

M Ű E G Y E T E M 1 7 8 2

Bevezetés a Seam keretrendszer használatába

KÉSZÍTETTE: CSIKÓS DONÁT

Készült: 2011. Február

Bevezetés

A modern Java EE alapú rendszerekben sok összetett eszközkészlet alakult ki, melyek a gyakorlatban sokszor felmerülő problémákra próbálnak megoldást nyújtani. A Seam keretrendszer egy saját objektummodell bevezetésével, és sok hasznos technológia integrációjával próbálja segíteni a programozók életét. A bemutató célja, hogy egy egyszerű példán keresztül bemutassa a Seam alapvető tulajdonságait. A leírás feltételezi az Eclipse IDE felhasználó ismeretét.

Szükséges szoftverek¹

Alkalmazáserver	Jboss Application Server 5.1
Seam Runtime	Seam 2.2.1.FINAL
Adatbáziskezelő	MySQL 5.1
Jdbc driver	MySQL Connector/J 4.1.14
Fejlesztőkörnyezet	Eclipse 3.6 for Java EE developers
Eszköztámogatás	Jboss Tools 3.2

Előkészületek

A bemutatóhoz az aktuális gépen futnia kell egy MySQL adatbázisnak, melyben létezik egy *szolgint_demo* adatbázis.

```
mysql> create database szolgint_demo;
Query OK, 1 row affected (0.00 sec)

mysql> □
```

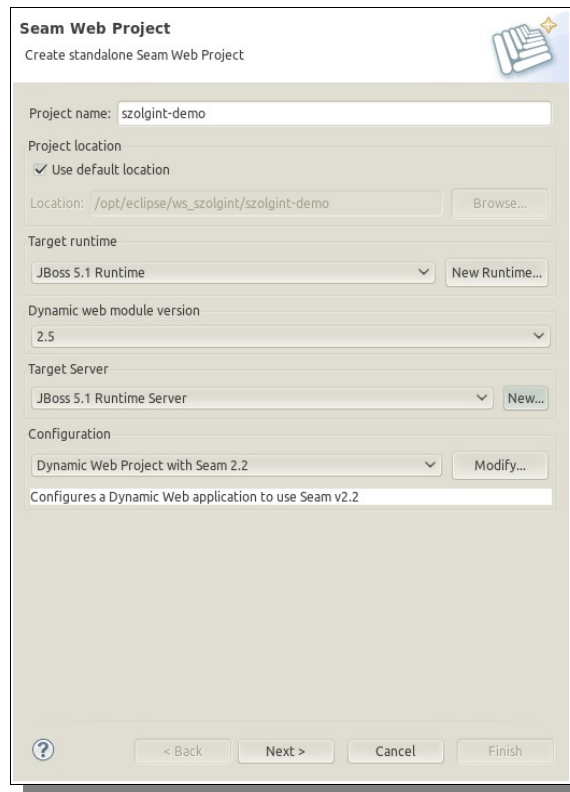
Új projekt létrehozása

A Seam kipróbálásához először a Jboss Tools segítségével létrehozunk egy új projektet, ami minden szükséges állományt és beállítást tartalmaz. Az induláshoz válasszuk ki az Eclipse főmenüjéből a *File*→*New*→*Other*→*Seam Web Project* pontot, és kövessük az utasításokat.

1 Alapinformációk megadása

A projekt neve mellett meg kell adni egy „Runtime”-ot, ami és egy „Target Server”-t. Itt lehet beállítani a felhasznált alkalmazáserver paramétereit, amin a programunkat futtatni fogjuk.

¹ A bemutató Fedora Linux 14 rendszeren készült, de Windows és Mac OS X Operációs rendszeren is működni kell.



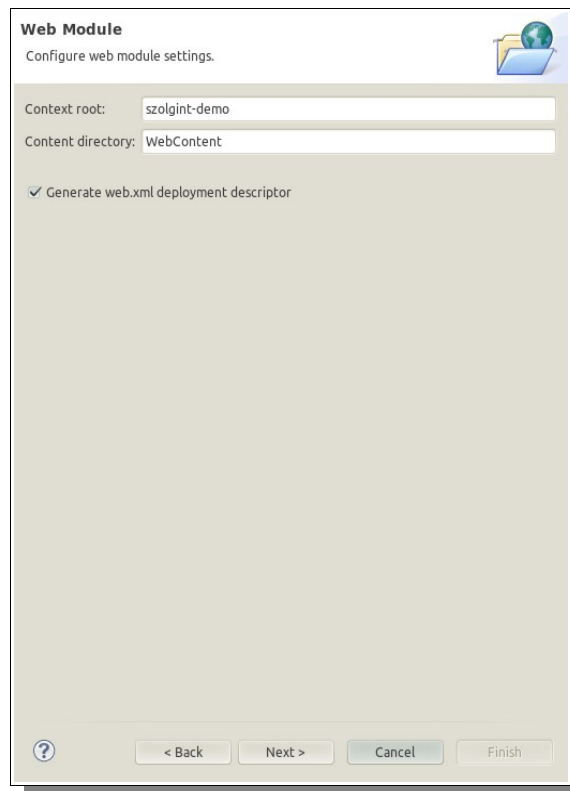
2 *Java beállítása*

Az alapbeállításként felkínált *src* könyvtár megfelelő, kattintsunk a „Next” gombra.

3 *Web Module*

A következő lépésben már fontosabb fontosabb beállítások következnek. A „Content directory” annak a mappának a neve, ami a webkiszolgáló gyökerébe kerül. A *web.xml* állomány a Servlet technológia² specifikációjának része, ami a kiszolgálón találgató programjaink leírását tartalmazza (elérési út, indulási paraméterek, stb.). A következő képen látható beállítások megfelelőek:

² A servlet egy olyan – JSR-154-es – specifikációnak megfelelő Java osztály, http kérésekre webes tartalmakat szolgáltat. Bővebb információ: <http://jcp.org/aboutJava/communityprocess/mrel/jsr154/index2.html>



Web Module
Configure web module settings.

Context root:

Content directory:

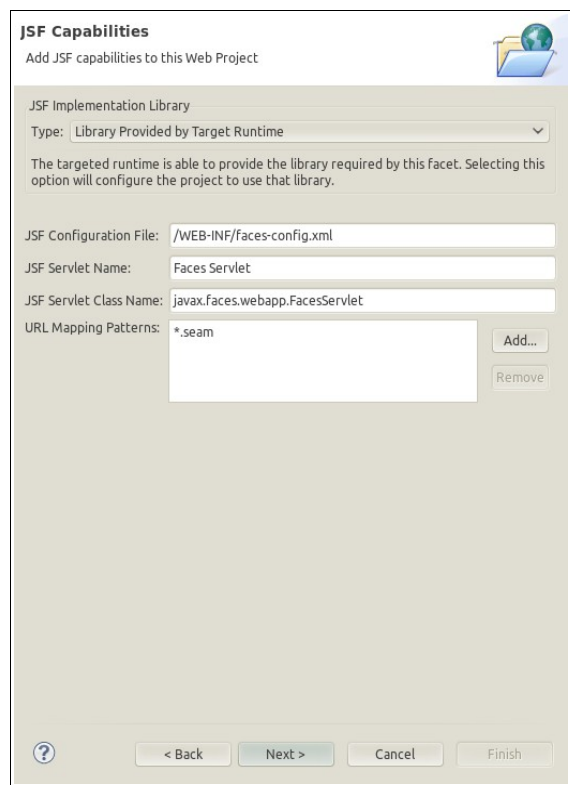
Generate web.xml deployment descriptor

4 JSF beállítások

A Seam alapértelmezetten Java ServerFaces-t használ a megjelenítéshez, ennek a beállításai:

- *Configuration File:* A megjelenítéshez kapcsolódó információkat tartalmaz (navigációs szabályok, felhasznált osztályok).
- *Servlet Name:* A JSF-et kezelő servlet neve.
- *Servlet Class Name:* A JSF servlet osztálya.
- *URL Mapping Patterns:* Milyen mintájú kéréseket szolgáljon ki a JSF.

Az előre beállított értékek megfelelőek.



JSF Capabilities
Add JSF capabilities to this Web Project

JSF Implementation Library
Type:

The targeted runtime is able to provide the library required by this facet. Selecting this option will configure the project to use that library.

JSF Configuration File:

JSF Servlet Name:

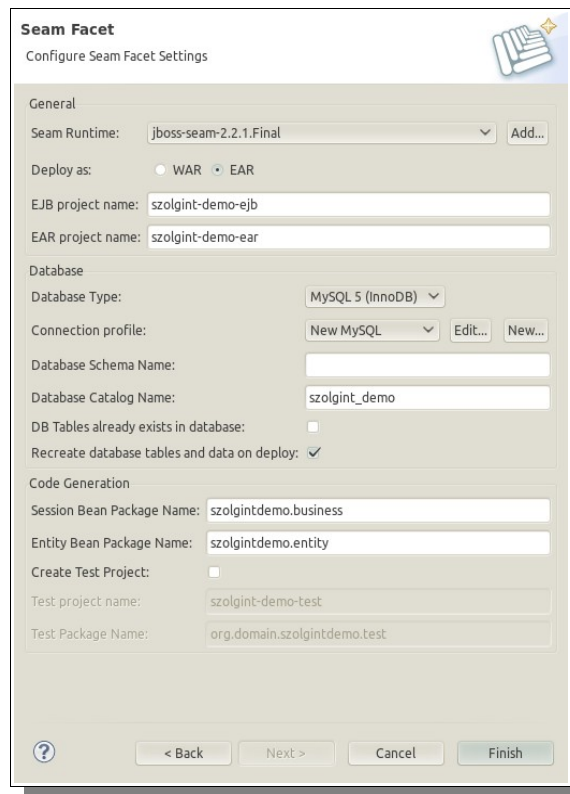
JSF Servlet Class Name:

URL Mapping Patterns:

5 Seam konfiguráció

A *Runtime*-ban meg kell adni a Seam disztribúció elérési útját. A „Deploy as” pontban állítható, hogy milyen formátumban kerüljön telepítésre a programunk az alkalmazáserverre. Ha a „WAR”-t választjuk, akkor a Seam-es komponenseink POJO-k (Plain Old Java Object-ek), ha pedig az „EAR”-t, akkor EJB Session Bean-ek lesznek.

Az adatbázis kapcsolatnál grafikusán adhatjuk meg, hogy a Seam-es alkalmazás hol tárolja az adatait. Ha készen vagyunk, akkor az alábbi képernyőképhez hasonló adatokat kell látunk.

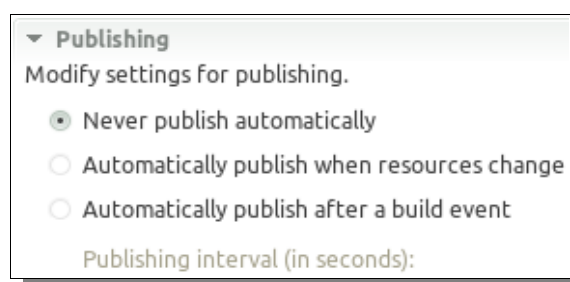


The screenshot shows the "Seam Facet" configuration dialog with the following settings:

- General**
 - Seam Runtime: jboss-seam-2.2.1.Final
 - Deploy as: WAR EAR
 - EJB project name: szolgint-demo-ejb
 - EAR project name: szolgint-demo-ear
- Database**
 - Database Type: MySQL 5 (InnoDB)
 - Connection profile: New MySQL
 - Database Schema Name: (empty)
 - Database Catalog Name: szolgint_demo
 - DB Tables already exists in database:
 - Recreate database tables and data on deploy:
- Code Generation**
 - Session Bean Package Name: szolgintdemo.business
 - Entity Bean Package Name: szolgintdemo.entity
 - Create Test Project:
 - Test project name: szolgint-demo-test
 - Test Package Name: org.domain.szolgintdemo.test

Buttons at the bottom: < Back, Next >, Cancel, Finish.

Alapértelmezetten az Eclipse minden mentés után automatikusan telepíti is a projektet. Ezt a „Server view”-ban érdemes kikapcsolni.



The screenshot shows the "Publishing" settings dialog with the following options:

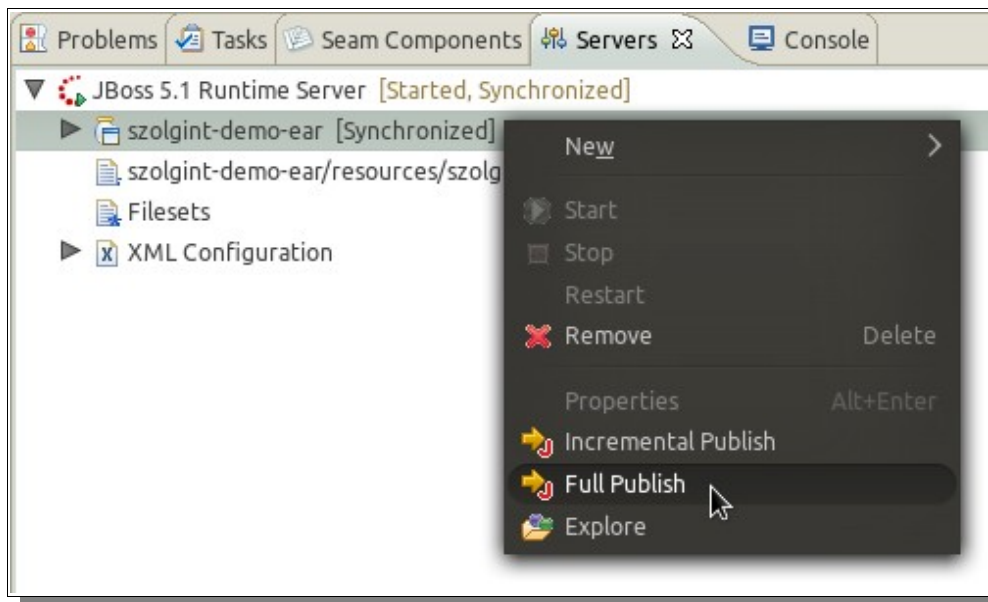
- Never publish automatically
- Automatically publish when resources change
- Automatically publish after a build event

Publishing interval (in seconds):

Az alkalmazás kipróbálása

Kattintsunk jobb gombbal a „Server view”-ban az alkalmazáserverre és válasszuk a „Start”-ot. A konzolban ekkor megjelenik az alkalmazáserver kimenete.

Megtörténhet, hogy az alkalmazáserver a programunkat korábban próbálja meg betölteni, mint a hozzá tartozó adatforrást-leíró (*szolgint-demo-ds.xml*), ami egy hosszú hibaüzenetet eredményezhet a kimeneten. A megoldáshoz elég még egyszer kezdeményezni a telepítést (deploy).



Ha minden rendben ment, akkor az alkalmazás elérhető a <http://localhost:8080/szolgint-demo> címen.

Seam komponens létrehozása és használata

Definiálni fogunk egy egyszerű Seam komponenst, mely segítségével elegánsan lehet kezelni a felhasználó felület eseményeit és a hozzá tartozó üzleti logikát. Bár a tooling ad segítséget a komponensek definiálásához, mégis egyszerűbb kézzel végrehajtani azt, mivel egyrészt nem bonyolult, másrészt pedig nem generálódik felesleges kód.

1 Adatmodell definiálása

Először hozzunk létre egy könyvet reprezentáló entitást, melynek metaadatait JPA annotációkkal adjuk meg:

```
package szolgintdemo.domain;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Scope;

@Entity
@Name("bookToStore")
@Scope(ScopeType.EVENT)
public class Book {
    Long id;
    String author;
    String name;

    @Id
    @GeneratedValue
    public Long getId() {return id;}

    public void setId(Long id) {this.id = id;}

    public String getAuthor() { return author;}

    public void setAuthor(String author) {this.author = author;}

    public String getName() {return name;}

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

A `@Name` annotációval lehet egy Seam komponenszt definiálni, ami a keretrendszer központi eleme. A komponensek egy saját konténerben tárolódnak, amiben a tárolt objektumok életciklusa automatikusan menedzselésre kerül.

Az egyes komponensek életciklusát a `@Scope` annotációval lehet definiálni. Beállítható egy JSF életciklus lefutásáig létező komponensből egészen az üzleti folyamatok szintjéig élő konfiguráció is. Kiemelt szerepe van a `conversation scope`-nak, ahol eseményvezérelten lehet az egyes elemek létrehozását és törését kivitelezni. Ennek bemutatására a második gyakorlaton kerül sor.

A komponensek dinamikusan létrehozhatóak, és érhetőek el, amire a következő pontban lesz példa. Ugyan a Seam komponensekhez külön konténer tartozik, ettől függően „szinte minden” lehet komponens. Ebben a példában a komponensek állapottal rendelkező (stateful) session bean-ek, amik az EJB konténerben futnak, az életciklusukat azonban a keretrendszer kezeli. A Seam2 keretrendszer az EJB-k mellett alapbeállításként támogatja a POJO és spring bean-ek használatát.

2 Üzleti metódus létrehozása

Hozzunk létre egy osztályt, amelynek segítségével a könyveinket tudjuk elmenteni és kilistázni.

```
package szolgintdemo.business;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import org.jboss.seam.annotations.In;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.faces.FacesMessages;
import szolgintdemo.domain.Book;

@Name("bookManager")
public class BookManager {

    // Fontos, hogy ugyanazt a változónevet adjuk meg, mint ami
    // a Book osztály @Name annotáció paraméterében szerepel.
    @In(required=false, create=true) Book bookToStore;
    @In EntityManager entityManager;
    @In FacesMessages facesMessages;

    public String storeNewBook(){
        entityManager.persist(bookToStore);
        facesMessages.add("Insert completed successfully.");
        return null;
    }

    public List<Book> getAllBooks(){
        Query q = entityManager.createQuery("select b from Book b");
        return q.getResultList();
    }
}
```

Ami az előző példához képest újdonság, az az `@In` annotáció. Ezzel lehet elérni egy komponensből egy másikat anélkül, hogy külön lookup metódussal le kellene kérdezni a referenciáját. A paramétereiben beállíthatjuk, hogy a komponensnek léteznie kell-e, illetve hogy a rendszer hozzon-e létre egy új példányt, ha a használatkor még nem létezik. Látható, hogy a saját komponensek mellett létezik sok előre definiált is. Az `@In`-jektálás a tagváltozó konkrét nevének felhasználásával történik (`bookToStore`).

3 Megjelenítés definiálása³

A programunk kipróbálásához egy egyszerű JSF megjelenítést használunk, amit a *home.xhtml*⁴ állományba írunk be:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:rich="http://richfaces.org/rich"
      xmlns:s="http://jboss.com/products/seam/taglib">

<h:messages />
<rich:panel>
  <f:facet name="header">Book manager test</f:facet>
  <h:form>
    <h:panelGrid columns="2">
      <h:outputText value="Author:" />
      <h:inputText value="#{bookToStore.author}" />

      <h:outputText value="Name:" />
      <h:inputText value="#{bookToStore.name}" />

      <h:outputText value="" />
      <h:commandButton action="#{bookManager.storeNewBook}"
        value="Add book" />
    </h:panelGrid>
  </h:form>
</rich:panel>
<h:dataTable value="#{bookManager.allBooks}" var="_b">
  <h:column>
    <f:facet name="header">Author</f:facet>
    #{_b.author}
  </h:column>
  <h:column>
    <f:facet name="header">Name</f:facet>
    #{_b.name}
  </h:column>
</h:dataTable>
</html>
```

A következő elemek fontosak:

- A `<h:messages />`-ben fogadja a backend üzeneteit; ez a szabványos JSF üzenetek kezelésének módszere.
- A `<h:form/>` pontosan egy `<form>` tag-et fog generálni a kimeneti html kódba.
- Az űrlapon kitöltött adatokat közvetlenül a *Book* típusú *bookToStore* nevű komponensbe kerülnek be,, a kapcsolatot pedig szabványos EL (Expression Language) kifejezés teremti meg..
- Az űrlap elküldését és a hozzá tartozó manager akciót szintén EL kifejezéssel adjuk meg. (`#{bookManager.storeNewBook}`).
- A komponensekben található attribútumokat a Java-ban szokásos getter-setter párosokon keresztül lehet elérni. Az itteni példában az egyszerűség kedvéért a getter metódusban van az összes adat lekérése.

Ha minden rendben van, akkor a böngészőben a következő tartalom jelenik meg:

³ Bevetérés a Java ServerFaces működésébe: <http://www.ibm.com/developerworks/library/j-jsf1/>

⁴ A Seam a Facelets templating technológiát is támogatja: <http://www.ibm.com/developerworks/java/library/j-facelets/>

• Insert completed successfully.

Book manager test

Author:

Name:

Author	Name
R.A. Salvatore	Home

Validáció hozzáadása

A Seam keretrendszer nagyon sok külső technológiát integrált magába, amit az alábbi példa szeretne demonstrálni. Jelen esetben a bemeneti adatok ellenőrzésére használható a Hibernate Validator eszköz ami out-of-the-box működik.

Ez az eszköz közvetlenül az adatmodellel definiált kényszeretek figyelni és megszakítja a program futását, ha valaki (pl. a megjelenítésen keresztül) megpróbál érvénytelen adatot felvenni. A kipróbáláshoz a *Book.java* állományt kell az alábbi módon kiegészíteni:

```
@org.hibernate.validator.Length(min=4, message="Too short name")
public String getName() {
    return name;
}
```

Hogy az eszköz a JSF életciklus validációs szakaszába integrálódjon, mindössze annyit kell tenni, hogy az ellenőrzendő űrlapot (<h:form/>) körbevevessük az <s:validateAll> jelöléssel.

Ha ez megvan, akkor az alkalmazáserverre való feltöltés után a nem megfelelő adatokat a rendszer automatikusan elutasítja:

• Too short name

Book manager test

Author:

Name:

Author	Name
R.A. Salvatore	Home