Java Persistence dengan JBoss Seam

Wiranto Herry Utomo[1], Jazi Eko Istiyanto[2]
[1]Progdi Teknik Informatika
Fakultas Teknologi Informasi
Universitas Kristen Satya Wacana, Salatiga
Jl. Diponegoro 52-60, Salatiga, Jawa Tengah
[2]Program Ilmu Komputer
Fakultas MIPA
Universitas Gadjah Mada, Yogyakarta
E-mail : wiranto.uksw@gmail.com [1], jazi@ugm.ac.id [2]

Abstract

Seam is based on Java EE, so it satisfies its framework duties in two fundamental ways: 1) Seam simplifies Java EE: Seam provides a number of shortcuts and simplifications to the standard Java EE framework, making it even easier to effectively use Java EE web and business components, 2) Seam extends Java EE: Seam integrates a number of new concepts and tools into the Java EE framework. These extensions bring new functionality within the Java EE framework. The most basic and widely used functionality of JBoss Seam is to be the glue between EJB3 and JSF. Seam allows seamless integration between the two frameworks through managed components. We use a registration example to show the use of an EJB3 session bean as a JSF action listener, and basic configuration of Seam. The registration example is a fairly trivial application that lets a new user store his username, real name and password in the database.

Keywords : Java EE, Seam, EJB3, JSF

1. Pendahuluan

Java Persistence API telah lama ditunggu kehadirannya, bahkan beberapa orang mengatakan telah sangat terlambat. Kehadiran standard *enterprise Java persistence* yang berbasis model pengembangan POJO telah mengisi gap dalam platform yang membutuhkannya. Upaya-upaya sebelumnya seperti yang dilakukan oleh *entity beans* dalam EJB 2.1 kurang berhasil karena canggung dan terlalu berat untuk aplikasi. EJB 2.1 ini tidak pernah diadopsi secara luas oleh sector industri. Namun justru produk persistence yang berkepemilikan semacam JBoss Hibernate dan Oracle TopLink telah meraih popularitas dalam industri dan berkembang sangat pesat [Kei06].

Dengan adanya Java Persistence API, pengembang sekarang dapat membuat kode persistence yang portable yang berjalan pada server Java EE 5, dan juga JVM stand-alone yang diluar server. Hal ini akan memudahkan pengembang yang telah mengunakan produk *persistence* untuk mengadopsi JPA karena mengijinkan pengembang *persistence* untuk mempelajari API secara cepat.

Pada saat ini hampir seluruh aplikasi memerlukan data persistent. Persistence merupakan proses penyimpanan dan pemanggilan data dari penyimpan permanent

(dapat berupa database). Pada saat membicarakan *persistence* dalam Java, maka secara normal juga membicarakan mengenai penyimpanan data dalam database relasional menggunakan SQL [Kei06].

Pemahaman mengenai *persistence* mengalami perubahan dengan dikeluarkannya Java Persistence API, yang merupakan standard baru untuk *object/relational mapping* (ORM) untuk Java EE dan Java SE yang dikembangkan dibawah Java Community Process sebagai bagian dari Enterprise JavaBeans 3.0. ORM merupakan pemetaan (*mapping*) objek dalam Java ke dalam entitas relasional dalam database [Yua07]. Dalam pengembangan Java Persistence API, EJB 3.0 Expert Group diuntungkan dari pengalaman framework *O/R mapping* yang telah digunakan dalam komunitas Java. Dalam pengembangan Java Persistence ini Hibernate mempunyai pengaruh nyata terhadap arah arah teknologi Java Persistence. Hal ini bukan hanya karena partisipasi dari Gavin King (pencipta Hibernate) dan anggota tim Hibernate dalam upaya standardisasi EJB 3.0, tetapi juga disebabkan oleh pendekatan langsung dan pragmatis Hibernate terhadap O/R mapping, serta kesederhanaan, kejelasan dan kekuatan API Hibernate [Bau07].

Hal ini semakin diperkuat dengan dikeluarkannya Hibernate 3, sebagai langkah ke depan dari para pengembang Hibernate. Kekuatan Hibernate 3 ini [Bau07], diantaranya adalah dukungan untuk operasi dataset yang besar, pilihan *mapping* yang lebih canggih, khususnya untuk penanganan database legacy (yang sudah ada); data filter; strategi dalam mengelola conversations, dan integrasi dengan **JBoss Seam** yang merupakan framework baru untuk pengembangan web yang menggabungkan JSF dan EJB 3.0. **JBoss Seam** mengelola persistence dengan dua cara yaitu menggunakan komponen *managed persistence context* (untuk JPA) atau menggunakan *managed session* (untuk Hibernate). *Seam-managed persistence context* merupakan komponen built-in dari Seam yang mengelola instance dari EntityManager atau Session dalam *conversation context*. Artikel ini bertujuan untuk mengimplementasikan pengembangan aplikasi web menggunakan Java Persistence dengan JBoss Seam, dengan menggunakan komponen *managed persistence context* (JPA).

2. Framework ORM JBoss Seam

Selama beberapa tahun, *persistence* telah menjadi topic hangat dalam perdebatan di lingkungan komunitas Java. Banyak pengembang tidak menyetujui lingkup dari masalah persistence ini. Seperti, apakah persistence merupakan masalah yang sudah terpecahkan melalui teknologi relasional (*query* dan *stored procedure*) atau masih tetap menjadi masalah yang harus diatasi dengan model komponen Java tertentu, semacam EJB *entity beans*? Haruskah pengembang tetap menuliskan kode operasi CRUD (*create, read, update, delete*) dalam bahasa SQL dan JDBC, ataukah hal ini dapat dilakukan secara otomatis ? Bagaimana mencapai portabilitas apabila setiap system manajemen database mempunyai dialek SQL sendiri-sendiri? Haruskah bahasa SQL dihilangkan sama sekali dan kemudian mengadopsi teknologi database yang berbeda, semacam system database object? Perdebatan

tetap berlanjut, namun solusi yang dinamakan *object/relational mapping* (ORM) sekarang ini telah diterima secara luas [Bau07]**.**

Hibernate merupakan implementasi layanan ORM yang bersifat *open source*. Hibernate merupakan project ambisius yang bertujuan melengkapi solusi terhadap masalah pengelolaan persistent data dalam Java. Hibernate menjadi mediasi interaksi aplikasi dengan database relasional, yang memungkinkan pengembang berkonsentrasi pada masalah logika bisnis saja. Dengan Hibernate, tidak harus mengikuti aturan-aturan spesifik dari Hibernate dan *design patterns* pada saat menuliskan logika bisnis dan kelas-kelas persistent. Sehingga Hibernate mengintegrasikan aplikasi baru dengan aplikasi yang sudah ada dan tidak memerlukan perubahan yang merugikan.

Framework ORM yang terkenal Hibernate diciptakan oleh Gavin King. Karena itu salah satu tujuan utama Seam yang juga diciptakan oleh Gavin King adalah memperbaiki solusi ORM. Dan pemecahannya adalah menggunakan framework *stateful* untuk memperbaiki solusi ORM. Salah satu tantangan terbesar dalam ORM adalah menjembatani paradigma antara dunia objek dalam Java dan dunial relasional dalam database. Konsep kunci dalam hal ini adalah "lazy loading". Konsep lazy loading adalah konsep dalam pemanggilan sebuah table dalam database yang menyertakan juga table yang menjadi relasinya. Sedangkan konsep eager loading adalah memanggil sebuah table tanpa menyertakan table relasinya. Pada saat memanggil table dalam database, tidak perlu memanggil seluruh table yang menjadi relasinya. Karena itu Seam menyediakan dukungan luas untuk dua arsitektur persistence Java yang terkenal yaitu Hibernate3 dan Java Persistence API yang dikenalkan dalam EJB 3.0 [Yua07].

JBoss Seam dikembangkan karena rasa frustasi dari tim Hibernate dengan jenis stateless dari generasi arsitektur aplikasi Java sebelumnya. Arsitektur manajemen *state* Seam awalnya dirancang untuk memecahkan masalah yang berkaitan dengan *persistence*, khususnya masalah yang berkaitan dengan *optimistic transaction processing*. Aplikasi online yang skalabel biasanya menggunakan transaksi optimistic. Hibernate [Bau07] dirancang untuk mendukung ide *persistence context* yang menggunakan transaksi optimistic. Sayangnya arsitektur yang disebut sebagai arsitektur *stateless* sebelum Seam dan EJB3.0 tidak diguakan untuk transaksi optimistic. EJB3.0 memperkenalkan ide mengenai komponen *stateful* (yaitu *statefull session bean*) dengan *extended persistence context.* Ini masih memecahkan problem sebagian saja, karena masih ada dua masalah yaitu :

- Siklus hidup *statefull session bean* harus dikelola secara manual melalui kode dalam lapisan web (sulit dipraktekan)
- *Propagation persistence context* antar komponen *stateful* dalam transaksi optimistic yang sama dimungkinkan tetapi tetap rumit.

Seam memecahkan problem pertama dengan menyediakan *conversations*, dan komponen *stateful session bean* untuk *conversation*. (Kebanyakan conversations menyajikan transaksi optimistic dalam lapisan data). Ini sudah memadai untuk aplikasi sederhana dimana *persistence context propagation* tidak diperlukan.

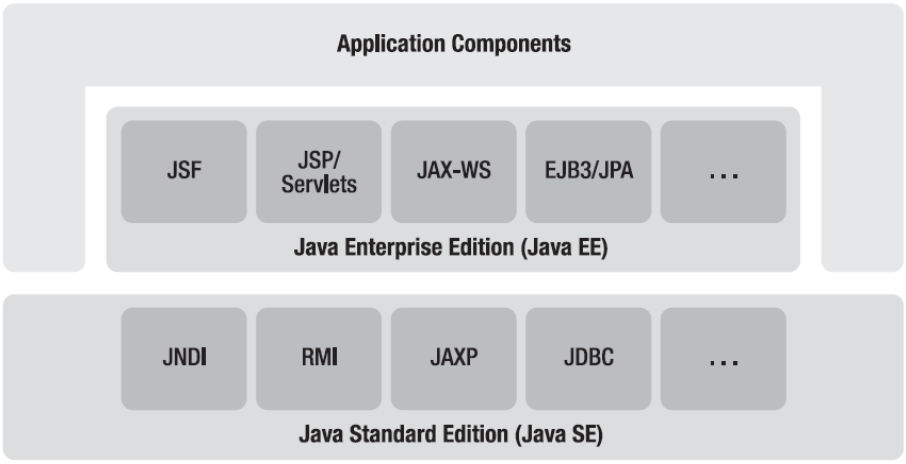
Untuk aplikasi kompleks, dengan banyaknya komponen yang berinteraksi secara longgar dalam setiap conversation, maka *propagation persistence context* lintas komponen menjadi isu yang sangat penting. Sehingga Seam memperluas model manajemen persistence context dari EJB 3.0 dengan menyediakan *extended persistence contexts* [Bau07].

Seam didasarkan pada Java EE, tetapi Seam menyederhanakan sekaligus memperluas Java EE. Seam menyederhanakan Java EE karena Seam menyediakan sejumlah *shortcut* dan penyederhanaan standard framework Java EE sehingga lebih mudah dan lebih efektif menggunakan web dan komponen bisnis Java EE. Seam memperluas Java EE karena Seam mengintegrasikan sejumlah konsep baru dan tool ke dalam framework Java EE. Perluasan ini memberikan fungsionalitas baru dalam framework Java EE [Bau07]**.**

Framework utama dalam Java EE 5.0 adalah EJB (Enterprise JavaBeans) 3.0 dan JSF (JavaServer Faces) 1.2. EJB3 merupakan framework ringan yang berbasis pada Plain Old Java Objects (POJO) untuk *business services* dan *database persistence*. JSF merupakan framework komponen Model-View-Controller (MVC) untuk aplikasi web. Kebanyakan aplikasi web Java EE 5.0 menggunakan modul EJB3 untuk *business logic* dan modul JSF untuk *front end web*. Namun walaupun EJB3 dan JSF saling melengkapi, EJB dan JSF dirancang sebagai framework yang terpisah, masing-masing mempunyai filosofinya sendiri-sendiri. Misalnya, EJB3 menggunakan anotasi untuk mengkonfigurasi layanan, disisi lain JSF menggunakan file XML. Selain itu, komponen EJB dan JSF saling terpisah sebagai level framework. Untuk membuat EJB3 dan JSF bekerja sama, diperlukan objek *façade* buatan (seperti JSF backing beans) untuk mengikat komponen bisnis ke halaman web, dan "mematri kode" (membuat kode baris demi baris dalam jumlah besar) untuk membuat "method call" melintasi batas framework. Perekatan kedua teknologi tersebut merupakan bagian dari tugas JBoss Seam [Bau07]**.**

Seam menghilangkan lapisan antara EJB3 dan JSF. Seam menyediakan pendekatan berbasis anotasi untuk mengintegrasikan EJB3 dan JSF. Dengan beberapa anotasi sederhana, komponen bisnis EJB3 dalam Seam dapat digunakan secara langsung ke web form JSF atau menangani *event UI web*. Seam memungkinkan pengembang untuk menggunakan POJO untuk semua komponen aplikasi.

Lingkungan Java EE standard mencakup Java Standard Edition (Java SE) dengan seluruh API (JDBC untuk akses database, JAXP untuk pemrosesan XML processing, dsb.) yang mendukung kemampuan level enterprise dari Java EE (JSF/JSP/servlets untuk komponen web, JAX-WS untuk web services, dsb.). Komponen aplikasi kemudian dibangun secara langsung pada puncak keseluruhan framework seperti yang ditunjukkan pada Gambar 1.

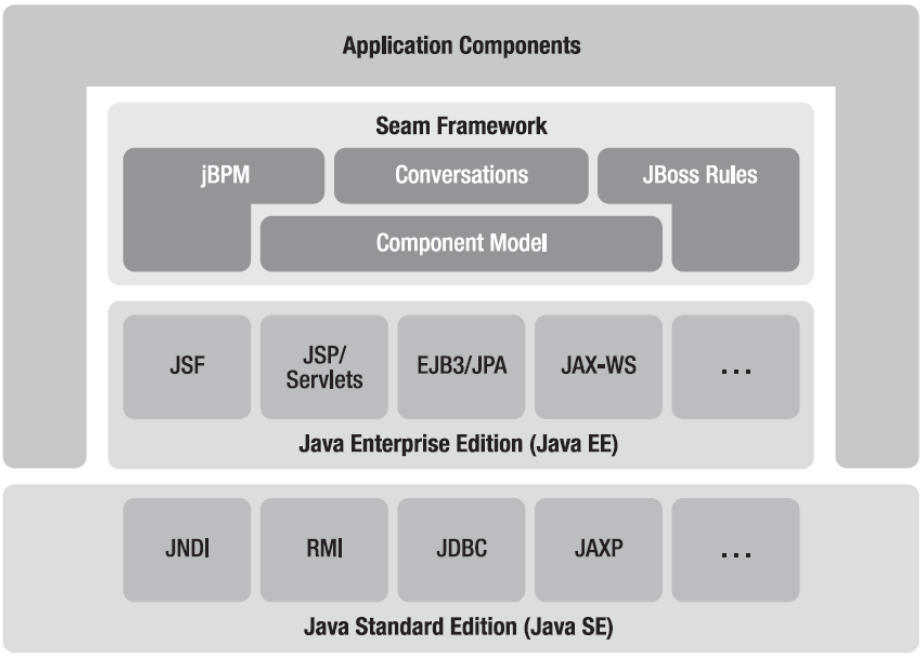


Gambar 1. Framework Java EE Standard [2]

Selain API dan jenis-jenis komponen seperti yang ditunjukkan pada Gambar 1, Java EE juga menyediakan *deployment services*, *runtime security*, dan layanan lain yang dibutuhkan untuk membuat aplikasi enterprise. Java EE menyediakan sejumlah peningkatan dibandingkan dengan framework J2EE sebelumnya (J2EE 1.4), seperti misalnya [Far07] :

- Anotasi Java 5.0 yang diintegrasikan secara bebas melalui API dalam Java EE memberikan pilihan yaitu dapat menggunakan XML eksternal atau anotasi kode yang ditempelkan (*embedded code annotation*)
- JavaServer Faces (JSF) 1.2, Java API for XML-based Web Services (JAX-WS) 2.0, dan Enterprise JavaBeans (EJB) 3.0 yang menyediakan model pemrograman yang lebih mudah dibandingkan J2EE pendahulunya, yang memungkinkan implementasi web, web service dan komponen bisnis menggunakan JavaBeans sederhana.
- EJB 3.0 menghilangkan kebutuhan interface yang banyak

Walaupun Java EE sudah mengalami peningkatan dibandingkan dengan J2EE sebelumnya, namun JBoss Seam masih melakukan perluasan dan penyederhanaan Java EE. Gambar 2 menunjukkan framework Seam melapisi antara kode aplikasi dan framework Java EE.



Gambar 2 : Framework Java EE yang ditingkatkan oleh Seam [Bau07]

Penyederhanaan yang dilakukan Seam adalah dengan model komponen. Komponen model Seam ini merupakan perpanjangan dari model komponen yang digunakan untuk *JSF managed beans*. Tetapi komponen Seam ini dapat digunakan lebih dari sekedar komponen lapisan web saja. Keuntungan dengan adanya model komponen Seam ini adalah penggunaan langsung dari komponen EJB sebagai *backing beans* untuk halaman JSF. Model standard JSF memungkinkan JavaBean digunakan sebagai *managed beans*, yang dikonfigurasi dalam file *JSF faces-config.xml*. Komponen EJB dapat diinvokasi dari *managed bean's callback methods*, dan bertindak sebagai façade untuk komponen EJB. Seam menyediakan jembatan langsung antara model komponen JSF dan model komponen EJB, yang memungkinkan penggunaan EJB secara langsung sebagai *JSF managed bean*. [Far07].

Penyederhanaan lain yang disediakan oleh Seam adalah kemampuan menggunakan anotasi kode untuk mengikat bean ke *JSF component names* secara langsung. , daripada menuliskan managed-bean dalam file faces-config.xml. Model komponen Seam mencakup anotas yang dapat digunkan untuk membuat link instance dari bean secara langsung ke *JSF managed bean name*. Ketika nama diguankan dalam sebuah JSF (misalnya satu property digunakan sebagai nilai dari field input HTML), kemudian *bean instance* secara otomatis diinisialisasi, jika perlu, dan digunkan sebagai *backing bean* untuk JSF. Karena itu koneksi bean ke *JSF managed bean name* dengan menggunakan faces-config.xml tidak lagi diperlukan [Far07]**.**

Model komponen Seam juga mendukung adanya versi umum dari *dependency injection*, yang disebut sebagai ***bijection*** [Far07]**.** *Dependency injection* standard

mencakup satu kali inisialisasi dari *bean reference* di dalam sebuah komponen, yang dikerjakan oleh beberapa container atau dikerjakan oleh layanan runtime lainnya. *Seam bijection* memperluas *dependency injection* dengan cara sebagai berikut :

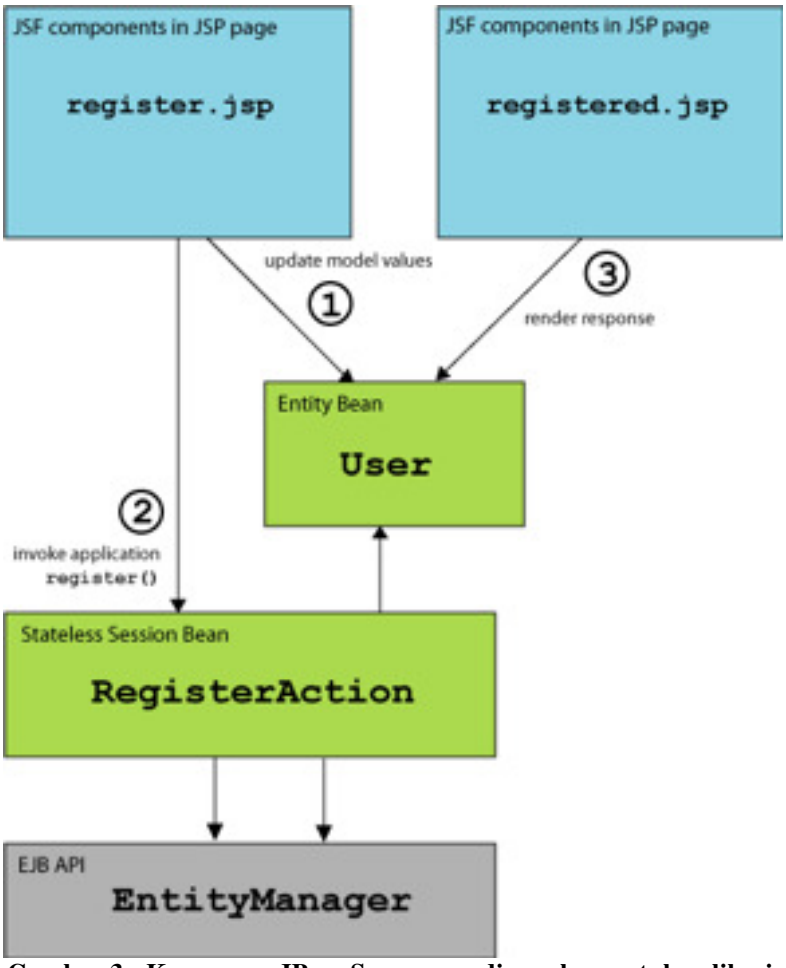
- Dua cara *propagation of references*: Komponen dapat mempunyai reference yang diinjeksikan oleh container, dan sebuah komponen dapat juga melakukan "*outject*" sebuah reference ke konteks yang terkait.
- *Dynamic updates*: Selain melakukan satu kali injeksi references, *bijection* dilakukan pada setiap kali invokasi komponen. Ini yang menjadi kunci dari model komponen Seam, karena itu komponen dapat menjadi stateful, dan bean dapat melakukan lintas invokasi.
- *Multiple contexts*: Dependencies (*incoming* dan *outgoing*) dapat dilakukan melintasi banyak konteks Seam, tidak hanya konteks tunggal saja. Sehingga *session-scoped component* dapat melakukan *inject request-scoped beans* dan melakukan *outject application-scoped beans*.

3. Contoh Kasus

Untuk menunjukkan aplikasi Java Persistence maka akan dibangun sebuah aplikasi sederhana yang diberi nama **registration.** Aplikasi ini merupakan aplikasi sederhana yang memungkinkan user untuk menyimpan username, realname dan password dalam database. Database yang digunakan dalam kasus ini adalah database HSQL yang "melekat" pada JBoss Aplication Server. Dengan aplikasi ini akan ditunjukkan bahwa pengembang tidak perlu lagi direpotkan dengan bahasa *query* untuk memanggil database.

Halaman awal menampilkan form yang sangat dasar dengan tiga field input. Setelah mengisi tiga field input tersebut dan melakukan submit dengan menekan tombol Register, maka aplikasi ini akan menyimpan objek user ke dalam database.

Contoh ini diimplementasikan dengan dua halaman JSP, satu berupa entity bean dan lainnya berupa stateless session bean.



Gambar 3 : Komponen JBoss Seam yang digunakan untuk aplikasi.

3.1. Kelas User

Untuk data user akan menggunakan EJB entity bean. Kelas **User** ini mendefinisikan *persistence* dan validasi secara deklaratif, melalui anotasi. Kelas ini juga memerlukan beberapa anotasi yang mendefinisikan kelas sebagai sebuah komponen Seam (Lihat contoh kode 1).

Contoh kode 1 : Kelas User yang berupa entity bean

```
@Entity  (1)
@Name("user")
@Scope(SESSION)
@Table(name="users") (2)
public class User implements Serializable {
    private static final long serialVersionUID =
1881413500711441951L;
    private String username; (3)
    private String password;
    private String name;
    public User(String name, String password, String
username) {
```

```
        this.name = name;
        this.password = password;
        this.username = username;
    }
    public User() {}
    @NotNull @Length(min=5, max=15)
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
    @NotNull
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    @Id @NotNull @Length(min=5, max=15)
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
}
```

Pada point **(1)** anotasi @Entity standard EJB3 menunjukkan bahwa kelas User merupakan sebuah entity bean. Pada point **(2)** Anotasi @Table standard EJB3 menunjukkan bahwa kelas User dipetakan ke table **users**, dan pada point **(3)** nama, password dan namauser adalah atribut persistent dari entity bean. Seluruh atribut persistent mendefinisikan metode accessor. Atribut persistent ini dibutuhkan ketika komponen ini digunakan oleh JSF dalam merender response dan melakukan update fase nilai model. Property dari kelas User diikat langsung ke komponen JSF, sehingga tidak dibutuhkan kode yang membosankan untuk mengkopi data bolak-balik antar halaman JSP dan model domain entity bean.

Namun, *entity bean* tidak melakukan manajemen transaksi atau akses database. Sehingga tidak harus menggunakan komponen ini sebagai *JSF action listener*. Sebagai *JSF action listener* akan menggunakan *session bean*.

3.2. Kelas RegisterAction

Kebanyakan aplikasi Seam menggunakan session bean sebagai *JSF action listeners* (dapat pula digunakan JavaBeans). Ada satu JSF action dalam aplikasi ini dan satu metode session bean yang ditempelkan. Dalam kasus ini, digunakan *stateless session bean*, karena seluruh state diasosiasikan dengan *action* yang ditangani oleh bean **User**.

Contoh kode 2 : Kelas RegisterAction yang berupa stateless session bean

```
@Stateless (1)
@Name("register")
public class RegisterAction implements Register {
```

```
    @In  (2)
    private User user;
    @PersistenceContext  (3)
    private EntityManager em;
    @Logger
    private Log log;
    public String register() (4) {
        List existing = em.createQuery(
             "select username from User where
username=#{user.username}") (5)
                .getResultList();
        if (existing.size()==0) {
            em.persist(user);
            log.info("Registered new user #{user.username}");
            return "/registered.xhtml";
        } else {
          FacesMessages.instance().add("User #{user.username}
sudah terdaftar");
            return null;
        }
    }
}
```

Pada point (1) anotasi @Stateless standard EJB menandai bahwa kelas tersebut sebagai stateless session bean. Selanjutnya pada point (2) anotasi @In menandai atribut bean karena diinjeksi oleh Seam. Dalam kasus ini, atribut diinjeksi dari variable konteks yang bernama user (instance dari nama variabel). Pada point (3) Anotasi @PersistenceContext standard EJB digunakan untuk menginjeksi entity manager EJB3. Pada point (5) metode *action listener* menggunakan EntityManager API standard EJB3 untuk berinteraksi dengan database, dan mengembalikan keluaran JSF. Karena ini sesson bean, maka transaksi secara otomatis dimulai ketika metode register() dipanggil dan dilakukan commit pada saat sudah lengkap. Pada point (6) Seam memungkinkan digunakannya JSF EL di dalam EJB-QL. Ini akan menghasilkan pemanggilan JPA setParameter() pada objek Query standard JPA.

Session bean action listener menampilkan *business logic* dan *persistence logic* untuk aplikasi mini ini. Dalam aplikasi yang lebih kompleks, diperlukan lapisan kode dan refactor *persistence logic* ke dalam komponen akses data.

Session bean mempunyai akses simultan ke konteks yang diasosiasikan dengan web request (nilai form dalam objek User, misalnya), dan state ditangani dalam transactional resources (objek EntityManager).

Secara alamiah, session bean membutuhkan *local interface*, yang dapat dilihat pada Contoh kode 3.

Contoh kode 3 : Kelas *local interface* Register

```
@Local
public interface Register {
    public String register();
}
```

3.3. Komponen deployment descriptor: components.xml

Jika sudah terbiasa menggunakan framework Java, maka akan terbiasa pula mendeklarasikan seluruh kelas komponen dalam beberapa macam file XML yang secara bertahap akan bertambah banyak dan tidak terkelola ketika proyek membesar. Seam tidak memerlukan komponen aplikasi yang didampingi oleh XML. Kebanyakan aplikasi Seam memerlukan sejumlah kecil XML yang tidak bertambah banyak ketika proyek menjadi lebih besar.

Meskipun demikian, file XML berguna pula untuk menyediakan beberapa konfigurasi eksternal dari beberapa komponen (khususnya komponen bawaan Seam). File components.xml, yang terletak di direktori WEB-INF digunakan untuk memberitahu Seam cara menemukan komponen EJB dalam JNDI :

Contoh kode 4 : Deployment descriptor **component.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<components xmlns="http://jboss.com/products/seam/components"
            xmlns:core="http://jboss.com/products/seam/core"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
            xsi:schemaLocation=
            "http://jboss.com/products/seam/core
http://jboss.com/products/seam/core-2.0.xsd
            http://jboss.com/products/seam/components
http://jboss.com/products/seam/components-2.0.<core:init
jndi-pattern="@jndiPattern@"/>
</components>
```

Kode tersebut mengkonfigurasi *property* yang bernama `jndiPattern` dari komponen bawaan Seam yang bernama `org.jboss.seam.core.init`. Simbol `@` yang lucu ini karena *Ant build script* menaruhkan JNDI ketika aplikasi di-*deploy*.

3.4. Deployment descriptor untuk EJB persistence

File persistence.xml memberitahu EJB persistence tempat untuk menemukan datasource, dan berisi beberapa setting vendor-specific. Dalam kasus ini, file ini memungkinkan untuk melakukan eksport schema secara otomatis pada saat startup.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/persistence

http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"
             version="1.0">
    <persistence-unit name="userDatabase">

<provider>org.hibernate.ejb.HibernatePersistence</provider>
        <jta-data-source>java:/DefaultDS</jta-data-source>
        <properties>
```

```
            <property name="hibernate.hbm2ddl.auto"
value="create-drop"/>
        </properties>
    </persistence-unit>
</persistence>
```

3.5. Halaman view register.xhtml dan registered.xhtml

Halaman view untuk aplikasi Seam dapat diimplementasikan menggunakan beberapa teknologi yang mendukung JSF. Dalam contoh ini digunakan Facelets menggunakan xhtml.

Contoh kode 5 : Halaman view **register.xhtml.**

```
<?xml version="1.0" encoding="utf-8"?>
<!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:s="http://jboss.com/products/seam/taglib"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
    <head>
        <title>Register New User</title>
    </head>
    <body>
        <f:view>
            <h:form>
                <s:validateAll>
                    <h:panelGrid columns="2">
                        Username: <h:inputText
value="#{user.username}" required="true"/>
                        Real Name: <h:inputText
value="#{user.name}" required="true"/>
                        Password: <h:inputSecret
value="#{user.password}" required="true"/>
                    </h:panelGrid>
                </s:validateAll>
                <h:messages/>
                <h:commandButton value="Register"
action="#{register.register}"/>
            </h:form>
        </f:view>
    </body>
</html>
```

Satu-satunya hal disini yang khusus untuk Seam adalah tag <s:validateAll>. Komponen JSF memberitahu JSF untuk melakukan validasi ke seluruh field input dengan anotasi Hibernate Validator yang ditentukan oleh entity bean.

Contoh kode 6 : Halaman view **registered.xhtml.**

```
<?xml version="1.0" encoding="utf-8"?>
<!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">
    <head>
```

```
        <title>Successfully Registered New User</title>
    </head>
    <body>
        <f:view>
            Selamat datang, #{user.name}, anda telah
terdaftar dengan nama user : #{user.username}.
        </f:view>
    </body>
</html>
```

Ini adalah halaman Facelets lama yang membosankan dengan menggunakan beberapa embedded EL. Tidak ada yang khusus untuk Seam disini.

4. Implementasi dan Hasil

Contoh kasus yang digunakan dalam artikel ini menggunakan EntityManager yang diinjeksi melalui EJB 3.0 container. Field anggota dalam EJB dianotasi dengan @PersistenceContext.

Pada saat melakukan submit form, JSF akan menanyakan Seam untuk menangani variable yang bernama user. Karena tidak ada nilai yang sudah diikat ke nama tersebut (dalam konteks Seam), maka Seam melakukan instantiasi komponen user dan mengembalikan *entity bean instance* User untuk JSF setelah menyimpannya dalam konteks Seam session.

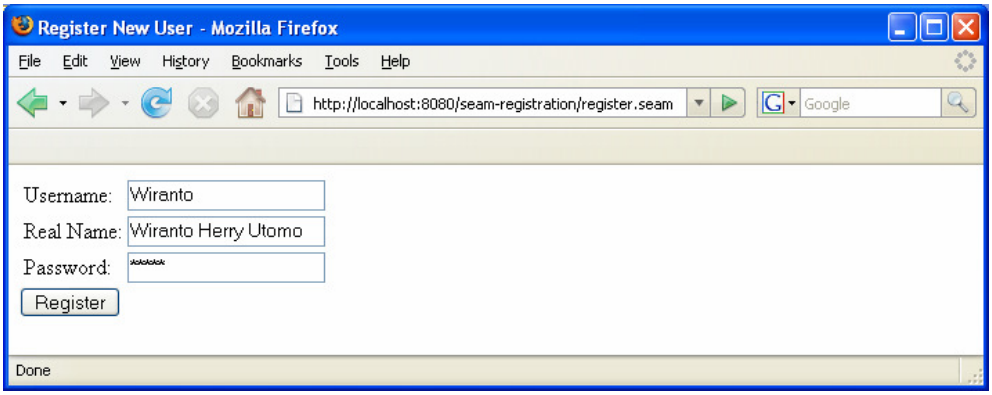
Nilai input form divalidasi dengan batasan Hibernate Validator seperti yang ditentukan pada entity User. Jika batasan dilanggar, JSF akan melakukan display ulang pada halaman web. Selain itu, JSF mengikat nilai input form ke property entity bean User.

Selanjutnya, JSF akan menanyakan Seam untuk menangani variable yang bernama register. Seam menemukan stateless session bean RegisterAction dalam konteks stateless dan mengembalikannya. JSF meminta metode register() action listener. Seam melakukan intercept panggilan metode dan melakukan injeksi ke entity User dari konteks session Seam, sebelum melanjutkan invokasi.

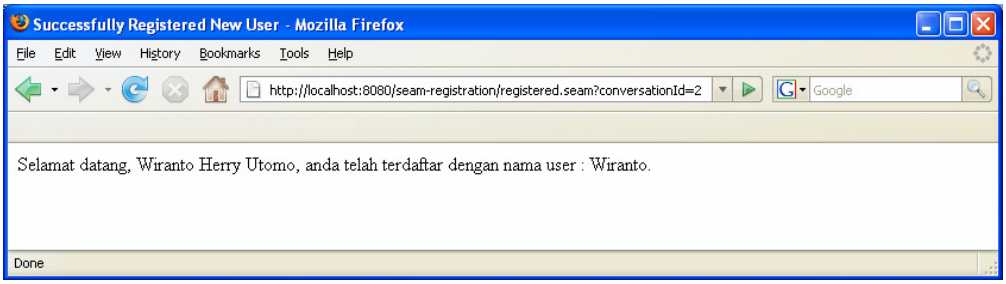
Metode register() akan melakukan cek apakah user dengan username yang sudah dmasukkan apakah sudah ada. Jika sudah ada, maak pesan error diantrikan dengan komponen FacesMessages, dan keluaran null akan dikembalikan, yang akan menyebabkan redisplay halaman. Komponen FacesMessages menginterpolasi JSF expression yang melekat pada pesan dan menambahkan JSF FacesMessage ke view.

Jika tidak ada user dengan username yang sudah ada, maka "/registered.xhtml" akan memicu browser untuk melakukan redirect ke halaman registered.xhtml. Pada saat JSF melakukan *render* halaman, JSF akan menanyakan Seam untuk menangani variable yang bernama user dan menggunakan nilai property dari User entity dari skope session Seam.

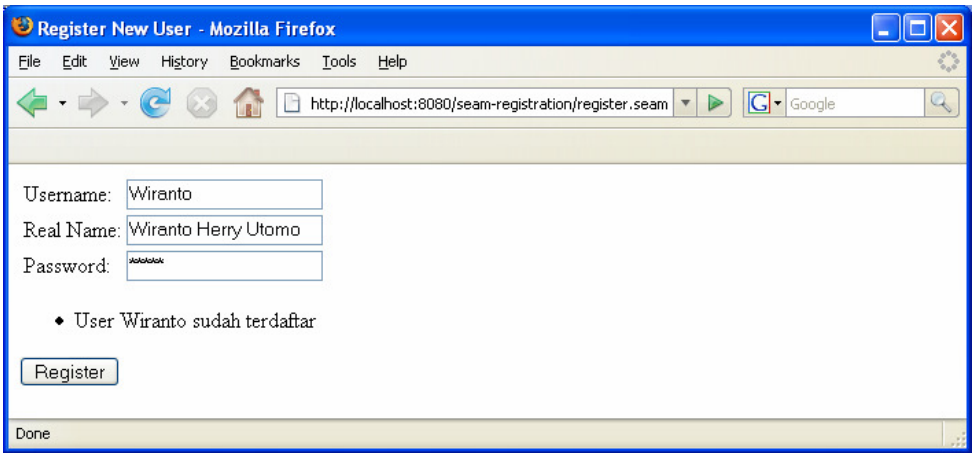
Gambar 4 - 7 menunjukkan hasil implementasi aplikasi **registration** yang menggunakan Java Persistence API (JPA) dari JBoss Seam.



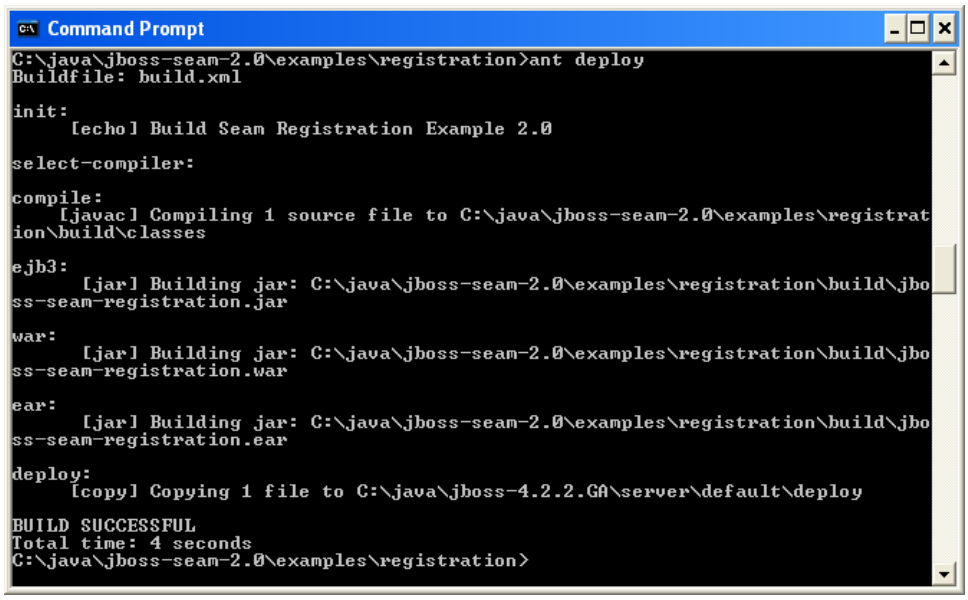
Gambar 4. Halaman muka aplikasi registration yang merupakan hasil tampilan dari file register.xhtml



Gambar 5. Halaman yang merupakan hasil tampilan dari file registered.xhtml



Gambar 6. Hasil pengujian dengan user yang sudah terdaftar



Gambar 7. Deploy aplikasi Registration menggunakan ant.

5. Kesimpulan

Pada tulisan ini dibahas pengembangan aplikasi sederhana registration dengan menggunakan Java Persistence API (JPA) dengan teknologi JBoss Seam. Dengan aplikasi ini telah dibuktikan bahwa pengembang yang menggunakan Java Persistence tidak perlu lagi direpotkan dengan bahasa query untuk memanggil dan menyimpan data ke dalam database.

Daftar Pustaka

[Kei06] Keith, Mike dan Schincariol, Merrick. 2006. Pro EJB 3: Java Persistence API. Apress. United State of America.

[Bau07] Bauer, Christian & King, Gavin, 2007. Java Persistence with Hibernate Revised Edition Of Hibernate In Action. Manning Publications Co. New York.

[Far07] Farley, Jim. 2007. Practical JBoss® Seam Projects. Apress. New York.

[Yua07] Yuan, Michael dan Heute, Thomas. 2007. JBoss® Seam Simplicity and Power Beyond Java™ EE, Prentice Hall. United State of America.