Padrões e AntiPadrões J2EE
Trabalho de Conclusão da Disciplina

Thuan S. Nabuco[1], **João Vianey**[2]

[1]Pós-Graduação em Engenharia de Software com Ênfase em Padrões de Projeto
Universidade Estadual do Ceará (UECE)
Fortaleza – CE – Brasil

thuan@ifce.edu.br, joaovianey@gmail.com

Resumo. *Este artigo consiste em uma pesquisa para disciplina de Padrões e AntiPadrões J2EE da Universidade Estadual do Ceará. Foi realizada uma análise sobre a perspectiva de seis padrões J2EE (Intercepting Filter, Composite View, Session Facade, Application Service, Data Access Object e Domain Store) que foram implementados durante o processo de desenvolvimento em duas diferentes aplicações corporativas (ReservaOnline e Palpiteiro).*

1. Presentation Tier - Intercepting Filter

1.1. Cenário

Em um sistema web JEE referente a uma rede de hotéis, há vários hotéis que utilizarão o mesmo sistema para cadastro de clientes, reservas e etc. Um determinado usuário ao acessar o sistema web é identificado por um perfil, seja ele usuário administrador ou usuário cliente, esse perfil consiste em uma associação das suas respectivas permissões de usuário à um determinado hotel que pertece a rede de hotéis.

1.2. Problema

Ao efetuar a autenticação do usuário no sistema web devemos identificar dentro do seu perfil quais são as suas permissões e quais hoteis terá acesso para o redirecionamento adequado para página do hotel.

1.3. Justificativa

O padrão intercepting Filter foi necessário para identificar através da requisição o hotel presente na sessão do usuário realizarmos a busca e armazenamento na sessão de um determinado hotel a partir da requisição a busca de um determinado hotel.

2. Presentation Tier - Composite View

2.1. Cenário

Em um sistema web JEE referente a uma rede de hotéis, o usuário poderá navegar pela aplicação utilizando uma barra de navegação. Essa barra será diferente para cada perfil de usuário, caso ele seja um usuário administrador poderá navegar por páginas administrativas, caso ele seja um usuário cliente, poderá realizar reservas, visualizar os quartos disponíveis, alterar seus dados e todas as opções referentes a pagamento.

2.2. Problema

Todas as páginas do sistema seguirão um layout predefinido, mas há diferenças em cada perfil de usuário haverá algumas modificações em cada página específica para cada perfil. As alterações são distintas e torna-se praticamente inviável modificar página por página.

2.3. Justificativa

Para viabilizar a manutenibilidade e a escalabilidade da aplicação J2EE o padrão Composite View irá ser aplicado primeiramente nas diferenças entre perfis, aplicando uma estrutura para cada perfil, seja ela uma pagina de clientes ou pagina de administradores, logo após a separação pelo perfil o padrão irá atuar na fragmentação de cada página em áreas como por exemplo, cabeçalho(header) ,barra de navegação(navbar), corpo da página(body) e rodapé(footer).

3. Business Tier - Session Facade

3.1. Cenário

Sistema J2EE que tem como objetivo gerenciar palpites de jogos de futebol. Para este exemplo será dado ênfase ao procedimento de cadastrar um novo usuário no sistema.

3.2. Problema

Como um palpite pode ser realizado de diversas fontes (smartphones, computadores e tablets) existe o risco de parte da lógica de negócio ficar espalhado nas soluções que ficam no lado do cliente. Essa situação contribui para um alto acoplamento dos componentes dificultando a manutenção do sistema.

3.3. Justificativa

Neste caso o padrão Session Facade irá criar uma camada a fim de centralizar e simplificar as requisições para as regras de negócio do servidor. Com isso o cliente não precisará ter conhecimento dos métodos e objetos específicos de negócio do sistema. A fim de facilitar utilização deste padrão, é sugerido a adoção também os padrões Transfer Object(TO) e Application Service.

4. Business Tier - Application Service

4.1. Cenário

Sistema J2EE que tem como objetivo gerenciar palpites de jogos de futebol. Para este exemplo será dado ênfase ao procedimento de cadastrar um novo usuário no sistema.

4.2. Problema

No sistema é necessário criar componentes que representem as informações de negócio. Por exemplo, para manipular as informações do usuário pode ser adotado o padrão Business Object (BO). Contudo utilizando-se deste padrão para centralizar todas as regras de manipulação desta informação, acaba-se criando um componente muito grande que irá possuir um alto acoplamento entre as regras de negócio e a informação em questão. Isso dificulta bastante a reutilização e manutenção do código.

4.3. Justificativa

Com a utilização do padrão Application Service cria-se uma separação entre as informações de negócio e a manipulação destes. Os objetos BO ficam especializados na representação da informação enquanto que o Application Service enfoca na manipulação destes objectos segundo as regras de negócio do sistema. Essa organização facilita a manutenção e o reaproveitado do código por outros Applications Services. Para um melhor uso deste padrão, é recomendado utilizar também dos padrões Transfer Object, Session Facade e Business Object.

5. Integration Tier - Data Access Object

5.1. Cenário

O sistema de reservas precisa armazenar as informações referentes aos seus clientes e aos seus serviços. Para isso, o modelo objeto relacional irá atrelar cada representação de Objeto(Entidade) específico como uma tabela no banco de dados. Um cliente será representado por uma entidade que possui atributos como nome, idade e endereço que precisam consequentemente ser (persistidas) armazenadas em um banco de dados.

5.2. Problema

Um sistema referente a uma uma rede de hotéis possui uma grande diversidade de hoteis e consequentemente de clientes e serviços. Dentro dessa diversidade as operações de persistência, atualização, leitura e exclusão serão aplicadas a várias entidades referentes a diferentes hotéis do sistema.

5.3. Justificativa

O padrão Data Access Object além de atuar como uma abstração para prover a reutilização das operações básicas em um banco de dados relacional, também conhecida como CRUD(Create,Read,Update,Delete), irá atuar na especialização das operações de cada entidade relacionada. Dentro do nosso escopo podemos exemplificar um DAOGenerico que irá usar uma Entidade específica e realizar as operações definidas na interface genérica.

6. Integration Tier - Domain Store

6.1. Cenário

Sistema J2EE que tem como objetivo gerenciar palpites de jogos de futebol. Para este exemplo será dado ênfase ao procedimento de cadastrar um novo usuário no sistema

6.2. Problema

Para salvar as informações do usuário, o sistema possui um gerenciador de banco de dados. Por conta disso o Business Object possui dentro dele informações de como realizar esta persistência. Por exemplo, ele terá informações como: qual gerenciador de banco de dados adotado, como se conectar a este gerenciador e comandos específicos para manipular os dados neste gerenciador. Desta forma o Business Object além de ficar com mais responsabilidade, ele fica muito acoplado a forma de persistência dificultando a manutenção do sistema.

6.3. Justificativa

O padrão Domain Store é utilizado neste caso para separar as responsabilidades envolvidas: representação dos dados para o sistema e procedimentos de como persistir estes dados no banco de dados adotado. Com essa divisão tem-se um maior desacoplamento entre os componentes facilitando o reúso e manutenção do código. Por exemplo, para alterar o gerenciador de banco de dados adotado não será necessário mudar o Businnes Object associado.

7. Diagramas de Classe e Trechos de Código

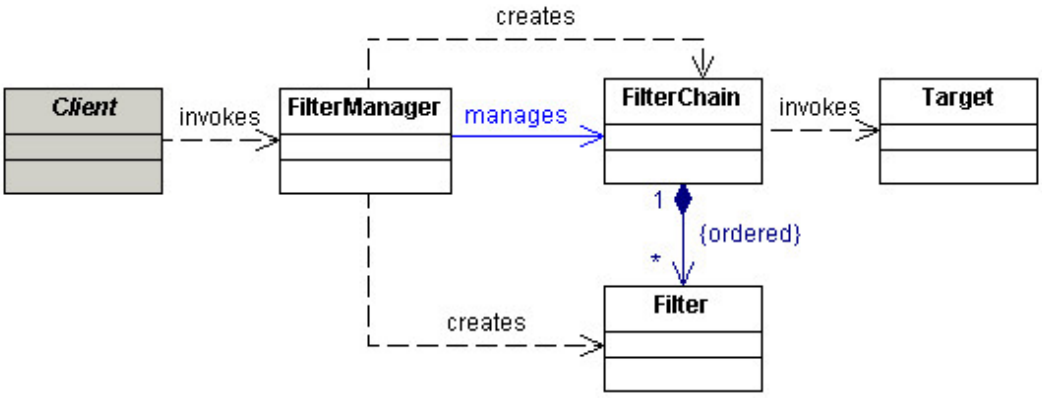


Figura 1. Diagrama - Intercepting Filter

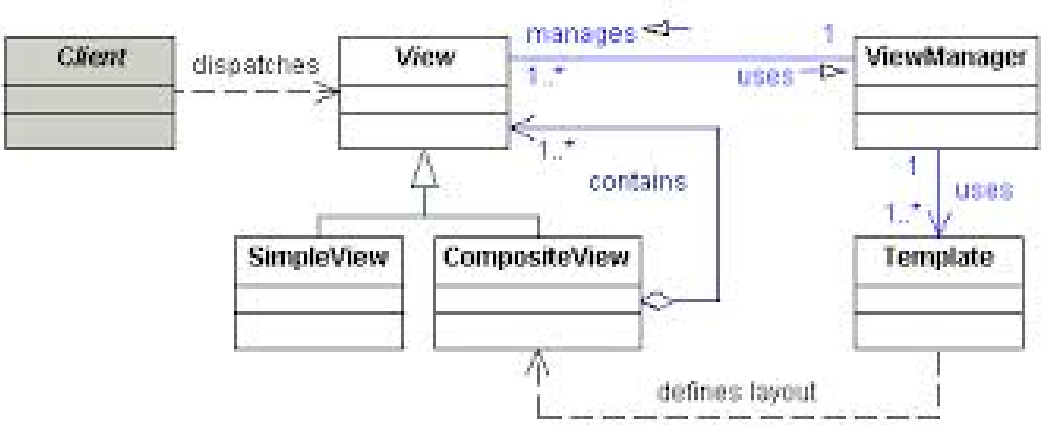


Figura 2. Diagrama - Composite View

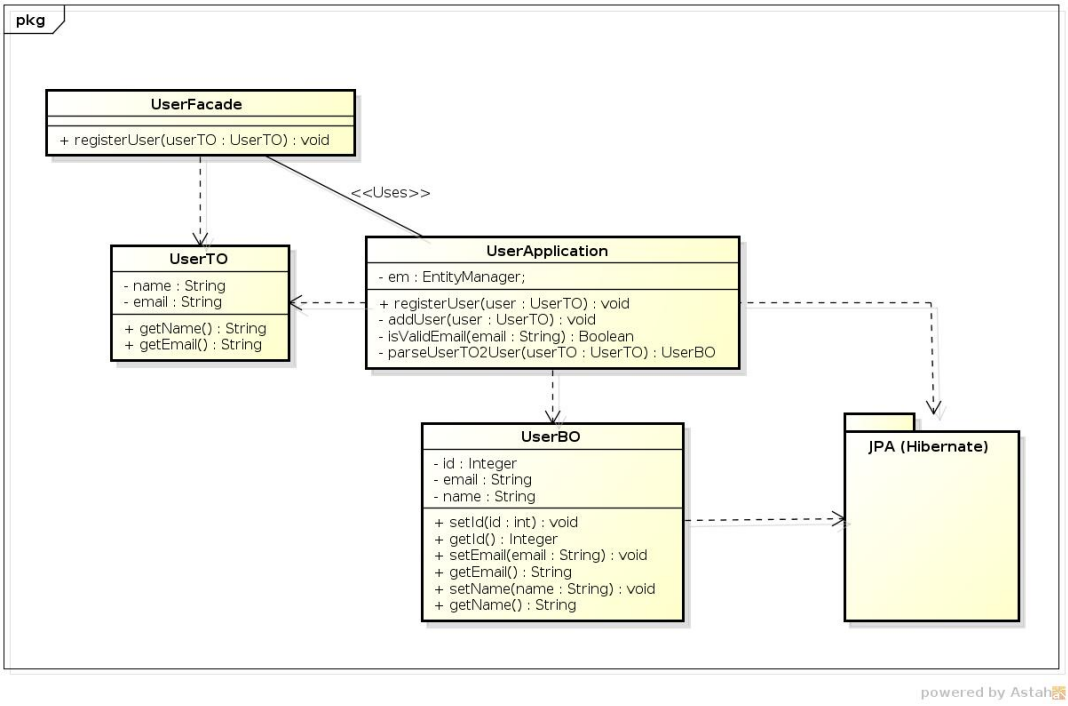


Figura 3. Diagrama - Application Service, Session Facade e Domain Store

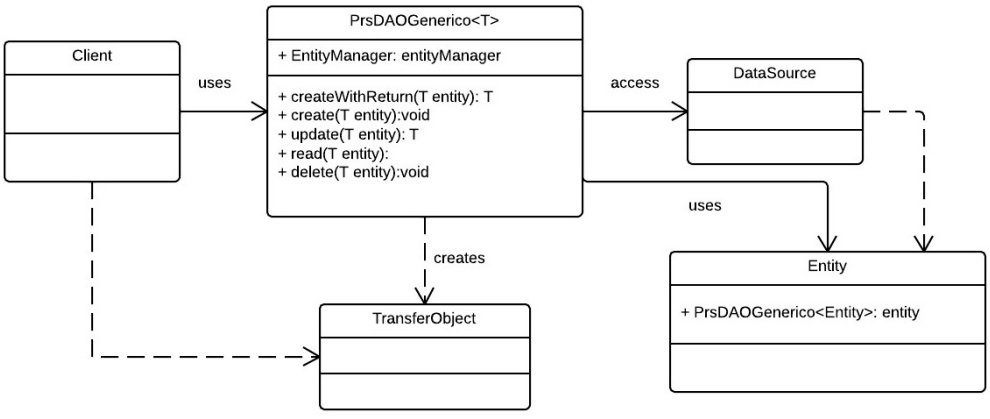


Figura 4. Diagrama - DAO

```java
public class FiltroValidaHotelSessao implements Filter {

    @Autowired
    UsuarioRepository usuarioRepository;
    @Autowired
    HotelImplRepository hotelRepository;
    @Autowired
    ClientePessoaFisicaRepository clienteRepository;

    @Override
    public void init(FilterConfig filterConfig) throws ServletException { }

    @Override
    public void destroy() { }

    @Override
    public void doFilter(ServletRequest request, ServletResponse response,
            FilterChain chain) throws IOException, ServletException {
        HttpServletRequest httpRequest = (HttpServletRequest) request;
        String path = httpRequest.getRequestURI();
        String serverName = httpRequest.getServerName();

        // Busca o id do hotel a partir da URL
        if (httpRequest.getSession().getAttribute("hotel") == null) {
            Properties props = System.getProperties();
            props.load(Email.class.getClassLoader().getResourceAsStream("hoteis.properties"));
            String hotelId = props.getProperty(serverName);
            if (hotelId != null) {
                HotelImpl hotel = hotelRepository.recuperar(Long.parseLong(hotelId));
                httpRequest.getSession().setAttribute("hotel", hotel);
            }
        }
        httpRequest.setAttribute("allowLocalization", true);
        httpRequest.setAttribute("userRole", "USER");
```

Figura 5. Code - Intercepting Filter

```
193                 <br />
194                 <button type="button" class="btn btn-success" onclick="changeEmail();"><fmt:message key="user.info.change.email" /></button>
195             </div>
196         </div>
197       </form>
198     </div>
199   </div>
200 </div>
201
202 <jsp:include page="commonFooter.jsp" />
203
```

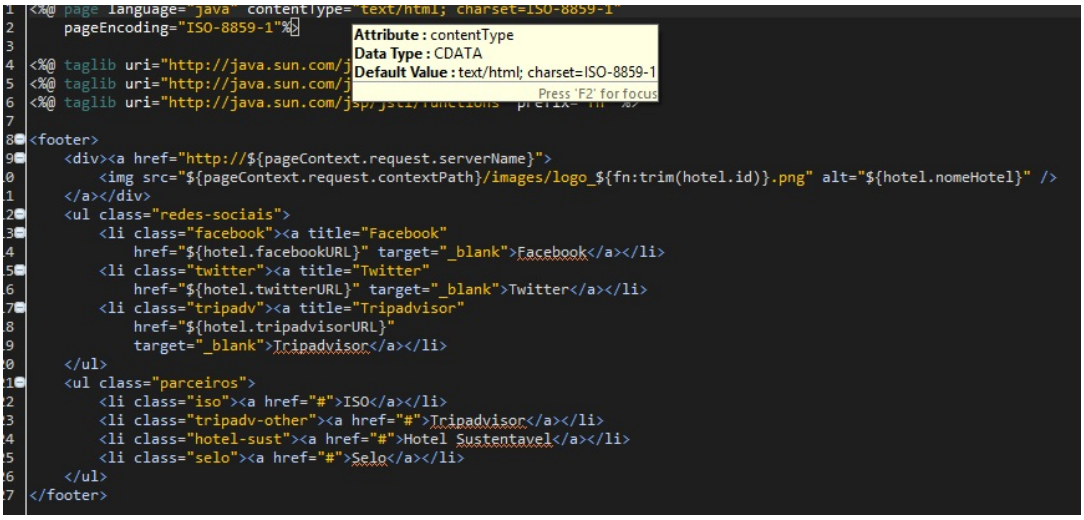
Figura 6. Code - Composite View

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

<jsp:include page="../template/commonInternal.jsp">
    <jsp:param name="pageNameKey" value="menu.client" />
</jsp:include>


<script src="${pageContext.request.contextPath}/resources/js/cliente/cliente.js"></script>

<fmt:setLocale value="${localeCode}"/>
<fmt:setBundle basename="reservaonline"/>

<!-- FORMULÁRIO DE CADASTRO -->
<div class="container" id="containerCadastro">
 <div id="myTabContent" class="tab-content">
```

Figura 7. Code - Composite View

```
<%@ page language="java" contentType="text/html" %>
<%@ page isELIgnored="false" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

<c:if test="${not empty param.currentLocale}">
    <c:set var="localeCode" value="${param.currentLocale}" scope="session" />
</c:if>
<c:set var="localeCode" value="${sessionScope.localeCode}" />

<jsp:useBean id="hotel" class="br.com.ifactory.model.HotelImpl" scope="session" />
<c:set var="localeCode" value="${sessionScope.localeCode}" />
<fmt:setLocale value="${localeCode}" />
<fmt:setBundle basename="reservaonline" />

<!DOCTYPE html>

<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />

    <meta http-equiv="Pragma" content="no-cache" />
    <meta http-equiv="Cache-Control" content="no-cache" />
    <meta http-equiv="Expires" content="Sat, 01 Dec 2001 00:00:00 GMT" />

    <link rel="stylesheet" type="text/css" href="${pageContext.request.contextPath}/resources/css/global.css" />
    <link rel="stylesheet" type="text/css" href="${pageContext.request.contextPath}/resources/bootstrap/css/bootstrap.css" />
    <link rel="stylesheet" type="text/css" href="${pageContext.request.contextPath}/resources/bootstrap/css/bootstrap-responsive.css" />
    <link rel="stylesheet" type="text/css" href="${pageContext.request.contextPath}/resources/bootstrap/css/datepicker.css" />
    <link rel="stylesheet" type="text/css" href="${pageContext.request.contextPath}/resources/css/dropzone.css" />
    <link rel='stylesheet' type="text/css" href='${pageContext.request.contextPath}/resources/fullcalendar/fullcalendar.css' />
    <link rel='stylesheet' type="text/css" href='${pageContext.request.contextPath}/resources/fullcalendar/fullcalendar.print.css' media='print' />
```

Figura 8. Code - Composite View



```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>

<%@ taglib uri="http://java.sun.com/j
<%@ taglib uri="http://java.sun.com/j
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>

<footer>
    <div><a href="http://${pageContext.request.serverName}">
        <img src="${pageContext.request.contextPath}/images/logo_${fn:trim(hotel.id)}.png" alt="${hotel.nomeHotel}" />
    </a></div>
    <ul class="redes-sociais">
        <li class="facebook"><a title="Facebook"
            href="${hotel.facebookURL}" target="_blank">Facebook</a></li>
        <li class="twitter"><a title="Twitter"
            href="${hotel.twitterURL}" target="_blank">Twitter</a></li>
        <li class="tripadv"><a title="Tripadvisor"
            href="${hotel.tripadvisorURL}"
            target="_blank">Tripadvisor</a></li>
    </ul>
    <ul class="parceiros">
        <li class="iso"><a href="#">ISO</a></li>
        <li class="tripadv-other"><a href="#">Tripadvisor</a></li>
        <li class="hotel-sust"><a href="#">Hotel Sustentavel</a></li>
        <li class="selo"><a href="#">Selo</a></li>
    </ul>
</footer>
```

Attribute : contentType
Data Type : CDATA
Default Value : text/html; charset=ISO-8859-1
Press 'F2' for focus

Figura 9. Code - Composite View

```
<%@ taglib uri="http://www.springframework.org/security/tags"
    prefix="sec"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>

<jsp:include page="../../template/commonInternal.jsp">
    <jsp:param name="pageNameKey" value="book.room" />
</jsp:include>

<fmt:setLocale value="${localeCode}" />
<fmt:setBundle basename="reservaonline" />

<script
    src="${pageContext.request.contextPath}/resources/js/reservas/efetuarReserva.js">
</script>

<div class="container" id="containerReserva">
    <form id="formReserva" class="form-horizontal" method="POST">
```

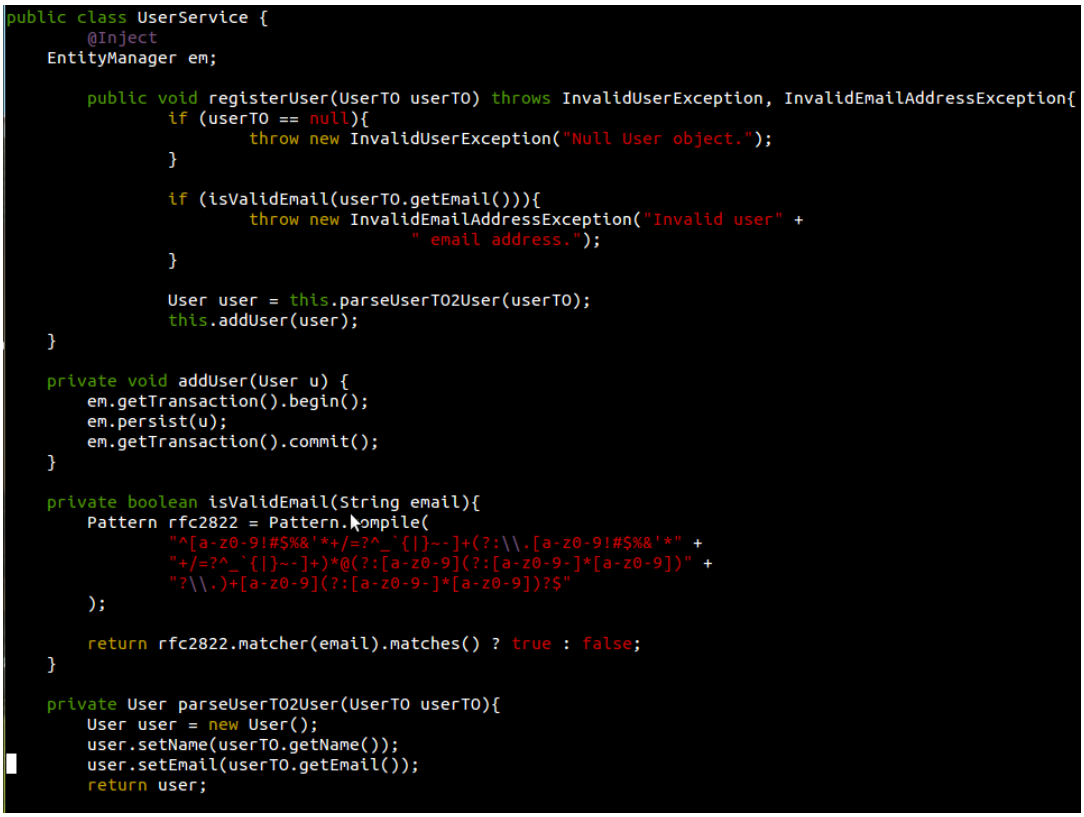
Figura 10. Code - Composite View

```
190     </div>
191     <div id="erroCadastro" class="alert alert-error" style="display: none;">
192         <strong><fmt:message key="geral.error" /></strong>
193         <fmt:message key="geral.error.msg" />
194     </div>
195 </div>
196 
197 <jsp:include page="../commonFooter.jsp" />
```

Figura 11. Code - Composite View

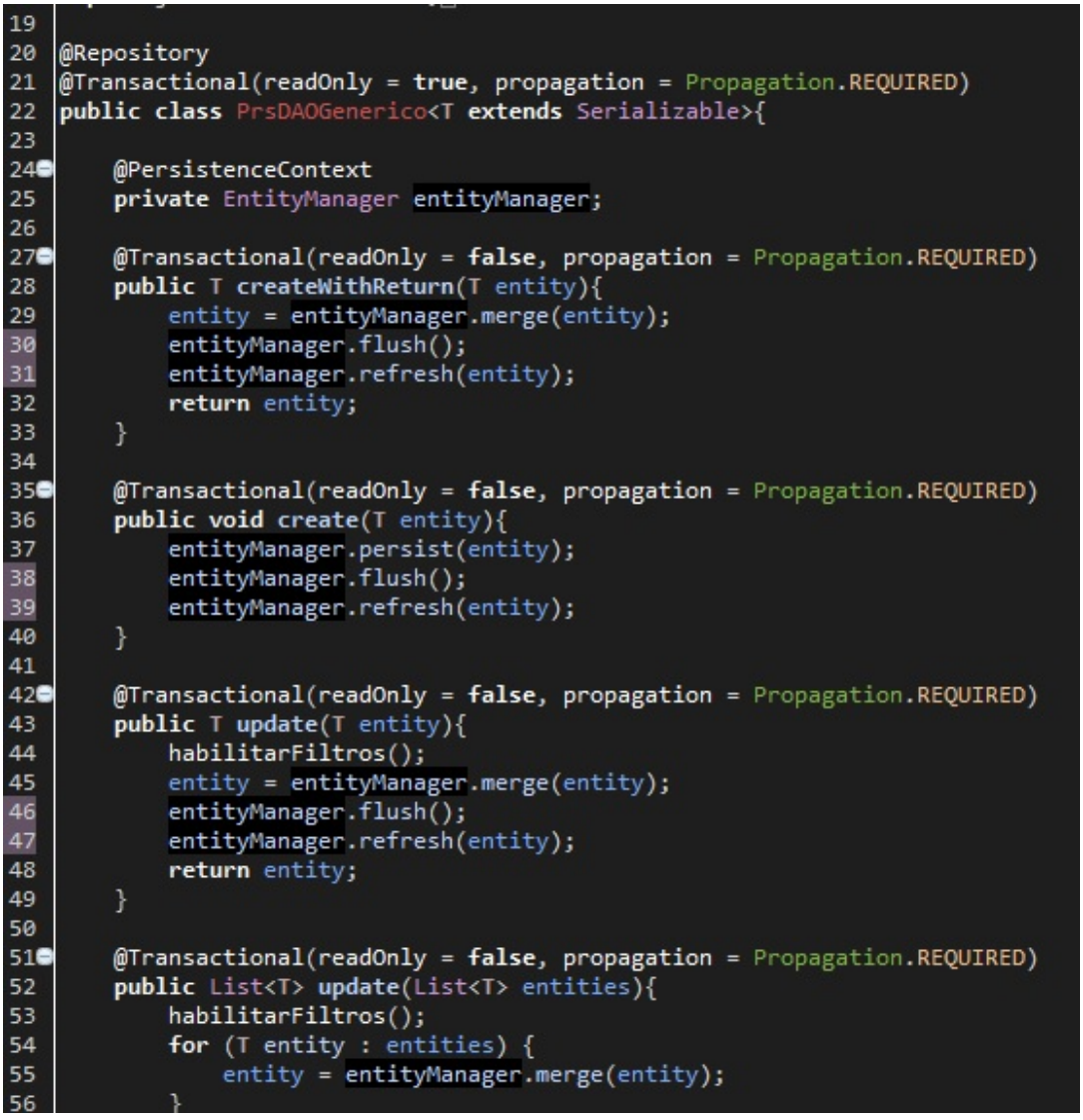
```
package com.joaovianey.palpiteiro.facades;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;

import com.joaovianey.palpiteiro.exceptions.InvalidEmailAddressException;
import com.joaovianey.palpiteiro.exceptions.InvalidUserException;
import com.joaovianey.palpiteiro.services.UserService;
import com.joaovianey.palpiteiro.transferobjects.UserTO;

/**
 * Session Bean implementation class UserFacade
 */
@Stateless
@LocalBean
public class UserFacade {
        private UserService userService;

    /**
     * Default constructor.
     */
    public UserFacade() {
        userService = new UserService();
    }

    public void registerUser(UserTO userTO) throws InvalidUserException, InvalidEmailAddressException{
        userService.registerUser(userTO);
    }

}
```

Figura 12. Code - Session Facade



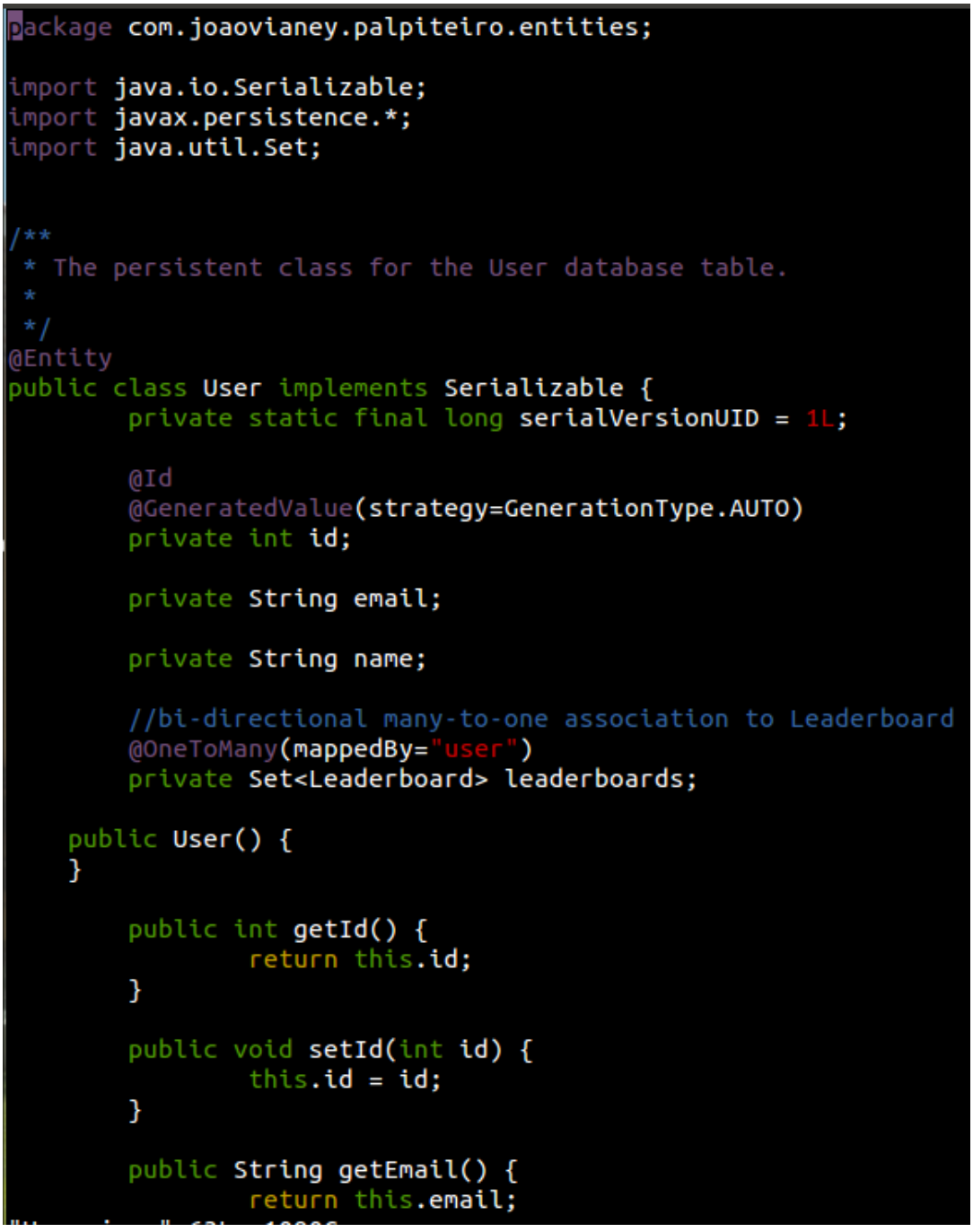
```
public class UserService {
        @Inject
    EntityManager em;

        public void registerUser(UserTO userTO) throws InvalidUserException, InvalidEmailAddressException{
                if (userTO == null){
                        throw new InvalidUserException("Null User object.");
                }

                if (isValidEmail(userTO.getEmail())){
                        throw new InvalidEmailAddressException("Invalid user" +
                                        " email address.");
                }

                User user = this.parseUserTO2User(userTO);
                this.addUser(user);
    }

    private void addUser(User u) {
        em.getTransaction().begin();
        em.persist(u);
        em.getTransaction().commit();
    }

    private boolean isValidEmail(String email){
        Pattern rfc2822 = Pattern.compile(
                "^[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\\.[a-z0-9!#$%&'*" +
                "+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])" +
                "?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?$"
        );

        return rfc2822.matcher(email).matches() ? true : false;
    }

    private User parseUserTO2User(UserTO userTO){
        User user = new User();
        user.setName(userTO.getName());
        user.setEmail(userTO.getEmail());
        return user;
```

Figura 13. Code - Application Service



```
19
20  @Repository
21  @Transactional(readOnly = true, propagation = Propagation.REQUIRED)
22  public class PrsDAOGenerico<T extends Serializable>{
23
24      @PersistenceContext
25      private EntityManager entityManager;
26
27      @Transactional(readOnly = false, propagation = Propagation.REQUIRED)
28      public T createWithReturn(T entity){
29          entity = entityManager.merge(entity);
30          entityManager.flush();
31          entityManager.refresh(entity);
32          return entity;
33      }
34
35      @Transactional(readOnly = false, propagation = Propagation.REQUIRED)
36      public void create(T entity){
37          entityManager.persist(entity);
38          entityManager.flush();
39          entityManager.refresh(entity);
40      }
41
42      @Transactional(readOnly = false, propagation = Propagation.REQUIRED)
43      public T update(T entity){
44          habilitarFiltros();
45          entity = entityManager.merge(entity);
46          entityManager.flush();
47          entityManager.refresh(entity);
48          return entity;
49      }
50
51      @Transactional(readOnly = false, propagation = Propagation.REQUIRED)
52      public List<T> update(List<T> entities){
53          habilitarFiltros();
54          for (T entity : entities) {
55              entity = entityManager.merge(entity);
56          }
```

Figura 14. Code - DAO



```java
package com.joaovianey.palpiteiro.entities;

import java.io.Serializable;
import javax.persistence.*;
import java.util.Set;


/**
 * The persistent class for the User database table.
 *
 */
@Entity
public class User implements Serializable {
	private static final long serialVersionUID = 1L;

	@Id
	@GeneratedValue(strategy=GenerationType.AUTO)
	private int id;

	private String email;

	private String name;

	//bi-directional many-to-one association to Leaderboard
	@OneToMany(mappedBy="user")
	private Set<Leaderboard> leaderboards;

    public User() {
    }

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

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

	public String getEmail() {
		return this.email;
```

Figura 15. Code - Domain Store