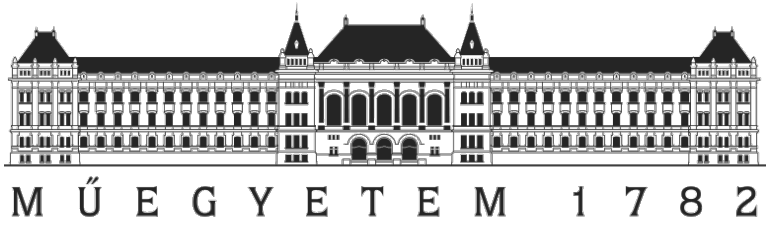

A Seam keretrendszer használata II.

Készítette: Csikós Donát
Email: csdonat@gmail.com

Készült: 2011. Március

Bevezetés

Az előző alkalommal elkezdett Seam bemutató éppen hogy csak érintette a keretrendszer képességeinek sokaságát. Ebben a dokumentumban néhány olyan tulajdonságát szeretném bemutatni, ami hasznos lehet az elkészítendő házi feladatok során. Részleteiben bemutatom a Seam conversation-ök használatát és a felhasználói felületbe való integrációját, valamint a klasszikus, és a manapság egyre népszerűbb RESTful webszolgáltatások definiálását is.

Az adatmodell elkészítése

A múltkoriak szerint hozzunk létre egy új projektet *bookstore* néven, az *ejb* projektben pedig helyezzünk el egy olyan JPA adatmodellt, amelyben felhasználók és az általuk megrendelt könyvek szerepelnek.

1 User.java

```
package bookstore.domain;

import java.util.LinkedList;
import java.util.List;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToMany;

import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Scope;

@Entity
@Name("currentUser")
@Scope(ScopeType.SESSION)
public class User {
      @Id
      @GeneratedValue
      Long id;
      String username;
      String password;
      @OneToMany(mappedBy = "customer")
      List<Order> orders = new LinkedList<Order>();

      public Long getId() {
            return id;
      }
      public void setId(Long id) {
            this.id = id;
      }
      public String getUsername() {
            return username;
      }
      public void setUsername(String username) {
            this.username = username;
      }
      public String getPassword() {
            return password;
      }
      public void setPassword(String password) {
            this.password = password;
      }
      public List<Order> getOrders() {
            return orders;
```

```
	}
	public void setOrders(List<Order> orders) {
		this.orders = orders;
	}
}
```

2 Order.java

```
package bookstore.domain;

import java.util.Date;
import java.util.LinkedList;
import java.util.List;
import javax.persistence.Entity;
import javax.persistence.Enumerated;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.ManyToOne;

@Entity
public class Order {
	public enum OrderStatus {
		OPENED, ORDERED, DELIVERED
	}

	@Id
	@GeneratedValue
	Long id;
	Date started;
	@Enumerated
	OrderStatus status;
	String shippingAddress;
	@ManyToOne
	User customer;
	@ManyToMany()
	List<Book> books = new LinkedList<Book>();

	public Long getId() {
		return id;
	}

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

	public Date getStarted() {
		return started;
	}

	public void setStarted(Date started) {
		this.started = started;
	}

	public OrderStatus getStatus() {
		return status;
	}

	public void setStatus(OrderStatus status) {
		this.status = status;
	}

	public String getShippingAddress() {
		return shippingAddress;
	}
```

```
	public void setShippingAddress(String shippingAddress) {
		this.shippingAddress = shippingAddress;
	}

	public User getCustomer() {
		return customer;
	}

	public void setCustomer(User customer) {
		this.customer = customer;
	}

	public List<Book> getBooks() {
		return books;
	}

	public void setBooks(List<Book> books) {
		this.books = books;
	}
}
```

3 *Book.java*

```
package bookstore.domain;

import java.util.LinkedList;
import java.util.List;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.ManyToMany;

@Entity
public class Book {
	@Id
	@GeneratedValue
	Long id;
	String author;
	String title;
	int cost;
	@ManyToMany(mappedBy = "books")
	List<Order> orders = new LinkedList<Order>();

	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 getTitle() {
		return title;
	}
```

```
	public void setTitle(String title) {
		this.title = title;
	}

	public int getCost() {
		return cost;
	}

	public void setCost(int cost) {
		this.cost = cost;
	}

	public List<Order> getOrders() {
		return orders;
	}

	public void setOrders(List<Order> orders) {
		this.orders = orders;
	}
}
```

4 import.sql

Az *import.sql* állományba tegyük be a következő kódot, hogy az induló alkalmazásunkban is rendelkezésre álljon néhány tesztadat.

```
use bookstore;
insert into User(username, password) values ('admin', 'admin');

insert into Book(author, cost, title) values("R.A. Salvatore", 2000, "Home");
insert into Book(author, cost, title) values("Frank Herbert", 1000, "Dune");
insert into Book(author, cost, title) values("J.R.R. Tolkien", 6000, "Lord of the
rings");
insert into Book(author, cost, title) values("Douglas Adams", 1500, "Hitchhikers' guide
to the galaxy");
insert into Book(author, cost, title) values("Isaac Asimov", 1700, "The Foundation");
insert into Book(author, cost, title) values("H.P. Lovecraft", 400, "Dagon");
```

Komponens modell

1 Általános leírás

A Seam keretrendszer központi eleme a központi komponens-modellje. A rendszer minden komponenshez hozzárendel egy egyedi nevet (ami a *@Name* annotációval adható meg), valamint egy hatókört (*@Scope* annotáció paramétere). A *@Scope* azt a kontextust határozza meg, amelyben az adott komponensből a rendszer eltárol egy példányt. Ezek a kontextusok egyfajta időbeli behatárolást jelentenek. A következő lehetséges hatókörök adhatóak meg egy komponensre:

- STATELESS: Állapotmentes komponens.
- EVENT: A komponens egy felhasználói kérés kezelésének idejére létezik.
- PAGE: Addig létezik a komponens, amíg egy adott (web)oldalon tartózkodik a felhasználó.
- CONVERSATION: A komponens létezése külső programozottan van megadva. Ez a hatókör a legfontosabb a Seam keretrendszerben.
- SESSION: A komponens a felhasználói munkamenet idejéig elérhető.

- APPLICATION: A komponens a program futása idején elérhető.
- BUSINESS_PROCESS: A komponens túléli az alkalmazásszerver újraindítását. A tárolás BPM motoron keresztül történik.

Ha például egy olyan *SESSION* hatókörű komponensre hivatkozunk az alkalmazásunkban, amelyik már létezik, akkor a rendszer ebből a kontextusból fogja előkeresni a *@Name* annotáció paraméterében megadott azonosítóhoz tartozó referenciáját. Gyakorlatiasan itt arról van szó, hogy a konténer minden kontextushoz karban tart egy-egy kulcs-érték párokból álló gyűjteményt, melyekben a kulcsot a *@Name* adja, az érték pedig vagy null, vagy egy objektum-referencia. Ezekhez a gyűjteményekhez a kontextusok létrejötte és megsemmisülése között alkalmazás programozottan hozzáférhetünk.

Fontos, hogy egy komponens a példányosítás pillanatában rendelődik hozzá a kontextushoz. Egy komponens-névhez pontosan egy objektum tartozik. Java kódban az *@In ClassName compoentName* formában szerezhetünk referenciát egy, a már létező komponensre. Lényeges, hogy a *componentName* itt pontosan az kell, hogy legyen, ami a komponens @Name annotációjában szerepel[1]. Ekkor, ha nem létezik az @In-jektált komponens, akkor NullPointerexception-t kapunk. Ez úgy kerülhetjünk el, hogy az *@In(create=true)* paramétert beállítjuk. Ekkor ha a komponens nem létezik, a keretrendszer automatikusan létrehoz nekünk egy példányt

A komponensek közé mi magunk is betehetünk elemeket. Ezt az *@Out ClassName compoentName* formában lehet lehet végrehajtani.

Kiemelt fontosságú hatókör a *CONVERSATION*, ahol az életciklus határai programozottan lehet megadni. Egy (*long-running*) conversation akkor indul el, ha egy komponens *@Begin* annotációval ellátott függvényét hívjuk meg, illetve akkor ér véget, ha egy *@End* annotációjú metódus fut le. Ha egyetlen conversation sincs elindítva, akkor minden egyes felhasználói kérés külön conversation-nek számít (*temporary conversation).*

A komponenseket, illetve a conversation-ok adatait a Component és Conversation osztályok metódusain keresztül explicite lehet lekérdezni és módosítani. Ezek az *org.jboss.seam.core* csomagban találhatóak.

Ha több conversation-t szeretnénk egyszerre futtatni, akkor az ehhez tartozó beállításokat a *@Begin* annotáció *nested* és *join* paraméterén keresztül tudjuk megtenni.

2 UserManager.java

Folytassuk a példánkat, definiáljunk egy *UserManager* típusú komponenst, ami a felhasználók azonosítását végzi:

```
package bookstore.business;

import java.util.List;

import javax.ejb.Remove;
import javax.ejb.Stateful;
import javax.persistence.EntityManager;
import javax.persistence.Query;

import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.In;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Out;
import org.jboss.seam.annotations.Scope;
import org.jboss.seam.security.Credentials;
```

1 Ha az *@In(value="componentName")* paraméterben megadjuk a komponens nevét, akkor lehet változó neve tetszőleges.

```
import org.jboss.seam.security.Identity;

import bookstore.domain.User;

@Name("bookstore.business.userManager")
@Scope(ScopeType.CONVERSATION)
public class UserManager {
      @In
      EntityManager entityManager;
      @In
      Identity identity;
      @In
      Credentials credentials;
      @Out(required = false)
      User currentUser;

      public boolean authenticateWithParams(String username, String password) {
            // Lekérdezzük a felhasználót, hogy létezik-e az adatbázisban
            Query q = entityManager
                        .createQuery("select u from User u where u.username=:username
and u.password=:password");
            q.setParameter("username", username);
            q.setParameter("password", password);

            @SuppressWarnings("unchecked")
            List<User> users = q.getResultList();

            // Siker esetén hozzáadjuk a felhasználót a munkamenethez
            // Egyébként pedig false-al térünk vissza
            if (users.size() == 1) {
                  identity.addRole("user");
                  currentUser = users.get(0);
                  return true;
            } else {
                  return false;
            }

      }

      // components.xml állományba ilyen szignatúrával kell bejegyezni az
      // authetikációs függvényt
      public boolean authenticate() {
            return authenticateWithParams(credentials.getUsername(),
                        credentials.getPassword());
      }

      // stateful session bean elvárt művelete
      @Remove
      public void remove() {
      }
}
```

Ha már saját authentikációt készítettünk, akkor regisztráljuk be a *components.xml*-be hogy a keretrendszer integráltan tudja használni.

```
<security:identity authenticate-method=
      "#{bookstore.business.userManager.authenticate}"
      remember-me="true"/>
```

3 BookManager.java

Definiáljanak egy, a könyvek lekérdezéséért felelős *BookManager* komponenst. Figyeljük meg, hogy csak akkor történik meg a lekérdezés, ha még nincs az *allBooks* attribútum példányosítva,

tehát az adatok frissítése csak új conversation kezdetekor történik meg.

```
package bookstore.business;

import java.util.List;

import javax.ejb.Remove;
import javax.ejb.Stateful;
import javax.persistence.EntityManager;

import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.In;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Scope;

import bookstore.domain.Book;

@Name("bookstore.business.bookManager")
@Scope(ScopeType.CONVERSATION)
@Stateful
public class BookManager implements BookManagerLocal {
      List<Book> allBooks;

      @In EntityManager entityManager;

      // könyvek listázása; a lekérdezés csak egyszer történik meg egy conversation-9n
bellül
      public List<Book> listAllBooks(){
            if (allBooks == null) {
                  allBooks = entityManager.createQuery("select b from Book b").getRe-
sultList();
            }
            return allBooks;
      }

      // SFSB elvárja
      @Remove
      public void remove() {
      }
}
```

Az interfész azokat a műveleteket tartalmazza, amik az implementációban is szerepelnek. A céljuk, hogy az EJB 3.0 specifikáció szerinti stateful session bean-t (SFSB) definiáljanak[2]. Ez csak egy kiegészítő információ, de érdemes tudni, hogy lehet úgy is definiálni Seam komponenst, hogy az egyben egy session bean is[3].

4 OrderManager.java

Következzen a megrendelések összeállításáért felelős *OrderManager* osztály:

```
package bookstore.business;

import java.util.Date;

import javax.persistence.EntityManager;

import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.Begin;
import org.jboss.seam.annotations.End;
import org.jboss.seam.annotations.In;
import org.jboss.seam.annotations.Name;
```

2 Erre vonatkozik a @Stateful annotáció is.

3 Azért lehet hasznos, mert a session bean-ekhez létező teljes eszköztár (p. klaszterezés) használható.

```
import org.jboss.seam.annotations.Scope;
import org.jboss.seam.faces.FacesMessages;
import org.jboss.seam.security.NotLoggedInException;

import com.sun.xml.internal.ws.developer.Stateful;

import bookstore.domain.Book;
import bookstore.domain.Order;
import bookstore.domain.Order.OrderStatus;
import bookstore.domain.User;

@Name("bookstore.business.orderManager")
@Scope(ScopeType.CONVERSATION)
@Stateful
public class OrderManager implements OrderManagerLocal {

	// üzenetküldés
	@In FacesMessages facesMessages;
	@In EntityManager entityManager;
	@In(required=false) User currentUser;

	// a megrendelést tartalmazó objektum
	Order order;

	// itt tároljuk a frissen kiválasztott könyveket, amit hozzáadunk majd az order-
hez
	Book bookToAdd = new Book();

	public Order getOrder() {
		return order;
	}

	public void setOrder(Order order) {
		this.order = order;
	}

	public Book getBookToAdd() {
		return bookToAdd;
	}

	public void setBookToAdd(Book bookToAdd) {
		this.bookToAdd = bookToAdd;
	}

	//új conversation inditasa
	@Begin(join = true)
	public String startPicking() {
		if(currentUser == null)throw new NotLoggedInException();
		System.out.println("Conversation started");
		order = new Order();
		return null;
	}

	// könyv hozzáadása a megrendeléshez
	public String addBookToOrder() {
		if(bookToAdd.getId() != null){
			order.getBooks().add(bookToAdd);
			bookToAdd = new Book();
			facesMessages.add("Book added");
		}
		return null;
	}

	// navigáció
	public String goToFinish(){
		return "/order.xhtml";
	}
	// a véglegesitő oldalon megjelenitjük, hány könyvet rendeltünk
```

```
	public int getTotal(){
		int i = 0;
		for(Book b : order.getBooks()){
			i += b.getCost();
		}
		return i;
	}

	// a megrendelés elküldése
	@End
	public String postIt(){
		// beállitjuk az order elemeit
		order.setStatus(OrderStatus.ORDERED);
		order.setStarted(new Date());
		order.setCustomer(currentUser);

		// elmentjük az entitást
		entityManager.persist(order);

		// értesités
		facesMessages.add("Package posted");

		// melyik oldalra megyünk (nem kell a navigation rule-okat beállitani)
		return "/home.xhtml";
	}

	// a megrendelés eldobása
	@End
	public String cancelIt(){
		// értesités és navigáció
		facesMessages.add("Package cancelled");
		return "/home.xhtml";
	}
}
```

5 Megjelenítés definiálása

Menü

Használjuk fel a generált megjelenítés template-jét és a *layout/menu.xhtml* állományban a menübe vegyük fel a saját oldalainkat.

```
<s:link id="SelectBooksId" view="/books.xhtml" value="Books" propagation="begin" acti-
on="#{bookstore.business.orderManager.startPicking()}"/>
```

Látszik, hogy a *books.xhtml* oldal betöltődésekor a *startPicking()* metódussal elindítjuk a conversation-t is.

books.xhtml

Ebben a szokásos JSF elemek mellett fontos, hogy az értékek a *<rich:table>* value részében egy seam komponensből vesszük az értékeiket, illetve a *<commandLink>-re* való kattintáskor a *<setPropertyActionListener>* segítségével kerül be a kiválasztott könyv a komponensbe *bookToAdd* attribútumába.

```
<!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">

        <h1>Books</h1>
     <rich:panel>
            <f:facet name="header">Pick books to order them</f:facet>
            <rich:dataTable var="_b" value="#{bookstore.business.bookManager.listAll-
Books()}">
                  <rich:column>
                        <f:facet name="header">Author</f:facet>
                        <h:outputText value="#{_b.author}" />
                  </rich:column>
                  <rich:column>
                        <f:facet name="header">Title</f:facet>
                        <h:outputText value="#{_b.title}" />
                  </rich:column>
                  <rich:column>
                        <f:facet name="header">Price</f:facet>
                        <h:outputText value="#{_b.cost}" />
                  </rich:column>
                  <rich:column>
                        <f:facet name="header">Order</f:facet>
                        <h:form>
                              <h:commandLink action="#{bookstore.business.orderMana-
ger.addBookToOrder()}" value="Order" >
                                    <f:setPropertyActionListener
target="#{bookstore.business.orderManager.bookToAdd}" value="#{_b}" />
                              </h:commandLink>
                        </h:form>
                  </rich:column>
            </rich:dataTable>

            <h:form>
                  <h:commandButton value="Finish" action="#{bookstore.business.order-
Manager.goToFinish()}" />
            </h:form>
            </rich:panel>

    </ui:define>
</ui:composition>
```

order.xhtml

Az *order.xhtml* oldalon, mivel a conversation él, ugyanazok a komponensek (komponens-példányok) érhetőek el. Itt csak a *Post* gombhoz kötött művelet hívása esetén kerül be a rendelés az adatbázisba, de mivel a *Cancel* gomb is egy *@End* annotációjú metódust hív, ezért a conversation mindenképpen véget ér.

```
<!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">
```

```
        <h1>Confirm order</h1>
     <rich:panel>
            <f:facet name="header">Confirmation</f:facet>
            <h:outputText value="Yout have selected "/><strong><h:outputText
value="#{bookstore.business.orderManager.order.books.size()}" /></strong><h:outputText
value=" book(s)." />
            <br/>
            <h:outputText value="Total cost: #{bookstore.business.orderManager.total}."
/>
            <h:form>
                  <h:outputText value="Shipping address:" />
                  <h:inputText value="#{bookstore.business.orderManager.order.shippin-
gAddress}" style=" width : 294px;"/>
                  <h:commandButton value="Post it" action="#{bookstore.business.order-
Manager.postIt()}" />
                  <h:commandButton value="Cancel it" action="#{bookstore.business.or-
derManager.cancelIt()}" />
            </h:form>
     </rich:panel>
    </ui:define>
</ui:composition>
```

Ha bármilyen probléma vagy kérdés felmerül az itt bemutatottakkal kapcsolatban, akkor az alábbi helyeken érdemes tájékozódni:

- **Hivatalos dokumentáció**: igen részletes és könnyen használható, jól érthető. Elérhető: http://docs.jboss.org/seam/2.2.1.Final/reference/en-US/html_single/
- **Példaalkalmazások**: A letölthető Seam disztribúció tartalmaz egy *examples* mappát, ami minden fontos használati esethez ad egy minta implementációt.
- **Fórumok**: A keretrendszer nyílt forrású és közösségi fejlesztésű jellegéből adódik, hogy sok-sok bejegyzés születik róla az interneten. Az itt található információkat érdemes óvatosan kezelni, mert néha akár félrevezető lehet, de sok esetben segít a hirtelen felmerülő problémákat megoldani. A hivatalos fórum címe: http://seamframework.org/Community/Forums

Fontos lehet még az alkalmazásunk fejlesztésénél, hogy a JBoss Application Serverberből létezik community és enterprise verzió. A kettő közötti nagy különbség, hogy az előbbihez nem tartozik aktív support, azaz a kiadási dátum óta felfedezett hibák javítását csak az enterprise tartalmazza.

RESTful web service definiálása

A RESTful webszolgatatás arra szolgál, hogy http kéréseken keresztül tegyük elérhetővé az alkalmazásunk egyes részeit. Ez alapvetően egy állapotmentes megvalósítást jelent. A kliens a http kéréssel szólítja meg az alkalmazást. A kérésnél nemcsak az elérési út fontos, hanem hogy milyen http kérés típust használunk (GET, POST, PUT, TRACE, DELETE, OPTIONS, stb.). Az alkalmazásunk az elérési utat és a típust kiértékelve dönti el, hogy amilyen műveletet végez el, illetve milyen erőforrást ad vissza a felhasználónak.

Seam esetén a RESTEasy komponens be van konfigurálva, az egyetlen teendőnk az, hogy az következő három jar állományt az EARcontent/lib mappába másoljuk:

- jaxrs-api.jar,
- resteasy-jaxrs.jar
- jboss-seam-resteasy.jar

Ezek biztosítják a megfelelő funkcionalitást. Amennyiben ezeket bemásoltuk, a szerver az alkalmazásunk telepítéskor végigpásztázza az osztályainkat. Ha valamelyiken megtalálja a *@Path* annotációt, akkor regisztrálja, hogy ez REST-es kiszolgálást fog végezni.

1 Példa

Hozzunk létre egy egyszerű *UserManagerRest* osztályt, aminek egyetlen egy feladata lesz: egy REST-es apin keresztül visszaadja, jó felhasználónév-jelszó párost adtunk-e meg.

```
package bookstore.business;

import java.util.List;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;

import org.jboss.seam.ScopeType;
import org.jboss.seam.annotations.In;
import org.jboss.seam.annotations.Name;
import org.jboss.seam.annotations.Scope;

import bookstore.domain.Book;

@Name("bookstore.business.userManagerRest")
@Scope(ScopeType.EVENT)
@Path("/user")
public class UserManagerRest{
      List<Book> allBooks;

      @In(value="bookstore.business.userManager", create=true) UserManager userManager;

      @GET                            // RESTful: get paraméterrel érjük el
      @Path("/{username}/{password}") // kérés paraméterezése
      @Produces("text/plain")         // visszatérési érték mime type-ja
      public String check(@PathParam("username") String username,
@PathParam("password") String password){ // @Path változójának leképezése
            boolean result = userManager.authenticateWithParams(username, password);
            return Boolean.toString(result);
      }
}
```

Amennyiben az alkalmazásunk települ a szerverre, akkor egy egyszerű plugin segítségével kipróbálható az szolgáltatás (pl.: RESTClient firefox plugin: https://addons.mozilla.org/en-us/firefox/addon/restclient).

Az URL amin elérhetjük az elkészített REST-es apinkat, a következő elemekből áll össze: .

- Az alkalmazásunk címével kezdődik: http://localhost:8080/booking
- A web.xml állományban megadott a Seam Resource Servlet url-patternjével folytatódik: /seam/resource
- A RESTEasy alapbeállításából adódóan /rest kulcsszó következik; ez konfigurálható.
- Az osztályunk *@Path* paraméterében megadott érték jön utána.
- A végén a metódus @Path paramétere van.

A példában szereplő teljes elérési út tehát:
http://localhost:8080/bookstore/seam/resource/rest/user/admin/admin

További információk a lehetséges beállításokról:

http://docs.jboss.org/seam/2.2.1.Final/reference/en-US/html/webservices.html#d0e22185

Webszolgatatás létrehozása

A JBoss alkalmazásszerver sok olyan kényelmi funkciót tartalmaz, aminek segítségével könnyedén lehet webszolgatatásokat definiálni. Ezeknek a technológiáknak a gyűjtőprojektje a Jboss-WS (http://www.jboss.org/jbossws) Alapvetően a JAX-WS specifikációt (Java Api for Xml Web Services) használja. Egy alap webszolgáltatást a következő módon lehet definiálni:

```
package bookstore.business;

import javax.jws.WebMethod;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.jws.soap.SOAPBinding.Style;

@SOAPBinding(style=Style.RPC)
@WebService
public class SimpleWS {

      @WebMethod
      public String echo(String in){
            return "echo " + in;
      }
}
```

Ahhoz, hogy a szolgáltatást el is érjük, a *web.xml* állományban definiálni kell egy Servletet, ami erre az osztályra mutat és meghatározza, hogy milyen „context path”-on szeretnénk a szolgáltatásunkat kiajánlani.

```
 <servlet>
      <servlet-name>SimpleWS</servlet-name>
      <servlet-class>bookstore.business.SimpleWS</servlet-class>
 </servlet>
 <servlet-mapping>
      <servlet-name>SimpleWS</servlet-name>
      <url-pattern>/simplews</url-pattern>
 </servlet-mapping>
```

Innentől kezdve, ha újratelepítjük az alkalmazást, minden egyes objektum automatikusan legenerálódik, a szolgáltatás leírója pedig elérhető lesz a http://localhost:8080/bookstore/simplews?wsdl címen.

A wsdl állományt legegyszerűbben a SoapUI eszközzel próbálhatjuk ki: http://sourceforge.net/projects/soapui/files/

Webszolgatatás hívása

Ha a kezünkben van egy wsdl állomány, amit szeretnénk a programunkból meghívni, akkor a megfelelő forráskód automatikusan legeneráltatható a *wsconsume* parancssori eszköz segítségével. Ez az alkalmazásszerver *bin* könyvtárában található meg.

```
# ./wsconsume.sh -k simplews.wsdl
```

A *-k* kapcsolóval forráskódot generálhatunk class állományok helyett. Ha az általunk elkészített szolgáltatás wsdl állományával próbáljuk ki az eszközt, akkor a generált forráskódot az alábbi módon lehet felhasználni:

```
SimpleWSService s = new SimpleWSService();
SimpleWS ws = s.getSimpleWSPort();
String result = ws.echo("ezt"));
```

Egy jóval részletesebb dokumentáció található a http://community.jboss.org/wiki/JBossWS oldalon.

Seam komponensek elérése webszolgatatásokon keresztül

Ahhoz, hogy a seam-es conversation-ok, illetve a Seam teljes fegyvertára elérhető legyen, ahhoz a webszolgatatások bináris állományai mellett levő *META-INF* mappában el kell helyezni egy *standard-jaxws-endpoint-config.xml* nevű leírót a következő tartalommal.

```
<jaxws-config xmlns="urn:jboss:jaxws-config:2.0"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xmlns:javaee="http://java.sun.com/xml/ns/javaee"
              xsi:schemaLocation="urn:jboss:jaxws-config:2.0 jaxws-config_2_0.xsd">
   <endpoint-config>
      <config-name>Seam WebService Endpoint</config-name>
      <pre-handler-chains>
         <javaee:handler-chain>
             <javaee:protocol-bindings>##SOAP11_HTTP</javaee:protocol-bindings>
            <javaee:handler>
                <javaee:handler-name>SOAP Request Handler</javaee:handler-name>
                <javaee:handler-class>org.jboss.seam.webservice.SOAPRequestHandler</ja-
vaee:handler-class>
            </javaee:handler>
          </javaee:handler-chain>
      </pre-handler-chains>
   </endpoint-config>
</jaxws-config>
```

Ez egy olyan szűrőt definiál, ami biztosítja, hogy a kérések kiszolgálásakor a Seam életciklussal kapcsolatos teendői végrehajtásra kerüljenek.

Ha ez be van állítva, akkor tetszőleges (de állapotmentes) Seam komponens kiszolgálhat webes kéréseket is. Ilyenkor jó stratégia, ha a conversation-ök indítását és leállítását delegáljuk egy másik, állapottal rendelkező komponens felé, miközben a kiszolgáló komponens nem tárol állapotot.

Mivel a webszolgáltatást tetszőleges implementáció elérheti és használhatja, ezért a SOAP headerben kapott *conversation id*-t a klienseknek el kell menteniük és megfelelően karban tartva vissza is kell küldeniük. Például ha egy ilyen kérés érkezik:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:seam="http://seambay.example.seam.jboss.org/">
 <soapenv:Header>
   <seam:conversationId xmlns:seam='http://www.jboss.org/seam/webservice'>2</seam:con-
versationId>
 </soapenv:Header>
 <soapenv:Body>
   <seam:confirmAuction/>
 </soapenv:Body>
</soapenv:Envelope>
```

akkor ilyen formában kell rá válaszolni:

```
<env:Envelope xmlns:env='http://schemas.xmlsoap.org/soap/envelope/'>
 <env:Header>
   <seam:conversationId xmlns:seam='http://www.jboss.org/seam/webservice'>2</seam:con-
versationId>
```

```
 </env:Header>
 <env:Body>
   <confirmAuctionResponse xmlns="http://seambay.example.seam.jboss.org/"/>
 </env:Body>
</env:Envelope>
```

A Seam disztribúció példái között található egy *seambay* nevű mintaalkalmazás, ami az ebben a pontban ismertetett „conversational web services”-ek elkészítéséhez ad útmutatást.

További részletek: http://docs.jboss.org/seam/2.2.1.Final/reference/en-US/html/webservices.html