException, IOException {

		service(request, response);
	}
}
```

Ne yazık ki Servlet teknolojisi ile geniş kapsamlı web programları oluşturmak mümkün değil, çünkü gösterim katmanı ile işlem yapılan katman iç içe, yani gösterim katmanı için gerekli HTML kodunu Servlet sınıfı bünyesinde oluşturmamız gerekiyor. Bu kodun bakımını ve geliştirilmesini zorlaştıran bir durumdur. Bu durum, gösterim katmanı için yeni bir Java teknolojisinin oluşturulmasını gerekli kıldı. Servlet teknolojisinden sonra JSP (Java Server Pages) [5] olarak tanıdığımız, HTML ve Java kodunun beraber kullanılabildiği bir web teknolojisi oluşturuldu.

JSP tabanlı bir web programında HTML arayüzler JSP sayfalarında tutulur. Bunun bir örneği bir sonraki kod da yer almaktadır.

[5] Bakınız: http://java.sun.com/products/jsp/

```
<html>
<body>
<%
   for(int i=0; i<5; i++)
   {
        out.print("<h1>Merhaba Dünya</h2>);
   }
%>
</body>
</html>
```

JSP sayfalarında HTML kodu ile Java kodunun beraber kullanılabilmesi özelliği, JSP teknolojisi ile HTML arayüzlerinin daha kolay tasarlanabilmesini sağladı. Ne yazık ki kısa bir zaman sonra JSP teknolojisinin de yeterli olmadığı anlaşıldı, çünkü JSP sayfalarında Java kodunun gömülerek kullanılması, bakımı ve geliştirilmesi zor JSP sayfalarının oluşmasını çabuklaştırdı. Nasıl bir çözüm sağlanmalıydı ki, tasarımcılar (web designer) gösterim (presentation) katmanını oluşturan HTML arayüzlerini, programcılar da işletim (business) katmanında yer alan Java kodu oluşturabilsinler ve bu çözüm iki katmanın birleştirilmesinde kullanılsın? Bu sorunun ilk cevabını JSTL (Java Standard Tag Library) [6] teknolojisi vermiştir.

Web komponent teknolojisi terimi ilk JSTL teknolojisi ile anılmaya başladı. Bu makaleyi yazmamın ana sebeplerinden birisi web tabanlı programlar için tekrar kullanılabilir (reusable) komponentlerin nasıl oluşturulabileceğini göstermektedir. Bu şimdiye kadar web tabanlı programlar için hemen hemen mümkün olmayan bir şeydi. Bunun neden böyle olduğunu ilerleyen satırlarda göreceğiz.

JSTL teknolojisi Java kodunun HTML arayüzlerinde (JSP sayfalarında) kullanımını kolaylaştırır. JSP sayfasında Java kodu yerine, tag olarak bilinen ve server tarafında bir Java sınıfında yer alan koda denk gelen birimler kullanılır. Bunun bir örneği bir sonraki JSTL ile hazırlanmış JSP sayfasında yer almaktadır.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>

  <p><h1>Müsteri Isimleri</h1></p>

  <c:forEach items="${addresses}" var="address">
    <c:choose>
      <c:when test="${not empty address.lastName}" >
        <c:out value="${address.lastName}"/><br/>
      </c:when>
      <c:otherwise>
        N/A<br/>
      </c:otherwise>
    </c:choose>
```

[6] Bakınız: http://java.sun.com/products/jsp/jstl/

```
</c:forEach>
```

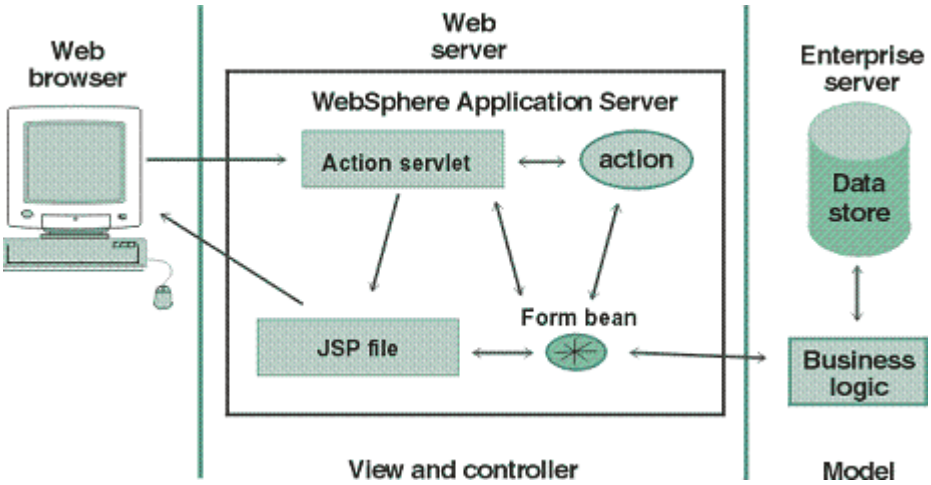
Üst bölümde yer alan JSP sayfasında bir for döngüsü kullanılarak ${addresses} listesinde yer alan müşteri adres bilgileri ekranda görüntülenmektedir. For döngüsünü oluşturmak için <forEach> JSTL tagı kullanılmaktadır. Örnekte görüldüğü gibi Java kodu kullanma zorunluluğu olmadan JSTL tagleri kullanarak dinamik HTML arayüzler oluşturmak mümkündür.

Bu örnekte gösterim katmanının (JSTL ile hazırlanan JSP sayfası) işletim katmanından (${addresses} listesinin oluşturulduğu Java sınıfları) JSTL teknolojisi kullanılarak nasıl ayrıldığını gördük. JSTL teknolojisi, web tasarımcılarının Java dilini kullanmalarına gerek kalmadan dinamik HTML arayüzler oluşturmalarını kolaylaştırır. Tasarımcılar JSP sayfalarındaki dinamik bölümler için JSTL tagler kullanırlar. Kullanılan her tag için bir Java sınıfı gerekli Java kodunu ihtiva eder. JSTL ile web programlama yapılırken standart JSTL tagler yanı sıra, Java programcıları tarafından oluşturulmuş yeni JSTL tagler kullanılabilir. Bu şekilde kullanılabilecek tag adedi artırılabilir.

Daha öncede belirttiğim gibi web tabanlı programlar için kullanılan komponentlere ilk olarak JSTL teknolojisinde rastlıyoruz. JSTL bünyesindeki her tag bir komponent olarak düşünülebilir. Tagleri JSP sayfalarında kullanmak mümkün olduğu için tagleri oluşturan Java kodunun tekrar kullanımı sağlanmaktadır ve bu yüzden JSTL taglerini komponent olarak düşünebiliriz. Bu tagler çok ufak Java birimleri olduklarından JSTL teknolojisinde gerçek anlamda web komponent teknolojisinden bahsetmek doğru olmaz.

JSTL kullanılmış olsa bile JSP teknolojisi ile geniş kapsamlı web programları oluşturmak kolay değildir. JSP sayfalarında HTML gösterimi yanı sıra navigasyon ve verilerin validasyonu (validation) gibi kompleks işlemlerin yapılması gerekmektedir. Bu işlemlerin JSP sayfalarında yapılması, JSP sayfalarının bakımını ve geliştirilmesini zora sokmaktadır. Navigasyon ve validasyon gibi işlemlerin JSP sayfalarının dışında, başka bir mekanizma kullanılarak yapılması gerekmektedir. Bu ihtiyacı karşılamak amacıyla Struts, Spring MVC ve WebWork gibi web frameworkler doğmuştur.

Bu yeni jenerasyon web frameworkler MVC (Model View Controller) tasarım şablonu ile implemente edilmiştir.



Resim 2 MVC web framework örneği

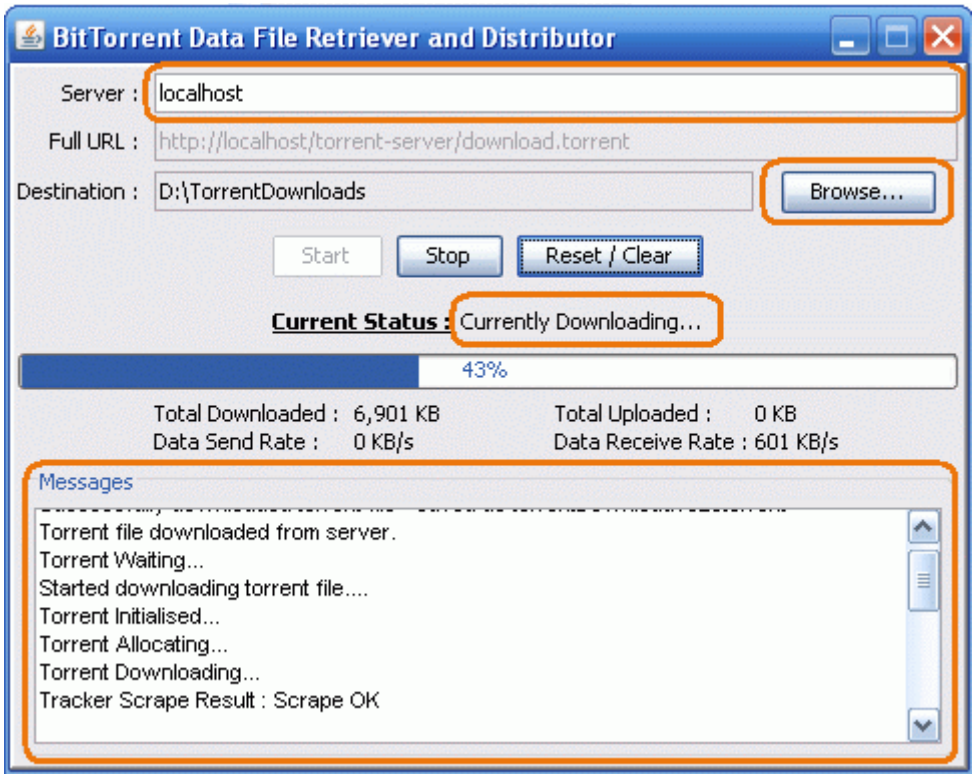
MVC tarzı çalışan web frameworklerde kullanıcı istekleri merkezi bir Controller (Action Servlet) sınıfı tarafından karşılanır. Controller sınıfı validasyon ve navigasyon gibi işlemlerden sorumludur. JSP sayfalarında gösterilmesi gereken veriler Model sınıflarında tutulur. Controller, verileri ihtiva eden model nesnelerini HttpServletRequet, HttpServletResponse ya da HttpSession (Java Servlet API sınıfları) sınıfları aracılığıyla JSP sayfaları tarafından kullanılır hale getirir. Modellerin ihtiva ettiği veriler View olarak isimlendirilen JSP sayfalarında gösterilir. JSP sayfaları HttpServletRequest, HttpServletResponse ya da HttpSession aracılığıyla gerekli model nesnelerine ulaşır. Çoğu web framework JSTL taglerini kullanarak, edinilen verilerin JSP sayfalarında gösterimini gerçekleştirir. JSP sayfalarının model nesnelerine erişimi frameworkün implementasyon tarzına göre değişen bir durumdur.

Java kökenli web programlamada ulaşılan en son durum MVC web frameworkleriydi. Bu durum Wicket öncesi geçerliydi. Wicket icat edildi, mertlik bozuldu :)

Wicket Geldi, Hoş Geldi!

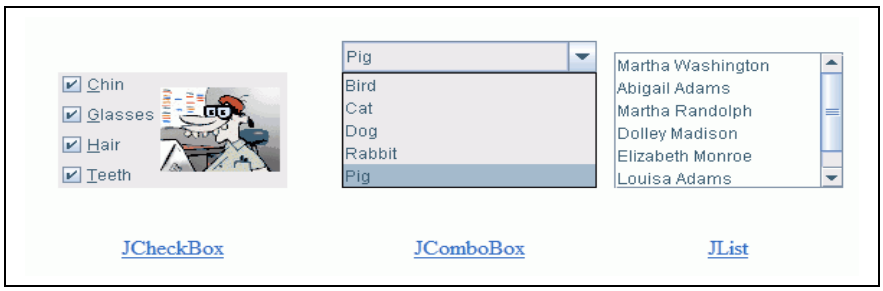
Wicket Apache Software Foundation tarafından geliştirilen bir web framework. Eğer daha önce Swing[7] ile masaüstü aplikasyonları programladıysanız, nasıl hazır komponentleri bir araya getirerek bir kullanıcı arayüzünün (GUI – Graphical User Interface) oluşturulduğunu biliyorsunuz demektir. Bunun ne anlama geldiğini Swing bilmeyen okuyucularımız yazının ilerleyen satırlarında görecekler. Wicket'in sağladığı web aplikasyonlarındaki avantajı görebilmek için arayüzlerde kullanılan komponentlerin iyi anlaşılması gerekiyor. Bir sonraki resimde tipik bir Swing penceresi yer almaktadır.

[7] Bakınız: http://java.sun.com/docs/books/tutorial/uiswing/



Resim 3 Swing penceresi

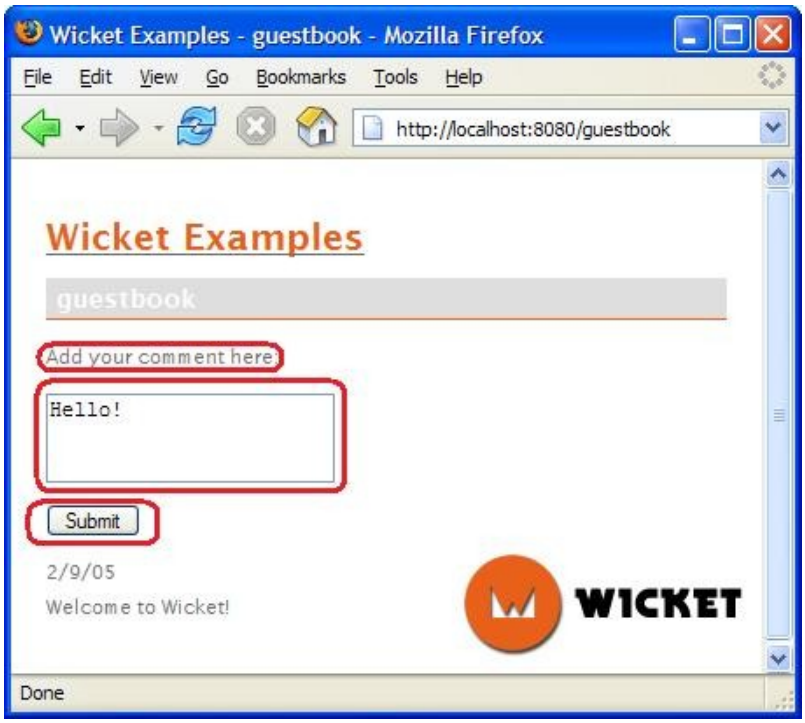
Kırmızı kare içinde yer alan alanlarda Swing komponentleri kullanılmıştır. Örneğin localhost kelimesinin yer aldığı alan için `JtextField` Swing komponenti kullanılmıştır. Swing bünyesinde buna benzer, arayüz oluşturmak için kullanılan birçok komponent bulunmaktadır. Bir sonraki resimde bazı Swing komponentleri yer almaktadır.



Resim 4 Swing komponentleri

Swing aplikasyonları standart Swing GUI komponentleri kullanılarak oluşturulur. Yazılan kodun tekrar kullanımı kolaylaştığı için komponentlerden söz edilmektedir, çünkü aynı komponentler kullanılarak değişik tipte arayüzler programlanabilmektedir.

HTML arayüzlerinde de böyle komponentler kullanmak mümkündür. Bir sonraki resimde yer alan web sayfasında <textarea> ve <input type=submit> gibi HTML tagler kullanılmıştır.



Resim 5 HTML komponentleri

Swing ile hazırlanan aplikasyonlarda komponenler JFrame tipi sınıflarda bir araya getirilerek arayüzler oluşturulmaktadır. Aplikasyon için tüm programlama Java dilinde yapılabilmektedir. Web aplikasyonlarında durum farklıdır. Daha öncede gördüğümüz gibi HTML arayüzlerinin özel programlamaya ihtiyaç duyan JSP sayfalarında, gerekli Java kodunun da Java sınıflarında programlanması gerekiyor. Bu durum JSP sayfalarının tekrar kullanımını ve komponent oluşturma işlemini güçleştirmektedir. Wicket ile bu sorun ortadan kalkıyor!

Wicket, Swing'de olduğu gibi bir tekrar kullanılabilir komponent kütüphanesine sahiptir. Örneğin bir önceki resimde yer alan arayüzü programlamak için JSP sayfalarına gerek kalmıyor, çünkü tüm programlama işlemi Swing'de olduğu gibi Wicket komponentleri kullanılarak Java sınıflarında yapılmaktadır. Wicket'in sunduğu komponent kütüphanesi aşağıdaki komponentleri ihtiva eder:

Ekran çıktısı için:

wicket.markup.html.basic.Label
wicket.markup.html.basic.MultiLineLabel

Layout oluşturma ve komponentleri gruplamak için:

wicket.markup.html.panel.Panel
wicket.markup.html.border.Border
wicket.markup.html.include.Include
wicket.markup.html.tabs.TabbedPanel (wicket-extensions)
wicket.markup.html.panel.Fragment

Sayfalar arası HTML linkleri oluşturmak için:

wicket.markup.html.link.Link
wicket.markup.html.link.ExternalLink
wicket.markup.html.link.PageLink
wicket.markup.html.link.BookmarkablePageLink

HTML formları oluşturmak için:

wicket.markup.html.form.Form
wicket.markup.html.form.Button
wicket.markup.html.form.SubmitLink
wicket.markup.html.form.TextField
wicket.markup.html.form.TextArea
wicket.markup.html.form.CheckBox
wicket.markup.html.form.CheckGroup
wicket.markup.html.form.Check
wicket.markup.html.form.CheckGroupSelector
wicket.markup.html.form.CheckBoxMultipleChoice
wicket.markup.html.form.palette.Palette
wicket.markup.html.form.DropDownChoice
wicket.markup.html.form.ListChoice
wicket.markup.html.form.RadioChoice
wicket.markup.html.form.RadioGroup and wicket.markup.html.form.Radio
wicket.markup.html.form.ListMultipleChoice
wicket.extensions.markup.html.form.select.Select
wicket.extensions.markup.html.palette.Palette

Wicket Örneği

Wicket ile bir ziyaretçi defteri oluşturarak, Wicket'in sahip olduğu komponent modelinin nasıl kullanıldığını yakından inceleyelim.

Resim 6 Wicket ile ziyaretçi defteri aplikasyonu

Bir önceki resimde yer aldığı gibi ziyaretçi defterine kayıt yapılmak üzere bir HTML arayüz oluşturulmuştur. Bu arayüz, bünyesinde bir TextArea ve bir Buton barındırmaktadır.

Böyle bir aplikasyonu geliştirirken, nesneler bazında düşünerek modelleme yapmamız faydalı olacaktır. Yapılan yorumların yer aldığı **Comment** isminde bir sınıf oluşturarak, programlama işlemine başlıyoruz.

```
package wicket.examples.guestbook;


import java.io.Serializable;
import java.util.Date;


public class Comment implements Serializable
{
    public String text;
    public Date date = new Date();
}
```

Wicket ile bir web sayfası oluşturabilmek için **WebPage** sınıfını genişleten bir Java sınıfı oluşturmamız gerekiyor. **GuestBook** isminde, web arayüzünü oluşturan ilk Java sınıfını programlıyoruz.

```
package wicket.examples.guestbook;


public final class GuestBook extends WebPage
{
    /** Use a Vector, as it is synchronized. */
    private static final List commentList = new Vector();


    public GuestBook()
    {
    }
}
```

Oluşturmak istediğimiz HTML arayüzünde bir HTML formu yer alıyor. Submit butonuna tıklandığında kullanıcının girmiş olduğu metnin işlem görmek üzere web aplikasyonuna gönderilmesi gerekiyor. HTML arayüzlerde kullanılan formları oluşturmak için Wicket bünyesinde Form (wicket.markup.html.form.Form) komponenti bulunmaktadır.

```
public final class CommentForm extends Form
   {
      private final Comment comment = new Comment();


      public CommentForm(final String componentName)
      {
         super(componentName);
         add(new TextArea("text", new PropertyModel(comment,
"text")));
      }


      public final void onSubmit()
      {
```

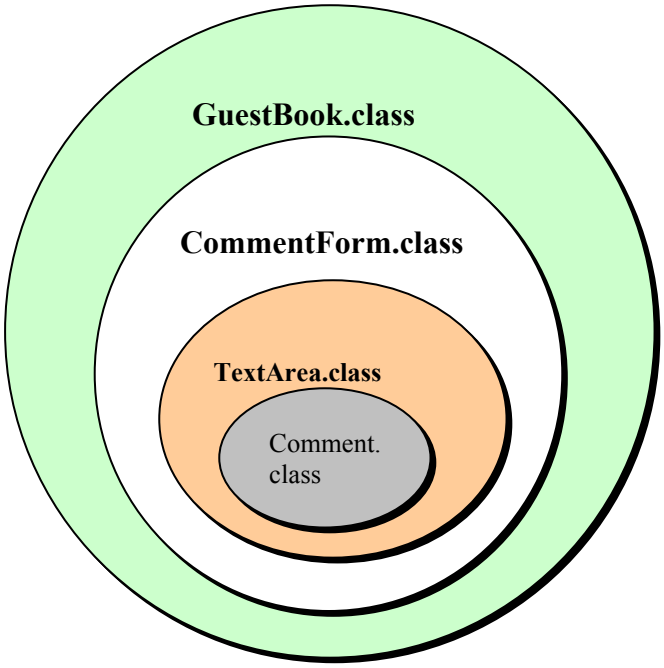
```
            final Comment newComment = new Comment();
            newComment.text = comment.text;

            commentList.add(0, newComment);
            commentListView.modelChanged();

            comment.text = "";
        }
    }
```

Swing aplikasyonlarında olduğu gibi arayüzü oluşturan komponentleri add() komutuyla Form komponentine ekliyoruz. Resim 6 da yer alan arayüzü oluşturmak için **CommentForm** sınıfına TextArea komponentini eklememiz yeterli olacaktır. **TextArea** (wicket.markup.html.form.TextArea) Wicket bünyesinde bulunan ve metin alanları oluşturmak için kullanılan bir komponenttir. **PropertyModel** ile kullanıcının ziyaretçi defterine yazdıkları bir **Comment** nesnesinde tutulmaktadır. **Comment** sınıfı bu durumda MVC frameworklerde kullanılan Model sınıfı haline gelmektedir. Model sınıflarında veriler barınır.

```
public final class GuestBook extends WebPage
{
    /** Use a Vector, as it is synchronized. */
    private static final List commentList = new Vector();
    private final ListView commentListView;

    public GuestBook()
    {
        add(new CommentForm("commentForm"));
        add(commentListView = new ListView("comments", commentL-
ist)
        {
            public void populateItem(final ListItem listItem)
            {
                final Comment comment =
(Comment)listItem.getModelObject();
```

```
                listItem.add(new Label("date", comment.date.toString()));
                listItem.add(new MultiLineLabel("text", comment.text));
            }
        });
    }
}
```

Oluşturduğumuz form komponentini (CommentForm) GuestBook sınıfına add() metodunu kullanarak ekliyoruz. Bu şekilde bir komponent hiyerarşisi oluşturmuş olduk. Oluşturduğumuz hiyerarşinin yapısı şu şekildedir: **GuestBook > CommentForm > TextArea > Comment**.



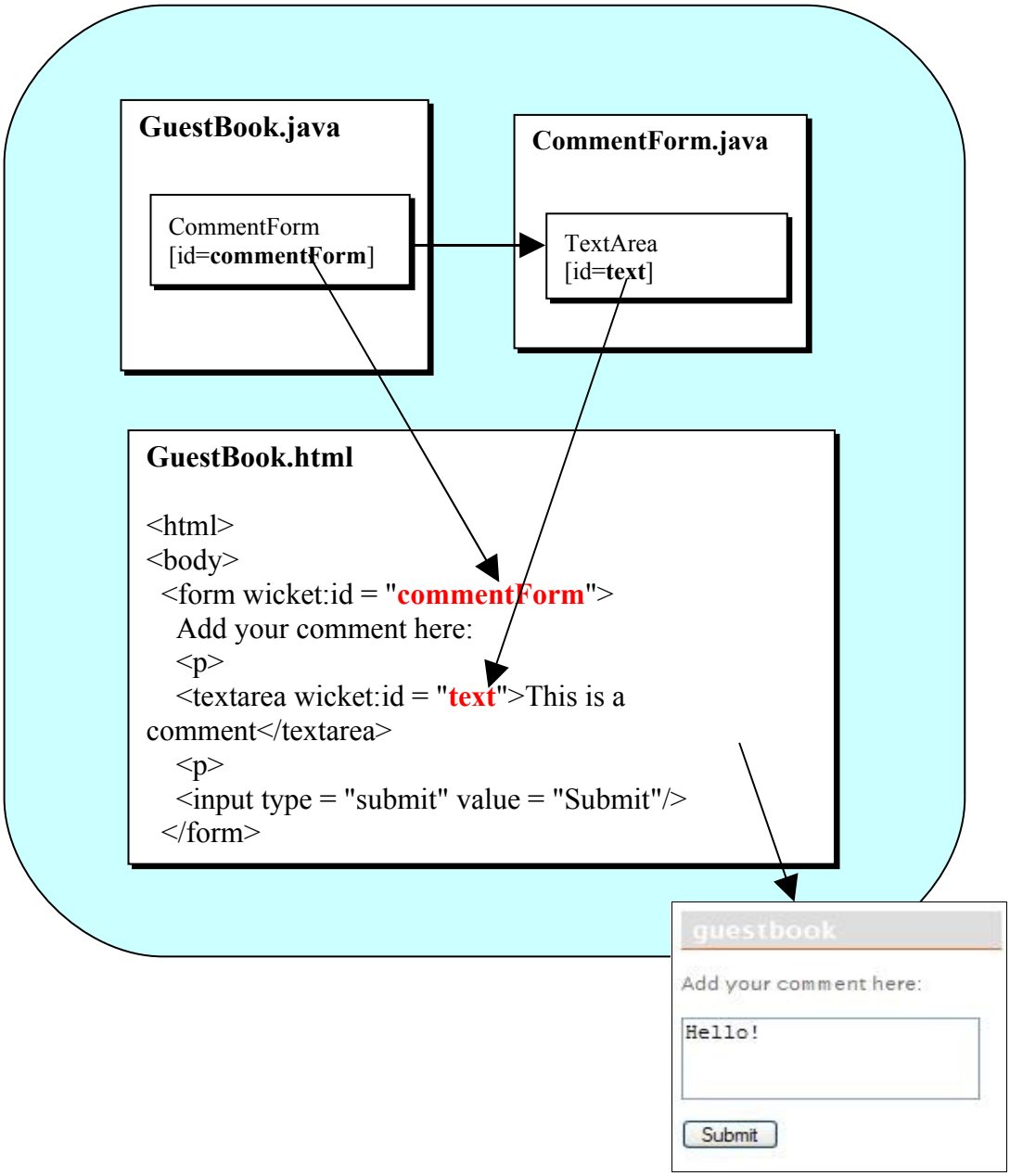
Wicket, bu komponent hiyerarşisinin server tarafında, sahip olduğu şekli kaybetmeden saklanmasını sağlar. Artık yavaş yavaş Swing aplikasyonlarında olduğu gibi komponentleri kullanarak aplikasyonu oluşturmaya başladık ve şimdiye kadar programlamanın hepsini Java sınıflarında ve Wicket komponenetlerini kullanarak yapıyoruz.

Kullanıcı ziyaretçi defteri için yazısını tamamladıktan sonra Gönder butonuna tıkladığında, **CommentForm** sınıfında yer alan onSubmit() metodu devreye girer. Bu durumda yeni bir **Comment** nesnesi oluşturularak **commentList** listesine eklenir. Sayfa yeniden görüntülendiğinde GuestBook konstrüktöründe görüldüğü gibi liste içinde yer alan Comment nesneleri için yeni bir **commentListView** nesnesi oluşturulur ve populateItem() metodu implemente edilerek, tüm listenin bir döngü içinde ekranda görüntülenmesi sağlanır. Bu işlem için Wicket bünyesinde bulunan **Label** ve **MultiLineLabel** komponentleri kullanılır. Döngü işlemi için burada for() döngüsü kullanılmamıştır. Wicket **ListView** komponenti kullanıldığı taktirde, Wicket otomatik olarak döngü mekanizmalarını oluşturarak, listenin ekranda görüntülenmesini sağlar.

Buraya kadar güzel, ama hiç HTML kodu olmayacak mı diye bir soru gelmiş olabilir aklınıza. Wicket ile web programlaması yaparken de HTML kodu oluşturmamız gerekiyor, lakin bunun için özel bir programlama yapmamız gerekmiyor. Wicket'i diğer frameworklerden ayıran en önemli özelliklerden birisi budur. GuestBook için kullanılan HTML kodu aşağıda yer almaktadır.

```
<html>
<body>
 <form wicket:id = "commentForm">
  Add your comment here:
  <p>
  <textarea wicket:id = "text">This is a comment</textarea>
  <p>
  <input type = "submit" value = "Submit"/>
 </form>
 <p>
 <span wicket:id = "comments">
  <p>
    <span wicket:id = "date">1/1/2004</span><br>
    <span wicket:id = "text">Comment text goes here.</span>
  </p>
 </span>
 <wicket:remove>
  <p>
    1/2/2004<br/>
    More comment text here.
  </p>
 </wicket:remove>
</body>
</html>
```

Yukarda yer alan HTML kodunu **GuestBook.html** isminde bir dosyaya yerleştirerek **GuestBook.java** dosyası ile aynı dizinde olmasını sağlıyoruz. Bir sonraki resimde kullanılan komponentler arasındaki ilişki yer almaktadır.



Resim 7 Wicket Java sınıfları ve HTML dosyaları arasındaki ilişki

GuestBook.java sınıfında kullandığımız komponentler ile **GuestBook.html** dosyasını ilişkilendirmek için **GuestBook.html** dosyasında **wicket:id** tagı kullanılır. Örneğin **CommentForm** sınıfını komponent olarak GuestBook sınıfına eklemek için add() komutunu kullandık. CommentForm sınıfından bir komponent oluştururken konstrüktör parametresi olarak *commentForm* değeri yer aldı. Bu isim **GuestBook.html** dosyasında <form wicket:id=”commentForm”> ile Java kodunun HTML koduyla birleştirilmesinde kullanılır. **GuestBook** sınıfında oluşturduğumuz komponent hiyerarşisini de birebir **GuestBook.html** sayfasında oluşturmamız gerekiyor. **GuestBook.html** dosyasına baktığımızda <form wicket:id=”commentForm”> tagı içinde <textarea wicket:id=”text”> tagını kullandık, bu da GuestBook > CommentForm > TextArea hiyerarşisine tekabüldür.

Wicket otomatik olarak **GuestBook.html** dosyasında yer alan ve wicket:id ile tanımlanmış alanlara GuestBook.java da kullanılan komponentleri ve oluşan değerleri yerleştirir.

```
package wicket.examples.guestbook;

import wicket.protocol.http.WebApplication;

public class GuestBookApplication extends WebApplication
{
    public GuestBookApplication()
    {
    }

    public Class getHomePage()
    {
        return GuestBook.class;
    }
}
```

Wicket ile oluşturulan bir web aplikasyonu birden fazla sayfadan (GuestBook gibi) oluşabilir. Sadece arayüzlerin bulunduğu sayfaların oluşturulması yeterli değildir. Aplikasyonun start alacağı ve **WepApplication** sınıfını genişleten yeni bir sınıf oluşturmamız gerekiyor. Bir önceki örnekte oluşturduğumuz **GuestBookApplication** sınıfı ile aplikasyonun start alabileceği sayfayı getHomePage() metodunda tanımlıyoruz. **GuestBookApplication** sınıfında aplikasyon genelinde geçerli olan işlemler yapılır.

Oluşturduğumuz Wicket tabanlı web aplikasyonu Tomcat gibi bir aplikasyon serverinde çalışır hale getirmek için web.xml ismini taşıyan bir konfigürasyon dosyasına ihtiyacımız var. GuestBook aplikasyonu için kullanılabilecek web.xml konfigürasyon dosyası aşağıda yer almaktadır.

```
<servlet>
  <servlet-name>GuestBookApplication</servlet-name>
  <servlet-class>wicket.protocol.http.WicketServlet</servlet-class>
  <init-param>
    <param-name>applicationClassName</param-name>
```

```
    <param-value>wicket.examples.guestbook.GuestBookApplica-
tion</param-value>
  </init-param>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>GuestBookApplication</servlet-name>
  <url-pattern>/app/*</url-pattern>
</servlet-mapping>
```

Wicket İle YazilimSozlugu.org Projesi

Bu bölüme Eclipse altında bir Wicket projesinin oluşturulma işlemini yakından inceleyeceğiz. Türkçe makale yazarken yabancı terimlerin Türkçe karşılıklarını bulmakta zaman zaman çok zorlandığımız için YazilimSozlugu.org isminde bir proje başlattık[8].

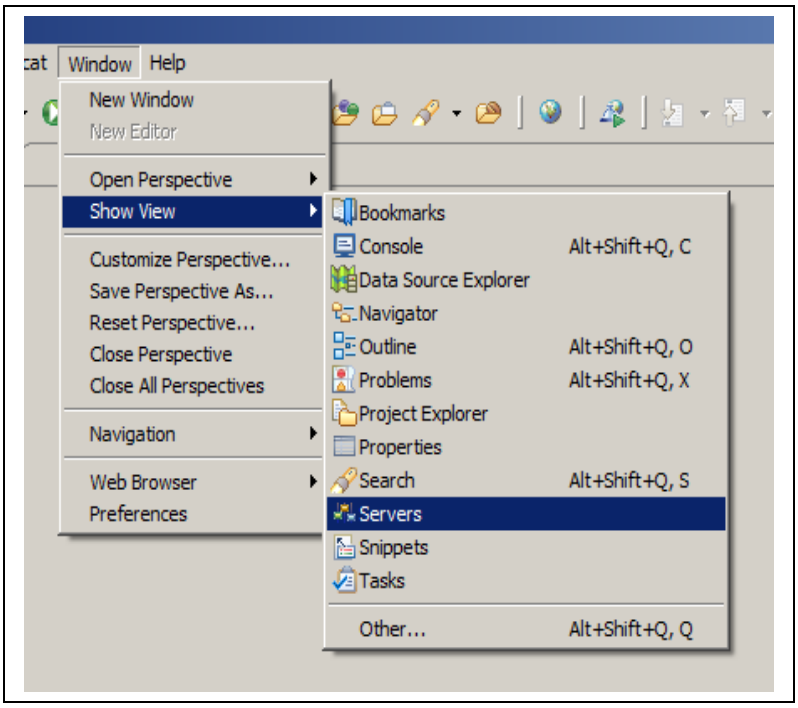
YazilimSozlugu.org projesi Wicket ile hazırlandı. Kullanılan teknolojik ögeler şöyledir:

- Wicket 1.3.4
- Spring 2.5
- Hibernate 3.2.6
- Eclipse 3.4
- JUnit 3.8
- DBUnit 2.2
- Ant 1.7
- HSQL 1.8.0
- Apache Commons kütüphaneleri

Yazının bundan sonraki bölümünde Wicket kullanılarak YazilimSozlugu.org projesi için gerekli teknik alt yapının nasıl oluşturulduğunu yakından inceleyeceğiz.

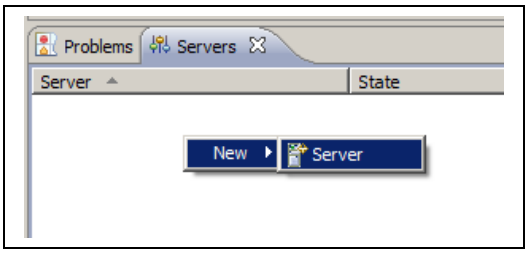
Eclipse altında web tabanlı bir program yazılımı yapabilmek için önce kullanmak istediğimiz server (Tomcat gibi bir application server) tipini tanımlamamız gerekiyor.

[8] Bakınız: http://www.kurumsaljava.com/projeler/



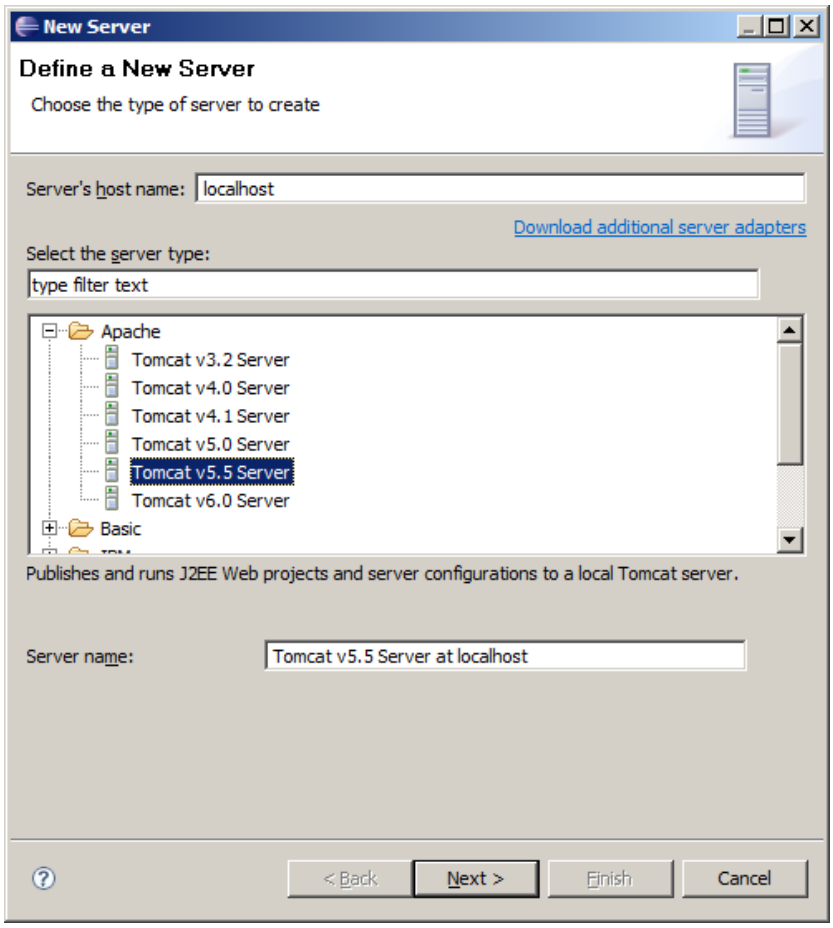
Resim 8 Servers paneline erişim

Kullanmak istediğimiz serveri tanımlamak için Window > Show View > Servers menüsünden Servers paneline erişiyoruz.



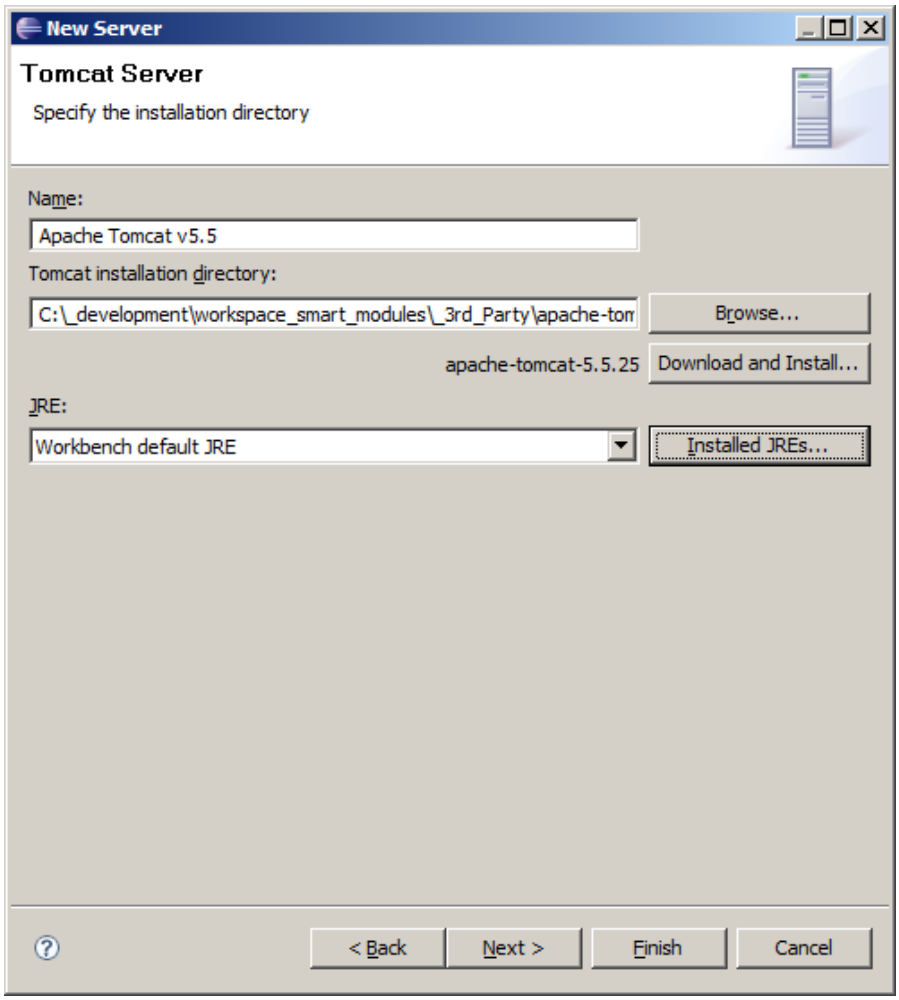
Resim 9 New > Server ile yeni bir server tanımlaması yapılabilir

Servers paneli içinde sağ tuşa tıkladığımızda resim 9 da yer alan menü oluşur. Bu menü üzerinden server tanımlamasını yapabiliriz.



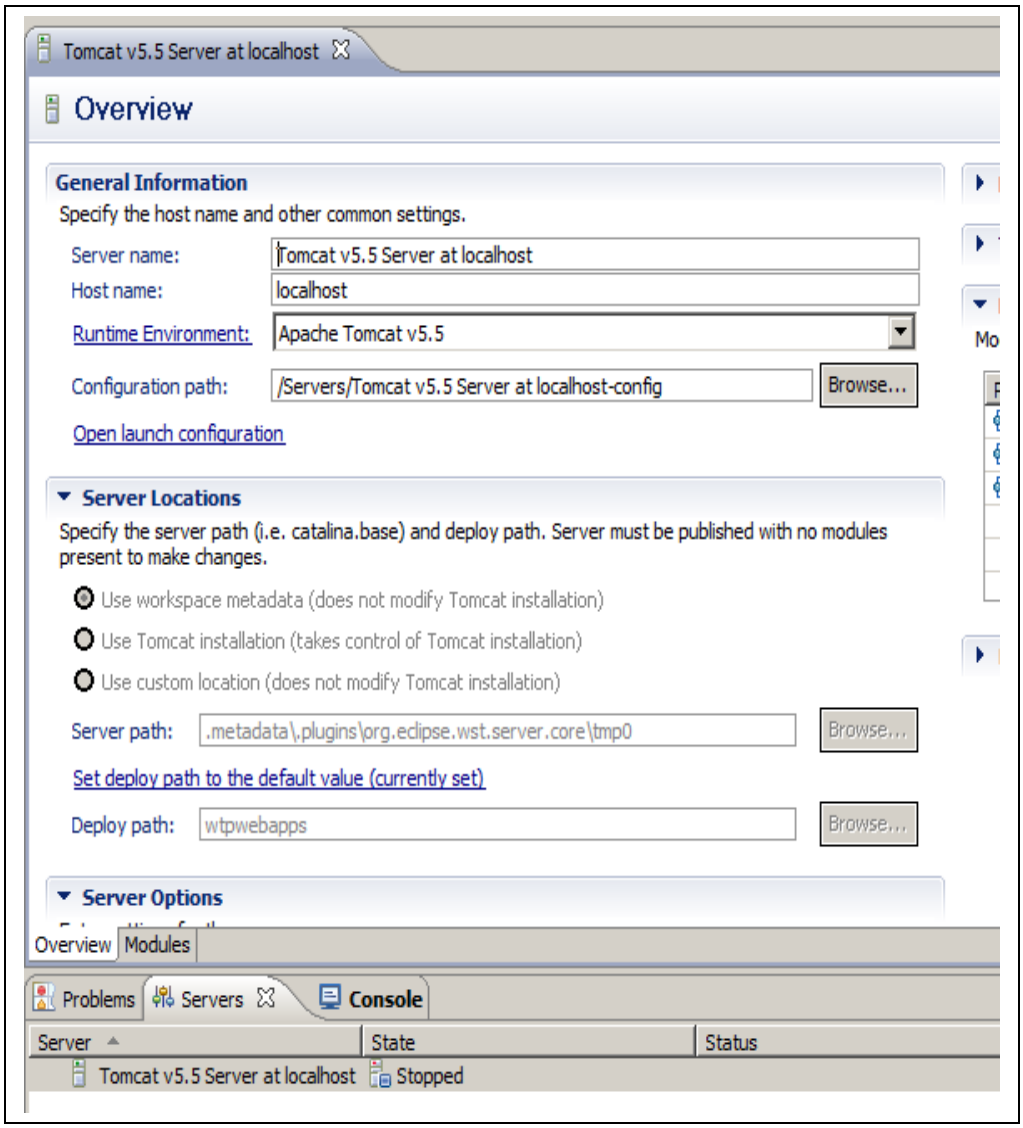
Resim 10 Server tipi seçimi

Kullanmak istediğimiz server tipi Tomcat 5.5 versiyonudur. Next butonuna tıklayarak, bir sonraki sayfaya geçiyoruz.



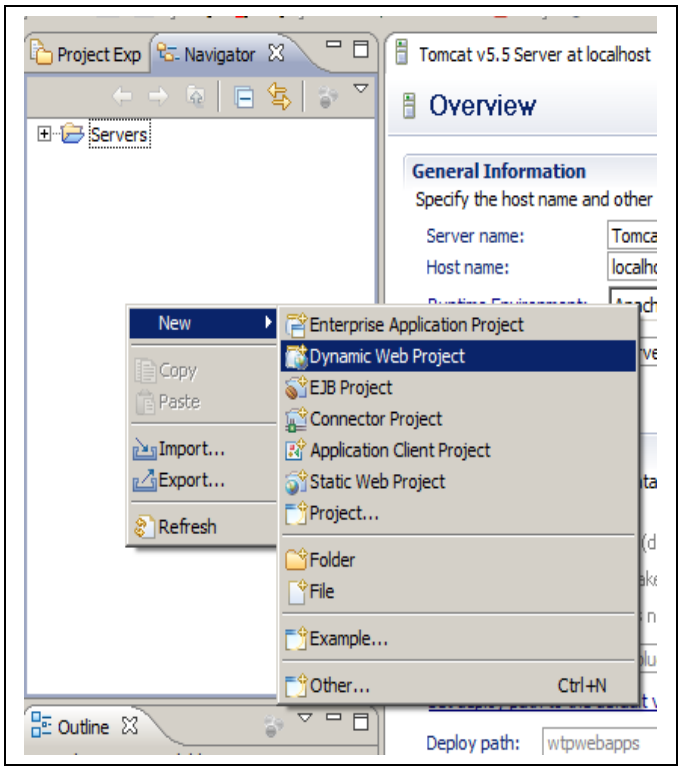
Resim 11 Tomcat konfigürasyonu

Browse... butonuna tıklayarak, daha önceden edindiğimiz ve herhangi bir dizine yerleştirdiğimiz Tomcat 5.5. sürümünün bulunduğu dizini seçiyoruz. Finish butonuna tıklayarak, ayarları tamamlıyoruz.



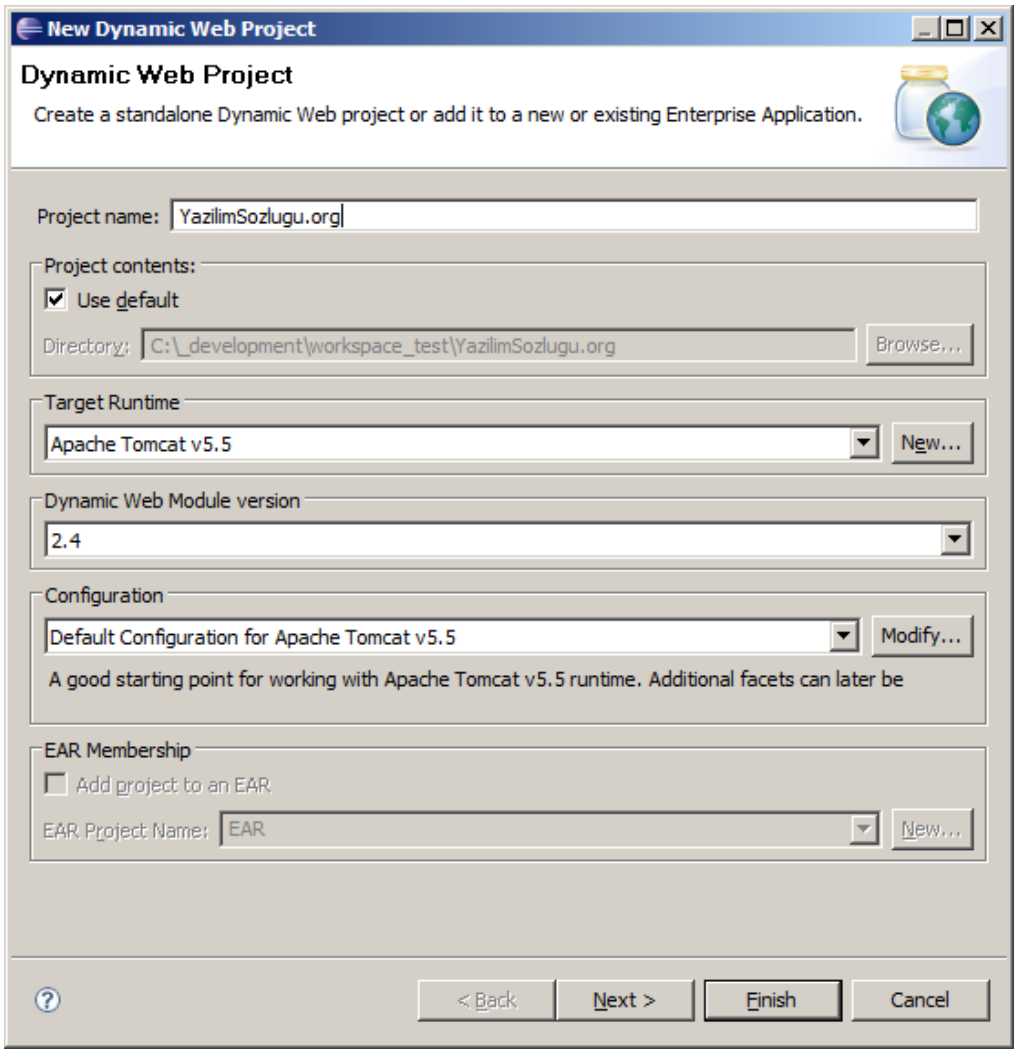
Resim 12 Tomcat server ayarları

Bu işlemlerin ardından Servers panelinde **Tomcat v5.5 Server at localhost** isminde yeni bir server kaydı oluşur. Bu bizim projemizde kullanacağımız serverdir.

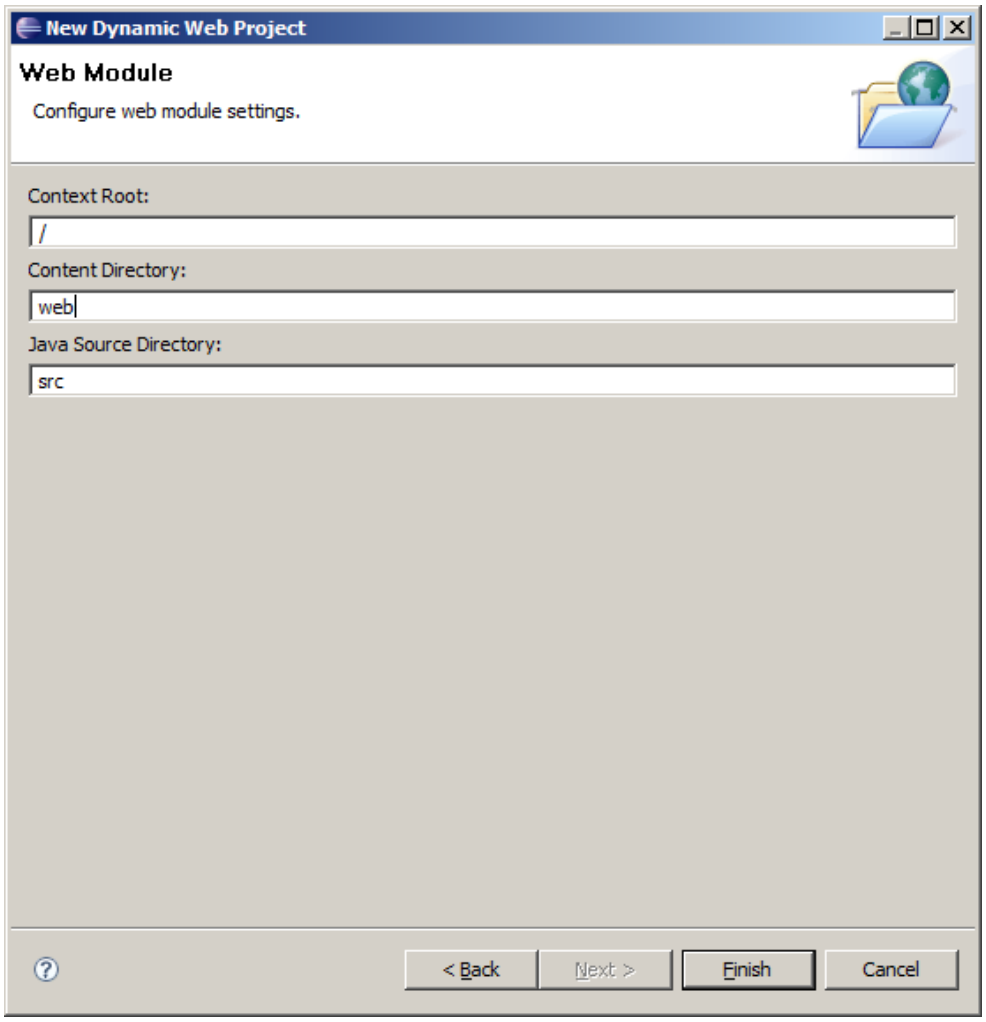


Resim 13 Yeni bir web projesi oluşturma menüsü

Resim 13 de yer aldığı gibi bir Dynamic Web Project (dinamik web projesi) oluşturuyoruz.

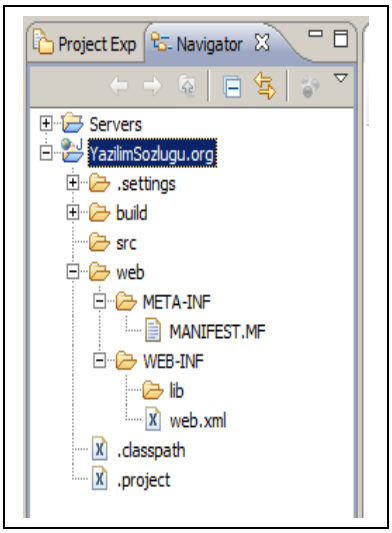


Resim 14 Yeni bir web projesi oluşturma paneli



Resim 15 Yeni bir web projesi oluşturma paneli

Resim 15 de yer alan panelde oluşturduğumuz ve Tomcat serverinde çalıştırılacak olan web aplikasyonu için / şeklinde bir context root tanımladık. Bu ne anlama gelmektedir? Bir aplikasyon bünyesinde birden fazla web aplikasyonu çalışıyor olabilir. Aplikasyonları birbirinden ayırt etmek için context root isminde bir değişken kullanılır. Bu genelde sadece bir aplikasyonun olduğu ortamlarda / şeklindedir, yani aplikasyona web tarayıcısı (browser) üzerinden erişmek için http://<IP-ADRESI>/ (örneğin http://localhost/) adresi kullanılır. Birden fazla web aplikasyonun web adreslerini ayırt etmek için **/app1** yada **/app2** seklinde context root kullanılabilir. Bu durumda bu web aplikasyonları http://<IP-ADRESI/app1 (örneğin http://localhost/app1) ya da http://<IP-ADRESI>/app2 (örneğin http://localhost/app2) şeklinde adreslenebilir.



Resim 16 Proje yapısı

Oluşturulan proje yapısı resim 16 da yer almaktadır. Dizin yapısı şu şekildedir:

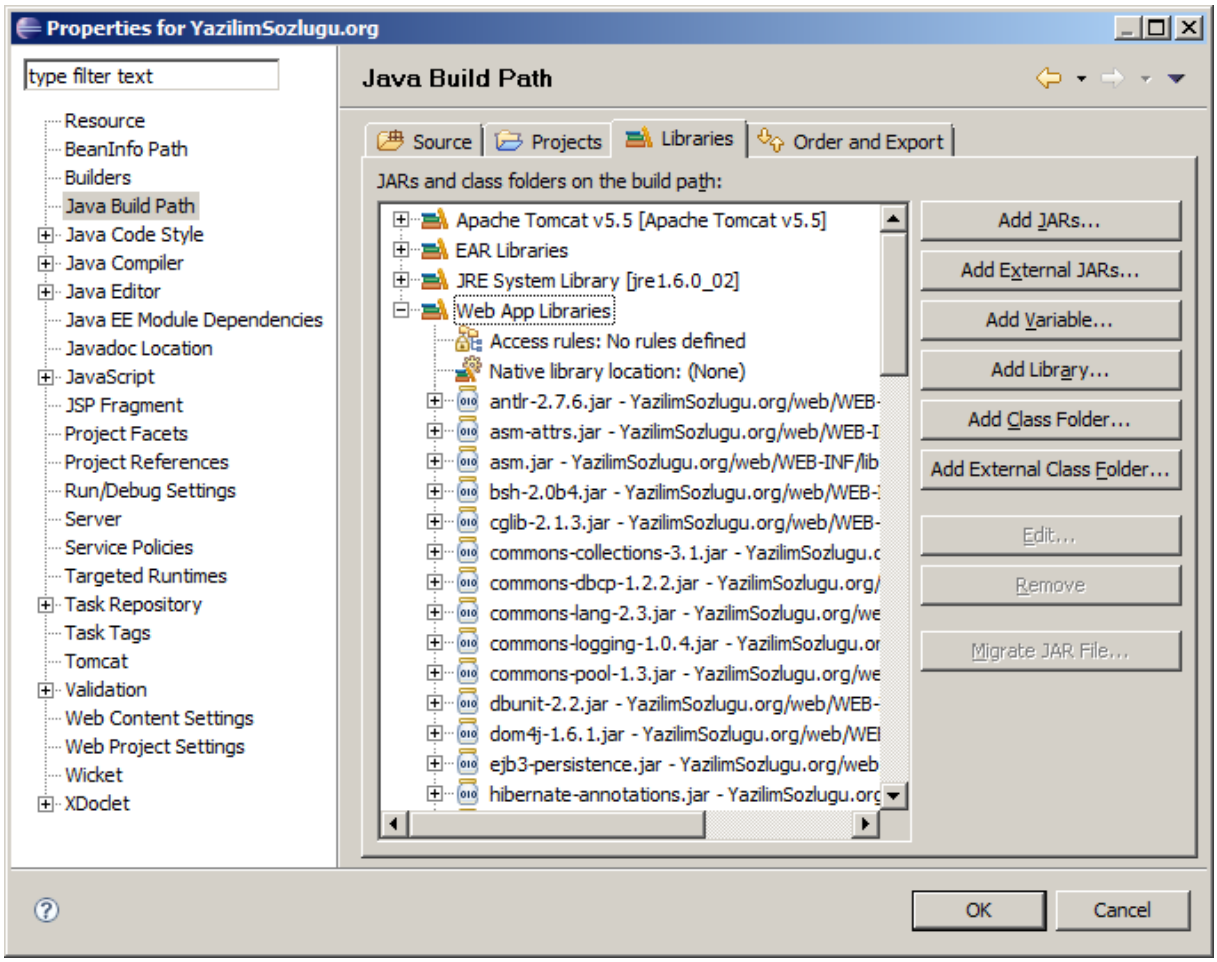
- **settings**: Projenin Eclipse ayarları bu dizinde yer alır.
- **build**: Eclipse tarafından proje bünyesindeki tüm sınıflar bu dizinde bulunan classes dizinine derlenir.
- **src**: Java sınıflarının yer aldığı dizin.
- **web**: Web aplikasyonu için oluşturulan ana dizin. WEB-INF ve META-INF gibi dizinleri bünyesinde barındırır.

Proje için gerekli tüm Wicket ve diğer framework kütüphanelerini **web/WEB-INF/lib** dizinine kopyalamamız gerekiyor. Kullanılan kütüphaneler:

antlr-2.7.6.jar
asm-attrs.jar
asm.jar
bsh-2.0b4.jar
cglib-2.1.3.jar
commons-collections-3.1.jar
commons-dbcp-1.2.2.jar
commons-lang-2.3.jar
commons-logging-1.0.4.jar
commons-pool-1.3.jar
dbunit-2.2.jar
dom4j-1.6.1.jar
ejb3-persistence.jar
hibernate-annotations.jar
hibernate-commons-annotations.jar
hibernate-tools.jar
hibernate-validator.jar
hibernate3.jar
hsqldb.jar
jta.jar

junit-3.8.1.jar
log4j-1.2.15.jar
slf4j-api-1.5.2-sources.jar
slf4j-api-1.5.2.jar
slf4j-log4j12-1.5.2-sources.jar
slf4j-log4j12-1.5.2.jar
slf4j-simple-1.5.2-sources.jar
slf4j-simple-1.5.2.jar
spring-aop-sources.jar
spring-aop.jar
spring-beans-sources.jar
spring-beans.jar
spring-context-sources.jar
spring-context-support-sources.jar
spring-context-support.jar
spring-context.jar
spring-core-sources.jar
spring-core.jar
spring-jdbc-sources.jar
spring-jdbc.jar
spring-jms-sources.jar
spring-jms.jar
spring-mock.jar
spring-orm-sources.jar
spring-orm.jar
spring-sources.jar
spring-test-sources.jar
spring-test.jar
spring-tx-sources.jar
spring-tx.jar
spring-validation-0.8.jar
spring.jar
wicket-1.3.4.jar
wicket-auth-roles-1.3.4.jar
wicket-datetime-1.3.4.jar
wicket-extensions-1.3.4.jar
wicket-ioc-1.3.4.jar
wicket-jmx-1.3.4.jar
wicket-objectsizeof-agent-1.3.4.jar
wicket-spring-1.3.4.jar
wicket-spring-annot-1.3.4.jar

web/WEB-INF/lib dizinine eklenen tüm dosyalar otomatik olarak projenin classpath değişkenine dahil edilir:



Resim 17 Proje Build Path paneli

Her web projesinin web.xml isminde ve web/WEB-INF/ dizininde yer alan bir konfigürasyon dosyası vardır. YazilimSozlugu.org projesi için kullandığımız web.xml şu şekildedir:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" 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">

      <display-name>YazilimSozlugu.org</display-name>

      <listener>
            <listener-class>
                  org.springframework.web.context.ContextLoaderListener
            </listener-class>
      </listener>

      <context-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:applicationContext.xml</param-value>
      </context-param>

      <servlet>
            <servlet-name>Application</servlet-name>
            <servlet-class>
                  org.apache.wicket.protocol.http.WicketServlet
            </servlet-class>
            <init-param>
                  <param-name>applicationClassName</param-name>
                  <param-value>
                        org.yazilimsozlugu.Application
                  </param-value>
            </init-param>
            <init-param>
                  <param-name>applicationFactoryClassName</param-name>
                  <param-value>

      org.apache.wicket.spring.SpringWebApplicationFactory
                  </param-value>
            </init-param>
            <load-on-startup>1</load-on-startup>
      </servlet>

      <servlet-mapping>
            <servlet-name>Application</servlet-name>
            <url-pattern>/*</url-pattern>
      </servlet-mapping>


      <welcome-file-list>
            <welcome-file>/index.html</welcome-file>
      </welcome-file-list>


</web-app>
```

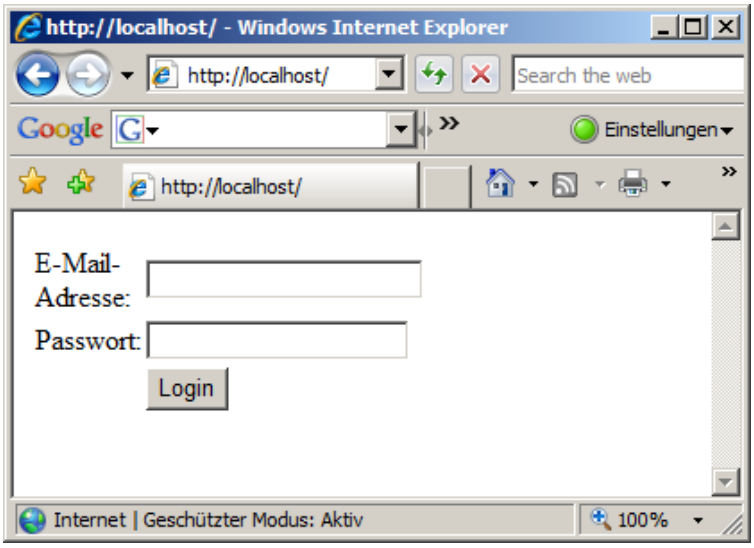
Proje için gerekli teknik ayarlar burada son buluyor. Bunu örnek alarak kendi Wicket projelerinizi oluşturabilirsiniz.

Komponentlere Daha Büyük Bir Çerçeveden Bakmak

Daha öncede belirttiğim gibi Wicket kodun tekrar kullanımını kolaylaştıran bir frameworktür. Tekrar kullanılabilir kodu **komponent** olarak düşünebiliriz. Eğer yazılımı, tekrar kullanılabilir komponentler olarak gerçekleştirebilirsek, oluşan komponentleri başka projeler bünyesinde de kullanabiliriz ki, bu da diğer projelerin süresini aşağıya çekebilecek bir unsundur. Yazının başında da belirttiğim gibi, Wicket'i kullanmamın en büyük sebeplerinden birisi de, oluşturmuş olduğum komponentleri başka bir projede kullanabilmemdir. Tabii bu sadece Wicket'in sahip olduğu standart komponentleri kullanarak yapılabilecek bir iş değil. Wicket komponentleri tekrar kullanılabilir özellikte, lakin işlevsel olarak çok küçük kod üniteleridir. Daha geniş kapsamlı komponentler oluşturabilmek için başka bir mekanizmaya ihtiyacımız var. Standart Wicket komponentlerini bir araya getirerek, daha büyük ve tekrar kullanılabilir komponentler oluşturma yeteneğine sahip olmamız gerekiyor. Tekrar kullanılabilir, geniş kapsamlı komponentler oluşturmak için Wicket **Panel** (wicket.markup.html.panel.Panel) komponenti kullanılabilir.

Panel sınıfını kullanarak, birden fazla standart Wicket komponentinin bir arada kullanıldığı, tekrar kullanılabilir türde komponentler oluşturmak mümkündür. Panel sınıflarından türetilen komponentlerin kendi HTML dosyaları vardır. Panel olarak oluşturulan ve işlevsel olarak daha kapsamlı olan komponentler herhangi bir **WebPage** sayfasında, JSP sayfalarındaki include mekanizmasına benzer bir şekilde kullanılabilir. Bu şekilde panel olarak oluşturulan komponentleri değişik projelerde kullanmak mümkün olmaktadır.

Yazımın başında da belirttiğim gibi, değişik projeler için aynı türde yazılımı yapmak ve kodu tekrar kullanamamak zaman içinde motivasyon kırıcı bir hal almıştı. İstediğim, örneğin bir login komponenti oluşturmak ve değişik projelerde, bir kere oluşturduğum login kodunu tekrar kullanabilmekti. Bu belki diğer frameworklerde copy/paste usulüyle mümkün olabilecek bir şey. Sonuç itibariyle bir proje bünyesinde oluşturduğum login modülünün ihtiva ettiği JSP sayfalarını ve kodu yeni bir projeye transfer ederek, kodu bir şekilde tekrar kullanabilirim. Ama bu gerçek anlamda bir komponent kullanımı değil! Tekrar kullanılabilir bir komponent, JSTL komponentlerinde olduğu gibi bir Jar dosyası içinde projeye dahil edilerek hemen kullanılabilir yapıda olmalıdır. Wicket ile bu mümkün!



Resim 19 Login sayfası

Resim 19 da, değişik projelerde kullanılmak üzere oluşturduğum Wicket login komponenti bulunuyor. Bu komponenti Panel sınıfını kullanarak oluşturdum.

```
package smart.web.login.presentation.panel;

import org.apache.wicket.markup.html.panel.Panel;
import org.apache.wicket.spring.injection.annot.SpringBean;
import smart.web.login.presentation.form.factory.LoginFormFactory;


public class LoginPanel extends Panel
{
	private static final long serialVersionUID = 1L;

	@SpringBean(name= "LoginFormFactory")
	private LoginFormFactory factory;

	public LoginPanel(final String arg0)
	{
		super(arg0);
		add(this.factory.getForm("loginform"));
	}
}
```

LoginPanel sınıfı *loginform* ismini taşıyan bir **Form** ihtiva ediyor. Login komponentinin *loginform* gibi ihtiva ettiği alt komponentleri, komponentin yapısını değiştirmek zorunda kalmadan her proje için değişik tarzda konfigüre edebilmek için, **login.properties** isminde bir konfigürasyon dosyası mevcut. Bir sonraki tabloda login.properties dosyası yer almaktadır.

```
# used panel type
```

```
panel=smart.web.login.presentation.panel.LoginPanel

# used form type
form=smart.web.login.presentation.form.LoginForm

# redirect page
redirect=smart.web.login.presentation.page.DummyRedirectPage

# used account type
account=smart.common.accountmanager.domain.Account
```

login.properties bünyesinde kullanılan Panel, Form ve diğer alt komponentler yer almaktadır. Login komponenti bir Jar dosyasında yer alır. Bu komponent başka bir projede kullanılmak istendiğinde, login.jar dosyasını projeye dahil etmek yeterli olacaktır. Komponent ayarlarını yapmak için login.properties dosyasını Spring yardımıyla değiştirebiliriz. Bunun bir örneği aşağıdaki kod da yer almaktadır.

```
<bean id="LoginManagerWebImplPropertyPlaceholder"

class="org.springframework.beans.factory.config.PropertyPlaceholderConfigu
rer">
            <property name="location"   value="classpath:login.properties"
/>
            <property name="placeholderPrefix" value="$login{" />
      </bean>


      <!--  PANEL FACTORY -->
      <bean id="LoginPanelFactory"

class="smart.web.login.presentation.panel.factory.LoginPanelFactory2"
scope="singleton">
            <property name="config" ref="LoginPanelFactoryConfig"/>
      </bean>
```

Spring yardımıyla, classpath içinde bulunan login.jar dosyasının ihtiva ettiği login.properties dosyasını değiştirebiliriz. Yukarda yer alan örnekte bir **LoginPanel** oluşturmak için **LoginPanelFactory2** sınıfı kullanılmaktadır.

Login komponenti, HTML login sayfasını oluşturmak için **LoginPanel.html** isminde bir dosyaya sahiptir. Bu dosya Wicket tarafından HTML arayüzü oluşturmak için kullanılır ve aşağıdaki yapıya sahiptir.

```
<wicket:panel>

      <form wicket:id="loginform">

      <table width="100%" cellspacing="1">

            <tr>
                  <td width="100%" class="text11" colspan="3"><div
```

```
wicket:id="feedback"></div></td>
            </tr>

            <tr>
                  <td width="13%" >
                        <wicket:message
      key="login.email">Email</wicket:message>:
                  </td>
                  <td width="82%">
                        <span wicket:id="email.border">
                              <input size="20" maxlength="50"
      wicket:id="email" class="inputfield" />
                        </span>
                  </td>
                  <td width="5%">
                         
                  </td>
            </tr>
            <tr>
                  <td width="13%" >
                        <wicket:message
      key="login.password">Password</wicket:message>:
                  </td>
                  <td width="82%">
                        <span wicket:id="password.border" class=feld>
                              <input size="20" maxlength="50"
      wicket:id="password" class="inputfield" type="password"/>
                        </span>
                  </td>
                  <td width="5%">
                         
                  </td>
            </tr>

            <tr>
                  <td class=text11> </td>
                  <td><input class="button" type="submit"

      wicket:message="value:login.common.button.login" /></td>
                  <td></td>
            </tr>
      </table>
      </form>

      <p>
</wicket:panel>
```

LoginPanel.html sayfasını bir JSP include olarak düşünebiliriz. Yukarda yer alan HTML sayfasını bir panel olarak tanımlayabilmek için `<wicket:panel/>` tagı kullanılmıştır. Bu komponent için gerekli tüm dosyaları **login.jar** dosyasına ekleyerek, tekrar kullanılabilir ve kodu değiştirmeden konfigüre edilebilir bir komponent oluşturmuş oluyoruz. LoginPanel komponentini aşağıda yer alan kod örneğinde görüldüğü gibi herhangi bir WebPage sınıfında kullanabiliriz.

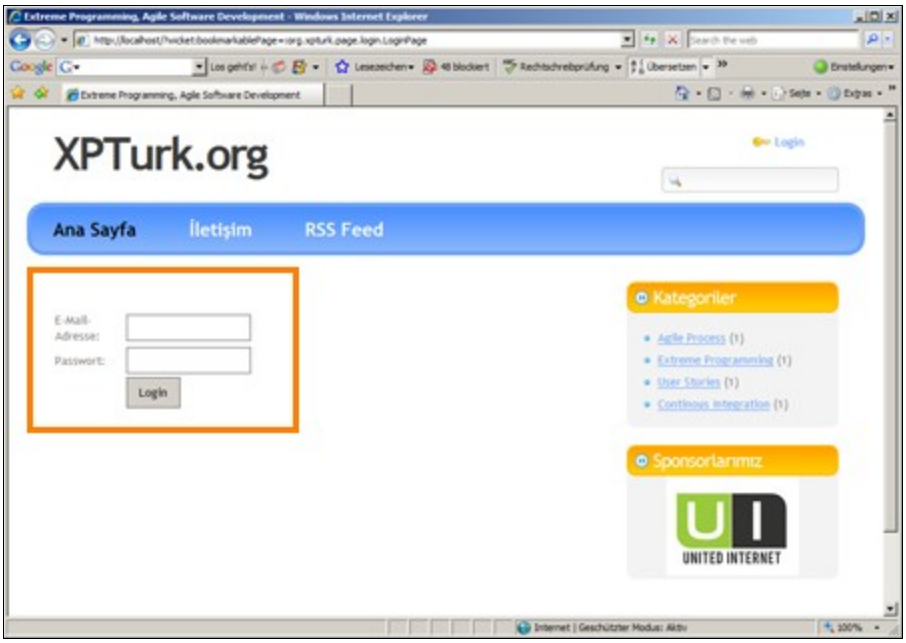
```
package smart.web.login.presentation.page;

import smart.web.login.presentation.panel.LoginPanel;
```

```
import smart.web.wicket.page.SmartWicketPage;

public class LoginPage extends SmartWicketPage
{
	private static final long serialVersionUID = 1L;

	public LoginPage()
	{
	  super();
	}

	public final void buildGui()
	{
		add(new LoginPanel("loginpanel"));
	}
}
```

LoginPage, herhangi bir projede kullanılan bir sayfadır. add() metodunu kullanarak, classpath içinde olan login.jar bünyesindeki LoginPanel sınıfını import ederek kullanabiliyoruz. LoginPage.html aşağıdaki yapıdadır.

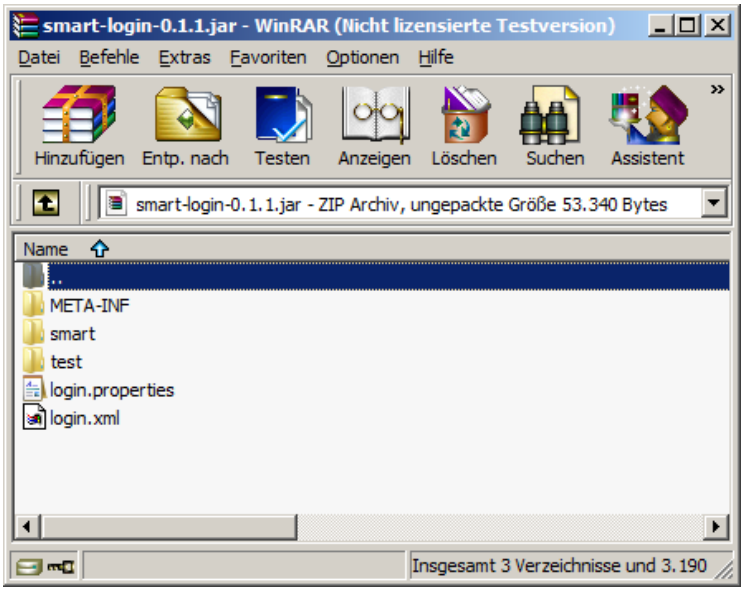
```
<html>

</html>
<body>
<div wicket:id="loginpanel"/>

</body>
```

Görüldüğü gibi Login modülü tam anlamıyla bir komponenttir ve tekrar kullanımı için sadece bir Jar dosyasında olması ve classpath içine eklenmesi yeterli olmuştur. Login komponentini bir sonraki resimde görüldüğü gibi XPTurk.org projesinde kullandım. Kırmızı kare içinde bulunan bölüm LoginPanel'dir.



Resim 20 Login komponentinin kullanıldığı XPTurk.org projesi

Login komponentini XPTurk.org bünyesinde kullanmak için login.jar dosyasını WEB-INF/lib dizinine atmam yeterli oldu. login.jar dosyasının yapısı bir sonraki resimde yer almaktadır.



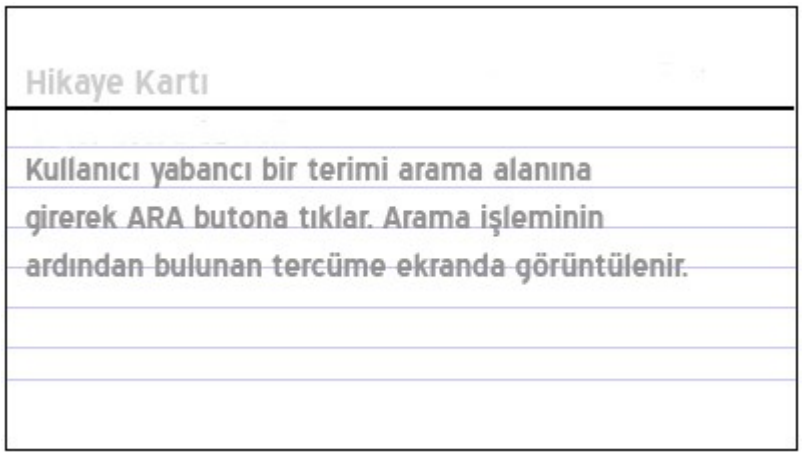
Resim 21 Login.jar

Wicket Panel sınıfı ile tekrar kullanılabilir web komponentleri oluşturmak mümkün hale gelmiştir.

Test Güdümlü Web Projesi Nasıl Yapılır?

Bu sorunun cevabını bu bölümde beraber arayacağız. Test güdümlü yazılım hakkında daha önce kaleme almış olduğum yazıyı http://www.kurumsaljava.com/2008/11/26/test-gudumlu-yazilim-test-driven-development-tdd/ adresinden temin edebilirsiniz.

Bu bölümde **YazilimSozlugu.org** projesi bünyesinde kelime arama işleminin test güdümlü olarak nasıl implemente edilebileceğini yakından inceleyeceğiz. Test güdümlü çalışabilmemiz için öncelikle gereksinimi (requirement) tanımamız ve bu gerensinim için testler oluşturmamız gerekiyor. Bir sonraki resimde arama işlemini tanımlayan kullanıcı hikayesi (user story) yer almaktadır. Extreme Programming[9]'de müşteri tarafından dile getirilen gereksinimler kullanıcı hikayesi (user story) olarak tanımlanır ve hikaye kartlarına (story card) yazılır.

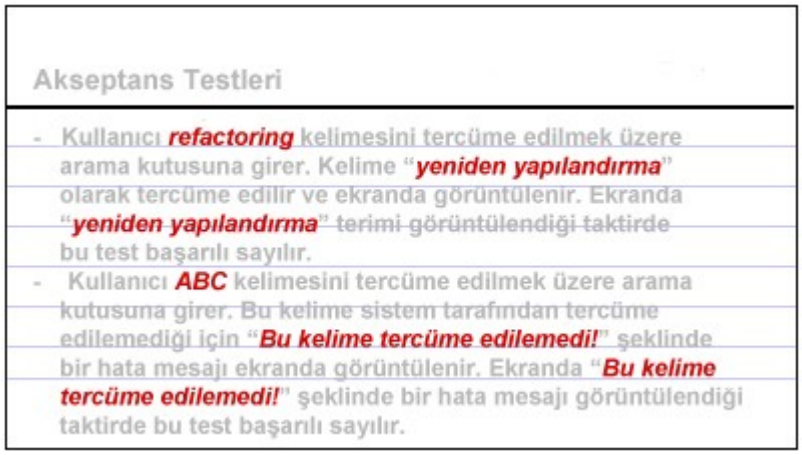


Resim 22 Hikaye kartı

Web tabanlı bir programı test güdümlü geliştirebilmek için akseptans (onay-kabul testleri – acceptance testing[10]) testlerinin oluşturulması gerekmektedir. Akseptans testleri kullanıcı gözüyle sistemi test eden testlerdir. Örneğin bir shop sisteminde müşterinin login işlemini test etmek için akseptans testleri oluşturulur. Bu testler shop sistemini bir kara kutu olarak düşünür ve sadece sistemle interaksiyona girerek, sistemi işlevsel olarak test eder. Örneğin akseptans testlerinden birisi, müşterinin yanlış şifreyi girmesi durumunda sistemin nasıl bir reaksiyon gösterdiğini test edebilir.

Resim 22 de yer alan kullanıcı hikayesinden yola çıkarak aşağıda yer alan akseptans testlerini oluşturabiliriz. Akseptans testlerini hikaye kartının arka tarafına not ediyoruz.

[9] Bakınız: http://www.kurumsaljava.com/category/xp/
[10] Bakınız: http://www.kurumsaljava.com/2008/11/28/unit-testing-konseptleri/



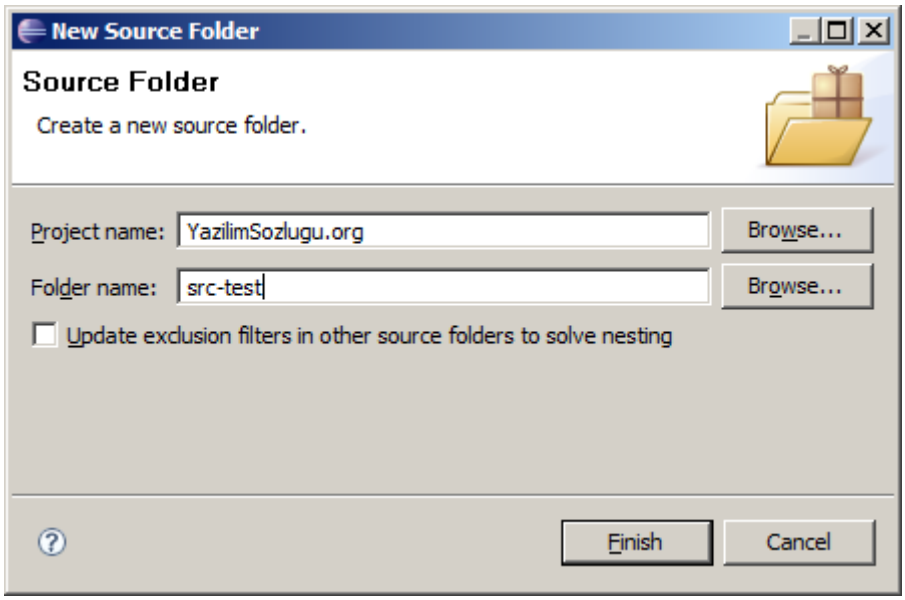
Resim 23 Akseptans testleri

Extreme Programming bünyesinde akseptans testleri müşteri tarafından dile getirilir. Sonuç itibariyle sistemin hangi durumlarda nasıl reaksiyon göstermesi gerektiğini en iyi müşteri bilebilir. Resim 23 de görüldüğü gibi akseptans testleri hikaye kartlarının arka bölümüne not edilir. Müşteri tarafından dile getirilen akseptans testlerini programcılar implemente ederler. Testlerin olumlu sonuç vermesi, tüm sistemin müşterinin istediği şekilde çalıştığı anlamına gelir. Müşteri, olumlu sonuç veren akseptans testleri ile sistemi kabul eder, onaylar. Bu yüzden akseptans testlerine onay – kabul testleridir.

Akseptans testleri için Selenium[11], JWebUnit[12], HttpUnit [13], Canoo WebTest[14], HTMLUnit[15] gibi açık kaynaklı araçlar mevcuttur. Bu araçların hepsini değişik projelerde kullanma fırsatı buldum ve önemli bir dezavantajları olduğunu keşfettim: Bir web aplikasyonunu akseptans test edebilmek için, web aplikasyonun ihtiyaç duyduğu tüm alt yapısı ile Tomcat[16] gibi bir aplikasyon serveri içinde çalışır durumda olması gerekmektedir. Bu durum doğal olarak test otomasyonunu zorlaştırmaktadır. Test güdümlü yazılımın önemli kurallarından birisi de, testlerin otomatik olarak çalıştırılabilir olmalarıdır. Yapılan her değişikliğin ardından testler otomatik olarak çalıştırılabilmelidirler. İsmi geçen akseptans test araçlarını kullanabilmek için web aplikasyonu ve kullandığı bilgibankası gibi diğer alt yapı komponentlerinin çalışır hale getirmemiz gerekiyor ki, bu zaman alıcı ve hata ihtiva edebilen bir işlemdir. Ayrıca bu işlem test otomasyonunu zorlaştırmaktadır.

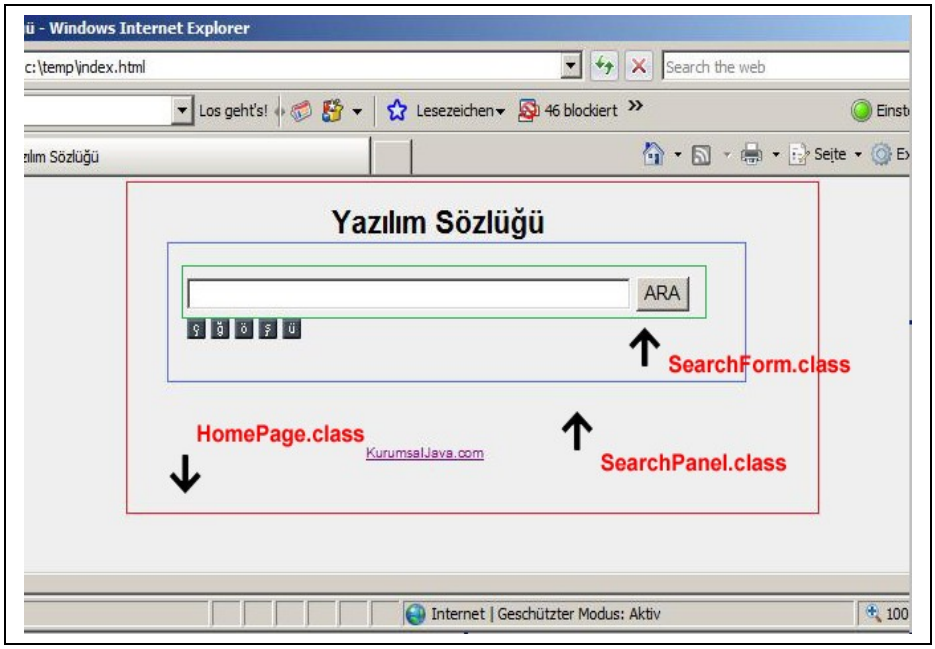
Akseptans testleri için Wicket kendi bünyesinde **WicketTester** ismini taşıyan bir akseptans test aracı barındırmaktadır. WicketTester ile herhangi bir aplikasyon serverine (örn. Tomcat) ihtiyaç kalmadan, JUnit tarzı akseptans testleri oluşturmak mümkündür.

[11] Bakınız: http://seleniumhq.org/
[12] Bakınız: http://jwebunit.sourceforge.net/
[13] Bakınız: http://httpunit.sourceforge.net/
[14] Bakınız: http://webtest.canoo.com/webtest/manual/WebTestHome.html
[15] Bakınız: http://htmlunit.sourceforge.net/
[16] Bakınız: http://tomcat.apache.org/



Resim 24 Test dizini

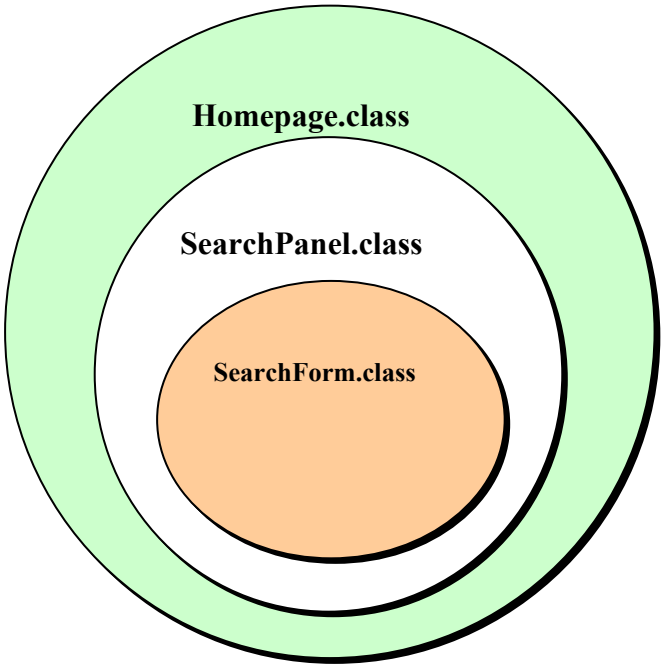
WicketTester ile akseptans testlerini oluşturmadan önce, testlerin yer alacağı src-test (resim 24) isminde yeni bir dizin (source folder) oluşturuyoruz. Akseptans testlerini kolaylaştırmak adına, test edilmek istenen fonksiyonlar için HTML prototiplerin oluşturulmasında fayda vardır. Resim 25 de arama fonksiyonu için oluşturduğumuz HTML prototip yer almaktadır.



Resim 25 Arama sayfasının prototipi

Arama yapılan sayfa üç değişik Wicket komponentinden oluşmaktadır. Ana sayfa olarak tabir edebileceğimiz **HomePage**, **SearchPanel** isminde bir panel ihtiva etmektedir. **SearchPanel** bünyesinde, arama formunu oluşturmak için **SearchForm** komponentini barındırmaktadır.

Akseptans test sınıfını anlayabilmek için komponentlerin bu şekilde birbirleriyle ilişkili olduklarını görmemizde fayda vardır.



Resim 26 Wicket komponent diagramı

Resim 23 de yer alan birinci akseptans testini aşağıdaki şekilde implemente ediyoruz.

```
package test.org.yazilimsozlugu.panel.search;

import junit.framework.TestCase;
import org.apache.wicket.util.tester.FormTester;
import org.apache.wicket.util.tester.WicketTester;

public class SearchPanelTest extends TestCase
{

	/**
	 * Wicket Tester
	 */
	private WicketTester tester;


	/**
	 *
	 * TestCase: Kullanici refactoring kelimesini
	 *           girerek, tercüme edilmesi icin
	 *           ARA butonuna tiklar. Bu kelime
	 *           bilgibankasinda mevcut oldugu icin
	 *           tercümesi bulunarak, ekranda gösterilir.
	 *
	 * INPUT:    Ingilizce refactoring kelimesi
	 *
	 * OUTPUT:   Ekranda yeniden yapilandirma tercümesi
```

```
         *            yer alir.
         */
        public void testWordRefactoringFound()
        {
            // PAGE
1.          tester.startPage(HomePage.class);
2.          tester.assertNoErrorMessage();
3.          tester.assertComponent("searchpanel",
                    SearchPanel.class);

            // PANEL
4.          tester.assertComponent("searchpanel:searchform",
                    SearchForm.class);

            // FORM
5.          FormTester form =
            tester.newFormTester("searchpanel:searchform");

6.          assertNotNull("form  null", form);

7.          form.setValue("word.border:word","refactoring");
8.          form.submit();

9.          tester.assertContains("yeniden yapılandırma");
        }
}
```

Yukarda yer alan test ile resim 23 de ki ilk akseptans testini programlamış olduk. Lakin bu test doğal olarak çalışır durumda değil, çünkü sistemin hiçbir bölümü henüz programlanmadı. Bu testi kısaca bir gözden geçirelim:

1. Aplikasyona giriş noktası olarak **HomePage.class** sınıfını tanımlıyoruz.
2. Oluşturulan ilk sayfada (HomePage) hata mesajlarının olmaması gerektiğini ifade ediyoruz. Doğal olarak ana sayfa ilk kez yüklendiğinde hata mesajlarının olmaması gerekiyor.
3. Yüklenen HomePage sınıfının *searchpanel* (SearchPanel) isminde bir komponenti ihtiva etmesi gerekiyor.
4. HomePage sınıfı ile yüklenen SearchPanel sınıfının *searchform* isminde bir komponenti ihtiva etmesi gerekiyor. Wicket'de birbirlerini ihtiva eden komponentler ***komponent1:komponent2*** seklinde ifade edilir. Bu komponent1'in komponent2'yi ihtiva ettiği anlamına gelir. Web aplikasyonumuzda SearchPanel SearchForm komponentini ihtiva ettiği için ***searchpanel:searchform*** seklinde bir notasyon kullanıyoruz.
5. Formu test etmek için yeni bir **FormTester** oluşturuyoruz.
6. Formun varlığını kontrol ediyoruz.
7. Arama formuna tercüme edilmek üzere *refactoring* kelimesini giriyoruz.
8. Ara butonuna tıklıyoruz ve form gönderiliyor (submit).
9. Oluşan yeni sayfanın *yeniden yapılandırma* kelimelerini ihtiva edip, etmediğini kontrol ediyoruz. Eğer yeni oluşan sayfada *yeniden yapılandırma* kelimeleri mevcutsa, akseptans test başarıyla çalışmış demektir. Bu durumda istenilen özellik sisteme eklenmiştir ve doğru şekilde görevini yerine getirmektedir.

Bu yazının amacı size Wicket frameworkünü tanıtmak. Yazının içeriğini daha fazla şişirmemek için akseptans test yazılımını burada noktalıyorum. Bu konuyu diğer bir yazımda [17] detaylı olarak kaleme aldım.

Özet

Wicket'in özelliklerini ve kullanım avantajlarını şu şekilde sıralayabiliriz:

- **Sadece Java:** Wicket'in sağladığı en büyük avantaj, tüm kodun Java dilinde yazılmasını desteklemesidir. Diğer web frameworklerde olduğu gibi JSP sayfalarını kodlamak için JSTL gibi yeni bir markup dile ihtiyaç kalmamaktadır. Java dilinde kod yazmak hata tespit süresini kısaltmakta ve yazılım sürecini hızlandırmaktadır.
- **Web komponent modeli**: Wicket Swing'den tanıdığımız komponent modeline sahiptir. Web sayfalar değişik komponentler kullanılarak kısa sürede oluşturulabilir. Wicket Panel sınıfı kullanılarak, Wicket komponentlerinden oluşan daha geniş kapsamlı komponentler oluşturmak mümkündür.
- **Kodun tekrar kullanımı**: JSP sayfalarında JSTL tagları ve include direktifi kullanılarak kodun tekrar kullanımı sağlanır. Wicket ile gerçek anlamda Java komponentleri oluşturularak, kodun tekrar kullanımı kolaylaştırılır.
- **JSP sayfaları emekli oldu**: Wicket ile JSP kullanımı son bulmuştur. Wicket bünyesinde HTML sayfaları wicket:id tagı kullanılarak Java koduyla ilişkilendirilir. HTML sayfalar için özel programlama teknikleri kullanılmaz.
- **Ajax desteği**: Wicket bünyesinde bulunan komponentler Ajax desteği sağlamaktadır.
- **Spring entegrasyonu**: Wicket ile Spring çok kolay bir şekilde entegre edilebilmektedir. Wicket ile oluşturulan sayfalarda **@SpringBean** anotasyonu kullanılarak Spring bünyesinde tanımlanan nesnelere erişim kolaylaştırılır.
- **Akseptans testleri**: WicketTester ile bir Wicket aplikasyonu Tomcat gibi bir aplikasyon servere gerek kalmadan akseptans test edilebilir.

Wicket'in komponent modeli ve aplikasyonun tamamen Java dilinde yazılabiliyor olması, web projeleri için Wicket'i kullanmamda etkili oldu. Umarım bu yazı sizin için faydalı olmuştur ve siz de bir Wicket hayranı ve kullanıcısı olmuşsunuzdur.

[17] Bakınız: http://www.kurumsaljava.com/2008/12/24/test-gudumlu-web-projesi-nasil-yapilir/