INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA SUL-RIO-GRANDENSE - IFSUL, CAMPUS PASSO FUNDO
CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

WILLIAM BRAGA DE LIMA

SISTEMA DE GERENCIAMENTO DE RECURSOS ESPORTIVOS

PASSO FUNDO
2015

WILLIAM BRAGA DE LIMA

SISTEMA DE GERENCIAMENTO DE RECUSROS ESPORTIVOS

Monografia apresentada ao Curso de Tecnologia em Sistemas para Internet do Instituto Federal Sul-rio-grandense, Campus Passo Fundo, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet.

Orientador: Jorge Luís Boeira Bavaresco

PASSO FUNDO
2015

WILLIAM BRAGA DE LIMA

SISTEMA DE GERENCIAMENTO DE RECURSOS ESPORTIVOS

Trabalho de Conclusão de Curso aprovado em ____/____/____ como requisito parcial para a obtenção do título de Tecnólogo em Sistemas para Internet

Banca Examinadora:

_____

Prof. Me. Jorge Luis Boeira Bavaresco (Orientador)

_____

Prof. Dr. Josué Toebe (Convidado)

_____

Prof. Willian Bouviér (Convidado)

_____

Prof. Me. Adilso Nunes de Souza

Coordenação do Curso

PASSO FUNDO

2015

AGRADECIMENTOS

Agradeço a minha mãe, Luciane Taufer Braga, e ao meu pai, Marcelo Francisco de Lima, por sempre estarem presentes em todos os momentos da minha vida e por sempre estarem apoiando, incentivando e ajudando nas minhas escolhas.

Agradeço ao professor Jorge Luis Boeira Bavaresco pela contribuição fundamental para que eu pudesse realizar este trabalho, tirando todas as minhas dúvidas e auxiliando nos momentos necessários.

Agradeço aos meus colegas que iniciaram o curso comigo e sempre motivaram e ajudaram a concluir mais uma etapa da minha vida. Também agradeço pelos diversos momentos de descontração e de estudo que tivemos ao longo desses três últimos anos, fazendo com que deixássemos de ser apenas colegas de faculdade para nos tornarmos grandes amigos.

Agradeço aos demais professores que proporcionaram um ensino de qualidade e contribuíram para meu aprendizado nesses anos que passei na instituição.

EPÍGRAFE

“Sabe viver quem dedica um dia ao sonho e outro a realidade.”

Kurt Cobain

RESUMO

Com o crescimento constante da tecnologia no cotidiano das pessoas auxiliando-as em suas tarefas, a utilização de sistemas informatizados nas empresas tem se tornado indispensável. Este trabalho tem como objetivo realizar uma análise sobre o contexto atual das quadras esportivas e por meio disso, desenvolver um sistema de gerenciamento de recursos esportivos para web. Tal sistema visa auxiliar os funcionários nas tarefas diárias do trabalho, suprindo as necessidades encontradas nesse ramo de negócios. A aplicação foi desenvolvida na plataforma Java EE, utilizando o banco de dados *PostgreSQL*, JPA com o servidor de persistência *Hibernate*, API *JavaServer Faces* e a biblioteca de componentes *Primefaces.*

Palavras-chave: Java EE, sistema web, análise e desenvolvimento de sistemas

ABSTRACT

With the steady growth of technology in daily life helping people in their tasks, the use of computerized systems in companies has become indispensable. This research aims to conduct an analysis of the current context of sports fields and thereby, develop a web management system of sports resources. The system is intended to assist employees in their daily work tasks, supplying the needs found in this business. The application was developed in Java EE platform, using the PostgreSQL database, JPA with Hibernate persistence Server, JavaServer Faces API and the component library of Primefaces.

Palavras-chave: Java EE, web system, analysis and systems development

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIAÇÕES E DE SIGLAS

JVM – *Java Virtual Machine* – Máquina Virtual Java

API – *Application Programming Interface* - Interface de Programação de Aplicações

JSE – *Java Standard Edition*

JME – *Java Micro Edition*

JEE – *Java Enterprise Edition*

JSR – *Java Specification Requests*

JSP – *JavaServer Pages*

JSF – *Java Server Faces*

EJB – *Enterprise JavaBeans*

EIS– *Enterprise Information System*

JDBC – *Java Database Connectivity* – Conexão de Banco de Dados Java

JCA – *Java Connector Architecture* – Arquitetura de Conectores Java

JTA – *Java Transaction API* – API de Transações Java

JPA – *Java Persistence API* – API de Persistência Java

JMS – *Java Message Service*

JNDI – *Java Naming and Directory Interface*

JAF – *JavaBeans Activation Framework*

XML – *Extensible Markup Language* – Linguagem Extensível de Marcação

XHTML – *eXtensible Hypertext Markup Language* – Linguagem Extensível de Marcação Hipertexto

JCP – *Java Community Process*

SQL - *Structured Query Language* – Linguagem de Consultas Estruturada

POJO – *Plain Old Java Objects* - Velho e Simples Objeto Java

JPQL – *Java Persistence Query Language* – Linguagem de Persistência de Consultas Java

UML – *Unified Modeling Language* – Linguagem de Modelagem Unificada

OOSE – *Object-oriented software engineering* – Engenharia de Software Orientada a Objetos

OMT– *Object-modeling technique* – Técnica de modelagem de objeto

SUMÁRIO

1 INTRODUÇÃO

A tecnologia está presente em nosso dia-a-dia sempre auxiliando-nos a realizar as tarefas mais simples até mesmo as mais complexas. Desta forma, a utilização de sistemas informatizados está se tornando praticamente indispensável, já que facilitam a realização de tarefas que, por muitas vezes, se tornam trabalhosas. A utilização da tecnologia traz diversos benefícios, podendo agilizar certas atividades.

Porém, certos lugares ou ramos de negócios utilizam muito pouco esse potencial que a tecnologia pode proporcionar. As quadras esportivas, em sua grande maioria, realizam o controle de pagamentos e horários ainda de forma manual, sem quase nenhum auxilio tecnológico. Isso pode trazer diversas dificuldades, já que os dados podem não ser armazenados de forma segura e até mesmo de maneira imprecisa.

Dessa forma, este projeto tem como objetivo realizar o desenvolvimento de um sistema que auxilie no gerenciamento de recursos esportivos, podendo realizar o controle de horários, pagamentos, recursos disponíveis, cadastro de clientes, estoque e venda de produtos de forma eficiente e segura.

2 TEMA

Desenvolvimento de um sistema de gerenciamento de recursos esportivo para web.

2.1 DELIMITAÇÃO DO TEMA

Desenvolvimento de um sistema de gerenciamento de recursos esportivo para web utilizando a linguagem de programação Java.

2.2 PROBLEMA A SER ABORDADO

Como realizar o gerenciamento de uma quadra esportiva de maneira mais eficiente e de fácil acesso a informações importantes?

2.3 HIPÓTESES

Com desenvolvimento de um sistema de gerenciamento esportivo, os responsáveis pelas quadras poderão optar por realizar o controle de seu negócio de forma precária. O sistema poderá auxiliar de forma eficiente a controlar os horários de jogos, os pagamentos e o estoque de produtos, armazenando todos os dados de forma segura e permitindo acesso a informações de maneira mais fácil e rápida.

2.4 OBJETIVOS

2.4.1 Objetivo Geral

Desenvolver um sistema de gerenciamento de recursos esportivos para web que permita realizar o controle de horários, cadastro de clientes, pagamentos, estoque e venda de produtos. Além disso, garantir que as informações sejam armazenadas de forma segura e sejam acessadas de forma ágil.

2.4.2 Objetivos Específicos

- Realizar um estudo sobre a linguagem de programação Java para web e suas tecnologias.
- Análise e desenvolvimento de um sistema para web utilizando a linguagem de programação estudada.
- Utilização das tecnologias Java EE, *Java Server Faces* e *Java Persistence* API.
- Estudo e utilização da biblioteca de componentes *Primefaces.*

2.5 JUSTIFICATIVA

A tecnologia está em constante crescimento e presente em nosso dia-a-dia. Grande parte das empresas possuem sistemas que facilitam e agilizam seus trabalhos. Apesar disso há certos ramos que ainda não possuem algum sistema informatizado que possa auxiliar no trabalho dos funcionários, como por exemplo, em quadras esportivas. Visto que o controle de negócios destes locais ainda é feitos de forma manual e sem a ajuda de alguma tecnologia, o desenvolvimento de um sistema que auxilie no gerenciamento pode permitir o controle mais eficiente e facilitar o acesso a informações importantes do negócio.

3 REFERENCIAL TEÓRICO

Neste capítulo serão explicados os principais conceitos das tecnologias e ferramentas que irão ser utilizados no desenvolvimento do trabalho.

3.1 JAVA

Java pode ser definido como uma tecnologia, linguagem de programação e uma plataforma de desenvolvimento muito poderosa e possuindo diversos recursos que possibilitam o desenvolvimento de aplicações voltadas para desktop, web, empresas e dispositivos móveis, suportando também a programação orientada a objetos. Sua primeira versão foi lançada em 1997 pela empresa *Sun Microsystems*.

A plataforma Java possui dois componentes principais: a *Java Virtual Machine* (JVM) e a *Java Application Programming Interface* (Java API), ambos disponibilizando funcionalidades comuns. A máquina virtual Java (JVM) atua como um simulador de computador possuindo suas próprias características como memória e processamento. A Java API, por sua vez, é um conjunto de bibliotecas que tem como objetivo disponibilizar diversas funcionalidades para o desenvolvedor, facilitando a programação (COSTA, 2008).

Conforme Costa (2008), a partir da versão 1.2 lançada em 1998, o Java passou a ter três edições: JSE (*Java Standard Edition*), JME (*Java Micro Edition*) e o JEE (*Java Enterprise Edition*). A edição SE é voltada para o desenvolvimento de aplicações desktop e servidoras, já a ME é utilizada no desenvolvimento de aplicações para dispositivos móveis. A edição EE tem como objetivo desenvolver aplicações corporativas e para web, e será melhor descrita na próxima seção.

3.2 JAVA EE

Publicado pela *Sun* e pelo *Java Community Process* (JSRs), o *Java Enterprise Edition* é baseado num conjunto de especificações utilizada com o foco para o desenvolvimento de aplicações distribuídas corporativas. Além disso, essa tecnologia disponibiliza os serviços utilizados para se desenvolver a aplicação desejada. Com o Java EE, o tempo de desenvolvimento é reduzido, já que o programador escreve menos código, diminuindo

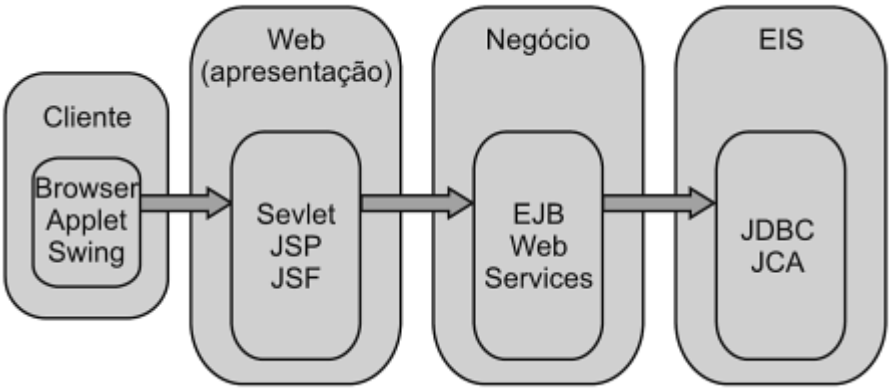
também os riscos da implementação do projeto e os problemas com a manutenção da aplicação (SAMPAIO, 2011).

Conforme Jendrock Et al. (2011), o desenvolvimento de aplicações corporativas se torna muito mais fácil e rápido ao se utilizar o *Java Enterprise Edition*. Isso ocorre, pois essa plataforma disponibiliza diversas *APIs* que auxiliam o desenvolvedor na criação de uma aplicação.

A plataforma Java EE permite definir restrições de segurança em tempo de execução. Além disso, oferece aos desenvolvedores uma ampla variedade de implementações de segurança sem deixa-lo preocupado com a complexidade de se implantar esses recursos. O Java EE já disponibiliza regras de controle de acesso que são definidas pelo desenvolvedor e interpretadas quando a aplicação é executada no servidor. Outro serviço de segurança oferecido pela plataforma é um mecanismo de login (JENDROCK Et al., 2011).

O Java EE possui um modelo de programação em containers que são responsáveis por fornecer diversos serviços que serão utilizados no desenvolvimento de uma aplicação e podem ser descritas por Sampaio (2011) da seguinte forma:

Figura 1: Modelo de Programação do Java EE



Fonte: Sampaio, 2011

- Cliente: É onde está localizado o browser ou *applets*, além de poder estar rodando aplicações Java dentro desse container.
- Web: Nessa camada estão localizados alguns componentes como o *JavaServer Pages* (JSP), os *Servlets* e o *JavaServer Faces* (JSF).

- Negócio: Essa camada possui componentes que implementam as regras de negócio da aplicação ou podem representar entidades de dados, os *Enterprise JavaBeans* (EJB) e os *Web Services*.
- Enterprise Information Systems (EIS): Nessa camada ficam localizados os servidores de banco de dados ou *Mainframes* que podem ser utilizados através das interfaces *Java DataBase Connectivity* (JDBC) e *Java Connector Architeture* (JCA).

Esse modelo utilizado pelo Java EE fornece para cada tipo de aplicação um determinado container. Por exemplo, aplicações para web são implementadas em um Conteiner Web, já as aplicações clientes são executadas no Conteiner Cliente. Todos os detalhes básicos da execução das aplicações são de responsabilidade dos serviços e APIs do contêiner utilizado, dessa forma o desenvolvedor preocupa-se apenas com a funcionalidade que deseja utilizar (SAMPAIO, 2011).

Algumas tecnologias e ferramentas que fazem parte do Java EE são:

- Java Transacion API (JTA)
- Java Persistence API (JPA)
- Java Message Service (JMS)
- Java Naming and Directory Interface (JNDI)
- Java Mail
- JavaBeans Activation Framework (JAF)
- Processamento de XML
- Java EE Connector Architecture (JCA)
- Serviços de segurança
- Serviços WEB

3.3 JSF

O *Java Server Faces* é definido como um modelo de programação voltado à criação de páginas web orientado a eventos. Possui um conjunto de API's que permitem validar entrada de dados, manipular eventos, definir e controlar a navegação em páginas web, entre outros recursos. Além disso, disponibiliza diversas *tags* que podem ser utilizadas em páginas JSP e XHTML representando componentes visuais (SAMPAIO, 2011).

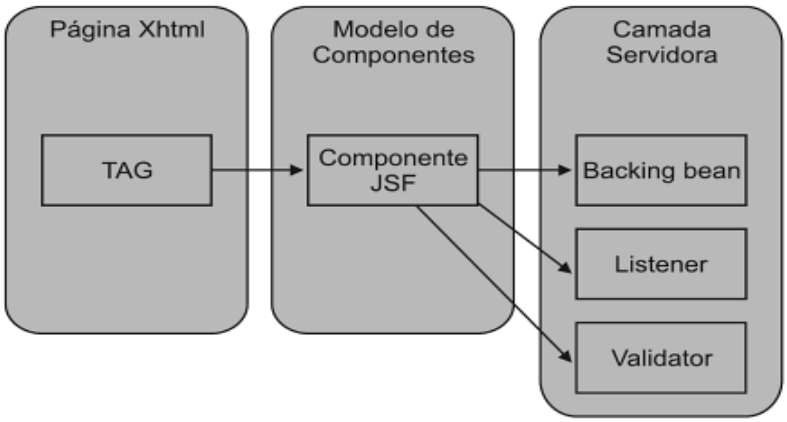
Com o objetivo de facilitar o desenvolvimento de aplicações web diversas companhias começaram a desenvolver frameworks. Em 2001, a *Java Community Process* (JCP), um grupo

composto por empresas como *Apache, Oracle e Sun*, se reuniu para encontrar uma forma de atingir esse objetivo. O resultado disso foi a criação do *JavaServer Faces* lançado em Março de 2004 (BERGSTEN, 2004).

O JSF permite a utilização de bibliotecas de componentes adicionais que oferecem outros recursos além dos que são disponibilizados pela tecnologia por padrão. Alguns exemplos são as bibliotecas *Primefaces* desenvolvida pela empresa *PrimeTek*, a "*Richfaces*" da empresa *Red Hat*, a *ADF Faces* fornecida pela *Oracle* e a *Myfaces/Tomahawk* disponibilizada pelo grupo *Apache Software Foundation* (SAMPAIO, 2011).

De acordo com Sampaio (2011), dentro de uma página JSF são colocados os componentes de uma determinada biblioteca e essa estrutura de componentes formada é chamada de raiz de componentes (*UIViewRoot)*. As propriedades de classes Java servidoras (*Backing Beans)* são associadas aos componentes e estes, por sua vez, são associados a *TAG's* presentes em uma página XHTML. Além disso podem existir outros elementos na camada do servidor como o *Listener*, que serve para responder aos eventos da página e o *Validator*, que é utilizado para validar conteúdo. A figura abaixo ilustra os elementos da aplicação JSF:

Figura 2: Elementos da aplicação JSF



Fonte: Sampaio, 2011

3.4 PRIMEFACES

O *PrimeFaces* é uma biblioteca de componentes utilizada pelo *Java Server Faces* possuindo mais de 90 componentes e sendo umas das bibliotecas mais completas disponível para o JSF 2.0 (LUCKOW;MELO, 2010).

Essa biblioteca inclui diversos componentes *Ajax*, como por exemplo, botões, painéis, calendários, tabelas de dados, efeitos e entre outros. O *PrimeFaces* disponibiliza uma rica quantidade de componentes sofisticados, com alto grau de usabilidade, flexibilidade e também interatividade. Outra característica é a utilização da biblioteca para aplicações em dispositivos móveis (HLAVATS, 2013).

Segundo Hlavats (2013), a biblioteca *Primefaces* é um dos componentes mais populares utilizados pelo JSF e possui a confiança de diversas empresas como a *Boeing*, *Ford*, *Cisco* e *Nvidia*.

Essa biblioteca é utilizada em instituições financeiras, bancos, companhias de softwares, universidades, entre outros locais. Para utilizá-la é necessário adicionar apenas um único aquivo *.jar* no projeto, não sendo necessário realizar configurações (FEITOZA, 2012).

3.5 JPA

O processo de mapear objetos Java para tabelas de banco de dados é chamado de mapeamento objeto relacional. Utilizando o JPA (Java *Persistence* API) o desenvolvedor é capaz de mapear, armazenar, atualizar e também recuperar os dados de um banco de dados relacional para um objeto Java e vice-versa. O JPA permite que o programador possa trabalhar com objetos ao invés de instruções SQL, podendo ser utilizada em aplicações Java EE e Java SE. (VOGEL, 2012).

O JPA surgiu da necessidade da plataforma Java realizar a persistência de dados. Sua primeira versão foi lançada em Maio de 2006, porém possuía alguns problemas já que alguns recursos existentes em outros frameworks não foram inseridos. Já a segunda versão disponibilizada em Dezembro de 2009, trouxe grandes melhorias adicionando mais recursos e atendendo a necessidade de mapeamento de sistemas legados (PAGANINI, 2010).

A versão 2.1 da JPA foi lançada em Maio de 2013 e faz parte da plataforma Java EE 7. Essa nova versão trouxe novas melhorias e recursos, entre elas a geração do esquema de banco de dados, suporte a *Stored Procedures*, mecanismo que permite converter os tipos utilizados nos atributos de entidades e os tipos das colunas que se referem a esses atributos na base de dados (*Converters*), suporte a atualização e exclusão de dados em lote entre outros (GARCIA, 2014).

Medeiros (2013) descreve algumas das funcionalidades do JPA:

- POJOS Persistentes**:** os POJOs (*Plain Old Java Object*) são objetos que possuem design simples que não dependem da herança de interfaces ou classes de frameworks externos. O mapeamento objeto-relacional com JPA dirigido a metadados, podendo ser feito com anotações no código ou através de um XML definido externamente.
- Consultas em Objetos**:** As consultas podem ser realizadas através da linguagem de consulta *Java Persistence Query Language* (JPQL), sendo derivada do EJB QL e transformada depois para SQL.
- Configurações simples**:** as configurações são feitas através de anotações, de um XML ou uma combinação das duas.
- Integração e Teste**:** Atualmente as aplicações normalmente rodam num servidor de aplicação, sendo que os testes nesses servidores são um grande desafio e quase impraticáveis. Porém, isto é resolvido com uma API que trabalha fora do servidor de aplicação. Isto permite que a JPA possa ser utilizada sem a existência de um servidor de aplicação. Sendo assim, os testes podem ser executados com maior facilidade.

3.6 UML

Segundo Booch et al. (2005, p. 13), "a UML, Linguagem Unificada de Modelagem, é uma linguagem gráfica para a visualização, especificação, construção e documentação de artefatos de sistemas complexos de software".

Até o começo da década de 90, as aplicações que estavam sendo desenvolvidas ficavam cada vez mais complexas e, além disso, estavam surgindo diversas linguagens orientadas a objetos. Durante essa época existiam três metodologias de modelagem que se destacavam o método de Booch desenvolvido por Booch, o método OOSE *(Objected-Oriented Software Engineering)* de Jacobson e o método OMT *(Object Modeling Technique)* criado por Rumbaugh. Os três métodos funcionavam separadamente até que no ano de 1995 ocorreu a união do método Booch e OMT visando a evolução da linguagem de modelagem. Então, dessa união surgiu a versão 0.8 da UML e um ano depois foi lançada a versão 0.9 que unia as três metodologias. No ano de 1997 foi liberada a versão 1.0 da UML (BOOCH et al., 2005).

Segundo Booch et al. (2005), a UML é uma linguagem visualização já que cada símbolo utilizado possui uma semântica definida, dessa forma, ao se escrever um modelo,

qualquer outro desenvolvedor ou ferramenta possui a capacidade de interpretá-lo corretamente.

Conforme Melo (2010), a UML 2.0 define treze tipos de diagramas divididos em duas categorias: estruturais e comportamentais, sendo que na versão 2.2 foi acrescentado outro diagrama, o Diagrama de Perfil. Os diagramas estruturais têm como objetivo apresentar as características do sistema que não sofrem alterações com o passar do tempo. Já os diagramas comportamentais demonstra como o sistema responde a determinadas requisições e de que forma ele evolui com o tempo. A figura abaixo mostra os diagramas divididos em suas respectivas categorias:

Figura 3: Diagramas UML

Diagramas Estruturais	Diagrama de Classes Diagrama de Objetos Diagrama de Componentes Diagrama de Pacotes* Diagrama de Implantação Diagrama de Estrutura Composta* Diagrama de Perfil**
Diagramas Comportamentais	Diagrama de Casos de Uso Diagramas de Interação: - Diagrama de Visão Geral* - Diagrama de Sequências - Diagrama Temporal* - Diagrama de Comunicação* Diagrama de Atividades Diagrama de Máquina de Estados

Fonte: Melo, 2010

Ainda conforme Melo (2010), cada diagrama pode ser definido da seguinte forma:

- Diagrama de Classes: elementos conectados por relacionamentos, exibindo entidades do mundo real e elementos de análise e projeto.
- Diagrama de objetos: instância do diagrama de classes. Mostra objetos e valores e apresenta o estado do sistema num determinado tempo.
- Diagrama de componentes: apresenta as dependências entre os componentes de um software.
- Diagrama de Implantação: apresenta a arquitetura do sistema durante a execução, plataformas de hardware e ambientes de software.
- Diagrama de Pacotes: organiza os elementos de modelo e mostra suas dependências
- Diagrama de estrutura composta: mostra a composição de uma estrutura.

- Diagrama de caso de uso: apresenta os casos de uso, os atores e relacionamentos que demonstram as funcionalidades que o sistema deve possuir.
- Diagrama de visão geral: variação do diagrama de atividades mostrando de maneira geral o fluxo de controle dentro do sistema ou processo de negócios.
- Diagrama de sequência: apresenta as interações entre os objetos e a ordem dessas interações.
- Diagrama temporal: apresenta a mudança de estado de um objeto durante o tempo.
- Diagrama de comunicação: mostra os objetos, os inter-relacionamentos e o fluxo de mensagens.
- Diagrama de atividades: demonstra a execução de ações e os fluxos para a conclusão da atividade.
- Diagrama de máquina de estados: mostra as ações ocorridas em resposta ao recebimento de eventos.
- Diagrama de perfil: permite que metaclasses sejam estendidos para se adaptar a diferentes propósitos.

4 METODOLOGIA

Neste capítulo serão descritos as etapas para realizar o desenvolvimento do sistema, apresentando o contexto atual, seus requisitos, os diagramas de caso de uso, classes e de atividades.

4.1 ESTUDO DE CASO

O trabalho tem como principal objetivo o desenvolvimento de um sistema de gerenciamento esportivo para web utilizando a linguagem de programação Java. Nesta seção será apresentado um estudo de caso que abordará o funcionamento das quadras esportivas e as principais necessidades desse ramo de negócios. A realização desse estudo foi baseada no funcionamento de duas empresas que possuem quadras esportivas: Arena Futsal e Brasil Society.

4.1.1 Contexto Atual

O sistema a ser desenvolvido tem como objetivo auxiliar os responsáveis pelas quadras esportivas a terem um controle mais qualificado de seu empreendimento que, por muitas vezes, é realizado de forma precária e ineficiente.

O funcionamento da maioria das quadras esportivas é muito parecido. O cliente interessado em alugar um horário liga ou vai até o estabelecimento. O funcionário responsável atende o cliente e verifica os horários livres para aluguel. Se o cliente estiver satisfeito com algum horário desocupado, o atendente anota o nome e telefone do cliente no horário solicitado.

Geralmente o pagamento do horário alugado é realizado após o término do jogo. O cliente vai até o responsável pelo atendimento e paga o valor referente ao tempo jogado. O atendente confere o pagamento e marca, no mesmo local onde foi anotado o nome do cliente e telefone, que o tempo foi pago. Alguns estabelecimentos ainda oferecem a venda de alimentos e bebidas para seus clientes.

Todo esse processo de aluguel de horários e pagamento é realizado manualmente e na maioria das vezes em uma folha de papel qualquer. Isso pode dificultar o controle eficiente

dos horários e de pagamentos, já que existe um grande risco de perda dessas informações ou até mesmo anotações feitas de forma precária que podem dificultar o entendimento.

Com um sistema de gerenciamento informatizado haverá um melhor controle no que se refere aos horários, pagamentos, cadastro de clientes, venda e controle de produtos no estoque. Além disso, garantirá a seguranças dessas informações e tornará o processo de aluguel e pagamento de horário muito mais ágil e fácil.

4.2 REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

Baseado no estudo de caso foram levantados os requisitos funcionais e não funcionais que estarão presentes no sistema a ser desenvolvido.

4.2.1 Requisitos Funcionais

- Manutenção de horários;
- Cadastro de responsáveis;
- Realizar pagamento;
- Venda de produtos;
- Controle de estoque;
- Gerar relatório;

4.2.2 Requisitos Não Funcionais

- O sistema será acessado em um navegador web;
- O sistema deve ter um login e senha para o funcionário acessá-lo;
- Linguagem de programação Java;

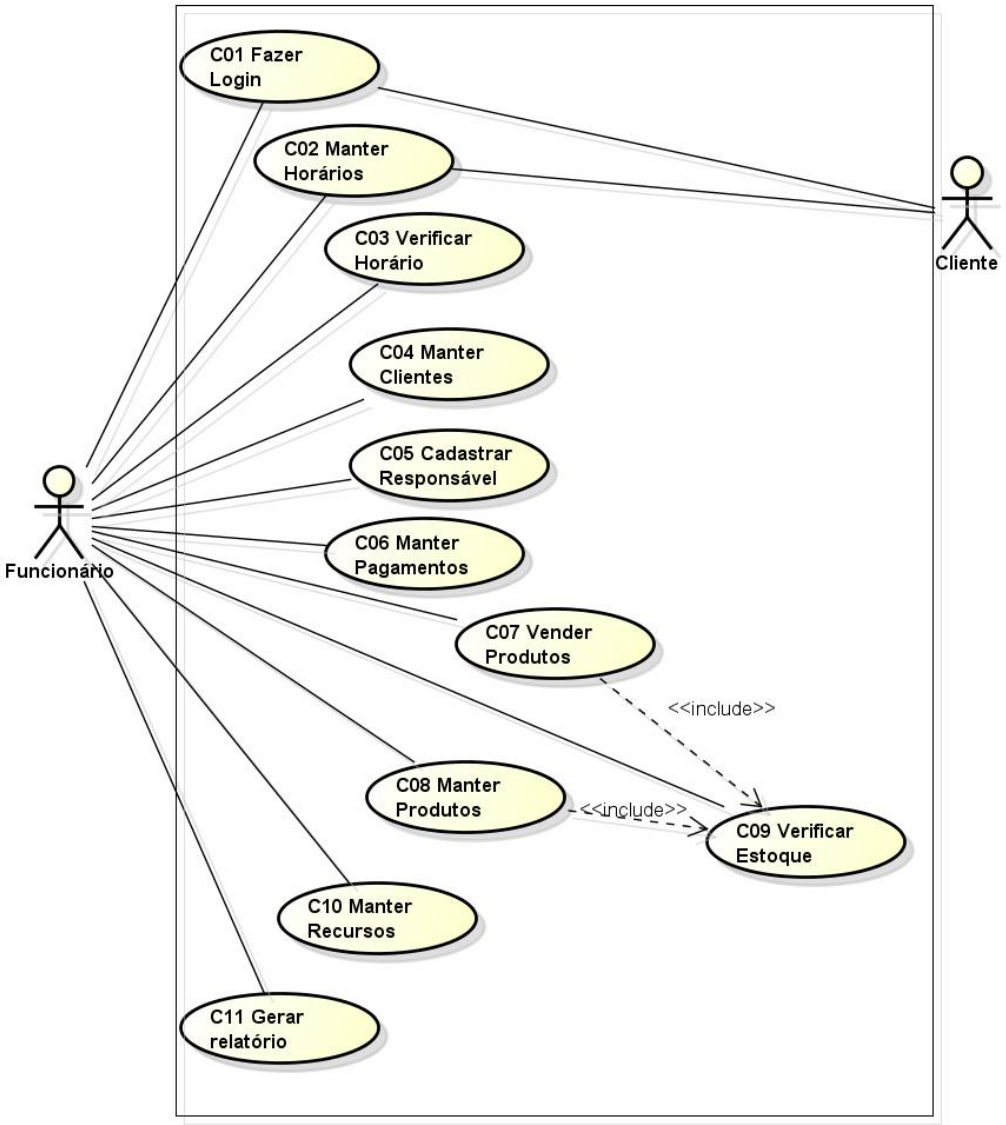
4.3 MODELAGEM DO SISTEMA

Nesta seção serão apresentados os diagramas de caso de uso, classes e atividades obtendo-se uma melhor visualização dos requisitos que o sistema deverá oferecer.

4.3.1 Diagrama de Caso de Uso

O diagrama de caso de uso (Figura 4) foi construído com o principal objetivo de apresentar as funcionalidades que o sistema deverá possuir e quem poderá interagir com o mesmo.

Figura 4: Diagrama de Caso de Uso



Fonte: próprio autor

O funcionário poderá realizar o login no sistema, registrar informações sobre os horários e realizar o cadastro de clientes e responsáveis. Além disso, irá controlar os pagamentos, os recursos disponibilizados no local, o cadastro e venda de produtos, podendo também verificar o estoque e gerar relatórios. O Cliente por sua vez também terá acesso ao

sistema através de um *login* e poderá inserir um horário que ficará dependendo da aprovação de um funcionário.

A seguir serão detalhados os casos de uso C02, C03, C05 e C07.

Tabela 1: Descrição do caso de uso C02

Nome do caso de Uso	**Manter Horários (C02)**
Caso de uso geral	---
Ator principal	Funcionário
Atores Secundários	---
Resumo	Registrar informações sobre os horários informando se estão disponíveis ou ocupados
Pré-Condições	Estar logado no sistema.
Pós-Condições	Lista dos horários é atualizada.
Fluxo Principal	1. Sistema é acessado pelo funcionário 2. É mostrada na tela uma lista com todos os horários. 3. O funcionário clica na opção inserir horário. (A1)(A2) 4. O funcionário seleciona um recurso. 5. Funcionário informa um horário. 6. O funcionário insere as informações sobre o horário selecionado. (RN1) 7. Funcionário clica na opção salvar. (A3)(A4) 8. Sistema salva as informações e retorna para a tela principal. 9. Caso de uso encerrado
	(A1) Funcionário clica em editar horário. 1. Sistema mostra as informações do horário para serem editadas. 2. Funcionário faz as alterações. 3.Sistema salva as alterações. 4.Caso de uso encerrado

Fluxo Alternativo	(A2)Funcionário clica em excluir horário. 1. Funcionário seleciona o horário que deseja excluir. 2. Sistema mostra na tela se deseja mesmo excluir o horário. 3. O Funcionário clica em “sim”. 4. Sistema exclui o horário. 5. Caso de uso encerrado. (A3) Funcionário clica em voltar 1. Sistema retorna a tela para a tela anterior 2. Caso de uso encerrado (A4) Funcionário clica em cancelar. 1. Sistema cancela a alteração das informações do horário selecionado. 2. Sistema retorna a tela anterior mostrando as informações do horário. 3. Caso de uso encerrado.
Regras de Negócio	(RN1) Funcionário deve informar o status do horário: ocupado ou disponível. Caso Ocupado deverá informar o responsável pelo horário.

Fonte: próprio autor

Tabela 2: Descrição do caso de uso C02 pelos clientes

Nome do caso de Uso	**Manter Horários pelos Clientes (C02)**
Caso de uso geral	---
Ator principal	Cliente
Atores Secundários	---
Resumo	Realizar a requisição ou cancelamento de um horário.

Pré-Condições	Estar logado no sistema.
Pós-Condições	Lista dos horários atualizada.
Fluxo Principal	1. Sistema é acessado pelo cliente 2. É mostrada na tela uma lista com todos os horários. 3. O cliente seleciona um horário desocupado. (A1) (RN1) 4. O sistema exibe as informações do horário selecionado. 5. O cliente seleciona a opção Requisitar Horário.(A2) 6. Sistema atualiza o status do horário para "Aguardando Aprovação". 9. Sistema retorna para a tela principal. 10. Caso de uso encerrado
Fluxo Alternativo	(A1) Cliente seleciona um horário que ele é o responsável. 1. Sistema exibe as informações do horário selecionado. 2. Cliente seleciona a opção "Desmarcar Horário" 3. Sistema atualiza o status do horário para "Desocupado". 4. Sistema retorna para a tela inicial. 5. Caso de uso encerrado. (A2) Cliente clica em voltar 1. Sistema retorna a tela principal mostrando a lista de horários. 2. Caso de uso encerrado
Regras de Negócio	(RN1) Sistema não permite selecionar horários ocupados em que o cliente não seja responsável.

Fonte: próprio autor

Tabela 3: Descrição do caso de uso C03

Nome do caso de Uso	**Verificar Horários (C03)**
Caso de uso geral	---
Ator principal	Funcionário
Atores Secundários	---
Resumo	Verifica o horário cadastrado por um cliente, aprovando ou não a solicitação.
Pré-Condições	Estar logado no sistema.
Pós-Condições	Lista dos horários é atualizada.
Fluxo Principal	1. Sistema é acessado pelo funcionário 2. É mostrada na tela uma lista com todos os horários. 3. Funcionário clica na opção "Verificar Horários Pendentes". 4. Sistema exibe os horários que estão aguardando aprovação. 5. O funcionário seleciona um horário. (A1) 6. O sistema exibe as informações do horário pendente. (RN1) 7. Funcionário seleciona a opção "Confirmar Horário". (A2) 8. Sistema retorna a tela principal mostrando os horários atualizados. 9. Caso de uso encerrado.
Fluxo Alternativo	(A1) Funcionário clica em voltar 1. Sistema retorna a tela principal mostrando a lista de horários. 2. Caso de uso encerrado (A2) Funcionário clica em "Rejeitar Solicitação". 1. Sistema exclui a solicitação de horário 2. Sistema volta a exibir a tela de horários

	aguardando aprovação atualizada. 3. Caso de uso encerrado.
Regras de Negócio	(RN1) Deve-se verificar se todas as informações do horário estão corretas.

Fonte: próprio autor

Tabela 4: Descrição do caso de uso C05

Nome do caso de Uso	**Cadastrar Responsável (C05)**
Caso de uso geral	---
Ator principal	Funcionário
Atores Secundários	---
Resumo	Registrar informações sobre o responsável pelo horário.
Pré-Condições	Estar logado no sistema
Pós-condições	Atualiza a lista de responsáveis cadastrados.
Fluxo Principal	1. Sistema é acessado pelo funcionário 2. Funcionário seleciona a opção Cadastrar Responsável 3. Sistema exibe na tela uma lista com os responsáveis já cadastrados. 4. Funcionário seleciona a opção Novo Cadastro. (A1) (A2) 5. Sistema exibe um formulário. 6. Funcionário preenche as informações com os dados do responsável. (RN1) 7. O funcionário clica em salvar. (A3) 8. Sistema salva as informações e retorna para a tela principal. 9. Caso de uso encerrado
Fluxo Alternativo	(A1) Funcionário seleciona a opção Editar Cadastro 1. Sistema exibe um formulário com as informações do responsável selecionado

	2. Funcionário altera os dados desejados. 3. Funcionário seleciona a opção Salvar. 4. Sistema salva as informações e retorna para a tela com a lista de responsáveis 5. Caso de uso encerrado (A2) Funcionário clica em excluir 1. Sistema solicita que o funcionário confirme a exclusão. 2. Funcionário confirma que deseja realizar a exclusão. 3. Sistema exclui o responsável selecionado. 4. Caso de uso encerrado (A3) Funcionário clica em cancelar 1. Sistema retorna a tela principal. 2. Caso de uso encerrado
Regras de Negócio	(RN1) Todas os dados do cliente devem ser inseridos no cadastro.

Fonte: próprio autor

Tabela 5: Descrição do caso de uso C07

Nome do caso de Uso	**Vender Produtos (C07)**
Caso de uso geral	---
Ator principal	Funcionário
Atores Secundários	---
Resumo	Registrar informações sobre a venda de produtos.
Pré-Condições	Estar logado no sistema
Pós-condições	
	1. Sistema é acessado pelo funcionário 2. O funcionário seleciona a opção “Realizar

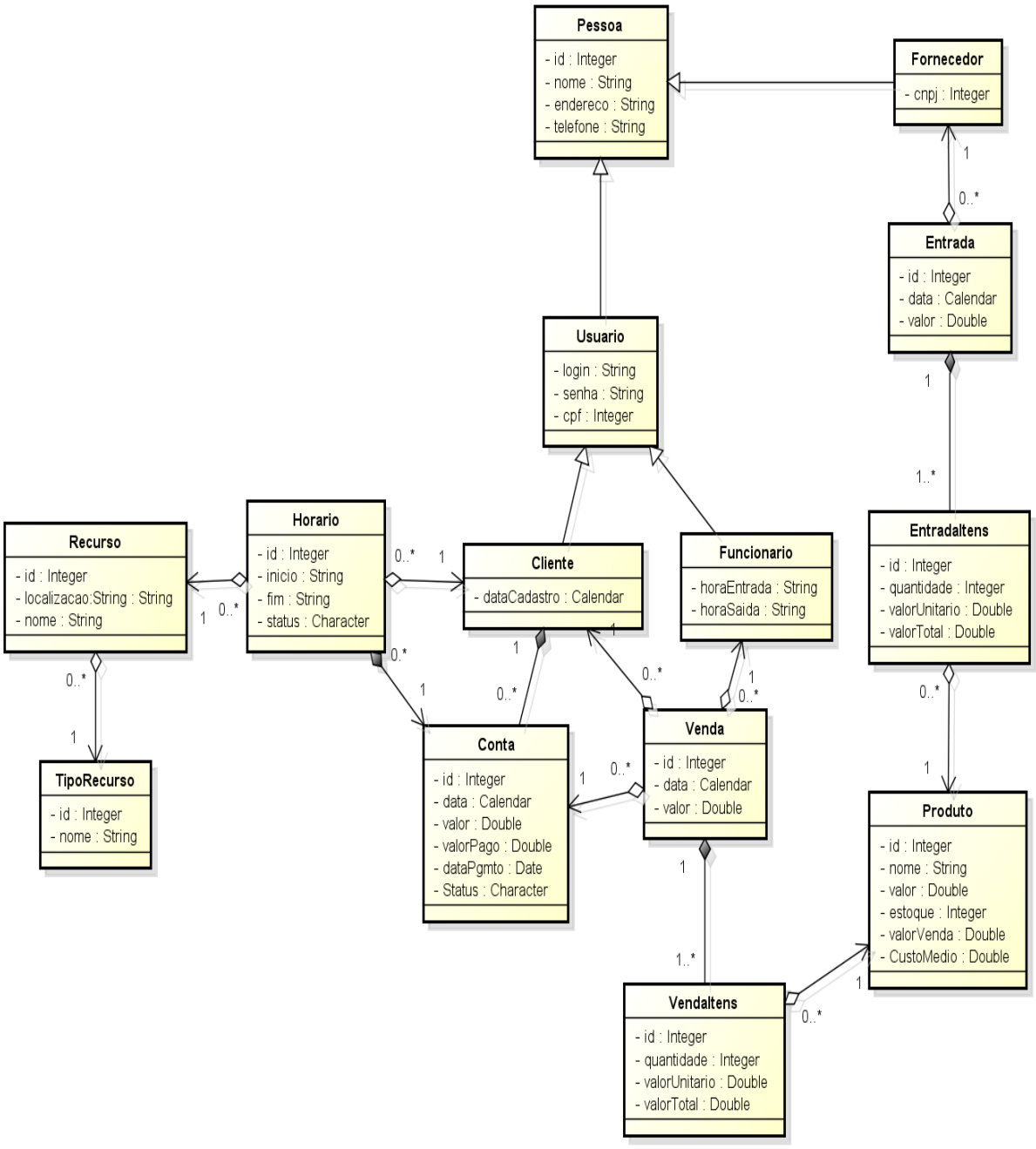
Fluxo Principal	Venda”. 3. Sistema abre uma nova tela para ser realizada a venda. 4. O funcionário informa o produto a ser vendido e quantidade. (RN1) 5. Sistema mostra o valor da venda. 6. Funcionário clica na opção “Concluir Venda”. (A1) 7. Sistema atualiza o estoque através do caso de uso C09. 8. Caso de uso encerrado.
Fluxo Alternativo	(A1) Funcionário seleciona a opção Cancelar 1. Sistema retorna a tela inicial. 2. Caso de uso encerrado. 4. Sistema salva as informações e retorna para a tela com a lista de responsáveis 5. Caso de uso encerrado (A2) Funcionário clica em excluir 1. Sistema solicita que o funcionário confirme a exclusão. 2. Funcionário confirma que deseja realizar a exclusão. 3. Sistema exclui o responsável selecionado. 4. Caso de uso encerrado (A3) Funcionário clica em cancelar 1. Sistema retorna a tela principal. 2. Caso de uso encerrado
Regras de Negócio	(RN1) Verificar se existe o produto em estoque

Fonte: próprio autor

4.3.2 Diagrama de Classes

O diagrama de classes da camada de persistência (FIGURA 5) apresenta todas as classes que serão utilizadas no desenvolvimento, mostrando seus relacionamentos e atributos.

Figura 5: Diagrama de Classes



Fonte: próprio autor

A classe Pessoa possui como atributos id, nome, endereço e telefone e é superclasse de Usuário e Fornecedor. A classe Usuário, por sua vez, também possui como atributos *login*, senha e CPF e é superclasse de Cliente e Funcionário.

A classe Cliente é implementada para realizar um controle dos responsáveis pelos horários e para armazenar os dados dos clientes. Um cliente pode ter vários horários, porém um horário é de apenas um cliente. A classe Horário possui como atributos id, inicio, fim e status. Cada horário possui um recurso, que por sua vez, pode ter vários horários e um tipo de recurso.

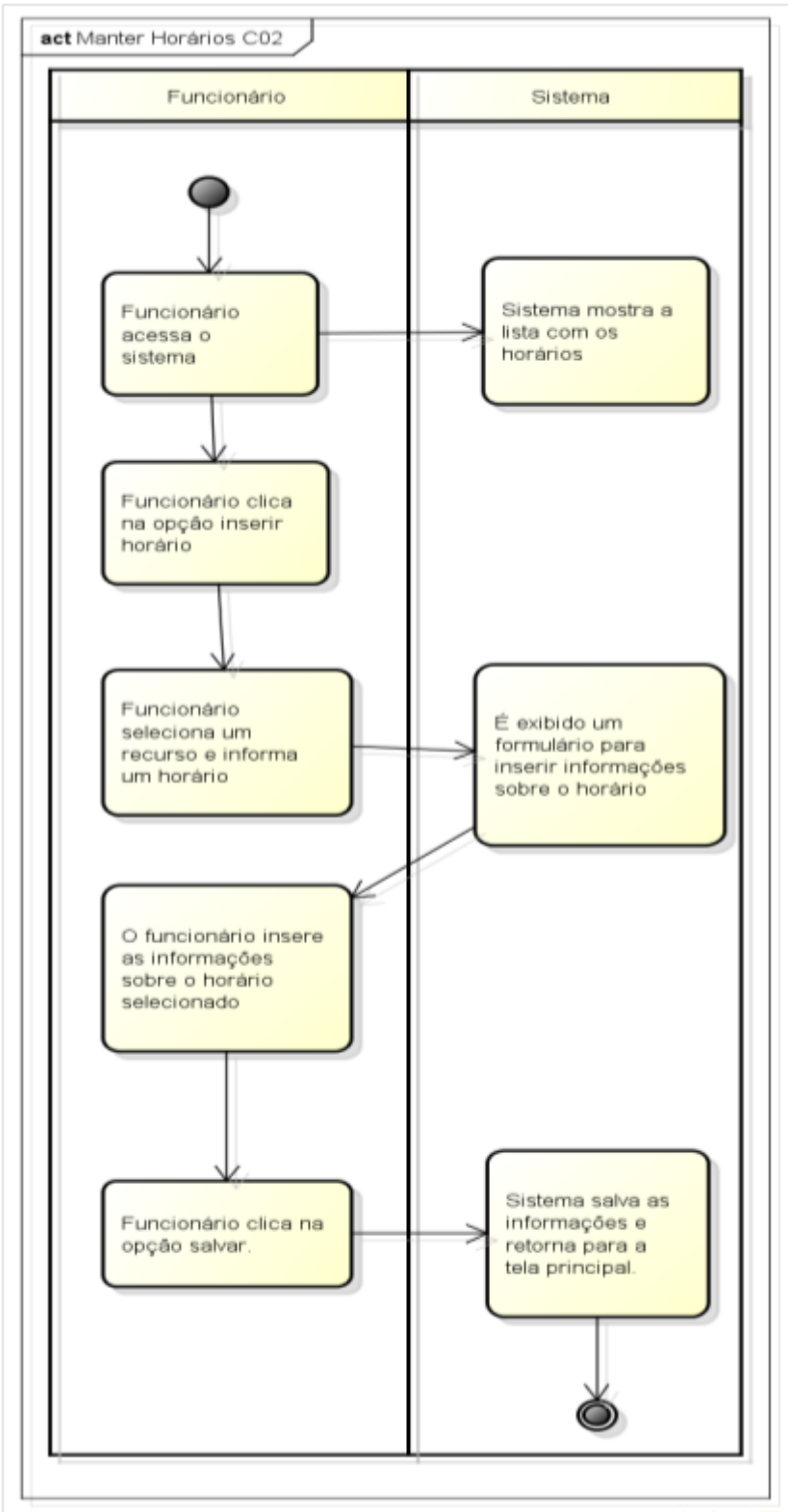
Uma conta se relaciona apenas com um cliente e pode ter vários horários. Também poderá se relacionar com várias vendas. Uma venda possui apenas um cliente, uma conta e um funcionário - que pode realizar diversas vendas. Além disso, cada venda pode ter diversos itens.

A entrada de itens deve obrigatoriamente se relacionar com apenas uma entrada. A entrada é realizada por apenas um fornecedor. A classe Fornecedor, especialização de Pessoa, possui como atributo um cnpj. As entradas de itens contem um tipo de produto e um produto pode estar ou não presente em diversas entradas de itens. Além disso, a venda de itens contém um produto e o produto pode estar em diversas vendas de itens.

4.3.3 Diagrama de Atividades

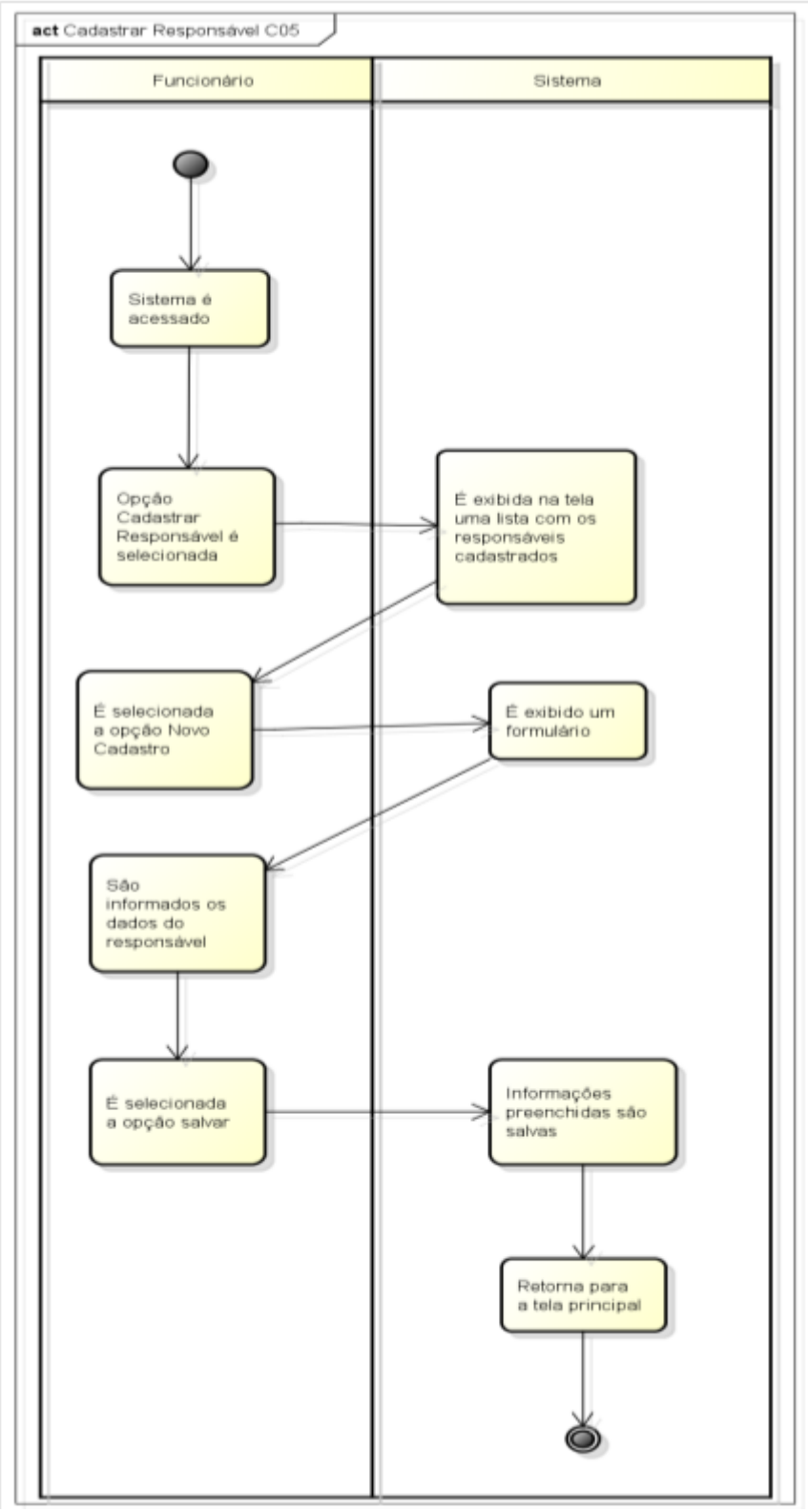
Nesta seção serão apresentados os diagramas de atividades dos casos de uso C02 e C05, apresentando seus fluxos.

Figura 6: Diagrama de Atividades do caso de uso C02



Fonte: próprio autor

Figura 7: Diagrama de Atividades do caso de uso C05

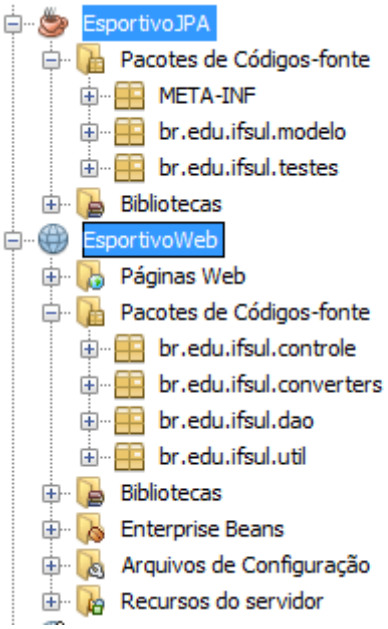


Fonte: próprio autor

4.4 PREPARAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

Para o desenvolvimento do sistema proposto neste trabalho foram criados dois projetos: um projeto Java SE para a camada de Modelo e outro projeto Java EE Web para as camadas de Controle e Visão. Ambos os projetos foram desenvolvidos na IDE NetBeans e a versão utilizada foi a 8.0.2. A figura 8 apresenta a estrutura dos projetos no NetBeans.

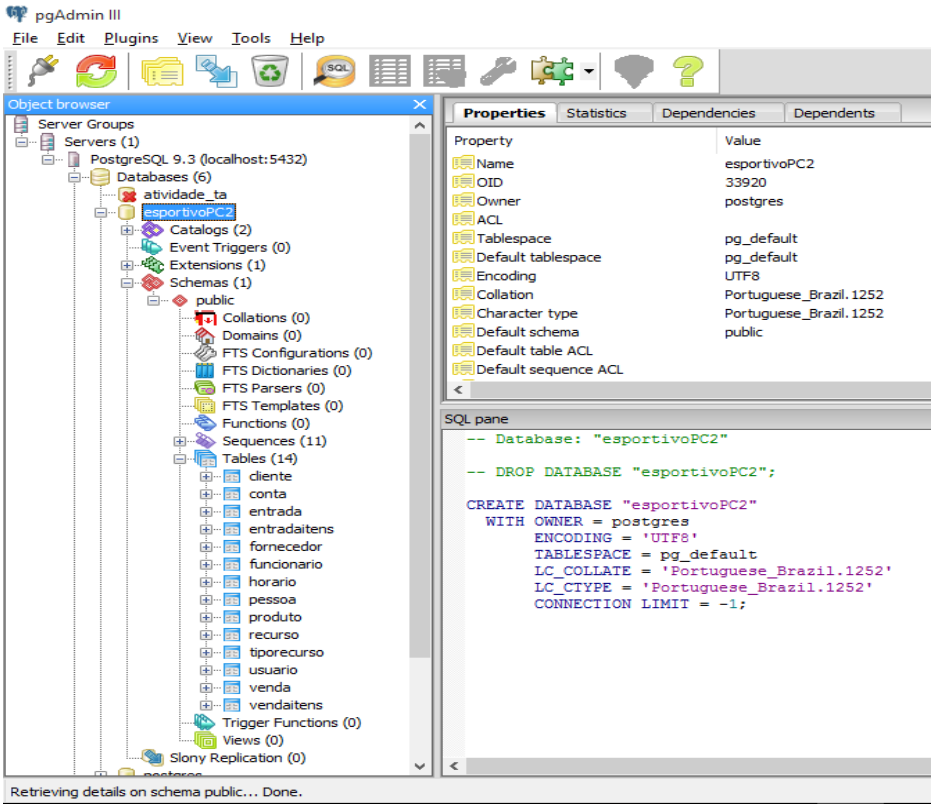
Figura 8: Projetos do NetBeans



Fonte: Próprio Autor

O SGBD utilizado foi o PostgreSQL e sua versão é a 9.4. Para a criação do banco de dados do sistema foi utilizada a ferramenta pgAdmin III apresentada na figura 9.

Figura 9: Banco de Dados



Fonte: Próprio Autor

Para o desenvolvimento do sistema foram utilizadas algumas bibliotecas presentes em ambos os projetos:

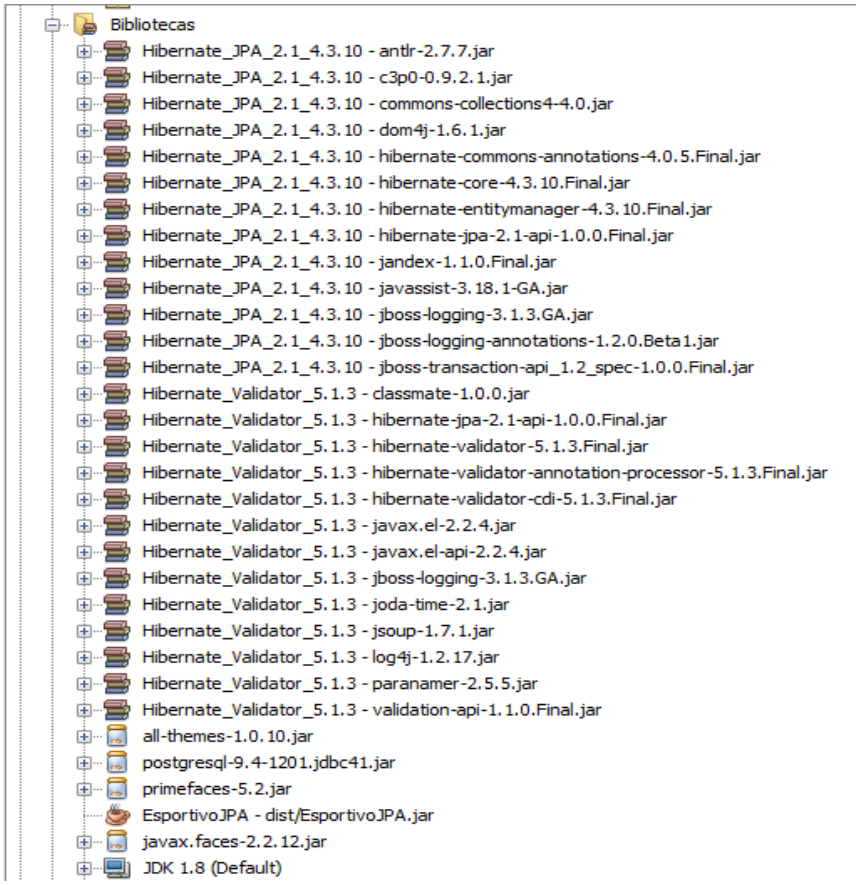
- Hibernate versão 4.3.10 - responsável por implementar a especificação 2.1 da JPA.
- Hibernate Validator versão 5.1.3
- Driver do PostgreSQL

Já para o projeto Java EE Web, além dessas bibliotecas, também foram importadas:

- Biblioteca do Primefaces versão 5.2
- Biblioteca com os temas do Primefaces (all-themes-1.0.10)
- Projeto Java SE que contêm a camada de modelo da aplicação

As bibliotecas utilizadas pelo sistema são apresentadas na figura 10.

Figura 10: Bibliotecas Utilizadas



Fonte: Próprio Autor

Foi criado o arquivo de persistência, com o objetivo de configurar a aplicação para utilizar o recurso JDBC do servidor GlassFish, e dessa forma, o gerenciamento transacional dos objetos *"Entity Manager"* é realizado pelo servidor. O código do arquivo "persistence.xml" é apresentado na figura 11, sendo que o mesmo é exigido pela especificação JPA, pois é responsável pela ligação entre o provedor de JPA com a base de dados da aplicação.

Figura 11: Aquivo persistence.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
3               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4               xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
5                                   http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
6    <persistence-unit name="EsportivoWebPU" transaction-type="JTA">
7      <provider>org.hibernate.ejb.HibernatePersistence</provider>
8      <jta-data-source>jdbc/esportivoPC2</jta-data-source>
9      <class>br.edu.ifsul.modelo.Cliente</class>
10     <class>br.edu.ifsul.modelo.Conta</class>
11     <class>br.edu.ifsul.modelo.Entrada</class>
12     <class>br.edu.ifsul.modelo.EntradaItens</class>
13     <class>br.edu.ifsul.modelo.Fornecedor</class>
14     <class>br.edu.ifsul.modelo.Funcionario</class>
15     <class>br.edu.ifsul.modelo.Horario</class>
16     <class>br.edu.ifsul.modelo.Pessoa</class>
17     <class>br.edu.ifsul.modelo.Produto</class>
18     <class>br.edu.ifsul.modelo.Recurso</class>
19     <class>br.edu.ifsul.modelo.TipoRecurso</class>
20     <class>br.edu.ifsul.modelo.Usuario</class>
21     <class>br.edu.ifsul.modelo.Venda</class>
22     <class>br.edu.ifsul.modelo.VendaItens</class>
23     <properties>
24       <property name="hibernate.hbm2ddl.auto" value="update"/>
25       <property name="hibernate.dialect" value="org.hibernate.dialect.PostgreSQLDialect"/>
26       <property name="hibernate.transaction.jta.platform"
27                 value="org.hibernate.service.jta.platform.internal.SunOneJtaPlatform"/>
28     </properties>
29   </persistence-unit>
30 </persistence>
```

Fonte: Próprio Autor

Na linha 6, a tag <*persistence-unit*> está definindo o nome da persistência e o tipo de transação que será realizada. O elemento <*provider*> especifica o provedor de persistência Hibernate e o elemento <*jta-data-source*> define o nome da fonte de dados utilizada pela aplicação. Este arquivo também define algumas propriedades que estão especificadas na tag <*properties*>. A propriedade <*property name="hibernate.hbm2ddl.auto" value="update"/*> indica que as tabelas serão atualizadas automaticamente quando o banco de dados for criado. A segunda propriedade define que o dialeto utilizado será do PostgreSQL gerando, dessa forma, um SQL para PostgreSQL. Já a terceira propriedade define a plataforma de transação do Hibernate.

Existe também o arquivo glassfish-resources.xml, que foi criado com a unidade de persistência e define algumas propriedades do pool de conexões do servidor GlassFish e é apresentado na figura 12.

Figura 12: Glassfish-resources.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1 Resource Definitions//EN"
"http://glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>
    <jdbc-connection-pool allow-non-component-callers="false" associate-with-thread="false"
                          connection-creation-retry-attempts="0"
                          connection-creation-retry-interval-in-seconds="10"
                          connection-leak-reclaim="false" connection-leak-timeout-in-seconds="0"
                          connection-validation-method="auto-commit"
                          datasource-classname="org.postgresql.ds.PGSimpleDataSource"
                          fail-all-connections="false" idle-timeout-in-seconds="300"
                          is-connection-validation-required="false" is-isolation-level-guaranteed="true"
                          lazy-connection-association="false" lazy-connection-enlistment="false"
                          match-connections="false" max-connection-usage-count="0" max-pool-size="32"
                          max-wait-time-in-millis="60000" name="esportivoPC2_Pool"
                          non-transactional-connections="false" pool-resize-quantity="2"
                          res-type="javax.sql.DataSource" statement-timeout-in-seconds="-1"
                          steady-pool-size="8" validate-atmost-once-period-in-seconds="0"
                          wrap-jdbc-objects="false">
        <property name="serverName" value="localhost"/>
        <property name="portNumber" value="5432"/>
        <property name="databaseName" value="esportivoPC2"/>
        <property name="User" value="postgres"/>
        <property name="Password" value="postgres"/>
        <property name="URL" value="jdbc:postgresql://localhost:5432/esportivoPC2"/>
        <property name="driverClass" value="org.postgresql.Driver"/>
    </jdbc-connection-pool>
    <jdbc-resource enabled="true" jndi-name="jdbc/esportivoPC2" object-type="user"
                   pool-name="esportivoPC2_Pool"/>
</resources>
```

Fonte: Próprio Autor

No elemento <*jdbc-connection-pool*> há um atributo chamado *name* que define o nome do recurso jdbc onde o servidor GlassFish irá instanciar as Entity Managers. Além disso, nesse elemento são definidas as propriedades como nome do servidor, número da porta, nome da base de dados, usuário, senha, url e o driver do banco de dados utilizado pela aplicação. Já o elemento <*jdbc-resource*> possui o atributo *jndi-name* o qual é responsável por definir o nome do recurso que será utilizado pelos beans de sessão e também possui o atributo *pool-name* que define o nome do pool de conexões.

4.5 CAMADA DE MODELO

Na camada de modelo é realizado o mapeamento objeto-relacional, realizado pela API de persistência do Java, a JPA. Essa API é responsável por criar uma correspondência entre os objetos e as tabelas do sistema. A figura 13 mostra a classe Horário com seus atributos e anotações necessários para ela ser persistida na base de dados. Essa classe faz parte da camada de Modelo do padrão MVC.

Figura 13: Classe Horario.java

```
@Entity
@Table(name = "horario")
public class Horario implements Serializable{
    @Id
    @Column(name = "id")
    @SequenceGenerator(name = "seq_id_horario", sequenceName = "gen_horario_id",
            allocationSize = 1)
    @GeneratedValue(generator = "seq_id_horario", strategy = GenerationType.SEQUENCE)
    private Integer id;
    @Length(max = 5,message = "O inicio não pode ultrapassar {max} caracteres")
    @NotEmpty(message = "O inicio deve ser informado")
    @Column(name = "inicio",length = 5,nullable = false)
    private String inicio;
    @Length(max = 5,message = "O fim não pode ultrapassar {max} caracteres")
    @NotEmpty(message = "O fim deve ser informado")
    @Column(name = "fim",length = 5,nullable = false)
    private String fim;
    @Column(name = "status",length = 1,nullable = false)
    private Character status;
    @NotNull(message = "O recurso deve ser informado")
    @ManyToOne
    @JoinColumn(name = "recurso", referencedColumnName = "id", nullable = false)
    private Recurso recurso;
    @NotNull(message = "O cliente deve ser informado")
    @ManyToOne
    @JoinColumn(name = "cliente", referencedColumnName = "id", nullable = false)
    private Cliente cliente;
    @OneToOne(cascade = CascadeType.PERSIST, optional = false, fetch = FetchType.EAGER, orphanRemoval = false)
    @PrimaryKeyJoinColumn
    private Conta conta;
```

Fonte: Próprio Autor

Para ser reconhecida como entidade, a classe Horário deve possuir a anotação @Entity. Já a anotação @Id é usada para especificar a chave primária e seu identificador é gerado automaticamente pelo provedor de persistência pela anotação @GeneratedValue. Foi definida a estratégia de geração por GenerationType.SEQUENCE. Ainda, foi definido um nome para a sequência através da anotação @SequenceGenerator.

Através da anotação @Column é possível definir o nome das colunas e especificar se as mesmas podem ser nulas ou não. Já as anotação @NotNull e @NotEmpty pertencem à API Hibernate Validator e indica uma restrição de integridade para que o atributo não seja nulo. A anotação @Lenght define o tamanho máximo que o atributo da tabela poderá possuir.

Para representar o relacionamento muitos para um entre a classe Horário e as classes Recurso e Cliente, foram utilizadas as anotações @ManyToOne e @JoinColumn(name = “recurso”, referencedColumnName = “id”, nullable = false). A primeira anotação define o relacionamento muitos para um e a segunda especifica o nome da coluna que será criada na tabela horário, a referência que será utilizada, que nesse caso, será a coluna id da tabela Recurso e a terceira indica que os atributos são obrigatórios.

4.6 A CAMADA LÓGICA FUNCIONAL

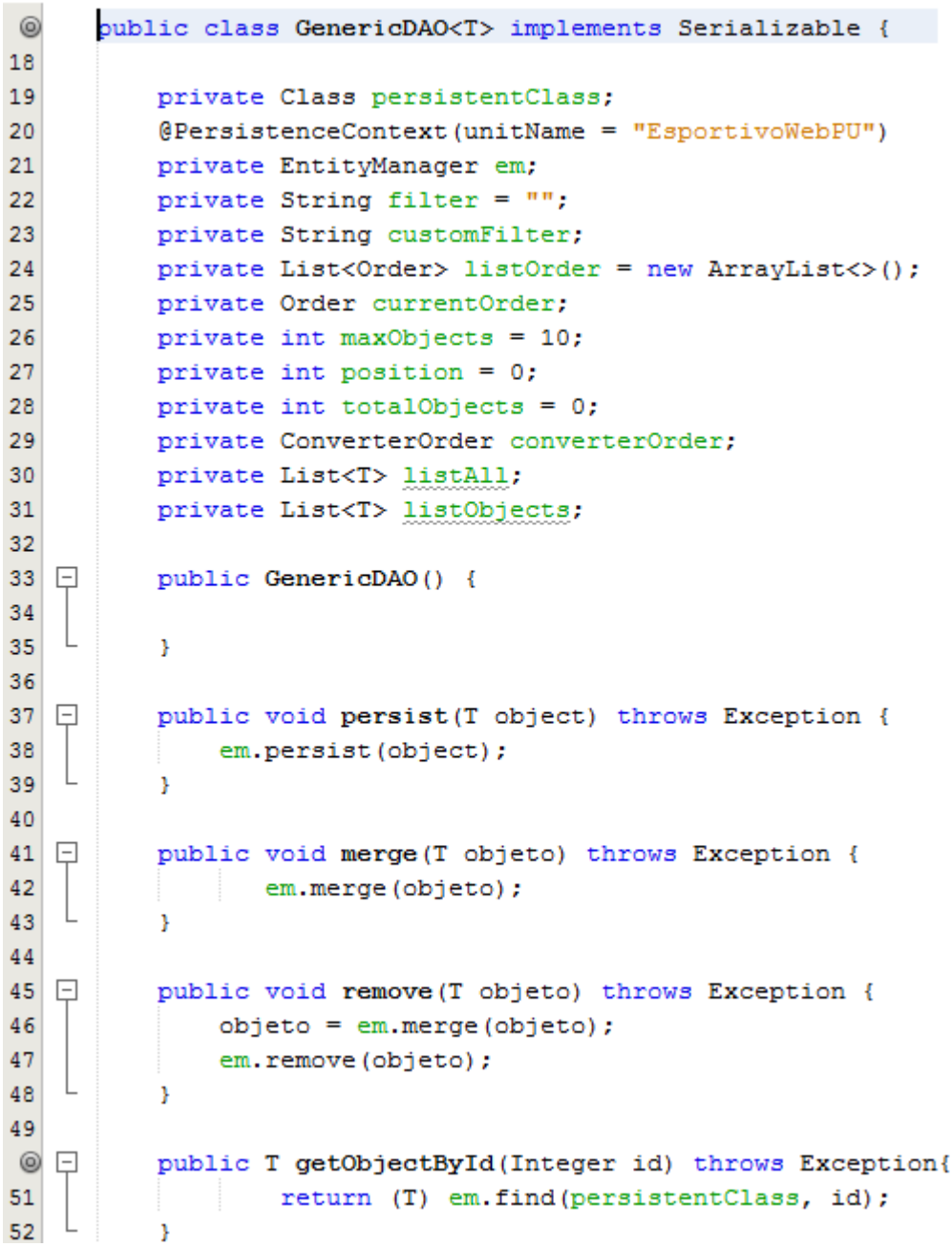
Para a criação da camada funcional da aplicação utilizou-se os EJBs que separam a camada de persistência da camada de apresentação, implementam a lógica funcional e adicionam gerenciamento de transações e segurança.

A classe HorarioDAO é um bean de sessão que não guarda estado, é transacional, e herda as operações básicas de criação, leitura, atualização e exclusão (CRUD)[1] da classe GenericDAO. As classes HorarioDAO e Horario são empacotadas e distribuídas no servidor de aplicações GlassFish, assim, a aplicação, na sua camada de controle, poderá chamar métodos no EJB. Para utilizar as transações, o bean de sessão acessa a base de dados através da fonte de dados (jdbc/ esportivoPC2), a qual foi criada no servidor GlassFish e ligada à base de dados esportivoPC2.

A figura 14, mostra o código da classe GenericDAO, ela possui um atributo do tipo Class chamado persistenceClass, o qual guarda o tipo da classe que está utilizando a classe genérica. A anotação @PersistenceContext(unitName = “EsportivoWebPU”) indica ao EJB o nome da persistência que será utilizada na aplicação. Nessa classe é instanciado um objeto Entity Manager chamado de *em*, o qual é diretamente injetado no bean de sessão e, através dele o EJB poderá realizar todas as transações necessárias. Os demais atributos são utilizados em outros métodos para realizar a ordenação, a listagem de objetos e utilização de filtros de pesquisa.

[1] CRUD: Create, Read, Update, Delete.

Figura 14: GenericDAO.java



```
public class GenericDAO<T> implements Serializable {

    private Class persistentClass;
    @PersistenceContext(unitName = "EsportivoWebPU")
    private EntityManager em;
    private String filter = "";
    private String customFilter;
    private List<Order> listOrder = new ArrayList<>();
    private Order currentOrder;
    private int maxObjects = 10;
    private int position = 0;
    private int totalObjects = 0;
    private ConverterOrder converterOrder;
    private List<T> listAll;
    private List<T> listObjects;

    public GenericDAO() {

    }

    public void persist(T object) throws Exception {
        em.persist(object);
    }

    public void merge(T objeto) throws Exception {
           em.merge(objeto);
    }

    public void remove(T objeto) throws Exception {
        objeto = em.merge(objeto);
        em.remove(objeto);
    }

    public T getObjectById(Integer id) throws Exception{
            return (T) em.find(persistentClass, id);
    }
```

Fonte: Próprio Autor

A figura 15 mostra o código da classe HorarioDAO, a qual possui a anotação @Stateless que define a classe como um bean de sessão sem estado. O gerenciamento de instâncias é feito, automaticamente, pelo contentor EJB.

Figura 15: HorarioDAO.java

```
7   @Stateless
8   public class HorarioDAO<T> extends GenericDAO<Horario> implements Serializable {
9
10      public HorarioDAO() {
11          super();
12          super.setPersistentClass(Horario.class);
13          // inicializar as ordenações possiveis
14          super.getListOrder().add(
15                  new Order("id", "ID", "="));
16          super.getListOrder().add(
17                  new Order("status", "Status", "like"));
18          super.getListOrder().add(
19                  new Order("inicio", "Inicio", "like"));
20          super.getListOrder().add(
21                  new Order("fim", "Fim", "like"));
22          // definir qual a ordenação padrão no caso o segundo elemento da lista (indice 1)
23          super.setCurrentOrder((Order) super.getListOrder().get(1));
24          // inicializando o filtro
25          super.setFilter("");
26          // inicializando o conversor da ordem
27          super.setConverterOrder(new ConverterOrder(super.getListOrder()));
28      }
29
30  }
```

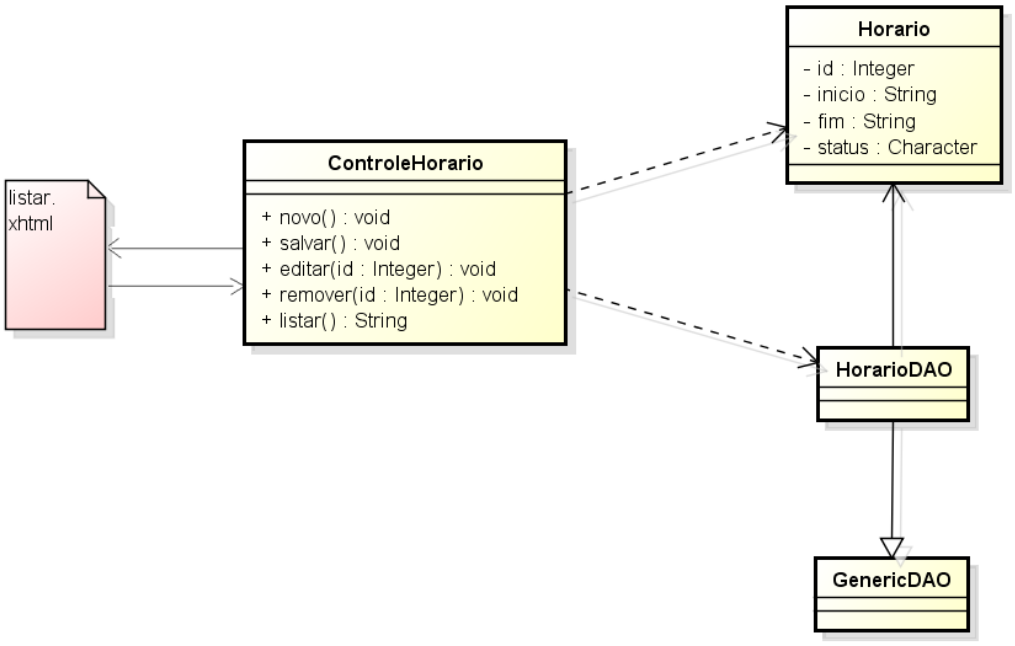
Fonte: Próprio Autor

O método super.setPersistentClass(Horario.class) define o tipo de classe Horário que está utilizando a classe GenericDAO. Os outros métodos implementados na classe são utilizados para realizar a paginação e ordenação dos registros.

4.7 CAMADA DE CONTROLE

Na camada de Controle do projeto, que faz parte do padrão MVC, estão os *beans* gerenciados, que tem como principal objetivo, interagir com as outras camadas da aplicação e também realizar validações. Na figura 16 é possível observar como funciona a interação entre os componentes da aplicação desenvolvida. Será apresentado o *Managed Bean* da classe “ControleHorario”, visto que as outras classes possuem implementação e funcionamento semelhantes.

Figura 16: Interação entre páginas e classes da aplicação



Fonte: Próprio Autor

A classe ControleHorario é definida como um *bean* gerenciado e interage com as classes Horario e HorarioDAO, deixando toda a lógica funcional da aplicação ao GenericDAO, que contém os métodos CRUD. A classe HorarioDAO é um EJB sem estado que herda todos os métodos e atributos da classe GenericDAO, sendo também responsável por manipular a classe Horario através da API Entity Manager. Já página “listar.xhtml” recebe uma lista de todos os horários cadastrados no banco de dados através de um método chamado *listObjects*() que é definido na classe GenericDAO.

Na figura 17 é possível observar o código da classe ControleHorario que é um *bean* gerenciado. É possível identificar a anotação *@ManagedBean*, que tem como objetivo indicar que a classe é um *bean* gerenciado com o nome de controleHorario.

Figura 17: ControleHorario.java

```
/**
 *
 * @author William
 */
@ManagedBean(name = "controleHorario")
@ViewScoped
public class ControleHorario implements Serializable{

    @EJB
    private HorarioDAO<Horario> dao;
    private Horario objeto;


    public ControleHorario() {

    }

    public String listar() {...3 linhas }

    public void novo() {...3 linhas }

    public void salvar() {...12 linhas }

    public void editar(Integer id) {...7 linhas }

    public void remover(Integer id) {...9 linhas }
```

Fonte: Próprio autor

A outra anotação chamada *@ViewScoped* define que o *bean* possuirá um escopo de visão, dessa forma a duração vida dele será maior do que de uma requisição e menor que de uma sessão. Este *bean* contém um atributo chamado *"dao"* do tipo HorarioDAO que é injetado com referência ao *EJB* através da anotação *@EJB* e outro atributo chamado "objeto" do tipo Horario que será mapeado em um formulário definido na pagina web "listar.xhtml" e persistido no banco de dados. A classe ainda possui os métodos *novo( )* e *salvar( )* que permitem a criação de um horário através do EJB. Já métodos *editar( )* e *remover( )* recebem por parâmetro um identificador e permitem, respectivamente, realizar a atualização e a exclusão de um registro. O método *listar()* apenas retorna uma *string* que possui um caminho específico para redirecionar até uma página web. A figura 18 apresenta os códigos dos cinco métodos.

Figura 18: Métodos da Classe ControleHorario

```
public String listar() {
    return "/privado/horario/listar?faces-redirect=true";
}

public void novo() {
    objeto = new Horario();
}

public void salvar() {
    try {
        if (objeto.getId() == null) {
            dao.persist(objeto);
        } else {
            dao.merge(objeto);
        }
        Util.mensagemInformacao("Objeto persistido com sucesso");
    } catch (Exception e) {
        Util.mensagemErro("Erro ao persistir objeto: " + e.getMessage());
    }
}

public void editar(Integer id) {
    try {
        objeto = dao.getObjectById(id);
    } catch (Exception e) {
        Util.mensagemErro("Erro ao recuperar objeto: " + e.getMessage());
    }
}
public void remover(Integer id) {
    try {
        objeto = dao.getObjectById(id);
        dao.remove(objeto);
        Util.mensagemInformacao("Objeto removido com sucesso");
    } catch (Exception e) {
        Util.mensagemErro("Erro ao remover objeto: " + e.getMessage());
    }
}
```

Fonte: Próprio Autor

O método *novo( )* apenas instancia um objeto do tipo Horario. Já método *salvar( )* verifica se um objeto já existe na base de dados. Caso um horário não exista, o atributo identificador do objeto é nulo, então o controlador chama um objeto do tipo HorarioDAO, que nesse caso é definido como “dao” e através do método *persist*(objeto), persiste um novo objeto na base de dados. Porém se o identificador do objeto existir, ou seja, não for nulo, o controlador chamará atributo “dao” e através do método *merge*(objeto), a base de dados será atualizada.

Ambos os métodos editar() e remover() atribuem um id para o objeto através do método *getObjectById()*, que é invocado utilizando o atributo “dao”. Dessa maneira, ao realizar a edição do objeto, o mesmo não terá um id nulo, então quando o método salvar() for chamado, a base de dados irá atualizá-lo. Já para remover, o atributo “dao” chama o método *remove()* passando por parâmetro o objeto.

4.8 CAMADA DE VISÃO

A figura 19 apresenta o código da página web listar.xhtml que faz parte da camada de Visão do padrão MVC e também possui componentes do PrimeFaces. Nesta página, criou-se um formulário que possui um elemento chamado *dataTable* responsável por receber registros da base de dados da aplicação.

Figura 19: Código da página web listar.xhml – Formulário de Listagem

```
<?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:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://primefaces.org/ui"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

    <ui:composition template="/templates/template.xhtml">
        <ui:define name="titulo">Manutenção de Horários</ui:define>
        <ui:define name="conteudo">
            <h:form id="formListagem">
                <p:growl/>
                <p:messages/>
                <div ...45 linhas /><!--Filtros, ordenação e paginação  -->
                <p:dataTable value="#{controleHorario.dao.listObjects}" var="obj"
                             emptyMessage="Nenhum registro encontrado" id="tabela">
                    <p:column headerText="ID">
                        <p:outputLabel value="#{obj.id}"/>
                    </p:column>
                    <p:column headerText="Inicio">
                        <p:outputLabel value="#{obj.inicio}"/>
                    </p:column>
                    <p:column headerText="Fim">
                        <p:outputLabel value="#{obj.fim}"/>
                    </p:column>
                    <p:column headerText="Status">
                        <p:outputLabel value="#{obj.status}"/>
                    </p:column>
                    <p:column headerText="Recurso">
                        <p:outputLabel value="#{obj.recurso.nome}"/>
                    </p:column>
                    <p:column headerText="Cliente">
                        <p:outputLabel value="#{obj.cliente.nome}"/>
                    </p:column>
                    <p:column headerText="Conta">
                        <p:outputLabel value="#{obj.conta.id}"/>
                    </p:column>
                    <p:column headerText="Ações">
                        <div ...14 linhas /> <!--Botões EDITAR e REMOVER -->
                    </p:column>
                    <f:facet name="footer">
                        <h:outputLabel value="#{controleHorario.dao.navigationMessage}" />
                    </f:facet>
                </p:dataTable>
            </h:form>
        </ui:define>
```

Fonte: Próprio Autor

O elemento <*p:datatable*> possui um atributo chamado *value* onde um *bean* gerenciado chama o método *listObjects()* através do atributo "dao". Nesse caso, esse método trará uma lista com todos os horários existentes na base de dados. Através do atributo *var*, é possível definir uma variável que realizará a função de iterar a lista. Os atributos do horário são mostrados nas colunas através das expressões *#{obj.id}*, *#{obj.inicio}*, *#{obj.fim}*, *#{obj.status}*, *#{obj.recurso.nome}*, *#{obj.cliente.nome}* e *#{obj.conta.id}*. Para melhor visualização e compreensão do código, as linhas referentes à navegação, filtro, ordenação e paginação foram omitidos.

O botão novo, apresentado na figura 20, ao ser clicado, aciona o método novo() do *bean* gerenciado "controleHorario" e então é persistido um novo horário na base de dados.

Figura 20: Botão Novo

```
<p:commandButton actionListener="#{controleHorario.novo()}"
                 value="Novo" oncomplete="PF('dlg').show();"
                 update=":formEdicao" icon="ui-icon-plus"/>
```

Fonte: Próprio Autor

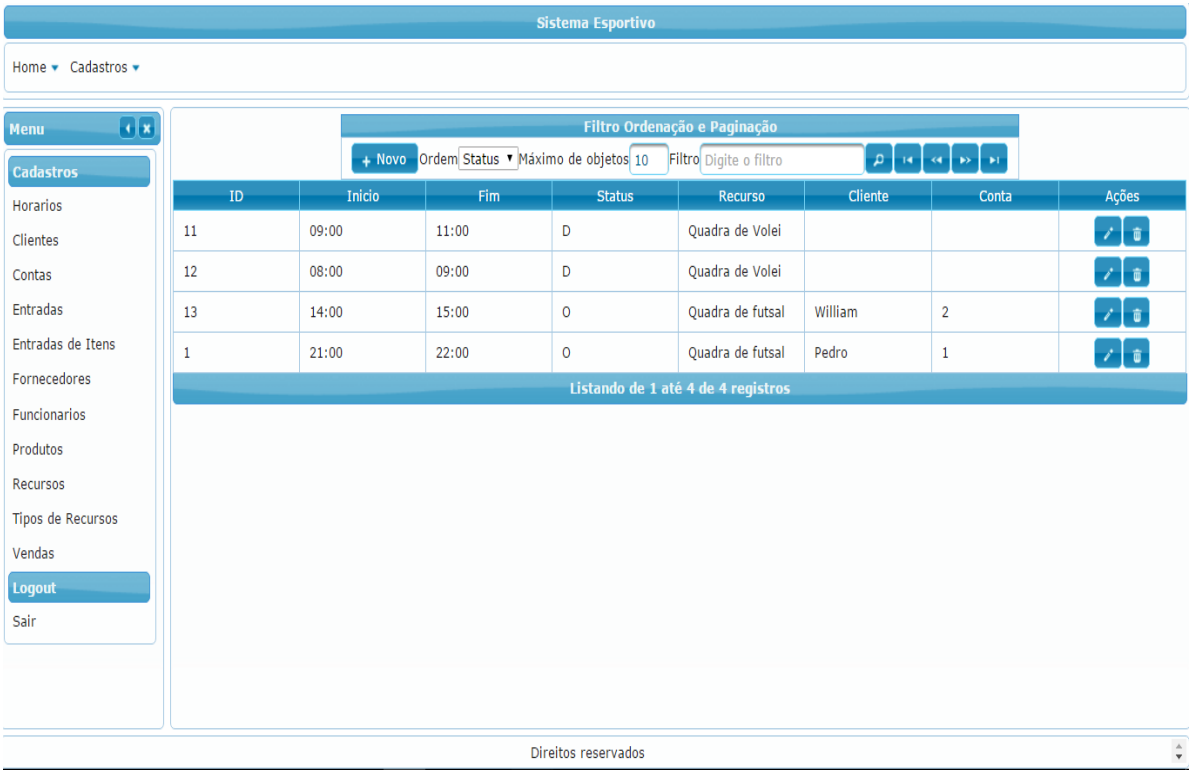
Da mesma forma ocorre com os botões editar e remover apresentados na figura 21. Ambos chamam seus respectivos métodos do *bean* gerenciado. No método do botão editar é passado por parâmetro o *id* do objeto e um formulário de edição será aberto. No método do botão remover também é passado por parâmetro o id do objeto e ao remove-lo o formulário de listagem é atualizado. A figura 22 apresenta a tela da pagina listar.xhtml.

Figura 21: Botões de Excluir e Remover

```
<p:column headerText="Ações">
    <div align="center">
        <p:commandButton actionListener="#{controleHorario.editar(obj.id)}"
                         icon="ui-icon-pencil" id="btnEditar"
                         update=":formEdicao"
                         oncomplete="PF('dlg').show();"/>
        <p:commandButton
            actionListener="#{controleHorario.remover(obj.id)}"
            icon="ui-icon-trash" id="btnExcluir"
            update=":formListagem"
            onclick="if (!confirm('Confirma a exclusão do objeto?'))
                    return false"/>
        <p:tooltip for="btnEditar" value="Editar registro"/>
        <p:tooltip for="btnExcluir" value="Excluir registro" showEffect="explode"/>
    </div> <!--Botões EDITAR e REMOVER -->
</p:column>
```

Fonte: Próprio Autor

Figura 22: Visualização da pagina listar.xhtml - formulário de listagem



Fonte: Próprio Autor

A figura 23 mostra a continuação do código da página listar.xhtml onde foi colocado o diálogo que irá ser exibido na tela ao inserir ou editar um objeto. Para melhor visualização, também omitiu-se algumas linhas do código.

Figura 23: Página web listar.xhtml – Caixa de diálogo(Formulário de Edição)

```
<ui:define name="dialogos">
    <p:dialog header="Edição" widgetVar="dlg" resizable="false" modal="true">
        <h:form id="formEdicao">
            <p:messages/>
            <p:panelGrid columns="2">
                <f:facet name="footer">
                    <div align="center">
                        <p:commandButton
                            value="Salvar" icon="ui-icon-disk"
                            actionListener="#{controleHorario.salvar()}"
                            update=":formListagem :formEdicao"
                            oncomplete="if(!args.validationFailed){PF('dlg').hide();}"/>
                    </div>
                </f:facet>
                <p:outputLabel value="ID" for="txtId"/>
                <p:inputText ...2 linhas />
                <p:outputLabel value="Inicio" for="txtInicio"/>
                <p:inputMask ...2 linhas />
                <p:outputLabel value="Fim" for="txtFim"/>
                <p:inputMask ...2 linhas />
                <p:outputLabel value="Status" for="txtStatus"/>
                <p:selectOneMenu ...7 linhas />
                <p:outputLabel value="Cliente" for="txtCliente"/>
                <p:selectOneMenu value="#{controleHorario.objeto.cliente}"
                                 id="txtCliente">
                    <f:converter converterId="converterCliente"/>
                    <f:selectItem ...2 linhas />
                    <f:selectItems value="#{controleCliente.dao.listarTodos}"
                                   var="e" itemLabel="#{e.nome}"/>
                </p:selectOneMenu>
                <p:outputLabel value="Recurso" for="txtRecurso"/>
                <p:selectOneMenu ...8 linhas />

                <p:outputLabel value="Conta" for="txtConta"/>
                <p:selectOneMenu ...8 linhas />


            </p:panelGrid>
        </h:form>
    </p:dialog>
```

Fonte: Próprio Autor

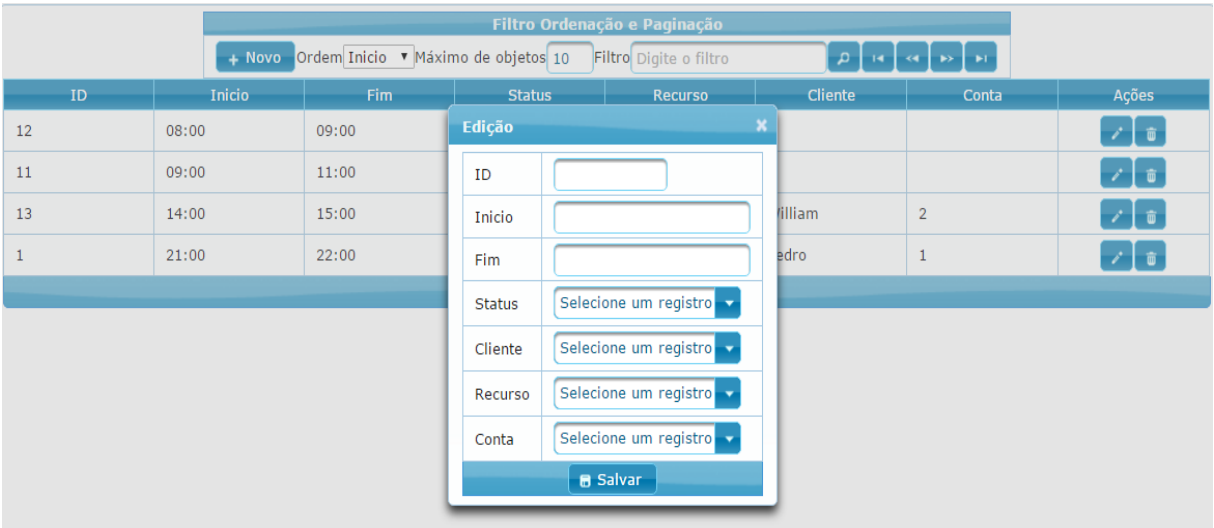
O elemento <*p:dialog*> é responsável por renderizar uma caixa de diálogo na tela e, através de seus atributos *resizable=”false”* e *modal=”true”*, torna-se incapaz de ser redimensionado e para alterar o foco da tela é necessário fechar o diálogo. Dentro dele foi inserido um formulário de edição, no qual serão informados os dados de um objeto horário, por exemplo, o campo Início e Fim são informados através de uma caixa de texto com uma máscara <*p:inputMask*> que é preenchida pelo usuário. Já os campos Status, Cliente, Recurso e Conta são informados através de uma caixa de seleção <*p:selectOneMenu*>. A caixa de

seleção do campo Cliente, por exemplo, é construída pelo atributo “dao” do *bean* gerenciado controleCliente que chama o método listarTodos().

No rodapé do diálogo, foi inserido um elemento chamado <*p:commandButton*> que, ao ser pressionado, invoca o ativa o método salvar( ) do *bean* gerenciado, persistindo um horário na base de dados da aplicação.

A figura 24 apresenta a tela da página listar.xhtml com a caixa de diálogo.

Figura 24: Visualização da página listar.xhtml - Dialogo



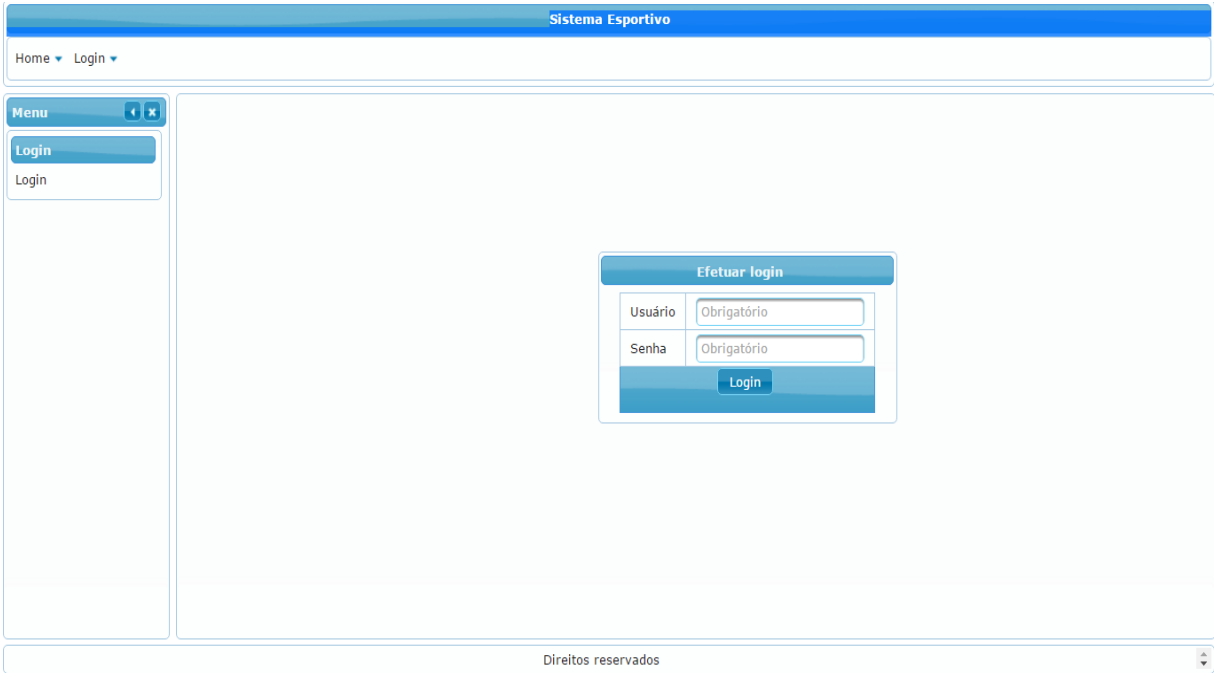
Fonte: Próprio Autor

4.9 RESULTADOS

A seguir será apresentado como ficou o funcionamento da aplicação, desde a realização do login pelo usuário até a realização das funções básicas de criação, leitura, atualização e exclusão e logout do sistema.

Ao acessar a aplicação o usuário é direcionado para uma tela de login, onde é necessário informar um usuário e senha válidos. A figura 25 apresenta a tela de login.

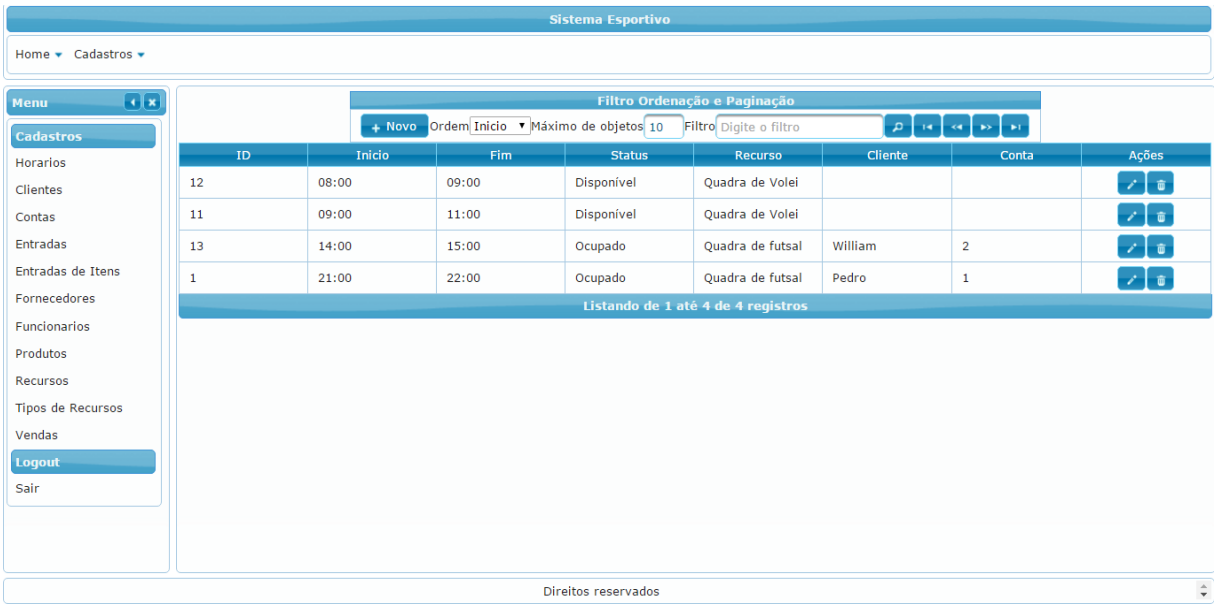
Figura 25: Visualização da página de Login - login.xhtml



Fonte: Próprio Autor

Após realizar login com sucesso, o usuário é direcionado até a pagina de listagem de horários, apresentada na figura 26. Nesta página é possível visualizar todos os horários cadastrados na base de dados e realizar manutenções.

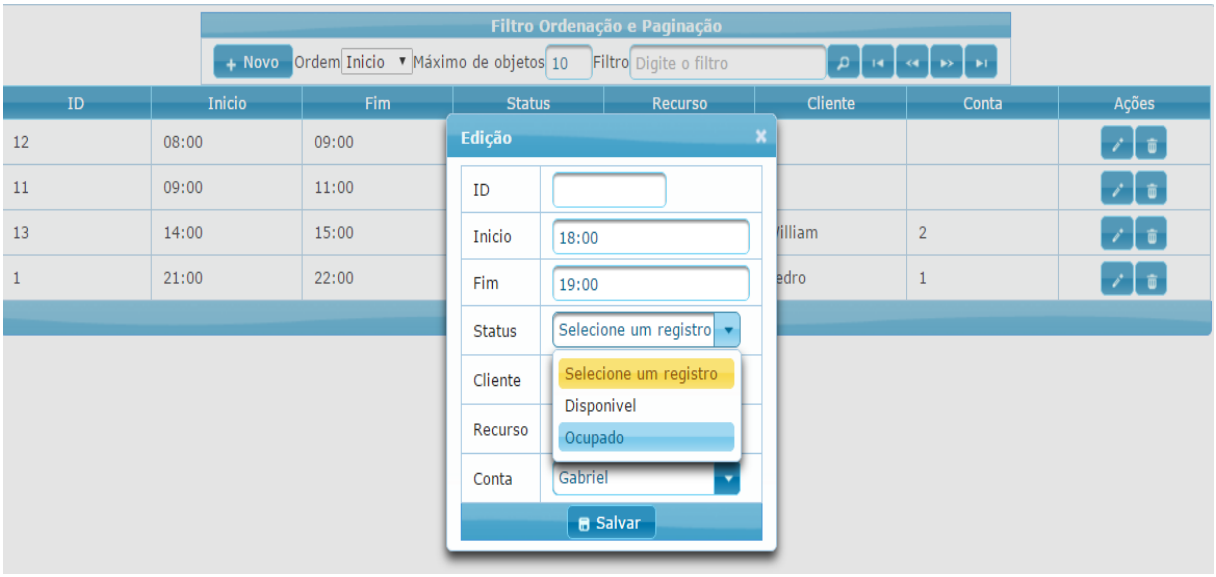
Figura 26: Visualização da pagina de manutenção de Horários



Fonte: Próprio Autor

Para cadastrar um novo horário é necessário clicar no botão “Novo”. Uma caixa de diálogo será aberta no sistema onde o usuário deverá informar os dados do novo horário cadastrado. A figura 27 apresenta a tela com o diálogo.

Figura 27: Inserindo um novo registro na manutenção de Horários



Fonte: Próprio Autor

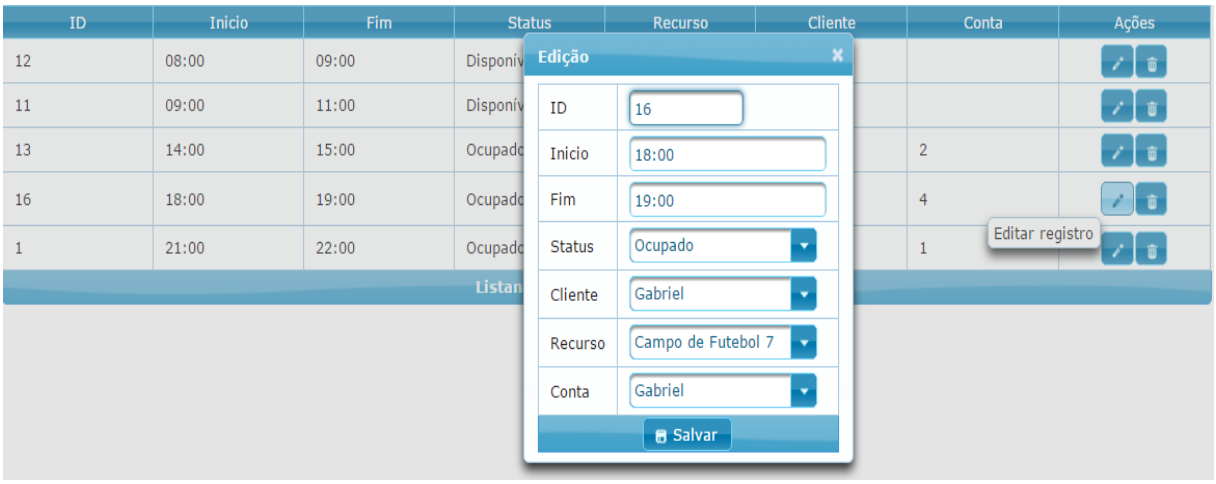
Ao clicar no botão salvar a pagina de listagem é atualizado e uma mensagem aparece na tela informando que um novo objeto foi persistido na base de dados. A figura 28 apresenta a pagina de listagem de horários após a inserção de um novo objeto.

Figura 28: Visualização da pagina de listagem de Horários - novo registro inserido

Fonte: Próprio Autor

É possível editar um registro ao clicar no primeiro ícone do menu “Ações”. Um diálogo será aberto com todos os dados do objeto que se deseja editar já carregados em tela. A figura 29 apresenta a tela após o usuário clicar no ícone de edição.

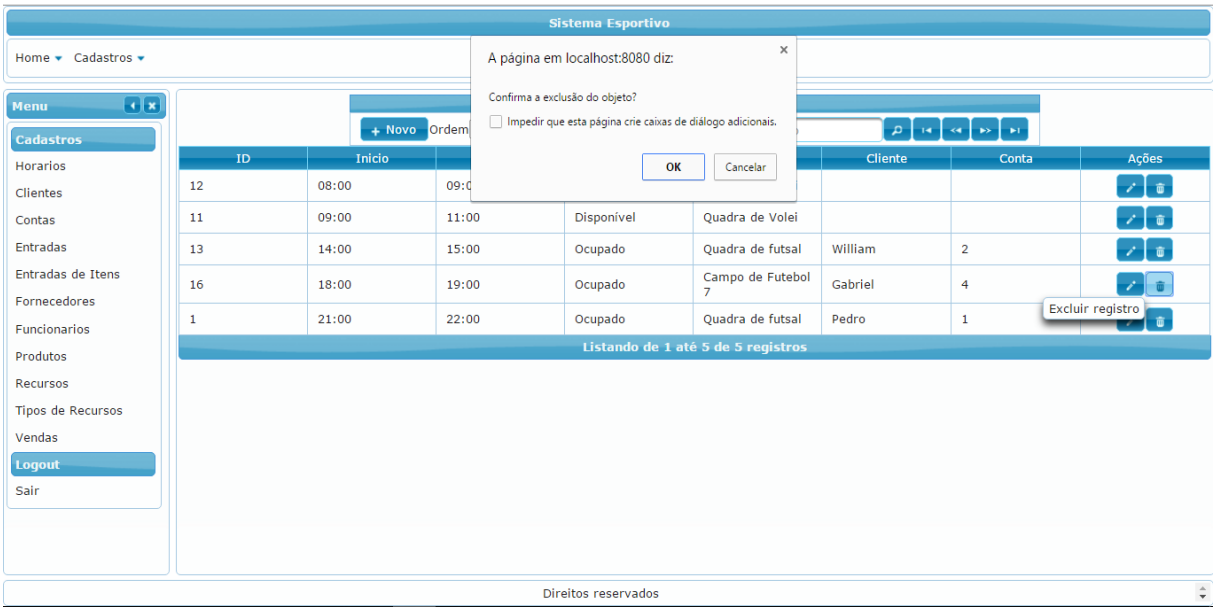
Figura 29: Visualização da pagina de Manutenção de Horários - botão de edição



Fonte: Próprio Autor

Ao clicar em salvar, a página de listagem é atualizada da mesma maneira ao se inserir um novo objeto. É possível também realizar a exclusão de um registro clicando no segundo ícone do menu ações. Ao clicar em “Excluir registro” uma caixa de diálogo é aberta no navegador solicitando a confirmação da exclusão como mostra a figura 30.

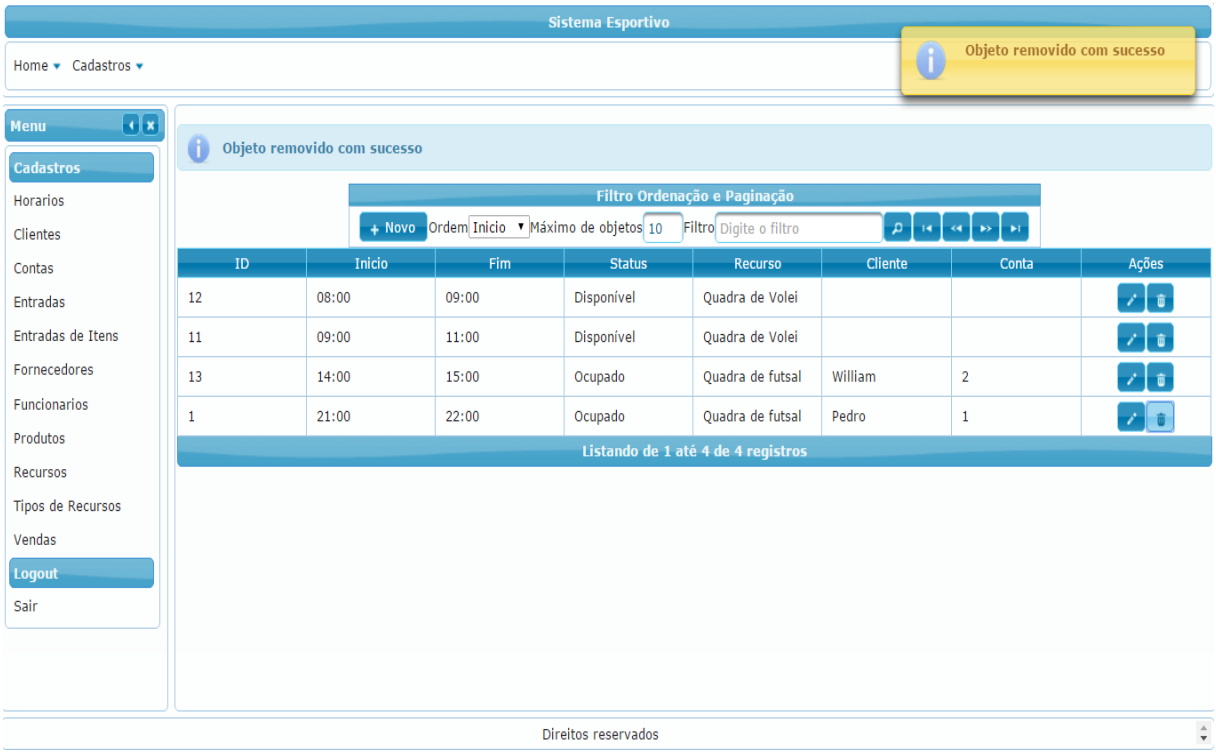
Figura 30: Visualização da página de listagem de Horários ao clicar no ícone de exclusão



Fonte: Próprio Autor

Ao confirmar a exclusão a tela de listagem é atualizada e uma mensagem informando que o objeto foi removido com sucesso surge na tela. A figura 31 apresenta o resultado.

Figura 31: Visualização da pagina de listagem de Horários após excluir um registro



Fonte: Próprio Autor

Para sair do sistema o usuário deve clicar na opção sair localizada no ultimo item do menu a esquerda da tela, como mostra a figura 32. Ao fazer isso o usuário é direcionado para a pagina de login.

Figura 32: Visualização da tela de listagem de Horários mostrando a localização da opção de sair do sistema

Fonte: Próprio Autor

5 CONSIDERAÇÕES FINAIS

Devido ao constante crescimento de sistemas informatizados presentes em nosso dia-a-dia, e com a facilidade em obter-se recursos tecnológicos, o desenvolvimento de um sistema que auxilie o gerenciamento de recursos esportivos poderá trazer diversos benefícios para uma empresa que não utiliza as tecnologias existentes para facilitar seu trabalho diário. Este trabalho teve como objetivo realizar estudos sobre as tecnologias de desenvolvimento de sistemas web que utilizam a linguagem de programação Java. Da mesma forma, foram estudadas a API *JavaServer Faces*, a utilização de JPA para realizar a persistência de dados com *Hibernate*, conceitos de banco de dados, da linguagem UML e uma análise para realizar o desenvolvimento de um sistema de gerenciamento de recursos esportivos. Com a utilização a plataforma Java EE para desenvolver a aplicação, o trabalho proposto foi facilitado devido ao trabalho em conjunto com JPA, o provedor *Hibernate*, o framework JSF e a biblioteca de componentes do *Primefaces.*

O início do trabalho trouxe algumas dificuldades devido às primeiras configurações que são necessárias realizar no ambiente de desenvolvimento, como por exemplo, a criação do pool de conexões e da unidade de persistência de dados. Porém, após muitos estudos e pesquisas, o desenvolvimento do sistema foi desenvolvido com mais facilidade e com muita rapidez, visto que a maioria dos módulos possui implementação e funcionamento semelhantes, facilitando a reutilização de códigos. Outro ponto positivo foi a utilização da biblioteca de componentes *Primefaces* que possui muitas ferramentas e uma boa documentação que auxilia o programador na criação das páginas web.

Na versão atual do sistema é possível realizar as funções básicas de criação, leitura, atualização e exclusão. Dessa forma é possível realizar as manutenções em todos os módulos do sistema, tornando-o funcional. Foi implementado também um sistema de login para acessar os módulos do sistema. Como trabalhos futuros, pretende-se realizar a geração de relatórios no sistema, restringir o acesso do cliente a certos módulos, além de continuar melhorando a parte visual do sistema, a fim de melhorar o serviço disponibilizado.

6 REFERÊNCIAS

MATTOS, Érico Casella Tavares. **Programação de Software em Java.** São Paulo: Digerati Books, 2007. Disponível em: <https://goo.gl/NKNvhD>. Acesso em: 18 de Abril de 2015

GONÇALVES, Rodrigo. **Universo Java.** São Paulo: Digerati Books, 2008. Disponível em: <https://goo.gl/qsMdRZ>. Acesso em: 18 de Abril de 2015

SAMPAIO, Cleuton. **Java Enterprise Edition 6: desenvolvendo aplicações corporativas.** Rio de Janeiro: Brasport, 2011. Disponível em: <https://goo.gl/i29jui>. Acesso em: 16 de Maio de 2015

JENDROCK, Eric. et al. **The Java EE 6 Tutorial: Basic Concepts.** 4. ed. 2011. Disponível em: <https://goo.gl/YXdT34>. Acesso em: 23 de Maio de 2015

BERGSTEN, Hans. **JavaServer Faces.** Sabastopol: O'Reilly Media, Inc.; 2004. Disponível em: <https://goo.gl/YXdT34>. Acesso em: 16 de Maio de 2015

LUCKOW, Décio Heinzelmann; MELO, Alexandre Altair de. **Programação Java para a Web.** São Paulo: Novatec, 2010. Disponível em: <https://goo.gl/21CxAc>. Acesso em: 16 de Maio de 2015

HLAVATS, Ian. **Instant PrimeFaces Starter.** Birmingham: Packt Publishing Ltd., 2013. Disponível em: <https://goo.gl/pB8M8B>. Acesso em: 3 de Junho de 2015

FEITOSA, Diego Bomfim. **Visão geral sobre Primefaces.** Sergipe, 2010. Disponível em: <https://williamgamers.wordpress.com/2012/06/04/visao-geral-sobre-primefaces/>. Acesso em: 3 de Junho de 2015

MEDEIROS, Higor. **Introdução à JPA – Java Persistence API.** 2013. Disponível em: <http://www.devmedia.com.br/introducao-a-jpa-java-persistence-api/28173#ixzz3d6Pxozn5> Acesso em: 3 de Junho de 2015

PAGANINI, Silvio. **JPA 2.0 – Persistência a toda prova.** 2010. Disponível em: <http://www.devmedia.com.br/jpa-2-0-persistencia-a-toda-prova-java-magazine-81/17437>. Acesso em: 6 de Junho de 2015

VOGEL, Lars. **JPA 2.0 with EclipseLink – Tutorial.** Vogella, 2012. Disponível em: <http://www.vogella.com/tutorials/JavaPersistenceAPI/article.html>. Acesso em: 6 de Junho de 2015

MELO, Ana Cristina. **Desenvolvendo aplicações com UML 2.2: do conceito a implemetação.** 3. ed. Rio de Janeiro: Brasport, 2010. Disponível em: <https://goo.gl/4skQdN>. Acesso em: 7 de Junho de 2015

BOOCH, Grady. et al. **UML, O guia do usuário.** Rio de Janeiro: Elsevier, 2006. Disponível em: <https://goo.gl/3SYwjo>. Acesso em: 7 de Junho de 2015.