**Web Application Security Made Easy With JBoss, Seam, and Hibernate**

Chris Anderson

School of Engineering, University of Colorado at Colorado Springs

Colorado Springs, Colorado

**Abstract**

The Internet is responsible for making data easily accessible on demand. The vast amount of data accessible on the Internet is beneficial for gaining knowledge and increased productivity; however, not all data should be open to the public. For this reason, companies have implemented secure web applications to protect their data from unauthorized people. The purpose of this paper is to investigate current technologies that can be utilized to quickly and easily develop a secure web application.

**1. Introduction**

The availability of information on the Internet has made it a breeding ground for hackers. Web developers that are in-experienced in security or that use outdated tools, make their web sites susceptible to attacks. This paper details instructions on how to quickly setup up a secure web application that is protected against cross-site scripting attacks, SQL injection attacks, and has role-based authentication.

**2. Technologies**

The technologies used in this paper were chosen based on the needs of my company. SQL Server was chosen as the database to utilize existing licenses. JBoss application server was chosen because it is open source and it has built in support for detecting and preventing cross-side scripting attacks. The Seam web framework is a requirement of my workplace because it is open source, it is supported by the JBoss server, and it has a built in workflow manager. The greatest benefit of these technologies; however, is that they make it simple to implement security measures out of the box.

**2.1 Microsoft SQL Server 2005**

Microsoft SQL Server is a relational database management system. It has the benefit over other database systems in that the graphical interface makes it simple to implement security for in-experienced developers. Also, when installing SQL Server, you are prompted to specify a password for the system administrator (sa) account. Other database systems do not require the default sa password to be changed, which is a security concern. For the demo presented in this paper, SQL Server 2005 express edition was used because it is free for evaluation and educational purposes.

**2.2 JBoss 4.2.0 Application Server**

JBoss is a free, open source Java application server used for hosting J2EE web applications. JBoss offers services such as clustering, persistence, Enterprise Java beans (EJBs), and caching [1]. From a security perspective, JBoss is able to detect cross-side scripting attacks and it makes it simple to enable the secure socket layer protocol.

**2.3 JBoss Seam 2.0.2 Web Framework**

JBoss Seam is a web framework that makes it simple for web developers to create Java web applications that utilize Asynchronous JavaScript and XML(AJAX), EJBs, Java Server Faces (JSF), and Java Business Process Management (JBPM). Seam also has built-in support for user role validation [2].

**2.4 Hibernate**

Hibernate is built into Seam and is a Java object relational mapping tool. It allows you, as the developer, to treat a database table record as an object and call methods to retrieve that records attributes [3]. If used correctly, Hibernate will prevent the occurrence of SQL injections attacks. This is done by using parameterized queries rather than queries that are concatenated based on user input.

**2.5 Apache Ant**

Seam is intended to be built and deployed using Apache Ant. After creating a Seam application, there exists a build.xml file that is used by Ant to compile the application source files to the application server directory.

**2.6 Eclipse**

Eclipse is an integrated development environment used to develop software applications. In this case, I used the JBoss Tools version of Eclipse, which comes equipped with tools for generating EJBs from a database schema or vice versa. A simple text editor can be used to write code; however, Eclipse provides a number of conventions that makes development much easier.

**3. Prerequisites**

Prior to beginning work on this secure web application demo, I already had JBoss EP 4.2.0, SQL Server 2005 express edition, Apache Ant, JBoss Seam 2.0.2, and the Java 1.6 JDK installed. The JAVA\_HOME environment variable was set to the bin directory of the Java installation and the Ant bin directory was added to the path environment variable.

**4. Database Creation**

The database for the secure web application demo was created with the following scripts:

-- RUN USING: SQLCMD -S LAPTOP\TESTDB -U sa -i C:\projects\test\database\scripts\createdatabase.sql

USE msdb;

GO

PRINT '\*\*\*\* DROPPING USERS AND LOGINS \*\*\*\*';

DROP LOGIN TESTDB;

DROP LOGIN TESTDBUSER;

DROP LOGIN JBOSS2;

PRINT '\*\*\*\* DROPPING DATABASE \*\*\*\*';

DROP DATABASE TESTDB;

PRINT '\*\*\*\* CREATING DATABASE \*\*\*\*';

CREATE DATABASE TESTDB

ON

( NAME = TESTDB\_DAT,

FILENAME = 'C:\DATA\TESTDATABASE\TESTDB\_DATA.MDF',

SIZE = 100MB,

MAXSIZE = 150MB,

FILEGROWTH = 25MB

)

LOG ON

( NAME = TESTDB\_LOG,

FILENAME = 'C:\DATA\TESTDATABASE\TESTDB\_LOG.LDF',

SIZE = 50MB,

MAXSIZE = 75MB,

FILEGROWTH = 15MB

);

GO

CREATE LOGIN TESTDB WITH PASSWORD = 'dsctestdb', DEFAULT\_DATABASE = TESTDB;

USE TESTDB;

CREATE USER TESTDB FOR LOGIN TESTDB;

GO

CREATE LOGIN TESTDBUSER WITH PASSWORD = 'testdbuser', DEFAULT\_DATABASE = TESTDB;

USE TESTDB;

CREATE USER TESTDBUSER FOR LOGIN TESTDBUSER;

GO

CREATE LOGIN JBOSS2 WITH PASSWORD = 'dscjboss2', DEFAULT\_DATABASE = TESTDB;

USE TESTDB;

CREATE USER JBOSS2 FOR LOGIN JBOSS2

GO

CREATE SCHEMA TESTDB AUTHORIZATION TESTDB;

GO

CREATE SCHEMA JBOSS2 AUTHORIZATION JBOSS2;

GO

ALTER USER JBOSS2 WITH DEFAULT\_SCHEMA = JBOSS2;

ALTER USER TESTDB WITH DEFAULT\_SCHEMA = TESTDB;

ALTER USER TESTDBUSER WITH DEFAULT\_SCHEMA = TESTDB;

USE TESTDB;

GRANT CONNECT, CREATE TABLE, DELETE, UPDATE, INSERT, SELECT TO TESTDB;

GRANT CONNECT, SELECT, INSERT, UPDATE, DELETE TO TESTDBUSER;

GRANT CONNECT, CREATE TABLE, SELECT, INSERT, UPDATE, DELETE TO JBOSS2;

GO

**4.1 Database Users**

In the above example, three database users are created. The TESTDB user is established for the developers, the TESTDBUSER user is needed for the application connection, and the JBOSS2 user is needed to create specific tables for the application.

**4.2 Tables**

This application implements basic security in which the authenticated user has access to one or more application roles. There are three database tables to implement the role-based security.

**4.2.1 USER**

The user table stores information specific to the user including personal information, username, and password. The password is encrypted using the SHA-1 hashing algorithm. The following is the code for creating the user table in SQL Server:

CREATE TABLE [TESTDB].[TEST\_USER](

[USER\_ID] [numeric](10, 0) IDENTITY(1,1) NOT NULL,

[USERNAME] [varchar](50) NOT NULL,

[FIRST\_NAME] [varchar](50) NOT NULL,

[LAST\_NAME] [varchar](50) NOT NULL,

[EMAIL] [varchar](100) NOT NULL,

[PHONE] [varchar](13) NULL,

[PASSWORD] [varchar](100) NOT NULL,

[DEFAULT\_SITE] [numeric](10, 0) NULL,

[CREATED\_BY] [varchar](50) NULL,

[UPDATED\_BY] [varchar](50) NULL,

[CREATED\_DATE] [datetime] NULL,

[UPDATED\_DATE] [datetime] NULL,

CONSTRAINT [PK\_TEST\_USER] PRIMARY KEY CLUSTERED

**4.2.2 ROLE**

The role table stores the different roles that are available for the application. I created two different user roles: admin and read-only. The following is the code for creating the application roles:

CREATE TABLE [TESTDB].[ROLE](

[ROLE\_ID] [numeric](10, 0) IDENTITY(1,1) NOT NULL,

[ROLE\_NAME] [varchar](100) NOT NULL,

[ROLE\_DESCRIPTION] [varchar](500) NOT NULL,

[CREATED\_BY] [varchar](50) NULL,

[UPDATED\_BY] [varchar](50) NULL,

[CREATED\_DATE] [datetime] NULL,

[UPDATED\_DATE] [datetime] NULL,

CONSTRAINT [PK\_ROLE] PRIMARY KEY CLUSTERED

**4.2.3 USER\_ROLE**

The USER\_ROLE table is used to map users to roles. In the following example, I demonstrate how to link the USER and ROLE tables:

CREATE TABLE [TESTDB].[USER\_ROLES](

[USER\_ROLE\_ID] [numeric](10, 0) IDENTITY(1,1) NOT NULL,

[USER\_ID] [numeric](10, 0) NOT NULL,

[ROLE\_ID] [numeric](10, 0) NOT NULL,

[CREATED\_BY] [varchar](50) NULL,

[UPDATED\_BY] [varchar](50) NULL,

[CREATED\_DATE] [datetime] NULL,

[UPDATED\_DATE] [datetime] NULL,

CONSTRAINT [PK\_USER\_ROLES] PRIMARY KEY CLUSTERED

**4.3 Stored Procedures**

In the database, user passwords are stored in the SHA-1 hashing algorithm. This is done so that database administrators cannot see user passwords.

**4.3.1 Create User**

A stored procedure is created to create new users. The stored procedure adds all of the new user info and assigns them a default password that must be changed on the first login.

**4.3.2 Change Password**

Another stored procedure is provided to the web designer to regulate the traffic going through the web site. When customers forget their password, we provide a method in SQL Server to reset their password. Upon logging in again, their password will be different but the update date will be different, indicating that the user account has been tampered with.

**5. Application Generation**

The Seam Setup command is used to create a new Seam application:

C:\jboss-seam-2.0.2.SP1>seam setup

[input] Enter your Java project workspace (the directory that contains your

Seam projects) [c:/Projects]

[input] Enter your JBoss home directory [C:/jboss/jbossEP-4.2.0.GA/jboss-as]

[input] Enter the project name [testproject]

[input] Do you want to use ICEFaces instead of RichFaces [n] (y, [n])

[input] Select a RichFaces skin [classic] (blueSky, [classic], ruby, wine, d

eepMarine, emeraldTown, sakura, DEFAULT)

[input] Is this project deployed as an EAR (with EJB components) or a WAR (w

ith no EJB support) [ear] ([ear], war)

[input] Enter the Java package name for your session beans [com.uccs.itapps.

testproject.beans.session]

[input] Enter the Java package name for your entity beans [com.uccs.itapps.t

estproject.beans.entity] [

[input] Enter the Java package name for your test cases [com.uccs.itapps.tes

tproject.testcases]

[input] What kind of database are you using? [mssql] (hsql, mysql, oracle, p

ostgres, [mssql], db2, sybase, enterprisedb, h2)

[input] Enter the Hibernate dialect for your database [org.hibernate.dialect

.SQLServerDialect]

[input] Enter the filesystem path to the JDBC driver jar [C:\Program Files\M

icrosort SQL Server 2005 JDBC Driver\sqljdbc\_1.2\enu\sqljdbc.jar]

[input] Enter JDBC driver class for your database [com.microsoft.sqlserver.j

dbc.SQLServerDriver]

[input] Enter database username [testdbuser]

[input] Enter database password [testdbuser]

[input] Enter the database schema name (it is OK to leave this blank) [TESTDB]

[input] Enter the database catalog name (it is OK to leave this blank) []

[input] Are you working with tables that already exist in the database? [y] ([y], n)

[input] Do you want to drop and recreate the database tables and data in imp

ort.sql each time you deploy? [n] (y, [n])

C:\jboss-seam-2.0.2.SP1>seam new-project

**6. Application Configuration**

Additional configuration is needed for the datasource.xml file. The port number of the database instance must be specified along with the name of the database.

<connection-url>

jdbc:sqlserver://127.0.0.1:50853;databaseName=TESTDB

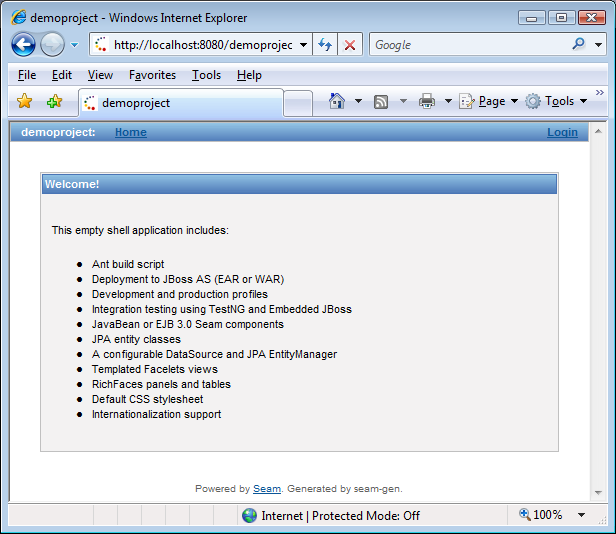
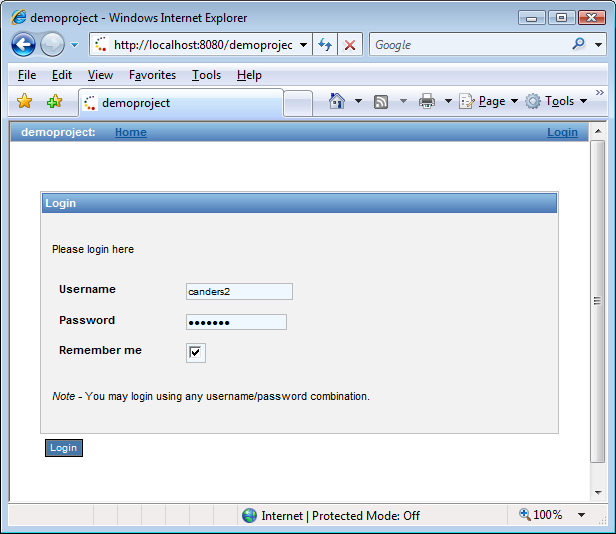
</connection-url>

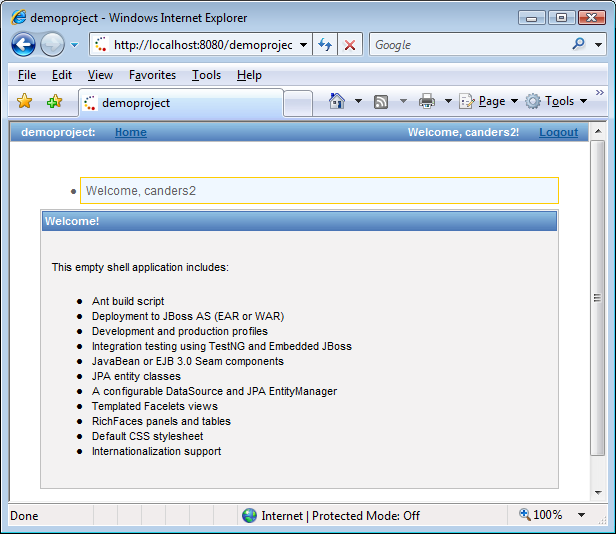
**7. Testing the Skeleton Application**

The application can be built and deployed using the command:

C:\Projects\demoproject>ant deploy

The JBoss application server can be started using the command:

C:\jboss\jbossEP-4.2.0.GA2\jboss-as\bin\run.bat -c default 



**8. Additional Security Measures**

**8.1 User Authentication**

In order to validate the username/password information, we must modify the Authentication bean:

User user = (User) em.createQuery("from User where username = :username and password = :password")

.setParameter("username", identity.getUsername())

.setParameter("password", getHashedPwd(identity.getPassword()))

.getSingleResult();

if(user.getRoles() != null){

for(Role mr : user.getRoles()){

System.out.println("adding role: " + mr.getRoleName());

identity.addRole(mr.getRoleName());

}

}

**8.2 Role-Based Security**

To enable role-based security, we need to create a new administration page:

<!DOCTYPE composition PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<ui:composition xmlns="http://www.w3.org/1999/xhtml"

xmlns:s="http://jboss.com/products/seam/taglib"

xmlns:ui="http://java.sun.com/jsf/facelets"

xmlns:f="http://java.sun.com/jsf/core"

xmlns:h="http://java.sun.com/jsf/html"

xmlns:rich="http://richfaces.org/rich"

template="layout/template.xhtml">

<ui:define name="body">

<h:messages globalOnly="true" styleClass="message"/>

<h:outputText value="This page is for site admins only!"/>

</ui:define>

</ui:composition>

The next step is to add a link to the administration page on the menu:

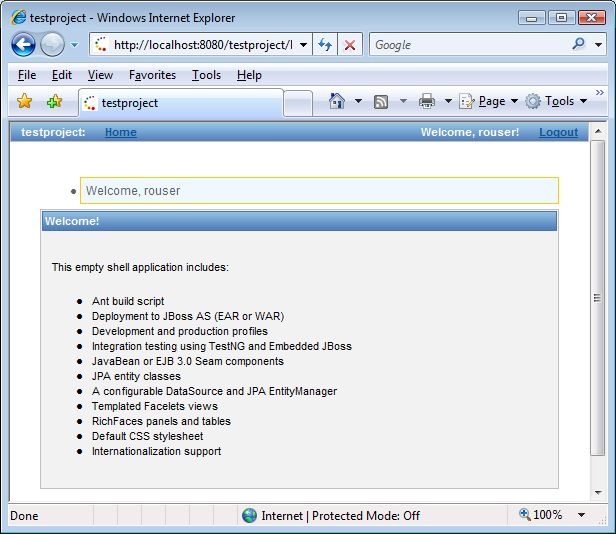
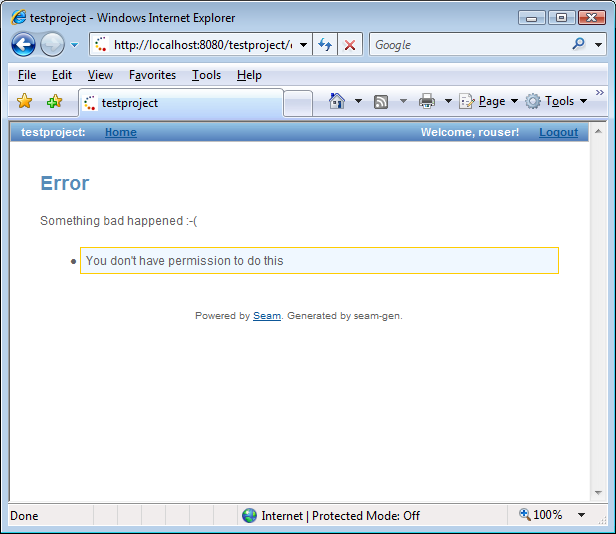
<s:link view="/admin.xhtml" action="administration" value="Administration" rendered="#{identity.loggedIn &amp;&amp; s:hasRole('ADMIN')}"/>

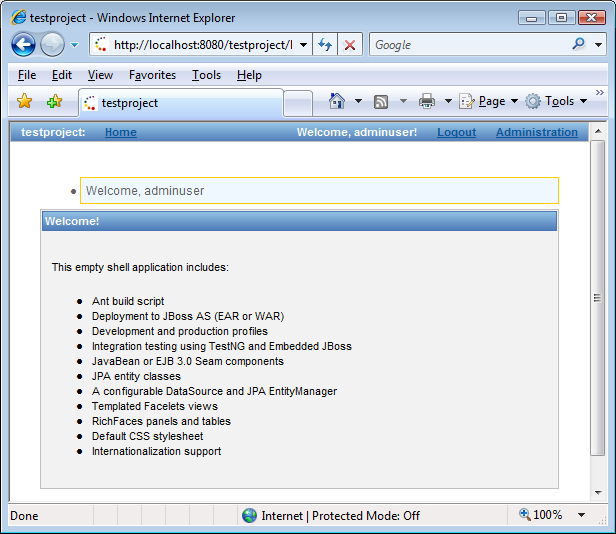
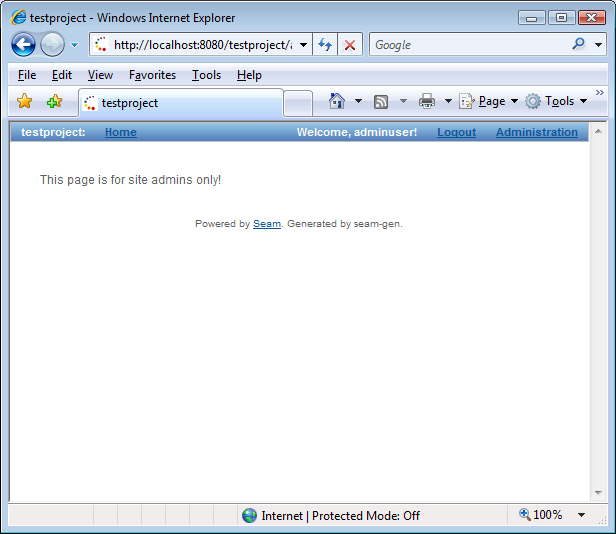
The link to the administration page is only displayed for logged in users who have the ADMIN role. The next step is to edit the pages.xml file. This file states how to map requests to different pages. The following rule was added to the pages.xml file to restrict the admin page to logged in administrators:

<page view-id="/admin.xhtml" login-required="true">

<restrict>#{s:hasRole('ADMIN')}</restrict>

</page>

**8.3 SSL**

To increase security measures, Secure Socket Layer must be enabled. The first step in enabling SSL is to create a self-signed certificate.

keytool -genkey -alias tomcat -keyalg RSA

The keystore that was just generated needs to be copied to the JBoss configuration directory. The tomcat server.xml file must be modified to comment out and the HTTP protocol and uncomment the SSL protocol:

<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"

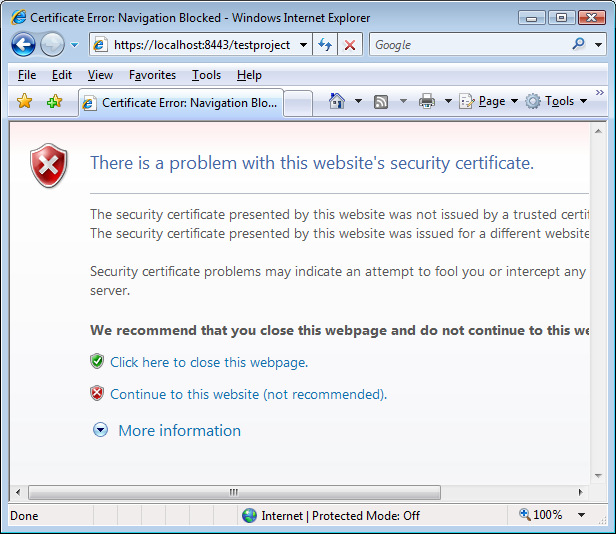
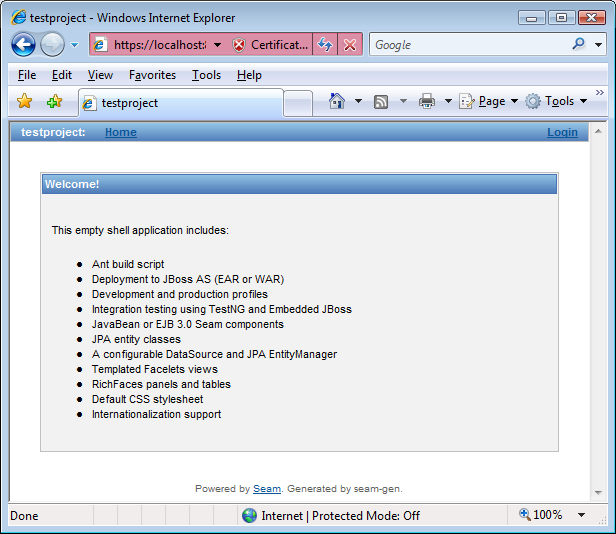
maxThreads="150" scheme="https" secure="true"

clientAuth="false" sslProtocol="TLS"

keystoreFile="${jboss.server.home.dir}/conf/testproject.keystore"

keystorePass=“Pass\_1" />

The JBoss application server may now be start up to validate SSL.

**9. Next Steps**

To further tighten security, I propose that further research be done on application server and database server. For example, firewall rules could be established to block anonymous access to the database server. Also, default settings need to be investigated in the application server to ensure that no security holes exist.

**10. Conclusion**

In conclusion, the JBoss Seam application web framework makes it simple to implement web application security with role-based validation; however, further investigation needs to be done to ensure that the application server is secure out of the box.

**11. References**

[1] JBoss Application Server. <http://www.jboss.org/jbossas/>

[2] JBoss Seam Framework. <http://seamframework.org>

[3] Hibernate Relational Persistence for Java and .NET. <http://www.hibernate.org/>