UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

FELIPE FERREIRA CAMPOS

Sistemas Web Single Page Application com Java e AngularJS

Goiânia
2016

UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

AUTORIZAÇÃO PARA PUBLICAÇÃO DE TRABALHO DE CONCLUSÃO DE CURSO EM FORMATO ELETRÔNICO

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos "reproduzir" e "publicar" conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: Sistemas Web Single Page Application com Java e AngularJS

Autor(a): Felipe Ferreira Campos

Goiânia, 25 de Fevereiro de 2016.

Felipe Ferreira Campos – Autor

Walison Cavalcanti Moreira – Orientador

FELIPE FERREIRA CAMPOS

Sistemas Web Single Page Application com Java e AngularJS

Trabalho de Conclusão apresentado à Coordenação do Curso de Sistemas de Informação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Área de concentração: Desenvolvimento Web.

Orientador: Prof. Walison Cavalcanti Moreira

Goiânia
2016

FELIPE FERREIRA CAMPOS

Sistemas Web Single Page Application com Java e AngularJS

Trabalho de Conclusão apresentado à Coordenação do Curso de Sistemas de Informação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação, aprovada em 25 de Fevereiro de 2016, pela Banca Examinadora constituída pelos professores:

Prof. Walison Cavalcanti Moreira
Instituto de Informática – UFG
Presidente da Banca

Prof. Leandro Luís Galdino de Oliveira
Instituto de Informática – UFG

Felipe Ferreira Campos

Graduou–se em Sistemas de Informação na UFG - Universidade Federal de Goiás. Durante sua graduação foi monitor de Banco de Dados I, estágiario e CLT na CMS/Scitech Produtos Médicos no departamento de TI. Foi estágiario na Directa Soluções de Impressão no departamento de Soluções e no Ministério Público do Estado de Goiás na Divisão de Assistência e Suporte de TI. Atualmente trabalha na TSI Tecnologia, onde ingressou na area de Desenvolvimento Web.

Dedicado às três mulheres mais importantes da minha vida. Minha avó que sempre rezou pelos seus netos amados. Minha mãe, que me criou sozinha através de muita luta e dedicação. E minha namorada, que me cuida bem demais.

Agradecimentos

Agradeço a todos os professores por me proporcionar o conhecimento não apenas racional, mas a manifestação do caráter e afetividade da educação no processo de formação profissional, por tanto que se dedicaram a mim, não somente por terem me ensinado, mas por terem me feito aprender. A palavra mestre, nunca fará justiça aos professores dedicados aos quais sem nominar terão os meus eternos agradecimentos.

"O conhecimento serve para encantar as pessoas, não para humilhá-las."

Mario Sergio Cortella,

.

Resumo

Campos Ferreira, Felipe. **Sistemas Web Single Page Application com Java e AngularJS**. Goiânia, 2016. 34p. Relatório de Graduação. Curso de Sistemas de Informação, Instituto de Informática, Universidade Federal de Goiás.

As aplicações web e as tecnologias utilizadas para o seu desenvolvimento estão em constante evolução. Atualmente diversos métodos e ferramentas são utilizados para produzir estruturas bem elaboradas, que organizam e agilizam os processos necessários para a construção de grandes sistemas. Este trabalho, não só aborda as técnicas de programação Single Page Application, como também expõe grandes frameworks que facilitam o desenvolvimento, os testes e a manutenção dos sistemas que utilizam essa técnica. Para exemplificação foi projetada e desenvolvida uma aplicação simples que utiliza todos os conceitos abordados. O AngularJS interage com o HTML e através de marcações faz sincronia com o JavaScript, implementando modularização e suporte a rotas, além de diversas outras funcionalidades. A tecnologia Java compõe parte fundamental da aplicação desenvolvida, provendo serviços web RESTful para serem consumidos de forma assíncrona pela camada de visão do sistema, o Jersey é o encarregado de proporcionar essa abordagem. O SGBD utilizado é o MySQL, juntamente com o framework de persistência Hibernate para fazer o mapeamento objeto relacional. A solução obtida é rápida, escalável e demonstra conceitos de desenvolvimento web.

Palavras–chave

Single Page Application, Java, AngularJS, Serviços

Abstract

Campos Ferreira, Felipe. **Sistemas Web Single Page Application com Java**. Goiânia, 2016. 34p. Relatório de Graduação. Curso de Sistemas de Informação, Instituto de Informática, Universidade Federal de Goiás.

Web applications and technologies used for its development are constantly evolving. Currently several methods and tools are used to produce elaborate structures that organize and streamline the processes needed for building large systems. This work not only addresses the Single Page Application programming techniques, but also exposes major frameworks that facilitate the development, testing and maintenance of systems that use this technique. For exemplification was designed and developed a simple application that uses all the concepts discussed. The AngularJS interacts with HTML and through markings make sync with JavaScript, implementing modularization and support routes, and several other features. Java technology makes up a key part of the developed application, providing RESTful web services to be consumed asynchronously by system view layer, Jersey is in charge of providing this approach. The DBMS used is MySQL, along with the Hibernate persistence framework to make the object relational mapping. The resulting solution is fast, scalable and demonstrates web development concepts.

Keywords

Single Page Application, Java, AngularJS, Services

Sumário

Lista de Figuras

Lista de Códigos de Programas

Lista de Abreviaturas e Siglas

AJAX Asynchronous Javascript and XML

API Application Programming Interface

Back-End Conteúdo Processado no Servidor

DAO Data Access Object

Front-End Conteúdo Processado no Cliente

GSON *Google Script Object Notation*

HQL Hibernate Query Language

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

Interface Ligação/Conexão Física ou Lógica

JSON JavaScript Object Notation

MVC Model-View-Controller

Open Source Código Fonte Aberto

REST Representational State Transfer

SGBD Sistema de Gerenciamento de Banco de Dados

SPA Single Page Application

SQL Structured Query Language

URI Uniform Resource Identifier

URL Uniform Resource Locator

CAPÍTULO 1

Introdução

1.1 Contexto

Tanto Marcondes (2013) como Mesbah e Deursen (2006) e Petersson (2012) explicam sobre a evolução da internet e das aplicações web, tal evolução ocorre pois a medida em que os anos se passam vão surgindo novas tecnologias e novos navegadores mais padronizados e mais velozes, enquanto antigamente as páginas eram estáticas e as aplicações web sofriam de má interatividade e capacidade de resposta em relação aos usuários finais. A Interação em aplicações web clássicas é baseada em um modelo de interface multi-página, em que para cada solicitação à interface é completamente atualizada pelo navegador. Por muitos anos, os serviços web tradicionais vem sendo construídos em cima de sistemas de *Back-End* complexos que servem arquivos HTML para os usuários, que são renderizados estiticamente no navegador. Quando a era da Web 2.0 chegou, técnicas de carregamento de conteúdo dinâmico, como o AJAX tornaram-se populares. Essas técnicas permitem que a aplicação busque os dados do servidor, sem recarregar a página inteira. No entanto, a maior parte da página é ainda processada inicialmente no lado do servidor. Nesse novo modelo de desenvolvimento web Single Page Application, que também será trabalhado nesta monografia, a interface web de uma única página é composta por componentes individuais que podem ser atualizados ou substituídos de forma independente, de modo que a página inteira não necessite ser recarregada em cada ação do usuário, fazendo com que a experiência do usuário se pareça mais com a utilização de uma aplicação desktop do que com a utilização de uma aplicação web. Este modelo, por sua vez, ajuda a aumentar os níveis de interatividade, capacidade de resposta e a satisfação do usuário.

1.2 Descrição do Problema

Mikowski e Powell (2014) apontam que uma das razões pelas quais as aplicações web tradicionais são lentas é porque os servidores populares com estruturas MVC estão

focados em disponibilizar páginas de conteúdo que não interagem com o cliente. Um exemplo dado é de quando clica-se em um link em um site tradicional de slides, a tela pisca branca e todos os componentes da página são novamente recarregados: a navegação, anúncios, manchetes, texto e rodapé são todos reprocessados. No entanto, a única coisa que muda de fato é a imagem do slide e, talvez, um texto de descrição. Pior, não há nenhum indicador de quando algum elemento da página se torna funcional. As vezes alguns desses elementos podem ser acessados assim que aparecem na página web; outras vezes temos que esperar até o redesenho ser totalmente completado, além de uma espera de mais alguns segundos. Esta experiência lenta, inconsistente, e desajeitada está se tornando inaceitável para um consumidor cada vez mais sofisticado na web.

Além dos problemas citados, relacionados à lentidão de aplicações multi-página e a forma de interação utilizada nesse modelo, este trabalho também utiliza formas já conceituadas para resolver problemas de estruturação de código e organização de arquivos. Adiante na etapa prática de construção da aplicação é mostrada a integração da tecnologia AngularJS com Java, utilizando o Jersey Framework para o processo, e a integração do Java com o MySQL, utilizando Hibernate para fazer o mapeamento objeto relacional, a aplicação desenvolvida auxilia no entendimento sobre os problemas de integração e manutenção de uma aplicação web Single Page Application.

1.3 Objetivo

Este trabalho visa aprofundar os conhecimentos de SPA com a utilização dos frameworks AngularJS, Jersey e Hibernate, com a tecnologia Java e o SGBD MySQL, essencialmente mostrando o desenvolvimento de uma aplicação simples e funcional para gerenciamento de gastos. Foram dados os requisitos e o projeto da aplicação foi feito e exemplificado passo a passo para o entendimento de todo o processo de desenvolvimento. Com as tecnologias serão mostradas suas interações com o Java, bem como o consumo dos serviços oferecidos para o transporte de dados, desde de a camada de visão até a camada de banco de dados. Para tais objetivos são primordiais o entendimento das tecnologias e das técnicas aplicadas no desenvolvimento, este trabalho abordará também os conceitos fundamentais que precisam ser entendidos antes da análise e desenvolvimento do sistema web. Todas as camadas da aplicação foram desenvolvidas para explorar ao máximo a experiência de um usuário real com uma interface SPA.

1.4 Escopo e Metodologia

O trabalho tem o foco principal em mostrar a técnica de SPA, que têm uma ligação clara ao desenvolvimento *Front-End*, e como os sistemas com essa estrutura in-

teragem com a tecnologia Java utilizando uma arquitetura REST e o padrão MVC para o AngularJS. Uma vez que o este trabalho é focado no projeto, na análise e no desenvolvimento do sistema, os aspectos relacionados com a infraestrutura não foram incluídos dentro do âmbito. O desenvolvimento do *Front-End* com AngularJS foi limitado ao *Back-End* utilizando Java, as tecnologias Jersey, Hibernate e MySQL foram utilizadas apenas para produzir um software totalmente funcional, porém os detalhes de funcionamento dessas tecnologias não serão abordados detalhadamente neste trabalho, a utilização de outras linguagens no *Back-End* podem não possuir os mesmos recursos para integração com as ferramentas utilizadas nos exemplos, porém devem ser capazes de lidar com o *Front-End* contanto que tenham uma interface comum para a comunicação. Ao desenvolver a estrutura, o suporte para navegadores mais antigos no mercado não foi levado em consideração.

1.5 Estrutura do Trabalho

Neste trabalho, primeiramente, foi feito um apanhado teórico envolvendo os conceitos básicos necessários para a contextualização do projeto. No Capítulo 2 tem-se uma fundamentação teórica, conceitualizando as tecnologias principais que foram utilizadas com suas respectivas diretrizes, explicando um pouco sobre cada uma delas. No Capítulo 3 são explicados os conceitos das tecnologias Java e MySQL, os requisitos funcionais do sistema, o modelo de arquitetura baseada em serviços e os diagramas utilizados para o desenvolvimento do sistema. No Capítulo 4 são demonstrados trechos da implementação das tecnologias utilizadas para o desenvolvimento deste trabalho. Por fim, no Capítulo 5, tem-se a conclusão do que foi abordado, bem como as contribuições dos métodos e das tecnologias utilizadas para o desenvolvimento web e os trabalhos futuros.

CAPÍTULO 2

Single-Page Application

2.1 Descrição

Segundo Branas (2015) "uma Single Page Application é uma aplicação que realiza suas transições dentro de uma mesma página, carregando seus componentes de forma dinâmica utilizando AJAX". Muitos desenvolvedores têm sua própria visão do que significa exatamente, Mesbah e Deursen (2006, p.1, tradução nossa) afirmaram outra definição que também é bastante clara.

> A interface Web de uma Single Page é composta por componentes individuais que podem ser atualizados/substituídos independentemente, de modo a que a página inteira não necessite ser recarregada em cada ação do usuário. Este, por sua vez, ajuda a aumentar os níveis de interatividade, agilidade e satisfação do usuário.

Para Petersson (2012) essa definição possui algumas propriedades que ajudam a definir uma SPA. A primeira delas é a interface web, que possui foco no usuário e na sua interação com a página web. A segunda propriedade envolve os componentes individuais, que são divididos em componentes menores que interagem uns com os outros. A terceira é a atualização e substituição dos elementos da página, a medida em que ocorre interação, os componentes da interface podem ser substituídos por outros. Por fim estão as propriedades de recarregamento e ações do usuário, onde fica explícito que a página inteira jamais deve ser recarregada, mesmo que novos conteúdos precisem ser inseridos ou alocados em alguma seção da página web, e que o SPA é sempre responsável por tratar todas as ações dos usuários, independentemente dos dispositivos de entrada e saída que estão sendo utilizados no processo.

> Ao executar uma SPA Javascript, o navegador primeiro faz uma solicitação para o servidor web. O servidor web vai então responder com o cliente Javascript incluindo os recursos necessários para executá-lo. Uma vez que a interface do cliente é transferida ele será inicializada e estará pronta para ser executada. Quando o usuário interage com o cliente, como, por exemplo, clicando em elementos gráficos, isso pode requerer novos dados que precisam ser obtidos a partir do servidor. Em vez de

> recarregar a página inteira, o cliente faz um pequeno pedido de requisição para a API no servidor web. A API é simplesmente uma interface que permite aos clientes se comunicarem com o servidor em que um formato de dados é fácil de compreender, tanto para o cliente quanto para o servidor. (LUCCA et al., 2012 apud PETERSSON; PETERSSON, p.13, 2012, tradução nossa).

2.2 HTML e JavaScript

A linguagem de marcação HTML tem por objetivo construir páginas web, que podem ser renderizadas através de navegadores. Ela possui diversas funcionalidades, dentre elas, a integração de textos, imagens, áudios e links. Um arquivo HTML é constituído por textos e estruturado com elementos chamados de *tag*, estes podem conter diversos tipos de atributos, onde é possível especificar os parâmetros que se deseja definir. (Cavalcante, 2004)

O JavaScript é uma linguagem de programação que executa no lado do cliente, interpretada através dos navegadores, é capaz de interagir com os documentos em HTML, acessando campos, valores, realizando cálculos e validação de dados, além de muitas outras funcionalidades. Outra característica importante é que pode ser utilizada em conjunto com diversas linguagens de programação. Para facilitar o reúso, a manutenibilidade e a entendibilidade dos códigos em JavaScript existem padrões que podem ser utilizados, o desenvolvimento em camadas, por exemplo, é um conceito importante que visa separar a estrutura dos conteúdos HTML, das camadas de apresentação e comportamento constituídas respectivamente pelas folhas de estilos e pelos *scripts* desenvolvidos com JavaScript. (Silva, 2010)

A utilização de SPA em conjunto com essas duas tecnologias trouxe para os desenvolvedores Front-End um novo paradigma de desenvolvimento. Onde o lado cliente da aplicação possui muito mais lógica de apresentação para otimizar seu comportamento nos navegadores.

2.3 Frameworks

Para o processo de construção da aplicação, utilizamos estruturas, implementações e documentações de frameworks já existentes, com o objetivo de fornecer mecanismos altamente eficazes de reutilização de software dentro do domínio da aplicação. (BUTLER, s. d.).

“Framework é um conjunto de classes abstratas e concretas que fornece uma infraestrutura genérica de soluções para um conjunto de problemas” (JOHNSON e FO-

OTE, 1998 apud RÉ; RÉ, p.1, 2002). “Essas classes podem fazer parte de uma biblioteca de classes ou podem ser específicas de um certo domínio de aplicação. Frameworks possibilitam reutilizar não somente componentes isolados, mas também toda a arquitetura de um domínio específico” (RÉ, p.1, 2002).

2.4 O AngularJS

O AngularJS é um framework JavaScript para o desenvolvimento de aplicações modernas na web, com ele é possível desenvolver aplicações interativas que utilizam a metodologia de SPA e prover alta produtividade na experiência de desenvolvimento web. Ele se integra com o HTML para estender funcionalidades na camada de apresentação e insere JavaScript onde for necessário, construindo de forma declarativa, interfaces com o usuário, usando modelos e diretivas e trazendo recursos poderosos como *data binding* e injeção de dependência. Como resultado tem-se aplicações com componentes expressivos, reusáveis e manuteníveis. (Dan Menard, 2013)

A riqueza de interatividade desse framework possibilita também a sua utilização em aplicações móveis baseadas em tecnologia web. Como PhoneGap, que permite escrever aplicações nativas para Android, iOS e outras plataformas móveis como se fossem aplicativos web. Ao reutilizar código em vários dispositivos, é possível cobrir mais plataformas com menos código.

CAPÍTULO 3

Requisitos e Arquitetura do Sistema

3.1 Java e MySQL

A tecnologia Java foi escolhida por ser uma linguagem de programação considerada simples e de alto desempenho, que possui uma série de características como orientação a objetos e a utilização de múltiplas *threads*. Essa linguagem permite o desenvolvimento de programas independentemente do sistema operacional ou da arquitetura do hardware utilizada; e com isso priva os programadores das preocupações com a infraestrutura. (Mendes, 2005)

O programa MySQL foi escolhido para compor a aplicação, pois é um sistema de gerenciamento de banco de dados relacional SQL robusto, rápido, multitarefa e multiusuário, que pode ser usado como um produto *Open Source*. Um banco de dados é uma coleção de dados estruturados na qual é possível adicionar, acessar, e processar dados armazenados em tabelas separadas para proporcionar velocidade e flexibilidade. (MySQL AB, 2006)

3.2 Requisitos Funcionais

Para demonstrar a utilização das técnicas e do frameworks apresentados, foi desenvolvida uma aplicação financeira simples, onde é possível fazer cadastro e login de usuários, para que possam fazer o controle de seus gastos, e para iniciar o projeto foram levantados os seguintes requisitos:

[RF01] Na etapa de login, o sistema deverá receber o email e a senha do usuário, esses dados deverão ser válidos e terem sido registrados previamente no [RF02].
Prioridade: Alta
Responsável: Stakeholders.

[RF02] Na etapa de cadastro, o sistema deverá receber os seguintes dados do usuário: email, nome e senha. Só poderá ser registrado um email para cada usuário, esse sendo um identificador único de cada um deles.
Prioridade: Alta
Responsável: Stakeholders

[RF03] Na etapa principal, o sistema deverá permitir aos usuários visualizarem suas despesas.
Prioridade: Alta
Responsável: Stakeholders

[RF04] Na etapa principal, o sistema deverá permitir ao usuário ter um saldo corrente, todas as movimentações vão fazer cálculos utilizando o saldo corrente como base.
Prioridade: Alta
Responsável: Stakeholders

[RF05] Na etapa principal, o sistema deverá permitir ao usuário cadastrar suas despesas, estas devem ser subtraídas do saldo corrente[RF04].
Prioridade: Alta
Responsável: Stakeholders

[RF06] Na etapa principal, o sistema deverá permitir ao usuário remover suas despesas, estas devem ser somas ao saldo[RF04] correspondente.
Prioridade: Média
Responsável: Stakeholders

3.3 Aplicação em Camadas Utilizando Serviços

A solução baseada em serviços é composta de múltiplos serviços que se comunicam uns com os outros através de mensagens. Na aplicação os serviços foram agrupados em um pacote chamado *service*, e cada serviço estará internamente associado a um componente do software, esses componentes ficarão logicamente agrupados dentro das camadas de dados, negócios e apresentação. Essas camadas ajudam a separar os tipos de tarefas que serão executadas por cada componente, facilitando a criação de modelos para serem reutilizados e fornecendo regras e funcionalidades que ajudam a melhorar a manutenibilidade do código. A aplicação em camadas baseada em serviços será utilizada para que

sejam fornecidos serviços para as aplicações desenvolvidas na camada de apresentação com AngularJS. É uma forma eficaz de fornecer uma visão alternativa para permitir que os clientes utilizem diferentes canais para acessar a aplicação. A camada de domínio ou de negócios, chamada de *domain* na estrutura do projeto, contem as entidades principais e suas propriedades, que implementam as funcionalidades do núcleo do sistema, bem como a lógica de negócios da aplicação. A camada de dados, ou, de acesso aos dados, denominada *DAO* na estrutura do projeto, contem entidades responsáveis por prover acesso aos dados hospedados no SGBD. Essa camada é acessada através dos serviços e atua como uma interface de componentes que podem ser consumidos pela camada de negócios. Por fim temos a camada de apresentação, que é constituída pelas chamadas *views*, que são dedicadas às funcionalidades de gerenciamento da interação do usuário com o sistema, consistem de componentes que promovem uma interface entre o cliente e a lógica de negócios da aplicação. (Microsoft Patterns and Practices Team, 2009)

3.4 Java REST com Jersey

No estilo de arquitetura REST são especificadas restrições para serem aplicadas aos serviços web a fim de proporcionar atributos de desempenho, escalabilidade e modificabilidade, trazendo assim melhor funcionamento na web. Os recursos do REST são acessados tipicamente por links na web, e através da URI em conjunto com o protocolo HTTP é possível construir uma arquitetura cliente-servidor onde os clientes utilizam esses recursos por meio de uma interface padronizada. Uma implementação RESTful utiliza as especificações da arquitetura REST mapeando os métodos HTTP para as operações que serão executas de criação, recuperação, atualização e exclusão. O Jersey é uma implementação para facilitar a construção de servidores web RESTful com Java utilizando anotações que definem os recursos que podem ser utilizados, e foi adicionado ao projeto utilizando a ferramenta de gerenciamento de projeto Maven. (Sun Microsystems, 2009)

3.5 Diagramas do Sistema

O diagrama de atividades abaixo agrupa tarefas que formam uma unidade executável, cada uma delas é independente e pode ser programada individualmente. Elas se comunicam através de um conjunto de mecanismos bem definidos que podem ser serviços de comunicações baseados em mensagens síncronas e assíncronas, chamadas de procedimento remoto, transmissão de eventos, etc.(Kruchten, 1995)

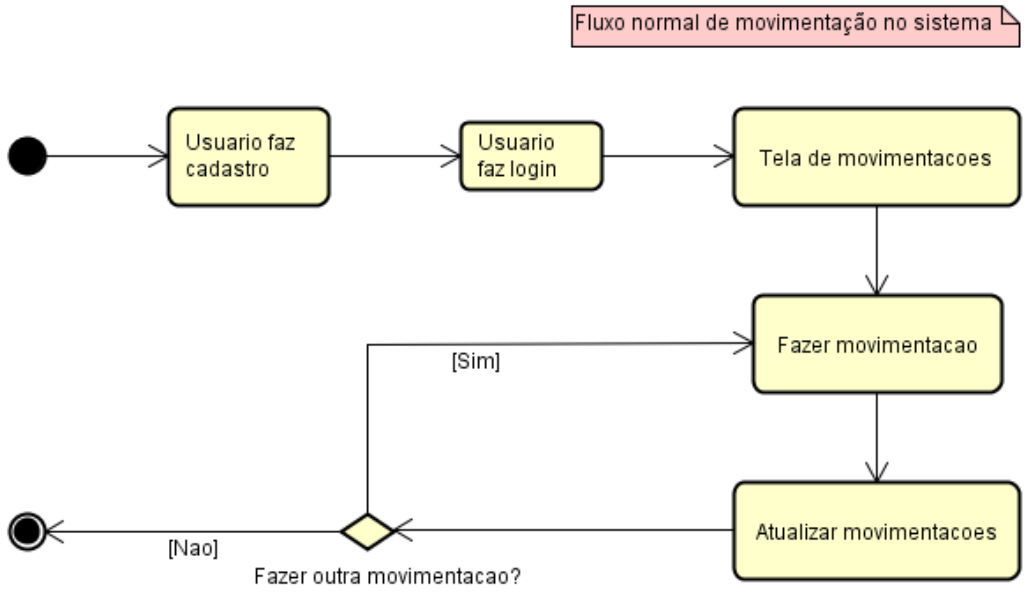

Figura 3.1: *Diagrama de Atividades*

O diagrama de casos de uso captura os cenários mais importantes e que são mais usados no sistema, mostrando as sequências de interações entre objetos e entre processos, no projeto pôde ser usado como ponto de partida para os testes de um protótipo de arquitetura.(Kruchten, 1995)

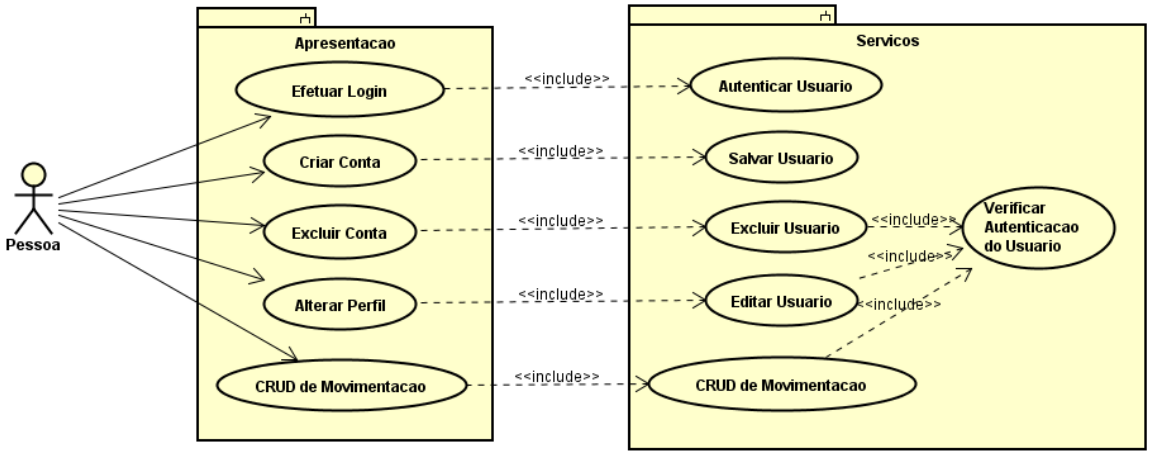

Figura 3.2: *Diagrama de Casos de Uso*

O diagrama de classes representa a arquitetura lógica do sistema que apoia principalmente os requisitos funcionais e por meio dele é possível fazer análises funcionais e identificar mecanismos e elementos comuns entre as várias partes do sistema. Cada classe representa uma abstração do domínio do problema e suas relações lógicas. (Kruchten, 1995)

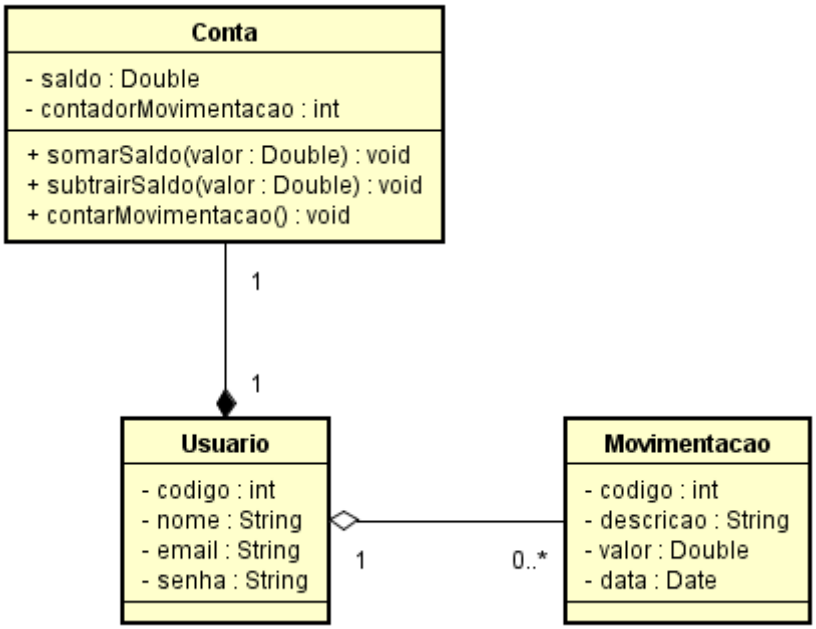

Figura 3.3: *Diagrama de Classes*

O modelo de entidade relacionamento descreve os dados como entidades, relacionamentos e atributos. É o projeto conceitual para o desenvolvimento do banco de dados que foi utilizado na aplicação, ele foi definido com base nos requisitos descritos anteriormente.(Elmasri, Navathe, 2005)

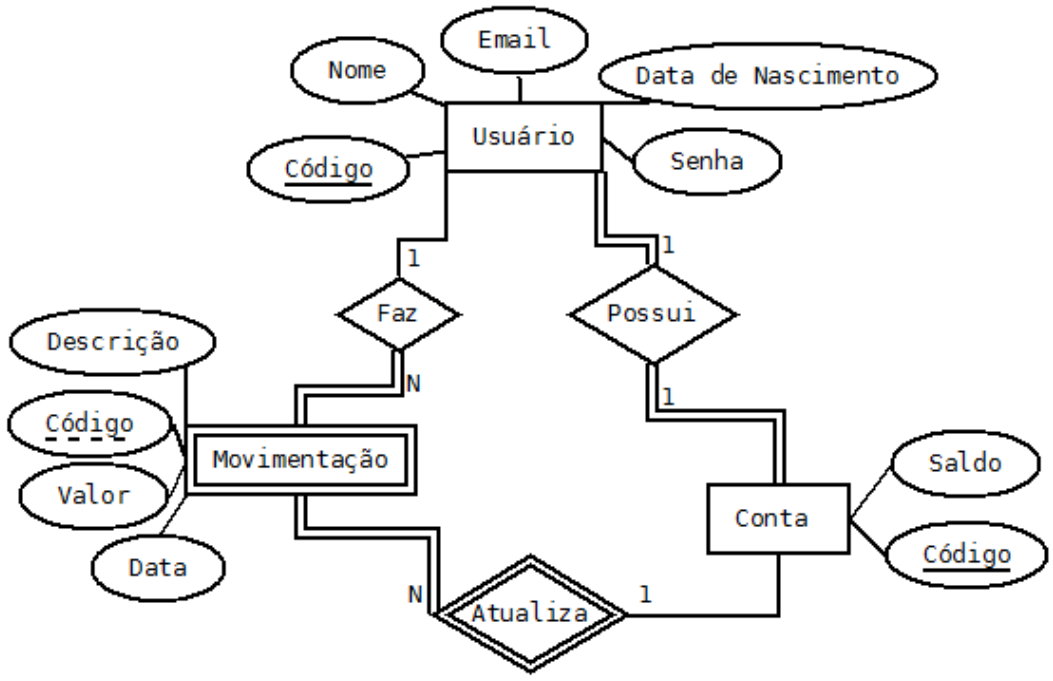

Figura 3.4: *Modelo Entidade Relacionamento*

CAPÍTULO 4

Implementação do Sistema

4.1 Implementação do Domínio com Hibernate

Abaixo foi criada uma classe cliente, nela foram colocadas anotações do Hibernate e consultas HQL, além dos atributos e os métodos necessários.

"A HQL é uma linguagem de consulta orientada a objetos, semelhante à SQL, mas em vez de operar em tabelas e colunas, HQL trabalha com objetos persistentes e suas propriedades. Consultas HQL são convertidas pelo Hibernate em consultas SQL convencionais que por sua vez realizam ação na base de dados." (Tutorials Point, 2016)

```
@Entity
@Table(name = "cliente")
@NamedQueries({
        @NamedQuery(name = "Cliente.listar",
                query = "SELECT cliente FROM Cliente cliente"),
        @NamedQuery(name = "Cliente.buscarPorCodigo",
                query = "SELECT cliente FROM Cliente cliente WHERE cliente.codigo = :codigo"),
        @NamedQuery(name = "Cliente.autenticar",
                query = "SELECT cliente FROM Cliente cliente "
                        + "WHERE cliente.login = :login AND cliente.senha = :senha")
})
public class Cliente {
```

Figura 4.1: *Dominio do Cliente*

O modelo, também conhecido como domínio do sistema, representa as regras de negócios que regem o acesso e atualização de dados, serve como uma associação do sistema a um processo do mundo real, as técnicas de modelagem mostradas no capítulo anterior foram uteis para definição dos atributos e métodos escolhidos para a classe de domínio abaixo. A separação da camada de modelo das outras camadas do sistema permite que vários pontos de vista possam usar o mesmo modelo de negócio, isso facilita a implementação e a manutenção, uma vez que todos os acessos feitos por outras camadas passam pelos componentes do modelo. (Sun Microsystems, 2002)

> Ao desenvolver um aplicativo para suportar um único tipo de cliente, às vezes é benéfico entrelaçar o acesso a dados e lógica das regras de negócio com a lógica específica de interface para apresentação e controle. Esta abordagem, no entanto, é inadequada quando aplicada a sistemas empresariais que necessitam suportar vários tipos de clientes. Diferentes aplicações precisam ser desenvolvidas, uma para apoiar cada tipo de interface do cliente. (Sun Microsystems, 2002, p1, tradução nossa)

```java
@Embeddable
public class Conta {
    @Column(nullable = false)
    private Double saldo;

    @Column(nullable = false)
    private int contadorMovimentacao;

    public void somarSaldo(Double valor) {
        this.saldo += valor;
    }

    public void subtrairSaldo(Double valor) {
        this.saldo -= valor;
    }

    public void contarMovimentacao() {
        this.contadorMovimentacao += 1;
    }
```

Figura 4.2: *Dominio da Conta*

4.2 Implementação do Objeto de Acesso aos Dados

O objeto de acesso aos dados, tem por objetivo abstrair a lógica necessária para acessar os armazenamentos de dados. Ela concentra a funcionalidade de acesso a dados para tornar a aplicação mais fácil de configurar e manter. Para o sistema foi utilizado o framework de mapeamento objeto relacional Hibernate, que implementa vários componentes automaticamente, reduzindo a quantidade de código necessário para as operações de acesso aos dados. As notações *@Entity* e *@Table* na figura 4.1, são usadas para gerar uma tabela que suporta o modelo de objeto e fornece um mapeamento entre as entidades de domínio e de banco de dados. Algumas das operações executadas na camada *DAO* do sistema desenvolvido são consideradas críticas, pois falhas na tentativa de salvar um cliente por exemplo, podem causar inconsistências nos dados gravados na base de dados, por esse motivo elas são envolvidas nas transações conforme mostra a figura 4.3. (Microsoft Patterns and Practices Team, 2009)

> As transações permitem que sejam executadas ações associadas a um banco de dados como uma unidade atômica, e garantem a integridade do banco de dados. A operação só é considerada completa se todas as informações e ações forem concluídas, e as mudanças no banco de dados associado são feitas permanentes. Transações apoiam desfazer ações (*Rollback*) de banco de dados em caso de um erro, o que ajuda a preservar a integridade dos dados no banco de dados. (Microsoft Patterns and Practices Team, 2009, p186, tradução nossa)

```
public void salvar(Cliente cliente) {
    Session session = HibernateUtil.getSessionFactory().openSession();
    Transaction transacao = null;

    try {
        transacao = session.beginTransaction();
        session.save(cliente);
        transacao.commit();
    } catch (RuntimeException ex) {
        if (transacao != null)
            transacao.rollback();
        throw ex;
    } finally {
        session.close();
    }
}
```

Figura 4.3: *DAO Salvar*

4.3 Implementação de Serviços REST com Jersey

Os serviços atuam como controladores, traduzindo interações com a camada de visão, em ações a serem executadas pelas camadas de domínio e de acesso aos dados. Interações do usuário na aplicação podem ser cliques em botões ou opções de menu, gerando solicitações HTTP dos tipos *GET* e *POST* por exemplo. As ações executadas pelos serviços incluem chamadas aos processos de negócio, bem como alterações de estado dos modelos. Com base nas interações do usuário, os controladores respondem com dados para a visão apropriada que os solicitou. (Sun Microsystems, 2002)

Abaixo é mostrado o serviço salvar, ele é invocado pela camada de visão através do método *POST*, e usa o GSON para converter o JSON em um objeto do tipo usuário, após isso instância uma nova conta e a inicializa, e, por fim, utiliza o objeto DAO para salvar o usuário recebido.

```
@POST
public int salvar(String json) {
    Gson gson = new Gson();
    Usuario usuario = gson.fromJson(json, Usuario.class);

    Conta conta = new Conta();
    conta.setContadorMovimentacao(0);
    conta.setSaldo(0.0);
    usuario.setConta(conta);

    UsuarioDAO usuarioDAO = new UsuarioDAO();
    try {
        usuarioDAO.salvar(usuario);
        return 1;
    } catch (Exception e) {
        return 0;
    }
}
```

Figura 4.4: *Service Usuário Salvar*

4.4 Implementação do Front-End com AngularJS

A camada de visão trabalha o conteúdo de um modelo. Ela acessa os dados da aplicação por meio de um serviço e específica como os dados devem ser apresentados. É responsabilidade da visão manter sua consistência e sua apresentação quando o modelo muda. (Sun Microsystems, 2002)

O AngularJS da suporte aos recursos de SPA por meio dos serviços de rotas, onde são mapeadas as URLs para associações entre *controllers* e *views*, permitindo também a passagem de parâmetros. (Branas, 2014)

No código abaixo o módulo foi inicializado com o nome Finanças e um vetor contendo as dependências utilizadas no sistema desenvolvido, são injetadas as dependências do módulo ngRoute para criamos o serviço de rotas conforme é mostrado no código 4.2, e o dirPagination que auxiliou no desenvolvimento da tabela de movimentações usada no sistema.

Código 4.1 `app.js`

```
1 angular.module("Financas", ['ngRoute', 'angularUtils.directives.dirPagination']);
```

Através da função *routeProvider*, é configurado o mecanismo de roteamento da aplicação usando a função *when*. Cada percurso do mapeamento configura um objeto, que pode ter as informações *controller*, *templateUrl*, *resolve* e *redirectTo*. Ao *controller* é atribuído o nome de um *controller* existente, por exemplo o *controller* mostrado no código 4.3, cujo o nome é cadastroCtrl. No *templateUrl* coloca-se o caminho do arquivo que será renderizado, no caso da aplicação é um arquivo HTML. Na informação *resolve* são colocadas as dependências que precisam ser resolvidas e injetadas dentro do *controller* caso necessário. No *redirectTo* é possível fazer os redirecionamentos desejados na aplicação. A função *otherwise* é usada para quando que se informa um caminho que não foi definido. (Branas, 2014)

Código 4.2 `routeConfig.js`

```
angular.module("Financas").config(function($routeProvider){
        $routeProvider.when("/movimentacoes", {
                templateUrl: "view/movimentacao.html",
                controller: "movimentacaoCtrl",
                resolve: {
                        movimentacoes: function($http){
                                return $http.get("/rest/movimentacao");
                        }
                }
        }).when("/login",{
                templateUrl: "view/login.html",
                controller: "loginCtrl"
        }).when("/cadastro",{
                templateUrl: "view/cadastro.html",
                controller: "cadastroCtrl"
        }).otherwise({
                redirectTo: "/login"
        });
});
```

O código abaixo mostra a criação de um *controller* chamado cadastroCtrl, ele recebe as dependências de *scope*, *http* e *location*. A função cadastrarUsuário que se inicia na linha 5 recebe um parâmetro chamado cadastro, que é um objeto montado na visão e passado ao *controller* através do *scope*, com esse objeto preenchido, ele invoca o serviço criado em java responsável por fazer os cadastros dos usuários, localizado no endereço http://localhost:8888/rest/usuario, na frente do endereço, após a vírgula, vai o objeto, e em caso de sucesso o serviço retorna um valor booleano verdadeiro, na linha 10, o *delete* é utilizado para apagar os dados do escopo do cadastro, a fim de limpar o formulário HTML e na linha 12 ele redireciona para a página de login. O *http.post* é uma requisição AJAX do tipo *POST*, o mesmo tipo de requisição que a camada de serviço espera receber.

> Uma aplicação com AngularJS baseia-se principalmente em controladores para controlar o fluxo de dados da aplicação. Um controlador é definido utilizando a diretiva *ng-controller*. Um controlador é um objeto JavaScript que contém atributos, propriedades e funções. Cada controlador aceita um escopo como parâmetro, que refere-se à aplicação ou módulo que o controlador precisa manusear. (Tutorials Point, 2014, p21, tradução nossa)

Código 4.3 `cadastroCtrl.js`

```
1  angular.module("Financas").controller("cadastroCtrl", function ($scope,
2                  $http, $location) {
3      $scope.titulo = "Cadastro";
4
5      $scope.cadastrarUsuario = function (cadastro) {
6          $http.post("http://localhost:8888/rest/usuario", cadastro)
7          .success(function (response) {
8              var response = parseInt(response);
9              if (response) {
10                 delete $scope.cadastro;
11                 alert("Cadastro realizado com sucesso");
12                 $location.path("/login");
13             } else {
14                 $scope.mensagem = "Cadastro Invalido";
15             }
16         }).error(function (response) {
17             alert("Erro " + response.status);
18         });
19     };
20 });
```

CAPÍTULO 5

Conclusão

5.1 Funcionamento do Sistema

O sistema utiliza rotas para implementar SPA, fazendo com que os módulos da aplicação sejam invocados a partir das interações do usuário com a página HTML, sem que seja preciso recarregar a página inteira. Quando o módulo precisa de dados contidos no SGBD, ele faz uma requisição AJAX para o serviço Java implementado com o Jersey framework, o serviço por sua vez, recebe os dados, instância um objeto Java do domínio para ser montado com os dados recebidos e os serve para a camada DAO, para que seja feita a operação solicitada, a camada DAO, através das configurações do Hibernate, comunica-se com o MySQL por meio de uma operação síncrona, e enquanto o SGBD processa a solicitação, o serviço Jersey aguarda a resposta do DAO, porém o AngularJS, na camada de visão, não fica preso esperando as operações serem finalizadas, ele permanece interagindo com o usuário e caso haja novas operações que necessitem da utilização do banco de dados, o AngularJS executa o AJAX normalmente, fazendo com que as novas solicitações entrem na fila dos envios assíncronos, quando os dados são retornados, o AngularJS os insere nos módulos da página, é muito importante implementar as interações do AngularJS com usuário, mesmo que as solicitações estejam na fila dos serviços assíncronos, senão o Front-End também fica travado aguardando as operações síncronas do Back-End serem finalizadas, uma outra abordagem para evitar esse problema é a utilização de um banco de dados assíncrono, porém esse tópico será abordado em trabalhos futuros.

5.2 Quando usar SPA e Trabalhos Futuros

Com a utilização de SPA, é possível proporcionar melhores experiências aos clientes web, com aplicações mais rápidas, escrevendo menos código do lado do servidor e mais código no lado do cliente, trazendo os conteúdos das páginas sob a demanda em que são solicitados e sem necessidade de recarrega-las. O desenvolvimento de aplicações

utilizando SPA exigem mais conhecimento em JavaScript e maior organização de código, se por um lado tem-se aplicações de funcionamento aparentemente mais nativo, por outro lado existem aspectos de segurança e de estruturação que precisam ser levados em consideração antes de se desenvolver aplicações SPA.

Com a utilização de JavaScript em alta e com frameworks cada vez mais poderosos, é interessante também apostar na tecnologia para o desenvolvimento do Back-End das aplicações, os próximos trabalhos abordarão o framework nodeJS e suas vantagens para determinados tipos de aplicação. A tecnologia GWT também será uma alternativa para trabalhos futuros, mostrando mais da maturidade da tecnologia Java inclusive nas camadas de Front-End dos sistemas.

5.3 Contribuição da Metodologia e das Tecnologias para a Área de Desenvolvimento WEB

As tecnologias apresentadas, atualmente são utilizadas em larga escala para o desenvolvimento dos mais diversos tipos de aplicações. É possível encontrar diversos vídeos, tutoriais, artigos e livros a respeito dos assuntos, e todos os procedimentos apresentados possuem métodos e meios alternativos de serem desenvolvidos nas mais diversas tecnologias e com frameworks variados, trazendo resultados diferentes inclusive para o mesmo tipo de aplicação. Além do AngularJS, frameworks como Backbone, Ember e GWT também podem ser utilizados, cada um com suas peculiaridades. Mesmo no desenvolvimento de uma aplicação simples foi possível perceber o quanto também são importantes as etapas de projeto, através dos diagramas é mais fácil de ter uma visão geral do sistema e de seu funcionamento, além de ser uma forma mais amigável de se apresentar o sistema para terceiros.

Referências Bibliográficas

[1] MARCONDES, R. A. *Single Page Applications e outras maravilhas da web moderna.* Disponível em: <http://imasters.com.br/desenvolvimento/single-page-applications-e-outras-maravilhas-da-web-moderna/?trace=1519021197>. Acesso em 02 de Fevereiro de 2016.

[2] MESBAH A; DEURSEN V. A. *Migrating multi-page web applications to single-page AJAX interfaces.* 2. ed. Delft University of Technology: Software Engineering Research Group, p. 181-190, 2006.

[3] PETERSSON, J. *Designing and implementing an architecture for single-page applications in Javascript and HTML5.* Linköpings universitet SE-581 83, p. 9-21, 2012.

[4] MIKOWSKI S. M.; POWELL C. J. *Single Page Web Applications.* Shelter Island: Manning Publications Co., p. 1-20, 2014.

[5] BRANAS, R. *AngularJS - 14 - Single-Page Application com ngRoute.* Disponível em: <https://www.youtube.com/watch?v=AljlfNqZp_E&index=14&list=PLQCmSnNFVYnTD5p2fR4EXmtlR6jQJMbPb>. Acesso em 02 de Fevereiro de 2016.

[6] CAVALCANTE, A. *Manual de Programação.* Disponível em: <www.andrecavalcante.com>. Acesso em 02 de Fevereiro de 2016.

[7] SILVA S. M. *JavaScript Guia do Programador.* São Paulo: Novatec Editora Ltda., p. 23-27, 2010.

[8] BUTLER G. *Object-Oriented Frameworks.* Concordia University: Department of Computer Science., p. 1, (s. d.).

[9] RÉ R. *Um Processo para Construção de Frameworks a partir da Engenharia Reversa de Sistemas de Informação Baseados na Web : Aplicação ao Domínio dos Leilões Virtuais.* ICMC/USP: Instituto de Ciências Matemáticas e de Computação, p. 1-2, 2002.

[10] MENARD, D. *Instant AngularJS Starter.* BIRMINGHAM - MUMBAI: Packt Publishing Ltd, 2013. 51p.

[11] MENDES, R. D. *Programação Java com Ênfase em Orientação a Objetos.* NOVATEC, Capítulo 1, p. 16-28, 2005.

[12] MYSQL AB. *Manual de Referência do MySQL 4.1.* p. 1-9, 2006.

[13] MICROSOFT PATTERNS AND PRACTICES TEAM. *Microsoft Application Architecture Guide.* 2. ed. p. 53-133, p. 163-274, 2009.

[14] SUN MICROSYSTEMS. *RESTful Web Services Developer' s Guide.* Santa Clara: Network Circle, 2009. 56p.

[15] KRUCHTEN, P. *Architectural Blueprints—The "4+1" View Model of Software Architecture.* Paper published in IEEE Software 12, p. 42-50, 1995.

[16] ELMASRI R; NAVATHE B. S. *Sistemas de Banco de Dados.* 4. ed. USP-Ribeirão Preto: Pearson Education do Brasil Ltda, p. 36-55, 2005.

[17] TUTORIALS POINT. *Hibernate - Query Language.* Disponível em: <http://www.tutorialspoint.com/hibernate/hibernate_query_language.htm>. Acesso em 02 de Fevereiro de 2016.

[18] SUN MICROSYSTEMS. *Java BluePrints Model-View-Controller.* 2000-2002. 3p.

[19] BRANAS, R. *AngularJS Essentials.* BIRMINGHAM - MUMBAI: Packt Publishing Ltd, 2014. 164p.

[20] TUTORIALS POINT PVT. LTD. *Learn AngularJS Web Application Framework.* p.21-24, 2014.

O sistema completo desenvolvido nesse trabalho encontra-se disponível em: https://github.com/FelipeJS/financas