

xaloon wicket components

Reference Documentation

1.2

Copyright © 2008-2009 Vytautas Racelis

Table of Contents

1. Preface	2
2. Getting started	2
2.1. System requirements	2
2.2. Using Maven	3
2.3. Configuration	4
2.4. Usage scenarios	7
2.4.1 Fully supported xaloon components based web application	7
2.4.2 Provided plugins	7
2.4.3 Using an existing plugin	8
3. Plugins	8
3.1. Plugin administration console	8
3.2. Recaptcha plugin	8
3.3. Comment plugin	9
3.4. File repository plugin	10
3.5. Email plugin	11
3.6. Google verify ownership plugin	13
3.7. Google related plugin	13
3.8. Blogging plugin	14
3.9. Sports plugin	14
4. Other components	16
4.1. VirtualPageFactory	16
4.2. PageScanner	16
5. Custom design	16
6. Writing custom plugin	16
7. References	16

1. Preface

Xaloon components are based on Apache Wicket – component-based java web development framework. It provides web ready components to the intention of minimizing the secondary component of programming and focus on the fundamental problems.

2. Getting started

The following chapter will guide you through the initial steps to start new xaloon based web application or to integrate into existing one. You should start [here](#) if you are not familiar with Apache wicket.

2.1. System requirements

Table 1.1. System requirements

Java Runtime	A JDK or JRE version 5 or greater. You can download a Java Runtime for Windows/Linux/Solaris here .
Hibernate Core	This instructions have been tested against Hibernate 3.3.x. You will need <code>hibernate-core.jar</code> and its transitive dependencies from the <code>lib</code> directory of the distribution.
Hibernate Annotations	The following instructions will use annotations for basic entity configuration (<code>@Entity</code> , <code>@Id</code> , <code>@OneToMany</code> ,...). The tutorial is tested against version 3.4.x of Hibernate Annotations.
Apache Wicket	With proper mark-up/logic separation, a POJO data model, and a refreshing lack of XML, Apache Wicket makes developing web-apps simple and enjoyable again. Swap the boilerplate, complex debugging and brittle code for powerful, reusable components written with plain Java and HTML
Wicket Web Beans	Wicket Web Beans is an Apache Wicket toolkit for JavaBeans. AJAX Web forms are automatically generated from bean properties. The toolkit normally does what you'd expect, but when it doesn't, you can override its behavior.
Brix CMS	Based on Wicket and JCR, it is the best Wicket-based CMS framework available today. BRIX is simple yet powerful and extensible, allowing designers the highest degree of freedom in developing a rich user experience.
Apache jackrabbit	Apache Jackrabbit is a fully conforming implementation of the Content Repository for Java Technology API (JCR). A content repository is a hierarchical content store with support for structured and unstructured content, full text search, versioning, transactions, observation, and more.
Spring framework	Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure so you can focus on your application.
Spring security	Spring Security provides comprehensive security services for J2EE-based enterprise software applications.

Rome	ROME is an set of open source Java tools for parsing, generating and publishing RSS and Atom feeds.
------	---

You can download all dependencies from the xaloon [download site](#).

2.2. Using Maven

Instead of managing all dependencies manually, maven users may use xaloon components repository
Just add the xaloon repository url to the *repositories* section of your `pom.xml` or `settings.xml`:

Example 1.1. Adding the xaloon maven repository to settings.xml

```
<repository>
  <id>repository.xaloon.org</id>
  <name>xaloon Maven Repository</name>
  <url>http://xaloon.googlecode.com/svn/maven2/releases</url>
</repository>
```

Example 1.2. Maven dependencies for xaloon components

```
<dependency>
  <groupId>org.xaloon</groupId>
  <artifactId>xaloon-wicket-components</artifactId>
  <version>${xaloon.version}</version>
</dependency>
```

```
<dependency>
  <groupId>org.xaloon</groupId>
  <artifactId>xaloon-jpa-components</artifactId>
  <version>${xaloon.version}</version>
</dependency>
```

```
<dependency>
  <groupId>org.xaloon</groupId>
  <artifactId>xaloon-wicket-repository</artifactId>
  <version>${xaloon.version}</version>
</dependency>
```

```
<dependency>
  <groupId>org.xaloon</groupId>
  <artifactId>xaloon-wicket-sports-bet</artifactId>
  <version>${xaloon.version}</version>
</dependency>
```

It is not necessary to include other dependencies. You might want to include only `xaloon-wicket-repository` or `xaloon-jpa-components` or both. `xaloon-wicket-components` will be included explicitly then because this component is a backbone for further development. `xaloon-wicket-sports-bet` is only required if you are

going to develop sports based application.

2.3. Configuration

Xaloon components are based on spring framework as IoC and hibernate as JPA based ORM. Spring security is used as security framework. In order to use all features, provided by xaloon components, you should configure web.xml. Sample below will provide JPA based core security, JPA core entity model and interfaces, spring support, file storage repository, blog plugin This configuration and all application startup may be found at xaloon SVN repository [here](#).

Example 1.3. web.xml configuration

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

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath*:META-INF/application-context.xml,
      classpath*:META-INF/application-security.xml,
      classpath*:META-INF/xaloon-jpa-security.xml,
      classpath*:META-INF/xaloon-jpa-core.xml,
      classpath*:META-INF/xaloon-repository-blog.xml,
      classpath*:META-INF/infrastructure-context.xml,
      classpath*:META-INF/entitymanager.xml</param-value>
  </context-param>

  <filter>
    <filter-name>spring.securityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    <init-param>
      <param-name>targetBeanName</param-name>
      <param-value>springSecurityFilterChain</param-value>
    </init-param>
  </filter>

  <filter>
    <filter-name>startup-app</filter-name>
    <filter-class>org.apache.wicket.protocol.http.WicketFilter</filter-class>
    <init-param>
      <param-name>applicationFactoryClassName</param-name>
      <param-value>org.apache.wicket.spring.SpringWebApplicationFactory</param-value>
    </init-param>
    <init-param>
      <param-name>configuration</param-name>
      <param-value>development</param-value>
    </init-param>
  </filter>

  <filter-mapping>
    <filter-name>spring.securityFilterChain</filter-name>
    <url-pattern>*/</url-pattern>
  </filter-mapping>

  <filter-mapping>
    <filter-name>startup-app</filter-name>
    <url-pattern>*/</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>ERROR</dispatcher>
  </filter-mapping>

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

Then you should be able to configure spring transaction manager and enable spring annotations, also to define entity manager factory. This might be done in the same **application-context.xml** or in separate configuration file named **infrastructure-context.xml**

Example 1.4. infrastructure-context.xml configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd"
       default-autowire="byType">

    <!-- infrastructure beans -->
    <bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor"
          lazy-init="true" />

    <bean id="customPostProcessor" class="org.xaloon.wicket.component.util.EntityProvidingPUPostProcessor">
    </bean>

    <bean class="org.springframework.orm.jpa.support.PersistenceAnnotationBeanPostProcessor" />
    <bean name="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
        <property name="entityManagerFactory" ref="entityManagerFactory" />
    </bean>

    <bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
          lazy-init="true">
        <property name="persistenceXmlLocation" value="classpath:META-INF/persistence.xml" />
        <property name="persistenceUnitName" value="startup-app" />
        <property name="persistenceUnitPostProcessors">
            <list>
                <ref bean="customPostProcessor" />
            </list>
        </property>
    </bean>

    <tx:annotation-driven />
</beans>
```

application-security.xml by default contains security configuration. Xaloon components use database based security, but it might be easily replaced by LDAP or JAAS based security.

Example 1.5. application-security.xml configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<!--!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN" "http://www.springframework.org/dtd/spring-beans-2.0.dtd"-->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:security="http://www.springframework.org/schema/security"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:jee="http://www.springframework.org/schema/jee"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-2.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-2.0.4.xsd"
       default-autowire="byType">

    <!-- SPRING SECURITY -->
    <!-- HTTP filter chain (resources security) settings -->
    <security:http auto-config="false" access-denied-page="/access-denied" access-decision-manager-
ref="accessDecisionManager">
```

```

        <security:intercept-url pattern="/resources/**" access="IS_AUTHENTICATED_ANONYMOUSLY" />
        <security:intercept-url pattern="/**" filters="none" />
        <security:intercept-url pattern="/admin/**" access="/SYSTEM/ADMIN" />
        <security:intercept-url pattern="/client/**" access="/SYSTEM/LOGIN" />
        <security:intercept-url pattern="/personal/**" access="/SYSTEM/LOGIN" />
        <security:form-login login-page="/login" default-target-url="/index" authentication-failure-
url="/login" />
        <security:logout logout-success-url="/index" />
        <security:anonymous username="anonymous" granted-authority="ROLE_ANONYMOUS" />
    </security:http>

    <bean id="daoAuthProvider"
class="org.xaloon.wicket.component.security.XaloonDaoAuthenticationProvider">
        <security:custom-authentication-provider />
        <property name="userService">
            <bean class="org.xaloon.wicket.component.security.UserDetailsServiceImpl"></bean>
        </property>
    </bean>
</beans>

```

application-context.xml by default contains application based bean definitions. It also contains default bean properties, such as menu configuration, jackrabbit configuration, page scanners, etc.

Example 1.5. application-context.xml configuration

```

<?xml version="1.0" encoding="UTF-8"?>
<!--!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN 2.0//EN" "http://www.springframework.org/dtd/spring-beans-2.0.dtd"-->

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:security="http://www.springframework.org/schema/security"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:jee="http://www.springframework.org/schema/jee"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-2.0.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context-2.5.xsd
http://www.springframework.org/schema/tx
http://www.springframework.org/schema/tx/spring-tx.xsd
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-2.0.4.xsd"
    default-autowire="byType">

    <!-- WICKET application -->
    <bean id="application" class="org.xaloon.wicket.application.StartupApplication"/>

    <!-- PAGE scanner to mount pages -->
    <bean id="startupPageScanner" class="org.xaloon.wicket.component.mounting.PageScanner">
        <property name="packageName" value="org.xaloon.wicket.application.page"/>
    </bean>

    <!-- STORAGE repository properties -->
    <bean id="cityContentProperties" class="org.xaloon.wicket.component.repository.util.ContentProperties">
        <property name="jcrRepository" value="/usr/local/jackrabbit/startup-app"/>
        <property name="jcrUsername" value="username"/>
        <property name="jcrPassword" value="password"/>
    </bean>
    <bean class="org.xaloon.wicket.component.repository.storage.ContentSessionFacadeImpl"/>

    <!-- MENU properties -->
    <bean id="menuProperties" class="org.xaloon.wicket.component.basic.MenuProperties">
        <property name="useConditionalMenu" value="true"/>
        <property name="showMenuGroupTitle" value="false"/>
    </bean>

    <!-- CUSTOMER BEAN PROPERTIES. Required if xaloon-jpa-core.xml is used -->
    <bean id="customerFacade" class="org.xaloon.wicket.component.customer.impl.CustomerFacadeImpl">
        <property name="generateDefaultAccount" value="true"/>
    </bean>
</beans>

```

2.4. Usage scenarios

2.4.1 Fully supported xaloon components based web application

In order to use fully supported xaloon components based web application, you should checkout [startup application](#) and modify under your requirements. Such application will contain these built-in features:

- ✓ Plugin support;
- ✓ File storage repository;
- ✓ Security;
- ✓ Customer related domain model;
- ✓ Registration, activation & login related components;
- ✓ Google analytics component;
- ✓ Design independent - inject components into existing design **WebPage**;
- ✓ Single annotation on **WebPage** provides several features:
 - ✓ Combining menu items into groups;
 - ✓ Disable menu group when plugin is disabled;
 - ✓ Generating SEO friendly url;
 - ✓ Dynamic menu based on security;
 - ✓ Dynamic **sitemap.xml**

Default startup application contains these plugins:

- ✓ [Blogging plugin](#);
- ✓ [Comment plugin](#);
- ✓ [Email plugin](#);
- ✓ [File repository plugin](#);
- ✓ [Google related plugin](#);
- ✓ [Recaptcha plugin](#);
- ✓ [Google verify ownership plugin](#).

2.4.2 Provided plugins

Xaloon components currently provide such plugins:

- ✓ [Blogging plugin](#);
- ✓ [Sports plugin](#);

- ✓ [Recaptcha plugin](#);
- ✓ [Comment plugin](#);
- ✓ [File repository plugin](#);
- ✓ [Email plugin](#);
- ✓ [Google verify ownership plugin](#);
- ✓ [Plugin administration console](#);
- ✓ [Google related plugin](#).

2.4.3 Using an existing plugin

To use an existing xaloon plugin spring configuration file should be included into **web.xml** definition file, e.g.,

Example 1.6. web.xml additional configuration

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>classpath*:META-INF/application-context.xml,
    classpath*:META-INF/application-security.xml,
    classpath*:META-INF/xaloon-jpa-security.xml,
    classpath*:META-INF/xaloon-mail.xml, <!-- this will include sending emails plugin -->
    classpath*:META-INF/xaloon-google.xml, <!-- this will include google analytics plugin -->
    classpath*:META-INF/xaloon-repository-blog.xml,
    classpath*:META-INF/infrastructure-context.xml</param-value>
</context-param>
```

3. Plugins

3.1. Plugin administration console

Plugins are registered when application is started up and initially they are disabled, except administration plugin and empty plugin. Administration page may found at <http://localhost:<PORT>/<CONTEXT>/admin/plugin-management.wspix>. Plugins are stored into jackrabbit repository by default.

3.2. Recaptcha plugin

Recapthca plugin implements captcha service provided by <http://recaptcha.net/>. You must obtain recaptcha private and public keys from recaptcha.net in order to use this plugin.

Recaptcha plugin



Recaptcha Plugin Bean

Save

Private Key

Public Key

Configuration:

include **xaloon-captcha.xml** file into spring context;

Usage:

Recaptcha panel is added into a form and is visible if recaptcha plugin is enabled.

```
form.add(new ReCaptchaPanel("rc", form));
```

3.3. Comment plugin

Comment plugin is used to add comment support for concrete page, blog, sport event or other entity model which implements **org.xaloon.wicket.component.comment.Commentable** interface. `getId()` identifies unique id of **Commentable** object, usually it is the same as model entity **@Id** column. `getComponent()` identifies unique group of **Commentable** object, e.g., blog group, photo group, sport event, etc.

Properties:

- Anonymous – allow to post comments anonymously;
- Apply by administrator – whether comments will be applied by administrator or not;
- Email visible – it is possible to enter email or not. If email column is visible and “Send email” is true then email will be sent to system administrator and it will be possible to reply to this email;
- Provider all – if true then all comments will be shown;
- Send email – if true then email will be sent to system administrator with comment body;
- Website visible – if true then it will be possible to enter web site url when creating new comment;
- Page size – how many comments it is visible per page.

Comment plugin

Comment Plugin Bean

Save

Anonymous <input checked="" type="checkbox"/>	Apply By Administrator <input checked="" type="checkbox"/>	Email Visible <input checked="" type="checkbox"/>
Page Size <input type="text" value="10"/>	Provider All <input checked="" type="checkbox"/>	Send Email <input checked="" type="checkbox"/>
Website Visible <input checked="" type="checkbox"/>		

Configuration:

include **xaloon-jpa-comment.xml** file into spring context;

Usage:

```
@SpringBean
private CommentPlugin commentPlugin;
...
commentPlugin.getFacade().createCommentPanel(id, model, authorId);
//model – implementation of Commentable interface;
//authorId – who wrote the comment. -1 – anonymously, otherwise customer id;
```

3.4. File repository plugin

File repository plugin provides file storage and retrieval feature based on Apache jackrabbit JCR functionality.

Usage to store image into repository:

```
private static final String IMAGE_ROOT_PATH = "images";
@SpringBean
private FileRepositoryPlugin fileRepositoryPlugin;
...
String imagePath = IMAGE_ROOT_PATH + "/" + upload.getClientFileName();
fileRepositoryPlugin.getFacade().storeFile(IMAGE_ROOT_PATH,
upload.getClientFileName(), "images/jpeg", input.getInputStream());
```

Usage to show image from repository:

```
add(new ImageLink("image", imagePath));
```

3.5. Email plugin

Email plugin is used to configure email service when sending emails to customers or system administrator. If plugin is disabled then emails cannot be sent.

Properties:

- Charset – email encoding, default **UTF-8**;
- Debug – email information will be printed into log file if debug is checked;
- From email – email will be shown as “From“. This email will be shown to customer ;
- From title – title will be shown as “From”. This title will be shown to customer;
- To email – System administrator's email;
- To title – Title, which will be shown near system administrator's email;
- Subject – Default subject of email when email is sent to system administrator;
- Hostname – SMTP host
- Port – SMTP port
- Start TTLS – check true if you are sending email via gmail;
- Required authentication – does SMTP host requires authentication;
- Username – username of SMTP account if required;
- Password – password of SMTP account if required.

Email plugin ✔

Email Template Save

Charset <input type="text" value="UTF-8"/>	Debug <input type="checkbox"/>	From Email <input type="text"/>
From Title <input type="text"/>	Host Name <input type="text"/>	Password <input type="text"/>
Port <input type="text" value="25"/>	Required Auth <input type="checkbox"/>	Start TTLS <input type="checkbox"/>
Subject <input type="text"/>	To Email <input type="text"/>	To Title <input type="text"/>
Username <input type="text"/>		

Configuration:

include **xaloon-mail.xml** file into spring context;

Usage:

You may use templates to render message body. `EmailContentTemplatePage` class is abstract class for email templates and it provides `getSource()` method to get rendered email body.

```

EmailContentTemplatePage contentTemplate = new
RegistrationEmailTemplatePage(customer.getUsername(), password, activationKey);
String content = contentTemplate.getSource ();
    if (emailPlugin.isEnabled()) {
        emailFacade.sendEmailTo(emailPlugin.getPluginBean(), content,
contentTemplate.getSubject(), customer.getEmail(), null);
    } else {
        throw new RuntimeException("Message was not sent due to system
configuration restrictions!");
    }

```

3.6. Google verify ownership plugin

Google verify ownership plugin is not used everyday. Sometimes google webmaster tool requires to verify your site by creating new page or modifying page head adding additional tag. This plugin allows you to add verification meta tag without modifying html code. Once site is verified you may disable this plugin and meta tag will be removed from html code.

Google webmaster plugin <input checked="" type="checkbox"/>	
Meta Tag Plugin Bean	<input type="button" value="Save"/>
Key	Value
<input type="text" value="google-site-verification"/>	<input type="text" value="sertwert"/>

Configuration:

include **xaloon-core.xml** file into spring context;

Usage:

Plugin is used when VirtualPageFactory is enabled. For custom usage, please refer to `org.xaloon.wicket.component.page.AbstractWebPage` class.

3.7. Google related plugin

There is google analytics support as google plugin currently.

Properties:

- Analytics tracker id – personal tracker id;
- Asynchronous – use asynchronous data gathering;

Google related plugin <input checked="" type="checkbox"/>	
Google Plugin Bean	<input type="button" value="Save"/>
Analytics Tracker Id	Asynchronous
<input type="text" value="20"/>	<input checked="" type="checkbox"/>

Configuration:

include **xaloon-google.xml** file into spring context;

Usage:

Add this code to parent template of your application. Generated source will be visible only if plugin is enabled.

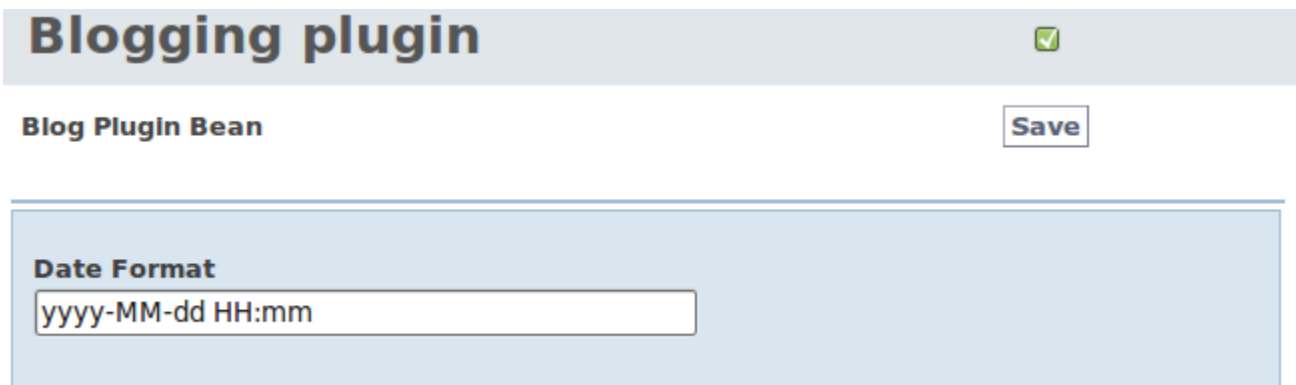
```
add(new GoogleAnalyticsPanel("analytics"));
```

3.8. Blogging plugin

Blogging plugin allows to create new blog entry, edit current, delete selected, show all blog entries. Blog entries are stored into jackrabbit repository currently. There is also JPA implementation of BlogFacade, but it is not supported yet.

Properties:

- Date format – how date will be displayed for readers;



Blogging plugin

Blog Plugin Bean

Date Format

Configuration:

include **xaloon-repository-blog.xml** file into spring context;

Usage:

/blog/list, **/blog/entry**, **/personal/new-blog-entry** pages are available by default

3.9. Sports plugin

Sports plugin is a dedicated business component for sport events management. Plugin contains such features:

- ✓ Customer features
 - ✓ Dashboard : last minute events, today's highlights, latest results, latest playing card list
 - ✓ Show my playing cards
 - ✓ Show my shared playing cards
 - ✓ Sports result panel : view event results by selected date, sport type, league

- ✓ Administrator features
 - ✓ Sports management panel : create different sport types, leagues, groups, teams and players
 - ✓ Sports type management panel : create various bet types for selected sport type
 - ✓ Event management panel
 - ✓ create new events
 - ✓ disable/enable/edit selected event
 - ✓ create child event for selected parent
 - ✓ Result management panel
 - ✓ enter results for finished events
- ✓ Event features
 - ✓ Show new events
 - ✓ Show events for today
 - ✓ Sports calendar
 - ✓ Today's highlights
 - ✓ Events by sport type, league
 - ✓ Vote for event
 - ✓ Comment selected event
- ✓ Playing cards features
 - ✓ Share playing card : if card is shared - other customer may see your selection
 - ✓ Comment shared playing card
 - ✓ Vote for shared playing card

Requirements:

Apache ActiveMQ is required in order to use this component. Ports and other connection information is configured using **xaloon-sports-jms.xml** file.

Configuration:

include **xaloon-sports.xml**, **xaloon-sports-jms.xml** files into spring context;

Usage:

Additional menu items will appear in menu panel when plugin is enabled.

4. Other components

4.1. VirtualPageFactory

`org.xaloon.wicket.component.application.VirtualPageFactory` is useful when you have several templates or custom already provided template. Then you should just create empty `WebPage`, add `@MountPage`, `@MountPanel` and `@MountPageGroup` annotations. Panel, provided via `@MountPanel` annotation, will be injected into existing layout page. This layout class should be defined in `WebApplication` class. More for configuration you should refer to [sample application](#).

4.2. PageScanner

Page scanner is used to scan `@MountPage` and `@MountPageGroup` annotations and add selected pages into menu, sitemap, mounting url.

```
<bean id="samplePageScanner" class="org.xaloon.wicket.component.mounting.PageScanner">  
    <property name="packageName" value="org.xaloon.wicket.component.blog.page"/>  
</bean>
```

5. Custom design

You may create your design just overriding existing package with html files.

6. Writing custom plugin

You may write your custom plugin by extending `org.xaloon.wicket.component.plugin.AbstractPlugin<T, K>`

T – should be plugin business facade;

K – plugin bean, containing specific properties for created bean;

New bean should be used by your panel and should be visible only if plugin is enabled.

You should add page scanner if you have created some pages for your plugin and want to include these pages into menu items, e.g.,

```
<bean id="samplePageScanner"  
class="org.xaloon.wicket.component.mounting.PageScanner">  
    <property name="packageName" value="org.xaloon.wicket.component.blog.page"/>  
</bean>
```

7. Brix CMS support

Currently there are some plugins prepared for usage in Brix CMS environment:

- ✓ Blogging plugin;
- ✓ Login panel;
- ✓ Google analytics panel;
- ✓ Leave a message for system administrator;
- ✓ Plugin management console.

You should include maven dependency in order to use provided features:

```
<dependency>  
  <groupId>org.xaloon</groupId>  
  <artifactId>xaloon-wicket-brix</artifactId>  
  <version>${xaloon.version}</version>  
</dependency>
```