

# EJB 3.0 & JBoss Seam

---

# Wyzwania współczesnych aplikacji?

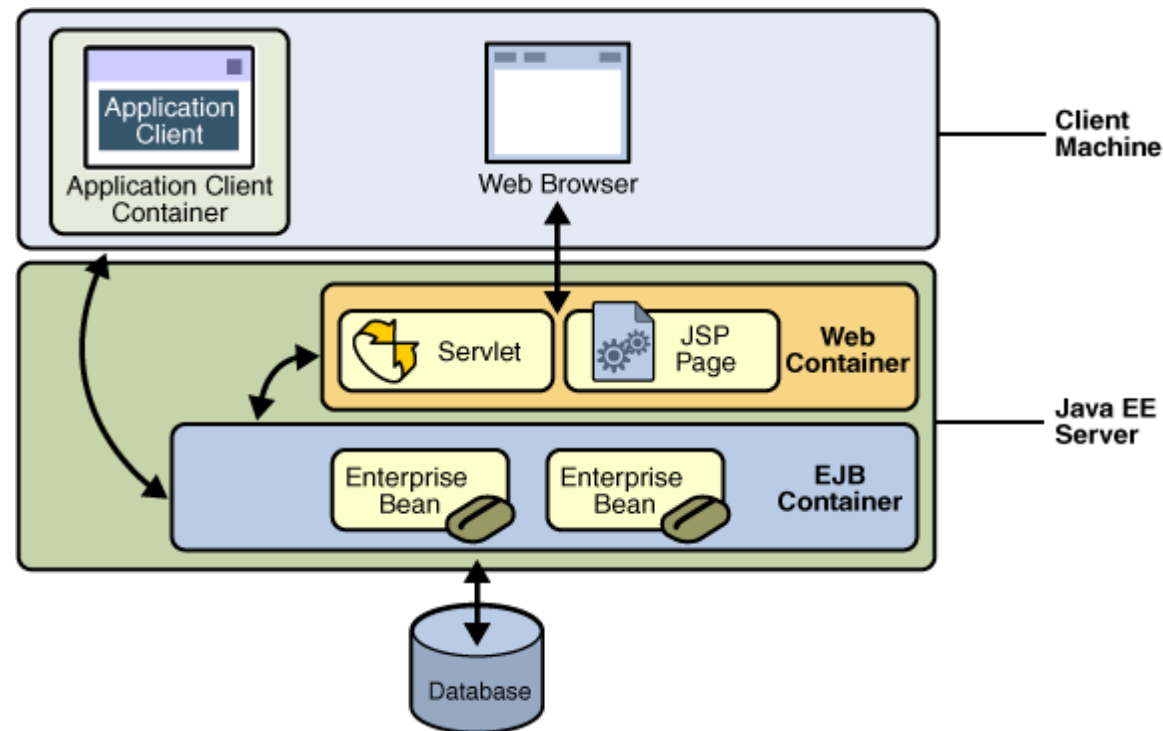
- Rozproszenie, zdalne wywołania
- Przetwarzanie transakcyjne
- Bezpieczeństwo
- Skalowalność
- Klastrowanie
- Łatwe wytwarzanie i rozwój

---

# Co to jest EJB?

- Enterprise Java Beans
- Służy do tworzenia rozproszonych aplikacji zorientowanych obiektowo.
- Framework udostępniający usługi, pozwalający developerowi skupić się na pisaniu logiki biznesowej.
- Jest częścią Java EE

# EJB w architekturze Java EE



---

# Typy komponentów EJB

- Session Bean
  - Stateless Session Bean (SSB)
  - Stateful Session Bean (SFSB)
- Message-Driven Bean (MDB)
- Entity (JPA)

---

# SSB & SFSB

- Reprezentuje akcję pojedynczego klienta po stronie serwera.
- @Stateless & @Stateful
- Nie jest wywoływany współbieżnie
- SFB przechowuje stan pomiędzy zapytaniami

---

# EJB - HelloWorld

```
import javax.ejb.Stateless;

@Remote
public interface HelloWorld {
    String sayHello(String name);
}

@Stateless
public class HelloWorldBean implements HelloWorld {
    public String sayHello(String name) {
        return "Hello "+name+"!";
    }
}
```

- Spakować do pliku o rozszerzeniu .ejb3
- Wrzucić do katalogu deploy jbossa
- Gotowe. Tylko tyle!

---

# EJB – HelloWorld klient

```
import org.junit.Test;

public class HelloWorldTest {
    @Test
    public void sayHelloCall() throws NamingException {
        HelloWorld hello = (HelloWorld)(new
            InitialContext()).lookup(„HelloWorldBean/local”)
        ;
        hello.sayHello(„Jacek”);
    }
}
```

---

# Inversion of Control

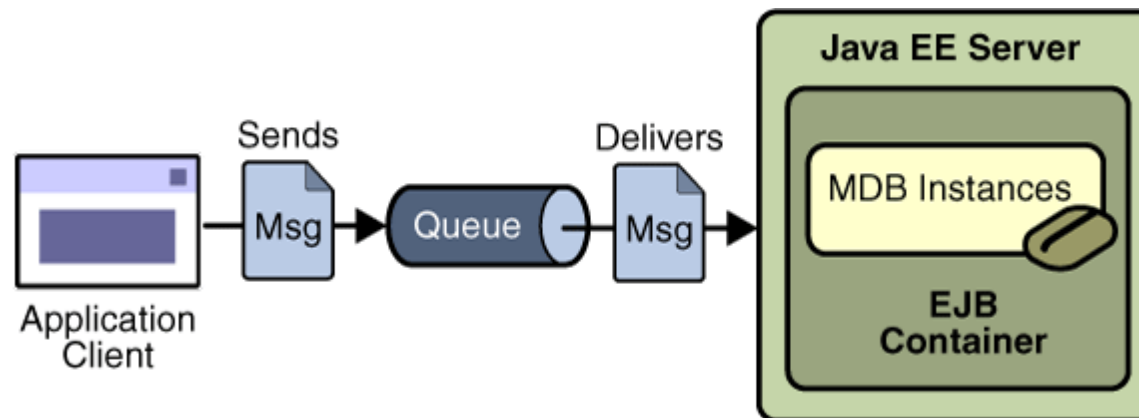
- Inversion of Control/Dependency Injection
- Skąd wziąć referencje do potrzebnych nam zasobów:
  - instancji innych komponentów EJB
  - źródeł danych (DataSource), itp. itd.
- Dwie możliwości
  - wzorzec Service Locator, np. wyszukiwanie przez JNDI:  
`Context.lookup("nazwa")`
  - IoC/DI
- Komponent deklaruje jakie zasoby są mu potrzebne, a dostarcza je kontener.

---

# Message-Driven Bean

- Przetwarza zapytania asynchronicznie
- Funkcjonuje jako JMS listener
- Zapytanie w postaci komunikatu może być wysłane przez dowolny komponent Java EE: klienta, innego beana, servlet lub dowolną inną aplikację korzystającą z JMS.
- Nie zachowuje stanu pomiędzy wywołaniami

# Message Driven Bean



---

# Entity

- Reprezentuje obiekt biznesowy przechowywany w bazie danych lub utrwalony innym mechanizmem
- @Entity
- Zwykle obiekt Entity odpowiada rekordowi tabeli z relacyjnej bazy danych

---

# Encje

- Java Persistence API (JPA)
- Język zapytań
- Adnotacje mapowania obiektowo-relacyjnego.

# Encja: przykład

```
@Entity
@Table(name="Kontakt")
public class Contact {

    @Id @GeneratedValue
    private int id;
    private String name;
    private String phone;

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    @Column(name="nazwa")
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getPhone() { return phone; }
    public void setPhone(String phone) { this.phone=phone; }
}
```

# Związki pomiędzy encjami

```
@Entity  
@Table(name="Kontakt")  
public class Contact {  
    @OneToMany(mappedBy="contact", cascade=CascadeType.REMOVE)  
    @OrderBy("created")  
    private List<Comment> comments = new ArrayList<Comment>();  
    ...  
}
```

```
@Entity  
@Table(name="Komentarz")  
public class Comment {  
    @ManyToOne  
    private Contact contact;  
    ...  
}
```

---

# Praca z encjami (CRUD)

```
EntityManager em;

//dodaj kontakt
Contact n = new Contact("Jacek Gerbszt", „500333222");
em.persist(n);

//zmień adres
n.setAddress("Traugutta 115C");
em.merge(n);
em.flush(); //opcjonalnie

//szukaj kontaktu o id = 123
Notatka n = em.find(Notatka.class, 123);

//usuń kontakt
em.remove(n);
```

---

# EJBQL

- Dynamiczne zapytania:
  - `EntityManager.createQuery()`
  - `EntityManager.createNativeQuery()`
- Zapytania zdefiniowane:
  - `EntityManager.createNamedQuery()`
- EJBQL – praktycznie pełna funkcjonalność SQL-  
a: `group by`, `having`, wewnętrzne i zewnętrzne  
złączenia, podzapytania, `update`, `delete`
- rzutowanie zapytania na dowolny typ

---

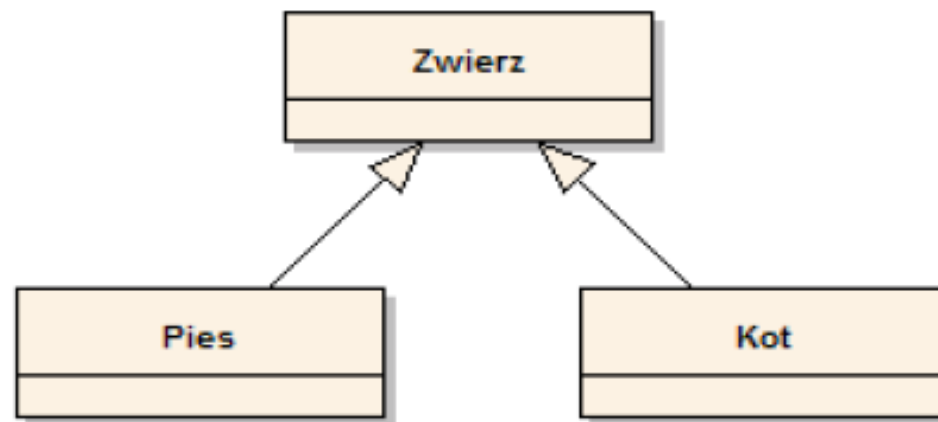
# EJBQL, przykłady

```
Query q = em.createQuery(
    "select from Contact n where n.name =
      :name" );
q.setParameter( "name", "Jacek Gerbszt" );
List<Contact> cncts = q.getResultList();
```

```
Query del = em.createQuery("delete from
Contact c where c.name = :name");
del.setParameter( "name", "Jacek
Gerbszt" );
del.executeUpdate();
```

# Dziedziczenie

- Encje mogą dziedziczyć po sobie
  - hierarchia dziedziczenia jest automatycznie odwzorowana w bazie danych
  - zapytania EJBQL mogą być polimorficzne



---

# Strategie dziedziczenia

- **SINGLE\_TABLE** – jedna tabela z polem (discriminator), które określa klasę konkretną (wymagane przez EJB 3.0 spec.)
- **JOINED** – tabela złączona z bazową na każdą klasę dziedziczącą
- **TABLE\_PER\_CLASS** – oddzielna tabela dla każdej klasy dziedziczącej

---

# Transakcje

- ACID:
  - Atomicy
  - Consistency
  - Isolation
  - durability

---

# Model transakcyjny EJB

- Model płaski
- Two-phase commit protocol
- Transakcje rozproszone
- Container-Managed Transactions
- Bean-Managed Transactions

---

# Atrybuty transakcji

- Required – jeśli nie ma, zostanie utworzona
- RequiresNew – zawsze nowa
- Supports – nie wymagana
- Mandatory – wymagana
- NotSupported – jeśli jest, zostanie wstrzymana
- Never – wyjątek, gdy transakcja

---

# Transakcje: przykład

```
@Stateful
public class CartBean implements Cart {

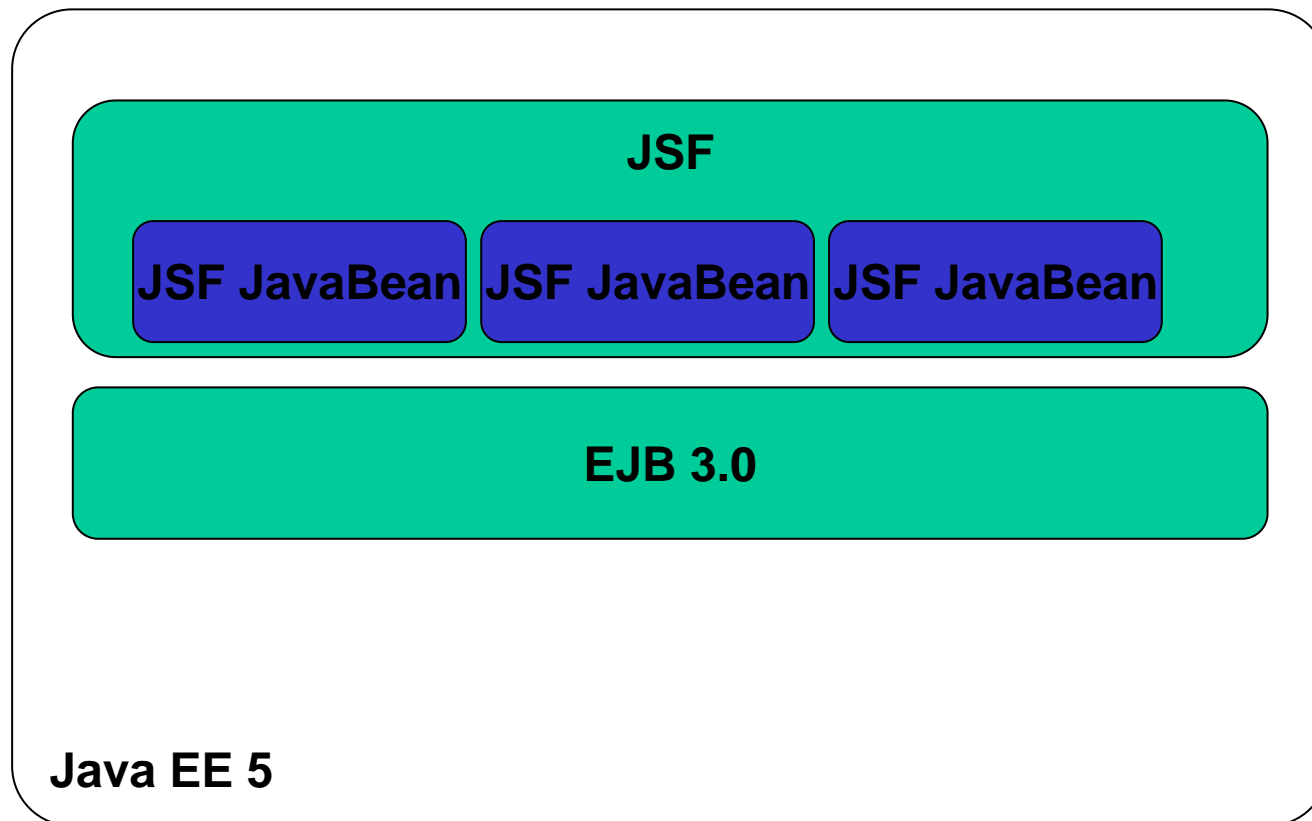
    @TransactionAttribute(TransactionAttributeType.REQUIRES
        _NEW)
    public void commit() {
        //...
    }
}
```

---

# Podsumowanie

- EJB 3.0
  - Pozwala tworzyć model obiektowy
  - łatwiejsze w przyswojeniu
  - większa funkcjonalność (komponenty encyjne)
  - szybsze tworzenie oprogramowania

# Współpraca JSF i EJB 3.0



---

# Współpraca JSF i EJB 3.0

- Typowy model współpracy:
  - Komponent JSF JavaBean (managed bean) wywołuje metody komponentu EJB i wykonuje operacje bazodanowe przy pomocy JPA
- Po co JSF JavaBean, skoro mamy już EJB?
- Jak zrealizować optymalnie sesję klienta?  
Sesja HTTP? Stateful Session Bean?
- Jak zrealizować długie transakcje?

# JBoss Seam

**JSF**

**JBoss Seam**

**EJB 3.0**

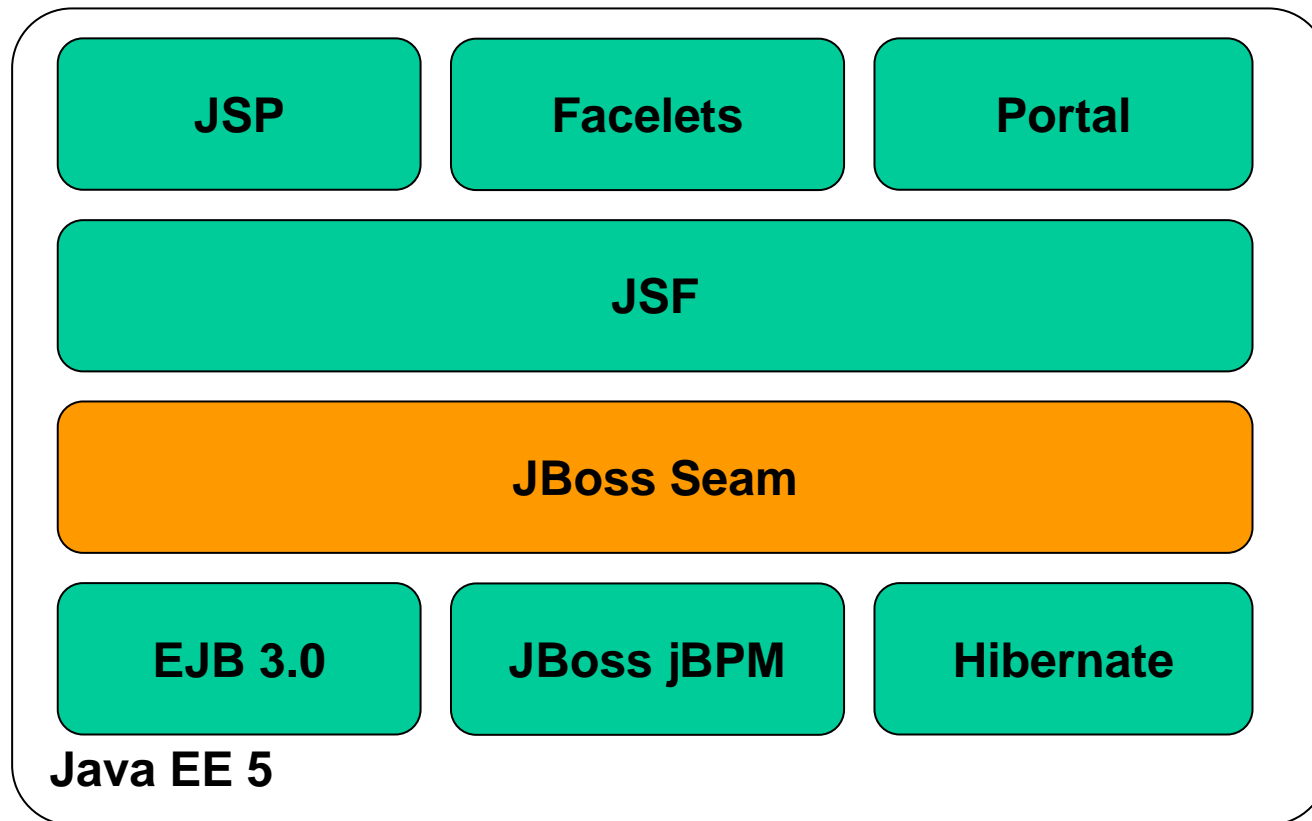
**Java EE 5**

---

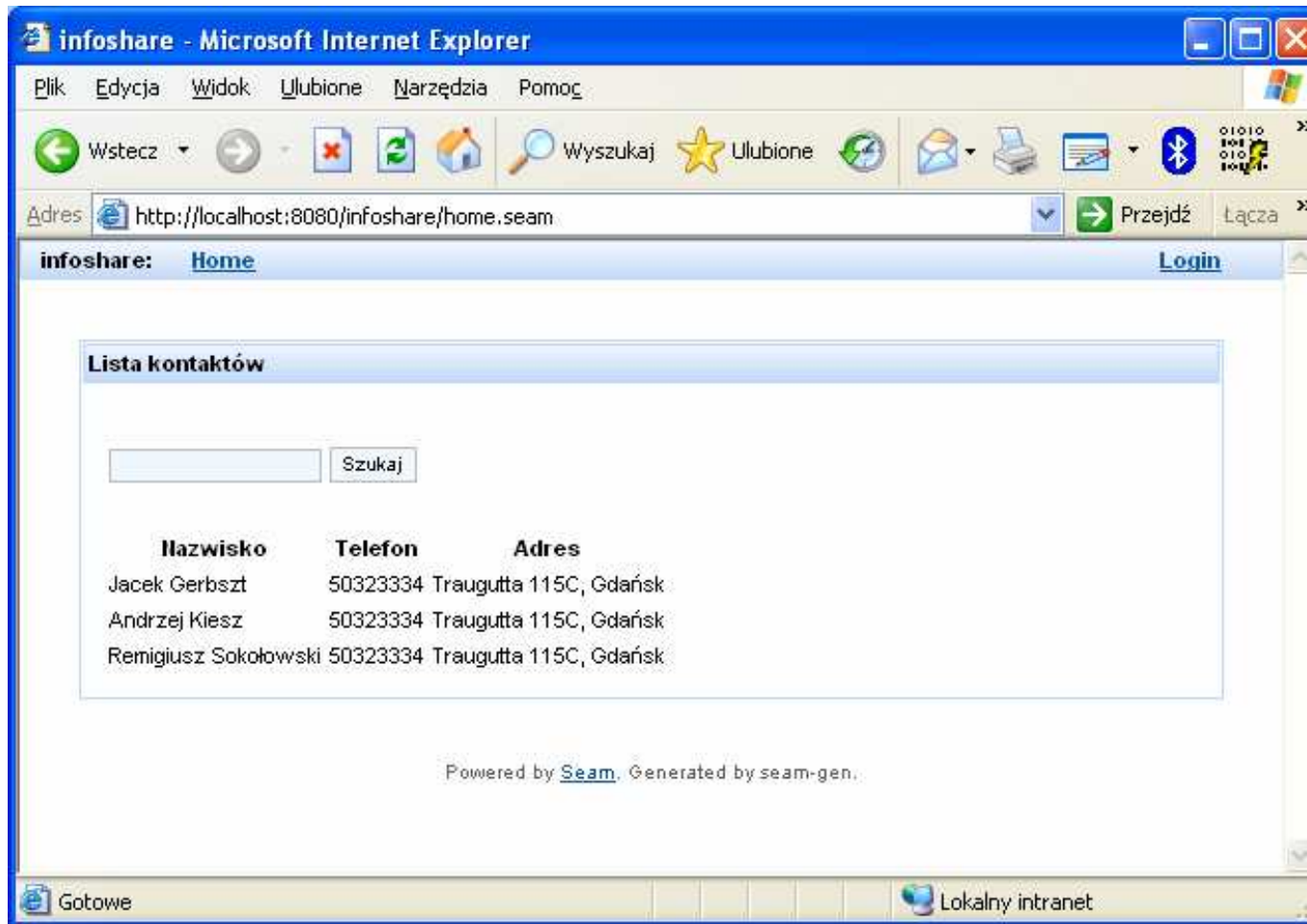
# JBoss Seam – cel powstania

- Integracja JSF z EJB 3.0
- Integracja AJAX-a (ICEFaces i Ajax4JSF)
- Wsparcie dla BPM (jBPM)
- Deklaratywne zarządzanie stanem aplikacji
- Bijection
- Zarządzanie workspace-ami
- Powszechne użycie adnotacji do konfiguracji

# JBoss Seam



# Seam przykład – lista kontaktów



The screenshot shows a Microsoft Internet Explorer browser window titled "infoshare - Microsoft Internet Explorer". The address bar displays "http://localhost:8080/infoshare/home.seam". The page content includes a search bar with the text "Lista kontaktów" and a "Szukaj" button. Below the search bar is a table with three columns: "Imię", "Telefon", and "Adres". The table contains three rows of contact information. At the bottom of the page, it says "Powered by Seam. Generated by seam-gen." The browser's status bar at the bottom shows "Gotowe" and "Lokalny intranet".

Imię	Telefon	Adres
Jacek Gerbszt	50323334	Traugutta 115C, Gdańsk
Andrzej Kiesz	50323334	Traugutta 115C, Gdańsk
Remigiusz Sokółowski	50323334	Traugutta 115C, Gdańsk

---

# Konteksty

- JBoss Seam umożliwia umieszczanie obiektów w kontekstach
  - Standardowe: bezstanowy, strony, sesji, zapytania, aplikacji
  - Niestandardowe: konwersacji i procesu biznesowego
- Najbardziej interesujący jest kontekst konwersacyjny

---

# Konwersacje

- Konwersacja – sekwencja wielu zapytań klienta realizująca jakieś zadanie
- Implementacje bez Seama:
  - Po każdym zapytaniu odtwarzany jest stan zapisany w bazie danych; problemy z wydajnością i skalowaniem
  - Stan zapisywany jest w HttpSession; trudne w implementacji.
- Seam wprowadza dodatkowy kontekst i dba, żeby konwersacje były od siebie dobrze odizolowane

---

# Bijection

- Dependency bijection: injection + outjection
- Pozwala na wymianę obiektów pomiędzy prezentacją i komponentami JSF w obie strony.

---

# SFSB

- Seam przywrócił do życia Stateful Session Beany
- Wydajne okazuje się często przechowywanie stanu aplikacji nie w sesji HTTP, ani w bazie danych, ale w SFSB

---

# Typowe problemy

- Seam adresuje wiele typowych problemów aplikacji WWW:
  - Praca w wielu zakładkach przeglądarki
  - Przycisk „Wstecz”
  - Ręczne wpisywanie adresu
  - Zmiana wielkości okna
  - Itd. ...

---

# Wiele udogodnień

- Klastrowanie nie tylko EJB, ale też zwykłych POJO
- Ułatwiona obsługa cache-a (np. keshowanie wyrenderowanych stron JSF)
- Scaffolding, czyli generowanie szkieletu aplikacji (podobnie jak w Ruby on Rails)
- Użycie walidatorów JPA do walidacji w JSF
- Deklaratywna autoryzacja i autentykacja
- Wsparcie dla skórek

---

# Środowisko dla Seam

- Może współpracować z każdym serwerem Java EE 5 (JBoss 4.0.5, GlassFish)
- Działa również na Tomcacie – zamiast EJB używamy zwykłych klas Java, lub zagnieżdżonego jądra JBoss.
- Może współpracować z dowolną biblioteką komponentów JSF: Facelets, Ajax4JSF, RichFaces, ICEFaces, itd....

---

# Dlaczego Seam?

- Najszybszy sposób na użycie AJAXa
- Bardzo prosty start z EJB 3.0
- Ułatwia wykorzystanie JSF
- Integruje BPM
- Ułatwia zautomatyzowane testy integracyjne
- Bazuje na otwartych standardach
- Sam też ma szansę wejść do standardu Java EE (JSR 299)

JG2

## Slajd 39

---

**JG2**

Jacek Gerbszt; 2007-04-23

---

# Dalsze informacje

- <http://java.sun.com/javase/5/docs/tutorial/doc/>
- <http://labs.jboss.com/portal/jbossseam/docs>
- Wiele blogów i artykułów w sieci.