Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação

DAGSON PATRICK VIEIRA DE SOUZA

DESENVOLVIMENTO DE UMA APLICAÇÃO FINANCEIRA PESSOAL PARA WEB

Belo Horizonte
2016

Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciências da Computação
Especialização em Informática: Ênfase: Engenharia de Software

DESENVOLVIMENTO DE UMA APLICAÇÃO FINANCEIRA PESSOAL PARA WEB

por

DAGSON PATRICK VIEIRA DE SOUZA

Monografia de Final de Curso
CEI-ES-0xxx-T20-2016-01

Prof. Dr. Roberto da Silva Bigonha

Belo Horizonte
2016

DAGSON PATRICK VIEIRA DE SOUZA

DESENVOLVIMENTO DE UMA APLICAÇÃO FINANCEIRA PESSOAL PARA WEB

Monografia apresentada ao Curso de Especialização em Informática do Departamento de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Especialista em Informática.

Área de concentração: Engenharia de Software

Orientador(a): Prof. Dr. Roberto da Silva Bigonha

Belo Horizonte
2016

Ficha catalográfica elaborada pela Biblioteca do ICEx - UFMG

Souza, Dagson Patrick Vieira de.

S729d Desenvolvimento de uma aplicação financeira pessoal para web. / Dagson Patrick Vieira de Souza. Belo Horizonte, 2016.
x, 90 f.: il.; 29 cm.

Monografia (especialização) - Universidade Federal de Minas Gerais – Departamento de Ciência da Computação.

Orientador: Roberto da Silva Bigonha.

1. Computação 2. Engenharia de software. 3. Java (Linguagem de programação de computador). I. Orientador. II.Título.

CDU 519.6*32(043)

Dedico este trabalho
a Deus, aos professores,
e aos meus familiares pelas
orações e incentivos que depositaram
em mim para conclusão desta
tal sonhada pós-graduação

UNIVERSIDADE FEDERAL DE MINAS GERAIS

INSTITUTO DE CIÊNCIAS EXATAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
ESPECIALIZAÇÃO EM INFORMÁTICA: ÁREA DE CONCENTRAÇÃO ENGENHARIA DE SOFTWARE

DESENVOLVIMENTO DE UMA APLICAÇÃO FINANCEIRA PESSOAL PARA WEB

DAGSON PATRICK VIEIRA DE SOUZA

Monografia apresentada aos Senhores:

Prof. Roberto da Silva Bigonha
Orientador
DCC - ICEx - UFMG

Profa. Mariza Andrade da Silva Bigonha
DCC - ICEx - UFMG

Belo Horizonte, 29 de fevereiro de 2016

Agradecimentos

Se diante dessa longa caminhada foi possível escrever esta monografia, agradeço primeiramente a Deus, por ter me dado força e determinação para vencer todos os obstáculos que surgiram ao longo dessa pós-graduação. Contudo, sinto-me realizado por ter alcançado esse titulo de especialista em Engenharia de Software tão almejado em uma universidade pública e de qualidade. Essa pós-graduação contribuiu muito para minha vida pessoal e profissional ao longo de um ano cursado.

Agradeço também a Universidade Federal de Minas Gerais: corpo docente, técnico administrativo, funcionários, colegas de sala por terem me agregado conhecimentos e experiência de vida ao logo desse curso. Tudo isso será de muita importância para minha carreira profissional.

Agradeço em especial ao professor Roberto da Silva Bigonha por ter aceitado ser meu orientador e ter acreditado em minha capacidade de realizar este trabalho. Confesso que todas as suas orientações foram de grande importância para realizar este trabalho de conclusão de curso. Além disso, pode contando com sua disponibilidade, paciência e conhecimento transmitido durante dois semestres consecutivos, tudo isso foi imprescindível para o desenvolvimento deste projeto.

Agradeço também em especial a professora Maria de Lourdes Coelho pela sua paciência e pelas suas instruções que de fato foi fundamental para concluir este trabalho.

Agradeço a todos os meus amigos que estudamos juntos no decorre dessa pós-graduação, em especial; André Araújo, Antônio Souza, Marcelo Santos, Rodrigo Nascimentos, Daniel Felizardo, Kênia de Jesus. E finalmente, agradeço também a minha amada namorada, Priscila Bifano pela paciência e compreensão que teve comigo no decorre dessa pós-graduação “Te Amo Amore”. Agradeço o apoio e compreensão de todos familiares nesse período tão marcante de minha vida.

Abraços a Todos!

Resumo

Com a evolução da Internet e das tecnologias de informação, surgiu a necessidade de desenvolver software *web* com mais qualidade e segurança. Para acompanhar esse fenômeno tecnológico, o universo JAVA, oferece diferentes tipos de recursos tecnológicos, cada um com sua área de atuação bem definida. Este trabalho tem por objetivo apresentar e aplicar recursos tecnológicos já existentes para o desenvolvimento de software na plataforma Java para *web*. A fim de aplicar essas técnicas, foi escolhido uma aplicação financeira pessoal para ser desenvolvida como referência. A aplicação desenvolvida utiliza diversas técnicas para desenvolvimento de software *web* em plataforma Java EE. A tecnologia JavaServer Faces e o *framework* de componente PrimeFaces, juntos, oferecem diferentes componentes de interface para usuário. Essas tecnologias otimizam a fase de desenvolvimento de software voltado para a *web*. As ferramentas como o Hibernate/JPA fornecem inúmeras funcionalidades, assim sendo possível criar uma aplicação com mais agilidade. Dentre suas funcionalidades, a que se destaca é capacidade de realizar o mapeamento Objeto-Relacional para a linguagem de programação Java. O Java *Persistence* API (JPA) é também uma API padrão da linguagem Java que especifica uma interface comum para *frameworks* de persistência de dados. Logo, a JPA auxilia a padronizar a interação da aplicação com o banco de dados. Como alvo de estudo deste trabalho foi desenvolvido uma aplicação financeira pessoal utilizando essas técnicas de desenvolvimento. Dentre diferentes tecnologias utilizadas nessa aplicação, as principais foram o JSF e o Hibernate. Essas técnicas utilizadas juntas oferecem ao desenvolvedor agilidade, manutenibilidade, portabilidade, reusabilidade e principalmente no quesito da evolução do software. Combinando essas duas tecnologias foi possível criar uma interface *web* mais amigável, automatizar operações em banco de dados, tratar questões de segurança, aplicar a internacionalização em páginas *web* e gerar relatórios.

Palavras-chave: JavaServer Faces, Hibernate, *Web*, Facelets, *Framework*, Java, Aplicação Financeira, Software, *Enterprise Edition.*

Abstract

With the evolution of the Internet and information technologies, the need to develop web software with more quality and safety. To keep up with this technological phenomenon, the JAVA universe offers different types of technological resources, each with its well-defined scope. This paper aims to present and apply technological resources already exist for software development on the Java platform for web. In order to apply these techniques, a personal financial investment to be developed as reference was chosen. The developed application uses are various techniques for web software development in Java EE platform. The Java Server Faces technology and Prime Faces component framework together provide for different user interface components. These technologies optimize the software development phase focused on the web. Tools like Hibernate / JPA provide numerous features, so it is possible to create an application more quickly. Among its features, what stands out is the ability to perform object-relational mapping for Java programming language. The Java Persistence API (JPA) is also a standard Java API that specifies a common interface for data persistence frameworks. Therefore, the JPA helps to standardize the application of interaction with the database. As this work subject of study was developed a personal financial application using these development techniques. Among different technologies used in this application, the main ones were the JSF and Hibernate. Together these techniques offer the agile developer, maintainability, portability, reusability and specially regarding the evolution of the software. Combining these two technologies was possible to create a more user-friendly web interface, automate operations in the database, handling security issues, apply the internationalization of web pages and generate reports.

Keywords*:* *Java Server Faces, Hibernate, Web, Facelets Framework, Java, Financial Application Software, Enterprise Edition.*

Lista de Figuras

Lista de Tabelas

Lista de Siglas

EE	*Enterprise Edition*
API	*Application Programming Interface*
XML	*eXtensible Markup Language*
JDBC	*Java Database Connectivity API*
JPA	*Java Persistence API*
JTA	*Java Transaction API*
JSF	*JavaServer Faces*
HQL	*Hibernate Query Language*
JSR	*Java Specification Request*
HTML	*Hyper Text Markup Language*
XHTML	*Extensible Hyper Text Markup Language*
JPQL	*Java Persistence Query Language*
EJB	*Enterprise Java Beans*
POJO	*Plain Old Java Objects*
SGDB	Sistema de Gerenciamento de Banco de Dados
SQL	*Structured Query Language*
GUI	*Graphical User Interface*
ORM	*Object Relational Mapping*
JCP	*Java Community Process*
JMS	*Java Message Service*
JCA	*Java Connector Architecture*
DAO	*Data Access Object*

Sumário

1. Introdução

Com o atual momento da economia Brasileira, surge à necessidade das pessoas tomarem conhecimento das suas próprias finanças. Segundo informações do Serasa Experian (2015), órgão responsável pelo acesso e proteção ao crédito do país, os índices de inadimplência vem se elevando nos últimos anos. Atualmente, a sociedade deve ter conhecimento sobre finanças pessoais. Segundo a teoria de Cherobim e Espejo (2010), finanças pessoais é a ciência que estuda a aplicação de conceitos financeiros nas decisões financeiras de uma pessoa.

Quando as pessoas tem o conhecimento das suas contas de receitas e despesas, os mesmos podem acompanhar periodicamente suas contas e elaborar um planejamento. Ainda segundo Hoji (2010), administrar rendimentos sem um norte físico é como andar no escuro. As pessoas precisam fazer uma gestão financeira pessoal ou familiar, mesmo com bons rendimentos, é necessário ter uma educação financeira. Uma tarefa importante relacionada a finanças pessoais é o controle doméstico ou individual. Faz se necessário acompanhar para onde se destina seus gastos.

Nesse contexto, e como objetivo especifico desse trabalho foi desenvolvida uma aplicação financeira pessoal para *web* para realizar a gestão financeira dos usuários cadastrados. A aplicação permite os usuários lançar todos os gastos, até mesmo os mais simples. Assim, os usuários podem acompanhar diversos tipos de gastos pessoais no decorrer de sua vida financeira.

A principio, o sistema deve disponibilizar de forma geral um cadastro de usuário. Para cada usuário, o sistema vai armazenar as seguintes informações: nome, data de nascimento, número de celular, e-mail, idioma, *login* e senha.

O sistema oferece para cada usuário um controle de contas bancárias. Logo, um usuário cadastrado no sistema vai poder cadastrar diversas contas bancárias de sua preferência. Esse cadastro permite que o usuário visualize o formulário de cadastro de contas e a listagem de todas as contas já cadastradas, na mesma página. Para cada conta, o sistema deve armazenar as seguintes informações: descrição da conta, data do cadastro e saldo inicial. O sistema também oferece aos usuários uma opção de escolher uma conta como favorita. Um requisito funcional do sistema é que todo usuário recém-cadastrado tem um conjunto básico de categorias de despesas e

receitas cadastradas para que não seja necessário um cadastro partindo do zero. O sistema também armazena a data que cada conta foi cadastrada pelos usuários. O sistema trabalha com uma conta bancária ativa para cada usuário e será a conta favorita. No decorrer do tempo o usuário poderá alterar essa informação. Quando o usuário realizar o *login* no sistema ele visualizará suas despesas relacionadas à sua conta favorita, mas depois poderá alterar para outra conta e visualizar as despesas vinculadas a essa conta.

O sistema também oferece uma área administrativa, para o usuário administrador do sistema. Nessa área, o usuário administrador pode editar, excluir, ativar e desativar todos os usuários cadastrados no sistema. A área administrativa permite ao usuário administrador ter uma visão geral de todos os usuários cadastrados no sistema até aquele determinado momento. Apenas alguns usuários vão ter acesso á essa área administrativa.

Para deixar o sistema ainda mais seguro, o sistema garante a autenticação e a autorização dos usuários. Assim, somente quem realmente é cadastrado no sistema tem acesso aos recursos restritos. O sistema também disponibiliza recurso para usuários não cadastrados, ou seja, páginas públicas às quais qualquer usuário visitante terá acesso. As páginas restritas somente são acessíveis após um processo de *login*. Logo, o sistema é composto por 4 níveis de autorização: usuário público, usuário normal, usuário administrador e usuário VIP. Resumidamente, o sistema tem a capacidade de restringir o acesso a determinadas páginas do sistema. O sistema também oferece recurso de gerenciar permissões, ou seja, incluir ou excluir permissões para um determinado usuário. O sistema garante que, se um usuário for excluído do sistema, seus registros de contas também sejam excluídos.

As categorias servem para classificar os lançamentos financeiros que o usuário realiza. Um lançamento financeiro é qualquer registro de despesa ou receita que o usuário registra no sistema. As categorias estão organizadas em árvore separando em subdivisões de categorias. O sistema já possui cadastradas duas categorias principais, que são despesa e receita. Mas o usuário pode também fazer qualquer manutenção na estrutura como excluir, alterar ou incluir nova categoria.

1.1. Objetivo Geral

Este trabalho tem como objetivo geral apresentar e aplicar técnicas já existentes para o desenvolvimento de software em plataforma JAVA para *Web*. Como alvo de estudo deste trabalho foi escolhido uma aplicação financeira pessoal para ser desenvolvida. Pretende-se apresentar as vantagens que essas técnicas oferecem para o desenvolvimento de software em plataforma Java EE (*Enterprise Edition*).

1.2. Objetivo Específico

O objetivo específico deste trabalho é desenvolver uma aplicação financeira pessoal utilizando as tecnologias oferecidas pela plataforma JAVA para *Web*. Aplicar a tecnologia JavaServer Faces (JSF) utilizando *framework* de componentes como Myfaces, ICEFaces, RichFaces e PrimeFaces. Utilizar as ferramentas Hibernate e JPA para ajudar a padronizar a interação da aplicação com o banco de dados. Aplicar técnica de desenvolvimento em camadas e MVC (Model-View-Controller), assim oferecendo mais controle da aplicação. Para o aspecto visual da aplicação será utilizado o framework de template Facelets, CSS e Tableless, com objetivo de criar uma aplicação com o visual atrativo, agradável e de fácil manutenção. Aplicar recursos de internacionalização com três idiomas diferentes, conforme a preferência do usuário. Gerar gráficos e relatórios em formato PDF com auxilio das ferramentas iReport e JasperReport. Aplicar recurso de segurança usando o Spring Security para realizar o controle de usuário. Assim, permitir o desenvolvimento de software com mais qualidade, segurança e organização. E finalmente, realizar a integração da aplicação financeira pessoal com o portal de finanças do Yahoo.

1.3. Estrutura da Monografia

O restante do trabalho é organizado da seguinte maneira, Capítulo 2 apresenta a fundamentação teórica que são os principais conceitos, tais como definição, elementos, características e técnicas existentes para o desenvolvimento de software em plataforma JAVA para *Web*. Capítulo 3 apresenta a metodologia que será utilizada para alcançar os objetivos desse trabalho, mostrando algumas ferramentas e recursos tecnológicos utilizados para o desenvolvimento deste trabalho. Capítulo 4 apresenta detalhes do desenvolvimento da aplicação financeira pessoal.

No Capítulo 5 são apresentados os resultados alcançados. Capítulo 6 apresenta as considerações finais deste trabalho. E, por fim, são apresentadas as referências bibliográficas que foram utilizadas como base para consultas.

2. Fundamentação Teórica

Este capítulo apresenta a fundamentação teórica que são os principais conceitos teóricos e ferramentas usadas no desenvolvimento deste trabalho. O Eclipse é o ambiente de programação utilizado para implementar a aplicação financeira pessoal. É um ambiente de software gratuito que é desenvolvido por pessoas voluntárias. Em particular, a plataforma Java EE foi escolhida devido sua popularidade e capacidade de enfrentar problemas que às aplicações *web* possuem. Desenvolvedores não precisam gastar tempo com questões de infraestrutura, o Java EE é composto de uma série de especificações. Essas especificações apresentam detalhes de como deve ser implementado um software. Para cada especificação existem uma ou mais implementação, tanto open *source* quanto pagas. As principais especificações utilizadas na aplicação financeira foram o JSF e o JPA, a qual foram escolhidas para agilizar o processo de desenvolvimento.

2.1. Plataforma Java EE

Plataforma Java EE (*Enterprise Edition*) é uma plataforma para desenvolvimento e execução de software para atender às necessidades do mercado corporativo. Os chamados aplicativos empresariais são projetados para resolver os problemas enfrentados por grandes empresas. Esses aplicativos são úteis, até mesmo essenciais, para desenvolvedores individuais, organizações de pequeno, médio e grande porte (SOUZA, 2015). As características de segurança e confiabilidade tornam esses aplicativos, muitas vezes mais complexos. A plataforma Java EE reduz a complexidade de desenvolvimento de aplicações corporativas, fornecendo um modelo de desenvolvimento, API, e ambiente de tempo de execução que permitem que os desenvolvedores se concentrem em funcionalidades (ORACLE).

A plataforma Java EE foi construída em cima da plataforma Java SE (*Standard Edition*), que é uma plataforma que define as principais características e funcionalidades da linguagem de programação Java. A plataforma Java SE define tipos básicos, classes de alto nível que são usados para redes, segurança, acesso a

banco de dados, interface gráfica do usuário (GUI – *Graphical User Interface*) desenvolvimento e análise de XML (*eXtensible Markup Language*). A plataforma Java EE fornece um ambiente de tempo de execução e API (*Application Programming Interface*) para desenvolvimento e execução de grande escala, multicamadas, aplicações de rede, confiáveis e seguras (GONCALVES, 2011). A plataforma Java EE melhora a portabilidade de aplicações e aumenta a produtividade do desenvolvedor em projeto de software corporativo. Java EE também pode ser considerado informalmente como um padrão, porque os fornecedores de software devem atender certos requisitos para declarar que seus produtos são compatíveis com Java EE (ORACLE).

Uma das grandes vantagens de utilizar Java para o desenvolvimento de aplicação é o custo baixo: a implementação Java EE da Oracle *Corporation* pode ser baixada gratuitamente pelo site da Oracle. Há também muitas ferramentas *open source* disponíveis para auxiliar ou até mesmo simplificar o desenvolvimento de software utilizando a linguagem de programação Java. O universo Java ainda conta com diversas ferramentas de desenvolvimento de terceiros, de código aberto como: NetBeans IDE, Eclipse, JDeveloper, JBuilder, Apache *Software Foundation* Apache Ant, Apache *Software* Foundation Apache Maven, Apache *Software Foundation* Apache Tomcat, Jetty, Spring e dentre outras ferramentas existentes.

A especificação J2EE original foi desenvolvida pela empresa Sun *Microsystems*. Atualmente, a plataforma Java EE é desenvolvida pela Java *Community Process*, com contribuições de especialistas da indústria, organizações comerciais e *open source*, Java *User Groups* e inúmeros indivíduos (LUCKOW e MELO, 2010). A cada versão são adicionados novos recursos para atender às necessidades da indústria. A Tabela 2.1 a seguir apresenta o histórico de versões da plataforma JAVA EE, lembrando que até a escrita deste trabalho, a Versão 8 já estava sendo desenvolvida pela *Community Process* e de seus colaboradores.

Versões	Data Publicação
J2EE 1.2	dezembro de 1999
J2EE 1.3	setembro de 2001
J2EE 1.4	novembro de 2003
Java EE 5	maio de 2006
Java EE 6	dezembro de 2009
Java EE 7	abril de 2013

Tabela 2.1: histórico de versões da plataforma Java EE

2.2. Camadas de software

A funcionalidade do aplicativo pode ser separada em áreas funcionais isoladas, chamadas de camadas (SCOTT, 1998). As camadas inclui o pensamento de separação de responsabilidades, que precisa ser bem-definida na arquitetura do projeto. A responsabilidade de cada camada é do tipo: gravar informações em banco de dados; tomar uma determinada decisão em situação especifica; apresentar para o usuário o layout da página com os dados. Normalmente, as aplicações multicamadas tem uma camada cliente, uma camada intermediária e uma camada de dados (CALÇADO, 2005). A camada do cliente consiste em um programa cliente que faz pedidos para a camada intermediária. A camada intermediária é dividida em uma camada da web e uma camada de negócio, que lida com solicitações do cliente, processando os dados e armazenando-o na camada de dados. Desenvolvimento de aplicações Java EE se concentra na camada intermediária para tornar o gerenciamento do aplicativo mais fácil, robusto e mais seguro (ORACLE).

As camadas de software permitem distribuir melhor as funções de uma aplicação. As camadas definem a direção do fluxo de mensagens entre certos tipos de classes. A direção do fluxo de mensagens entre as camadas normalmente é pré-definida, esse detalhe deve ser observado pelo desenvolvedor. Java não oferece recursos linguísticos para controlar a direção do fluxo de mensagens. É de total

responsabilidade do programador, definir e controlar a direção do fluxo de mensagens entre camadas (ARNOLD ET AL, 1997). A mensagem significa chamar qualquer método de um objeto, equivale a uma solicitação de serviço a um objeto. A resposta do objeto receptor da mensagem não é considerada uma mensagem, a mensagem é só na direção do objeto receptor. A Figura 2.1 apresenta em uma abstração de alto nível, um modelo de quatro camadas.

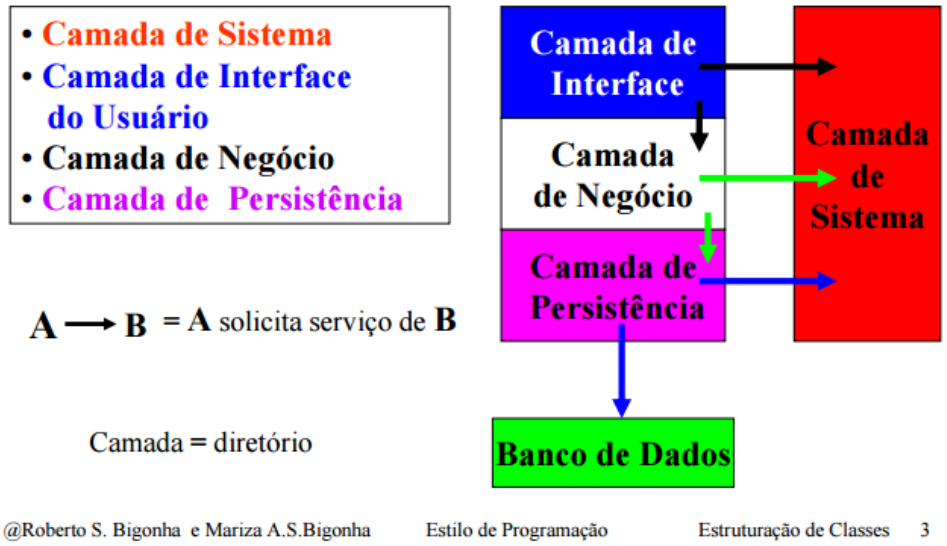


Figura 2.1 – Modelo de quatro camadas[1]

2.2.1. Camada Cliente

A camada cliente consiste em aplicativos clientes que acessam um servidor Java EE e que estão normalmente localizadas em uma máquina diferente do servidor. Os clientes fazem solicitações para o servidor. O servidor processa as solicitações e devolve uma resposta ao cliente. Diferentes tipos de aplicações podem ser aplicativos clientes Java EE. Os clientes podem ser um navegador Web, um aplicativo independente, ou até outros servidores, que estejam localizados em máquinas diferentes da maquina servidora (FOWLER *et al*, 2002).

[1] **Fonte:** nota aula (BIGONHA, R. S; BIGONHA, M. A. S, 2015)

2.2.2. Camada Web

A camada *web* consiste em componentes que lidam com a interação entre clientes e a camada de negócio. Suas principais tarefas são as seguintes:

- Gerar conteúdo dinâmico em vários formatos para o cliente
- Capturar entrada de usuários da interface do cliente e retornar resultados apropriados de acordo com a camada de negócio
- Controlar o fluxo de telas ou páginas no cliente
- Manter o estado dos dados para uma sessão do usuário
- Executar a lógica básica e manter alguns dados temporariamente em *Beans* gerenciados

Bean é um padrão que especifica como escrever classes Java. Basicamente, é uma classe Java que tem um construtor público sem parâmetros, e para cada atributo *private* existe um método *setter* e *getter*. A classe é escrita de acordo com uma convenção particular. Alguns *frameworks,* como o JPA, exige que suas classes estejam nesse padrão. Segundo a *Sun Microsystems,* JavaBeans são componentes reutilizáveis de software escritos na linguagem de programação Java.

A Tabela 2.2 lista algumas das principais tecnologias Java EE que são usadas na camada *web* em Aplicativos Java EE.

A camada de interface é um conjunto de classes que permitem implementar a interação dos usuários com a aplicação. Essa camada é responsável pela interface gráfica do usuário (GUI). As mensagens devem fluir da camada de interface para a camada de negócio, e nunca no sentido contrário. Se ocorrer alguma mudança na interface, essa mudança não afeta a camada de negócio. Nunca deve haver fluxo de mensagens entre a camada de interface e a camada de persistência (SCOTT, 1998).

Tecnologia	Finalidade
JavaServer Faces	Um *framework* de componentes para interface de usuário *web*, que permite que os desenvolvedores incluam componentes UI (tais como campos e botões) em uma página XHTML, chamada de página Facelets. Esses componentes de interface, converte, valida e salva os dados dos componentes de UI para posteriormente armazená-los no servidor.
Expression Language	Um conjunto de *tags* padrão utilizadas em páginas Facelets para referir a componentes de Java EE.
Servlets	Classes da Linguagem de programação Java, que tem a finalidade de processar dinamicamente requisições e construir respostas, geralmente para páginas HTML.
Contexts and Dependency Injection	Um conjunto de serviços contextuais que tornam mais fácil para os desenvolvedores usar as *enterprise beans*, juntamente com JavaServerFaces tecnologia em aplicações web (FOWLER, 2007).

Tabela 2.2: tecnologia Java EE da camada Web

Enterprise JavaBeans, ou EJBs, são componentes utilizados em servidores e são parte da plataforma Java EE. É um dos principais componentes da plataforma Java *Enterprise Edition.* EJB roda em *contêiner* de servidor de aplicação, com base em componentes distribuídos. Atualmente na Versão 3.2, EJB é definido entre grandes empresas com IBM, Oracle como também a comunidade JCP. A Tabela 2.3 a seguir apresenta o histórico de versões da EJBs, lembrando que é até a data da escrita deste trabalho.

Versões	Data de publicação
EJB 1.0	março de 1998
EJB 1.1	dezembro de 1999
EJB 2.0	agosto de 2001
EJB 2.1	novembro de 2003
EJB 3.0	maio de 2006
EJB 3.1	dezembro de 2009
EJB 3.2	maio de 2015

Tabela 2.3: histórico de versões Enterprise Java Beans

O compomente EJB possui três tipos fundamentais: *entity beans*, *session Beans* e *Message Driven Beans*. *Entity Beans* é um objeto que vai ser persistido em uma base de dados. *Session Beans* executa uma tarefa para o cliente, mantem o estado durante uma sessão com o cliente. Message Drive Beans processa mensagens de modo assíncrono entre os EJB's, cuja API de mensagens é JMS.

2.2.3. Camada de Negócio

A camada de negócio consiste em componentes que fornecem a lógica de negócio para uma aplicação. A lógica de negócio é o código que fornece funcionalidade para um domínio de negócio em particular, como o setor financeiro, ou um *site* de *e-commerce* (LUCKOW e MELO, 2010). As seguintes tecnologias Java EE estão entre aquelas que são usadas na camada de negócio em aplicações Java EE:

- *Enterprise JavaBeans (enterprise bean) components*
- *JAX-RS RESTful web services*
- *Java Persistence API entities*

Essa camada encapsula um conjunto de classes do domínio da aplicação ou do negócio. Essa camada não se preocupa com a camada de interface e camada de dados. Classes de negócio são aquelas classes identificadas durante a fase de análise de requisitos. O fluxo de mensagens é sempre no sentido da camada de interface para a camada de negócio (BIGONHA, R. S; BIGONHA, M. A. S, 2015). A camada de negócio fica responsável para comunicar com a camada de persistência. As classes de negócio, em geral não são afetadas quando é transportada para outro ambiente.

2.2.4. Camada de Sistemas de Informação Empresariais

A camada de sistemas de informação empresarial consiste em servidores de banco de dados ou de outras fontes de dados legados, como *mainframes*. Esses recursos normalmente estão localizados em uma máquina separada do servidor Java EE, e são acessados por componentes da camada de negócios (ORACLE). As seguintes tecnologias Java EE são usados para acessar a camada sistemas em Java EE:

- *The Java Database Connectivity API (JDBC)*
- *The Java Persistence API*
- *The Java EE Connector Architecture (JCA)*
- *The Java Transaction API (JTA)*

JTA é uma poderosa API responsável por gerenciar as transações de forma transparente. JCA é uma solução tecnológica que visa conectar servidores de aplicação a sistemas de informação empresarial. Enquanto JDBC é usado para conectar aplicativos ao banco de dados, JCA é uma arquitetura mais genérica para conexão com sistemas legados.

A camada de sistema é um conjunto de bibliotecas que fornecem acesso a recursos de hardware, do sistema operacional, dos dispositivos de comunicação. Essa camada provê funções específicas do ambiente de execução de uma aplicação. Normalmente, a camada de sistema recebe mensagens de todas as

outras camadas (STAA, 2000). Portanto, o fluxo de mensagens na camada de sistema pode ser bi-direcional.

2.2.5. Camada de Persistência

Essa camada encapsula um conjunto de classes para prover acesso aos dados de armazenamento permanente. Essa camada não deve ser confundida com um banco de dados. A camada é apenas um *front-end* que empacota as classes que acessa ao banco de dados, torna a aplicação independente do SGBD e de suas versões (BIGONHA, R.S; BIGONHA, M.A.S, 2015). As classes de persistência normalmente devem ser modificadas quando ocorre uma mudança no SGBD ou no sistema de arquivos.

2.2.6. Vantagens do Modelo de Camadas

As camadas contribuem para aumentar a extensibilidade, manutenabilidade e portabilidade da aplicação. Se as mensagens fluem sempre da camada de interface para a camada de negócio, então pode-se trocar a interface sem afetar as classes de negócio. Dentro de uma camada, as mensagens podem fluir livremente. Trocar uma interface não tem efeito na camada de negócios nem na camada de persistência (SCOTT, 1998). Portável, pois se um desenvolvedor trocar o banco de dados da aplicação, a mudança afeta somente a camada de persistência, mas não a camada de negócios e muito menos a camada de interface.

Um sistema orientado por objeto é formado por suas camadas. Uma camada é um conjunto de classes que estão organizadas em arquivos, que são módulos. Os módulos estão organizados em diretórios, que são chamadas de bibliotecas, que também estão organizadas em diretórios que são acervos (conjunto de bibliotecas). Os acervos estão organizados em camadas (STAA, 2000).

2.3. Java Server Faces

Java Server Faces (JSF) foi formalizado com um padrão e foi desenvolvido por meio de *Java Community Process* sob JSR-314 (*Java Specification Request*) e faz parte da Plataforma Java, *Enterprise Edition* (CORDEIRO, 2012). Com as contribuições de grupos de especialistas, as JavaServer Faces APIs estão sendo projetadas de modo que possam ser aproveitadas pelas ferramentas que irão tornar o desenvolvimento de aplicações web ainda mais fácil. Vários fornecedores de ferramentas eram membros do grupo de especialista da JSR-314, que desenvolveram a especificação JavaServer Faces 1.0. Esses fornecedores estão empenhados em apoiar a tecnologia JavaServer Faces em suas ferramentas, promovendo, assim, a adoção do padrão de tecnologia JavaServer Faces (LUCKOW e MELO, 2010). O *download* da versão mais recente da especificação e implementação de tecnologia JavaServer Faces estão disponível em Mojarra Project.

A tecnologia JavaServer Faces inclui: (i) um conjunto de APIs para representar componentes de interface do usuário, (ii) gerenciamento de estado, (iii) manipulação de eventos, (iv) validação de entrada, (v) definição da navegação das páginas e apoio à internacionalização e acessibilidade (ORACLE). Projetada para ser flexível, a tecnologia JavaServer Faces aproveita o padrão UI e o conceito de camada *Web* sem limitar os desenvolvedores em uma linguagem particular de *mark-up*, protocolo ou dispositivo cliente. As classes de componentes de interface do usuário que faz o uso da tecnologia JavaServer Faces encapsula a funcionalidade do componente, e não a apresentação específica do cliente (CORDEIRO, 2012). Permitindo, assim, JavaServer Faces componentes de interface serem compatíveis com vários dispositivos cliente.

A facilidade de uso é um dos objetivos principais do JSF. A arquitetura do JavaServer Faces define claramente uma separação entre a camada lógica da aplicação e a camada de apresentação. Assim, tornando mais fácil para integrar a camada de apresentação com o código do aplicativo (camada lógica). Logo, permite também que cada membro de uma equipe de desenvolvimento se concentre em sua parte de desenvolvimento (LUCKOW e MELO, 2010). A Tabela 2.4 apresenta um histórico das versões da JSF publicadas até a escrita deste trabalho.

Desenvolvedores de diferentes níveis de habilidade podem implementar uma aplicação web a partir da criação ou reutilização de componentes de interface do usuário (ORACLE). Também é permitido conectar esses componentes em fonte de dados e aplicar diferentes técnicas para manipular informações no banco de dados.

JavaServer Faces (JSF) é uma especificação Java para a construção de componentes, baseados em interfaces de usuário que utiliza um modelo de programação orientado a eventos.

Versão	Data	Descrição
2.2	21/05/2013	Introduziu novos conceitos como vistos apátridas, fluxo de página e a capacidade de criar contratos de recursos portáteis.
2.1	22/11/2010	*Release* de manutenção 2 de JSF 2.0. Só quantidade muito menor de mudanças de especificação.
2.0	01/07/2009	Maior liberação para facilidade de uso, funcionalidade aprimorada e desempenho. Coincide com Java EE 6.
1.2	11/05/2006	Muitas melhorias nos sistemas centrais e APIs. Coincide com Java EE 5. Adoção inicial em Java EE.
1.1	27/05/2004	Versão de correção de *bugs*. Não há mudanças de especificação.
1.0	11/03/2004	Especificação inicial liberada.

Tabela 2.4: versões da tecnologia JavaServer Faces.

2.3.1. Ciclo de Vida de uma Requisição JavaServer Faces

O ciclo de vida define como uma requisição se comporta internamente quando está processando uma determinada página. Segundo LUCKOW e MELO (2012) esse ciclo de vida foi muito bem pensado e desenhado. Em alguns casos, algumas etapas definidas podem ser puladas, mas recomenda-se deixar as requisições percorrer todas as etapas. O ciclo de vida JSF é vinculado ao do objeto *Lifecycle*, o *Faces Context* trabalha com seis fases que formam o ciclo de vida da

JSF (CORDEIRO, 2012). A Figura 2.2 apresenta um diagrama do ciclo de vida de requisições JSF.

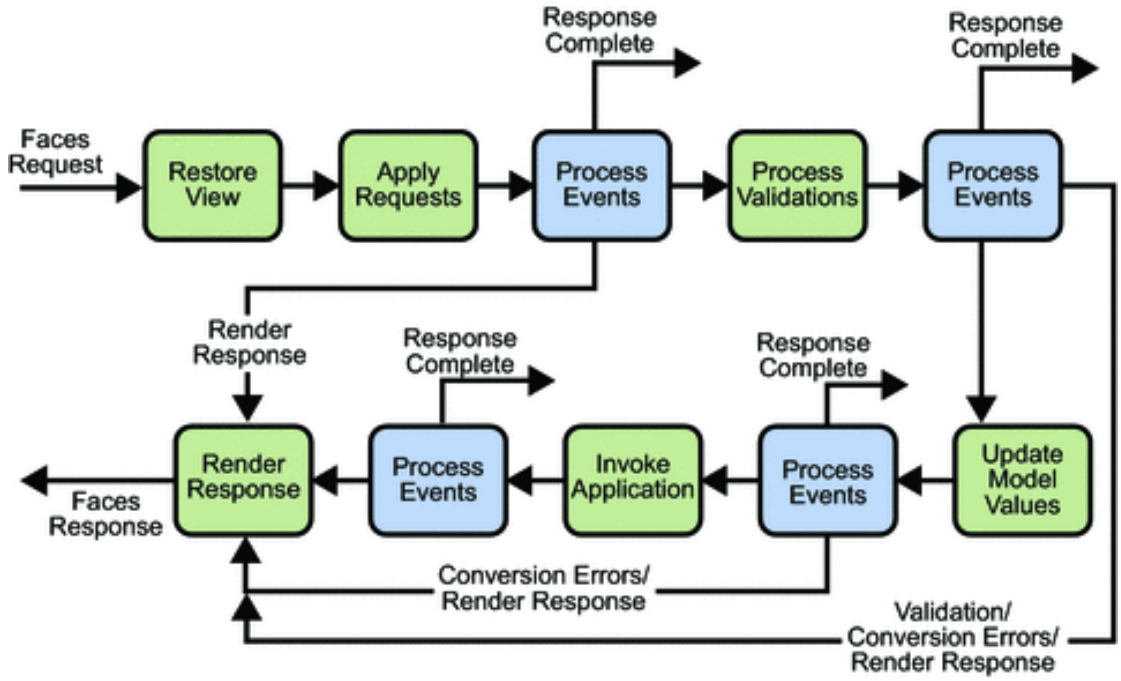


Figura 2.2: ciclo de vida de requisições JavaServer Faces

Fonte: www.devmedia.com.br

2.3.1.1. Restore View

Essa fase começa quando uma requisição é feita por meio de um link ou botão clicado. Quando uma requisição chega no servidor, o JSF extrai da URL solicitada o nome da página que deve ser exibida. Caso a requisição da página seja uma requisição inicial, o JSF fará uma leitura da página solicitada e carregará em memória todos os componentes, formando uma árvore de objetos em memória. Momento em que se cria uma instância da classe *Faces Context* para a requisição (LUCKOW e MELO, 2010).

Caso a requisição solicitada corresponda à página que já existe em memória, o JSF restaura a árvore de componentes que já foi montada anteriormente. Nessa situação, o JSF restaura a "*view*" por usar as informações do estado salvo dentro de um cliente ou servidor (CORDEIRO, 2012). Normalmente, essa situação ocorre quando um formulário for submetido ou um botão for acionado.

2.3.1.2. Apply Request Values

Nessa fase, o JSF preenche a árvore de componentes da página com os valores que foram enviados na requisição. O JSF extrai os valores do *request* e aplica esses valores nos componentes em memória e não aplica nas propriedades da classe Bean para a qual o componente aponta (COELHO, 2013).

Porém, existe um cenário na qual o valor de um componente será diretamente atribuído à propriedade da classe Bean para o qual o componente aponta. Isto é, quando um componente estiver configurado com o atributo *immediate="true"*, o valor dele será diretamente ligado à propriedade da classe Bean. Esse atributo existe na maioria dos componentes JSF e tem a característica de aplicar os valores imediatamente na classe Bean, em vez de esperar a fase Update Model Values (LUCKOW e MELO, 2010). Se ocorre algum erro de conversão, será gerada uma mensagem de erro que será adicionada na instância do *FacesContext*, para posteriormente ser apresentada para o usuário.

2.3.1.3. Process Events

Nessa fase, o JSF percorre componente por componente e executa os eventos registrados. Esses eventos podem sinalizar a necessidade de reexibir a página imediatamente. Segundo LUCKOW e MELO (2010), essa situação ocorre sempre em caso de erros de conversão de dados.

2.3.1.4. Process Validation

Durante essa fase, o JSF percorre todos os componentes e executa todos os validadores e valores registrados conforme as regras definidas na fase de desenvolvimento. Essas regras são definidas pelos desenvolvedores, uma simples regra é definir um campo com o atributo *required="true"* ou por meio de outros

validadores registrados (COELHO, 2013). Se um campo não for validado corretamente, o ciclo de vida da requisição JSF é interrompido e a página é reexibida.

2.3.1.5. Update Model Values

Nessa fase, os valores já estão validados e registrados nos componentes que serão atribuídos à respectiva propriedade na classe Bean. Nesse processo também acontece à conversão dos tipos de dados (CORDEIRO, 2012). Em Java, já existe os conversores básicos dos tipos de dados. Por exemplo, conversores de data e número, ou conversores personalizados que são criados e registrados nos componentes.

2.3.1.6. Invoke Application

Depois que todos os valores foram validados, convertidos e atribuídos às respectivas propriedades da classe Bean, pelas fases anteriores o JSF aciona, se for o caso, o método da classe Bean que acionou a requisição. Geralmente são eventos que retornam uma *string,* que está associada a uma navegação de uma página dentro da aplicação. Nessa fase, o JSF manipula qualquer nível de evento da aplicação, desde o envio de um formulário ou chamada para outra página por meio de um *link* (LUCKOW e MELO, 2010).

2.3.1.7. Render Responde

É a fase final do ciclo de vida de uma requisição JSF, a página será construída e devolvida para o *browser*. Cada componente de tela utilizado contém em sua estrutura suas propriedades, comportamentos e forma física. Assim, o JSF

solicita que cada componente de tela gere seu próprio HTML, construindo a página resultante a ser apresentada ao usuário (COELHO, 2013).

2.4. JAVA PERSISTENCE API

O Java *Persistence* API (JPA) é uma especificação de interface de programação de aplicações Java que descreve o gerenciamento de dados relacionais, tanto em aplicações Java SE ou Java EE. O Java *Persistence* API fornece também um mapeamento objeto-relacional para os desenvolvedores Java, no quesito de gerenciamento de dados relacionais em aplicativos Java (ORACLE). JPA também descreve uma interface comum para *frameworks* de persistência de dados e fornece um modelo de persistência POJO para mapeamento objeto-relacional. Java *Persistence* consiste em três áreas: (1) o Java *Persistence* API, definida no *packge javax.persistence,* (2) a linguagem de consulta, a *Java Persistence Query Language* (JPQL) e (3) objeto-relacional *mapping metadata*, o mapeamento é inteiramente dirigido a metadados (CORDEIRO, 2012).

A JPA auxilia a padronizar a interação da aplicação com o banco de dados. Define um meio de mapeamento objeto-relacional para objetos Java simples. Os frameworks Hibernate, OpenXava, Defrost e Oracle Toplink implementam a especificação da JPA (LUCKOW e MELO, 2010).

O Java Persistence API foi desenvolvido pelo grupo de peritos software EJB 3.0 como parte do JSR 220, mas seu uso não é limitado aos componentes de software EJB. Ele também pode ser utilizado diretamente pelos aplicativos da Web e aplicativos clientes, e até mesmo fora da plataforma Java EE, por exemplo, em aplicações Java SE (ORACLE).

JPA é um *framework* baseado em POJOS (*Plain Old Java Objects*) para persistir objetos Java, qualquer objeto com um construtor *default* pode ser feito persistente sem alterar uma linha de código. A especificação JPA trata de entidades, mapeamentos, interfaces para gerenciar a persistência e linguagem de consulta.

JPA não é apenas um framework para Mapeamento Objeto-Relacional (ORM - *Object-Relational Mapping*), JPA também oferece diversas funcionalidades essenciais para aplicação corporativa (CORDEIRO, 2012). JPA também provê

diversas funcionalidades para os programadores, como será mais detalhado nas próximas seções. Na seção seguinte será apresentado o histórico de versões da JPA, até a escrita deste trabalho.

2.4.1. Histórico de Versões

A data de lançamento final da especificação JPA 1.0 foi 11 de maio de 2006 como parte do Java *Community Process* JSR 220. A especificação JPA 2.0 foi lançada em 10 de dezembro de 2009. A especificação JPA 2.1 foi lançado 22 de Abril de 2013. JPA 2.1 foi iniciada o em julho de 2011 com JSR 338, e foi aprovado em 22 de maio de 2013 (COELHO, 2013). Os principais recursos incluídos foram: (1) conversores, permitindo conversões de código personalizado entre banco de dados e objetos, (2) critérios em *Update* e *Delete*, que permitem atualizações e exclusões em massa via API. (3) *stored procedures*, que permitem que as consultas sejam definidas para os procedimentos de banco de dados armazenado, (4) geração de esquema, (5) entidade *graphs*, que permitem buscar ou fusão de objetos parcial ou especificado e (6) JPQL, critérios de melhorias para sub-consultas aritméticas, funções genéricas de banco de dados, cláusula de *join* ON, opção TREAT. A Tabela 2.5 apresenta as versões publicadas até a data da escrita deste trabalho.

Versões	Data de publicação
JPA 1.0	maio de 2006
JPA 2.0	dezembro de 2009
JPA 2.1	abril de 2013

Tabela 2.5: histórico de versões JPA

2.4.2. Entidades

Tipicamente, uma entidade representa uma tabela em um banco de dados relacional, e cada instância da entidade corresponde a uma linha na tabela do banco

de dados. O estado persistente de uma entidade é representada por meio de campos persistentes ou propriedades persistentes (CORDEIRO, 2012). Os campos ou propriedades usam anotações objeto-relacional para mapear as entidades e relacionamentos entre entidades de dados relacionais.

2.4.3. Requisitos para Classes de Entidade

Uma classe de entidade deve seguir os seguintes requisitos: (1) a classe deve ser definida com a anotação *javax.persistence.Entity*, (2) a classe deve ser pública ou protegida, construtor sem argumento, a classe também pode ter outros construtores, (3) a classe os métodos ou até mesmos as variáveis de instância persistentes não devem ser declaradas como final, (4) se uma instância de entidade for passada por valor como um objeto individual, pro meio de uma *session bean's*, a classe deve implementar a interface *Serializable*, (5) as classes entidades podem estender as classes entidade e não-entidade, e as classes não-entidade podem estender classes de entidade. (6) as variáveis de instância persistentes devem ser declaradas como privada, protegida ou com pacote-privado, e só pode ser acessada diretamente via métodos da classe de entidade (ORACLE).

2.4.4. Campos e propriedades persistentes em classes de entidade

O estado persistente de uma entidade pode ser acessado por meio de qualquer variável de instância da entidade ou via de propriedades *JavaBeans*. Os campos ou propriedades devem ser dos seguintes tipos da linguagem Java:

- *Java primitive types*
- java.lang.String
- *Other serializable types including:*
- *Wrappers of Java primitive types*
  - java.math.BigInteger
  - java.math.BigDecimal
  - java.util.Date
  - java.util.Calendar

    - java.sql.Date
    - java.sql.Time
    - java.sql.TimeStamp
    - User-defined serializable types
    - byte[]
    - Byte[]
    - char[]
    - Character[]
- *Enumerated types*
- *Other entities and/or collections of entities*
- *Embeddable classes*

Entidades podem utilizar campos persistentes ou propriedades persistentes. Se as anotações forem aplicadas nas variáveis de instância da entidade, a entidade usa campos persistentes. Se as anotações forem aplicadas nos métodos getter da entidade, a entidade usa propriedades persistentes. Segundo LUCKOW e MELO (2012), você não pode aplicar anotações de mapeamento para campos e propriedades numa única entidade.

2.5. Conceito de ORM

ORM (*Object Relational Mapping*) é uma técnica de mapeamento objeto relacional que visa criar uma camada de mapeamento entre o modelo de objetos (aplicação) e o modelo relacional (banco de dados) de forma a abstrair o acesso ao mesmo (ELMASRI e NAVATHE, 2005). ORM, resumidamente pode ser entendido como uma forma automatizada e transparente de persistir objetos em tabelas de um banco de dados relacional. É uma técnica que visa minimizar as diferenças entre o modelo de dados relacional e o modelo orientado por objeto.

Uma solução ORM contém as seguintes características: (1) uma API para realizar as operações (*create, read, update and delete*) em objetos de classes persistentes, (2) uma linguagem ou API para especificar consultas que se referem às classes ou propriedades das classes, (3) facilidade de especificar o metadado de mapeamento e (4) uma técnica para que a implementação ORM interaja com objetos transacionais (DATE, 2003).

Ao longo do tempo, foram criados diversos *frameworks* ORM, que auxiliam nas tarefas de persistência e recuperação de dados. Assim, possibilitar mais

produtividade para os desenvolvedores em suas tarefas diárias, além de nivelar e padronizar a camada de acesso a dados do sistema. Embora alguns programadores optam por criar suas próprias ferramentas ORM. Os principais ORMs que trabalham com o .NET são o Entity Framework e NHibernate. Para a linguagem Java existe o Hibernate, TopLink da Oracle e OpenJPA, da Apache. Independente do framework, para aplicação Java EE todos tendem a seguir a especificação JPA (LUCKOW e MELO, 2010). A Figura 2.3, ilustra uma camada de mapeamento, chamada ORM.

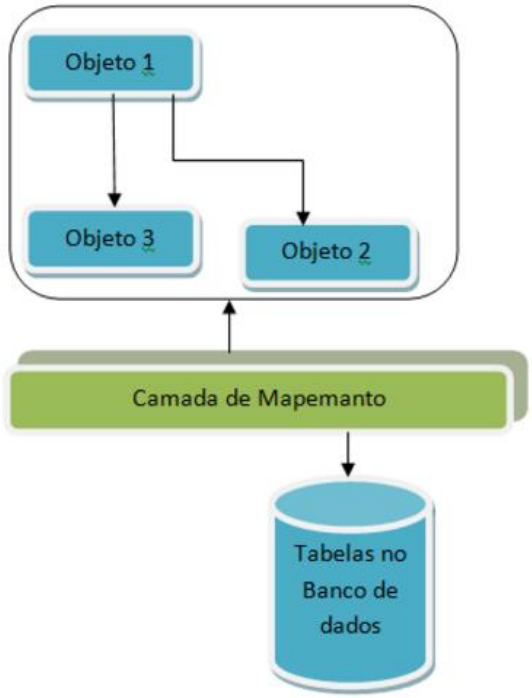


Figura 2.3: ilustração do mapeamento

Os *frameworks* se localizam em uma camada intermediária entre a lógica da sua aplicação e o SGDB. O *framework* passa a receber as solicitações de interação com o SGDB via os objetos de sua aplicação (CORDEIRO, 2012). Após isso, o *framework* gera de forma automatizada todo o SQL necessário para a operação solicitada. Assim, evitando trabalho de escrita e manutenção destes SQLs. Além disso, os *frameworks* normalmente tratam as variações de tipos de dados existentes entre a aplicação e o SGDB. Como mostra na Figura 4.2, a aplicação interage com o *framework* e não com a base de dados diretamente, desacoplando a aplicação de SGDBs específicos. Com uso de *frameworks* ORM, a produtividade aumenta devido

ao fato de não precisar escrever códigos SQL para inserir, alterar, excluir e recuperar dados do banco de dados (BITTENCOURT, 2005).

2.6. Hibernate

Hibernate é uma ferramenta para realizar o mapeamento de objeto/relacional que contém todos os benefícios da tecnologia ORM. É uma solução ORM existente para a linguagem de programação Java. O Hibernate contém as características descritas na Seção 4.4, que caracteriza como uma aplicação ORM (LUCKOW e MELO, 2010).

O mapeamento objeto/relacional (ORM) refere-se à técnica de mapeamento de dados, utilizando um esquema baseado na SQL (PUGH e GRADECKI, 2004). O Hibernate não fica somente responsável pelo mapeamento das classes de um sistema, ele também proporciona facilidades na recuperação e consulta de dados, o que pode reduzir significativamente o tempo de desenvolvimento (BAUER e KING, 2007). Hibernate é um projeto de Fonte Aberto Profissional e componente crítico do Sistema Jboss (KONDA, 2014). O Hibernate tem três frentes de software para lidar com a especificação JPA:

(1) Hibernate Core, é a base de todo conjunto de soluções para persistência, contendo uma API nativa e metadados de mapeamento guardados em arquivos XML. Com uma linguagem própria, denominada HQL, do estilo da SQL. Tem também a interface Criteria, para realizar consultas.

(2) Hibernate Annotations, uma forma de fazer o mapeamento de objetos relacional utilizando *annotations,* um tipo de tags especiais*.* A vantagem de utilizar *annotations* para realizar o mapeamento é a redução de linhas de código em relação a fazer o mapeamento via arquivo XML (LUCKOW e MELO, 2010)*.*

(3) Hibernate EntityManager é uma camada que trata do conceito de programação de interfaces e funcionalidades de consulta. A interface lê o metadado ORM de uma entidade e realiza operações de persistência. A Figura 2.4 apresenta uma arquitetura simplificada do Hibernete.

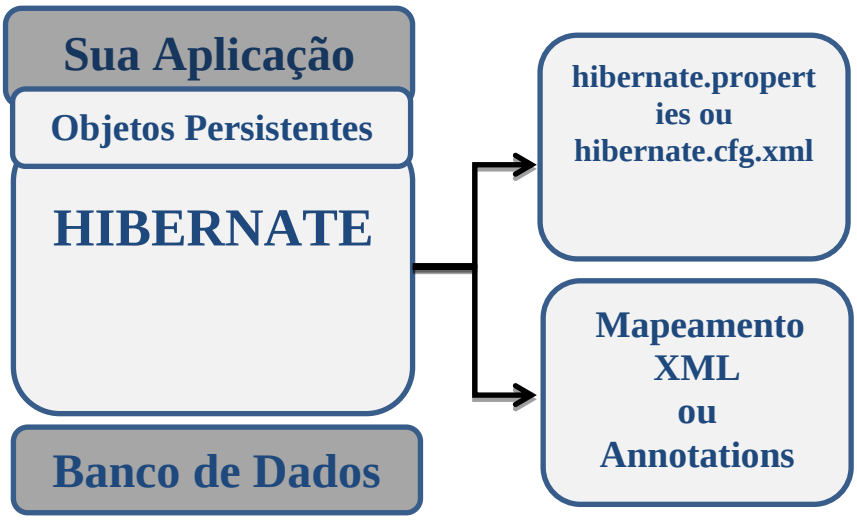


Figura 2.4 – Arquitetura simplificada do Hibernate

Resumidamente, o Hibernate tem foco na persistência de dados que se aplica aos bancos de dados relacionais. Além de sua própria API nativa, o Hibernate também é uma implementação da especificação JPA. É usado em qualquer ambiente, incluindo aplicativos Java SE e servidores de aplicativos Java EE (CORDEIRO, 2012).

O Hibernate também permite que desenvolvedores desenvolvam classes persistentes seguindo o conceito de orientação por objetos. Incluindo o conceito de herança, polimorfismo, associação, composição e as estruturas de coleções existente em Java. O Hibernate permite também que qualquer classe ou estrutura de dados seja persistente.

Algumas das vantagens de utilizar o Hibernate são a: (1) alto desempenho, suporta inúmeras estratégias de busca, não necessita de tabelas ou campos especiais para banco de dados. Ele gera a maior parte do SQL em tempo de inicialização do sistema, em vez de em tempo de execução. Hibernate oferece desempenho superior sobre o código JDBC, tanto em termos de produtividade do desenvolvedor e desempenho de tempo de execução. (2) escalabilidade, foi projetado para trabalhar em *cluster* de servidor de aplicativos e oferecer uma arquitetura altamente escalável. (3) confiabilidade, Hibernate é bem conhecido pela sua excelente estabilidade e qualidade, comprovado pela aceitação e utilização por dezenas de milhares de desenvolvedores Java. (4) Extensibilidade, Hibernate é altamente configurável e extensível.

2.7. Spring Framework e Spring Security

O Spring Framework é um *framework* de código aberto para a plataforma Java criado por Rod Johnson, que visa facilitar o desenvolvimento JAVA EE. Esse *framework* é baseado nos padrões de Inversão de Controle (IoC) e Injeção de Dependências (MACHACEK et al, 2008). Sua característica dentro de um projeto Java é a instanciação de classes, aplicando as dependências entre as classes com base com base em definições criadas pelo desenvolvedor em um arquivo XML. Algumas vantagens de utilizar o Spring é que ele permite um baixo acoplamento entre classes, facilita testes unitários, alto desempenho da aplicação, ainda contempla mecanismos de segurança e controle de transações (LUCKOW e MELO, 2010).

O Spring Security surgiu da necessidade de melhorar o suporte à segurança oferecido pela especificação Java EE. O *framework* centraliza a configuração em um único arquivo XML, dispensando configurações do *container* e tornando a aplicação web um arquivo WAR auto contido.

O *framework* Spring também oferece diversos módulos que podem ser utilizados de acordo com as necessidades de cada projeto. No Spring existem módulos voltados para desenvolvimento *Web*, persistência de dados, acesso remoto e programação orientada a aspectos (POA) (JOHNSON, 2002). O Spring Security é um dos projetos Spring mais consolidado. Sua primeira versão foi lançada em 2003, o *Spring* é utilizados em milhares de projetos pelo mundo, incluindo agências governamentais e militares (LUCKOW e MELO, 2010). O Spring Security tem um mecanismo de autenticação e controle de acesso para aplicações *web*. De forma fácil e personalizável, o *Spring Security* supera vários aspectos de segurança tradicional do Java EE.

O Spring Security é útil para aplicações *web* que necessitam restringir seus recursos para diferentes tipos de usuários. O *Spring Security* assegura o processo de autenticação de forma prática e segura (Spring Security).

Para utilizar o Spring Security basta adicionar seus arquivos JARs ao *classpath*, configurar um filtro e um *listener* no arquivo *web.xml* e criar um XML de configuração. O arquivo XML armazena todas as configurações de autenticação e autorização. As *tags* **<intercept-url>** definem papéis (*roles*) podem acessar cada

grupo de URLs. A *tag* **<authentication-provider>** define a fonte de dados para as informações de usuários (banco de dados, arquivo de propriedades, etc). O Spring *Security* depende de alguns JARs do *Spring Framework* “core”. Mas, não é necessário que uma aplicação seja construída com o modelo de programação do *Spring Framework* (LUCKOW e MELO, 2010). Ou seja, uma aplicação existente que não usa Spring pode passar a utilizar o *Spring Security* sem grandes modificações.

O *Spring Security* possui uma abordagem declarativa para segurança, baseada em papéis. A abordagem é declarativa, pois a aplicação não precisa chamar método para realizar a autenticação, tudo é feito via configuração no arquivo XML.

2.7.1. Segurança de pastas

Um sistema possui diferentes tipos de arquivos e esses arquivos estão localizados em diferentes pastas, que forma a estrutura do projeto. A construção das páginas relativas ao usuário podem ser criadas e distribuídas em locais diferentes. Separar os recursos do sistema em diferentes pastas e após configurar o *Spring Security* para essas pastas pode garantir a segurança do acesso a determinadas pastas.

2.7.2. Controle da página Login

Sempre que um usuário tentar acessar alguma pasta do sistema que tenha a segurança garantida pelo *Spring Security,* será automaticamente direcionado para a página de *login* para o usuário identificar suas credenciais. Caso o usuário informe um login e senha válidos, ele será direcionado para a página solicitada inicialmente.

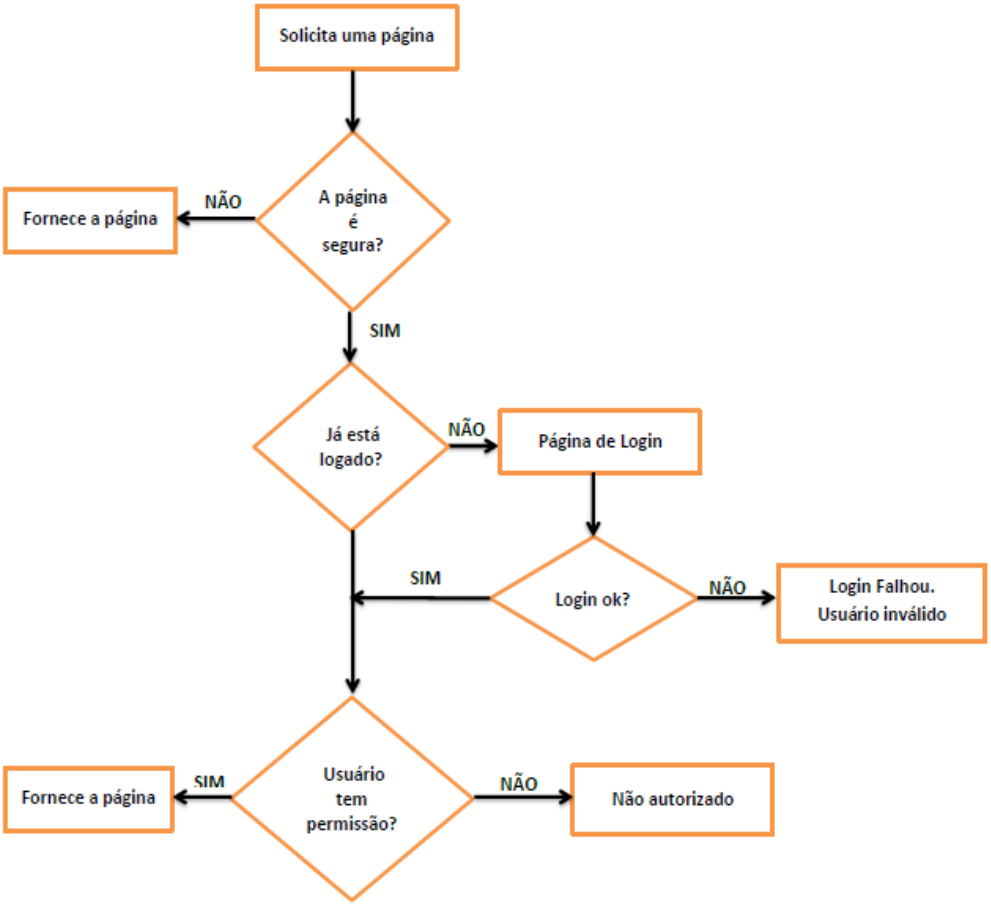


Figura 2.5 – Funcionamento geral do Spring *Security*

Caso contrário, será exibida uma página de erro ou até mesmo na página de *login* será exibido uma mensagem que o *login* ou senha não foram validados. O funcionamento geral do Spring Security pode ser visto na Figura 2.5.

Nos arquivos de configuração do Spring Security o desenvolvedor vai apenas informar qual é a página de *login* ou autenticação de usuário. O desenvolvedor também informa no arquivo de configuração do Spring, qual página deve ser exibida depois do *login* e qual página será exibida após o *logout*. Logo, o desenvolvedor registra todas as regras de segurança do sistema em um arquivo de configuração do Spring.

2.8. Relatórios com iReport e Jasper Reports

As ferramentas iReport e *Jasper Reports* são extremamente úteis para os desenvolvedores e programadores Java durante o processo de desenvolvimento de software.

O *framework Jasper Reports* permite gerar diversos tipos de relatórios em vários formatos, como PDF, HTML, XML, XLS dentre outros (HEFFELFINGER, 2006). *Jasper Reports* é uma biblioteca versátil Java responsável pela execução dos relatórios, permite o uso de diagramas, gráficos, e até códigos de barras. O *Jasper Reports* também aceita diferentes tipos de entrada de dados, como um arquivo XML, CSV, banco de dados, uma sessão do Hibernate (SOUZA, 2013). A licença dessa ferramenta é baseada no modelo de Licença Pública Geral Menor (LGPL), que é uma ferramenta totalmente *open source* e gratuita (LUCKOW e MELO, 2010).

A ferramenta iReport permite definir o projeto de um relatório dentro de um ambiente gráfico. O ambiente do iReport oferece vários recursos para construir relatório de forma consistente e bem elaborada (SOUZA, 2013). Resumidamente, a função do iReport é apenas disponibilizar um ambiente para montar o relatório. A Figura 2.6 apresenta em alto nível como seria a execução de um relatório em JasperReports.

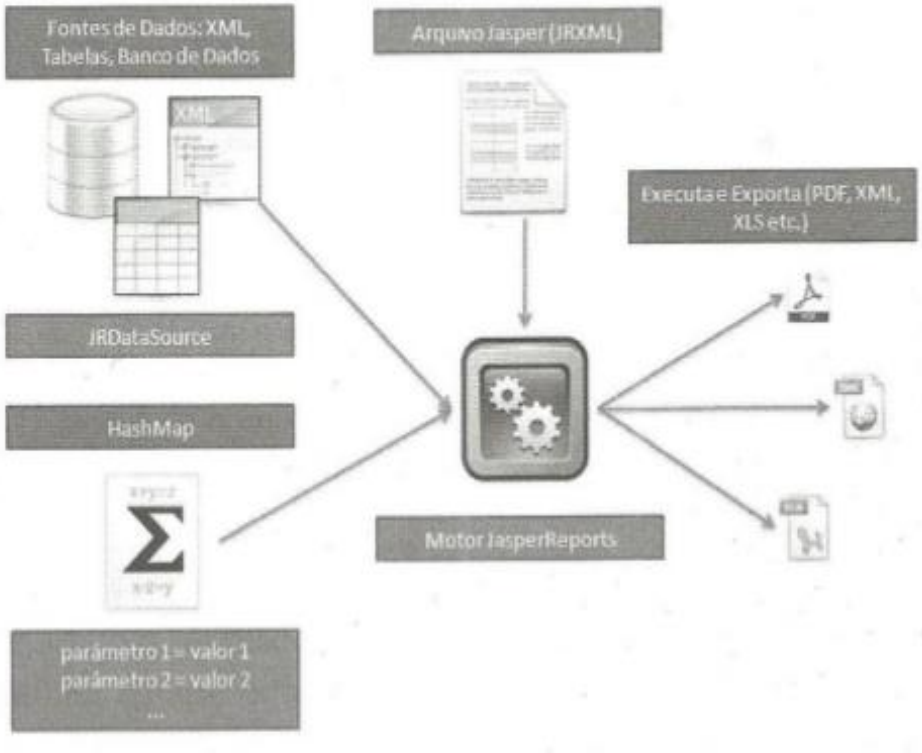


Figura 2.6 – Fluxo de execução de um relatório.

Fonte: Luckow; Melo (2010, p.495).

2.9. Conclusão

Java EE é uma plataforma que contém um conjunto de tecnologias que reduz significativamente o custo e a complexidade do desenvolvimento, implantação e gerenciamento de aplicações. Java EE oferece um conjunto de APIs para desenvolvimento e execução de aplicações robustas, escaláveis e seguras do lado do servidor.

O Desenvolvimento de software em camadas é uma das técnicas mais utilizadas em arquitetura cliente/servidor. Segundo Martin Fowler, alguns dos benefícios de decompor um software em camadas são: (1) capacidade de compreender uma camada coerentemente como um todo, sem muito conhecimento das demais camadas, (2) a dependência de camada é reduzida, (3) padronização e (4) utilização por outros serviços de nível mais alto. Cada camada tem sua responsabilidade bem definida.

A tecnologia Java *Server Faces* estabelece o padrão para a construção de interfaces com usuário do lado do servidor. JSF reduz o esforço na criação e manutenção de aplicativos que são executados em um servidor de aplicativos Java. Fornece componentes de interface reutilizáveis, simplifica a conexão da interface do usuário com fontes de dados. Alguns dos recursos disponibilizados pela tecnologia são: validação de dados, conversão de dados, composição e reutilização de componentes.

JPA é um *framework* que permite aos desenvolvedores e programadores armazenar e gerenciar os dados utilizando o mapeamento (ORM) de objetos em banco de dados relacional. A tecnologia permite persistir objetos em banco de dados relacional.

Hibernate é um framework de mapeamento objeto relacional. Sua principal característica é mapear classes Java para tabela de banco de dados relacional. Ele também automatiza processos repetitivos para manipular dados que alivia o desenvolvedor de criar extensas SQLs.

3. Metodologia

Este capítulo apresenta os procedimentos metodológicos utilizados para alcançar os objetivos deste trabalho, mostrando as ferramentas e recursos tecnológicos utilizados para o seu desenvolvimento.

3.1. Ambiente de desenvolvimento

Definir quais ferramentas a ser utilizadas em um projeto de software envolve vários aspectos. O critério que foi utilizado para escolher as ferramentas para o desenvolvimento da aplicação financeira pessoal foi a própria popularidade dessas ferramentas. O ambiente de desenvolvimento da aplicação financeira é composto inicialmente pela linguagem de programação Java, Apache Tomcat, Eclipse e MySQL. Essas ferramentas possuem comunidades altamente atuantes, que constantemente contribuem para sua evolução.

O arquivo para instalação do Java pode ser encontrado em http://www.oracle.com/technetwork/java/javase/downloads/index.html. Os principais pacotes são JRE (Java Runtime Environment), que é o pacote mínimo necessário para a execução de aplicativos Java. Pacote JDK (Java Development kit), que é necessário para o desenvolvimento de aplicativos Java. Para o desenvolvimento deste trabalho foi utilizado a versão Java 7.

O Apache Tomcat é um *contêiner* Java e um servidor *web* ao mesmo tempo. Tomcat suporta a execução das tecnologias Java Servlet e JavaServer Pages (JSP), o que permite que Java funcione para um ambiente *web*. Apache Tomcat é robusto e permite integração com outros servidores como: Apache HTTP e IIS da Microsoft. Para o desenvolvimento deste trabalho foi utilizado a versão 7 do Apache Tomcat. O pacote de instalação pode ser obtido do *site* http://tomcat.apache.org/.

A ferramenta Eclipse foi inicialmente desenvolvida pela IBM. Atualmente, Eclipse é uma plataforma de desenvolvimento de código aberto e não apenas uma IDE Java. Empresas e comunidade têm usado a plataforma para criar *plug-ins* para diversas linguagens de programação. O pacote de instalação pode ser obtido pelo

site https://eclipse.org/. Para o desenvolvimento desse trabalho foi utilizado a versão Eclipse Mars.

O MySQL é um banco de dados de código aberto mais popular do mundo, tendo mais de 70 milhões de instalações. É utilizado por grandes empresa como Amazon.com, Google, Motorola, MP3.com, NASA e Yahoo! *Finance*. O arquivo de instalação pode ser obtido pelo *site* https://www.mysql.com/downloads/.

Para deixar a aplicação financeira segura foi utilizado o *framework* de segurança *Spring Security,* que garante a autenticação e a autorização dos usuários. Os usuários só têm acesso ao conteúdo para o qual receba permissão. O *Spring Framework* é um *framework* Java de código-aberto que facilita o desenvolvimento de aplicação Java EE. Sua funcionalidade básica é a instanciação de classes, realizando a injeção de dependências conforme a definição realizada pelo desenvolvedor em um arquivo XML. O *Spring Security* está disponível para *download* no *site* http://projects.spring.io/spring-security/.

O Facelets é um *framework* de *templates* que ajuda a diminuir o volume de código fonte em páginas *web*. Esse *framework* também padroniza e centraliza as definições visuais e estruturais das páginas de um sistema *web*.

3.2. Técnicas utilizadas para desenvolver a aplicação

Para a aplicação financeira pessoal foi utilizado Java *Server Faces* 2.0 que é uma especificação para um *framework* de componentes para desenvolvimento *web* em Java. Grandes empresas como Apache, IBM, Macromedia, Novell, Oracle, Siemens e Sun participaram na definição do JSF. Por esse motivo e outros, o JSF tornou um padrão de mercado. A implementação utilizada na aplicação financeira pessoal foi da Sun Mojarra, que de fato é a implementação mais conhecida do JSF. Seu pacote está disponível no *site* oficial do Mojara: http://javaserverfaces.java.net/. Essa implementação disponibiliza todos os recursos padrão do JSF, como componentes HTML de formulário, tabelas, layout, conversão e validação de dados.

A biblioteca de componentes utilizada na aplicação financeira pessoal foi a *Prime Faces*. Com mais de 90 componentes, essa biblioteca sempre está acompanhando

a evolução do JSF. Essa biblioteca pode ser obtida pelo *site* http://www.primefaces.org/.

O processo de instalação do JSF é bem simples, basta obter os arquivos do JSF, implementação Mojarra, arquivo JSTL e arquivos do projeto Apache *Commons*. Os arquivos do JSF são: jsf-api.jar e jsf-impl.jar. O JSTL é um conjunto de *tags* de apoio para o desenvolvimento *web* em Java. O arquivo JSTL pode ser obtido no *site* https://jstl.java.net/download.html. Os arquivos do projeto Apache *Commons* são necessários para desenvolver a aplicação financeira pessoal, mas não obrigatórios para o JSF. Esses arquivos são: *commons-beanutils-1.7.jar*, utilitário para acessar as propriedade dos componentes JavaBeans; *commons-collections-3.2.jar*, extensão do Java 2 SDK *Collections Framework; commons-digester-2.1.jar,* utilizado para realizar o processamento de arquivos XML; commons-logging-1.1.1.jar, para a geração de mensagens de log. Esses arquivos podem ser obtidos no *site* http://commons.apache.org/.

Todos esses arquivos citados são copiados para a pasta WEB-INF/lib do projeto da aplicação financeira pessoal. Existem quatro passos básicos para iniciar um desenvolvimento usando JSF: (1) criar a classe Bean; (2) realizar o mapeamento da classe Bean; (3) criar a página JSF e (4) realizar o mapeamento da navegação entre as páginas. Esses passos foram utilizados frequentemente no desenvolvimento da aplicação financeira pessoal. A classe Bean é uma classe Java normal.

Para as páginas ter acesso às propriedades e operações da classe Bean é necessário fazer o mapeamento da classe. Esse mapeamento pode ser feito de duas formas: (1) via arquivo de configuração, face-config.xml, ou (2) usando *annotations*, que é muito mais prático. Na aplicação financeira pessoal foi utilizado o mapeamento via *annotations*.

Para armazenar os dados da aplicação financeira pessoal é utilizado o banco de dados MySQL, que de fato é um banco de dados relacional. A tecnologia de banco de dados relacional, atualmente está bem fundamentada, seja por causa da fundamentação teórica ou pela flexibilidade quanto ao gerenciamento dos dados. Em uma aplicação Java existem duas formas para comunicar com o banco de dados: (1) via JDBC, que é uma especificação de como a linguagem Java faz a comunicação com o banco de dados ou (2) via frameworks ORM, como o Hibernate.

Na aplicação financeira pessoal foi utilizado o *framework* Hibernate para persistir os dados. Esse *framework* faz a persistência de forma automatizada e transparente.

Esse *framework* está disponível no *site* http://www.hibernate.org. O Hibernate também possibilita realizar o mapeamento de objetos via arquivo XML ou via *annotations.* Na aplicação financeira pessoal desenvolvida foi utilizado o mapeamento via *annotations*. Essa técnica escolhida se baseia no uso de *tags* estilo JavaDoc dentro do código-fonte, em vez de arquivos XML.

As opções de consultar dados com Hibernate são: (1) uso da classe *Query*, que é a maneira mais direta de fazer consulta, (2) uso da classe Criteria, que permite construir consultas de forma dinâmica e (3) consultas nomeadas que permite a reutilização de consultas. Na aplicação financeira pessoal desenvolvida foi utilizada a classe Criteria e a classe *Query* para realizar as consultas.

3.3. Conclusão

Java é uma tecnologia usada para desenvolver diferentes tipos de aplicações como: *desktop, web, mobile*, jogos dentre outras. É uma plataforma que estar em constante evolução.

O Eclipse é um ambiente de desenvolvimento integrador (IDE) sua principal utilização é para desenvolvimento de aplicativos Java, mas pode ser usado para desenvolver aplicações em outras linguagens de programação.

O tomcat é um servidor *web* Java de código aberto que é distribuído como software livre. Ele provê um servidor *web* HTTP em Java e cobre parte da especificação do Java EE.

MySql é um banco de dados *open source* mais popular do mundo para uso em aplicações web. Funciona em muitas plataformas de sistemas operacionais como Windows e Linux.

Spring Security é uma estrutura que se concentra em fornecer autenticação e autorização para aplicativos Java.

Facelets é um *framework* de *templates* que suporta todos os componentes JSF. Fornece suporte para reúso, permite incluir em uma determinada página *web* um conteúdo de outro arquivo HTML.

PrimeFaces é a biblioteca de componentes que pode ser incluída em aplicações JSF. PrimeFaces inclui componentes que proporcionam maior funcionalidade em comparação com a biblioteca de componentes JSF padrão.

4. Desenvolvimento da Aplicação Financeira

Esse capítulo apresenta a descrição da aplicação financeira pessoal e de como os recursos foram utilizados em cada etapa do desenvolvimento para gerar uma solução completa.

4.1. Arquitetura definida

A arquitetura da aplicação foi preparada de forma a integrar de forma clara e eficiente os recursos tecnológicos do JSF, Hibernate dentre outros. A arquitetura da aplicação é baseada no conceito de separação de responsabilidades e arquitetura MVC.

O sistema foi dividido em camadas para organizar o desenvolvimento e cada camada tem sua responsabilidade bem-definida na arquitetura do projeto.

As responsabilidades definidas foram: (1) gravação de informações em banco de dados, (2) tomar de decisões e (3) gerar páginas para exibição de dados.

As camadas definidas foram: (1) acesso a dados, (2) regras de negócio e (3) apresentação. Essas camadas são classes Java que tem seus objetivos bem-definidos.

A camada de acesso a dados contém classes especialista em acesso aos dados, providos do banco de dados MySQL. Essas classes de acesso a dados seguem o padrão de projeto (*Design Pattern)* DAO (*Data Access Object*). Isso significa que essas classes são exclusivas para realizar operações com o banco de dados. É adotado um padrão de nomenclatura para essas classes como, UsuarioDAO, ContaDAO e LancamentoDAO.

A camada de regra de negócio é responsável pelas tomadas de decisão. Essa camada decide quais operações de banco de dados são necessárias e não como fazê-las. As classes dessa camada possui a nomenclatura da seguinte forma, UsuarioRN, ContaRN e LancamentoRN.

A camada de apresentação é representada pelas telas do sistema, sua responsabilidade é exibir e coletar informações do usuário. As classes dessa

camada possui a nomenclatura da seguinte forma: UsuarioBean, ContaBean e LancamentoBean. A Figura 4.1 apresenta de forma ilustrativa a arquitetura em camadas.

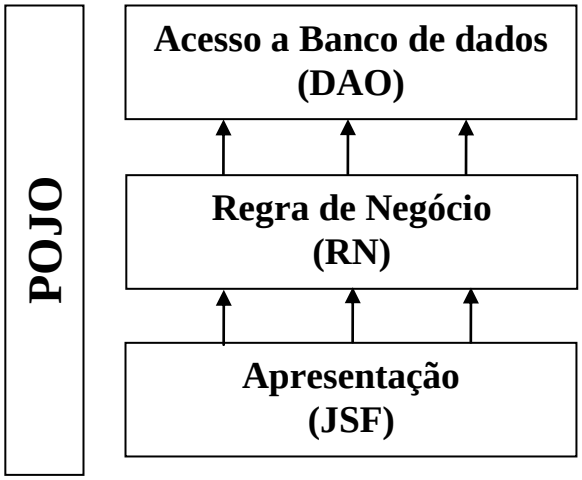


Figura 4.1 - Representação da arquitetura em camadas

4.2. Detalhes de Desenvolvimento

O projeto é composto de algumas classes que sevem de apoio para o desenvolvimento da aplicação financeira. Essas classes fornecem a infraestrutura para o funcionamento do sistema.

Foi definida a classe HibernateUtil para prover a conexão ao Hibernate. Foi também criado três classes de exceções-padrão para a arquitetura do projeto. As classes, DAOException, UtilException e RNException encapsula a exceção original lançada. As três classes implementam *java.lang.Exception.*

A classe DAOFactory tem como objetivo único construir DAOs. Essa classe é o único ponto do sistema em que as classes DAO são instanciadas. Existe um método correspondente para instanciar cada classe ContaDAO, UsuarioDAO, LancamentoDAO e CategoriaDAO.

Todas as páginas da aplicação foram desenvolvidas utilizando XHTML. Visto que diversas melhorias na estrutura do HTML foram realizadas na criação do XHTML. Logo, a criação do XHTML gerou um HTML “reformado” e estruturalmente mais robusto. Para os navegadores atuais é mais fácil e rápido interpretar um XHTML do

que um HTML. A Figura 4.2 representa a página de cadastro de usuário construída como arquivo XHTML.

A página de cadastrar usuário possui o elemento <h:messagens> em seu código fonte, esse elemento é utilizado para exibir mensagem de erro apresentado no formulário. Observe nessa figura a mensagem, “você não tem nome?” exibida acima do nome após a submissão do formulário. Todos os campos desse formulário de cadastro são necessários, ou seja, todos os campos devem ser preenchidos pelo usuário. O JSF oferece recursos de validação e conversão de dados. O desenvolvedor também pode criar seus próprios validadores. O conversor utilizado nesta página está localizado no campo “data de nascimento”, a conversão é feita para o objeto do tipo java.util.Date.

Na área administrativa do sistema é possível o usuário administrador visualizar todos os usuários cadastrados no sistema. O administrador também pode fazer exclusão, edição, ativação e desativação de usuários. O objetivo da página administrativa é permitir uma visão geral de todos os usuários. A Figura 4.3 apresenta a página administrativa da aplicação financeira pessoal. Essa página utiliza recurso de imagens com endereço dinâmico, tabelas e operações nas linhas das tabelas.

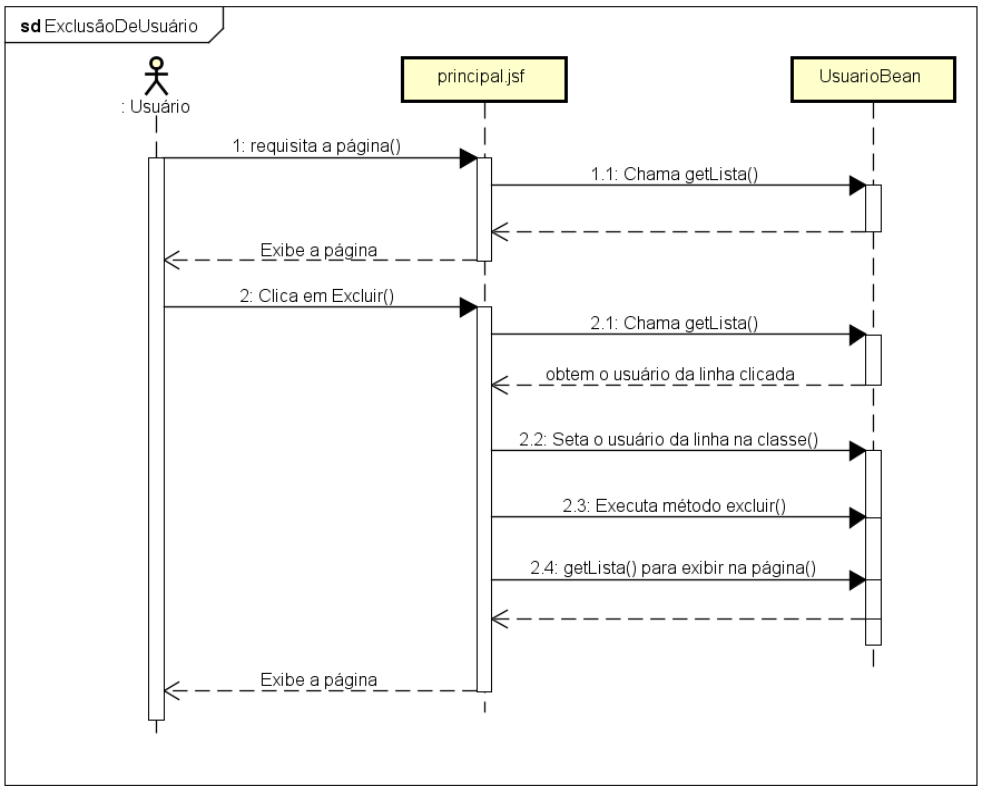
A tabela que apresenta a listagem de usuário é um componente padrão do JSF. Por meio da *tag* <h:dataTable/> é possível adicionar a tabela de maneira rápida e organizada. Toda a geração da estrutura da tabela está encapsulada nessa *tag*. Existem vários recursos presentes nessa tabela, o *emoticon* na primeira coluna indica se o usuário está ativo ou inativo, recurso de imagens intercambiáveis. Uso de botões (permissões, editar e excluir) e links nas linhas.

A Figura 4.4 abaixo apresenta o diagrama de sequência para exclusão de usuário. Vale lembrar que no Apêndice A podem ser encontrados mais diagramas de sequência como também outros diagramas da aplicaçã**o.**

Para o cadastro de contas utilizam-se os recursos Ajax do JSF, isso permite que a manutenção e a listagem dos registros fiquem na mesma página. O Ajax evita o carregamento total de uma determinada página durante sua utilização. A simples presença da *tag* <f:ajax/> possibilita o uso do Ajax nas páginas desenvolvidas. Essa *tag* sempre é adicionada dentro de outro componente, como: botões, campos e formulários. A Figura 4.5 apresenta o visual da página conta para um usuário administrador. Para um usuário normal, essa página só muda a barra de menu, em especial o ícone de acesso a área administrativa.

Figura 4.2 - Página cadastro de usuário

Figura 4.3 - Página administrativa



A Figura 4.4 - diagrama de sequência para excluir usuário

A Figura 4.5 – visual da página conta para um usuário administrador

Figura 4.6. - Página de categorias

A aplicação disponibiliza para o usuário um cadastro de categoria. Resumidamente, a categoria classifica os lançamentos financeiros realizados pelo usuário. As principais categorias são despesa e receita, essas categorias são inicializadas pelo sistema. As categorias são organizadas em um componente *Tree* do PrimeFaces que representa uma árvore. A página de categoria pode ser vista na Figura 4.6.

As demais páginas da aplicação financeira pessoal estão localizadas no Apêndice A deste trabalho. Existem também nesse apêndice alguns diagramas da UML, como diagrama de caso de uso, diagrama de sequência, entidade e relacionamento e diagrama de componente.

4.3. Conclusão

A estrutura da aplicação financeira foi definida com base no conceito de separação de responsabilidade e da arquitetura MVC. O sistema foi dividido em camadas para organizar o processo de desenvolvimento, e cada camada tem sua responsabilidade bem-definida. Essa definição também facilita a manutenção da aplicação financeira pessoal.

Para o desenvolvimento da aplicação financeira pessoal foi desenvolvido algumas classes que servem de apoio para o funcionamento do sistema. Existem duas classes *Exception* com objetivo de padronizar as exceções que podem acontecer nas camadas de acesso a dados e regra de negócio.

Na página de cadastro de usuário foi utilizados recursos do JSF como: *tag* para exibir mensagens de erro no formulário, campos de texto, validadores, conversores, componente *select.* Na área administrativa da aplicação financeira foi utilizado o componente dataTable do JSF. Esse componente permite o usuário administrador visualizar todos os usuários cadastrados na aplicação financeira. Esse componente também permite ao administrador editar, ativar, desativar, excluir um determinado usuário. Na página de cadastro de contas bancárias foi utilizado recurso Ajax, permitindo que operações de telas sejam feitas com a mínima atualização possível.

A página de categoria serve para classificar os lançamentos financeiros que o usuário realizou. Essa página utiliza o componente *Tree* (árvore) do PrimeFaces. Esse *framework* possui uma vasta biblioteca de componentes que auxilia o JSF.

O *facelets* ajudou a padronizar e centralizar as definições visuais e estruturais das páginas da aplicação. Também ajudou reduzir a quantidade de código-fonte nas páginas do sistema, facilitando a manutenção das páginas.

5. Resultados

Neste trabalho foi explorado o *framework Java Server Faces* utilizando formulário e suas bibliotecas de *tags* principais seguindo um modelo de desenvolvimento *web* tradicional. Também foi utilizado a implementação do JSF da Sun Mojarra que é uma das implementações mais conhecida do JSF. Essa implementação permitiu o uso de componentes HTML, tabelas, layout, conversores, validadores de dados e eventos, além de outros recursos oferecidos pelo JSF, como interação com o *contêiner* Java. A grande vantagem de utilizar o JSF é pelo fato de ele ser um padrão de mercado. Logo, muitas empresas que desenvolvem software *web* utilizando a tecnologia Java investem em desenvolver seus próprios componentes para o JSF. Para a aplicação financeira pessoal foi utilizado uma biblioteca de extensão do JSF, o PrimeFaces. Essa biblioteca contêm suas próprias implementações de tabelas, recurso de ordenação, paginação e filtragem de dados. Dessa biblioteca foi utilizado os recursos de DataGrids com ordenação, paginação e edição, calendário, Treeviews. Também foi utilizado o recurso Ajax do JSF que permite que operações de tela sejam realizadas com a mínima atualização possível, apenas a parte necessária e na hora certa. A vantagem de utilizar o JSF em relação ao desenvolvimento de página web usando JSP e Servlets é a organização e disponibilidade de recursos oferecidos pelo JSF.

Para persistir os dados dos usuários da aplicação financeira foi utilizado o banco de dados MySQL, que é um banco de dados que utiliza uma tecnologia relacional. Atualmente, essa tecnologia de armazenamento de dados está muito bem fundamentada. Como a aplicação financeira pessoal foi desenvolvida utilizando o paradigma de programação orientada por objeto foi necessário utilizar uma solução ORM. Essa solução trata-se de persistir objetos em respectivas tabelas de um banco de dados relacional. Para suprir essa necessidade, na aplicação foi utilizado o Hibernate, que é uma ótima ferramenta de persistência de dados, completa e abrangente. O Hibernate pode lidar com aplicações robustas, sem precisar escrever extensas linhas de SQL, possui uma API bem definida para realizar operações básicas em objetos. De fato essa ferramenta automatizou muitos dos processos repetitivos da aplicação, referentes à manipulação de dados em banco. Com o uso

dessa ferramenta ganhou-se produtividade no decorre do desenvolvimento da aplicação.

Para deixar a aplicação financeira mais segura foi utilizado o *framework* Spring Security que é baseado em injeção de dependência. Na aplicação financeira, esse *framework* ficou responsável de restringir o acesso de usuários indevidos. Foi também utilizado o recurso formulário de *login,* que esse *framework* oferece, garantindo a autenticação e a autorização dos usuários, garantindo assim um baixo acoplamento entre as classes da aplicação financeira pessoal.

Na parte visual da aplicação foi também utilizado o *framework* de *templates Facelets*. Com esse *framework* foi possível padronizar as definições visuais e estruturais das páginas da aplicação financeira. Além disso, esse *framework* ajudou a reduzir a quantidade de código-fonte nas páginas da aplicação.

A aplicação também utiliza recurso de multi-idiomas, assim os usuários podem visualizar a página de cadastro de cheque em três idiomas diferentes: português, inglês e espanhol.

Outro recurso interessante que foi implementado foi a integração da aplicação financeira com o portal de finanças do Yahoo para os usuários visualizarem as cotações de ações na bolsa de valores. A API do Yahoo ofereceu recurso gráfico para a aplicação financeira, deixando o visual da aplicação mais elegante.

A aplicação também utiliza a técnica de envio de e-mail pelo Google Gmail. Para isso é necessário apenas ter uma conta cadastrada no Google. Assim, após o usuário se cadastrar na aplicação financeira pessoal, ele recebe em seu email uma mensagem de boas vindas, com seus dados de *login* e senha para acessar a aplicação. Essa técnica é muito utilizada em aplicação *web*.

E por fim, a aplicação disponibiliza para os usuários a geração de relatórios de diversos formatos, incluindo PDF, Excel, HTML e Open Office. Os relatórios disponibilizados para o usuário são relatórios de contas cadastradas, que se resumem em um relatório mais simples e um relatório mais elaborado, que é um extrato de movimentação realizado pelo usuário. Esse recurso foi desenvolvido utilizando a ferramenta iReport em conjunto com a biblioteca Java, *Jasper Reports*.

5.1. Conclusão

A aplicação financeira pessoal para web foi desenvolvida utilizando as melhores técnicas que a plataforma Java para *web* oferece. O ambiente de desenvolvimento é composto pela integração das ferramentas Java, Apache Tomcat, Eclipse e Mysql. O projeto da aplicação financeira também é composto por várias bibliotecas do Java.

É utilizada a técnica de *annotations* do JSF para realizar o mapeamento. Também foi utilizado recurso Ajax oferecido pelo JSF. A identidade visual para o sistema foi definido pelo *framework* de *templates Facelets*. Também foi utilizado vários componentes do *framework* PrimeFaces, como DataTable e componentes de calendário. O componente DataTable é utilizado para manipulação de dados em tabela.

Foi implementado o recurso de internacionalização que exibe a página de cheques em três idiomas diferentes. O Hibernate realizou os mapeamentos clássicos do modelo relacional usando objetos. Foi possível gerar relatórios em diversos formatos, com a biblioteca JasperReport, incluindo PDF, Excel e HTML.

O desenvolvimento em camadas e MVC foi utilizado para manter a qualidade e organização da aplicação. Foi implementado recurso de segurança usando o Spring Security. Também foi implementado recurso de envio de e-mail após o usuário realizar o cadastro na aplicação.

A aplicação financeira comprova a aplicabilidade das técnicas e ferramentas que auxiliam a plataforma Java para *web*.

Vale lembrar que o aplicativo financeiro não tem por objetivo ser um produto completo de grande porte: ele serve apenas de base para aplicar as tecnologias de Java para *web*. Existem ainda dezenas de recursos que podem ser adicionados ao projeto.

6. Considerações Finais

Este trabalho apresenta diversas técnicas poderosas e populares que existem no ambiente de programação Java para *web*.

A tecnologia Java Server Faces que é um framework baseado em componentes, surgiu com a necessidade de agilizar e automatizar processos de desenvolvimento. O JSF proporciona recursos como validação, conversão, ordenação, paginação dentre outros inúmeros recursos que essa tecnologia oferece. O desenvolvedor também fica livre para criar seus próprios componentes visuais. Existem também muitas bibliotecas de terceiros que auxiliam esse framework, assim estendendo ainda mais suas funcionalidades. A grande vantagem de utilizar o JSF é a disponibilidade de recursos que ele oferece para um projeto de software. Atualmente, virou um padrão de mercado para o desenvolvimento de aplicações em plataforma Java para *web*.

Outra tecnologia apresentada neste trabalho é o Hibernate que tem o propósito de padronizar a interação da aplicação com o banco de dados. Essa tecnologia também automatiza muitos processos referentes à manipulação de dados em um banco de dados. A grande vantagem de utilizar a ferramenta Hibernate é o seu ferramental oferecido para persistência de objetos de forma automatizada e transparente. A tecnologia pode lidar com aplicações robustas, sem precisar escrever extensas linhas de SQL. É também uma tecnologia consolidada no mercado de desenvolvimento, grandes empresas fazem seu uso.

Como alvo de estudo deste trabalho foi desenvolvido uma aplicação financeira pessoal utilizando essas técnicas de desenvolvimento da plataforma Java para *web*. Dentre várias tecnologias utilizadas nessa aplicação, as principais foram o JSF e o Hibernate. Unindo essas duas tecnologias foi possível criar uma interface *web* mais amigável, automatizar operações em banco de dados, tratar questões de segurança, aplicar a internacionalização em páginas *web*, gerar gráficos e relatórios. Essas técnicas utilizadas juntas oferecem ao desenvolvedor agilidade, manutenibilidade, portabilidade, reusabilidade e principalmente no quesito da evolução do software.

REFERÊNCIAS*

ARNDT, Von. Staa. **Programação Modular**. Editora Campus, 2000. Cap 4: Padrões de Programação; Cap 5: Princípios de Modularidade.

BAUER, Christian; KING, Gavin. **Java Persistence with Hibernate**. New York: Manning Publications Co, 2007. ISBN: 1-932394-88-5.

BERTRAND, Meyer. **Object-oriented Software Construction**, Prentice-Hall International Series in Computer Science, C.A.R. Hoare Series Editor, 2nd Edition, 1997.

BIGONHA, Roberto. S; BIGONHA, Mariza A. S. **Programação Modular**. Notas de Aula, DCC, UFMG, 2015.

BITTENCOURT, M. V. S. **Estudo Comparativo entre Frameworks Java para Construção de Aplicações Web.** Universidade Federal de Santa Maria, RS (2004). Disponível em: <http://www-app.inf.ufsm.br/bdtg/arquivo.php?id=20&download=1> Acessado em: 25 de junho de 2015.

BOND, M. et al**.: Aprenda J2EE com EJB, JSP, Servlets, JNDI, JDBC e XML em 21 dias**. São Paulo: MAKRON Books, 2003.

CALÇADO, P. **Arquitetura de Camadas em Java EE. Mundo Java**. Rio de Janeiro, v.3, n.15, p.34-43, 2005.

CHEROBIM A. P. M. S; ESPEJO M. M. S. B. **Finanças pessoais: conhecer para enriquecer**. São Paulo: Atlas, 2010.

COELHO H. **JPA Eficaz: As melhores práticas de persistência de dados em Java**. Casa do Código. Nov/2013. ISBN: 978-85-66250-31-2

COELHO H. **JSF Eficaz: As melhores práticas para o desenvolvedor web Java**. Casa do Código. Fev/2013. ISBN: 978-85-66250-19-0

CORDEIRO, Gilliard. **Aplicações Java para a web com JSF e JPA**. 1 ed. São Paulo: Casa do Código, 2012.

DATE, Christopher J. **Introdução a Sistemas de Banco de Dados**. Tradução de Daniel Vieira. 8. Ed. Rio de Janeiro: Editora Elsevier, 2003.

DEITEL, M. Harvey; DEITEL, J. Paul; NIETO, R. Nieto. **Internet & World Wide Web, como programar**. Trad. Edson Furmankiewicz – 2.ed. Porto Alegre: Bookman, 2003.

* Baseadas na norma NBR 6023, de 2002, da Associação Brasileira de Normas Técnicas (ABNT).

ELMASRI, R. E.; NAVATHE, S.B. **Sistemas de banco de dados**. 4.ed. Rio de Janeiro: Assidon-Weslesy, 2005.

FOWLER, M. **Inversion of Control Containers and the Dependency Injection pattern.** Disponível em: http://www.martinfowler.com/articles/injection.html. Acesso em 20 Outubro 2015.

GONCALVES, A. **Introducao a Plataforma Java EE6 com GlassFish 3**. 1ª Edição, 2011. ISBN: 9788539900961

HEFFELFINGER, R. David. **JasperReports for Java Developers:** Create, Design, Format, and Export Reports with the Worlds's Most Popular Java Reporting Library. Estados Unidos: editora Packt publishing, August 2006. ISBN 139781904811909.

HOJI, Masakazu. **Administração financeira e orçamentária: matemática financeira aplicada, estratégias financeiras, orçamento empresarial**. São Paulo: Atlas, 2010.

JOHNSON, Rod. **Expert One-on-One J2EE Design and Development.** Estados Unidos: editora **Wrox,** October 2002. ISBN: 978-0-7645-4385-2.

KEN, Arnold; GOSLING, James; HOLMES, David. **A Linguagem de Programação Java**. Bookman, Quarta Edição, 2007, ISBN 978-85-60031-64-1.

KONDA, Madhusudhan. **Introdução ao Hibernate**. Brasil: Novatec Editora, 2014. ISBN: 8575223550

LUCKOW, Décio Heinzelmann; MELO, Alexandre Altair. **Programação Java para Web: aprenda a desenvolver uma aplicação financeira pessoal com as ferramentas mais modernas da plataforma Java**. São Paulo: Novatec Editora, 2010. ISBN: 978-85-7522-238-6.

MOJARRA JAVASERVER FACES - **Oracle's open source implementation of the JSF standard**. Disponível em: < https://javaserverfaces.java.net >. Acesso em: Setembro de 2015.

MACHACEK, Jan; VUKOTIC, Aleksa; CHAKRABORTY, Anyrvan; DITT, Jessica. **Pro Spring 2.5**. New York, USA: Editora Ciência Moderna, 2008. ISBN: 9788573938159.

MAHMOUD, Qusay. H. **Developing Web Applications with JavaServer Faces.** Oracle Articles, agosto 2004. Disponível em: http://www.oracle.com/technetwork/articles/javase/javaserverfaces-135231.html. Acesso em agosto 2015.

ORACLE, **Your First Cup**: **An Introduction to the JavaTM EE Platform**. Janeiro 2013. Disponível em: http://docs.oracle.com/javaee/6/firstcup/doc/firstcup.pdf. Acesso em: 02 novembro 2015.

ORACLE, **Your First Cup: Differences between Java EE and Java SE** Disponível em: http://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html. Acesso em 15 dez 2015.

ORACLE, **Java EE at a Glance.** Disponível em: **http://www.oracle.com/technetwork/java/javaee/overview/index.html** Acesso em 15 dez 2015.

ORACLE WHITE PAPER. **Introduction to Java Platform, Enterprise Edition 7**. 2013. Disponível em: http://www.oracle.com/technetwork/java/javaee/javaee7-whitepaper-1956203.pdf . Acesso em 20 junho 2015.

ORACLE. **Java Platform, Enterprise Edition: The Java EE Tutoria**. 2014. Disponível em: https://docs.oracle.com/javaee/7/JEETT.pdf. Acesso em agosto de 2015.

ORACLE. **JavaServer Faces Technology**. Disponível em: http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html. Acesso em novembro de 2015.

PUGH, Eric; GRADECKI, D. Joseph. **Professional Hibernate**. Estados Unidos: John Wiley Consumer, 2004. ISBN: 0764576771.

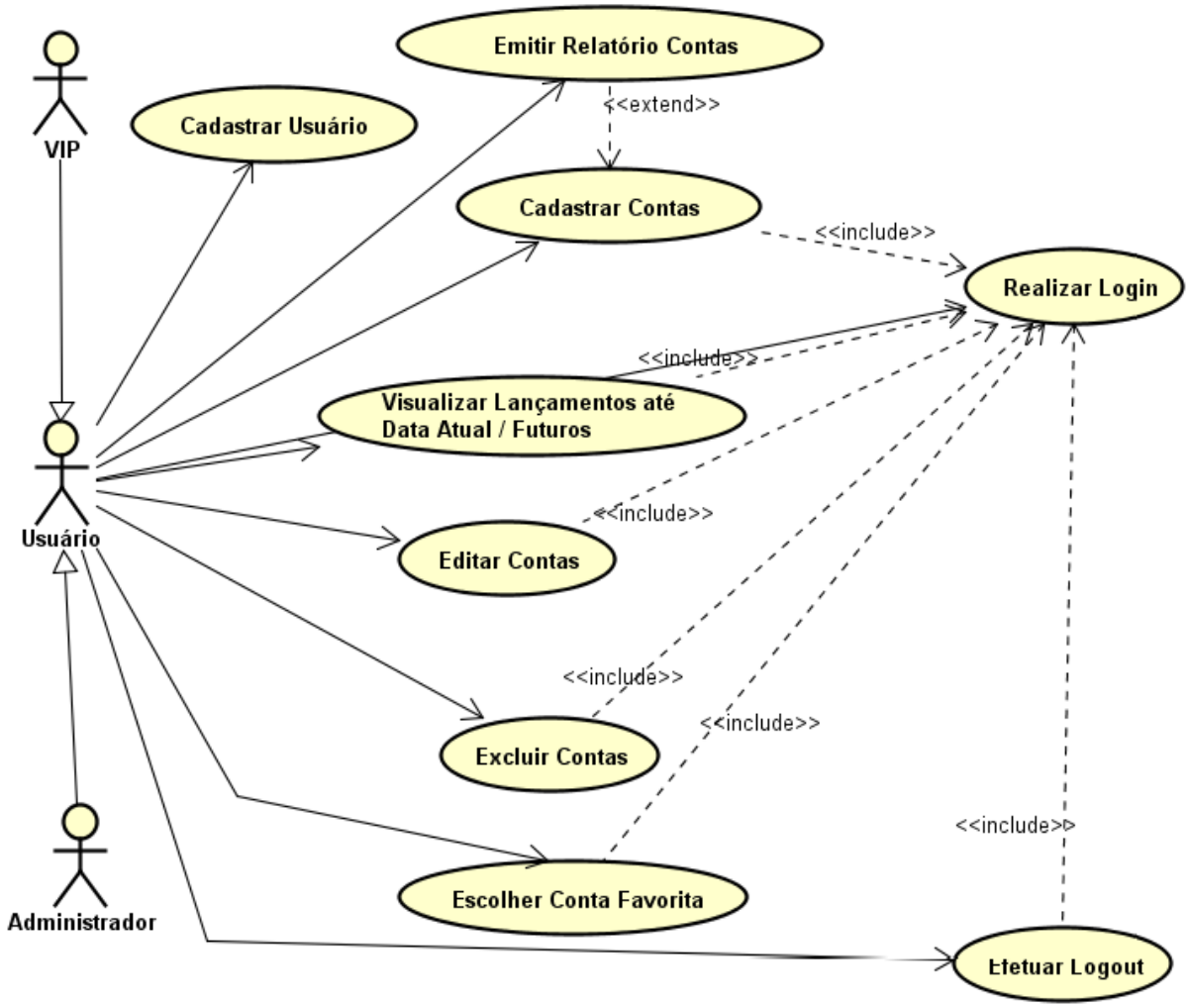
SOUZA, Thiago Hernandes. **Java + Primefaces + iReport:** Desenvolvendo um CRUD para web. Brasil: Editora Ciência Moderna, 2013 ISBN: 9788539904228.

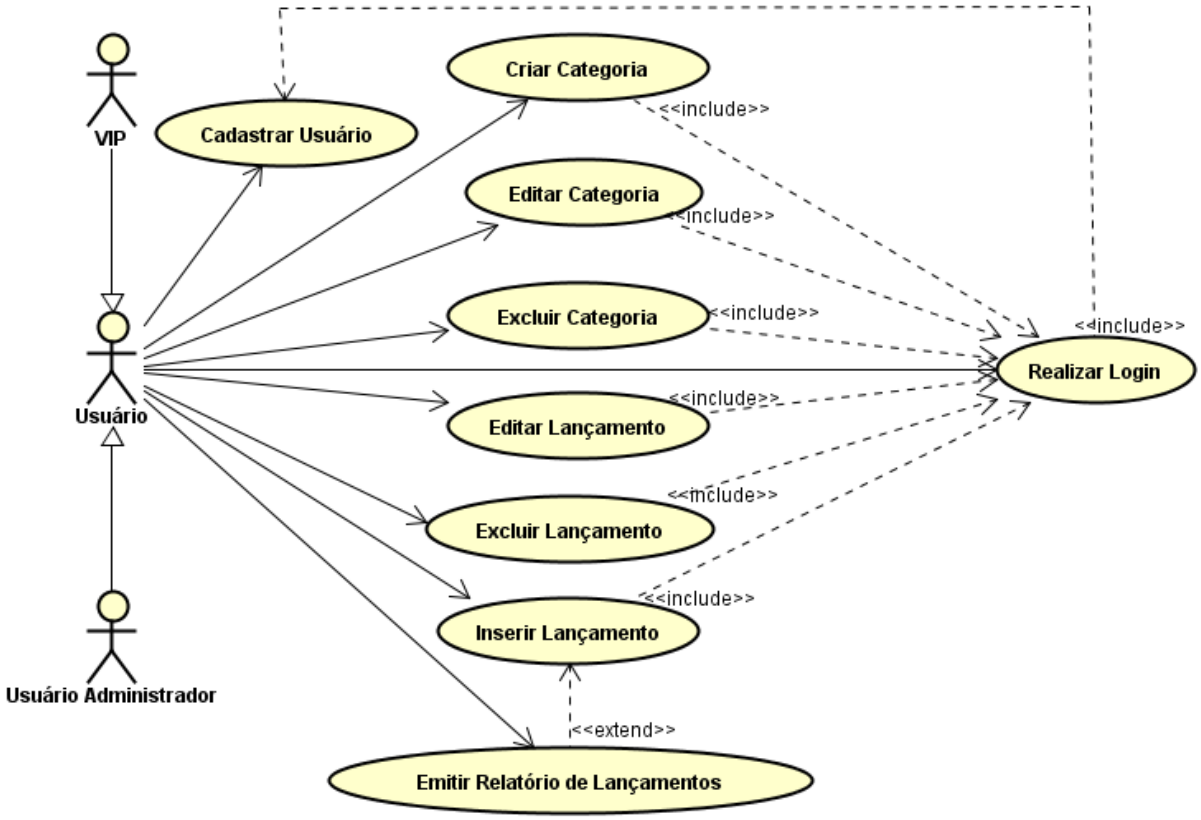
SCOTT, W. Ambler. **Análise e Projeto Orientados a Objeto**, Volume 2, IBPI Press, Livraria e Editora Infobook S.A., 1998. Cap 3: Camadas de Software.

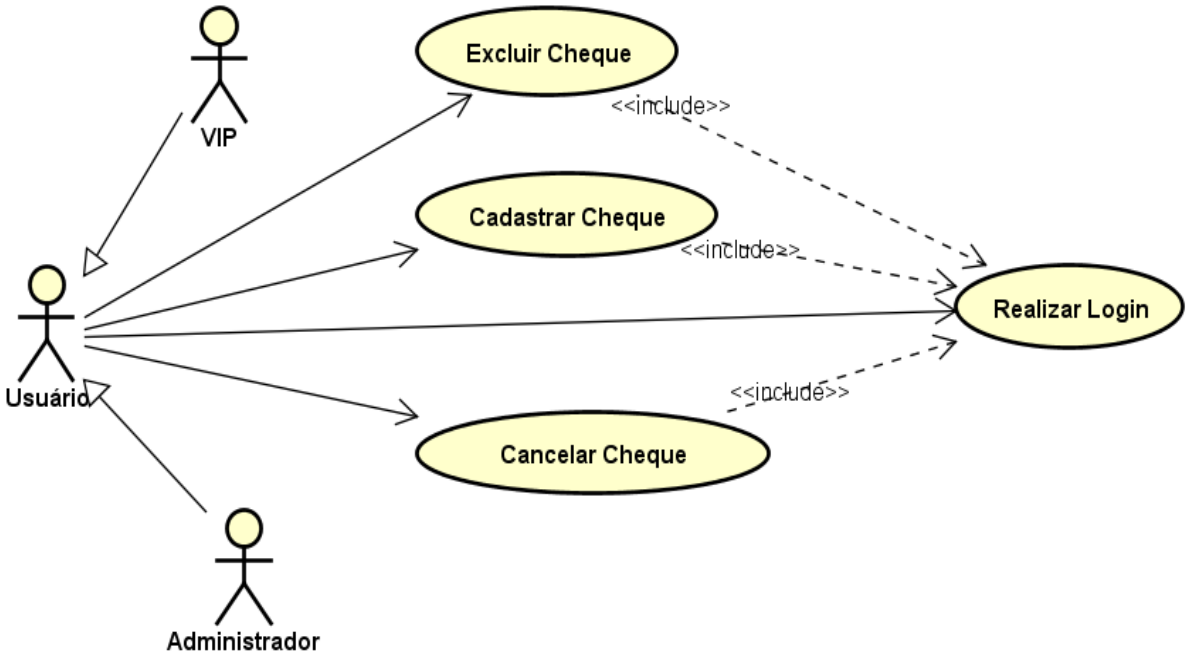
SERASA EXPERIAN. **Estudo inédito da Serasa Experian traça o Mapa da Inadimplência no Brasil em 2014.** Disponível em: <http://www.serasaexperian.com.br/estudo-inadimplencia/>. Acesso em 19 dezembro 2015.

Apêndice A

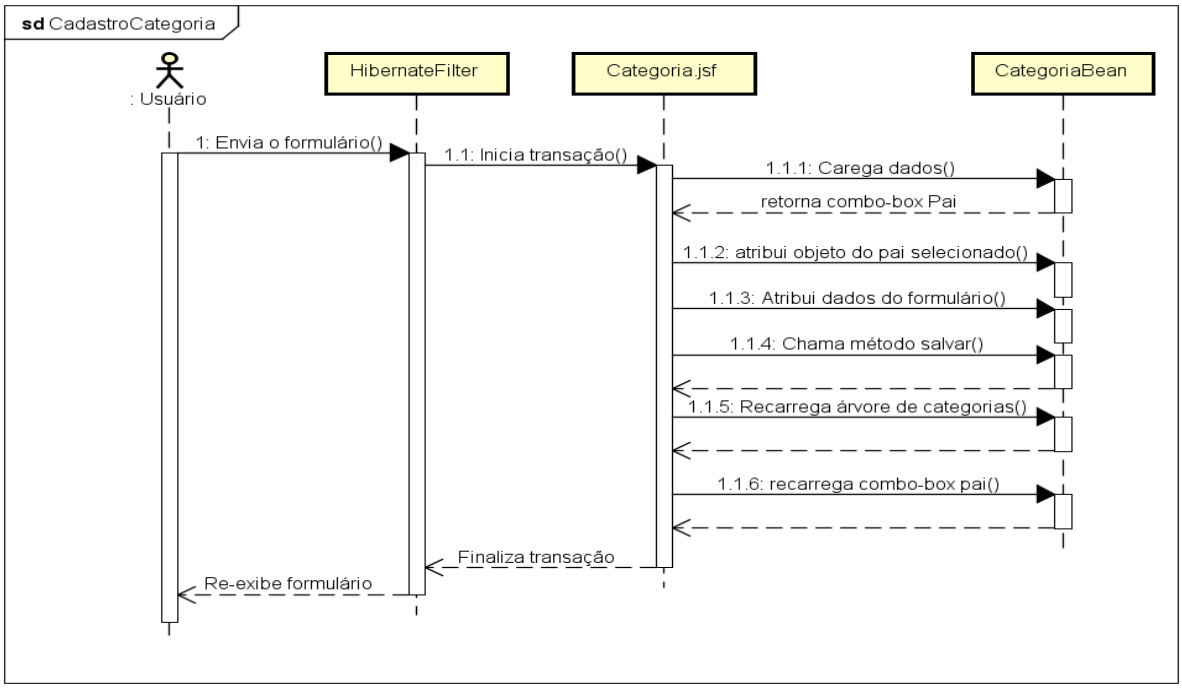
Diagramas de Caso de Uso

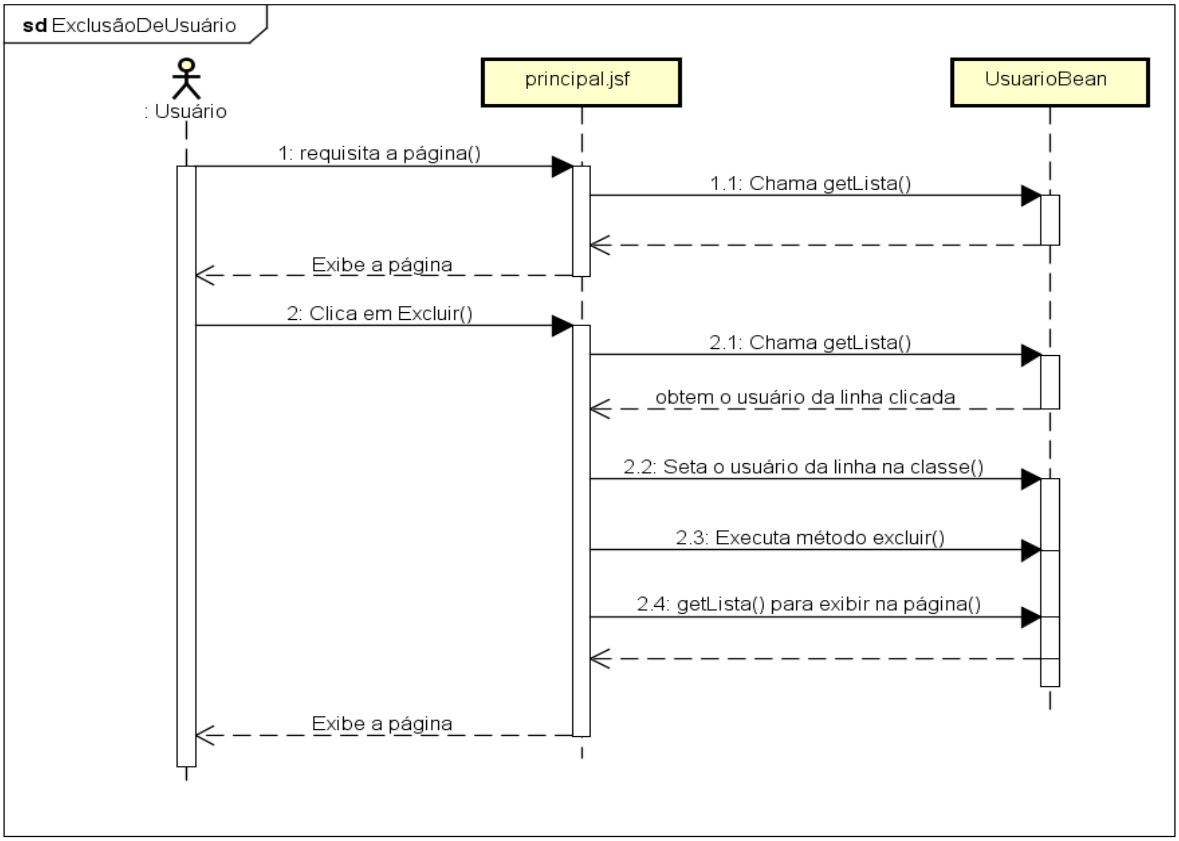


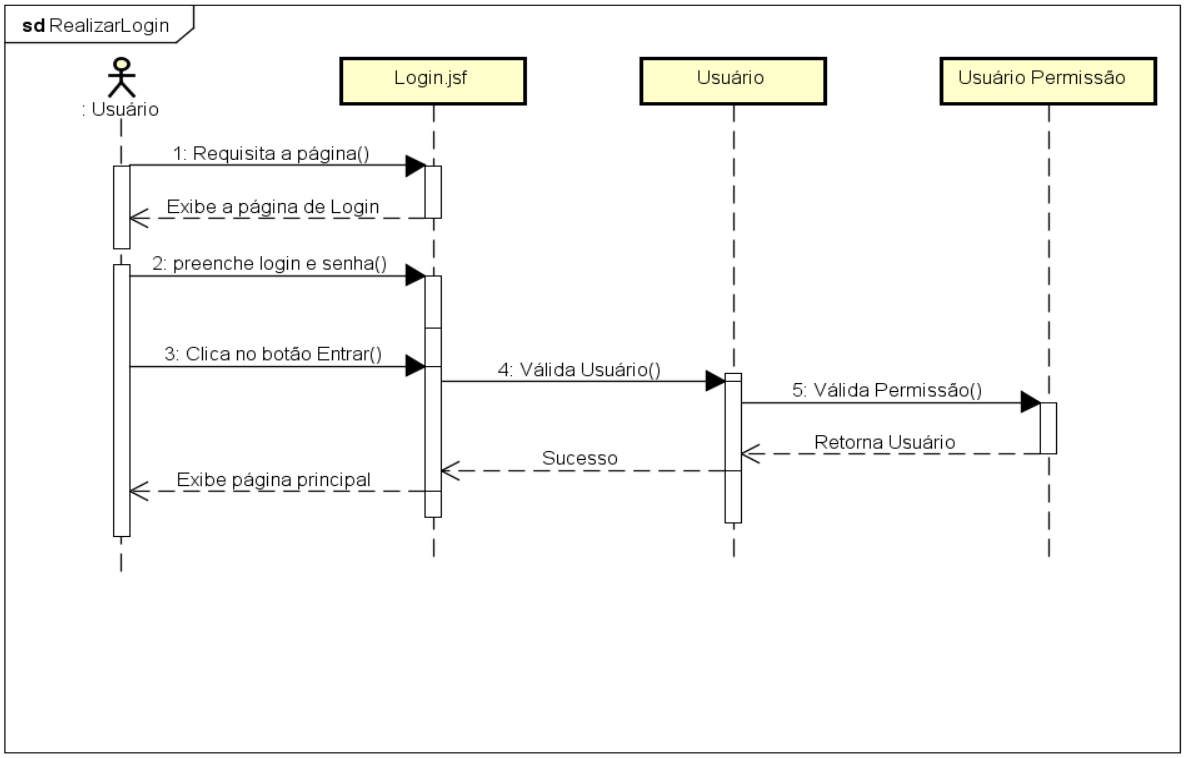

VIP
Cadastrar Usuário
Criar Categoria
Editar Categoria
Excluir Categoria
Editar Lançamento
Excluir Lançamento
Inserir Lançamento
Emitir Relatório de Lançamentos
Realizar Login
Usuário
Usuário Administrador
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
<<include>>
<<extend>>


VIP
Excluir Cheque
Cadastrar Cheque
Cancelar Cheque
Realizar Login
Usuário
Administrador
<<include>>
<<include>>
<<include>>

Diagramas de Sequencia






sd RealizarLogin
: Usuário
Login.jsf
Usuário
Usuário Permissão
1: Requisita a página()
Exibe a página de Login
2: preenche login e senha()
3: Clica no botão Entrar()
4: Válida Usuário()
5: Válida Permissão()
Retorna Usuário
Sucesso
Exibe página principal

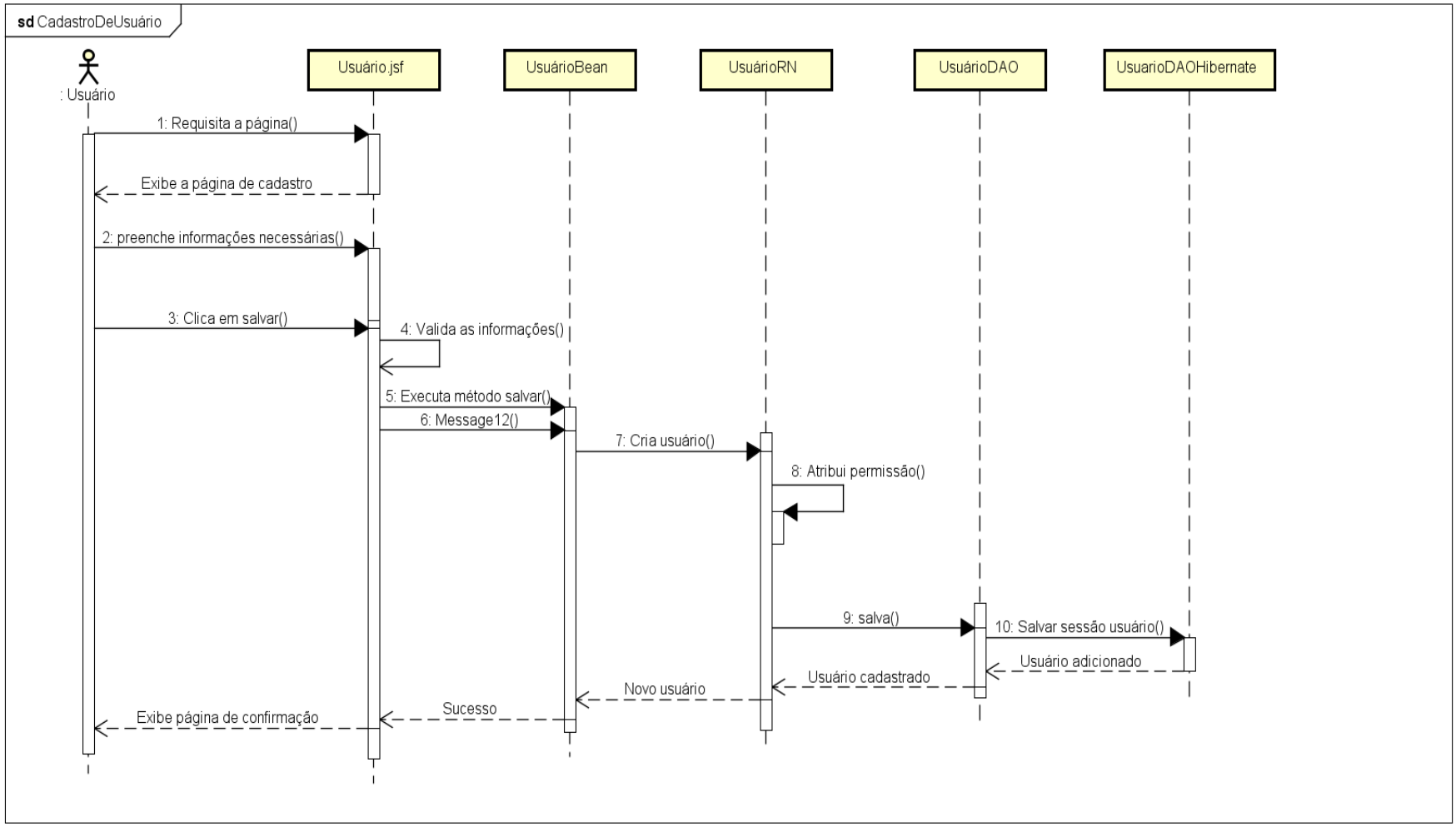

sd CadastroDeUsuário
: Usuário
Usuário.jsf
UsuárioBean
UsuárioRN
UsuárioDAO
UsuarioDAOHibernate
1: Requisita a página()
Exibe a página de cadastro
2: preenche informações necessárias()
3: Clica em salvar()
4: Valida as informações()
5: Executa método salvar()
6: Message12()
7: Cria usuário()
8: Atribui permissão()
9: salva()
10: Salvar sessão usuário()
Usuário adicionado
Usuário cadastrado
Novo usuário
Sucesso
Exibe página de confirmação

Diagrama de Componente

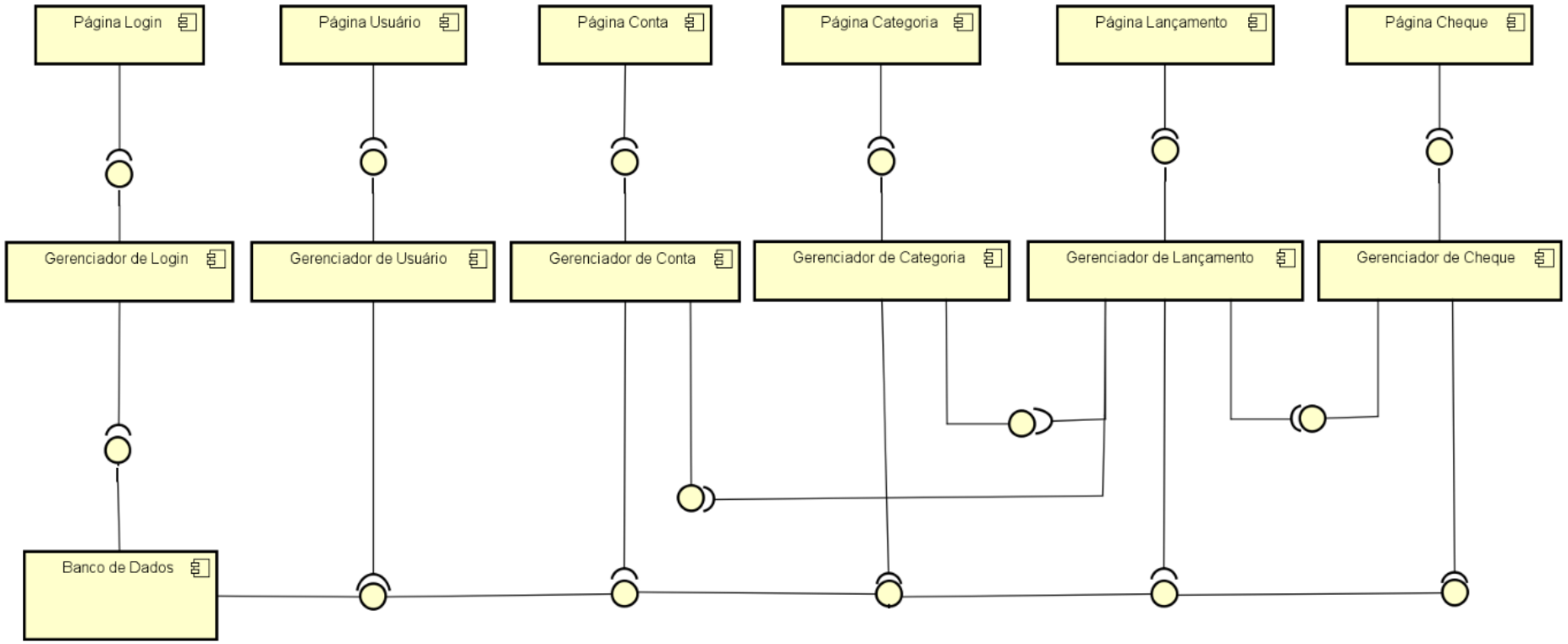
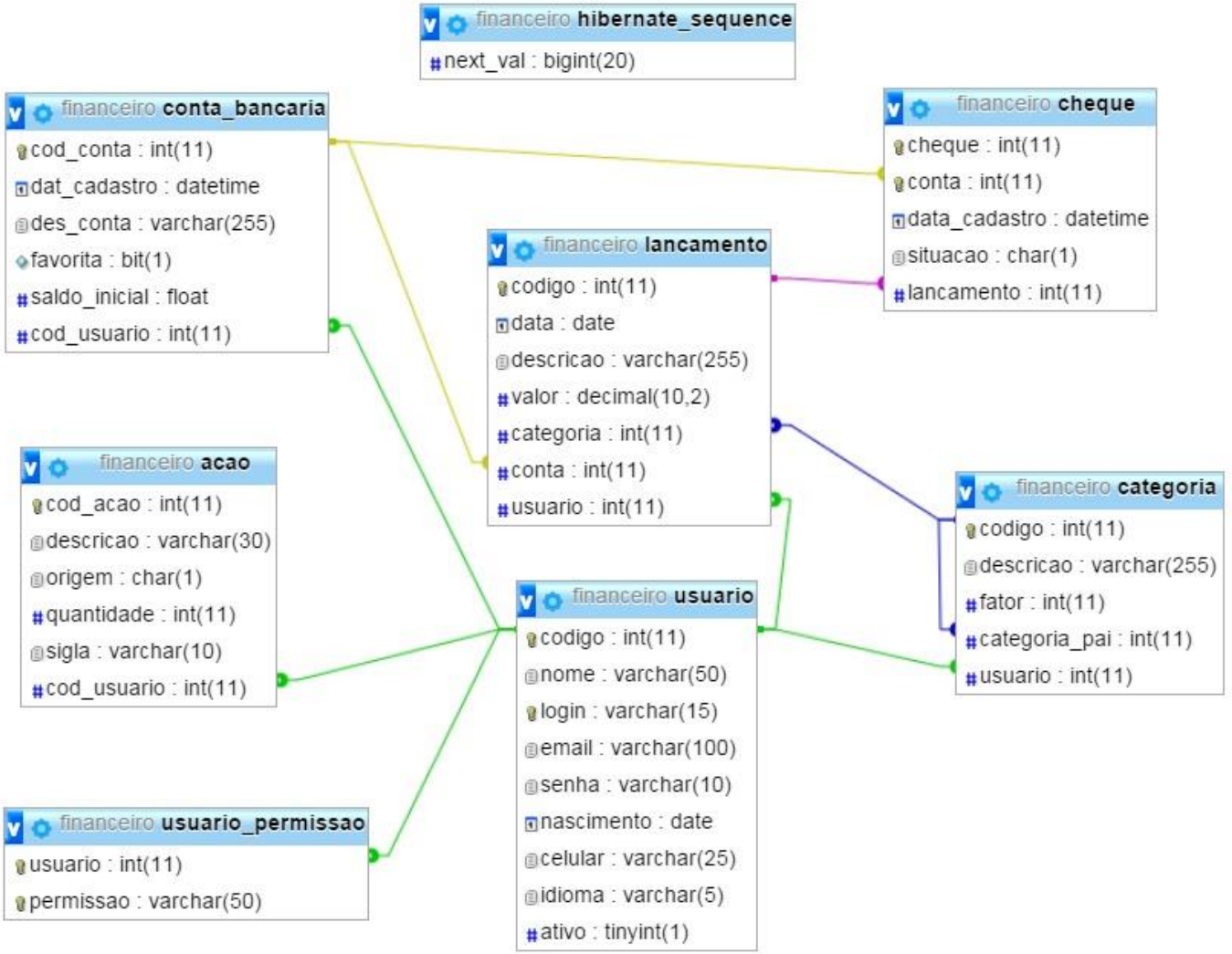


Diagrama de Entidade e Relacionamento

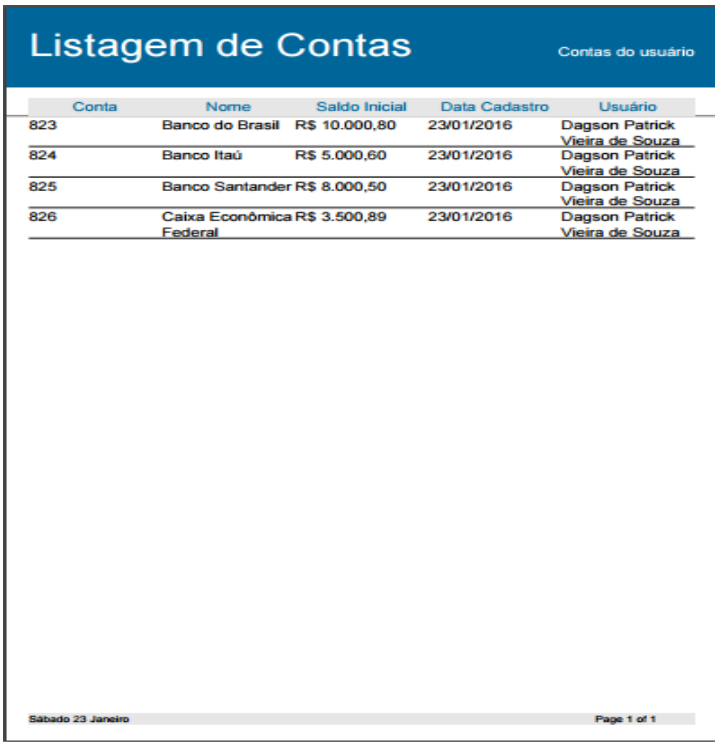


Páginas Login

Páginas principal, após o login do usuário

Página de Lançamentos

Página Cadastro de Cheques



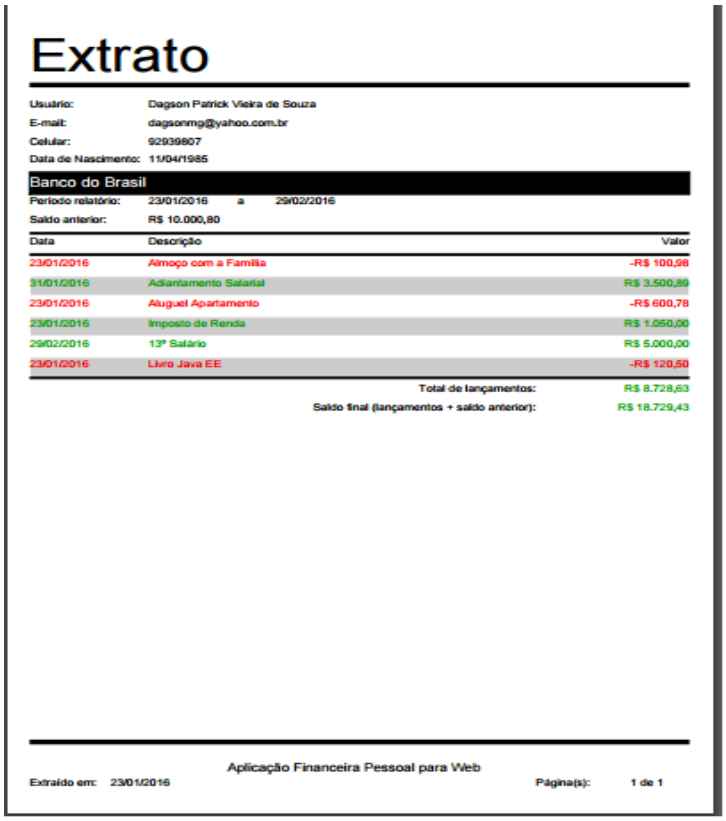
Listagem de Contas

Contas do usuário

Conta	Nome	Saldo Inicial	Data Cadastro	Usuário
823	Banco do Brasil	R$ 10.000,80	23/01/2016	Dagson Patrick Vieira de Souza
824	Banco Itaú	R$ 5.000,60	23/01/2016	Dagson Patrick Vieira de Souza
825	Banco Santander	R$ 8.000,50	23/01/2016	Dagson Patrick Vieira de Souza
826	Caixa Econômica Federal	R$ 3.500,89	23/01/2016	Dagson Patrick Vieira de Souza

Sábado 23 Janeiro Page 1 of 1

Layout da Listagem de Contas



Extrato

Usuário: Dagson Patrick Vieira de Souza
E-mail: dagsonmg@yahoo.com.br
Celular: 92939807
Data de Nascimento: 11/04/1985

Banco do Brasil

Período relatório: 23/01/2016 a 29/02/2016
Saldo anterior: R$ 10.000,80

Data	Descrição	Valor
23/01/2016	Almoço com a Família	-R$ 100,98
31/01/2016	Adiantamento Salarial	R$ 3.500,89
23/01/2016	Aluguel Apartamento	-R$ 600,78
23/01/2016	Imposto de Renda	R$ 1.050,00
29/02/2016	13º Salário	R$ 5.000,00
23/01/2016	Livro Java EE	-R$ 120,50

Total de lançamentos: R$ 8.728,63
Saldo final (lançamentos + saldo anterior): R$ 18.729,43

Aplicação Financeira Pessoal para Web

Extraído em: 23/01/2016 Página(s): 1 de 1

Layout do Extrato

Especificação dos casos de uso

UC001 – Cadastrar Usuário

Identificação: UC001	
Nome: cadastrar Usuário	
Atores: usuário, usuário administrativo, usuário vip	
Tipo: primário	
Pré-condições: usuário deve ter acesso a aplicação financeira pessoal por algum meio (via Internet ou localhost).	
Pós-condições: usuário cadastrado na base de dados e uma mensagem de cadastrado realizado com sucesso.	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no link Registre-se	4. Valida informações preenchidas pelo usuário
2. Preenche informações do formulário (nome, data nascimento, celular, email, idioma, login, senha, descrição e saldo inicial)	5. Realiza o cadastro do usuário
3. Clica no botão salvar	
Sequencia alternativa	
4a. Informações preenchidas incorretamente	
1.a Exibe mensagem de erro e informa ao usuário o primeiro campo com problema.	

UC002 – Realizar Login

Identificação: UC002
Nome: realizar login
Atores: usuário, usuário administrativo, usuário vip
Tipo: primário

Pré-condições: estar cadastrado na aplicação financeira pessoal para web	
Pós-condições: usuário redirecionado para a página principal (home) do sistema.	
Sequencia típica de eventos	
Ator	Sistema
1. Preenche informações do formulário de login (login e senha, informado no cadastro de usuário).	4. Valida login e senha do usuário no banco de dado da aplicação.
2. Clica no link Entrar	5. Redireciona o usuário para a página principal (home) do sistema.
Sequencia alternativa	
4a. Usuário não encontrado pelo sistema no banco de dados.	
1. Exibe mensagem “Atenção, erro ao efetuar o login. Motivo: credenciais incorreta”.	

UC003 – Cadastrar Contas

Identificação: UC003	
Nome: cadastrar contas	
Atores: usuário, usuário administrativo, usuário vip	
Tipo: primário	
Pré-condições: estar logado na aplicação financeira pessoal para web	
Pós-condições: conta cadastrada na base de dados da aplicação	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no botão “Contas” na barra de menu principal	2. Envia o usuário à página de contas, essa página contém um formulário para cadastrar uma nova conta e uma lista com as contas já castradas pelo usuário.

3. Preenche informações do formulário de Contas (descrição da conta e saldo inicial)	5. Valida informações preenchidas pelo usuário
4. Clica no botão “Salvar”	6. Realiza o cadastro da conta informada
Sequencia alternativa	
5a. Conta não cadastrada no sistema devido algum erro de validação.	
3. Exibe mensagem “Erro de validação, informa os campos obrigatórios”.	

UC004 – Editar Contas

Identificação: UC004	
Nome: editar Contas	
Atores: usuário, usuário administrativo, usuário vip	
Tipo: secundário	
Pré-condições: estar logado e já possuir conta cadastrada no sistema	
Pós-condições: atualizar a tabela de contas e apresentar ao usuário a conta alterada	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no botão “Contas” na barra de menu principal	2. Envia o usuário à página de contas, essa página contém um formulário para cadastrar uma nova conta e uma lista (com opção editar e excluir) para as contas já castradas.
3. Usuário clica no ícone alterar da conta que deseja modificar.	4. Recupera as informações da conta selecionada e disponibiliza no formulário de cadastro de conta
5. Altera as informações desejadas	7. Valida as informações preenchidas pelo usuário
6. Usuário clica no botão salvar.	8. Efetiva a alteração

Sequencia alternativa	
7a. Conta não alterada no sistema devido algum erro de validação.	
5. Exibe mensagem “Erro de validação, informa os campos obrigatórios”.	

UC005 – Excluir Contas

Identificação: UC005	
Nome: excluir Contas	
Atores: usuário, usuário administrativo, usuário vip	
Tipo: secundário	
Pré-condições: estar logado e já possuir conta cadastrada no sistema	
Pós-condições: atualizar a tabela de contas e apresentar ao usuário	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no botão “Contas” na barra de menu principal	2. Envia o usuário à página de contas, essa página contém um formulário para cadastrar uma nova conta e uma lista (com opção editar e excluir) para as contas já castradas.
3. Usuário clica no ícone lixeira da conta que deseja excluir.	4. Efetiva a exclusão da conta seleciona pelo usuário
	5. Atualiza a tabela de contas cadastradas na página contas
Sequencia alternativa	
2a. Não existe contas cadastradas pelo usuário.	
3. Usuário tem a opção de cadastrar uma nova conta.	

UC006 – Escolher Conta Favorita

Identificação: UC006	
Nome: escolher Conta Favorita	
Atores: usuário, usuário administrativo, usuário vip	
Tipo: secundário	
Pré-condições: estar logado e já possuir conta cadastrada no sistema	
Pós-condições: ativar conta como favorita e apresentar o resultado (ícone destacado) ao usuário	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no botão “Contas” na barra de menu principal	2. Envia o usuário à página de contas, essa página contém um formulário para cadastrar uma nova conta e uma lista (com ícone editar, excluir e favorita) para cada conta cadastrada.
3. Usuário clica no ícone ☆ estrela da conta que deseja tornar como favorita.	4. Altera a conta favorita do usuário no sistema.
	5. Atualiza a tabela de contas
Sequencia alternativa	
2a. Não existe contas cadastradas pelo usuário.	
3a. Usuário tem a opção de cadastrar uma nova conta.	

UC007 – Efetuar Logout

Identificação: UC007
Nome: efetuar logout
Atores: usuário, usuário administrativo, usuário vip
Tipo: secundário

Pré-condições: estar logado no sistema	
Pós-condições: encerrar sessão do usuário e redireciona o usuário para a página de login	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no ícone  “Sair” na barra de menu principal	2. Encerrar a sessão do usuário no sistema
	3. Redireciona o usuário para a página de login.
Sequencia alternativa	
2a. Sistema não consegue encerrar a sessão do usuário devido fatores externos.	
3a. Apresenta para o usuário o evento ocorrido e o motivo dele.	

UC008 – Emitir Relatório Contas

Identificação: UC008	
Nome: emitir Relatório Contas	
Atores: usuário, usuário administrativo, usuário vip	
Tipo: primário	
Pré-condições: estar logado no sistema e já possuir conta cadastrada no sistema	
Pós-condições: enviar o arquivo solicitado para a máquina do usuário	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no botão “Contas” na barra de menu principal	2. Envia o usuário à página de contas, essa página contém um ícone para cada formato de relatório desejado pelo usuário.
3. Clica no ícone do formato desejado	4. Enviar o arquivo solicitado a máquina do usuário.

Sequencia alternativa
2a. Não existe contas cadastradas pelo usuário.
3. Enviar o arquivo solicitado, porém sem dados.

UC009 – Visualizar Lançamentos

Identificação: UC009	
Nome: visualizar vançamentos	
Atores: usuário, usuário administrativo, usuário vip	
Tipo: primário	
Pré-condições: já possuir lançamento registrado no sistema	
Pós-condições: enviar a página principal (home) com os lançamentos realizados até o momento	
Sequencia típica de eventos	
Ator	Sistema
1. Realizar caso de uso: efetuar login	2. Enviar a página principal com os lançamentos realizados até a data atual e com lançamentos de datas futuras
Sequencia alternativa	
2a. Não existe lançamento registrado pelo usuário.	
2b Envia a página principal para o usuário, porém sem dados de lançamento.	

UC010 – Criar Categoria

Identificação: UC010
Nome: criar Categoria
Atores: usuário
Tipo: primário

Pré-condições: usuário estar logado no sistema	
Pós-condições: categoria cadastrada na base de dados e atualiza a página de categoria	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no botão “Categoria” na barra de menu principal	2. Envia a página de categorias para o usuário. Essa página contém algumas categorias básicas já cadastradas.
3. Clica no botão novo	4. Envia o formulário de categoria para o usuário.
5. Preenche o formulário, seleciona a categoria pai e informa a descrição da categoria	7. Valida as informações preenchidas pelo usuário
6. clique no botão salvar	8. Efetiva a operação de inserção de nova categoria
Sequencia alternativa	
7a Exibe mensagem “Erro de validação, informa os campos obrigatórios”.	
3 Reenvia a página categoria para o usuário novamente.	

UC011 – Editar Categoria

Identificação: UC011	
Nome: editar Categoria	
Atores: usuário, usuário administrativo, usuário vip	
Tipo: secundário	
Pré-condições: usuário estar logado no sistema e já possuir categoria cadastrada no sistema	
Pós-condições: categoria editada na base de dados conforme usuário.	
Sequencia típica de eventos	
Ator	Sistema

1. Clica no botão “Categoria” na barra de menu principal	2. Envia o formulário de categorias para o usuário. Essa página contém algumas categorias básicas (despesas e receitas) já cadastradas.
3. Seleciona a categoria pai deseja e informa a descrição da categoria	5. Valida as informações preenchidas pelo usuário
4. Clica no botão salvar	6. Atualiza a página de categoria para o usuário
Sequencia alternativa	
5a Exibe mensagem: erro de validação, clique em novo e informe a descrição da categoria novamente	
5b Reenvia a página categoria para o usuário novamente	

UC012 – Excluir Categoria

Identificação: UC012	
Nome: excluir Categoria	
Atores: usuário, usuário administrativo, usuário vip	
Tipo: secundário	
Pré-condições: usuário estar logado no sistema e já possuir categoria cadastrada no sistema	
Pós-condições: categoria deletada na base de dados conforme o usuário.	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no botão “Categoria” na barra de menu principal	2. Envia o formulário de categorias para o usuário. Essa página contém algumas categorias básicas (despesas e receitas) já cadastradas.
3. Seleciona a categoria que deseja excluir	5. Deleta a categoria no banco de dados do sistema
4. Clica no botão excluir	6. Atualiza a página de categoria para o usuário

Sequencia alternativa	
5a Exibe alguma mensagem de exceção levantada pelo sistema naquele momento. Logo, a categoria não é deletada no sistema.	

UC013 – Inserir Lançamento

Identificação: UC013	
Nome: inserir lançamento	
Atores: usuário, usuário administrativo, usuário vip	
Tipo: primário	
Pré-condições: usuário estar logado no sistema	
Pós-condições: lançamento cadastrado na base de dados	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no botão “Lançamento” na barra de menu principal	2. Envia a página de lançamentos para o usuário. Essa página apresenta os lançamentos realizados pelo usuário e algumas opções sobre cada lançamento.
3. Seleciona a data que deseja realizar o lançamento e a categoria. Preenche a descrição, valor e se desejar informe o número do cheque	5. Valida as informações preenchidas pelo usuário
4. Clica no botão salvar	6. Efetiva a operação de inserir lançamento no banco de dados do sistema
	7. Atualiza a página de lançamento para o usuário visualizar
Sequencia alternativa	
5a Exibe mensagem “Erro de validação, informa os campos obrigatórios”.	

UC014 – Alterar Lançamento

Identificação: UC014	
Nome: alterar lançamento	
Atores: usuário, usuário administrativo, usuário vip	
Tipo: primário	
Pré-condições: usuário estar logado no sistema e o usuário já possuir algum lançamento já cadastrado	
Pós-condições: lançamento cadastrado na base de dados	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no botão “Lançamento” na barra de menu principal	2. Envia a página de lançamentos para o usuário. Essa página apresenta os lançamentos realizados pelo usuário e algumas opções sobre cada lançamento.
3. Clica no ícone alterar do lançamento que deseja modificar.	4. Recupera as informações do lançamento selecionado e disponibiliza no formulário de lançamento
5. Altera as informações desejadas do lançamento	7. Valida as informações preenchidas pelo usuário
6. Usuário clica no botão salvar.	8. Efetiva a alteração no banco de dados do sistema
Sequencia alternativa	
7a Exibe mensagem “Erro de validação, informa os campos obrigatórios”.	

UC015 – Excluir Lançamento

Identificação: UC015
Nome: excluir lançamento
Atores: usuário, usuário administrativo, usuário vip
Tipo: secundário

Pré-condições: estar logado e já possuir lançamento cadastrado no sistema	
Pós-condições: lançamento excluído na base de dados do sistema	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no botão “Lançamento” na barra de menu principal	2. Envia a página de lançamentos para o usuário. Essa página apresenta os lançamentos realizados pelo usuário e algumas opções sobre cada lançamento.
3. Usuário clica no ícone  lixeira do lançamento que deseja excluir.	4. O sistema pergunta em forma de mensagem para o usuário, se o mesmo deseja realmente excluir esse lançamento.
5. Usuário escolhe a opção sim	6. Efetiva a exclusão do lançamento selecionado pelo usuário
	7. Atualiza a tabela de lançamentos na página de lançamento
Sequencia alternativa	
5a. Usuário escolhe a opção não dejeta excluir esse lançamento.	
5b. Lançamento não é deletado no banco de dados do sistema e a mensagem de alerta fecha.	
6a Exibe alguma mensagem de exceção levantada pelo sistema naquele momento. Logo, o lançamento não é deletado no banco de dados.	

UC016 – Emitir Relatório de Lançamentos

Identificação: UC016
Nome: emitir relatório de lançamentos
Atores: usuário, usuário administrativo, usuário vip
Tipo: primário
Pré-condições: estar logado no sistema e já possuir lançamento cadastrado no sistema
Pós-condições: Enviar o extrato de lançamentos em formato PDF para a máquina do usuário

Sequencia típica de eventos	
Ator	Sistema
1. Clica no botão “Lançamento” na barra de menu principal	2. Envia a página de lançamentos para o usuário. Essa página apresenta os lançamentos realizados pelo usuário e algumas opções sobre cada lançamento.
3. No campo paramêtros do relatório o usuário seleciona a data inicial e a data final desejada	5. Valida as informações preenchidas pelo usuário
4. Clica no botão imprimir	6. Envia um arquivo extrato de lançamentos para a máquina do usuário
Sequencia alternativa	
5a. Reenvia o formulário de lançamento para o usuário visualizar novamente	

UC017 – Cadastrar Cheque

Identificação: UC017	
Nome: cadastrar cheque	
Atores: usuário, usuário administrativo, usuário vip	
Tipo: primário	
Pré-condições: estar logado na aplicação financeira pessoal para web	
Pós-condições: cheque cadastrado na base de dados da aplicação	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no botão “Cheque” na barra de menu principal	2. Envia para o usuário a página de cheque com um formulário para o usuário cadastrar um novo talão de cheque
3. Preenche as informações do formulário de Cheque (número incial do cheque e número final do cheque)	5. Valida informações preenchidas pelo usuário
4. Clica no botão “Salvar”	6. Realiza o cadastro dos cheques

	informado pelo usuário
	7. Atualiza a tabela de cheques cadastrados para o usuário
Sequencia alternativa	
5a. Exibe mensagem “Erro de validação, informa os campos obrigatórios”.	

UC018 – Excluir cheque

Identificação: UC018	
Nome: excluir cheque	
Atores: usuário, usuário administrativo, usuário vip	
Tipo: secundário	
Pré-condições: estar logado na aplicação financeira pessoal para web e possuir cheque cadastrado	
Pós-condições: cheque removido da base de dados da aplicação	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no botão “Cheque” na barra de menu principal	2. Envia para o usuário a página de cheque com um formulário para o usuário cadastrar um novo talão de cheque. E uma lista com todos os cheques cadastrados.
3. Usuário clica no ícone lixeira na linha do cheque que deseja excluir	4. Deleta na base dado o cheque selecionado pelo usuário
	5. Atualiza a tabela de cheques cadastrados para o usuário
Sequencia alternativa	
3a. Exibe uma mensagem para o usuário informando que não é possível excluir um cheque baixado ou cancelado	
4a. Exibe alguma mensagem de exceção levantada pelo sistema naquele momento. Logo, o cheque não é deletada no sistema.	

UC019 – Cancelar cheque

Identificação: UC019	
Nome: cancelar cheque	
Atores: usuário, usuário administrativo, usuário vip	
Tipo: secundário	
Pré-condições: estar logado na aplicação financeira pessoal para web e possuir cheque cadastrado	
Pós-condições: cheque cancelado na base de dados da aplicação	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no botão “Cheque” na barra de menu principal	2. Envia para o usuário a página com todos os cheques cadastrados no sistema. Essa página tem um formulário para cadastrar um novo talão de cheque e opção de excluir e cancelar um determinado cheque.
3. Clica no ícone cancelar cheque na linha do cheque que deseja cancelar	4. Cancela na base dado o cheque selecionado pelo usuário
	5. Atualiza a tabela de cheques cadastrados para o usuário
Sequencia alternativa	
4a. Exibe alguma mensagem de exceção levantada pelo sistema naquele momento. Logo, o cheque não é cancelado no sistema.	

UC020 – Gerenciar Usuários

Identificação: UC020
Nome: gerenciar usuário
Atores: usuário administrador
Tipo: primário

Pré-condições: estar cadastrado como usuário administrador e logado no sistema	
Pós-condições: acessar à área administrativa do sistema	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no ícone “Administrativo” na barra de menu principal	2. Envia para o usuário à página administrativa do sistema, essa página contém uma lista com todos os usuários cadastrados no sistema.
3. Visualiza uma lista com todos os usuários cadastrados no sistema. Com opção de gerenciar cada usuário individualmente.	
Sequencia alternativa	
3a. Visualiza uma lista de usuários vazia, caso não exista usuários cadastrados no sistema.	

UC021 – Ativar Usuário

Identificação: UC021	
Nome: ativar Usuário	
Atores: usuário administrador	
Tipo: Primário	
Pré-condições: estar cadastrado como usuário administrador e logado no sistema	
Pós-condições: ativar o usuário no banco de dados do sistema	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no ícone “Administrativo” na barra de menu principal	2. Envia para o usuário à página administrativa do sistema, essa página contém uma lista com todos os usuários cadastrados no sistema.
3. Clica no ícone “desativado” sobre a linha do usuário que deseja ativar.	4. Alterar a situação do usuário no banco de dados
	5. Atualiza o ícone do usuário para “ativado”, na lista de usuário do sistema

Sequencia alternativa
4a. Apresenta para o usuário uma mensagem que ocorreu algum erro inesperado. Neste caso o usuário não foi ativado no banco de dados do sistema.
5.a Apresenta para o usuário mensagem que não foi possível encontrar ícone. Neste caso, o sistema desativa o usuário no banco de dados e exibe o ícone na linha do usuário na tabela.

UC022 – Desativar Usuário

Identificação: UC022	
Nome: desativar usuário	
Atores: usuário administrador	
Tipo: primário	
Pré-condições: estar cadastrado como usuário administrador e logado no sistema	
Pós-condições: desativa o usuário no banco de dados do sistema	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no ícone “Administrativo” na barra de menu principal	2. Envia para o usuário à página administrativa do sistema, essa página contém uma lista com todos os usuários cadastrados no sistema.
3. Clica no ícone “ativado” sobre a linha do usuário que deseja desativar.	4. Altera a situação do usuário no banco de dados
	5. Atualiza o ícone do usuário para “desativado”, na lista de usuário do sistema
Sequencia alternativa	
4a. Apresenta para o usuário uma mensagem que ocorreu algum erro inesperado. Neste caso o usuário não foi desativado no banco de dados do sistema.	

UC023 – Editar Usuário

Identificação: UC023

Nome: editar usuário	
Atores: usuário administrador	
Tipo: primário	
Pré-condições: estar cadastrado como usuário administrador e logado no sistema	
Pós-condições: dados de usuário atualizado no banco de dados do sistema	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no ícone “Administrativo” na barra de menu principal	2. Envia para o usuário à página administrativa do sistema, essa página contém uma lista com todos os usuários cadastrados no sistema.
3. Clica no ícone “editar” sobre a linha do usuário que deseja editar.	4. Envia para o usuário um formulário com os dados do usuário solicitado
5. Atualiza os dados do usuário que deseja alterar	7. Valida os dados informados pelos usuário administrador
6. Clica no botão salvar	8. Efetiva o update dos dados do usuário no banco de dados.
	9. Atualiza a tabela de usuários cadastrados no sistema
Sequencia alternativa	
7a. Exibe mensagem “Erro de validação, informa os campos obrigatórios”.	
8a. Apresenta para o usuário uma mensagem que ocorreu algum erro inesperado. Neste caso o usuário não foi alterado no banco de dados do sistema.	

UC024 – Excluir Usuário

Identificação: UC024
Nome: excluir usuário
Atores: usuário administrador

Tipo: Primário	
Pré-condições: estar cadastrado como usuário administrador e logado no sistema	
Pós-condições: usuário deletado no banco de dados do sistema	
Sequencia típica de eventos	
Ator	Sistema
1. Clica no ícone “Administrativo” na barra de menu principal	2. Envia para o usuário à página administrativa do sistema, essa página contém uma lista com todos os usuários cadastrados no sistema.
3. Clica no ícone “lixeira” sobre a linha do usuário que deseja excluir.	4. Envia para o usuário a mensagem: Confirma a exclusão do usuário selecionado?
5. Clica no botão ok, da mensagem apresentada.	6. Efetiva o delete dos do usuário no banco de dados.
	7. Atualiza a tabela de usuários cadastrados no sistema
Sequencia alternativa	
5a. Clica no botão cancelar, da mensagem apresentada.	
6a. Fecha a mensagem exibida para o usuário, e não exclui o usuário do sistema	

UC025 – Conceder Permissões

Identificação: UC025
Nome: conceder permissões
Atores: usuário administrador
Tipo: primário
Pré-condições: estar cadastrado como usuário administrador e logado no sistema
Pós-condições: conceder permissão para o usuário no sistema
Sequencia típica de eventos

Ator	Sistema
1. Clica no ícone  “Administrativo” na barra de menu principal	2. Envia para o usuário à página administrativa do sistema, essa página contém uma lista com todos os usuários cadastrados no sistema.
3. Clica no ícone  “Usuário Administrador” sobre a linha do usuário que deseja conceder permissão de administrador. Ou clique no ícone  “Usuário Vip” para conceder permissão de usuário vip.	4. Altera a permissão do usuário no banco de dados conforme seu privilegio concedido.
	5. Atualiza o ícone do usuário para  de “Administrador” ou  de “Usuário Vip” na lista de usuário do sistema
Sequencia alternativa	
4a. Apresenta para o usuário uma mensagem que ocorreu algum erro inesperado. Neste caso o usuário não recebeu as permissões no concedidas.	