UNIVERSIDADE DO EXTREMO SUL CATARINENSE - UNESC

CURSO DE CIÊNCIA DA COMPUTAÇÃO

DAIANA CAMBRUZZI ÁVILA

UTILIZAÇÃO DO FRAMEWORK ANGULARJS ASSOCIANDO INTEGRAÇÕES COM SPRING MVC, HIBERNATE E BLUETOOTH 4.0 PARA A CONSTRUÇÃO DE UMA FERRAMENTA DE AGENDAMENTO E ENVIO DE MENSAGENS

CRICIUMA

2015

DAIANA CAMBRUZZI ÁVILA

UTILIZAÇÃO DO FRAMEWORK ANGULARJS ASSOCIANDO INTEGRAÇÕES COM SPRING MVC, HIBERNATE E BLUETOOTH 4.0 PARA A CONSTRUÇÃO DE UMA FERRAMENTA DE AGENDAMENTO E ENVIO DE MENSAGENS

Trabalho de Conclusão de Curso, apresentado para obtenção do grau de Bacharel, no curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC.

Orientador: Prof. MSc. Gustavo Bisognin

CRICIUMA

2015

DAIANA CAMBRUZZI ÁVILA

UTILIZAÇÃO DO FRAMEWORK ANGULARJS ASSOCIANDO INTEGRAÇÕES COM SPRING MVC, HIBERNATE E BLUETOOTH 4.0 PARA A CONSTRUÇÃO DE UMA FERRAMENTA DE AGENDAMENTO E ENVIO DE MENSAGENS

Trabalho de Conclusão de Curso aprovado pela Banca Examinadora para obtenção do Grau de Bacharel, no Curso de Ciência da Computação da Universidade do Extremo Sul Catarinense, UNESC, com Linha de Pesquisa em Desenvolvimento para Web.

Criciúma, 25 de junho de 2015.

BANCA EXAMINADORA

Prof. Gustavo Bisognin – Mestre – (UNESC) – Orientador

Prof. Fabrício Giordani – Especialista – (UNESC)

Prof. Gilberto Vieira da Silva – Especialista – (UNESC)

Dedico este trabalho a minha família e as pessoas que estiveram ao meu lado me apoiando ao longo desta caminhada, para que esse sonho se tornasse realidade.

AGRADECIMENTOS

À Deus por ter me dado saúde e forças para vencer os obstáculos e superar as dificuldades.

Agradeço a minha família, que sempre estiveram ao meu lado me apoiando e incentivando para sempre seguir em frente e para realizar os meus sonhos. Agradeço também ao meu namorado que teve paciência na minha ausência e compreensão do meu nervosismo.

Aos meus amigos que estiveram ao meu lado nessa caminhada, me incentivando e pelas palavras de encorajamento que foram essenciais neste percurso onde rimos, choramos e nos ajudamos mutuamente.

Ao meu professor orientador Gustavo Bisognin, que compartilhou seus conhecimentos e experiências, e que também sempre me motivou e acreditou no meu potencial.

Aos demais docentes do curso de Ciência da Computação, que ao decorrer destes quatro anos e meio, compartilharam seus conhecimentos da melhor forma possível, agregando muito valor na minha vida acadêmica e pessoal.

“O único lugar onde o sucesso vem antes do trabalho é no dicionário.”

Albert Einstein

RESUMO

Com o constante crescimento da internet e a indispensável utilização de serviços rápidos e instantâneos, há uma necessidade de demanda considerável para o desenvolvimento de aplicações que desfrute da Web como ambiente operacional, necessitando da utilização das melhores ferramentas para dispor das melhores qualidades que a aplicação na *web* pode oferecer ao usuário final. Além do crescimento das aplicações na *web,* há um crescimento no desenvolvimento de aplicativos móveis devido ao alto consumo de compras de dispositivos móveis como *smartphones* e *tablets*, e devido ao crescimento do setor é possível realizar implementações de aplicativo que traga facilidade e mobilidade aos usuários finais, assim como dispor de serviços de localização e transferências de dados para que haja uma conexão entre o usuário e a região. Devido a esse crescimento *web*, notou se a necessidade de realizar o estudo dos *frameworks* para sanar os problemas de desenvolvimento, tornando a implementação mais segura e rápida, para que traga resultados rápidos aos usuários, utilizando as melhores práticas de desenvolvimento para obter um melhor desempenho da aplicação. Com a utilização da sincronização da aplicação web com o aplicativo móvel, é possível agregar a facilidade ao usuário final, além de oferecer mobilidade para que através do Bluetooth seja possível detectar o dispositivo iBeacon, que é um protocolo padronizado pela Apple, que emite sinais a radio de curto alcance, e de acordo com o recebimento desses sinais, o aplicativo detecta esses sensores e é capaz de ativar comportamentos no aplicativo móvel, como receber informações pré-configuradas da aplicação *web*, fazendo com que haja comunicação com o usuário de acordo com a localização do dispositivo móvel.

Palavras-chave: Internet, Framework, Dispositivo móvel, Bluetooth, iBeacon.

ABSTRACT

With the steady growth of the internet and the indispensable use of rapid and instant services, there is a need for considerable demand for the development of applications that enjoy the Web as operating environment, requiring the use of the best tools to provide the best qualities that the application in web can offer the end user. In addition to the growth of web applications, there is a growth in the mobile application development due to the high consumption purchases of mobile devices like smartphones and tablets, and due to the growth of the sector can perform application implementations that bring ease and mobility to end users as well as providing location services and data transfers so there is a connection between the user and the region. Because of this web growth, noted the need for the study of frameworks to remedy the problems of development, making implementation more secure and quick, so bring quick results to users, using the best development practices for better performance application. By using the sync web application with the mobile application, you can add the facility to the end user, while offering mobility to via Bluetooth is possible to detect iBeacon device, which is a standard protocol for Apple, A Bleeping short-range radio, and according to receive these signals, the application detects these sensors and is able to activate behaviors in the mobile application, such as receiving preset information from the web application, so that there is communication with the user according to the location of the mobile device.

Keywords: Internet, Framework, Mobile Device, Bluetooth, iBeacon.

LISTA DE ILUSTRAÇÕES

Figura 1 - Arquitetura mínima do Hibernate. 16
Figura 2 - Arquitetura compreensiva do Hibernate 17
Figura 3 – Configuração do Hibernate 20
Figura 4 – Mapeamento com hibernate annotations 23
Figura 5 – Camadas de controle do Spring MVC 28
Figura 6 – Configuração do Spring MVC 29
Figura 7 – Declaração do DispatcherServlet e mapeamento 30
Figura 8 – Implementação do controller 31
Figura 9 – Utilização de *tags* 34
Figura 10 – Estrutura básica de um documento 36
Figura 11 – Estrutura básica de um seletor 37
Figura 12 – Sintaxe do seletor 38
Figura 13 – Utilização das diretivas: ng-app, ng-init e ng-model. 45
Figura 14 – Criação de diretivas. 45
Figura 15 – Utilização do *data binding* 46
Figura 16 – Utilização do *controller* 48
Figura 17 – Utilização do escopo 49
Figura 18 – Utilização do *factory* 50
Figura 19 – Utilização do escopo 50
Figura 20 – Arquitetura do Android 53
Figura 21 – Ciclo de vida da activity. 55
Figura 22 - Diagrama do processo metodológico 62
Figura 23 - Modelo lógico da aplicação web 63
Figura 24 - Estrutura de packages da aplicação 65
Figura 25 - Configuração do Spring MVC 66
Figura 26 - Configuração do pacote base da aplicação 66
Figura 27 - Configuração do banco de dados do Hibernate no Spring MVC 67
Figura 28 - Arquivo com as propriedades do JDBC 68
Figura 29 - Configurações do Hibernate 68
Figura 30 - Classe POJO cidade 69
Figura 31 - Requisições no Spring MVC 70
Figura 32 - Classe de serviço da filial 71

Figura 33 - Controller do Spring MVC 72
Figura 34 - Menu da aplicação web 73
Figura 35 - Arquivo de *controller* do JavaScript.....74
Figura 36 - Tela de listagem das empresas 74
Figura 37 - Arquivo JavaScript do controller da Empresa 75
Figura 38 - Tela inicial do aplicativo de agendamento. 76
Figura 39 – Função para ler os dados JSON. 77
Figura 40 – Definição das variáveis para definir a região do iBeacon. 78
Figura 41 – Função para conectar com o iBeacon. 78
Figura 42 - Mensagem de notificação 79
Figura 43 - Tela de inserção de mensagem ao usuário 80

LISTA DE TABELAS

Tabela 1 – Principais tipos de bancos de dados e dialetos para configurar no Hibernate.....24

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JPA	Java Persistence API
JSTL	JavaServer Pages Standard Tag Library
MOR	Mapeamento Objeto-Relacional
MVC	Model View Controller
MVW	Model View Whatever
ORM	Object-relational mapping
POJO	Plain Old Java Object
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
XML	eXtensible Markup Language

SUMÁRIO

1 INTRODUÇÃO 11
1.1 OBJETIVO GERAL 12
1.2 OBJETIVOS ESPECÍFICOS 12
1.3 JUSTIFICATIVA 12
1.4 ESTRUTURA DO TRABALHO 14
2 HIBERNATE 15
2.1 ARQUITETURA 15
2.2 CONFIGURAÇÕES 19
2.3 MAPEAMENTO 22
2.3.1 Hibernate annotations 22
3 SPRING WEB MVC 26
3.1 CONFIGURAR O SPRING MVC 28
3.2 DISPATCHERSERVLET 29
3.3 CONTROLLER 30
4 TECNOLOGIAS WEB 32
4.1 HTML 32
4.1.1 História 33
4.1.2 Tags 34
4.1.3 Estrutura básica de um documento 35
4.2 CSS 36
4.2.1 Seletores 38
4.3 JAVASCRIPT 39
4.3.1 TIPOS 40
4.3.2 OPERADORES 40
4.3.3 INTERAÇÕES 41
5 FRAMEWORK ANGULARJS 43
5.1 DIRETIVAS 44
5.2 DATA BINDING 46
5.3 INJEÇÃO DE DEPENDÊNCIAS 47
5.4 CONTROLLERS 47
5.5 SCOPE 48
5.6 ROTAS 49

5.7 FACTORY 49
5.8 SERVICE 50
6 DISPOSITIVOS MÓVEIS 52
6.1 PLATAFORMA ANDROID 52
6.1.1 Arquitetura 53
6.1.2 Blocos de construção 54
6.2 BLUETOOTH 56
6.2.1 Frequência 57
6.2.2 Versões da tecnologia 58
6.3 IBEACON 59
7 TRABALHOS CORRELATOS 60
7.1 ESTUDO DE TECNLOGIAS PARA O DESENVOLVIMENTO DE APLICAÇÃO WEB PARA GERENCIAMENTO DE CORRETORA DE SEGUROS 60
7.2 COMPARAÇÃO DE DESEMPENHO E CONSUMO DE MEMÓRIA ENTRE FRAMEWORKS DE MAPEAMENTO OBJETO-RELACIONAL JAVA HIBERNATE E ECLIPSELINK 60
7.3 SISTEMA DE CONCESSÃO DE FINANCIAMENTO PARA A APRESENTAÇÃO DE ARTIGOS ACADÊMICOS EM SPRING MVC 61
7.4 DESENVOLVIMENTO DE APLICAÇÕES GOOGLE ANDROID E INTEGRAÇÃO COM WEB SERVICE 61
8 DESENVOLVIMENTO DO SISTEMA 62
8.1 MODELAGEM DE DADOS 62
8.2 FERRAMENTAS DE DESENVOLVIMENTO 63
8.3 CONSTRUÇÃO DA FERRAMENTA WEB E MOBILE 64
8.3.1 Estrutura inicial do projeto da aplicação web 64
8.3.2 Configuração Spring MVC 66
8.3.3 Configuração do Spring MVC com o Hibernate para acesso ao banco de dados 67
8.3.4 Mapeamento dos objetos 68
8.3.5 Requisições no Spring MVC 70
8.3.6 Desenvolvimento do Front-End 72
8.3.6 Desenvolvimento Mobile 76
8.4 RESULTADOS OBTIDOS 80
9 CONCLUSÃO 82

REFERÊNCIAS..83

1 INTRODUÇÃO

Com o crescimento de criações de *sites* com diversos propósitos, é necessário utilizar tecnologias robustas que forneçam segurança e confiabilidade nas transações realizadas pela aplicação *web*, sendo assim identificamos os *frameworks* de desenvolvimento que, em sua essência, contemplam algumas características como rapidez, agilidade e flexibilidade de desenvolvimento.

O *framework* é um conjunto de classes que colaboram para realizar uma responsabilidade para um domínio de um subsistema da aplicação. Com o advento da geração de conteúdo na *web*, bem como com a crescente demanda dos sistemas rodando em nuvem, identifica-se uma essencialidade quanto a utilização dos frameworks no desenvolvimento de *sites*, sistemas e aplicativos nos mais diversos níveis de aplicabilidade, pois abstrai os códigos redundantes e provê funcionalidades genéricas que podem ser evoluídas no decorrer do desenvolvimento.

Um dos objetos de estudo deste trabalho, aborda o AngularJS, por ser um *framework* desenvolvido em *JavaScript*, que foi criado e mantido pela Google desde 2010, sendo que uma das suas principais características é a implementação baseada em diretivas nas *tags* agregando potencial no *HyperText Markup Language* (HTML). Outra constatação importante, é a associação do *framework* Spring MVC com o framework AngularJS, pois entendemos, que esta, é essencial para construir uma aplicação robusta e flexível, pois o Spring MVC nos fornece o gerenciamento de objetos e transações de uma forma simplificada, para que no desenvolvimento não exista a necessidade de tanta preocupação com a infraestrutura, principalmente no que refere-se a questões relacionadas ao sincronismo de transações.

A tecnologia *mobile* proporciona aos seus usuários comodidade, pois é capaz de atingir o cotidiano e fazer parte da vida das pessoas, devido a mudança de rotinas e hábitos, além de formas de tomar decisões, tornando se uma facilidade ao alcance rápido e instantâneo, pois fornece acesso aos dados e informações em qualquer lugar.

No contexto atual, identificamos um problema enfrentado por diversas pessoas que utilizam ferramentas de agendamento, as quais trabalham de forma síncrona e geralmente conectadas a internet. Diversos estabelecimentos comerciais trabalham com a política de agendamento. Dentre eles, podemos destacar consultórios médicos, odontológicos, clínicas, empresas de manutenção e

montagem, salões de beleza e entre outros.

O presente trabalho pretende abordar o estudo da integração de tecnologias de *frameworks web* para a implementação de um protótipo que visa à resolução do problema de agendamento, e visa sua aplicação em salões de beleza, além do estudo da tecnologia iBeacon utilizando um aplicativo móvel que detecta a região que o usuário está contido e a localização do Beacon, e fornece informações ao usuário através do *web service*, sendo que também há a possibilidade de realizar a sincronização dos agendamentos para que o usuário possa verificar os horários vagos e realizar o agendamento com o salão de beleza.

1.1 OBJETIVO GERAL

Compreender a utilização do *framework* AngularJS com as tecnologias Spring MVC, Hibernate e Bluetooth 4.0 através da implementação de um protótipo para agendamento e envio de mensagens utilizando a tecnologia iBeacon.

1.2 OBJETIVOS ESPECÍFICOS

Para o desenvolvimento deste projeto de pesquisa foram selecionados os seguintes objetivos:

a) estudar as tecnologias Hibernate e Spring MVC;
b) estudar o *framework* AngularJS;
c) disponibilizar um protótipo utilizando Integrações do Framework Angular JS, Spring MVC e Hibernate;
d) estudar a tecnologia Bluetooth 4.0;
e) implementar o protótipo *web* e *mobile*.

1.3 JUSTIFICATIVA

Observando o contexto de diversos serviços que necessitam de agendamento, identifica-se que existem várias possibilidades de agendas, cada uma com suas regras de validação e fluxo de navegação, as quais dependem do processo do negócio para serem funcionais, sendo que estas, geralmente funcionam para um nicho específico dentro de um domínio de negócio.

Assim sendo, uma solução que utilize os *frameworks* existentes e proponha uma integração inteligente entre eles torna-se mais viável em vários aspectos, pois permite o aproveitamento de regras de validações específicas e não exige um tempo suficientemente grande para adaptação aos sistemas existentes e novos que venham a surgir dentro do paradigma *web*.

O AngularJS é um *framework* que apresenta entre suas características, várias indicações favoráveis a sua utilização. Segundo Daniel Schmitz e Douglas Lira (2013), uma dessas particularidades é o seu funcionamento como uma extensão ao documento HTML, adicionando novos parâmetros e interagindo de forma dinâmica com vários elementos.

Com a utilização deste *framework*, pretende-se identificar uma possível agilidade e flexibilidade de implementação *web*, pois além de trabalhar com a arquitetura *Model-View-Controller* (MVC), também fornece um conjunto de praticidades como a utilização do d*atabinding*, *client-side templates* e injeção de dependências, sendo que com a utilização de ligação bidirecional de dados proporciona ao desenvolvedor uma redução de códigos para demonstrar os dados processados pelo servidor, inferindo maior produtividade e facilidade na manutenção do código fonte, sendo que na utilização das diretivas é possível modularizar e abstrair a complexidade.

A tecnologia Hibernate é desenvolvida para o uso na linguagem Java, que pretende-se agregar um elevado desempenho na persistência dos dados na base e serviços de consulta, pois a mesma utiliza o conceito de mapeamento objeto-relacional, que é considerado um dos conceitos que enfatiza o melhor aproveitamento da transformação entre objetos e linhas de tabelas, pois realiza a conversão destes dois pontos para que exista uma comunicação viável.

Com a evolução do estudo do framework AngularJS com as tecnologias Spring MVC e Hibernate, pretende-se viabilizar a implementação de um módulo de agendamento, a qual deve ser validada em um sistema de agendamento para salões de beleza que deve controlar os agendamentos.

1.4 ESTRUTURA DO TRABALHO

Este trabalho divide-se em capítulos de forma a estruturar e facilitar a leitura e compreensão, sendo que o primeiro capítulo é destinado à introdução, justificativa, os objetivos gerais e específicos e estrutura do trabalho.

O segundo capítulo aborda sobre o *framework* Hibernate, explicando desde a sua arquitetura, configuração para a utilização e formas de mapeamentos que podem ser utilizado no mapeamento objeto-relacional da aplicação.

No terceiro capítulo é apresentado o *framework* Spring MVC, detalhes e parâmetros para a configuração, exibindo as funções do *DispatcherServlet* que funciona como *Servlet* da aplicação e a criação do *Controller* que irá conter as requisições.

O quarto capítulo apresenta as tecnologias *web*¸ abordando os motivos da utilização das tecnologias HTML, CSS e JavaScript, dando ênfase no desenvolvimento da aplicação e explicando os conceitos mais utilizados na programação.

No quinto capitulo é abordado o *framework* AngularJS, assim como as principais utilidades e facilidades que podem ser empregadas no desenvolvimento.

O sexto capítulo apresenta a importância dos dispositivos móveis na vida dos usuários e a forma que agrega facilidade no cotidiano, explicando sobre a plataforma Android, sendo utilizado com a tecnologia Bluetooth 4.0 com o dispositivo Estimote Beacon.

Os trabalhos correlatos que possuem relação com este trabalho estão contidos no sétimo capítulo.

O desenvolvimento da aplicação web e do aplicativo estão contidos no oitavo capítulo, explicando a metodologia aplicada no desenvolvimento.

E por fim no oitavo capitulo consiste a apresentação da conclusão do trabalho realizado e sugestões para trabalhos futuros.

2 HIBERNATE

Com as modernas linguagens de programação, assim como o Java, o conceito de orientação a objeto está cada vez mais difundido, os dados são manipulados no formato de objetos, porém na maioria das vezes são persistidos em bancos de dados relacionais, e para tratar este cenário, existem as ferramentas de Mapeamento Objeto-Relacional (MOR), como o *framework* Hibernate.

O Hibernate é um *framework* que atua como uma ferramenta de persistência de dados, que tem como intuito de abstrair os códigos *Structured Query Language* (SQL) para mapeamento objeto/relacional em Java, funcionando como intermediário entre a aplicação e o banco de dados.

Com a utilização da tecnologia Hibernate desenvolvida para o uso na linguagem Java, pretende-se agregar um elevado desempenho na persistência dos dados na base e serviços de consulta, devido ao melhor aproveitamento da transformação entre objetos e linhas de tabelas, de acordo com a conversão destes dois pontos para que exista uma comunicação viável.

O *framework* pode ser utilizado com ou sem o *Java Persistence API* (JPA), devido à implementação das *interfaces* de programação e regras de ciclo de vida definidas na especificação 2.0 da JPA. Diversas características que estão contidas na JPA foram movidas pela inspiração no Hibernate, como por exemplo, o modelo de objetos, gerenciamento de entidades, linguagens de busca e entre outros (BAUER; KING, 2007, tradução nossa).

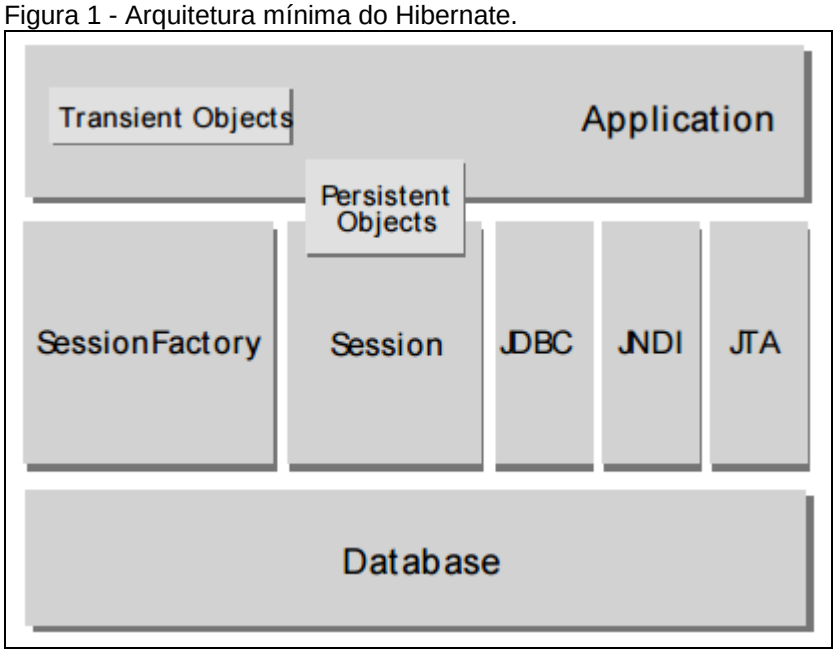
Uma das vantagens de utilizar o Hibernate é devido abstração do código SQL, uma forma da aplicação ficar independente das particularidades do banco de dados, ou seja, independência entre o modelo de objetos e o banco de dados, sendo este um recurso da camada persistência, fazendo parte da arquitetura do Hibernate.

2.1 ARQUITETURA

A arquitetura do Hibernate pode ser representada com uma arquitetura mínima e uma arquitetura compreensível, devido à flexibilidade e suporte para diversas abordagens (Documentação Hibernate). Conforme a figura 1, a arquitetura mínima fornece suas próprias conexões JDBC e gerencia suas transações. Esta

abordagem usa o mínimo de subconjuntos das Applications Programming Interface (API) do Hibernate.

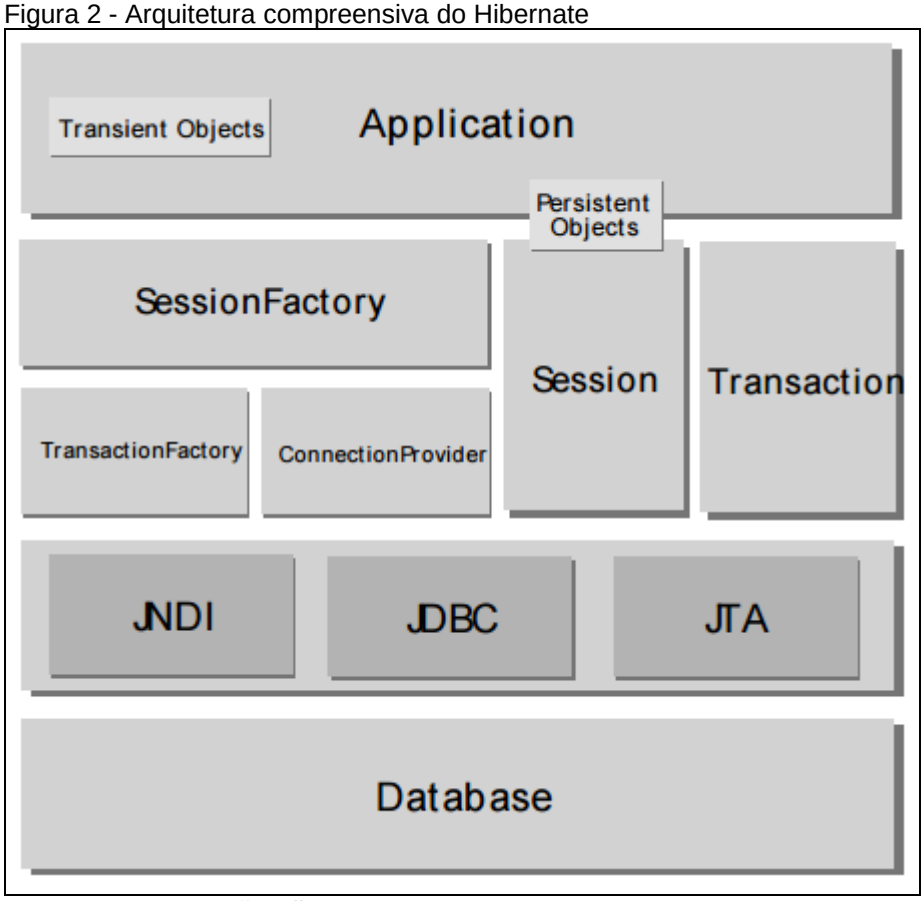
Figura 1 - Arquitetura mínima do Hibernate.



Fonte: Documentação Hibernate.

A arquitetura compreensiva abstrai a aplicação do JDBC/JTA e APIs adjacentes, fazendo com que o Hibernate tome conta dos detalhes (Documentação Hibernate).

Figura 2 - Arquitetura compreensiva do Hibernate



Fonte: Documentação Hibernate

Na arquitetura mínima e compreensiva, seguem as definições dos objetos descritos nas arquiteturas conforme a documentação do Hibernate:

a) *sessionFactory*: responsável por manter o mapeamento objeto relacional em memória. Um objeto *SessionFactory* é *threadsafe*, sendo que deve existir apenas uma instância na aplicação, para que não haja sobrecarga na aplicação;
b) *session*: possibilita a comunicação entre a aplicação e a persistência, através de uma conexão JDBC;
c) objetos persistentes e coleções: são objetos *single-threaded*, ou seja, de vida curta, contendo estado persistente e função de negócios,sendo os modelos de objetos;
d) objetos e coleções desanexados e transientes: instâncias de classes persistentes, que por enquanto não estão associadas a uma *session*;
e) *transaction*: utilizada para representar uma unidade indivisível de uma operação de manipulação de dados, sendo opcional a utilização na aplicação;

f) *connection provider*: empregado para abstrair a aplicação dos *DataSources* ou *DriverManager* adjacentes, sendo opcional a utilização;
g) *transaction factory*: fábrica de instâncias para o *transaction*, sendo opcional a utilização na aplicação;
h) *extension interfaces*: extensões de *interfaces* para a implementação da customização da camada persistente.

Dentre os objetos das camadas de arquitetura do Hibernate, os objetos persistentes possuem funções primordiais no funcionamento do framework, contendo os ciclos de vida definidos em três etapas:

a) *transiente*: a instância não é, e nunca foi associada com nenhum contexto persistente, sendo assim, não possuindo o valor da chave primária;
b) persistente: a instância está atualmente associada a um contexto persistente, possuindo uma identidade persistente e, talvez, correspondente a um registro no banco de dados. Para um contexto persistente em particular, o Hibernate garante que a identidade persistente é equivalente à identidade Java;
c) *destached*: a instância foi associada com um contexto persistente, porém este contexto foi fechado, ou a instância foi serializada por outro processo. Possui uma identidade persistente, e, podendo corresponder a um registro no banco de dados. Para instâncias *destached*, o Hibernate não garante o relacionamento entre identidade persistente e identidade Java.

O entendimento do funcionamento do ciclo de vida dos objetos persistentes do Hibernate é essencial para usar os recursos de maneira adequada e para que não haja problemas nas manutenções dos códigos da aplicação.

Devido ao fato do Hibernate possuir suporte para ser utilizado em diversos ambientes, existe um grande número de configurações para serem realizadas, porém felizmente alguns já possuem valores padrões e que são utilizados na maioria das vezes.

2.2 CONFIGURAÇÕES

Para desenvolver uma aplicação Hibernate é necessário realizar duas configurações, sendo elas a criação das conexões de banco de dados e o mapeamento das tabelas (KONDA, 2014, tradução nossa).

A conexão do banco de dados com o *framework* Hibernate é necessário disponibilizar algumas informações tais como: *Uniform Resource Locator* (URL), credencias e dialetos, pois durante a execução da aplicação, estas informações são lidas para estabilizar a conexão, sendo que estas informações podem estar contidas no arquivo de propriedades: *hibernate.properties,* ou então no arquivo *eXtensible Markup Language* (XML): *hibernate.cfg.xml* (KONDA, 2011, tradução nossa).

Algumas das propriedades que devem ser configuradas são:

a) hibernate.dialect: define o dialeto utilizado para criação dos comandos SQL;
b) hibernate.connection.driver_class: define o *driver* de conexão com o banco de dados;
c) hibernate.connection.url.: URL do banco de dados;
d) hibernate.connection.username: usuário de conexão ao banco de dados;
e) hibernate.connection.password: senha do usuário para conexão;
f) hbm2ddl.auto: caso esta propriedade for configurada com *update*, o Hibernate poderá criar a tabela no banco de dados caso não exista, ou então, corrigir a tabela para que fique idêntica com as definições nos mapeamento.

Figura 3 – Configuração do Hibernate

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
    <property name="hibernate.dialect">
        org.hibernate.dialect.Oracle10gDialect
    </property>
    <property name="hibernate.connection.driver_class">
        com.oracle.jdbc.Driver
    </property>

    <!-- Assume test is the database name -->
    <property name="hibernate.connection.url">
        jdbc:mysql://localhost/test
    </property>
    <property name="hibernate.connection.username">
        root
    </property>
    <property name="hibernate.connection.password">
        root123
    </property>

    <!-- List of XML mapping files -->
    <mapping resource="Employee.hbm.xml"/>

</session-factory>
</hibernate-configuration>
```

Fonte: Do autor.

Conforme a figura 3, foi realizada a configuração do arquivo *hibernate.cfg.xml* utilizando o banco de dados Oracle, com o usuário “root” e senha “root123”. Posteriormente com a configuração do banco de dados na aplicação, deve ser realizado o mapeamento das tabelas e campos da base de dados, que poderá ser realizado através do Java ou então de arquivos XML.

É possível criar a configuração do Hibernate com diversos outros tipos de bancos de dados conforme tabela 1.

Tabela 1 – Principais tipos de bancos de dados e dialetos para configurar no Hibernate.

Banco de dados	Dialeto
DB2	org.hibernate.dialect.DB2Dialect
HSQL	org.hibernate.dialect.HSQLDialect
HypersonicSQL	org.hibernate.dialect.HSQLDialect
Informix	org.hibernate.dialect.InformixDialect
Ingres0	org.hibernate.dialect.IngresDialect
Interbase	org.hibernate.dialect.InterbaseDialect
Microsoft SQL Server 2000	org.hibernate.dialect.SQLServerDialect
Microsoft SQL Server 2005	org.hibernate.dialect.SQLServer2005Dialect
Microsoft SQL Server 2008	org.hibernate.dialect.SQLServer2008Dialect
MySQL	org.hibernate.dialect.MySQLDialect
Oracle	org.hibernate.dialect.OracleDialect
Oracle 11g	org.hibernate.dialect.Oracle10gDialect
Oracle 10g	org.hibernate.dialect.Oracle10gDialect
Oracle 9i	org.hibernate.dialect.Oracle9iDialect
PostgreSQL	org.hibernate.dialect.PostgreSQLDialect
Progress	org.hibernate.dialect.ProgressDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
Sybase	org.hibernate.dialect.SybaseDialect
Sybase Anywhere	org.hibernate.dialect.SybaseAnywhereDialect

Fonte: Tutorials Point

Conforme a tabela 1, pode se analisar que o Hibernate possui suporte para diversos tipos de banco de dados, trazendo um maior conforto ao desenvolvedor para escolher a melhor tecnologia de banco de dados que irá satisfazer no desenvolvimento da sua aplicação com o Hibernate, sendo que não há necessidade de preocupação com qual de banco está sendo utilizado, para realizar o mapeamento das tabelas na aplicação, sendo essa a principal característica do *framework*.

2.3 MAPEAMENTO

A principal função do *framework* que segue o modelo de mapeamento objeto-relacional é estabelecer o mapeamento dos conceitos do modelo orientado a objetos e os conceitos do modelo relacional. Conforme, Linwood e Minter (2010, tradução nossa), os mapeamentos podem ser realizados através de anotações ou então um arquivo de mapeamento XML.

2.3.1 Hibernate annotations

Para realizar o mapeamento do banco de dados com a aplicação, o framework trabalha com a associação de cada tabela do banco de dados a um *Plain Old Java Object* (POJO), que são objetos Java que possuem como estrutura os construtores padrões sem argumentos, variáveis e os métodos *getters* e *setters* para os atributos.

No emprego do JPA para realizar o mapeamento das classes, pode se destacar as principais inclusões:

a) POJOs persistentes;

b) consultas em objetos;

c) configuração simples;

d) integração e teste.

Portanto essas inclusões são importantes para realizar o desenvolvimento do *software,* sendo imprescindível a utilização em qualquer aplicação corporativa hoje em dia, devido à manutenção de uma forma mais simples, fácil e rápida, além de fornecer confiabilidade e criação de testes.

Com a utilização das anotações através do Java, é possível realizar o mapeamento conforme a figura 4.

Figura 4 – Mapeamento com hibernate annotations

```java
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
@Entity
@Table(name="PESSOA")
public class Pessoa {
    @Id
    @GeneratedValue
    private Long codigo;

    @Column(length=1000)
    private String nome;

    @Column
    public Pessoa() {}

    public Pessoa(String nome) {
        this.nome = nome;
    }

    public Long getCodigo() {
        return codigo;
    }
    public void setCodigo(Long codigo) {
        this.codigo = codigo;
    }
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
}
```

Fonte: Do autor.

É necessário instanciar na classe um método construtor padrão sem argumentos, pois o Hibernate necessita instanciar somente com o *construtor.newInstance()*. Com os métodos *getters* e *setters,* o *framework* tem o acesso às propriedades para persistir os dados, além de precisar do identificador único, independente da regra da aplicação.

De acordo com Perkins (2013, tradução nossa), a classe possui a anotação *@Entity*, que determina que o Hibernate deve mapear a classe como uma tabela de banco de dados e a anotação *@Table* deverá ser o nome da tabela. O campo código é definido com o *@Id* e *@GeneratedValue*, para que o Hibernate saiba que o campo deve ser chave primária da tabela e que o valor deve ser gerado automaticamente, e para definir as outras colunas da tabela, deve ser adicionado o *@Column*.

Quando são determinadas as colunas no mapeamento através da anotação @*Column*, podem ser definidos alguns atributos tais como:

a) *name*: nome da coluna;

b) *length*: define o tamanho da coluna;

c) *nullable*: permite a coluna a seja marcado como *not null*;

d) *unique*: permite que a coluna a seja marcado como contendo apenas valores únicos.

Umas das características fundamentais em linguagens de programação orientadas a objetos como Java são as associações, que possuem a chave estrangeira referindo-se a uma chave primária de outra tabela, criando então um esquema relacional. De acordo com Konda (2014, tradução nossa), os relacionamentos entre as tabelas são expressas através das chaves primárias /estrangeiras e outras restrições.

Existem quatro tipos de associações: *one-to-one, one-to-many, many-to-one e many-to-many*, que são utilizadas para compor a estrutura do esquema relacional. A multiplicidade é definida de acordo com Konda (2014, tradução nossa) que é quando se refere à maneira que muitos dos objetos específicos estão relacionados com os objetos de destino, sendo então as associações definidas como múltiplas:

a) *one-to-one*: existem três formas para criar uma relação *one-to-one*, sendo através da relação por chave primária e com tabela associativa, utilizando a anotação @*JoinColumn*, através da chave primária com a anotação @*PrimaryKeyJoinColumn* que realiza a ligação entre as chaves primárias, ou então é possível realizar a associação através da anotação @*JoinTable*, e informando a tabela e quais são os campos que irão realizar a relação através da anotação @*JoinColumn*;

b) *one-to-many*: a associação pode ser realizada de acordo com a anotação @*OneToMany*;

c) *many-to-one:* a associação pode ser realizada de acordo com a anotação @*ManyToOne;*

d) *many-to-many:* a relação *many-to-many* é realizada através da utilização da anotação @*ManyToMany,* e posteriormente a anotação @*JoinTable* repassando os atributos que definem o nome da tabela e as chaves para efetuar a relação.

Além de diversas outras anotações que podem ser utilizadas para realizar o mapeamento, também é possível realizar o mapeamento através de arquivos XML.

O Hibernate suporta a utilização de mapeamento do modelo através do XML, sendo que no mapeamento pode ser definida apenas uma classe ou então múltiplas classes da base de dados, podendo também conter as consultas padrões e os filtros (LINWOOD; MINTER, tradução nossa, 2010).

3 SPRING WEB MVC

O Spring Web MVC faz parte do conjunto do Spring Framework que consiste em um *container* leve que fornece serviços para a aplicação, como no gerenciamento de objetos ou transação. O Spring MVC é um *framework* projetado para as construções de aplicações *web* flexíveis e de fraco acomplamento, baseado no padrão MVC fazendo com que não exista a necessidade de se preocupar com o protocolo HTTP (WALLS, 2011, tradução nossa).

Devido a dificuldade de implementação de aplicações utilizando os *Servlets* e *JSPs* puro, a própria Sun decidiu investir mais na utilização do padrão MVC e de *patterns* como o *FrontController*, sendo que era comum as empresas implementar estes padrões criando então *mini-frameworks*, então foi neste momento que houve a necessidade de implementar o *framework* MVC.

Este *framework* trabalha com pedidos entre um *DispatcherServlet*, *Handler Mapping*, C*ontrollers* e V*iew Resolvers*, sendo que cada componente em Spring MVC possui uma finalidade especifica, utilizado através dos *containers* e a API de Servlets para encapsular o protocolo *HyperText Transfer Protocol* (HTTP). O manipulador padrão é o *@Controller* e *@RequestMapping*, que fornece diversos métodos de manipulações flexíveis.

De acordo com a documentação de referência do Spring MVC, as principais características (JOHNSON et al, 2011, tradução nossa), são:

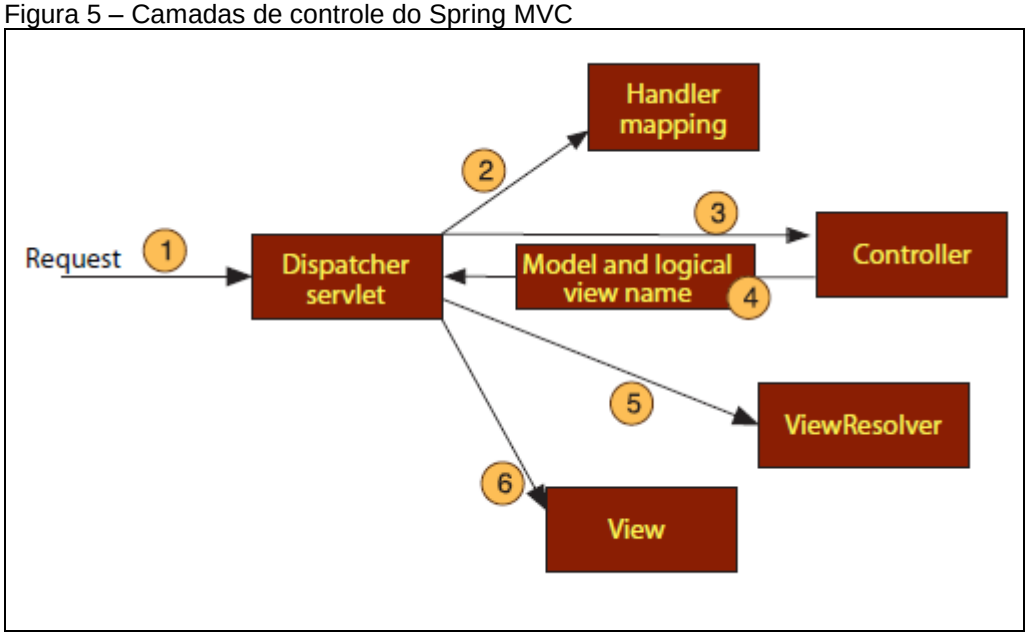
a) separação de papéis: cada arquivo, *controller*, *validator*, objeto de comando, objeto de formulário, objeto de modelo, *dispatcher servlet*, *handler mapping* e *view resolver* são definidos por um objeto especializado;
b) configuração simples e eficiente de classes de aplicações e *frameworks* como JavaBeans, fazendo com que a configuração inclua com facilidade a referência em diferentes contextos, como as dos controladores web para objetos de negócio e validadores;
c) adaptabilidade, não-intrusão e flexibilidade, devido a utilização das anotações de *@RequestParam, @RequestHeader, @PathVariable*, para um determinado cenário;

d) vínculos e validações customizáveis: possibilidade de implementações de validações de tipos de nível da aplicação;
e) *handler mapping* e *view resolutions* customizáveis: o Spring utiliza uma técnica particular para customizar os *handler mapping* e *view resolutions* baseada na simples configuração da *Uniform Resource Locator* (URL);
f) modelo flexível de transferência: modelo de transferência com mapeamento nome/valor;
g) localidade customizável e resolução de temas, suporte para JSPs, suporte para *JavaServer Pages Standard Tag Library* (JSTL), suporte para *Velocity* sem a necessidade de *plugins* extras e entre outros.
h) utilização da biblioteca *Spring Tag Library*, que suporta características como vinculo de dados e temas, dando maior flexibilidade devido as *tags* customizadas;
i) utilização do JSP com *Form Fag Library*, que facilita a escrita de formulários em páginas JSP;
j) classe cujo ciclo de vida tem como escopo a solicitação atual HTTP ou a sessão HTTP, sendo que esta característica vem do *framework* Spring MVC devido a utilização do *container WebApplicationContext*.

Com a utilização do *framework* nas aplicações *web*, se torna mais fácil a manipulação de gravação dos objetos, pois não há a necessidade de montar parâmetro por parâmetro para criar um objeto de uma classe, apenas chamamos os métodos *setters*, pois apenas recebe o objeto do tipo da classe que já estaria populado através da requisição.

As camadas de controle do Spring MVC são: *DispatcherServlet, HandlerMapping, HandlerAdapter, View Resolver e Controller.*

Figura 5 – Camadas de controle do Spring MVC



Fonte: Adaptado de Walls (2011).

De acordo com a figura 5, as camadas de controle do Spring MVC e suas respectivas funções são:

a) *dispatcherServlet*: processa todas as requisições do usuário e invoca elementos c*ontroller* apropriados;
b) *handlerMapping*: baseado na URL da requisição indica ao *DispatcherServlet* qual é o controlador a ser invocado;
c) *handlerAdapter*: responsável por delegar a requisição para mais processamentos;
d) *viewResolver*: interface que devolve ao *DispatcherServlet* qual visão deve ser buscada e instanciada, baseada em seu nome e localização;
e) *controller*: processa os dados e executa alguma regra de negócio da aplicação.

3.1 CONFIGURAR O SPRING MVC

Para adicionar o *framework* na aplicação, deve ser adicionada primeiramente a biblioteca do Spring que contém o Spring MVC, e posteriormente deve ser adicionado um *Servlet* que servirá de *Front Controller* no diretório WEB-INF/lib, que receberá as requisições e enviará as lógicas corretamente. Para configurar o *Servlet* deve ser declarado e adicionado no web.xml as seguintes configurações.

Figura 6 – Configuração do Spring MVC

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  xsi:schemaLocation="http://www.springframework.org/schema/mvc
    http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-3.0.xsd">

  <context:component-scan base-package="exemplo" />
  <mvc:annotation-driven />

  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/"/>
    <property name="suffix" value=".jsp"/>
  </bean>

</beans>
```

Fonte: Do autor.

Na configuração do *Servlet*, deve ser habilitado as anotações com a *tag mvc:annotation-driven* e de qual pacote deverá ser a base para o Spring encontrar as classes. A classe *InternalResourceViewResolver* é utilizada para definir alguns padrões configurando as propriedades de “*sufix*” e “prefix”, sendo que a propriedade “*sufix*” estabelece que as *views* irão terminar com o sufixo .jsp, e a propriedade “prefix” define que estas *views* estarão dentro do diretório “/WEB-INF/views/”.

3.2 DISPATCHERSERVLET

Como os outros *frameworks* o Spring MVC Web é projetado em torno de um *Servlet*, que tem como função despachar os pedidos dos controladores e dispor das funcionalidades que agilizam o desenvolvimento das aplicações, porém com a utilização do *DispatcherServlet* é possível usufruir das características do IOC *Container Spring* devido a integração do Spring MVC com o Spring.

De acordo com Walls (2011, tradução nossa), a função do *DispatcherServlet* é enviar a solicitação para um controlador Spring MVC, que é um componente que processa o pedido, porém uma aplicação comum pode conter vários controladores, sendo assim será necessário decidir pra qual controlador

deverá ser enviado o pedido, então é realizado uma consulta no mapeamento do manipulador para redirecionar corretamente.

O *DispatcherServlet* é um *Servlet* que possui a herança da classe *HttpServlet*, sendo que é necessário mapear as solicitações que serão utilizadas através do *DispatcherServlet* através do mapeamento de URL no mesmo arquivo web.xml que declaramos o arquivo *Servlet* (JOHNSON et al, 2011, tradução nossa).

Figura 7 – Declaração do DispatcherServlet e mapeamento

```
<web-app>

    <servlet>
        <servlet-name>exemplo</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>exemplo</servlet-name>
        <url-pattern>/exemplo/*</url-pattern>
    </servlet-mapping>

</web-app>
```

Fonte: Do autor.

Com a configuração do *DispatcherServlet* na aplicação, as solicitações que iniciarem com /exemplo serão manuseadas pela instância do *DispatcherServlet*, sendo que é possível também configurar diversas outras classes para serem instanciadas a partir do *Servlet*.

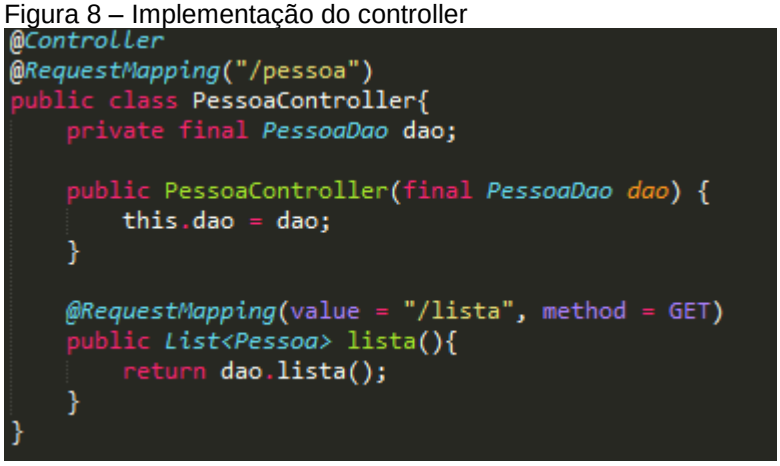
3.3 CONTROLLER

Os *controllers* proporcionam o acesso ao comportamento da aplicação que é definido através de uma *interface* de serviço, sendo que possuem a capacidade de interpretar e transformar em modelo de entrada do usuário para posteriormente exibir ao usuário, sendo que um *controller* é implementado de uma forma abstrata, para possibilitar a criação de diversas variedades de controladores (JOHNSON et al, 2011, tradução nossa).

Para definir um *controller* no Spring MVC é necessário utilizar a anotação @Controller. A anotação para definir um *controller* é utilizada, pois o Spring introduziu um modelo de programação baseada em anotações, assim como o *RequestMapping, RequestParam, ModelAttribute* e etc, dessa forma não existe a

necessidade de estender de classes base especificas ou então implementar interfaces (JOHNSON et al, 2011, tradução nossa).

Figura 8 – Implementação do controller



```
@Controller
@RequestMapping("/pessoa")
public class PessoaController{
    private final PessoaDao dao;

    public PessoaController(final PessoaDao dao) {
        this.dao = dao;
    }

    @RequestMapping(value = "/lista", method = GET)
    public List<Pessoa> lista(){
        return dao.lista();
    }
}
```

Fonte: Do autor.

No *controller* da figura 8, é utilizado a anotação *@Controller* para definir o *controller* “pessoa”, e a anotação *@RequestMapping* define que irá atender as requisições que contenham /pessoa na URL, posteriormente na classe é definido outra anotação com o *@RequestMapping* que irá atender a URL “pessoa/lista” retornando uma lista do tipo Pessoa para o jsp, que caso não esteja explicito para qual jsp deverá retornar, o Spring MVC interpreta que deverá procurar pelo arquivo no /WEB-INF/jsp/pessoa/lista.jsp.

Com a criação dos *controllers* e adicionando a anotação *@RestController*, é possível criar um *WebService*, e com as devidas configurações do *@RequestMapping*, fazer com que o *WebService* retorne um arquivo JSON para ser consumido.

O Spring MVC possui o recurso de criar *controllers* com suporte ao REST, que é um estilo de arquitetura projetado para sistemas distribuídos, realizando uma comunicação entre o cliente e o servidor, sendo comumente associado ao protocolo HTTP (Documentação SPRING).

Segundo Fisher e Murphy (2010, tradução nossa) o REST faz o aproveitamento do HTTP, devido ao fornecimento de uma abordagem coerente, orientada para o que o recurso represente os dados, que ao contrário de uma chamada de procedimento remoto, o REST centraliza a representação de um determinado recurso que é concretizado como uma URL.

4 TECNOLOGIAS WEB

A demanda de criações de sites com o propósito institucional, informativos, *e-Commerce*, armazenamento de informações, portais, mídias sociais ou então instrumentos de publicidades, vem crescendo de forma exponencial no comércio mundial. Desta forma, é possível identificar a necessidade de utilizar tecnologias robustas que permitam maior segurança e confiabilidade nas transações comerciais ou serviços utilizados na *web* em nosso dia a dia.

A internet vem se destacando como um meio de comunicação que possui um alto nível de eficácia, devido à fácil interação e agilidade para realizar o tráfego e trocas de informações. Para a criação destas páginas que estão disponíveis na internet, são utilizadas as tecnologias *web*, que são definidas pelo código e recursos que fornecem para um servidor *web*, e que tem por finalidade a execução do conteúdo num navegador *web*.

Após o surgimento de diversos tipos de navegadores para exibir as páginas, foi indispensável à padronização das tecnologias que os navegadores interpretam, sendo assim, ficou definido as três linguagens padrões: o *HyperText Markup Language* (HTML), *Cascading Style Sheets* (CSS) e JavaScript.

Através da linguagem de marcação HTML é que é possível desenvolver páginas da *web*, pois é utilizada para realizar a formatação do conteúdo que deverá ser exposto, sendo que o CSS codifica o estilo da página e o *JavaScript* é utilizado para realizar as interações. Conforme Wright (2012, tradução nossa), para a construção de uma página *web* é necessário manter o HTML, CSS e JavaScript separados, e referir como três camadas, sendo elas de estrutura que se refere ao HTML, a apresentação que é a utilização do CSS e o comportamento que é a utilização do JavaScript para lidar com as interações da página.

4.1 HTML

O HTML é uma linguagem de marcação de texto, utilizada na formatação de páginas publicadas na *web* ou na intranet, que possibilita a leitura em qualquer navegador e computador.

Conforme Powel (2010, tradução nossa), um arquivo HTML é um documento que possui diversas informações as quais se deseja publicar na *web*, e

as instruções de marcações para que o navegador estruture e apresente a página conforme o desejado.

Esta linguagem é baseada no conceito de hipertexto, que são conjuntos de elementos nos quais podem ser as palavras, vídeos, áudios, imagens e entre outros, que unidos compõe uma rede de informação, a qual nos dá a liberdade de navegação na *web* através dos links.

A última versão do HTML, o HTML 5, possui como um dos principais objetivos, facilitar a manipulação dos elementos, permitindo que o desenvolvedor possa alterar as características dos objetos de uma forma não instrusiva. Antigamente era normal desenvolver sites com um enorme encadeamento de *divs*, sendo que eram apenas diferenciadas por *ids* para organizar o corpo da página, tornando o documento de desenvolvimento poluído e de difícil manutenção. Com o HTML5 é possível utilizar as *tags* de estruturas do corpo da página, eliminando então a necessidade de criação de *divs* encadeadas (BATISTA, 2009).

Um dos principais motivos que levaram a criação do HTML5, era a necessidade de implementar recursos que permitissem que as ferramentas para o CSS e o JavaScript pudessem ser utilizados da melhor forma possível, fazendo com que os desenvolvedores programem páginas e aplicações *web* com o código mais nativo possível, para que não exista a necessidade de utilização de plug-ins.

4.1.1 História

O físico britânico Tim Berners-Lee inicialmente sugeriu e propôs um projeto com um conceito de hipertexto, que era chamado de *Enquire* no ano de 1980, que inicialmente foi planejado e desenvolvido em Pascal, posteriormente no ano de 1989, com o auxílio de um universitário, Robert Cailliau, foi possível o desenvolvimento da primeira comunicação bem sucedida entre um cliente *HyperText Transfer Protocol* (HTTP) e um servidor, sendo assim surgindo o *World Wide Web*, apesar do protocolo HTTP ter sido implementado apenas no fevereiro de 1993 (WILLIAM, 2012).

Segundo William (2012), a primeira versão do HTML surgiu em 1990, e foi baseado na linguagem *Standard Generalized Markup Language* (SGML), que utilizava o conceito de estruturação dos documentos e utilização de *tags*, e em 1995 os padrões *web* foram regularizados pela *World Wide Web Consortium* (W3C) .

Ao decorrer das primeiras versões lançadas do HTML, esta linguagem passou por várias evoluções até chegar à forma de desenvolvimento que utilizamos atualmente, o HTML5, que nos fornece como vantagens, a facilidade de usar os elementos *canvas*, áudio e vídeo, e diminuição de encadeamentos de *divs*, além de facilitar a visualização das *tags* no documento devido a limpeza de poluição que o encadeamento trazia ao documento, podendo então ler com mais clareza as *tags* que estão formatando o documento.

4.1.2 Tags

O documento HTML é formado por diversas *tags* ou etiquetas, que rotulam como o navegador deve apresentar a página *web*, ou seja, o texto que está contido entre a *tag* de abertura e fechamento, será processado de acordo com o comando da *tag*.

Figura 9 – Utilização de *tags*

Fonte: Do autor.

As etiquetas possuem uma marca de início, e não necessariamente uma marca de fim, mas quando possuem a marca de fim é definida por uma barra dentro da *tag*, sendo que esta *tag* pode envolver algum conteúdo que deverá ser alterado de acordo com o comando especificado (POWEL, 2010, tradução nossa).

Além de existir *tags* de aberturas, fechamentos e *tags* isoladas, ainda pode ser adicionados atributos nas marcações, assim como, por exemplo, da *tag* de imagem: <img>, que definimos no atributo o caminho da imagem que deverá ser renderizada no HTML.

As *tags* mais utilizadas para o desenvolvimento de páginas *web* e suas funcionalidades:

a) <html> </html> : início e término de um arquivo em HTML;

b) <head> </head>: área do cabeçalho da página;
c) <boby> </body>: área do corpo da página, que será visível no documento;
d) <title> </title>: define o título da página;
e) <body bgcolor = ?>: cor de fundo;
f) <b> </b: texto em negrito;
g) : texto em itálico;
h) : define um parágrafo;
i)
: adiciona uma quebra de linha;
j) <font size = ? >: tamanho da fonte das letras;
k) <a href=”URL”>: definição de um hiperlink;
l) <table> </table>: inserção de uma tabela;
m) <tr> </tr>: linha de uma tabela;
n) <td> </td>>: célula de uma linha;
o) <frameset> </frameset>: define um conjunto de *frames*.

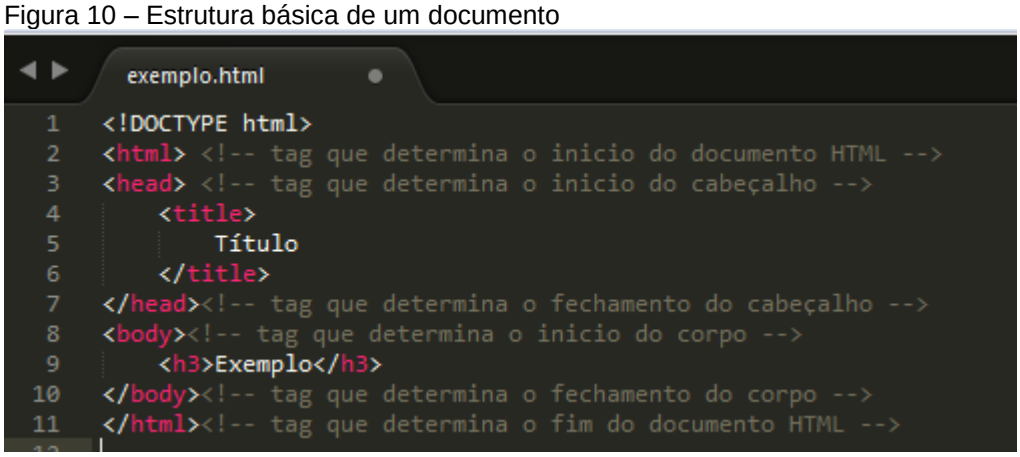
Dentre as *tags* citadas acima, o HTML nos disponibiliza diversas outras *tags* para utilizar na formatação do documento ou então para estruturação do arquivo.

4.1.3 Estrutura básica de um documento

No documento de HTML, existem algumas *tags* que definem uma estrutura básica da página, sendo elas:

a) <html>: define o início de um arquivo HTML, e possui como responsabilidade informar ao navegador que os códigos abaixo devem ser interpretados como códigos HTM;
b) <head>: determina o cabeçalho da página *web*;
c) <body>: determina o corpo do documento, o qual contém todo o conteúdo principal que irá aparecer no navegador.

Figura 10 – Estrutura básica de um documento



Fonte: Do autor.

Conforme a figura 10, o documento contém as *tags* de estrutura básica para implementação de uma página *web*, determinando o início do documento, logo em seguida o cabeçalho, o corpo do documento e por fim a finalização da *tag* que determina o contexto do documento HTML. Na estrutura de básica do HTML, pode ser adicionado a tecnologia que associada proporciona uma grande flexibilidade da manipulação da estrutura *front end*, conhecida como CSS.

4.2 CSS

Com a utilização do CSS no desenvolvimento das aplicações *web*, é possível definir a cor do texto, estilos de fontes, espaçamentos entre os parágrafos, imagens de fundos, as cores e diversos outros características que podemos personalizar na aparência da página.

Conforme Pouency e York (2011, tradução nossa) o CSS é uma linguagem que tem como finalidade definir a aparência dos documentos que estão escritos na linguagem de marcação, como por exemplo, o HTML. O principal objetivo é separar o aspecto visual e o conteúdo do documento, pois a definição do estilo é definida em outro arquivo, sendo que no desenvolvimento da aplicação deve ser criada uma ligação deste arquivo que contém os estilos com as páginas da aplicação.

Figura 11 – Estrutura básica de um seletor

```
1   h1 {
2       color: #666;
3       font-weight: bold;
4   }
```

Fonte: Do autor.

Conforme a figura 11, podemos definir a cor da fonte e que a fonte será em negrito quando o documento da página web tiver a *tag* <h1>, sendo assim, definindo uma regra.

De acordo com Pouency e York (2011, tradução nossa), em 1994 foi sugerido por Hakon Wium Lie o *Cascading Style Sheets HTML*, que posteriormente se transformou em CSS, pois não era aplicada somente com a linguagem de marcação HTML, e em 1996 foi lançado publicamente o CSS 1.

Segundo Olsson e O'Brien (2008, tradução nossa) a primeira versão do CSS, possuía as propriedades para controle de tipografia, como fontes, alinhamento de textos, margens, espaçamentos e formatação da lista, permitindo também especificações das dimensões das caixas de blocos e especificações de caixas com bordas, porém deixava a desejar com as personalizações de *layout* e *design*.

Após o lançamento do CSS 1, os desenvolvedores já iniciaram os trabalhos para desenvolver o CSS 2, devido aos erros da primeira implementação, como por exemplo, os modelos de caixa que ameaçaram o fim desta linguagem. O CSS 2 foi lançado em 1998, e teve como as principais características os novos recursos de células de tabelas, novos seletores, estilos de folhas que poderiam ser importados para outras folhas de estilo e entre outros.

Posteriormente com o término do desenvolvimento do CSS 2, logo se iniciou a construção da versão do CSS 2.1, juntamente com a implementação do CSS 3, que teve como diferencial a criação de módulos, ao invés de ser um longo documento como nas versões anteriores, totalizando então mais de 30 documentos. (OLSSON; O'BRIEN, 2008, tradução nossa).

Com a criação do CSS3, podemos levantar os principais pontos que foram aperfeiçoados:

a) selecionar primeiro e último elemento;

b) selecionar elementos pares ou ímpares;

c) selecionar elementos específicos de um determinado grupo de elementos;

d) gradiente em textos e elementos;

e) bordas arredondadas;

f) sombras em texto e elementos;

g) manipulação de opacidade;

h) controle de rotação;

i) controle de perspectiva;

j) animação;

k) estruturação independente da posição no código HTML.

Para realizar a associação do documento HTML com o arquivo do CSS, é comum a adição de um elemento que serve como link no *head* do arquivo HTML, da seguinte forma:

<link rel = "stylesheet" hrel = "/syles.css" media="screen" title="Estilo1" />

Desta forma é possível definir diversos estilos para as páginas e atribuir um título a cada um, sendo que na escrita de um arquivo de CSS, devem ser utilizados os seletores que permitem a seleção de algum elemento na página para atribuir as devidas formatações, formando então uma lista de regras, podendo conter parênteses, colchetes e chaves para organizar estas regras.

4.2.1 Seletores

A sintaxe do CSS é definida por seletores que tem como principal característica selecionar um elemento, para posteriormente modificar de acordo com as propriedades e valores determinados na regra de formatação. Segundo Olsson e O'Brien (2008, tradução nossa), os seletores é um conjunto de regras, que estão contidos em blocos de chaves { }, sendo que todos os elementos contidos nesta seleção, deverão ter a sua regra aplicada.

Figura 12 – Sintaxe do seletor

```
1   seletor {
2       propriedade: valor;
3   }
```

Fonte: Do autor.

Sendo então o seletor o elemento que será selecionado do arquivo do HMTL, a propriedade e o valor são as particularidades que serão modificadas do elemento selecionado.

Os seletores são divididos em seletores simples e seletores complexos, sendo que o seletor simples é subdividido em seletores encadeados e seletores agrupados, e os seletores complexos são subdivididos em seletores por *CLASS* e seletores por *ID*. Com o desenvolvimento de estilos com o CSS 3, podemos contar com os seguintes novos seletores: seletores por atributos, por oposição e por estado.

Além de o CSS proporcionar maior eficiência no gerenciamento do layout devido à redução de códigos HTML e descrevendo apenas estilos, é necessário utilizar o JavaScript para melhorar a interatividade com o usuário.

4.3 JAVASCRIPT

O JavaScript é uma das linguagens de programação mais populares por ser a linguagem de HTML, para a *web*, computadores, servidores, *notebooks*, *tablets, smartphones* e muito mais (CROCKFORD, 2008, tradução nossa).

Esta é uma linguagem de programação interpretada, utilizada para ser executada do lado do cliente, e para fazer as interações com o usuário sem a obrigação de fazer a ida e volta dos dados ao servidor, controles no navegador, comunicações assíncronas e alterações das informações no documento.

Para introduzir o código JavaScript, basta incluir a *tag* <script> para que a página aguarde o *parsing* do *script* e a sua execução, podendo estar introduzido dentro da *tag*, ou então em um documento externo. Conforme Zakas (2013) quando o navegador se depara com a *tag* <script>, o mesmo interrompe o processamento da página e executa o código JavaScript, para posteriormente prosseguir com o carregamento da página, caso o código esteja em um documento externo declarado no atributo *src*, primeiramente o navegador irá fazer o download do documento para em seguida prosseguir com a execução do código.

A linguagem JavaScript suporta os elementos da sintaxe de programação da linguagem C, tendo como exceção o escopo, pois o JavaScipt faz uso do escopo somente a nível de função, segundo Zakas (2013), o escopo implica na determinação de quais variáveis a função poderá ter acesso através do *this*. Outra

característica é a tipagem dinâmica, que são os tipos associados aos valores e não a variáveis, sendo assim uma linguagem de tipagem fraca, sendo que também possui a características de funções de primeira classe, que são objetos que contém propriedades e métodos, que permite a passagem de argumentos e retornar qualquer objeto.

4.3.1 TIPOS

O JavaScript possui a tipagem dinâmica que consiste na habilidade da linguagem verificar o tipo da variável em tempo de compilação ou execução, sendo assim é possível ter na mesma variável e em momentos diferentes, valores de tipos distintos.

A tipagem dinâmica pode causar dois problemas como: perda de perfomance devido ao tempo de processamento necessário para descobrir o tipo de variável, e os erros intermitentes, que são erros que ocorrem sem motivo aparente, devido aos enganos dos programadores ao atribuir valores a variável.

Os tipos de variáveis suportados pelo JavaScript são: *String* para letras e palavras, *Number* para números inteiros e decimais, *Boolean* para os valores de verdadeiro e falso, *Array* para poder armazenar várias informações de tipos diferentes e entre outros.

4.3.2 OPERADORES

Os operadores são blocos de construção de expressões, que executa uma operação e retorna um valor. Segundo Wright (2012, tradução nossa), os operadores são símbolos em JavaScript que possam ser reconhecidos na matemática, sendo então os operadores de subtração, multiplicação, comparação e definição de valores. Os operadores são:

a) operador + : adicionar ou concatenar;
b) operador - : subtração;
c) operador * : multiplicação;
d) operador / : divisão;
e) operador ++ : incrementar;
f) operador --: decrementar;

g) operador = : definir valor.

Com a utilização destes operadores, os valores que podem adicionar, subtrair, multiplicar, dividir, incrementar e decrementar são apenas os valores do tipo *number*, e os valores do tipo *string* apenas comparam e concatenam.

4.3.3 INTERAÇÕES

Existe a camada de interação para executar as ações que estão contidas nos arquivos JavaScript. A interação *loop* permite a execução de um bloco de código em certo número de vezes, sendo que este bloco de código pode estar contida em uma função ou então ser uma função.

Outras interações são as condicionais que conforme Wright (2012, tradução nossa), são utilizadas para executar um código diferente de acordo com certas condições, abrindo então oportunidade para realizar comparações entre valores.

As interações condicionais existentes são: *if* e *switch.* A instrução *if* é a mais utilizada, podendo ser definida com três tipos:

a) declaração normal do *if*;

b) instrução *if / else,* que então possui duas condições;

c) instrução if/ else if/ else, que possui um número infinito de condições.

Podendo ter os valores comparados para as condições com os seguintes operadores:

a) < : inferior;

b) > : superior;

c) <=: menor ou igual a;

d) >= : maior do que ou igual a;

e) != : não igual a;

f) == : igual;

g) === : exactamente igual a;

h) !== : exatamente não igual a.

A instrução *switch*, segundo Wright (2012, tradução nossa) analisa a existência de uma determinada condição, ou então, determinado caso, e quando encontrar, a instrução *switch* irá parar ao contrário da instrução *if* irá continuar

percorrendo todas as condições. Existem outras interações como a criação das funções, funções anônimas, métodos, eventos e entre outros.

5 FRAMEWORK ANGULARJS

O AngularJS é um *framework open-source* desenvolvido pela Google, que fornece aos desenvolvedores *JavaScript* uma estrutura robusta para elevar a produtividade e desenvolvimento de aplicações ricas e com potencial (DIETZ, 2013, tradução nossa).

A construção do *framework* iniciou em 2009 como um projeto pessoal de Misko Hevery, funcionário da Google, e posteriormente a Google Inc. oficialmente começou a apoiar este projeto, sendo que a primeira versão foi lançada em junho de 2012 (KOZLOWSKI; DARWIN, 2013, tradução nossa).

Uma das principais particularidades que torna a utilização deste *framework* tão assíduo no desenvolvimento de páginas *web* é devido à funcionalidade de trabalhar como uma extensão do documento HTML, com o *DataBinding*, *templates* e a fácil utilização do *Ajax*.

Segundo Schmitz e Lira (2012), este *framework* é praticamente uma linguagem declarativa, pois utiliza a adição de propriedades para modificar o comportamento da linguagem e interage de forma dinâmica com os elementos HTML, sendo que estes parâmetros são denominados de diretivas.

Adicionado recentemente à lista de *frameworks* que utiliza o padrão *Model-View-Controller* (MVC), acabou atraindo muita atenção devido ao sistema de modelos inovadores, facilidade de desenvolvimento e práticas de engenharia sólidas, sendo que a modelagem de sistema se torna diferencial devido à utilização do HTML como linguagem de *templates*, abstração da manipulação do *Document Object Model* (DOM) explicitamente e possui componentes extensíveis. (KOZLOWSKI; DARWIN, 2013, tradução nossa).

Inicialmente a aplicação do AngularJS utilizava o modelo arquitetural de *Model-View-Controller* (MVC) no desenvolvimento de software, e posteriormente os próprios autores informaram a adesão do modelo *Model-View-Whatever (*MVW). Independente do padrão de projeto de software que será utilizado para o desenvolvimento da página web, é necessário sempre seguir os conceitos de *model*, *view* e *controller*, para separar corretamente as funcionalidades para cada arquivo, dos recursos mais importantes. O que difere o *framework* AngularJS com os demais *frameworks* JavaScript, é a utilização de diretivas, que agregam um valor elevado ao HTML pois lida diretamente com o DOM.

5.1 DIRETIVAS

Este recurso amplia o vocabulário do HTML, permitindo novos comportamentos de forma declarativa, pois com a adição das *tags* é possível ensinar o navegador sobre como deve exibir a página web, pois altera o comportamento padrão do HTML, sendo que é possível personalizar e reutilizar as *tags* especificadas, criando então componentes customizáveis dentro da aplicação.

Segue abaixo algumas das principais diretivas:

a) ng-app: define o elemento como elemento raiz da aplicação;
b) ng-bind: modifica o texto de um elemento HTML automaticamente, de acordo com o seu resultado, vindo das regras de negócio;
c) ng-model: proporciona a ligação da *view* com o escopo, criando uma ligação bidirecional;
d) ng-class: carrega dinamicamente os atributos da classe;
e) ng-controller: determina um *controller* JavaScript;
f) ng-repeat: interação com cada elemento de um *array*;
g) ng-show: mostra o elemento HTML;
h) ng-hide: esconde o elemento HTML;
i) ng-switch: permite instanciar um *template* em uma lista de opções, de acordo com o valor resultante da expressão;
j) ng-view: utilizada para manipulação de rotas;
k) ng-if: declaração de condição.

Figura 13 – Utilização das diretivas: ng-app, ng-init e ng-model.

exemplo.html

```
<!DOCTYPE html>
<html>

<head>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.26/angular.min.js"></script>
</head>

<body>

<div ng-app="" ng-init="nome='Daiana'">
<!-- ng-app: inicializa uma aplicação em angular.
ng-init: inicia os dados da aplicação. -->
<p>Digite o seu nome: <input type="text" ng-model="nome"></p>
<!-- ng-model: faz a ligação dos campos do modelo com os elementos HTML -->
<p>Seu nome: {{ nome }}</p>

</div>

</body>
</html>
```

Fonte: Do autor.

Conforme a figura 13, pode se observar a utilização das diretivas: ng-app, ng-init e ng-model, sendo que a ng-app foi utilizada para iniciar a aplicação de exemplo, a diretiva ng-init para iniciar os dados da aplicação e ng-model para criar a ligação bidirecional da *view* com o campo “nome”, utilizando o recurso do *data binding*.

As diretivas também podem ser criadas de acordo com a necessidade de cada desenvolvedor. De acordo com Green e Seshadri (2013, tradução nossa), as criações das diretivas passam por dois processos, primeiramente pela compilação, sendo que as diretivas que estão ligadas a algum elemento do DOM são encontradas e processadas, além disso, qualquer manipulação do DOM acontece na etapa de compilação, sendo assim a segunda etapa ocorre à função da ligação entre o âmbito e o elemento.

Figura 14 – Criação de diretivas.

```
angular.module('exemploDirective', [])
.controller('myController', ['$scope', function($scope) {
  $scope.pessoa = {
    nome: 'Daiana',
    endereco: 'Avenida Professor'
  };
}])
.directive('myDirective', function() {
  return {
    template: 'Nome: {{pessoa.nome}} Endereco: {{pessoa.endereco}}'
  };
});
```

Fonte: Do autor.

Conforme a figura 14, a criação da diretiva *myDirective*, é criada a partir do módulo *directive*, retornando então o *template* com as informações de nome e endereço

5.2 DATA BINDING

O *Data Binding* é a ligação automática de variáveis, dando a capacidade de alterar o valor da propriedade fazendo com a interface do usuário seja modificada em tempo real, sendo que na maioria das vezes é utilizado para realizar a ligação de alguma variável do JavaScript a algum elemento do documento HTML (SCHIMTZ;LIRA, 2013).

Este recurso se torna trivial na utilização do *framework*, pois os desenvolvedores não precisam se preocupar em escrever diversas linhas de código para que os objetos alterados sejam atualizados dinamicamente na interface.

Figura 15 – Utilização do *data binding*

```
<!DOCTYPE html>
<html>

<head>
<script
  src= "http://ajax.googleapis.com/ajax/libs/angularjs/1.2.26/angular.min.js">
</script>
</head>

<body>

    <div data-ng-app="" data-ng-init="valor=2;quantidade=2">

        <h2>Calcular</h2>

        Quantidade: <input type="number" ng-model="quantidade">
        Valor: <input type="number" ng-model="valor">

        <p><b>Valor total:</b> {{quantidade * valor}}</p>

    </div>

</body>
</html>
```

Fonte: Do autor.

Conforme a figura 15, é um exemplo de uma aplicação para calcular o valor total, que inicia com um modelo com os campos de “valor” e “quantidade” e com os seus respectivos valores, e associando os dois campos de texto com as

diretivas ng-model, para que se tornem sincronizados, sendo que o valor total é calculado conforme as alterações do “valor” e “quantidade” através da expressão {{quantidade * valor }}.

5.3 INJEÇÃO DE DEPENDÊNCIAS

Segundo Green e Seshadri (2013, tradução nossa), a injeção de dependências do AngularJS é o que permite aos desenvolvedores a seguir um estilo de desenvolvimento, podendo reutilizar as funções criadas em diferentes escopos, seguindo o padrão de projeto chamado de Lei de Deméter, ou Principio do Conhecimento Mínimo. Este recurso possibilita a injeção do código fonte em outra parte do código da aplicação, requerendo então uma divisão de funcionalidades, como serviços, controladores ou provedores.

Utilizando a injeção de dependência no desenvolvimento, podemos ter um controle maior na organização do código fonte por módulos, dando maior liberdade de criação de componentes, para que não exista a necessidade de duplicação de códigos, tornando a aplicação mais escalável e de fácil manutenção, geralmente utilizado para isolar os serviços, *resource*, diretivas e principalmente *controllers*.

5.4 CONTROLLERS

Os *controllers* em AngularJS centralizam as regras de negócios das telas desenvolvidas, sendo que não deve fazer referência ou manipulações no DOM de forma direta (DIETZ, 2013, tradução nossa).

Segundo Schmitz e Lira (2014), geralmente um *controller* é um arquivo JavaScript, que no seu conteúdo possui funcionalidades ligadas ao arquivo HTML, não necessitando a criação de um *controller* para cada arquivo HTML, mas sim criando devido a necessita de centralização das regras de negócios.

Figura 16 – Utilização do *controller*

```
<!DOCTYPE html>
<html>

<head>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.2.26/angular.min.js"></script>
</head>

<body>

<div ng-app="" ng-controller="exemploController">

Nome: <input type="text" ng-model="nome"><br>
Curso: <input type="text" ng-model="curso"><br>
<br>

{{nome}} faz curso de {{curso}}

</div>

<script>
function exemploController($scope) {
    $scope.nome = "Daiana",
    $scope.curso = "Ciencia da computacao"
}
</script>

</body>
</html>
```

Fonte: Do autor.

Conforme a figura 16 o *controller* pode ser definido através da diretiva *ng-controller*, e posteriormente entre as *tags* de <script> desenvolver as funcionalidades necessárias, como por exemplo, quais são as informações que os campos “nome” e “curso” irão iniciar, sendo que estão ligadas através do $scope.

5.5 SCOPE

Para criar um *model* no AngularJS, é necessário definir o $scope, que de acordo com Kozlowski e Darwin (2013, tradução nossa), é um objeto o qual determinamos os campos do modelo da *view*, o qual é possível realizar diversas manipulações de valores para a renderização de um modelo, e a fácil adição de funcionalidades especificas de uma *view*.

Segue na figura 18 um exemplo de definição do $scope, com os dados de “nome” e “curso”, que nos permite controlar o domínio do modelo e para que seja possível utilizar as operações.

Figura 17 – Utilização do escopo

```
<script>

function exemploController($scope) {
    $scope.nome = "Daiana",
    $scope.sobrenome = "Cambruzzi",
    $scope.curso = "Ciencia da computacao"
}

</script>
```

Fonte: Do autor.

De acordo com a figura 17, é possível definir através do $scope, quais são os valores que o ”nome”,”sobrenome” e “curso” irão receber.

5.6 ROTAS

Para criar uma estruturação de páginas, é necessária a criação de rotas para realizar a navegação da aplicação web, que deve ser alterada de acordo com a navegação do usuário. As rotas definem uma estrutura básica para a aplicação, sendo que de acordo com Kzolowski e Darwin (2013, tradução nossa), a rota mapeia os possíveis modelos parciais conforme as Uniform Resource Locator (URL), podendo então analisar com clareza quais telas compõe o caminho.

Conforme Schmitz e Lira (2013), para implementar as rotas é utilizado o recurso chamado de Deep Linking, que consiste na criação das rotas na URL do documento HTML, que separa ainda mais as camadas da aplicação, devido a manipulação de independentes dos códigos HTML.

5.7 FACTORY

Segundo Green e Seshadri (2013, tradução nossa) o *factory* é um serviço não configurável com a lógica de criação complexa, que pode especificar uma função que, quando chamada, retorna a instância do serviço.

Figura 18 – Utilização do *factory*

```
var app = angular.module('appExemplo', []);

app.factory('exemploFactory', function(){
    return {
        sayHello: function(text){
            return "Factory \"Hello " + text + "\"";
        }
    }
});

function HelloCtrl($scope, exemploFactory)
{
    $scope.fromFactory = exemploFactory.sayHello("World");
}
```

Fonte: Do autor.

Conforme a figura 18, a *factory* possui a seguinte sintaxe: *module.factory('nomeFactory', função)*, e quando é realizada a chamada da *factory* através do nome, é chamada a função passando o argumento do texto para retornar a frase completa.

5.8 SERVICE

Os *services* do AngularJS são objetos que são ligados entre si, utilizando a injeção de dependência, sendo utilizados para organizar os serviços e compartilhar o código (Documentação AngularJS).

Figura 19 – Utilização do escopo

```
angular.module('myServiceModule', []).
controller('MyController', ['$scope','notificar', function ($scope, notificar) {
    $scope.callNotificar = function(msg) {
        notificar(msg);
    };
}]).
factory('notificar', ['$window', function(win) {
    var msgs = [];

    return function(msg) {
        msgs.push(msg);

        if (msgs.length == 3) {
            win.alert(msgs.join("\n"));
            msgs = [];
        }
    };
}]);
```

Fonte: Do autor.

Conforme a figura 19, o serviço de notificar é injetado passando o parâmetro da mensagem, retornando a função e conforme o número de mensagens aparece então a janela de alerta com as mensagens digitadas.

6 DISPOSITIVOS MÓVEIS

Com o elevado número de compras de dispositivos móveis e que crescem a cada dia, esses dispositivos já representam a maioria dos sistemas computadorizados atualmente, causando então, uma evolução no desenvolvimento de aplicativos para esses aparelhos (DEVMEDIA, 2015).

Entre os sistemas operacionais existentes para os dispositivos móveis, o sistema operacional mais utilizado em portáteis do mercado é o Android, que existem cerca de mais de um bilhão de aparelhos ativos (TECHMUNDO, 2014).

Entretanto com a utilização dos dispositivos móveis com o sistema operacional Android, é possível utilizar recursos importantes para a comunicação com outros dispositivos, como o Bluetooth do aparelho, que permite realizar transferências de dados e outras funcionalidades que podem ser consumidas pelos aplicativos desenvolvidos.

6.1 PLATAFORMA ANDROID

O Android consiste em uma plataforma *open source,* projetada para dispositivos móveis e defendida pelo Google e Open Handset Alliance, sendo que esta aliança visava acelerar as inovações móveis e disponibilizar uma experiência móvel mais rica, já que a parceria consistia entre duas empresas que são envolvidas em tecnologia de telefonia móvel (GARGENTA, 2011, tradução nossa).

Com o surgimento do Sistema Operacional Android, os desenvolvedores tiveram uma maior liberdade para implementar os aplicativos, pois os outros sistemas operacionais eram proprietários e não permitiam aplicativos de terceiros. Com o Android, o desenvolvedor tem a possibilidade de desenvolver os seus aplicativos móveis utilizando os recursos do hardware (MEIER, 2010, tradução nossa).

De acordo com Rogers et al (2009), com o surgimento do Android houve um crescimento no ambiente de desenvolvimento de aplicativos móveis, pois anteriormente sofria com algumas barreiras, como por exemplo, um ambiente de aplicação de acordo com cada marca, softwares proprietários e o bloqueio aplicativos de terceiros.

O Android foi projetado com base na versão 2.6 do Kernel do Linux, possuindo então uma arquitetura bem estruturada e baseada em camadas, com a finaliade de tratar isoladamente as funções de aplicações, execuções paralelas, serviços centrais e entre outros (PEREIRA; SILVA, 2009).

6.1.1 Arquitetura

O Sistema Operacional Android é baseado no Kernel do Linux, e as aplicações são desenvolvidas em Java, sendo assim é necessário compreender as camadas e de que forma ocorrem as interações dos processos. A arquitetura do Android está dividida em quatro camadas: Aplicações, Framework, Bibliotecas/Android Runtime e Kernel Linux.

Figura 20 – Arquitetura do Android

Fonte: Pereira e Silva (2009).

De acordo com Burnette (2010) e Lee (2011), o sistema operacional Android é dividido em quatro camadas conforme representada na figura 20, sendo que cada uma desempenha seu devido papel no processo, sendo elas:

a) linux Kernel: núcleo do Android, onde possui todos os *drivers* de baixo nível de dispositivos para os diversos componentes de *hardware*, e é

utilizado internamente para o gerenciamento de memória, de processos e outros serviços;

b) bibliotecas: possui todo o código que faz as principais funcionalidades do sistema operacional. Estas bibliotecas compartilhadas são compiladas para a arquitetura específica utilizada pelo aparelho e que já estão pré-instaladas pelo fabricante;
c) *android runtime*: funciona na mesma camada que ao do *Framework*, pois possui a função de executar o código compilado que está no *Framework*. A *runtime* possui diversas bibliotecas que possibilita aos desenvolvedores a criação de aplicativos com a linguagem Java, sendo que nesta camada se encontra a máquina virtual Dalvik, que possibilita com que cada aplicativo seja executado em seu próprio processo com sua própria estância;
d) *framework*: é a camada que contém o *Application Program Interfaces* (API) e os recursos em alto nível que é disponibilizado aos desenvolvedores para a criação dos aplicativos, incluindo como o controle do ciclo de vida de aplicações e localização geográfica do aparelho;
e) aplicações: é a camada mais alta da arquitetura, pois é a qual se localiza os aplicativos padrões do sistema e as aplicações pessoais que estão disponíveis aos usuários.

6.1.2 Blocos de construção

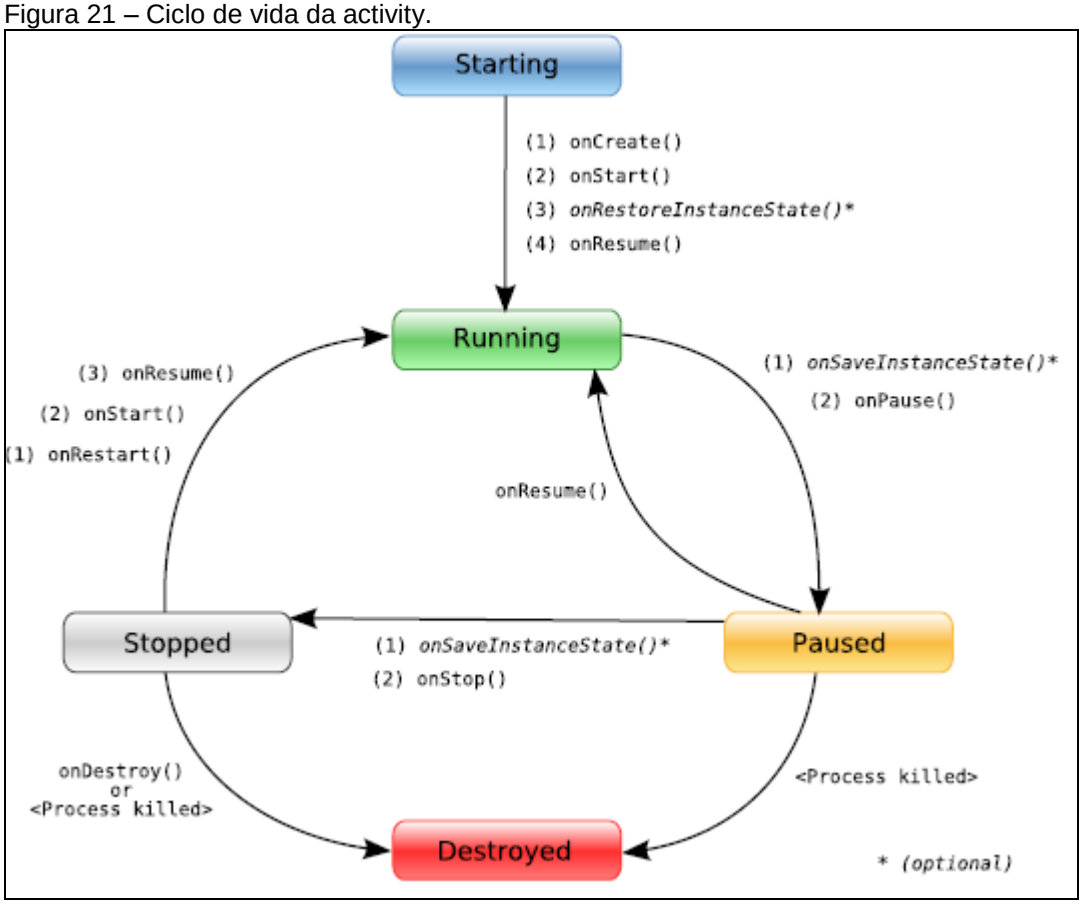
De acordo com Gargenta (2011, tradução nossa), os blocos de construção são os componentes que são utilizados para a construção de aplicativos Android, sendo que cada bloco possui um conceito possibilitando o desenvolvimento independente da aplicação.

Os blocos de construção são divididos em: *Activity, Service, BroadcastReceiver* e *ContentProvider*, que possuem diversas funcionalidades distintas, como executar uma função com foco no usuário, processos em *background*, receber mensagens e gravar e recuperar dados.

6.1.2.1 Activities

Uma *activity* geralmente é uma tela da aplicação que o usuário interage no dispositivo e que controla o ciclo de vida das aplicações (GARGENTA, tradução nossa, 2011). A *activity* em sua vida útil pode estar em um de vários estados conforme a figura 21.

Figura 21 – Ciclo de vida da activity.



Fonte: Burnette (2010).

De acordo com Burnette (2010), é possível sobre-escrever os métodos acima na sua própria classe *Activity*, para que seja executado o que o desenvolvedor deseja a cada estado.

6.1.2.2 Services

Um *service* é uma tarefa executada em segundo plano, sem que exista a necessidade de interação com o usuário, e é independente de qualquer *activity*, ou seja, não possui interface (BURNETTE, tradução nossa, 2010).

Um *service* pode ser implementado de duas formas diferentes (DEVELOPER ANDROID, 2013, tradução nossa):

a) started Service: este serviço é executado a partir do método StartService(), e quando é iniciado não possui um tempo determinado para finalizar, geralmente é utilizado em operações que não necessitam retornar resultados ao componente que fez a chamada, como por exemplo em downloads de arquivos;
b) bound Service: ao contrário do Started Service, esta forma de implementação possui uma ligação com o componente que fez a chamada do método, oferecendo uma interface cliente-servidor que possibilita interações com os usuários.

6.1.2.3 Content Providers

O Content Providers, ou provedores de conteúdos são interfaces utilizadas para compartilhar os dados entre aplicações, desta forma o Android executa cada aplicativo em sua própria *SandBox* para que todos os dados que pertencem a um aplicativo estejam isolados, e repassados através de aplicações por meio de *Intent* (GARGENTA, tradução nossa, 2011).

6.1.2.4 Broadcast Receivers

O Broadcast Receivers são componentes responsáveis pela resposta dos eventos do sistema ou eventos da aplicação, caso um aplicativo deseja receber e responder um evento global, como por exemplo, um toque de telefone, mensagem de texto recebida, deve ser registrado como um BroadcastReceiver (ABLESON et al, 2012, tradução nossa).

Para utilizar um broadcast receiver, é necessário implementar a interface BroadcastReceiver, que possui apenas o método onReceive(), o qual deve conter todo o código que deverá ser executado baseado na *intent* recebida (MURPHY, 2011, tradução nossa).

6.2 BLUETOOTH

O Bluetooth é um padrão de tecnologia para áreas de redes pessoais sem fio, ou seja, uma rede *Personal Area Network* (PAN), que serve para interligar

celulares, palmtops e outros dispositivos de uso pessoal em curtas distâncias (MORIMOTO, 2011). Conforme Harte (2004, tradução nossa), a tecnologia Bluetooth foi desenvolvida a fim de proporcionar comunicação sem fio entre dois ou mais dispositivos capazes de enviar e receber ondas de rádio de curto alcance.

A tecnologia Bluetooth opera sobre uma banda de radiofrequência denominada *Industrial, Scientific and Medicine* (ISM) utilizando a técnica de transmissão *Frequency Hopping Spread Spectrum* (FHSS) (PEDRALHO, 2005). Esta tecnologia é um padrão de comunicação com baixo consumo de energia e possui um raio de alcance de acordo com as características de cada aparelho, dividindo em classes de alcances como:

a) classe 1: alcance aproximadamente de 100 a 300 metros, com potência máxima de 100 mW;
b) classe 2: alcance aproximadamente de 10 a 33 metros, com potência máxima de 2.5 mW;
c) classe 3: alcance aproximadamente de 1 a 3 metros, com potência máxima de 1.0 mW.

O Bluetooth opera na faixa de frequência de 2,4 GHz, que é reservada pela Agência Nacional de Telecomunicações (ANATEL) para equipamentos de radiação restrita, que não precisam de licença (SVERZUT, 2008). Essa frequência é chamada de ISM, determinada de acordo com um acordo internacional para a utilização de dispositivos científicos, médicos e industriais.

6.2.1 Frequência

Para que seja possível que os vários dispositivos possam utilizar a mesma faixa do espectro de frequência, o transmissor realiza a mudança de 1.600 vezes por segundo o segmento da frequência que permite a transmissão das informações. Segundo SVERZUT (2008), dentro do intervalo que a ANATEL reserva de 2,4 GHz existem 79 canais diferentes que possuem um espaçamento de 1 MHz possibilitando a transmissão, e o canal que está transmitindo é alterado randomicamente para possibilitar a comunicação de vários dispositivos distintos.

A ISM foi empregada para que os dispositivos que emitam muita interferência de rádio não impeçam que as transmissões de dados não causem interferências em outras aplicações sensíveis a ruídos. Devido à faixa ser uma

frequência aberta, a ISM pode ser utilizado por quaisquer aplicações comerciais que troquem informações por wireless, podendo estas ser desde redes locais sem fio até sistemas de segurança de empresas. Com a evolução das versões do Bluetooth, alguns aspectos foram melhoras, como por exemplo, a transmissão.

6.2.2 Versões da tecnologia

Segundo Alecrim (2008), as versões do Bluetooth até o presente momento são:

a) Bluetooth 1.0: esta versão significa o inicio da tecnologia, sendo esta a versão lançada, e que possuía vários problemas que faziam com que o desenvolvimento se tornasse, difíceis e limitados para algumas marcas e modelos de específicos celulares;
b) Bluetooth 1.1: versão lançada oficialmente em fevereiro de 2001, firmando o Bluetooth como o padrão IEEE 802.15;
c) Bluetooth 1.2: com o lançamento desta versão em novembro de 2003, houve um destaque devido as conexões com maior velocidade, melhor proteção contra interferências externas, suporte aperfeiçoado aos *piconets* e evolução no processamento de voz;
d) Bluetooth 2.0: lançamento realizado em novembro de 2004, com o aperfeiçoamento na diminuição do consumo de energia, aumento na velocidade de transmissão de dados, correção a vários erros e uma melhor comunicação entre os dispositivos;
e) Bluetooth 2.1: lançamento em agosto do ano de 2007, possuindo como principal progresso o aumento de mais informações nos sinais de busca, permitindo uma seleção aperfeiçoada dos dispositivos antes de realizar a conexão, havendo também melhorias na segurança, criptografia e novamente aperfeiçoamento no consumo de energia;
f) Bluetooth 3.0: as principais características desta versão lançada em abril de 2009 são taxas de velocidades maiores durante a transferência de dados e aperfeiçoamento do consumo de energia;
g) Bluetooth 4.0: esta é a ultima versão disponível da tecnologia, lançada no final de 2009, tendo como aprimoramento o consumo de energia de

um modo inteligente, economizando ainda mais a energia durante a execução.

Todas as versões são compatíveis entre si, porém a velocidade é decidida de acordo com a velocidade do menor protocolo que está sendo utilizado. Com a evolução das versões da tecnologia Bluetooth, pode ser aplicada em diversos dispositivos, assim como é aplicada a tecnologia recente Beacon.

6.3 IBEACON

O iBeacon é um protocolo padronizado pela Apple, que utiliza os serviços de localização, fazendo com que emita um alerta com a localização de um dispositivo iBeacon (APPLE,2015, tradução nossa).

Vários fornecedores utilizam essa tecnologia e expandem a tecnologia destes transmissores. Essa tecnologia utiliza o *Bluetooth Low Energy,* que é uma tecnologia de rede de área pessoal utilizada para realizar transmissões de dados em pequenas distancias projetado para ter um baixo consumo de energia e de custos, além de manter um alcance de comunicação semelhante ao *Bluetooth Classic*, tendo como alcance 100 metros para realizar a descoberta do dispositivo iBeacon (IBEACON, 2015, tradução nossa).

A Apple realizou a padronização do formato *Bluetooth Low Energy*, sendo assim um pacote de dados transmitido possui quatro principais informações:

a) UUID: cadeia de 16 *bytes* utilizados para diferenciar os iBeacons;
b) maior: cadeia de 2 bytes utilizados para distinguir um subconjunto de iBeacons;
c) menor: sequência de 2 bytes, destinados a identificar os iBeacons individuais;
d) taxa Potência: utilizado para determinar a proximidade a partir do iBeacon.

Para que os *smartphones* que utilizam o sistema operacional Android possam ler os sinais emitidos dos iBeacons, devem possuir uma estrutura mínima de utilização da versão do Android 4.3, como por exemplo os dispositivos: Samsung Galaxy S3 / S4 / S4 Mini, Samsung Galaxy Note 2/3, HTC One, Google / LG Nexus 7 2013 versão / Nexus 4/5 Nexus, HTC Borboleta, OnePlus One.

7 TRABALHOS CORRELATOS

Devido à necessidade de utilização de aplicações web confiáveis, com tecnologias robustas e flexíveis, e da facilidade de possuir um aplicativo na plataforma Android integrado ao sistema web com a utilização de web services, para facilitar a comunicação dos usuários finais com o sistema web, são efetuadas diversas pesquisas para auxiliar no desenvolvimento, sendo assim esta seção tem como objetivo apresentar os trabalhos relacionados com a presente pesquisa.

7.1 ESTUDO DE TECNLOGIAS PARA O DESENVOLVIMENTO DE APLICAÇÃO WEB PARA GERENCIAMENTO DE CORRETORA DE SEGUROS

Em 2010, foi desenvolvido pelo acadêmico Gabriel Casagrande Stabel para o curso de Engenharia de Computação para a Universidade Federal do Rio Grande do Sul, uma pesquisa que visava ressaltar a importância da utilização de tecnologias voltadas para a implementação de um sistema web complexo e rico (STABEL, 2014).

Para aplicar os estudos das novas tecnologias como o AngularJS, foi implementado uma aplicação web para o gerenciamento de uma corretora de seguros, com foco no potencial que a utilização destas tecnologias agregam no desenvolvimento, satisfazendo as necessidades propostas (STABEL, 2014).

7.2 COMPARAÇÃO DE DESEMPENHO E CONSUMO DE MEMÓRIA ENTRE FRAMEWORKS DE MAPEAMENTO OBJETO-RELACIONAL JAVA HIBERNATE E ECLIPSELINK

Na Universidade do Extremo Sul Catarinense de Santa Catarina foi desenvolvido o projeto de pesquisa para o curso de Ciência da Computação, pelo acadêmico Juliano Marques, cujo objetivo era comparar duas ferramentas que utilizam o Mapeamento Objeto-Relacional, o Hibernate e o EclipseLink, sendo que o parâmetros de comparação utilizados foram os critérios de desempenho e consumo de memória e avaliação do custo-benefício (MARQUES, 2012).

Para realizar as comparações foi desenvolvido dois protótipos para a execução dos testes, um utilizando a tecnologia do Hibernate e outro com a

tecnologia do EclipseLink, sendo executados os testes de inserções, edições, remoções e consultas de registros na base de dados. Os resultados das execuções dos testes alegaram que a utilização do framework Hibernate teve um alto desempenho em relação ao EclipseLink e menor consumo de memória (MARQUES, 2012).

7.3 SISTEMA DE CONCESSÃO DE FINANCIAMENTO PARA A APRESENTAÇÃO DE ARTIGOS ACADÊMICOS EM SPRING MVC

No ano de 2011, na Universidade Federal do Estado do Rio de Janeiro, o acadêmico Rafael Fogel, desenvolveu um projeto de pesquisa para o Centro de Ciências Exatas e Tecnologia, para a Escola de Informática Aplicada, um trabalho para utilizar a tecnologia do Spring MVC, aplicando os conceitos estudados na implementação de um sistema de concessão de financiamento para a apresentação de artigos acadêmicos.

Para a realização do trabalho, foi estudada a tecnologia Spring MVC, por se tratar de um framework de desenvolvimento em ascensão no mercado de trabalho, e também por ser considerado um dos frameworks mais completos em Java, além de ser recomendado para qualquer projeto, também possui a característica de trabalhar através de anotações, simplicando e reduzindo os códigos da aplicação (FOGEL, 2011).

7.4 DESENVOLVIMENTO DE APLICAÇÕES GOOGLE ANDROID E INTEGRAÇÃO COM WEB SERVICE

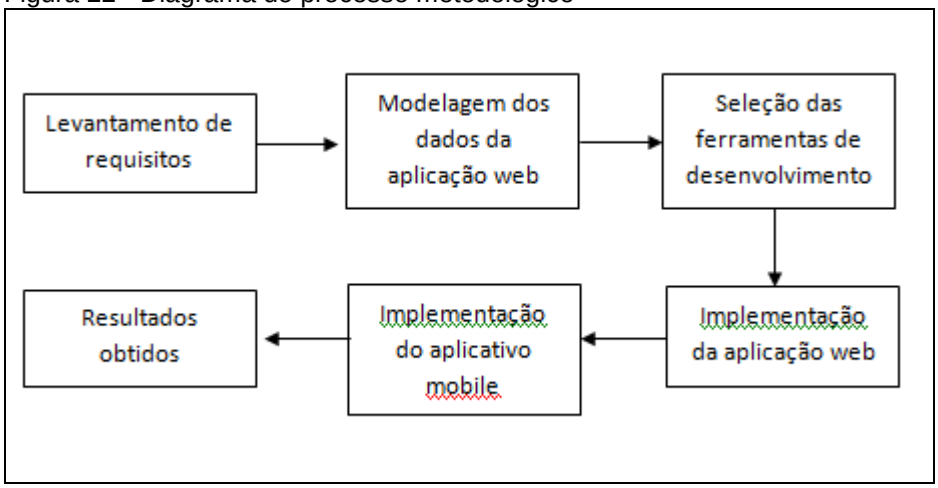
No Instituto Municipal de Ensino Superior de Assis, o acadêmico Vinicius José Silva dos Santos, desenvolveu um projeto de pesquisa no ano de 2012, um trabalho de conclusão de curso, cujo objetivo era o aprendizado sobre o conhecimento de web service, integrando essa tecnologia com um protótipo desenvolvido para a plataforma Android.

Após o levantamento bibliográfico para compreender os conceitos de web services e sua implementações, tanto na parte de construção do serviço como na utilização do *client* no aplicativo mobile (SANTOS, 2012).

8 DESENVOLVIMENTO DO SISTEMA

O processo metodológico da pesquisa desenvolvida é composta pelo levantamento de requisitos, modelagem de dados, seleção das ferramentas de desenvolvimento, implementação da aplicação *web* e aplicativo *mobile*, e por fim, a análise e escrita dos resultados obtidos através da construção da aplicação *web*.

Figura 22 - Diagrama do processo metodológico



Fonte: Do autor.

Com o início do desenvolvimento *web*, utilizou se os *frameworks* Hibernate, Spring MVC e AngularJS, aplicando o baseamento teórico estudado para a construção da aplicação *web* para um agendamento de salão beleza, com o principal propósito de utilizar as novas tecnologias para a construção de um sistema web flexível, robusto, confiável e de fácil manutenção, assuntos estes abordados e estudados no levantamento bibliográfico.

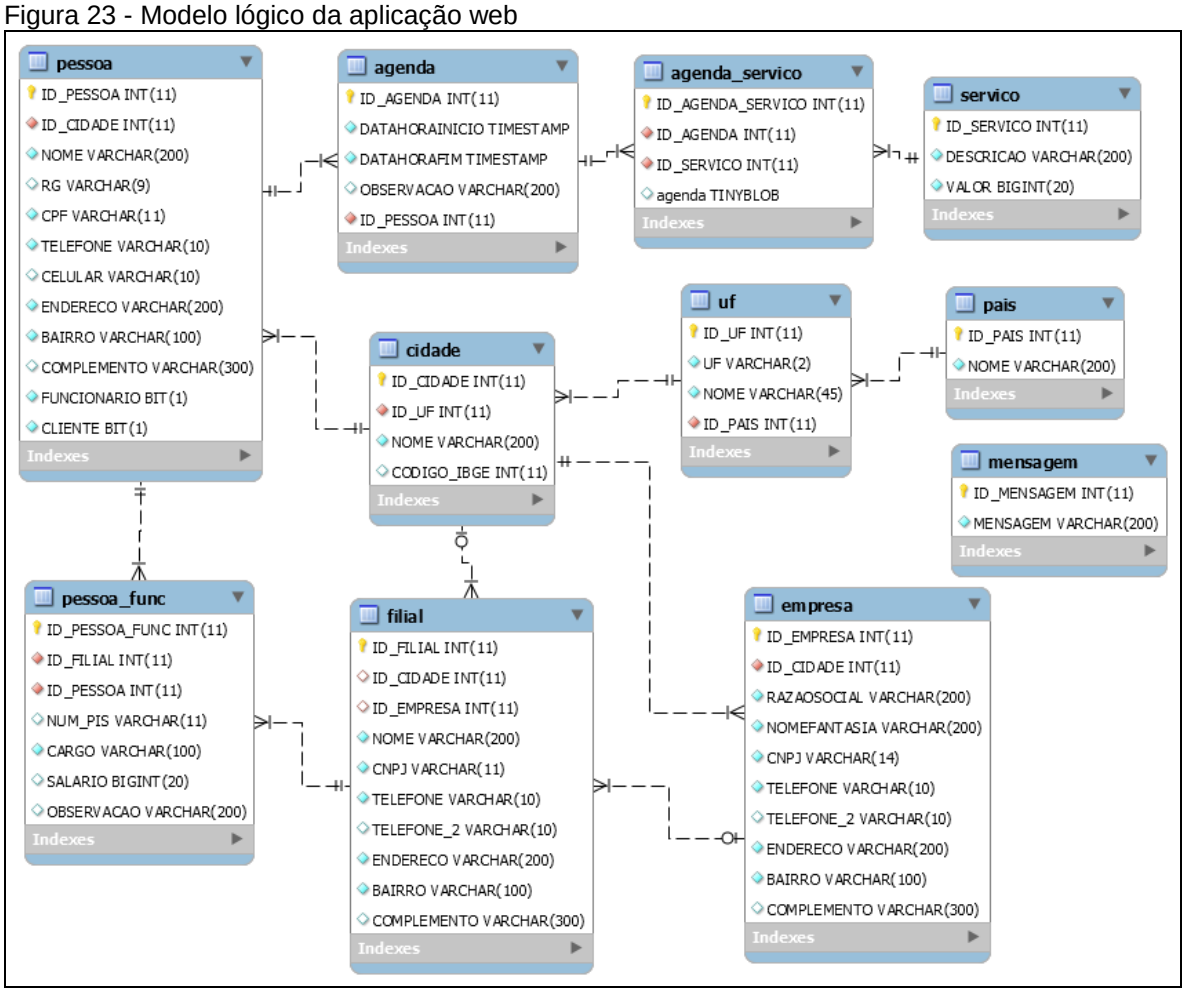
8.1 MODELAGEM DE DADOS

Realizou se um estudo sobre agendamentos de salões de beleza, para analisar quais são os requisitos necessários, para o desenvolvimento de uma aplicação web, que posa suprir todas as necessidades de um agendamento para um salão de beleza.

A modelagem de dados se torna um requisito importante para a construção de uma ferramenta, devido à demonstração de características de funcionamento e comportamento do *software*, facilitando a compreensão do projeto

e diminuindo os erros de programação, projeto e funcionamento, sendo assim, conforme a figura 23 exibe o modelo lógico da aplicação *web*:

Figura 23 - Modelo lógico da aplicação web



Fonte: Do autor.

O modelo lógico serve para mostrar as ligações entre as tabelas de banco de dados, as chaves primárias e demais componentes.

8.2 FERRAMENTAS DE DESENVOLVIMENTO

Após a definição da modelagem de dados, esta etapa demonstra a definição das ferramentas de desenvolvimento, que ficou determinado para a implementação da aplicação *web* o *software* Eclipse IDE for Java Developers na versão Luna Service Release 2 (4.4.2), juntamente com o servidor Tomcat v7.0 Server, e para o armazenamento dos dados, o banco MySQL Server 5.6, e utilizando o MySQL Workbench como ferramenta de administração do banco de dados.

Para realizar a integração dos três *frameworks*: Hibernate, Spring MVC e AngularJS, foram utilizadas as seguintes versões:

a) Hibernate: 3.6.10.Final

b) Spring MVC: 4.1.0.RELEASE

c) AngularJS: 1.3.15

Para o *design*, utilizou se o *framework* Bootstrap que contém modelos de *design* baseados em tipografia, formas, botões, navegação e outros componentes da *interface*, sendo então utilizada a versão 3.3.2, juntamente com a versão do *framework* jQuery 2.1.4, que deve ser utilizado pois alguns *plugins* dependem deste *framework*. Para a criação do calendário do agendamento, utilizou se o *framework DayPilot for JavaScript* da versão 8.0.1658 Trial.

No desenvolvimento da aplicação *mobile,* utilizou se a ferramenta Android Studio 1.2.1.1, empregando a utilização da implementação de acordo com o referencial bibliográfico, usando o projeto biblioteca *Android-View-Week* para a construção do calendário de agendamento, e a biblioteca SDK do Estimote para realizar a conexão e procura dos Beacons.

8.3 CONSTRUÇÃO DA FERRAMENTA WEB E MOBILE

Posteriormente com as ferramentas definidas, versões dos frameworks e as bibliotecas que serão utilizados no desenvolvimento da ferramenta de agendamento tanto na *web* como *mobile*, deve ser dado inicio a criação do projeto e a integração dos frameworks para criar uma aplicação robusta e flexível, assim como a criação da aplicação *mobile*.

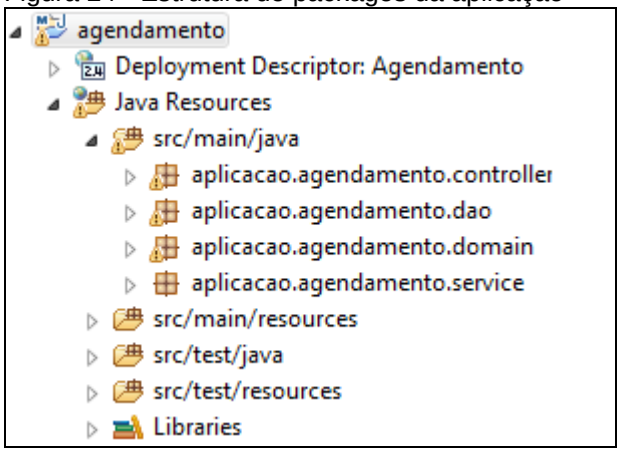
8.3.1 Estrutura inicial do projeto da aplicação web

A criação do projeto deve ser realizada com a utilização do Eclipse, utilizando a opção de criar um projeto Maven, que simplifica o gerenciamento de dependências, e a criação e distribuição dos projetos. Para a utilização do Maven, deve ser realizado o download no site oficial e posteriormente descompactar na pasta no diretório que deverá ser instalado, e logo após o caminho da pasta deverá ser adicionado nas variáveis de ambiente.

Com a criação do projeto, irá ser criado também o arquivo “*pom.xml*”, que é considerado o coração do projeto, pois neste arquivo será adicionado as dependências do projeto, como por exemplo, as dependências do Spring MVC e do Hibernate, e que também contém a centralização da documentação e principalmente o modo de compilação e distribuição da aplicação.

Os arquivos Java foram divididos em quatro *packages* conforme a figura 24.

Figura 24 - Estrutura de packages da aplicação



Fonte: Do autor.

As *packages* contém as classes Javas, e são divididas com os controladores na *package aplicacao.agendamento.controller*, as classes de modelos devem estar contidas na *package aplicacao.agendamento.domain* que deve possuir os arquivos POJOs que são responsáveis pelos mapeamentos dos objetos com as tabelas e os campos do banco de dados, além de possuir as associações e subclasses persistentes. As classes que realizam o acesso aos serviços e os arquivos para acessar o banco de dados, devem estar nas *packages aplicação.agendamento.service* e *aplicação.agendamento.dao*.

Os arquivos *web* por padrão devem estar contidos na pasta *webapp*, contendo outras pastas como configurações do Spring MVC, as propriedades do JDBC, os arquivos estáticos, além de conter também os arquivos JavaScript utilizados com o framework *AngularJS*. Posteriormente com a estrutura do projeto criada e devidamente distribuída, deverá ser iniciado o processo de configuração do Spring MVC na aplicação.

8.3.2 Configuração Spring MVC

Para realizar a configuração do Spring MVC na aplicação, é necessário criar o arquivo web.xml, que contém a configuração normal de *Servlet*, conforme figura 25.

Figura 25 - Configuração do Spring MVC

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="stsm" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

    <display-name>Agendamento</display-name>

    <session-config>
        <session-timeout>60</session-timeout>
    </session-config>

    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>index.web</welcome-file>
    </welcome-file-list>
</web-app>
```

Fonte:Do autor.

Na figura 25**,** possui a configuração do *servlet-class*, e também é definido o arquivo de configuração do *framework* Spring, o *spring-servlet*.

A configuração do framework Spring é feito através do XML *spring-servlet*, que contém diversas informações, e inicialmente contém a configuração para habilitar o uso de anotações do Spring MVC, além de configurar o pacote base da aplicação conforme a figura 26.

Figura 26 - Configuração do pacote base da aplicação

```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:p="http://www.springframework.org/schema/p" xmlns:tx="http://www.springframework.org/schema/tx"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc
                        http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd
                        http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                        http://www.springframework.org/schema/context
                        http://www.springframework.org/schema/context/spring-context-3.0.xsd
                        http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd">


    <mvc:annotation-driven />
    <context:component-scan base-package="aplicacao.agendamento" />
```

Fonte: Do autor.

Conforme a figura 26 é necessário configurar o pacote para que o Spring detecte aonde se encontra as classes, neste arquivo também é possível realizar a configuração com o banco de dados, o *bean* “sessionFactory” e o *bean* “transactionManager” para realizar a configuração do Spring MVC com o Hibernate para ter acesso ao banco de dados.

8.3.3 Configuração do Spring MVC com o Hibernate para acesso ao banco de dados

No arquivo de configuração do Spring MVC, também contém a configuração do Hibernate com o banco de dados, que é feita através das seguintes configurações da figura 27.

Figura 27 - Configuração do banco de dados do Hibernate no Spring MVC

```
<bean id="dataSource"
    class="org.springframework.jdbc.datasource.DriverManagerDataSource"
    p:driverClassName="${jdbc.driverClassName}"
    p:url="${jdbc.databaseurl}" p:username="${jdbc.username}" p:password="${jdbc.password}" />

<bean id="sessionFactory"
    class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean"
    p:packagesToScan="aplicacao.agendamento.domain">
    <property name="dataSource" ref="dataSource"></property>
    <property name="configLocation">
        <value>classpath:hibernate.cfg.xml</value>
    </property>

    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
            <prop key="hibernate.show_sql">true</prop>
        </props>
    </property>
</bean>
```

Fonte: Do autor.

O arquivo contém diversos mapeamentos do Spring, como o *Bean PropertyConfigurer*, que é utilizado para carregar o arquivo *jdbc.properties*, que contém as propriedades e detalhes de conexão do banco de dados que são utilizados para realizar a conexão com o Hibernate.

Figura 28 - Arquivo com as propriedades do JDBC

```
jdbc.driverClassName= com.mysql.jdbc.Driver
jdbc.dialect=org.hibernate.dialect.MySQLDialect
jdbc.databaseurl=jdbc\:mysql\://localhost\:3306/agendamento
jdbc.username=root
jdbc.password=Daiana123.
```

Fonte: Do autor.

O *Bean DataSource* é utilizado como a fonte de dados Java, para se conectar ao banco de dados, através da classe do driver JDBC, URL de conexão, dialeto, nome de usuário e senha. O *Bean SessionFactory* é usado para definir as diversas configurações do Hibernate, que estão no arquivo hibernate.cfg.xml.

Figura 29 - Configurações do Hibernate

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property name="hibernate.connection.pool_size">1</property>

        <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <property name="show_sql">true</property>

        <property name="packagesToScan"> aplicacao.agendamento.domain</property>
    </session-factory>
</hibernate-configuration>
```

Fonte: Do autor.

Neste arquivo hibernate.cfg.xml, contém quais são as classes que deverão ser visualizadas com as anotações de entidades. Por fim, o *Bean TransactionManager*, é o gerenciador de transações do Hibernate, que é ligado através da propriedade *sessionFactory*, com o *Bean SessionFactory.*

8.3.4 Mapeamento dos objetos

O Hibernate oferece duas técnicas para criar os mapeamentos dos objetos com os modelos da base de dados, podendo ser criados arquivos XML, ou através de anotações nos objetos POJO.

Na construção da ferramenta de agendamento, utilizou se o mapeamento através das anotações nas classes POJO, devido à facilidade de compreensão do código fonte em comparação com o mapeamento realizado com arquivos XML.

As classes POJO foram criadas utilizando as anotações JPA, que estão contidas no pacote *javax.persistence*, conforme a figura 30.

Figura 30 - Classe POJO cidade

```
package aplicacao.agendamento.domain;

import java.util.Set;

@Entity
public class Cidade implements java.io.Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @Column(name = "id_cidade")
    private int idCidade;

    private String nome;

    @Column(name = "codigo_ibge")
    private Integer codigoIbge;

    @ManyToOne
    @JoinColumn(name = "id_uf")
    private Uf uf;

    @OneToMany(mappedBy = "cidade")
    @JsonIgnore
    private Set<Filial> filiais;

    @OneToMany(mappedBy = "cidade")
    @JsonIgnore
    private Set<Empresa> empresas;

    public Cidade() {
    }

    //getters e setters
```

Fonte: Do autor.

Ao criar a classe com os atributos privados e com seus *getters* e *setters* públicos, é necessário a adição das anotações, inicialmente é imprescindível a utilização da anotação *@Entity*, que define a classe como um *bean* de entidade, tornando então uma entidade persistente, caso o nome o nome da classe, seja igual ao nome da tabela do banco de dados, não irá precisar adicionar a anotação *@Table*, caso contrário, deverá ser utilizado essa anotação, informando qual é o nome da tabela no banco de dados.

Nas classes POJO deve ser indicado qual coluna é a chave-primária através da anotação *@Id*, e caso algum atributo tenha o nome diferenciado da coluna do banco de dados, deve ser adicionado à anotação *@Column* informando o valor correto.

Para mapear as associações é necessário fazer a ligação através das associações *@ManyToOne* e com a anotação *@JoinColumn* definir qual coluna fará a ligação entre as associações.

Após criar as classes de modelos, é preciso criar os arquivos que irão realizar os acessos ao banco de dados, que são definidos como classes *Data Access Object* (DAO), que possuem como as principais funções de obter as conexões, realizar o mapeamento dos objetos Java para os tipos de dados SQL e executar comando SQL, de acordo com as requisições solicitadas e a necessidade de manipulação dos dados.

8.3.5 Requisições no Spring MVC

As classes DAO, possuem a anotação do Spring *@Repository*, que serve para indicar ao *framework* que são classes da camada de persistência e que funcionará como repositório de dados, conforme a figura 31.

Figura 31 - Requisições no Spring MVC

```
@Repository
public class FilialDao {

    @Autowired
    SessionFactory sessionFactory;

    public void save(Filial filial) {
        sessionFactory.getCurrentSession().save(filial);
    }

    @SuppressWarnings("unchecked")
    public List<Filial> findAll() {
        return sessionFactory.getCurrentSession().createQuery("from Filial")
                .list();
    }

    public Filial get(Integer id) {
        Filial filial = (Filial) sessionFactory.getCurrentSession().get(Filial.class, id);
        return filial;
    }

    public void delete(Integer id) {
        Filial filial = get(id);
        sessionFactory.getCurrentSession().delete(filial);
    }
}
```

Fonte: Do autor.

Com a criação das classes de acesso ao banco de dados, é necessário criar as classes que irão possuir os serviços que irão realizar as chamadas das

funções que contém nas classes DAO para trazer as informações do banco de dados ou realizar alguma inserção, alteração ou remoção de dados.

Para criar uma classe de serviço é indispensável a utilização da anotação do *@Service* do Spring MVC, para que a classe seja detectada como classes da camada de serviço. Na implementação da classe de serviço, são criadas as funções que fazem a chamada das funções das classes DAO, possuindo a anotação *@Transactional*, que determina que essa função seja uma transação, conforme a figura 32.

Figura 32 - Classe de serviço da filial

```
@Service("filialService")
public class FilialService {

    @Autowired
    FilialDao filialDao;

    @Transactional(propagation = Propagation.REQUIRED, readOnly = false)
    public void save(Filial filial) throws Exception {
        filialDao.save(filial);
    }

    @Transactional(propagation = Propagation.REQUIRED, readOnly = false)
    public List<Filial> findAll() {
        return filialDao.findAll();
    }

    @Transactional(propagation = Propagation.REQUIRED, readOnly = false)
    public void delete(Integer id) {
        filialDao.delete(id);
    }

    @Transactional(propagation = Propagation.REQUIRED, readOnly = false)
    public Filial get(Integer id) {
        return filialDao.get(id);
    }

}
```

Fonte: Do autor.

Com as classes de acesso ao banco de dados e o serviços que fazem essas chamadas, é necessário criar os controladores que são responsáveis pelo envio e respostas aos usuários. Assim como as demais classes, o *controller* deve conter a anotação *@Controller*, para que o Spring MVC detecte que são classes em nível de camada de apresentação.

Os *controllers* irão conter as requisições que são anotadas por *@RequestMapping*, contendo as propriedades *value* que irá definir a requisição, a propriedade *method* indica qual o tipo de pedido HTTP que está sendo realizado,

pois poderá ser uma requisição com o método*: GET, POST, PUT ou DELETE*, além de conter outras propriedades que podem ser definidas no mapeamento da requisição, como por exemplo, como será produzida a resposta que poderá ser através de um JSON conforme a figura 33.

Figura 33 - Controller do Spring MVC

```
package aplicacao.agendamento.controller;

import java.util.List;

@RestController
@Controller
public class AgendaController {

    @Autowired
    private AgendaService agendaService;

    @Autowired
    public AgendaController(AgendaService agendaService) {
        this.agendaService = agendaService;
    }

    @RequestMapping(value = "/agenda/listar", method = RequestMethod.GET, produces = {
            "application/xml", "application/json" })
    public @ResponseBody List<Agenda> getListAgendas() {
        return agendaService.findAll();
    }

    @RequestMapping("/agenda/webServiceListar")
    public List<Agenda> getListAgendasWebService() {
        return agendaService.findAll();
    }

    @RequestMapping(value = "/agenda/salvar", method = RequestMethod.POST, headers = "Accept=application/json")
    public @ResponseBody void salvar(@RequestBody Agenda agenda)
            throws Exception {
        agendaService.save(agenda);
    }
```

Fonte: Do autor.

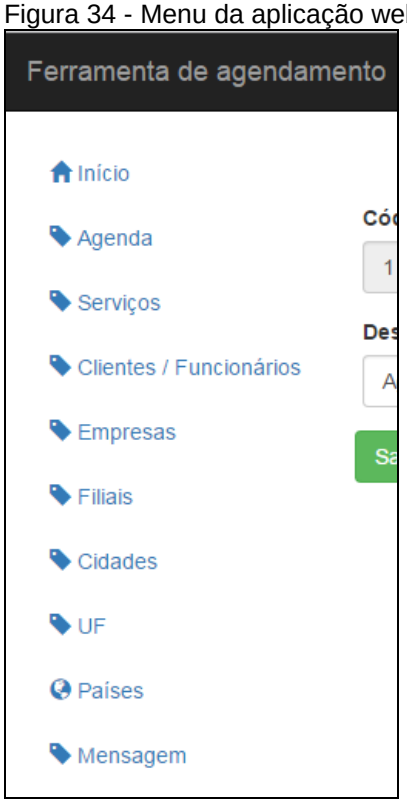
Posteriormente com a criação dos *controllers* que possuem todas as requisições mapeadas, já é possível iniciar o desenvolvimento das telas que irão realizar as chamadas desses *controllers*, através dos *controllers* do *framework* AngularJS.

8.3.6 Desenvolvimento do Front-End

Com as requisições implementadas é necessário iniciar o desenvolvimento das telas utilizando o *framework* AngularJS, que irá realizar as chamadas dessas requisições de acordo com cada situação. Os arquivos HTML foram divididos em pastas de acordo com cada função, sendo que os cadastros possuem arquivos de lista, cadastro e o arquivo JavaScript que contem os *controllers* e configuração de rotas.

Para o desenvolvimento *front-end*, a página inicial possui a diretiva *ng-app*, que é utilizada para definir o nome do módulo da aplicação, e a diretiva *ng-view* que será utilizada pelas rotas do AngularJS. A página possui um menu lateral que possui ligação com os cadastros da ferramenta de agendamento, e utilizou se o conceito de rotas do AngularJS.

Figura 34 - Menu da aplicação web



Fonte: Do autor.

Utilizou se o conceito de rotas para acessar as funções da aplicação, devido à facilidade de manipulação de partes do código HTML, de forma independente, além de separar as camadas da aplicação, tornando a mais gerenciável.

Para utilizar as rotas do AngularJS na aplicação, é necessário fazer utilização do serviço *$routeProvider*, e realizar a configuração do modulo através do método *config* conforme figura 35.

Figura 35 - Arquivo de *controller* do JavaScript

```
'use strict';

angularEmpresa = angular.module('agendamento.empresa', [ 'ngRoute' ]);

angularEmpresa.config([ '$routeProvider', '$locationProvider', function($routeProvider) {
    $routeProvider
    .when('/empresa', {
        templateUrl : 'static/empresa/lista.html',
        controller : 'EmpresaController'
    })
    .when('/empresa/novo', {
        templateUrl : 'static/empresa/cadastro.html',
        controller : 'EmpresaController'
    })
} ]);
```

Fonte: Do autor.

Sendo então necessário informar qual será o *template* que deverá ser adicionado na *tag ng-view* e *controller* do respectivo HTML, sendo que a configuração da rota e o *controller* como na figura 35, estão criados no mesmo arquivo JavaScript do determinado cadastro, para que exista uma separação lógica na aplicação *web*.

Os *templates* criados para as listas possuem como padrões os botões de Novo e Listar, e logo abaixo a lista com as funções de editar ou remover o registro conforme a figura 36.

Figura 36 - Tela de listagem das empresas

Fonte: Do autor.

Os botões “Novo” e “Editar” possuem a *tag ng-click* para chamar as suas respectivas funções que estão contidas no *controller* da lista, que está definido de acordo com a diretiva *ng-controller*. Ao clicar no botão “Novo”, de acordo com a rota criada, irá ser inserido o *template* do cadastro, que irá realizar as chamadas para o servidor para salvar o registro, e então utilizar as requisições criadas com o Spring MVC.

Para realizar a comunicação do cliente com o servidor, é utilizado o conceito RESTful. O RESTful é uma comunicação HTTP, que possui um padrão simples, que utiliza os cabeçalhos como POST, GET, PUT e DELETE (Documentação AngularJS).

A comunicação RESTful está contida no *controller* de cada *template*, e é utilizada de acordo com cada função específica. Para utilizar é necessário utilizar a variável $http que é repassada pelo parâmetro no *controller*, e o *framework AngularJS* se encarrega de injetar o serviço HTTP na variável conforme a figura 37.

Figura 37 - Arquivo JavaScript do controller da Empresa

```
angularEmpresa.controller('EmpresaController', [ '$scope', '$http', function($scope, $http) {
    $scope.listarEmpresas = function() {
        $http.get("empresa/listar").success(function(response) {
            $scope.empresas= response;
        });
    };

    $scope.save = function() {
        $http.post("empresa/salvar", $scope.empresa).success( function(response){
            $scope.empresa = angular.copy($scope.master);
        });
    };
    //...
}]);
```

Fonte: Do autor.

As requisições para listar todas as empresas, é utilizada através do método *get* do $http, e repassando a resposta para o $scope.empresas que contém na lista de empresas, pois esta função é chamada através do botão “Listar” e a diretiva *ng-repeat* da tabela, se encarrega de listar os registros.

Destaca se então que para realizar as chamadas ao servidor é utilizado os serviços HTTP do AngularJS, de acordo com cada necessidade, como por exemplo, para buscar os horários específicos de cada agendamento, para popular a agenda criada através do framework DayPilot.

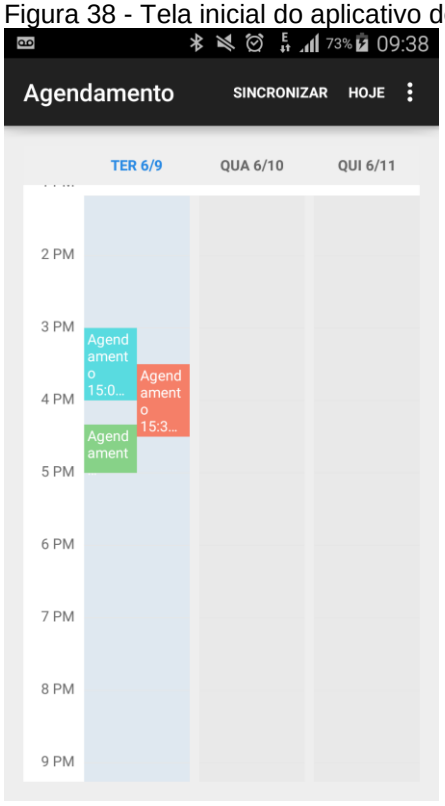
Com a aplicação *web* construída com as funções de criar o agendamento, é possível realizar a sincronização com o aplicativo mobile, utilizando das funções de sincronizar no aplicativo.

8.3.6 Desenvolvimento Mobile

O desenvolvimento do aplicativo *mobile* se realizou utilizando a ferramenta Android Studio, compilando o projeto da biblioteca *Android Week View* para fazer a utilização do calendário com os eventos marcados e a utilização do SDK do *Estimote Beacon* para realizar a conexão com o dispositivo iBeacon.

Na tela inicial da aplicação, possui a opção de “Sincronizar” que realiza a chamada da requisição do *RestController* da Agenda, criado com o *framework* Spring MVC, retornando o arquivo JSON, para ser lido através do método “lerJsonAgenda”, que faz a leitura dos dados e retorna para a lista de eventos sincronizados com o calendário conforme a figura 38.

Figura 38 - Tela inicial do aplicativo de agendamento.



Fonte: Do autor.

Para realizar a sincronização, foi criada função “sincronizarAgenda” que executa a classe “lerJsonAgenda”, que é uma classe AsyncTask, por ser uma tarefa

assíncrona, passando como parâmetro a URL do *web service* que irá retornar o arquivo JSON com os dados. Após ter feito a conexão com a URL e o download do arquivo JSON, é executada a função “onPostExecute” para ler os dados e adicionar nos eventos do calendário.

Figura 39 – Função para ler os dados JSON.

```
protected void onPostExecute(String result) {
    try {
        JSONArray jsonObject = new JSONArray(result);

        int lengthJsonArr = jsonObject.length();
        eventosSincronizados.clear();
        for (int i = 0; i < lengthJsonArr; i++) {
            JSONObject jsonChildNode = jsonObject.getJSONObject(i);

            Calendar startTime = Calendar.getInstance();
            Long datahorainicio = (jsonChildNode.optLong("datahorainicio"));
            Timestamp stamp = new Timestamp(datahorainicio);
            Date date = new Date(stamp.getTime());
            startTime.setTime(date);

            Calendar endTime = Calendar.getInstance();
            Long datahorafim = (jsonChildNode.optLong("datahorafim"));
            Timestamp stampFim = new Timestamp(datahorafim);
            Date dateFim = new Date(stampFim.getTime());
            endTime.setTime(dateFim);

            WeekViewEvent event = new WeekViewEvent(1, getEventTitle(startTime), startTime, endTime);
            event.setColor(getResources().getColor(R.color.event_color_01));
            event.setId(jsonChildNode.optInt("idAgenda"));

            eventosSincronizados.add(event);
            retornarEventosSinc = true;
        }
        mWeekView.goToToday();
    } catch (Exception e) {
        Log.d("LeituraDadosAgenda", e.getLocalizedMessage());
    }
}
```

Fonte: Do autor.

Conforme a figura 39 é realizada leitura dos dados através da interação de cada nó do arquivo JSON, e então capturando as devidas informações e adicionando na lista de eventos sincronizados.

Com a aplicação rodando é possível realizar a localização do dispositivo Beacon, através da implementação utilizando o SDK do Estimote, configurando o UUID que deverá ser localizado na determinada região. Com a implementação do método de conexão “beaconStart”, é possível informar ao usuário que está dentro da região do Beacon, informando uma mensagem configurada na aplicação *web* e recebida através do *web service* criado pelo Spring MVC.

A classe “BeaconsMonitoringService” é responsável por executar as funções que localizam o iBeacon. Inicialmente é definido a região do iBeacon, configurando pelo UUID do dispositivo, conforme na figura 40.

Figura 40 – Definição das variáveis para definir a região do iBeacon.

```
private static final String ESTIMOTE_PROXIMITY_UUID = "e2c56db5-dffb-48d2-b060-d0f5a71096e0";
private static final Region ALL_ESTIMOTE_BEACONS = new Region(ESTIMOTE_PROXIMITY_UUID, null, null);
```

Fonte: Do autor.

Ao criar a classe, é criado o gerenciador do iBeacon, e realizada a conexão e definição da função que irá realizar o monitoramento do dispositivo.

Figura 41 – Função para conectar com o iBeacon.

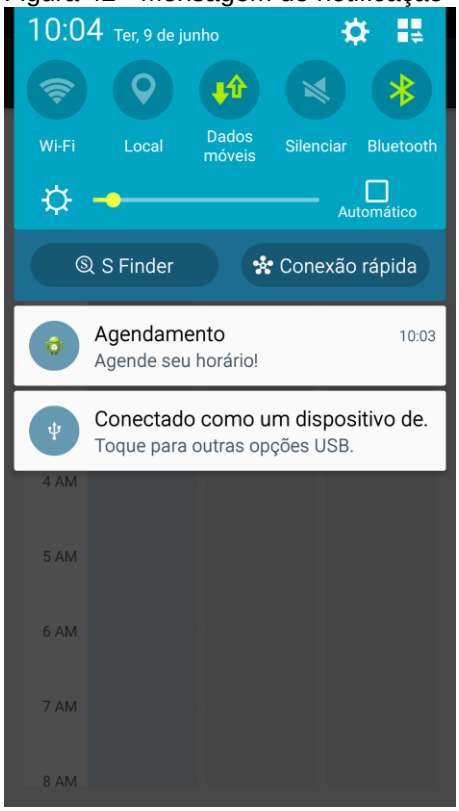
```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    beaconManager.connect(() -> {
            try {
                beaconManager.startRanging(ALL_ESTIMOTE_BEACONS);
            } catch (RemoteException e) {
                Log.e(TAG, "Não foi possível iniciar o monitoramento", e);
            }
    });

    beaconManager.setRangingListener(new BeaconManager.RangingListener() {
        @Override
        public void onBeaconsDiscovered(Region region, final List<Beacon> beacons) {
            if ((beacons.size() > 0) && (!foundBeacon)) {
                if (verificarConexao()) {
                    sincronizarMensagem();
                }
                else {
                    showNotification("Agende seu horário");
                }
                foundBeacon = true;
            } else if (beacons.size() == 0) {
                foundBeacon = false;
            }
            Log.e(TAG,"found" + foundBeacon);
        }
    });
    return START_STICKY;
}
```

Fonte: Do autor.

Na função “onStartCommand”, ao definir a função que irá monitorar o dispositivo, é verificado se há conexão do aparelho móvel com a *internet*, caso haja conexão é realizado então o procedimento de realizar a comunicação com o *web service* responsável por retornar a mensagem pré-configurada pela aplicação *web,*

utilizando um método semelhante a sincronização da agenda, caso não exista conexão com a *internet,* o aplicativo emite uma mensagem padrão.

Figura 42 - Mensagem de notificação



Fonte: Do autor.

A mensagem de notificação conforme a figura 42 é inserida de acordo com o formulário na aplicação *web* na figura 43.

Figura 43 - Tela de inserção de mensagem ao usuário

Fonte: Do autor.

Realizando a configuração da mensagem na aplicação *web*, faz com que haja uma interação entre o aplicativo *mobile* e a aplicação *web*, e que isso retorne ao usuário devido à aproximação do local o qual o Beacon está distribuindo os sinais.

8.4 RESULTADOS OBTIDOS

Com o desenvolvimento do projeto de pesquisa deste trabalho, pôde se obter a aplicação do estudo para realizar a integração dos *frameworks* Hibernate e Spring MVC com AngularJS na construção de uma ferramenta de agendamento, juntamente com a aplicação dos estudos da plataforma Android e integração com o dispositivo Beacon.

Na integração dos *frameworks* para aplicar os estudos realizados, pode se obter uma aplicação com diversas vantagens de criação, como com a utilização do Hibernate a facilidade de desenvolvimento e a fácil manutenção do software devido à utilização das especificações JPA, tornando mais dinâmico a tarefa de tratar os acessos aos dados, além de abstrair o banco de dados para que não haja a

preocupação na criação e manutenção das tabelas no banco, devido ao mapeamento realizado no Hibernate. Obteve uma velocidade no desenvolvimento da aplicação, devido a não necessidade de reescrever códigos SQL, pois o Hibernate realiza o encapsulamento de todo o código.

Baseando a aplicação com o *framework* Spring MVC, obteve um aproveitamento das injeções de dependências, devido ao container que se encarrega de instanciar as classes Java e definir suas dependências, possuindo uma arquitetura baseada em POJOs e *interfaces*, além de realizar a navegação através de anotações.

Com a integração do AngularJS para realizar o desenvolvimento *front-end* da aplicação, obteve se a vantagem que a renderização da página da aplicação é realizada diretamente no lado do cliente, diretamente no navegador do usuário, além de aplicar o conceito de *single-page-application* que também diminui o trafego de dados entre o cliente e servidor, aumentando a perfomance e a experiência do usuário na utilização do sistema. Com a utilização do AngularJS, agregou se um alto valor na aplicação, devido a utilização das melhores práticas com o padrão MVC e injeção de dependências.

Devido a utilização do *framework* Spring MVC na aplicação *web,* pode ser utilizar o *web service* criado para realizar a integração entre a aplicação *web* e o aplicativo *mobile*, agregando mobilidade aos usuários do aplicativo, para que possam realizar as consultas dos agendamentos realizados no salão e analisar os horários vagos para realizar o agendamento de acordo com a data desejada. Com o estudo da plataforma Android juntamente com o dispositivo Beacon, pode se aplicar o referencial bibliográfico para que o aplicativo *mobile* detectasse a região que o dispositivo Beacon está contido, informando uma mensagem desejada e pré-configurada na aplicação *web*.

9 CONCLUSÃO

Devido a grande demanda de aplicações rodando no ambiente operacional da *web,* pode se atender os objetivos apresentados neste projeto de pesquisa que consiste nos estudos dos *frameworks* Hibernate, Spring MVC e AngularJS, aplicando no desenvolvimento uma ferramenta de agendamento para salões de beleza, consistindo em uma aplicação robusta e flexível, e de fácil manutenção devido a utilização das melhores técnicas que estes *frameworks* dispõem.

Com o desenvolvimento do trabalho foi introduzido diversos conceitos importantes e obteve uma facilidade de implementação devido aos recursos que o Hibernate dispõe para que não haja a necessidade de reescrever códigos SQLs, assim como a facilidade de utilizar a estrutura do Spring MVC baseado na injeção de dependências e a navegação realizada através de anotações.

O usuário final obtém um elevado ganho em relação a performance da camada de apresentação que utilizou se o *framework* AngularJS, devido a utilização do conceito de *Single-Page-Application*, fazendo com que o usuário obtenha uma experiência agradável na utilização do sistema, além deste *framework* utilizar os melhores conceitos para otimizar o desempenho.

Na integração dos dados da aplicação *web* com a aplicação *mobile*, pode se aplicar os estudos realizados com a tecnologia beacon, trazendo maior comodidade aos usuários do aplicativo que podem receber informações cadastradas na aplicação *web* do estabelecimento do salão de beleza.

Sugere se que seja abordado como futuros trabalhos, a partir deste projeto de pesquisa os seguintes tópicos:

a) utilização de ibeacon para descrição de produto como um manual;
b) desenvolvimento do aplicativo em multiplataforma com gerenciamento de fluxo de agendamento;
c) desenvolvimento de um módulo de agendamento automático de compromissos, com integração de agendas, alteração em cascata.

REFERÊNCIAS

ABLESON, W. Frank; SEN, Robi; KING, Chris. **Android in Action**. 2.ed. [S. l.]: Manning Publications, 2011.

BATISTA, Diogo T. C. **O impacto do HTML5 no desenvolvimento para a internet**. 2009. Disponível em: <www.diogocezar.com/files/html5/artigo_html5.pdf>. Acesso em: 15 set. 2014.

BAUER, Christian; KING, Gavin. **Hibernate in Action**. Greenwich: Manning, 2005. 431 p.

BURNETTE, Ed. Hello, **Android:** Introducing Google's Mobile Development Platform. 2nd edition [S. l.]: Pragmatic Bookshelf, 2009.

CROCKFORD, Douglas. **JavaScript:** The Good Parts. 1. ed. Estados Unidos: Yahoo! Inc., 2008.

DEVELOPER ANDROID. API Guide. Disponível em: <http://developer.android.com/guide/components/index.html>. Acesso em: 12 nov 2014.

CZERVENY, Arno. **Web Services:** Onde estamos e para onde vamos. Revista Mundo Java, Ano I nº 02, Mundo OO, Rio de Janeiro, RJ. 2004.

DEVMEDIA. Projetando e criando Aplicativos para Dispositivos Móveis. Disponível em: <http://www.devmedia.com.br/projetando-e-criando-aplicativos-para-dispositivos-moveis/30671> Acesso em: 10 de marc. 2015.

DIETZ, Frederik. **Recipes with Angular.js.** Disponível em: <https://leanpub.com/recipes-with-angular-js/read>. Acesso em: 22 de set. 2014.

DOCS ANGULAR. API Guide. Disponivel em:<https://docs.angularjs.org/guide/>. Acesso em: 22 de set. 2015.

FISHER, Paul Tepper; MURPHY, Brian D. **Spring Persistence with Hibernate.** Estados Unidas: Appress, 2010.

FOGEL, Rafael. **Sistema de concessão de financiamento para a apresentação de artigos acadêmicos em Spring MVC.** 2011. Trabalho de Conclusão de Curso (Graduação) – Centro de Ciências Exatas e Tecnologia, Universidade Federal do Estado do Rio de Jnaeiro, 2011.

GARGENTA, Marko. **Learning Android**. Sebastopol: O'reilly, 2011. 266 p.

GREEN, Brad; SESHADRI, Shyam. **AngularJS.** O’Reilly Media,2013.

HARTE, Lawrence. **Introduction to Bluetooth:** Technology, Market, Operation, Profiles, and Services. Althos, 2004.

HIBERNATE REFERENCE. Disponível em: <https://docs.jboss.org/hibernate/orm/3.5/reference/pt-BR/pdf/hibernate_reference.pdf > . Acesso em: 29 de set. 2014.

JOHNSON, Rod et al. **Reference Documentation – Spring Framework 3.0**. Disponível em: <http://docs.spring.io/spring/docs/3.1.0.M1/spring-framework-reference/pdf/spring-framework-reference.pdf>. Acesso em: 29 de set. 2014.

KONDA, Madbusudban. **Just Spring Integration.** Estados Unidos: O'Reilly Media, 2013.

KONDA, Madbusudban. **Just Spring.** Estados Unidos: O'Reilly Media, 2011.

KOZLOWSKI, Pawel; DARWIN B. Peter. **Mastering Web Application Development with AngularJS**. Birmingham: Packt Publishing, 2013.

LINWOOD, Jeff; MINTER, Dave. **Beginning Hibernate.** 2. Ed. Estados Unidos: Apress, 2010.

MARQUES, Juliano. **Comparação de Desempenho e Consumo de Memória entre Frameworks objto-relacional Java hibertnate e eclipselink.** 2012. Trabalho de Conclusão de Curso (Graduação) – Curso de Ciência de computação, Universidade do Extremo Sul Catarinense, 2012.

MEIER, Reto. **Professional Android™ 2 Application Development**. New York: Wiley Publishing, 2010.

MORIMOTO, Carlos E. **Redes:** Guia Prático. 2.Ed. GDH Press e Sul Editores, 2011.

MURPHY, Mark. **Beginning Android 3.** United States: Apress, 2013.

OLSSON, Tommy; O'BRIEN, Paul. **The ultimate CSS reference**. 1. ed. Estados Unidos: SitePoint Pty Ltd, 2008.

PEDRALHO, A. **Bluetooth**: Da teoria à prática. O mundo sem cabos – Parte I. Número 3, Pág. 16-20, 2005.

PEREIRA, Lúcio Camilo Oliva; SILVA, Michel Lourenço da. **Android para desenvolvedores**. Rio de Janeiro: Brasport, 2009.

PERKINS, Steve. **Hibernate Search by Example.** Ingraterra: Packt Publishing Ltd., 2013.

POTTS,Stephen; KOPACK,Mike. **Aprenda Web Services em 24 horas**.Campus, Rio de Janeiro, RJ. 2003.

POUNCEY, Ian; YORK, Richard. **Beginning CSS:** Cascading Style Sheets for Web Design. 3. ed. Estados Unidos: Wiley Publishing, 2011.

POWEL, Thomas.**HTML & CSS:** The Complete Reference, Fifth Edition. 5. ed.Estados Unidos: Mc Graw Hill, 2010.

ROGERS, Rick; LOMBARDO, Jhon; MEDNIEKS, Zigurd; MEIKE, Blake. **Desenvolvimento de Aplicações Android.** São Paulo: Novatec, 2009.

SANTOS, Vinicius José Silva dos. Desenvolvimento de aplicações Google android e integração com web service, 2012. Trabalho de Conclusão de Curso (Graduação). Fundação Educacional do Muncipio de Assis, 2012.

SCHMITZ, Daniel; LIRA, Douglas. Disponível em: <https://leanpub.com/livro-angularJS/read#leanpub-auto-o-que--angularjs>. Acesso em: 10 de abril de 2014.

STABEL, Gabriel Casagrande. **Estudo de Tecnologias para o Desenvolvimento de Aplicação Web para Gerenciamento de Corretora de seguros.** 2014. Trabalho de Conclusão de Curso (Graduação) – Curso de Engenharia de computação, Universidade Federal do Rio Grande do Sul, 2014.

SUPPORT APPLE. Disponível em: <https://support.apple.com/en-us/HT202880>. Acesso em: 20 de marc. 2015.

SVERZUT, José Umberto. **Redes GSM, GPRS, EDGE e UMTS:** evolução a caminho da quarta geração (4G). São Paulo: Érica, 2008.

TECHMUNDO. iOS, Android e Windows Phone: números dos gigantes comparados [infográfico]. Disponível em: <http://www.tecmundo.com.br/sistema-operacional/60596-ios-android-windows-phone-numeros-gigantes-comparados-infografico.htm >. Acesso em: 20 de marc. 2015.

WALLS, Craig. Spring in Action. Estados Unidos: Manning Publications, 2011.

WILLIAM, David. **A história do HTML**. Disponível em: <http://www.frontendbrasil.com.br/artigos/a-historia-do-html/>. Acesso em: 15 set.2014.

WRIGHT, Tim. **Learning JavaScript:** a hands-on guide to the fundamentals of modern JavaScript.Estados Unidos: Addison-Wesley Professional, 2012.

ZAKAS, Nicholas C. **Professional JavaScript for Web Developers**. 3. ed. Estados Unidos: Wrox, 2012.

UTILIZAÇÃO DO FRAMEWORK ANGULARJS ASSOCIANDO INTEGRAÇÕES COM SPRING MVC, HIBERNATE E BLUETOOTH 4.0 PARA A CONSTRUÇÃO DE UMA FERRAMENTA DE AGENDAMENTO E ENVIO DE MENSAGENS

Daiana Cambruzzi Ávila[1], Gustavo Bisognin[2]

[1]Universidade do Extremo Sul Catarinense (UNESC) Caixa Postal 3167 – 88806-00-Criciúma – SC - Brasil

[2]MSc. Professor do Curso de Ciência da Computação – Universidade do Extemo Sul Catarinense (UNESC) Caixa Postal 3167 – 88806-00-Criciúma – SC - Brasil

daiana.avila@gmail.com, gbisog@gmail.com

Abstract. *With the steady growth of the Internet and mobile application development has become indispensable to the use of fast and instant services, requiring the use of the best tools for building a web application. Because of this growth, he noted the need for the study of frameworks, using the best development practices. With the use of the web application synchronization with the mobile application, it is possible to provide mobility in the Bluetooth iBeacon is possible to detect the device, which is a standardized protocol by Apple, and in accordance with the receipt of these signals, you are capable of activating behavior in the mobile application.*

Resumo. *Com o constante crescimento da internet e de desenvolvimento de aplicativos móveis, tornou se indispensável à utilização de serviços rápidos e instantâneos, necessitando da utilização das melhores ferramentas para a construção de uma aplicação web. Devido a esse crescimento, notou se a necessidade de realizar o estudo dos frameworks, utilizando as melhores práticas de desenvolvimento. Com a utilização da sincronização da aplicação web com o aplicativo móvel, é possível oferecer mobilidade para que através do Bluetooth seja possível detectar o dispositivo iBeacon, que é um protocolo padronizado pela Apple, e de acordo com o recebimento desses sinais, é capaz de ativar comportamentos no aplicativo móvel.*

1. Introdução

Com o crescimento de criações de sites com diversos propósitos, é necessário utilizar tecnologias robustas que forneçam segurança e confiabilidade nas transações realizadas pela aplicação web, sendo assim identificamos os frameworks de desenvolvimento que, em sua essência, contemplam algumas características como rapidez, agilidade e flexibilidade de desenvolvimento.

O framework é um conjunto de classes que colaboram para realizar uma responsabilidade para um domínio de um subsistema da aplicação. Com o advento da geração de conteúdo na web, bem como com a crescente demanda dos sistemas rodando em nuvem, identifica-se uma essencialidade quanto a utilização dos frameworks no desenvolvimento de sites, sistemas e aplicativos nos mais diversos níveis de aplicabilidade, pois abstrai os códigos redundantes e provê funcionalidades genéricas que podem ser evoluídas no decorrer do desenvolvimento.

Um dos objetos de estudo deste trabalho, aborda o AngularJS, por ser um framework desenvolvido em JavaScript, que foi criado e mantido pela Google desde 2010, sendo que

uma das suas principais características é a implementação baseada em diretivas nas tags agregando potencial no HyperText Markup Language (HTML). Outra constatação importante, é a associação do framework Spring MVC com o framework AngularJS, pois entendemos, que esta, é essencial para construir uma aplicação robusta e flexível, pois o Spring MVC nos fornece o gerenciamento de objetos e transações de uma forma simplificada, para que no desenvolvimento não exista a necessidade de tanta preocupação com a infraestrutura, principalmente no que refere-se a questões relacionadas ao sincronismo de transações.

A tecnologia mobile proporciona aos seus usuários comodidade, pois é capaz de atingir o cotidiano e fazer parte da vida das pessoas, devido a mudança de rotinas e hábitos, além de formas de tomar decisões, tornando se uma facilidade ao alcance rápido e instantâneo, pois fornece acesso aos dados e informações em qualquer lugar.

No contexto atual, identificamos um problema enfrentado por diversas pessoas que utilizam ferramentas de agendamento, as quais trabalham de forma síncrona e geralmente conectadas a internet. Diversos estabelecimentos comerciais trabalham com a política de agendamento. Dentre eles, podemos destacar consultórios médicos, odontológicos, clínicas, empresas de manutenção e montagem, salões de beleza e entre outros.

O presente trabalho pretende abordar o estudo da integração de tecnologias de frameworks web para a implementação de um protótipo que visa à resolução do problema de agendamento, e visa sua aplicação em salões de beleza, além do estudo da tecnologia iBeacon utilizando um aplicativo móvel que detecta a região que o usuário está contido e a localização do Beacon, e fornece informações ao usuário através do web service, sendo que também há a possibilidade de realizar a sincronização dos agendamentos para que o usuário possa verificar os horários vagos e realizar o agendamento com o salão de beleza.

2. Hibernate

Com as modernas linguagens de programação, assim como o Java, o conceito de orientação a objeto está cada vez mais difundido, os dados são manipulados no formato de objetos, porém na maioria das vezes são persistidos em bancos de dados relacionais, e para tratar este cenário, existem as ferramentas de Mapeamento Objeto-Relacional (MOR), como o framework Hibernate.

O Hibernate é um framework que atua como uma ferramenta de persistência de dados, que tem como intuito de abstrair os códigos Structured Query Language (SQL) para mapeamento objeto/relacional em Java, funcionando como intermediário entre a aplicação e o banco de dados.

Com a utilização da tecnologia Hibernate desenvolvida para o uso na linguagem Java, pretende-se agregar um elevado desempenho na persistência dos dados na base e serviços de consulta, devido ao melhor aproveitamento da transformação entre objetos e linhas de tabelas, de acordo com a conversão destes dois pontos para que exista uma comunicação viável.

O framework pode ser utilizado com ou sem o Java Persistence API (JPA), devido à implementação das interfaces de programação e regras de ciclo de vida definidas na especificação 2.0 da JPA. Diversas características que estão contidas na JPA foram movidas pela inspiração no Hibernate, como por exemplo, o modelo de objetos, gerenciamento de entidades, linguagens de busca e entre outros (BAUER; KING, 2007, tradução nossa).

Uma das vantagens de utilizar o Hibernate é devido abstração do código SQL, uma forma da aplicação ficar independente das particularidades do banco de dados, ou seja, independência entre o modelo de objetos e o banco de dados, sendo este um recurso da camada persistência, fazendo parte da arquitetura do Hibernate.

3. Spring MVC

O Spring MVC faz parte do conjunto do Spring Framework que consiste em um container leve que fornece serviços para a aplicação, como no gerenciamento de objetos ou transação. O Spring MVC é um framework projetado para as construções de aplicações web flexíveis e de fraco acomplamento, baseado no padrão MVC fazendo com que não exista a necessidade de se preocupar com o protocolo HTTP (WALLS, 2011, tradução nossa).

Devido a dificuldade de implementação de aplicações utilizando os Servlets e JSPs puro, a própria Sun decidiu investir mais na utilização do padrão MVC e de patterns como o FrontController, sendo que era comum as empresas implementar estes padrões criando então mini-frameworks, então foi neste momento que houve a necessidade de implementar o framework MVC.

Este framework trabalha com pedidos entre um DispatcherServlet, Handler Mapping, Controllers e View Resolvers, sendo que cada componente em Spring MVC possui uma finalidade especifica, utilizado através dos containers e a API de Servlets para encapsular o protocolo HyperText Transfer Protocol (HTTP). O manipulador padrão é o @Controller e @RequestMapping, que fornece diversos métodos de manipulações flexíveis.

3.1 Controller

Os controllers proporcionam o acesso ao comportamento da aplicação que é definido através de uma interface de serviço, sendo que possuem a capacidade de interpretar e transformar em modelo de entrada do usuário para posteriormente exibir ao usuário, sendo que um controller é implementado de uma forma abstrata, para possibilitar a criação de diversas variedades de controladores (JOHNSON et al, 2011, tradução nossa).

Para definir um controller no Spring MVC é necessário utilizar a anotação @Controller. A anotação para definir um controller é utilizada, pois o Spring introduziu um modelo de programação baseada em anotações, assim como o RequestMapping, RequestParam, ModelAttribute e etc, dessa forma não existe a necessidade de estender de classes base especificas ou então implementar interfaces (JOHNSON et al, 2011, tradução nossa).

O Spring MVC possui o recurso de criar controllers com suporte ao REST, que é um estilo de arquitetura projetado para sistemas distribuídos, realizando uma comunicação entre o cliente e o servidor, sendo comumente associado ao protocolo HTTP (JOHNSON et al, 2011, tradução nossa).

Segundo Fisher e Murphy (2010, tradução nossa) o REST faz o aproveitamento do HTTP, devido ao fornecimento de uma abordagem coerente, orientada para o que o recurso represente os dados, que ao contrário de uma chamada de procedimento remoto, o REST centraliza a representação de um determinado recurso que é concretizado como uma URL.

4. Tecnologias Web

A demanda de criações de sites com o propósito institucional, informativos, e-Commerce, armazenamento de informações, portais, mídias sociais ou então instrumentos de publicidades, vem crescendo de forma exponencial no comércio mundial. Desta forma, é possível identificar a necessidade de utilizar tecnologias robustas que permitam maior segurança e confiabilidade nas transações comerciais ou serviços utilizados na web em nosso dia a dia.

A internet vem se destacando como um meio de comunicação que possui um alto nível de eficácia, devido à fácil interação e agilidade para realizar o tráfego e trocas de informações.

Para a criação destas páginas que estão disponíveis na internet, são utilizadas as tecnologias web, que são definidas pelo código e recursos que fornecem para um servidor web, e que tem por finalidade a execução do conteúdo num navegador web.

Após o surgimento de diversos tipos de navegadores para exibir as páginas, foi indispensável à padronização das tecnologias que os navegadores interpretam, sendo assim, ficou definido as três linguagens padrões: o HyperText Markup Language (HTML), Cascading Style Sheets (CSS) e JavaScript.

Através da linguagem de marcação HTML é que é possível desenvolver páginas da web, pois é utilizada para realizar a formatação do conteúdo que deverá ser exposto, sendo que o CSS codifica o estilo da página e o JavaScript é utilizado para realizar as interações. Conforme Wright (2012, tradução nossa), para a construção de uma página web é necessário manter o HTML, CSS e JavaScript separados, e referir como três camadas, sendo elas de estrutura que se refere ao HTML, a apresentação que é a utilização do CSS e o comportamento que é a utilização do JavaScript para lidar com as interações da página.

5. Framework AngularJS

O AngularJS é um framework open-source desenvolvido pela Google, que fornece aos desenvolvedores JavaScript uma estrutura robusta para elevar a produtividade e desenvolvimento de aplicações ricas e com potencial (DIETZ, 2013, tradução nossa).

A construção do framework iniciou em 2009 como um projeto pessoal de Misko Hevery, funcionário da Google, e posteriormente a Google Inc. oficialmente começou a apoiar este projeto, sendo que a primeira versão foi lançada em junho de 2012 (KOZLOWSKI; DARWIN, 2013, tradução nossa).

Uma das principais particularidades que torna a utilização deste framework tão assíduo no desenvolvimento de páginas web é devido à funcionalidade de trabalhar como uma extensão do documento HTML, com o DataBinding, templates e a fácil utilização do Ajax.

Segundo Schmitz e Lira (2012), este framework é praticamente uma linguagem declarativa, pois utiliza a adição de propriedades para modificar o comportamento da linguagem e interage de forma dinâmica com os elementos HTML, sendo que estes parâmetros são denominados de diretivas.

Adicionado recentemente à lista de frameworks que utiliza o padrão Model-View-Controller (MVC), acabou atraindo muita atenção devido ao sistema de modelos inovadores, facilidade de desenvolvimento e práticas de engenharia sólidas, sendo que a modelagem de sistema se torna diferencial devido à utilização do HTML como linguagem de templates, abstração da manipulação do Document Object Model (DOM) explicitamente e possui componentes extensíveis. (KOZLOWSKI; DARWIN, 2013, tradução nossa).

Inicialmente a aplicação do AngularJS utilizava o modelo arquitetural de Model-View-Controller (MVC) no desenvolvimento de software, e posteriormente os próprios autores informaram a adesão do modelo Model-View-Whatever (MVW). Independente do padrão de projeto de software que será utilizado para o desenvolvimento da página web, é necessário sempre seguir os conceitos de model, view e controller, para separar corretamente as funcionalidades para cada arquivo, dos recursos mais importantes. O que difere o framework AngularJS com os demais frameworks JavaScript, é a utilização de diretivas, que agregam um valor elevado ao HTML pois lida diretamente com o DOM.

6. Dispositivos Móveis

Com o elevado número de compras de dispositivos móveis e que crescem a cada dia, esses dispositivos já representam a maioria dos sistemas computadorizados atualmente,

causando então, uma evolução no desenvolvimento de aplicativos para esses aparelhos (DEVMEDIA, 2015).

Entre os sistemas operacionais existentes para os dispositivos móveis, o sistema operacional mais utilizado em portáteis do mercado é o Android, que existem cerca de mais de um bilhão de aparelhos ativos (TECHMUNDO, 2014).

Entretanto com a utilização dos dispositivos móveis com o sistema operacional Android, é possível utilizar recursos importantes para a comunicação com outros dispositivos, como o Bluetooth do aparelho, que permite realizar transferências de dados e outras funcionalidades que podem ser consumidas pelos aplicativos desenvolvidos.

6.1 Plataforma Android

O Android consiste em uma plataforma open source, projetada para dispositivos móveis e defendida pelo Google e Open Handset Alliance, sendo que esta aliança visava acelerar as inovações móveis e disponibilizar uma experiência móvel mais rica, já que a parceria consistia entre duas empresas que são envolvidas em tecnologia de telefonia móvel (GARGENTA, 2011, tradução nossa).

Com o surgimento do Sistema Operacional Android, os desenvolvedores tiveram uma maior liberdade para implementar os aplicativos, pois os outros sistemas operacionais eram proprietários e não permitiam aplicativos de terceiros. Com o Android, o desenvolvedor tem a possibilidade de desenvolver os seus aplicativos móveis utilizando os recursos do hardware (MEIER, 2010, tradução nossa).

De acordo com Rogers et al (2009), com o surgimento do Android houve um crescimento no ambiente de desenvolvimento de aplicativos móveis, pois anteriormente sofria com algumas barreiras, como por exemplo, um ambiente de aplicação de acordo com cada marca, softwares proprietários e o bloqueio aplicativos de terceiros.

O Android foi projetado com base na versão 2.6 do Kernel do Linux, possuindo então uma arquitetura bem estruturada e baseada em camadas, com a finaliade de tratar isoladamente as funções de aplicações, execuções paralelas, serviços centrais e entre outros (PEREIRA; SILVA, 2009).

6.2 Bluetooth

O Bluetooth é um padrão de tecnologia para áreas de redes pessoais sem fio, ou seja, uma rede Personal Area Network (PAN), que serve para interligar celulares, palmtops e outros dispositivos de uso pessoal em curtas distâncias (MORIMOTO, 2011). Conforme Harte (2004, tradução nossa), a tecnologia Bluetooth foi desenvolvida a fim de proporcionar comunicação sem fio entre dois ou mais dispositivos capazes de enviar e receber ondas de rádio de curto alcance.

A tecnologia Bluetooth opera sobre uma banda de radiofrequência denominada Industrial, Scientific and Medicine (ISM) utilizando a técnica de transmissão Frequency Hopping Spread Spectrum (FHSS) (PEDRALHO, 2005). Esta tecnologia é um padrão de comunicação com baixo consumo de energia e possui um raio de alcance de acordo com as características de cada aparelho, dividindo em classes de alcances como:

a) classe 1: alcance aproximadamente de 100 a 300 metros, com potência máxima de 100 mW;
b) classe 2: alcance aproximadamente de 10 a 33 metros, com potência máxima de 2.5 mW;
c) classe 3: alcance aproximadamente de 1 a 3 metros, com potência máxima de 1.0 mW.

O Bluetooth opera na faixa de frequência de 2,4 GHz, que é reservada pela Agência Nacional de Telecomunicações (ANATEL) para equipamentos de radiação restrita, que não precisam de licença (SVERZUT, 2008). Essa frequência é chamada de ISM, determinada de acordo com um acordo internacional para a utilização de dispositivos científicos, médicos e industriais.

6.3 iBeacon

O iBeacon é um protocolo padronizado pela Apple, que utiliza os serviços de localização, fazendo com que emita um alerta com a localização de um dispositivo iBeacon (APPLE,2015, tradução nossa).

Vários fornecedores utilizam essa tecnologia e expandem a tecnologia destes transmissores. Essa tecnologia utiliza o Bluetooth Low Energy, que é uma tecnologia de rede de área pessoal utilizada para realizar transmissões de dados em pequenas distancias projetado para ter um baixo consumo de energia e de custos, além de manter um alcance de comunicação semelhante ao Bluetooth Classic, tendo como alcance 100 metros para realizar a descoberta do dispositivo iBeacon (IBEACON, 2015, tradução nossa).

A Apple realizou a padronização do formato Bluetooth Low Energy, sendo assim um pacote de dados transmitido possui quatro principais informações:

a) UUID: cadeia de 16 bytes utilizados para diferenciar os iBeacons;
b) maior: cadeia de 2 bytes utilizados para distinguir um subconjunto de iBeacons;
c) menor: sequência de 2 bytes, destinados a identificar os iBeacons individuais;
d) taxa Potência: utilizado para determinar a proximidade a partir do iBeacon.

Para que os smartphones que utilizam o sistema operacional Android possam ler os sinais emitidos dos iBeacons, devem possuir uma estrutura mínima de utilização da versão do Android 4.3, como por exemplo os dispositivos: Samsung Galaxy S3 / S4 / S4 Mini, Samsung Galaxy Note 2/3, HTC One, Google / LG Nexus 7 2013 versão / Nexus 4/5 Nexus, HTC Borboleta, OnePlus One.

7. Desenvolvimento do Sistema

O processo metodológico da pesquisa desenvolvida é composta pelo levantamento de requisitos, modelagem de dados, seleção das ferramentas de desenvolvimento, implementação da aplicação *web* e aplicativo *mobile*, e por fim, a análise e escrita dos resultados obtidos através da construção da aplicação *web*.

Com o início do desenvolvimento *web*, utilizou se os *frameworks* Hibernate, Spring MVC e AngularJS, aplicando o baseamento teórico estudado para a construção da aplicação *web* para um agendamento de salão beleza, com o principal propósito de utilizar as novas tecnologias para a construção de um sistema web flexível, robusto, confiável e de fácil manutenção, assuntos estes abordados e estudados no levantamento bibliográfico.

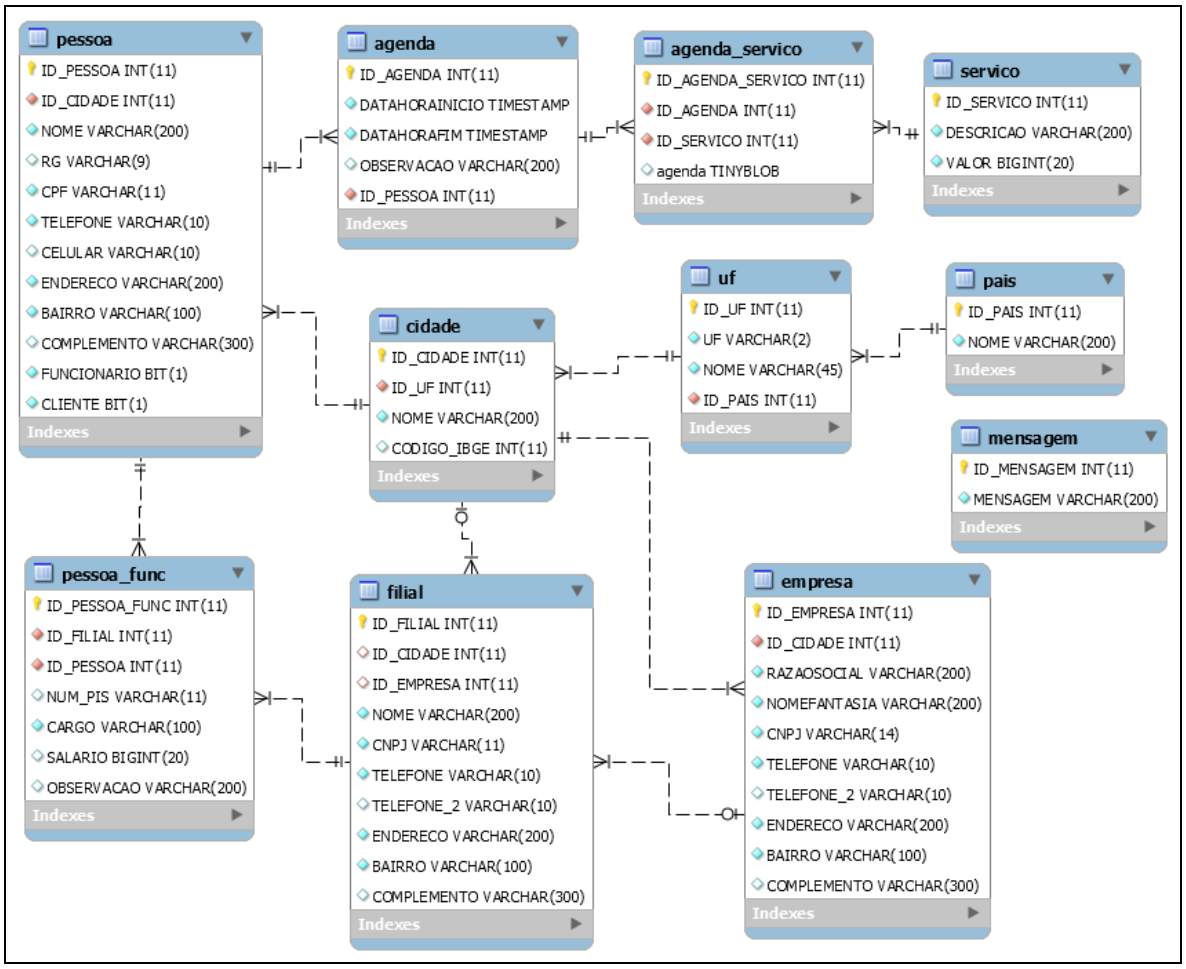
7.1 Modelagem de Dados

Realizou se um estudo sobre agendamentos de salões de beleza, para analisar quais são os requisitos necessários, para o desenvolvimento de uma aplicação web, que posa suprir todas as necessidades de um agendamento para um salão de beleza.

A modelagem de dados se torna um requisito importante para a construção de uma ferramenta, devido à demonstração de características de funcionamento e comportamento do

software, facilitando a compreensão do projeto e diminuindo os erros de programação, projeto e funcionamento, sendo assim, conforme a figura 1 exibe o modelo lógico da aplicação *web*:

Figura 1 - Modelo lógico da aplicação web



Fonte: Do autor (2015).

O modelo lógico serve para mostrar as ligações entre as tabelas de banco de dados, as chaves primárias e demais componentes.

7.2 Construção da Ferramenta Web

A criação do projeto deve ser realizada com a utilização do Eclipse, utilizando a opção de criar um projeto Maven, que simplifica o gerenciamento de dependências, e a criação e distribuição dos projetos. Para a utilização do Maven, deve ser realizado o download no site oficial e posteriormente descompactar na pasta no diretório que deverá ser instalado, e logo após o caminho da pasta deverá ser adicionado nas variáveis de ambiente.

Com a criação do projeto, irá ser criado também o arquivo "*pom.xml*", que é considerado o coração do projeto, pois neste arquivo será adicionado as dependências do projeto, como por exemplo, as dependências do Spring MVC e do Hibernate, e que também contém a centralização da documentação e principalmente o modo de compilação e distribuição da aplicação.

As *packages* contém as classes Javas, e são divididas com os controladores na *package aplicacao.agendamento.controller*, as classes de modelos devem estar contidas na *package aplicacao.agendamento.domain* que deve possuir os arquivos POJOs que são responsáveis pelos mapeamentos dos objetos com as tabelas e os campos do banco de dados,

além de possuir as associações e subclasses persistentes. As classes que realizam o acesso aos serviços e os arquivos para acessar o banco de dados, devem estar nas *packages aplicação.agendamento.service* e *aplicação.agendamento.dao*.

Os arquivos *web* por padrão devem estar contidos na pasta *webapp*, contendo outras pastas como configurações do Spring MVC, as propriedades do JDBC, os arquivos estáticos, além de conter também os arquivos JavaScript utilizados com o framework *AngularJS*. Posteriormente com a estrutura do projeto criada e devidamente distribuída, deverá ser iniciado o processo de configuração do Spring MVC na aplicação.

Para realizar a configuração do Spring MVC na aplicação, é necessário criar o arquivo web.xml, que contém a configuração normal de Servlet. A configuração do framework Spring é feito através do XML spring-servlet, que contém diversas informações, e inicialmente contém a configuração para habilitar o uso de anotações do Spring MVC, além de configurar o pacote base da aplicação.

É necessário configurar o pacote para que o Spring detecte aonde se encontra as classes, neste arquivo também é possível realizar a configuração com o banco de dados, o bean "sessionFactory" e o bean "transactionManager" para realizar a configuração do Spring MVC com o Hibernate para ter acesso ao banco de dados.

No arquivo de configuração do Spring MVC, também contém a configuração do Hibernate com o banco de dados. O arquivo contém diversos mapeamentos do Spring, como o *Bean PropertyConfigurer*, que é utilizado para carregar o arquivo *jdbc.properties*, que contém as propriedades e detalhes de conexão do banco de dados que são utilizados para realizar a conexão com o Hibernate.

O *Bean DataSource* é utilizado como a fonte de dados Java, para se conectar ao banco de dados, através da classe do driver JDBC, URL de conexão, dialeto, nome de usuário e senha. O *Bean SessionFactory* é usado para definir as diversas configurações do Hibernate, que estão no arquivo hibernate.cfg.xml.

Neste arquivo hibernate.cfg.xml, contém quais são as classes que deverão ser visualizadas com as anotações de entidades. Por fim, o *Bean TransactionManager*, é o gerenciador de transações do Hibernate, que é ligado através da propriedade *sessionFactory*, com o *Bean SessionFactory.*

O Hibernate oferece duas técnicas para criar os mapeamentos dos objetos com os modelos da base de dados, podendo ser criados arquivos XML, ou através de anotações nos objetos POJO.

Na construção da ferramenta de agendamento, utilizou se o mapeamento através das anotações nas classes POJO, devido à facilidade de compreensão do código fonte em comparação com o mapeamento realizado com arquivos XML. As classes POJO foram criadas utilizando as anotações JPA, que estão contidas no pacote *javax.persistence.*

As classes DAO, possuem a anotação do Spring *@Repository*, que serve para indicar ao *framework* que são classes da camada de persistência e que funcionará como repositório de dados. Com a criação das classes de acesso ao banco de dados, é necessário criar as classes que irão possuir os serviços que irão realizar as chamadas das funções que contém nas classes DAO para trazer as informações do banco de dados ou realizar alguma inserção, alteração ou remoção de dados.

Para criar uma classe de serviço é indispensável a utilização da anotação do *@Service* do Spring MVC, para que a classe seja detectada como classes da camada de serviço. Na implementação da classe de serviço, são criadas as funções que fazem a chamada das funções das classes DAO, possuindo a anotação *@Transactional*, que determina que essa função seja uma transação.

Com as classes de acesso ao banco de dados e o serviços que fazem essas chamadas, é necessário criar os controladores que são responsáveis pelo envio e respostas aos usuários.

Assim como as demais classes, o *controller* deve conter a anotação *@Controller*, para que o Spring MVC detecte que são classes em nível de camada de apresentação.

Os *controllers* irão conter as requisições que são anotadas por *@RequestMapping*, contendo as propriedades *value* que irá definir a requisição, a propriedade *method* indica qual o tipo de pedido HTTP que está sendo realizado, pois poderá ser uma requisição com o método*: GET, POST, PUT ou DELETE*, além de conter outras propriedades que podem ser definidas no mapeamento da requisição

Posteriormente com a criação dos controllers que possuem todas as requisições mapeadas, já é possível iniciar o desenvolvimento das telas que irão realizar as chamadas desses controllers, através dos controllers do framework AngularJS.

Com as requisições implementadas é necessário iniciar o desenvolvimento das telas utilizando o *framework* AngularJS, que irá realizar as chamadas dessas requisições de acordo com cada situação. Os arquivos HTML foram divididos em pastas de acordo com cada função, sendo que os cadastros possuem arquivos de lista, cadastro e o arquivo JavaScript que contem os *controllers* e configuração de rotas.

Para o desenvolvimento *front-end*, a página inicial possui a diretiva *ng-app*, que é utilizada para definir o nome do módulo da aplicação, e a diretiva *ng-view* que será utilizada pelas rotas do AngularJS. A página possui um menu lateral que possui ligação com os cadastros da ferramenta de agendamento, e utilizou se o conceito de rotas do AngularJS.

Utilizou se o conceito de rotas para acessar as funções da aplicação, devido à facilidade de manipulação de partes do código HTML, de forma independente, além de separar as camadas da aplicação, tornando a mais gerenciável.

Para realizar a comunicação do cliente com o servidor, é utilizado o conceito RESTful. O RESTful é uma comunicação HTTP, que possui um padrão simples, que utiliza os cabeçalhos como POST, GET, PUT e DELETE (Documentação AngularJS).

7.3 Desenvolvimento Mobile

O desenvolvimento do aplicativo *mobile* se realizou utilizando a ferramenta Android Studio, compilando o projeto da biblioteca *Android Week View* para fazer a utilização do calendário com os eventos marcados e a utilização do SDK do *Estimote Beacon* para realizar a conexão com o dispositivo iBeacon.

Na tela inicial da aplicação, possui a opção de "Sincronizar" que realiza a chamada da requisição do *RestController* da Agenda, criado com o *framework* Spring MVC, retornando o arquivo JSON, para ser lido através do método "lerJsonAgenda", que faz a leitura dos dados e retorna para a lista de eventos sincronizados com o calendário.

Para realizar a sincronização, foi criada função "sincronizarAgenda" que executa a classe "lerJsonAgenda", que é uma classe AsyncTask, por ser uma tarefa assíncrona, passando como parâmetro a URL do *web service* que irá retornar o arquivo JSON com os dados. Após ter feito a conexão com a URL e o download do arquivo JSON, é executada a função "onPostExecute" para ler os dados e adicionar nos eventos do calendário.

Com a aplicação rodando é possível realizar a localização do dispositivo Beacon, através da implementação utilizando o SDK do Estimote, configurando o UUID que deverá ser localizado na determinada região. Com a implementação do método de conexão "beaconStart", é possível informar ao usuário que está dentro da região do Beacon, informando uma mensagem configurada na aplicação *web* e recebida através do *web service* criado pelo Spring MVC.

A classe "BeaconsMonitoringService" é responsável por executar as funções que localizam o iBeacon. Inicialmente é definido a região do iBeacon, configurando pelo UUID do dispositivo. Na função "onStartCommand", ao definir a função que irá monitorar o

dispositivo, é verificado se há conexão do aparelho móvel com a *internet*, caso haja conexão é realizado então o procedimento de realizar a comunicação com o *web service* responsável por retornar a mensagem pré-configurada pela aplicação *web,* utilizando um método semelhante a sincronização da agenda, caso não exista conexão com a *internet,* o aplicativo emite uma mensagem padrão.

A mensagem de notificação é inserida de acordo com o formulário na aplicação *web*.

Realizando a configuração da mensagem na aplicação *web*, faz com que haja uma interação entre o aplicativo *mobile* e a aplicação *web*, e que isso retorne ao usuário devido à aproximação do local o qual o Beacon está distribuindo os sinais.

7.4 Resultados Obtidos

Com o desenvolvimento do projeto de pesquisa deste trabalho, pôde se obter a aplicação do estudo para realizar a integração dos *frameworks* Hibernate e Spring MVC com AngularJS na construção de uma ferramenta de agendamento, juntamente com a aplicação dos estudos da plataforma Android e integração com o dispositivo Beacon.

Na integração dos *frameworks* para aplicar os estudos realizados, pode se obter uma aplicação com diversas vantagens de criação, como com a utilização do Hibernate a facilidade de desenvolvimento e a fácil manutenção do software devido à utilização das especificações JPA, tornando mais dinâmico a tarefa de tratar os acessos aos dados, além de abstrair o banco de dados para que não haja a preocupação na criação e manutenção das tabelas no banco, devido ao mapeamento realizado no Hibernate. Obteve uma velocidade no desenvolvimento da aplicação, devido a não necessidade de reescrever códigos SQL, pois o Hibernate realiza o encapsulamento de todo o código.

Baseando a aplicação com o *framework* Spring MVC, obteve um aproveitamento das injeções de dependências, devido ao container que se encarrega de instanciar as classes Java e definir suas dependências, possuindo uma arquitetura baseada em POJOs e *interfaces*, além de realizar a navegação através de anotações.

Com a integração do AngularJS para realizar o desenvolvimento *front-end* da aplicação, obteve se a vantagem que a renderização da página da aplicação é realizada diretamente no lado do cliente, diretamente no navegador do usuário, além de aplicar o conceito de *single-page-application* que também diminui o trafego de dados entre o cliente e servidor, aumentando a perfomance e a experiência do usuário na utilização do sistema. Com a utilização do AngularJS, agregou se um alto valor na aplicação, devido a utilização das melhores práticas com o padrão MVC e injeção de dependências.

Devido a utilização do *framework* Spring MVC na aplicação *web,* pode ser utilizar o *web service* criado para realizar a integração entre a aplicação *web* e o aplicativo *mobile*, agregando mobilidade aos usuários do aplicativo, para que possam realizar as consultas dos agendamentos realizados no salão e analisar os horários vagos para realizar o agendamento de acordo com a data desejada. Com o estudo da plataforma Android juntamente com o dispositivo Beacon, pode se aplicar o referencial bibliográfico para que o aplicativo *mobile* detectasse a região que o dispositivo Beacon está contido, informando uma mensagem desejada e pré-configurada na aplicação *web*.

8. Conclusão

Devido a grande demanda de aplicações rodando no ambiente operacional da *web,* pode se atender os objetivos apresentados neste projeto de pesquisa que consiste nos estudos dos *frameworks* Hibernate, Spring MVC e AngularJS, aplicando no desenvolvimento uma ferramenta de agendamento para salões de beleza, consistindo em uma aplicação robusta e

flexível, e de fácil manutenção devido a utilização das melhores técnicas que estes *frameworks* dispõem.

Com o desenvolvimento do trabalho foi introduzido diversos conceitos importantes e obteve uma facilidade de implementação devido aos recursos que o Hibernate dispõe para que não haja a necessidade de reescrever códigos SQLs, assim como a facilidade de utilizar a estrutura do Spring MVC baseado na injeção de dependências e a navegação realizada através de anotações.

O usuário final obtém um elevado ganho em relação a performance da camada de apresentação que utilizou se o *framework* AngularJS, devido a utilização do conceito de *Single-Page-Application*, fazendo com que o usuário obtenha uma experiência agradável na utilização do sistema, além deste *framework* utilizar os melhores conceitos para otimizar o desempenho.

Na integração dos dados da aplicação *web* com a aplicação *mobile*, pode se aplicar os estudos realizados com a tecnologia beacon, trazendo maior comodidade aos usuários do aplicativo que podem receber informações cadastradas na aplicação *web* do estabelecimento do salão de beleza.

References

BAUER, Christian; KING, Gavin. **Hibernate in Action**. Greenwich: Manning, 2005. 431 p.

DEVMEDIA. **Projetando e criando Aplicativos para Dispositivos Móveis.** Disponível em: <http://www.devmedia.com.br/projetando-e-criando-aplicativos-para-dispositivos-moveis/30671> Acesso em: 10 de marc. 2015.

DIETZ, Frederik. **Recipes with Angular.JS**. Disponível em: <https://leanpub.com/recipes-with-angular-js/read>. Acesso em: 22 de set. 2014.

DOCS ANGULAR. API Guide. Disponivel em:<https://docs.angularjs.org/guide/>. Acesso em: 22 de set. 2015.

FISHER, Paul Tepper; MURPHY, Brian D. **Spring Persistence with Hibernate**. Estados Unidas: Appress, 2010.

GARGENTA, Marko. **Learning Android**. Sebastopol: O'reilly, 2011. 266 p.

HARTE, Lawrence. **Introduction to Bluetooth:** Technology, Market, Operation, Profiles, and Services. Althos, 2004.

JOHNSON, Rod et al. Reference Documentation – Spring Framework 3.0. Disponível em: <http://docs.spring.io/spring/docs/3.1.0.M1/spring-framework-reference/pdf/spring-framework-reference.pdf>. Acesso em: 29 de set. 2014.

KOZLOWSKI, Pawel; DARWIN B. Peter. **Mastering Web Application Development with AngularJS**. Birmingham: Packt Publishing, 2013.

MEIER, Reto. **Professional Android™ 2 Application Development**. New York: Wiley Publishing, 2010.

MORIMOTO, Carlos E. **Redes:** Guia Prático. 2.Ed. GDH Press e Sul Editores, 2011.

PEDRALHO, A. **Bluetooth:** Da teoria à prática. O mundo sem cabos – Parte I. Número 3, Pág. 16-20, 2005.

PEREIRA, Lúcio Camilo Oliva; SILVA, Michel Lourenço da. **Android para desenvolvedores**. Rio de Janeiro: Brasport, 2009.

ROGERS, Rick; LOMBARDO, Jhon; MEDNIEKS, Zigurd; MEIKE, Blake. **Desenvolvimento de Aplicações Android**. São Paulo: Novatec, 2009.

SCHMITZ, Daniel; LIRA, Douglas. Disponível em: <https://leanpub.com/livro-angularJS/read#leanpub-auto-o-que--angularjs>. Acesso em: 10 de abril de 2014.

SVERZUT, José Umberto. **Redes GSM, GPRS, EDGE e UMTS:** evolução a caminho da quarta geração (4G). São Paulo: Érica, 2008.

TECHMUNDO. iOS, Android e Windows Phone: números dos gigantes comparados [infográfico]. Disponível em: <http://www.tecmundo.com.br/sistema-operacional/60596-ios-android-windows-phone-numeros-gigantes-comparados-infografico.htm >. Acesso em: 20 de marc. 2015.

WALLS, Craig. **Spring in Action**. Estados Unidos: Manning Publications, 2011.

WRIGHT, Tim. **Learning JavaScript:** a hands-on guide to the fundamentals of modern JavaScript.Estados Unidos: Addison-Wesley Professional, 2012.