UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

DIOGO RIGOLLI DADALT

Uma ferramenta para experimentação em deduplicação de dados pessoais

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Carlos Alberto Heuser

Porto Alegre
2014

RESUMO

Um problema recorrente encontrado por pessoas que trabalham com bases de dados genealógicas são as duplicatas. No contexto de bases de dados genealógicas, estas duplicatas usualmente descrevem o mesmo registro de forma diferente e muitas vezes incorreta. Por isto é importante encontrar e eliminar estes registros. A este processo se dá o nome de deduplicação. Esta é uma tarefa conhecida por ser difícil de automatizar. A ferramenta proposta por este trabalho dá o primeiro passo na direção da automatização. Ela recebe como entrada uma base de dados GEDCOM (formato de bases genealógicas) juntamente com uma lista de evidências, sugerindo ao usuário os registros que se referem a um mesmo indivíduo. Estas sugestões são obtidas com o auxílio de conceitos de aprendizado de máquina providos pela biblioteca WEKA. Além disso, a ferramenta está disponível através da internet, o que permite que ela seja acessada de qualquer lugar do mundo e por qualquer pessoa.

Palavras-chave: Deduplicação. Bases de dados genealógicas. GEDCOM.

A tool for experimenting in deduplication of personal data

ABSTRACT

A recurrent problem found by people that work with genealogy databases are duplicates. In the context of genealogy databases, usually these duplicates describe the same record in a different way and sometimes wrongly. Therefore it is important to find and eliminate these records. To this process is given the name deduplication. This is a task known to be difficult to automate. The tool proposed by this work takes the first step towards the automation of this process. It receives as input a GEDCOM (genealogy database format) along with a list of evidences, providing as output a suggestion of the records that refer to the same individual. These suggestions are obtained with the help of machine learning concepts provided by the WEKA library. Besides that, the tool is available over the internet, thus letting it be accessed by anyone, anywhere.

Keywords: Deduplication. Genealogy databases. GEDCOM.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
ARFF	Attribute-Relation File Format
CSS	Cascading Style Sheets
GEDCOM	GEnealogical Data COMmunication
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
JSON	JavaScript Object Notation
MVC	Model View Controller
REST	Representational State Transfer
UFRGS	Universidade Federal do Rio Grande do Sul
XML	Extensible Markup Language
WEKA	Waikato Environment for Knowledge Analysis

SUMÁRIO

1 INTRODUÇÃO

No contexto de bancos de dados, a redundância de dados pode ser utilizada como uma forma de melhorar a performance de consultas. Isto se tornou comum recentemente com a popularização dos bancos de dados noSQL. Alguns destes bancos de dados incentivam a desnormalização dos dados, sugerindo que se guarde os registros de modo que não seja necessário combinar dados entre tabelas.

Mas quando a redundância não é proposital, tipicamente ela é encarada como um problema. Estes registros duplicados devem então ser encontrados e eliminados. O processo de eliminar dados redundantes é conhecido como deduplicação.

Em bases de dados genealógicas, o cadastro de dados repetidos é um fenômeno muito comum. Estas duplicatas normalmente descrevem de forma diferente o mesmo registro, e muitas vezes fornecendo informações imprecisas. Isto torna difícil encontrar estas duplicatas, tanto de forma manual como de forma automatizada.

Este trabalho visa fornecer uma ferramenta para facilitar o processo de encontrar duplicatas em bases de dados genealógicas através do uso de conceitos de aprendizagem de máquina. Mais especificamente em bases de dados do tipo GEDCOM. A ferramenta referida se utiliza de evidências sobre os registros, que podem ser selecionadas pelo usuário, para comparar registros dois a dois. Estas evidências são comparadas através do cálculo de sua similaridade que é determinado por algoritmos que também podem ser escolhidos pelo usuário. Algoritmos diferentes são disponibilizados para escolha conforme o tipo de dado da evidência a ser comparada. Além disso o usuário também deve selecionar um algoritmo de classificação que terá a função de fazer as previsões sobre os dados.

A intenção da ferramenta proposta é fornecer uma forma de testar todas estas possíveis variações de configurações de uma forma fácil, afim de que se descubra qual a melhor combinação de evidências, algoritmos de cálculo de similaridade e algoritmo de classificação

geram as melhores previsões. É importante frisar também que a ferramenta tem como alvo usuários especialistas, ou seja, que tenham conhecimento técnico de bancos de dados e algoritmos.

O presente documento está organizado conforme o que se segue. O capítulo 2 descreve o formato GEDCOM e em que contexto ele foi utilizado neste trabalho. O capítulo 3 apresenta a plataforma WEKA, a aplicação falando sobre os algoritmos utilizados, sua API e o formato de arquivo ARFF. O capítulo 4 detalha como a aplicação pode ser utilizada do ponto de vista do usuário, mostrando casos de uso e telas. O capítulo 5 enfoca a arquitetura da aplicação, falando sobre as tecnologias envolvidas e detalhes da implementação.

2 O FORMATO GEDCOM

Neste capítulo serão abordados detalhes sobre o formato GEDCOM. Inicialmente será provida uma definição do formato, falando um pouco de sua origem e características. E nas seções seguintes serão abordados exemplos e explicações das partes importantes do formato além de explanar como ele foi utilizado no contexto deste trabalho.

2.1 Definição

GEDCOM é um formato proprietário criado pelo Departamento de História da Família da Igreja de Jesus Cristo dos Santos dos Últimos Dias para armazenar e trocar informações genealógicas promovendo a interoperabilidade entre aplicações. Foi originalmente desenvolvido em 1985 e se tornou um padrão de fato por ser amplamente aceito pelas aplicações relacionadas a genealogia. A versão atual da especificação GEDCOM é a 5.5 que data de 1º de Novembro de 2000.

Arquivos do tipo GEDCOM tem extensão ".ged" e são arquivos de texto que contém informações genealógicas. Eles usam tags para descrever informações relacionadas a família. Alguns exemplos destas tags são: INDI para indivíduo, FAM para família, BIRT para nascimento e DATE para data. Exemplos de arquivos GEDCOM serão mostrados na próxima seção.

Como mencionado anteriormente, os arquivos GEDCOM são utilizados para guardar informações genealógicas, sendo utilizado pela maioria das aplicações relacionadas a genealogia como forma de carregar estas informações, exportá-las, possibilitando que essas informações sejam transportadas de uma aplicação para outra de forma simples. Esta é a grande vantagem de se utilizar este formato.

É importante salientar que algumas aplicações utilizam uma versão modificada do padrão GEDCOM, que muitas vezes não são compatíveis com o padrão original e portanto

não podem ser utilizados por outras aplicações. Isto faz com que se perca os benefícios referidos anteriormente, de se utilizar um mesmo formato.

2.2 Anatomia do formato

Nesta seção é apresentada uma descrição simplificada de um arquivo GEDCOM. Apresentando apenas as características mais importantes e mais relevantes no contexto do presente trabalho.

Um arquivo GEDCOM é um conjunto de registros, que podem ser um indivíduo, uma fonte ou uma família. Ele não pode ter linhas em branco nem indentações. Cada linha tem em seu início um número que designa o nível da informação contida nela em relação ao registro a que ela pertence. E a primeira linha de cada registro contém o número “0”, sinalizando o início de um registro.

Depois do número, cada linha contém também uma tag, que é uma abreviação do tipo de dado contido nesta linha. A maioria das tags podem ser facilmente relacionadas às suas correspondentes palavras em inglês. Alguns exemplos são: HUSB que vem de *husband* e significa marido, PLAC que vem de *place* e significa lugar, MARR que vem de *marriage* que significa casamento, entre outros. Estas tags também podem ser ponteiros para outros registros. As tags são seguidas pela informação em si.

O primeiro registro é o cabeçalho que começa com uma linha com o número 0 e vai até o próximo registro, no caso, a próxima linha com o número 0. O cabeçalho contém informações introdutórias, como o programa origem usado para criar o arquivo GEDCOM e quais versões do programa e do GEDCOM foram utilizadas. Na Figura 2.1 é mostrado um exemplo de como um registro de cabeçalho se parece.

```
0 HEAD
1 SOUR Legacy
2 VERS 7.5
2 NAME Legacy (R)
2 CORP Millennia Corp.
3 ADDR PO Box 9410
4 CONT Surprise, AZ 85374
1 DEST Legacy
1 DATE 14 Nov 2013
1 SUBM @S0@
1 FILE \\vmware-host\Shared Folders\Documentos\genealogia\legacy\paraDiogo.ged
1 GEDC
2 VERS 5.5.1
2 FORM LINEAGE-LINKED
1 CHAR UTF-8
0 @S0@ SUBM
1 NAME Carlos Heuser
```

Figura 2.1: Cabeçalho de um documento GEDCOM

Após o cabeçalho, os registros que se seguem se referem a indivíduos. Portanto sendo os registros mais importantes. Um registro de indivíduo reúne informações como nome, sobrenome, local de nascimento, data de nascimento, local de falecimento, data de falecimento entre outras informações. Além disso, podem conter referências a registros de fonte e de família. Na Figura 2.2 pode ser visto como um registro de indivíduo se parece.

```
0 @I3@ INDI
1 NAME Jeannette (Joanna) /Diefenthaeler/
2 GIVN Jeannette (Joanna)
2 SURN Diefenthaeler
1 SEX F
1 BIRT
2 DATE 18 May 1826
2 PLAC São Leopoldo, Rio Grande do Sul, Brasil
2 SOUR @S39@
3 PAGE Livro 3 - Batismos
3 QUAY 3
3 DATA
4 TEXT JEANNETTE DIEFENTHÄLER
5 CONT pai: JACOB DIEFENTHÄLER, de Biebelnheim, em Groß-Hessen, 4
5 CONC 1 anos, morador na Costa da Serra, agricultor;
5 CONT mãe: ELISABETH, nasc. DIEHL, 37 anos, evangélica;
5 CONT padrinhos: Jeannette Fajette; Franz Wilhelm Petersen;
5 CONT filha: JEANNETTE, nasc. a 18 de maio; bat. em 28 de maio, p
5 CONC or autorização do Pastor Voges
5 CONT                              por J. G. Ehlers
1 DEAT
2 DATE 28 Feb 1916
2 PLAC Estância Velha, Rio Grande do Sul, Brasil
2 SOUR @S54@
1 _TAG
1 _UID EFD4716629B34C71BBF2BD0EDF78E5C0A7DE
1 CHAN
2 DATE 27 Dec 2007
1 FAMS @F3@
1 FAMC @F1154@
```

Figura 2.2: Exemplo do registro de um Indivíduo (INDI) de um arquivo GEDCOM

Após os registros dos indivíduos, se seguem os registros de fontes. Os registros de fontes são usados para prover uma descrição bibliográfica da fonte citada. Na Figura 2.3 pode ser visto um exemplo de um registro de fonte.

```
0 @S49@ SOUR
1 ABBR A história da Comunidade Evangélica em Estância Velha
1 TITL A história da Comunidade Evangélica em Estância Velha - Igr
2 CONC eja Apóstolo Paulo - IECLB
1 AUTH Paulo Garlip Filho
1 PUBL 2000
1 _PAREN Y
```

Figura 2.3: Exemplo de um registro de uma Fonte (SOUR) de um arquivo GEDCOM

Seguindo-se aos registros de fontes, aparecem os registros de famílias. Eles são utilizados para registrar casamentos e uniões familiares, quando dois indivíduos se tornam pais. Não pode existir mais do que um pai/marido e uma mãe/esposa listados em cada registro de família. Preferivelmente os filhos são ordenados de forma cronológica. Na Figura 2.4 encontra-se um exemplo de um registro de família.

```
0 @F8@ FAM
1 HUSB @I2875@
2 _PREF Y
1 WIFE @I9@
2 _PREF Y
1 MARR
2 DATE 9 Feb 1796
2 PLAC Steinwenden, Rheinhessen, Hessen-Darmstadt, Deutschland
2 SOUR @S67@
3 PAGE 491-492
3 QUAY 3
2 SOUR @S41@
3 PAGE 112
3 QUAY 3
2 NOTE @NF8@
3 SOUR @S67@
4 PAGE 491-492
4 QUAY 3
3 SOUR @S41@
4 PAGE 146
4 QUAY 3
1 CHIL @I1338@
2 _PREF Y
1 CHIL @I3173@
1 CHIL @I3163@
1 CHIL @I1337@
1 CHIL @I69@
1 CHIL @I3174@
1 CHIL @I163@
1 CHAN
2 DATE 7 Nov 2009
```

Figura 2.4: Exemplo de um registro de Família (FAM) de um arquivo GEDCOM

O último registro, que é requerido em um arquivo GEDCOM é chamado de TRAILER, que sinaliza o fim do arquivo.

```
0 TRLR
```

Figura 2.5: Linha que marca o final de um arquivo GEDCOM

3 WEKA E SUA API

3.1 Apresentação

Com o passar dos anos, a experiência mostrou que não existe um algoritmo ou mesmo uma combinação de algoritmos que possa resolver todos os problemas de mineração de dados. Isso porque cada base de dados tem suas particularidades e para se fazer previsões sobre estes dados, é necessário que se conheça o seu domínio de antemão.

WEKA (Waikato Environment for Knowledge Analysis) é uma coleção de ferramentas e algoritmos de aprendizado de máquina para mineração de dados. Foi desenvolvido e é ativamente mantido pela Universisade de Waikato, na Nova Zelândia. As versões mais recentes foram desenvolvidas em Java. E é disponibilizado sob a licença GNU. WEKA também é o nome de um pássaro local da Nova Zelândia, que não voa.

Além da coleção de algoritmos de aprendizado de máquina, que foram mencionados, o WEKA contém ferramentas para pré processamento de dados, classificação, regressão, agrupamento, regras de associação e visualização. Para utilizar estas ferramentas o WEKA disponibiliza uma aplicação de linha de comando e uma aplicação com interface gráfica. Outra forma de se utilizar suas ferramentas é através de código, tipicamente código que roda na máquina virtual do Java (JVM). Para isso, o WEKA mantém uma biblioteca que pode ser chamada em programas pessoais.

Esta última abordagem foi a utilizada no presente trabalho. E nas próximas seções será mostrada a API do WEKA, o que foi utilizado dela e como ela foi utilizada.

3.2 A API do WEKA

Como foi mencionado anteriormente, o WEKA também é disponibilizado através de uma biblioteca, que foi desenvolvida em Java e portanto pode ser utilizada de qualquer linguagem que rode no ambiente da Máquina Virtual do Java. Sua API é documentada usando

Javadoc (http://weka.sourceforge.net/doc.stable/). Na sua documentação, os protótipos das funções são mostrados, detalhando tipo da saída, parâmetros de entrada com os seus tipos. Também são mostrados alguns exemplos de uso.

3.2.1 Formato ARFF

Para se utilizar a API do WEKA é necessário que se tenha, antes de mais nada, uma base de dados na qual se deseja classificar, agrupar, ou efetuar qualquer das operações que o WEKA disponibiliza. Para tanto, o WEKA estabeleceu um formato para que ele possa processar conteúdo.

O nome deste formato é ARFF. Um arquivo ARFF (*Attribute-Relation File Format*) é um arquivo texto que descreve uma lista de instâncias compartilhando um conjunto de atributos. Ele foi criado pelo Projeto de Aprendizado de Máquina no Departamento de Ciência da Computação da Universidade de Waikato.

Arquivos ARFF são arquivos muito simples. Eles basicamente tem duas seções. A primeira seção é o Cabeçalho, que é seguido pela seção de Dados. O Cabeçalho do arquivo ARFF contém o nome da relação, a lista de atributos e seus tipos. E a seção de Dados contém os dados em si, separados por vírgula, de forma muito semelhante a um arquivo CSV. Na Figura 3.1 pode-se ver um exemplo de como um arquivo ARFF se parece.

```
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature numeric
@attribute humidity numeric
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
overcast,83,86,FALSE,yes
rainy,70,96,FALSE,yes
rainy,68,80,FALSE,yes
rainy,65,70,TRUE,no
overcast,64,65,TRUE,yes
sunny,72,95,FALSE,no
sunny,69,70,FALSE,yes
rainy,75,80,FALSE,yes
sunny,75,70,TRUE,yes
overcast,72,90,TRUE,yes
overcast,81,75,FALSE,yes
rainy,71,91,TRUE,no
```

Figura 3.1: Exemplo de um arquivo ARFF

Cada atributo descrito na Figura 3.1 caracteriza uma coluna, informando o seu nome e tipo de dados contido nela. E cada linha da seção de Dados representa um registro, também chamados de instâncias.

3.2.2 Exemplos de uso

Tendo a base de dados em mão no formato que a API do WEKA compreende, pode-se então começar a processar de maneira que se deseja estes dados. Uma forma de aprendizado de máquina bastante popular é o aprendizado de máquina para classificação. Que tem o intuito de fazer previsões sobre uma base de dados, partindo de uma base de dados pré classificada. Os exemplos que serão mostrados nesta seção focam na apresentação de como fazer previsões sobre uma base de dados utilizando-se o WEKA.

Para tanto, é necessário inicialmente definir-se uma base de treinamento, que deve conter dados previamente classificados e servirá para treinar o algoritmo de classificação. A partir destes dados o algoritmo de classificação construirá a estrutura que vai permitir que ele classifique dados desconhecidos (faça previsões).

A partir da base a qual se quer classificar e da base de treinamento, a primeira coisa a ser feita é ler um arquivo ARFF. Para que se possa ler estas duas bases de dados. Além disso, é necessário também que se escolha a classe, ou seja o atributo que servirá para classificar as instâncias da base de dados. Na Figura 3.2 segue um exemplo de como se fazer isso.

```
1 import weka.core.Instances;
2 import java.io.BufferedReader;
3 import java.io.FileReader;
4
5 // Leitura do arquivo ARFF
6 BufferedReader reader = new BufferedReader(new FileReader("/some/where/data.arff"));
7 Instances data = new Instances(reader);
8 reader.close();
9 // Escolhendo a classe
10 data.setClassIndex(data.numAttributes() - 1);
```

Figura 3.2: Exemplo de leitura de um arquivo ARFF utilizando a API do WEKA

Após a leitura dos arquivos ARFF pode-se utilizá-los para fazer o treinamento do algoritmo de classificação escolhido. Uma abordagem conhecida é misturar uma base de treinamento pré selecionada com algumas instâncias de uma base de teste e então utilizar esta base compilada, aplicando a técnica de *cross-validation.*

Esta técnica consiste em escolher um número de partições. Suponha-se que seja escolhido 10. Então os dados são divididos em 10 partes aproximadamente iguais. A seguir este processo é repetido 10 vezes. Sendo que em cada uma das vezes, uma das 10 partes é

utilizada para teste e as restantes para treinamento. Desta forma cada instância será utilizada somente uma vez para teste. O número 10 parece ter sido escolhido de forma arbitrária, mas este valor é amplamente aceito. Alegadamente por gerar a melhor estimativa de erro, além de ser suportado por algumas evidências teóricas (WITTEN et al., 2011). Na figura 3.3 segue um exemplo.

```
 1 import java.io.BufferedReader;
 2 import java.io.FileReader;
 3 import weka.core.Instances;
 4 import weka.classifiers.Evaluation;
 5 import weka.classifiers.trees.J48;
 6 import java.util.Random;
 7
 8 // Leitura do arquivo ARFF
 9 BufferedReader reader = new BufferedReader(new FileReader("/some/where/data.arff"));
10 Instances trainAndTest = new Instances(reader);
11 reader.close();
12 // Algoritmo de classificação
13 J48 tree = new J48();
14 Evaluation eval = new Evaluation(trainAndTest);
15 // Rodando o cross-validation
16 eval.crossValidateModel(tree, trainAndTest, 10, new Random(1));
```

Figura 3.3: Exemplo de treinamento de um classificador utilizando-se da técnica de *cross-validation*

Por fim, tendo treinado o classificador com as instâncias de treinamento, pode-se fazer as previsões. Na Figura 3.4 há um exemplo de como se fazer isso. Note que o exemplo da Figura 3.4 se utiliza de partes do exemplo da Figura 3.3.

```
 1 import java.io.BufferedReader;
 2 import java.io.BufferedWriter;
 3 import java.io.FileReader;
 4 import java.io.FileWriter;
 5 import weka.core.Instances;
 6
 7 ... // Trecho de código do exemplo anterior
 8
 9 // Carrega dados aos quais se deseja fazer as previsões
10 Instances toBeClassified = new Instances(new BufferedReader(
11 new FileReader("/some/where/tobeclassified.arff")));
12 // Escolhe-se o atributo que será a classe
13 toBeClassified.setClassIndex(toBeClassified.numAttributes() - 1);
14 // Cria-se uma cópia
15 Instances classified = new Instances(toBeClassified);
16
17 // Faz-se as previsões utilizando-se do classificador
18 for (int i = 0; i < toBeClassified.numInstances(); i++) {
19         double clsLabel = tree.classifyInstance(toBeClassified.instance(i));
20         classified.instance(i).setClassValue(clsLabel);
21 }
22  // Salva-se os dados classificados em um arquivo
23  BufferedWriter writer = new BufferedWriter(
24 new FileWriter("/some/where/classified.arff"));
25  writer.write(labeled.toString());
26  writer.newLine();
27  writer.flush();
```

Figura 3.4: Exemplo de como se fazer previsões utilizando-se um classificador previamente treinado

4 FERRAMENTA

Em alto nível, a aplicação permite que o usuário envie uma base de dados GEDCOM na forma de um arquivo, juntamente com um arquivo CSV contendo uma lista de identificadores (originários da base dados) agrupados dois a dois, caracterizando os registros que o usuário sabe que se referem a mesma pessoa.

A partir de então o usuário deve selecionar uma lista de evidências, que servirão para comparar registros entre si, além de um algoritmo para cada evidência, que será utilizado para efetuar o cálculo de sua similaridade. É importante notar que os algoritmos disponíveis para seleção tem relação com o tipo do dado da evidência. As evidências disponíveis para seleção são: nome, sobrenome, data de nascimento, data de morte, local de nascimento e local de morte. Pode-se notar que as evidências tem basicamente dois tipos de dados: string e data. Por conseguinte, os algoritmos disponibilizados atendem a estes dois tipos de dado.

Para as evidências do tipo data, está disponível apenas um algoritmo. Ele faz uma comparação simples entre a distância entre as datas, normalizando o resultado para que se encaixe no intervalo de 0 a 1.

Para as evidências do tipo string, são disponibilizados para escolha do usuário os seguintes algoritmos para o cálculo de sua similaridade: Jaro-Winkler, Levenshtein, N-Gram (3 grams), Overlap, Ratcliff-Obershelp e Weighted Levenshtein.

Por último, o usuário deve escolher o algoritmo de classificação, que tem a função de fazer previsões para a classe escolhida. No caso desta ferramenta, a classe é um valor booleano que determina se dois registros se referem a mesma pessoa. A previsão é feita levando em consideração dados de teste que são gerados a partir do resultado das similaridades das evidências para cada dois registros. Os algoritmos disponibilizados são: ADTree, Decision Stump, j48 e j48graft. Todos eles disponibilizados pela biblioteca WEKA.

O propósito desta aplicação é fornecer uma ferramenta para facilitar a experimentação e descoberta da combinação ideal de evidências, algoritmos para calcular a similaridade entre

elas e algoritmo de classificação, que gere uma melhor sugestão dos registros duplicados em uma dada base de dados. É importante salientar também que esta ferramenta tem como público alvo usuários especialistas, que tenham pelos menos conhecimento superficial de bancos de dados e algoritmos.

Neste capítulo serão discorridos detalhes das funcionalidades providas pela aplicação. Para tanto, utilizar-se-á da linguagem UML (*Unified Modeling Language*). Tal linguagem é amplamente utilizada na área de Engenharia de Software para documentação e comunicação de relacionadas ao desenvolvimento de *software*. Especialmente *software* utilizando o paradigma de programação orientado a objetos (FOWLER, 2005).

4.1 Entradas da aplicação

No contexto do presente trabalho, são utilizados arquivos GEDCOM como entrada da aplicação, juntamente com um arquivo CSV. O arquivo GEDCOM informa à aplicação a base de dados que pode conter duplicatas e o arquivo CSV informa os registros que sabidamente se referem às mesmas pessoas.

Estas entradas são processadas pela aplicação, que gera como saída sugestões de registros que possam se referir às mesmas pessoas.

4.2 Diagramas de caso de uso

Diagramas de caso de uso em UML, tem o intuito de representar graficamente como os usuários interagem com um sistema. Um ponto interessante do diagrama de casos de uso da UML, é que se pode explicitar as permissões de cada tipo de usuário (Atores). No caso da aplicação referida neste trabalho, existe apenas um ator. A Figura 4.1 mostra o diagrama de casos de uso da aplicação.

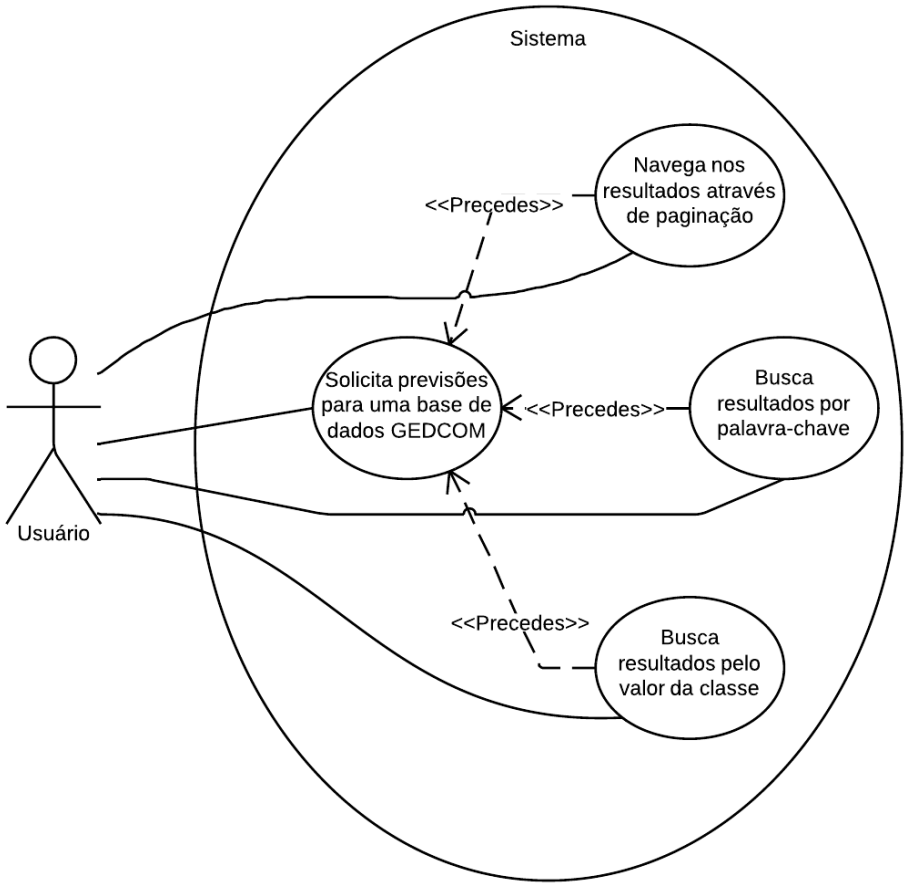

Figura 4.1: Diagrama de casos de uso da aplicação

No diagrama de casos de uso da Figura 4.1 pode ser visto um único ator. Ele representa o usuário do sistema, tendo garantidas permissões para executar todas as tarefas disponíveis.

Pode-se notar que há entre o ator e os casos de uso, uma associação, representada por um arco ligando as duas partes no diagrama, que caracteriza a interação entre o ator e alguma funcionalidade do sistema. Além disso há também uma associação entre os casos de uso. Esta associação denota uma ordem de precedência. No diagrama, o caso de uso “Solicita previsões para uma base de dados GEDCOM” precede os casos de uso “Navega nos resultados através de paginação”, “Busca resultados por palavra-chave” e “Busca resultados pelo valor da classe”.

O caso de uso “Solicita previsões para uma base de dados GEDCOM” tem por finalidade possibilitar ao usuário obter previsões de quais registros de um arquivo GEDCOM se referem ao mesmo indivíduo, através do envio de uma base GEDCOM juntamente com um

arquivo CSV contendo informações de registros que sabidamente se referem ao mesmo indivíduo.

Os casos de uso “Navega nos resultados através de paginação”, “Busca resultados por palavra-chave” e “Busca resultados pelo valor da classe” permitem ao usuário, em suma, filtrar os resultados e visualizar os resultados referentes aos indivíduos que mais lhe interessam.

4.3 Descrições textuais dos casos de uso e telas de interface gráfica

Nesta seção serão apresentadas descrições textuais dos casos de uso vistos na seção anterior na Figura 4.1, acompanhada das telas de interface gráfica. Diagramas de caso de uso são importantes para explicitar graficamente as funcionalidades disponibilizadas pela aplicação. Contudo, eles não informam detalhes da interação entre o usuário e a aplicação. Estas informações, juntamente com a lógica do processo do caso de uso, se somam as informações já fornecidas pelo diagrama, enriquecendo a descrição dos casos de uso.

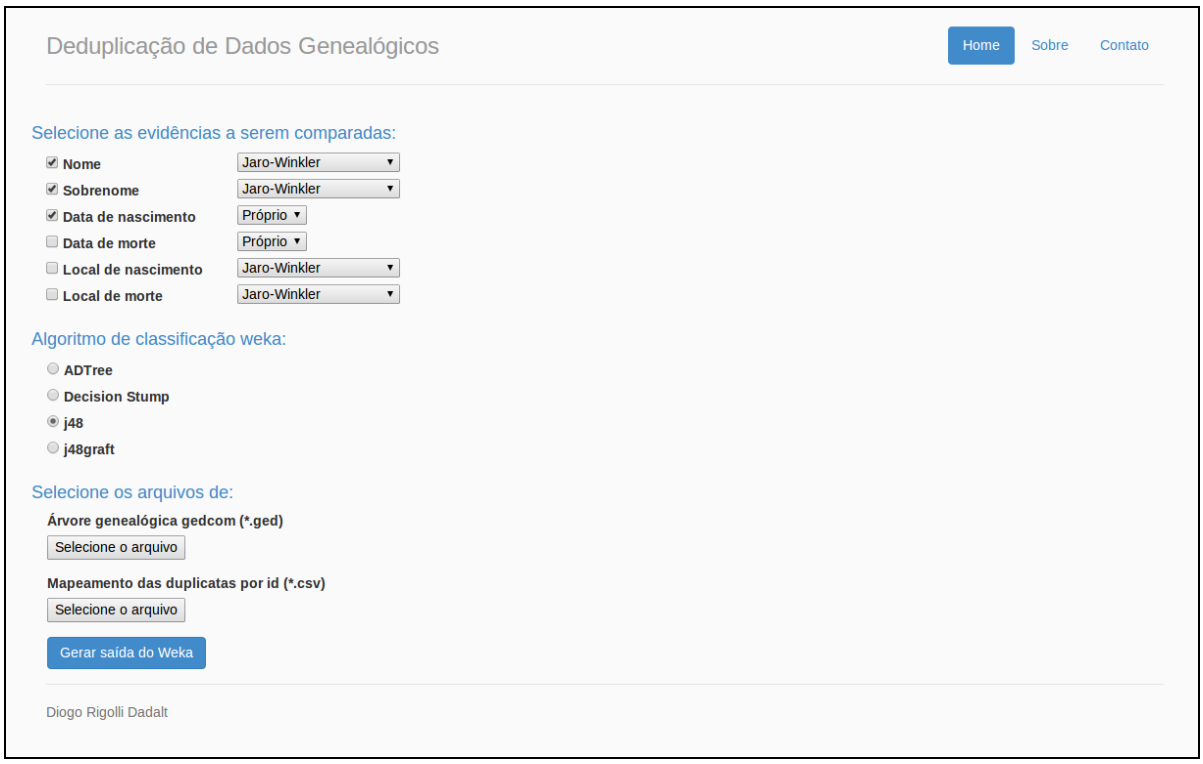

Figura 4.2: Tela inicial da aplicação

A Figura 4.2 mostra a tela inicial da aplicação. Nela pode ser visto que há uma área a esquerda que contém uma série de componentes que permitem o usuário interagir com a aplicação. O lado direito, que na imagem aparece em branco, é onde são mostrados os resultados do processamento da aplicação.

Tabela 4.1: Descrição do caso de uso “Solicita previsões para uma base de dados GEDCOM”

UC 1 - Solicita previsões para uma base de dados GEDCOM	
Atores:	Usuário.
Pré-condição:	Nenhuma.
Pós-condição:	Previsões quanto a se cada indivíduo tem duplicatas ou não, são mostradas.
Descrição:	O usuário envia um arquivo GEDCOM contendo a base de dados a qual ele deseja detectar indivíduos duplicados, juntamente com um arquivo CSV contendo ids de indivíduos que sabidamente tem duplicatas, associando estes ids aos ids das duplicatas. O usuário deve também selecionar as evidências que deseja, e para cada evidência o algoritmo de comparação a ser utilizado. Por fim, ele deve selecionar o algoritmo de classificação. Previsões relativas a cada indivíduo são mostradas.
Sequência de eventos	
Ator	Sistema
1. Marca as evidências que deseja que a aplicação leve em consideração no momento da comparação entre indivíduos.	
2. Seleciona o algoritmo de comparação para cada evidência marcada.	
3. Seleciona o algoritmo de classificação.	
4. Seleciona o arquivo contendo a base de dados gedcom para ser enviada.	
5. Seleciona o arquivo CSV contendo alguns ids de indivíduos com duplicatas, associando a cada id de dulicata.	
6. Clica no botão “Gerar previsões”.	
	7. Exibe uma lista contendo em cada linha a combinação dois a dois de cada indivíduo com os

	demais indivíduos e a previsão se esse se refere a mesma pessoa.
8. Visualiza a tabela de resultados e o caso de uso é encerrado.	
RN 1 - Regra de Negócio 1:	
1. A partir dos ids de duplicatas do arquivo CSV e de uma lista de ids escolhidos aleatoriamente na base de dados GEDCOM, será gerada a lista de instâncias que servirá para treinamento.	

A Tabela 4.1 detalha o caso de uso “Solicita previsões para uma base de dados GEDCOM”, mostrando os passos tomados pelo usuário e a resultante consequência no sistema.

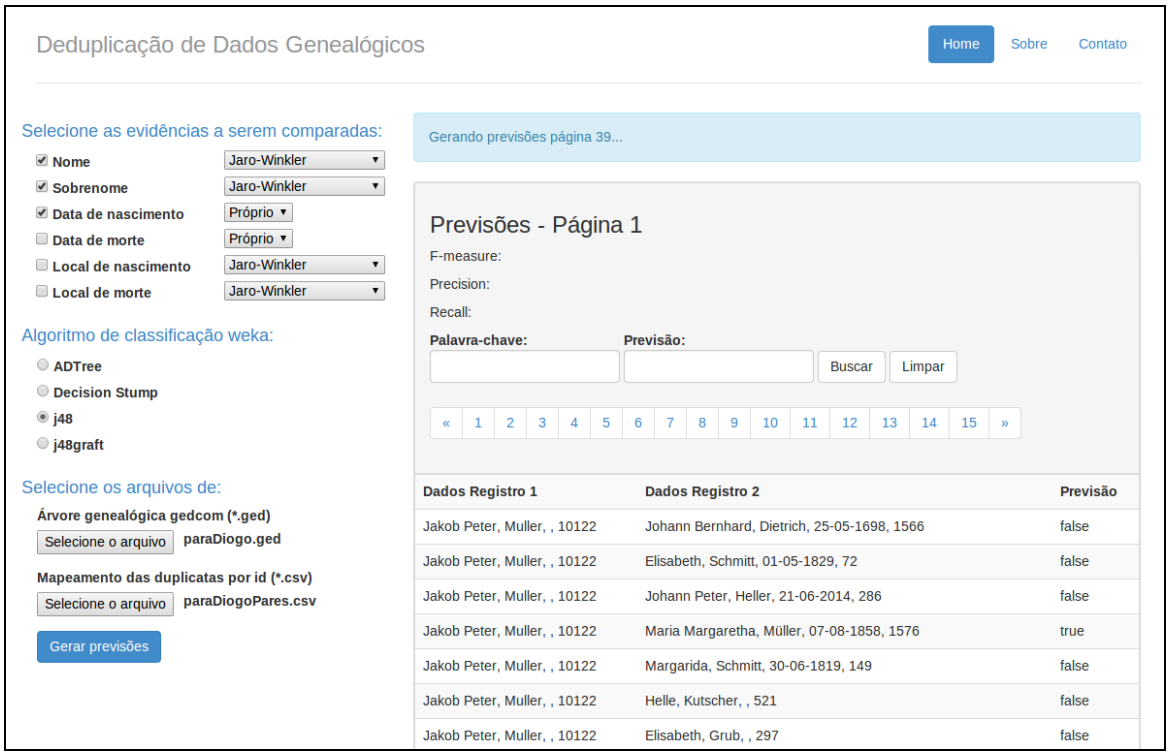

Figura 4.3: Paginação dos resultados (em cima da tabela)

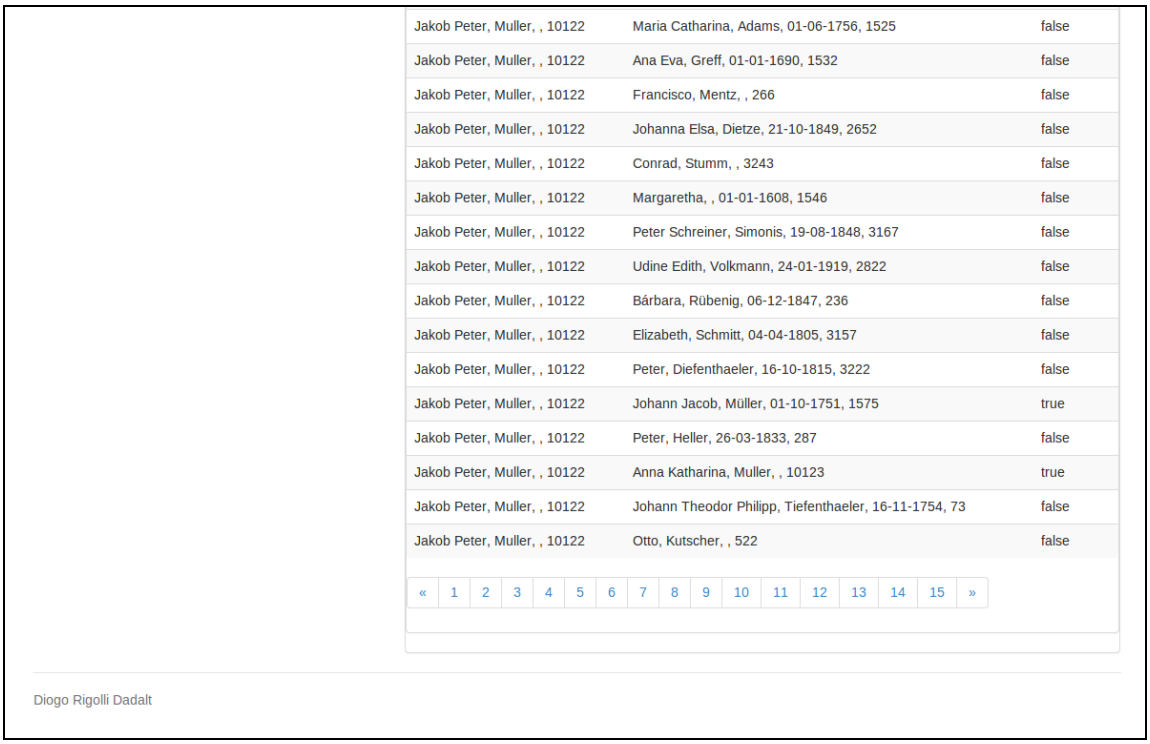

Figura 4.4: Paginação dos resultados (embaixo da tabela)

As figuras 4.3 e 4.4 mostram em seu lado direito os resultados da processamento do arquivo GEDCOM. Logo acima da tabela, na Figura 4.3 e logo abaixo da tabela, na Figura 4.4, encontra-se o componente de paginação. Que permite ao usuário navegar pelas páginas dos resultados.

Tabela 4.2: Descrição do caso de uso “Navega nos resultados através de paginação”

UC 2 - Navega nos resultados através de paginação	
Atores:	Usuário.
Pré-condição:	Resultados devem ter sido gerados e devem exceder o número de itens que cabem em uma página.
Pós-condição:	Os resultados referentes à página selecionada são mostrados.
Descrição:	O usuário clica em um link referente a uma determinada página e os resultados desta são mostrados.
Sequência de eventos	
Ator	Sistema
1. Clica em um link no componente de paginação, referente a uma determinada	

página.	
	2. Mostra os resultados referentes à página escolhida, informando ao usuário em que página ele se encontra, na parte superior da tela.
3. Visualiza a tabela de resultados referente à página escolhida e o caso de uso é encerrado.	

Na Tabela 4.2 é detalhada a navegação dos resultados através da paginação. É mostrado a descrição do caso de uso, bem como os passos da interação entre o usuário e o sistema.

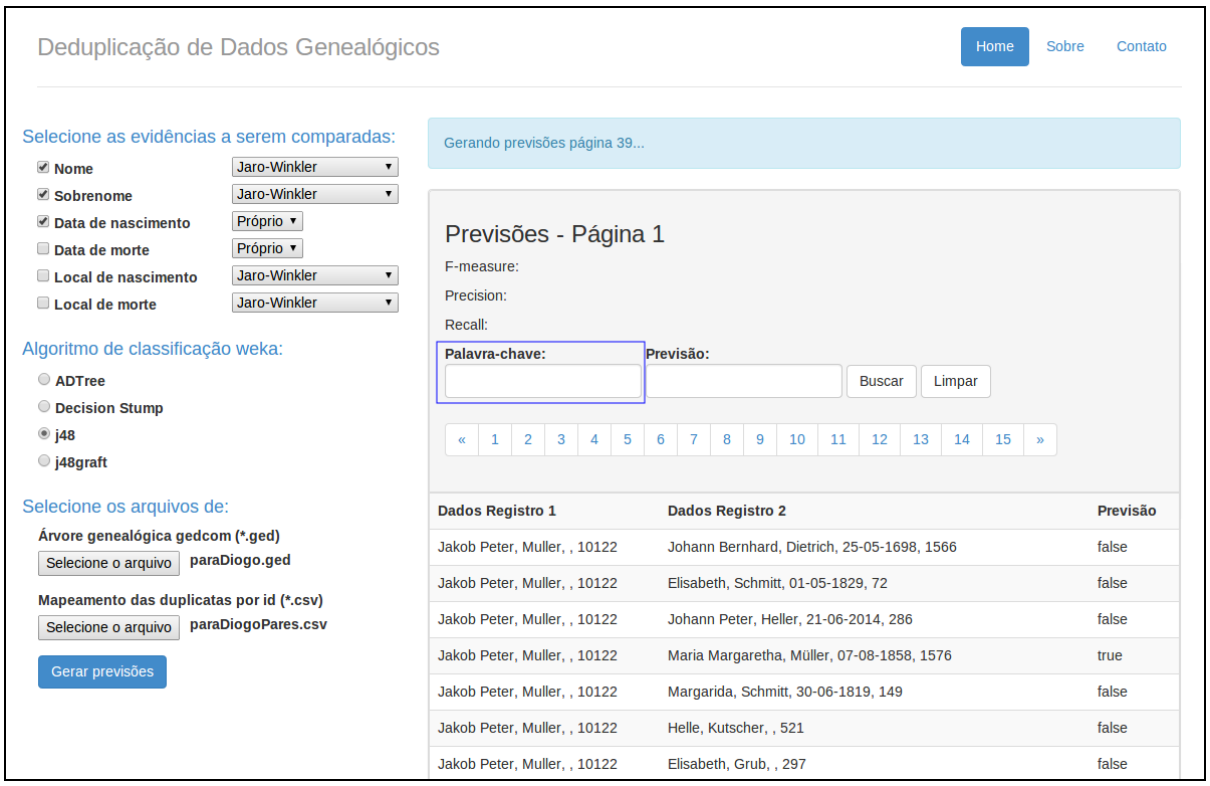

Figura 4.5: Filtragem dos resultados por palavra-chave

A Figura 4.5 mostra a funcionalidade de filtragem de resultados por palavra-chave. O componente que propicia isto está destacado por um quadrilátero transparente de borda azul. Através desta funcionalidade o usuário pode filtrar resultados, escolhendo uma palavra-chave que esteja em qualquer dos campos.

Tabela 4.3: Descrição do caso de uso “Busca resultados por palavra-chave”

UC 3 - Busca resultados por palavra-chave	
Atores:	Usuário.
Pré-condição:	Resultados devem ter sido gerados.
Pós-condição:	Os resultados referentes à palavra-chave escolhida são mostrados.
Descrição:	O usuário pesquisa pelo termo de sua escolha, que é usado para filtrar os resultados, aplicando a busca em todos os campos dos resultados e mostrando apenas os resultados que contém o termo.
Sequência de eventos	
Ator	Sistema
1. Digita o termo de sua escolha no campo de texto “palavra-chave”.	
2. Clica no botão “Buscar”.	
	3. Mostra os resultados que contém a palavra-chave escolhida. Se o resultado tem mais itens do que cabe em uma página, o componente de paginação também é mostrado e o usuário é levado à primeira página.
4. Visualiza a tabela de resultados referente à palavra-chave escolhida e o caso de uso é encerrado.	

A Tabela 4.3 mostra a descrição do caso de uso “Busca resultados por palavra-chave”. É importante ressaltar que além de propiciar ao usuário fazer a filtragem dos resultados por palavra-chave. O usuário também poderá se utilizar da funcionalidade de paginação combinada com esta última funcionalidade, no caso de haver mais resultados do que a quantidade estabelecida para ser mostrada por página.

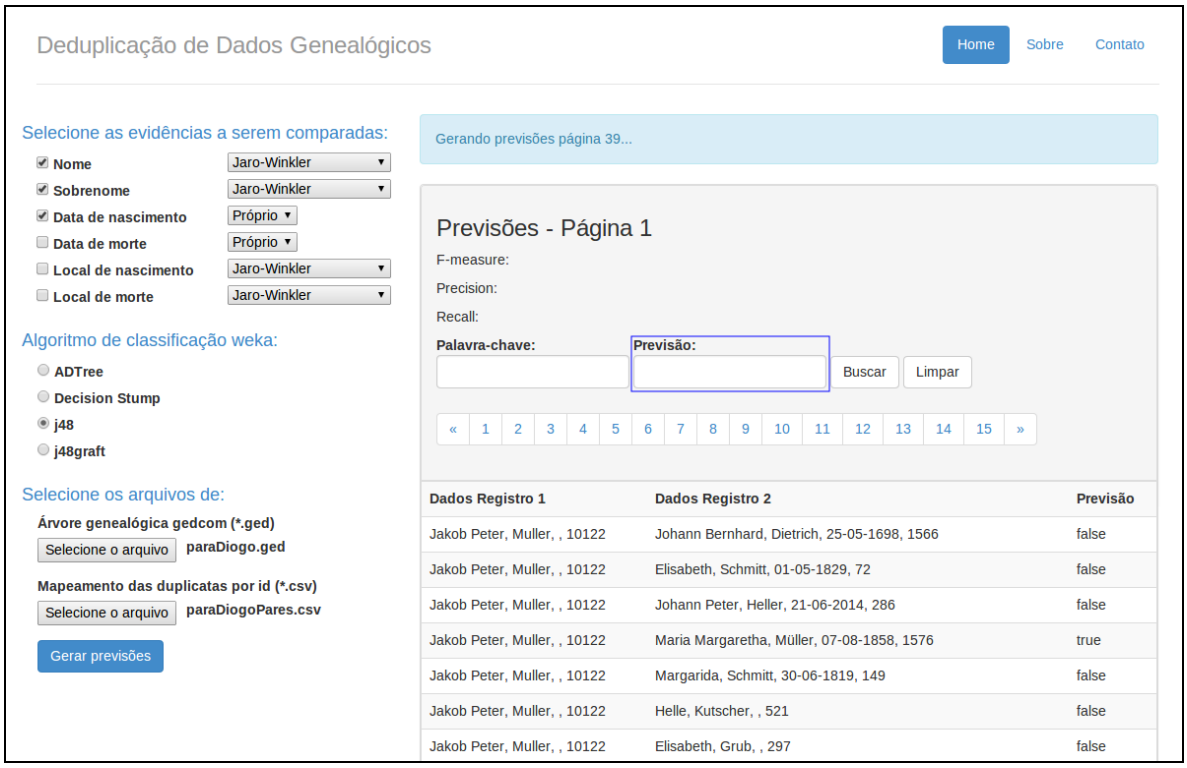

Figura 4.6: Filtragem dos resultados pelo resultado da previsão

A Figura 4.6 mostra a funcionalidade de filtragem de resultados pelo resultado da previsão. O componente que propicia isto está destacado por um quadrilátero transparente de borda azul. Através desta funcionalidade o usuário pode filtrar resultados, escolhendo o resultado da previsão.

Tabela 4.4: Descrição do caso de uso “Busca resultados pelo valor da classe”

UC 4 - Busca resultados pelo valor da classe	
Atores:	Usuário.
Pré-condição:	Resultados devem ter sido gerados.
Pós-condição:	Os resultados referentes ao valor da classe são mostrados.
Descrição:	O usuário pesquisa pelo valor da classe que deseja, que é usado para filtrar os resultados. O sistema, mostra apenas os resultados que contém o valor requisitado.
Sequência de eventos	
Ator	Sistema
1. Digita o valor da classe que deseja no campo de texto “previsão”.	
2. Clica no botão “Buscar”.	

	3. Mostra os resultados que contém o valor escolhido como previsão. Se o resultado tem mais itens do que cabe em uma página, o componente de paginação também é mostrado e o usuário é levado à primeira página.
4. Visualiza a tabela de resultados referente ao valor da classe escolhida e o caso de uso é encerrado.	

A Tabela 4.4 mostra a descrição do caso de uso “Busca resultados pelo valor da classe”. É importante ressaltar que além de propiciar ao usuário fazer a filtragem dos resultados pelo valor da classe (ou previsão). O usuário também poderá se utilizar da funcionalidade de paginação combinada com esta última funcionalidade, no caso de haver mais resultados do que a quantidade estabelecida para ser mostrada por página. Além disso, também é possível fazer esta filtragem juntamente com uma palavra-chave, desta forma mostrando apenas resultados que contém a palavra-chave e a previsão escolhidas.

5 ARQUITETURA DA APLICAÇÃO

Esta seção aborda a arquitetura da aplicação, detalhando-a em vários níveis. Para tanto, utilizar-se-á novamente de diagramas UML, particularmente diagramas de classe e de sequência. Além disso, serão utilizados outros diagramas, apresentando uma visão geral da arquitetura e a visão de implantação da aplicação. Também serão apresentadas as tecnologias envolvidas.

5.1 Visão geral

A aplicação, basicamente, segue a arquitetura cliente-servidor. Sendo a aplicação cliente totalmente desacoplada da aplicação servidor. Portanto, na verdade, são duas aplicações. Na Figura 5.1 é mostrada uma visão alto nível da arquitetura geral, mostrando como os componentes e tecnologias envolvidas interagem.

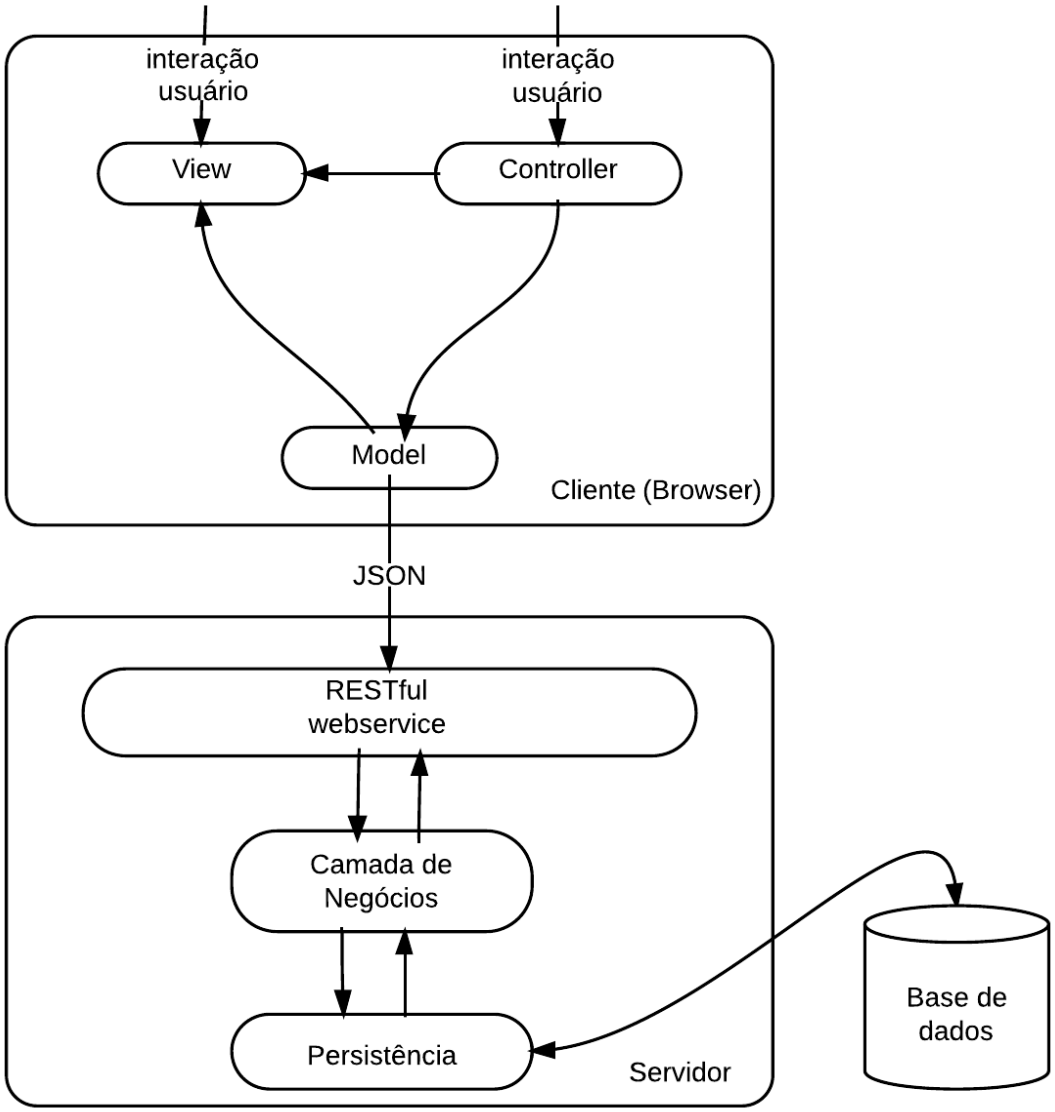

Figura 5.1: Visão geral da arquitetura do sistema

A Figura 5.1 mostra uma visão geral da arquitetura da aplicação. Como foi mencionado anteriormente, e também pode-se ver na figura, a aplicação tem uma arquitetura cliente-servidor. E dado que a aplicação cliente é totalmente desacoplada da aplicação servidor, percebe-se que a aplicação está preparada para escalar. Podendo-se ter diversos clientes rodando, fazendo requisições ao mesmo servidor.

Os elementos contidos na Figura 5.1 bem como as tecnologias que eles envolvem serão detalhados a seguir. Isto é necessário, para que se tenha um melhor entendimento de como eles interagem.

5.1.1 Aplicação cliente

Como mostrado na Figura 5.1, a aplicação cliente é uma aplicação desenvolvida para rodar em um navegador e utiliza o padrão arquitetural MVC (Model View Controller). Isto acontece com a ajuda de algumas tecnologias, conceitos e bibliotecas. Elas serão descritas abaixo.

HTML (*Hypertext Markup Language*) - É uma linguagem de marcação que descreve a estrutura de uma página *web.*

CSS (*Cascading Style Sheets*) - linguagem que tem o intuito de prover uma forma de descrever as características visuais de componentes do HTML, como botões, caixas de textos e *links.* Alguns exemplos do que pode ser descrito utilizando CSS são cores, medidas e posicionamento.

JavaScript - linguagem interpretada que roda no navegador. Permite o acesso programático aos elementos de uma página além de permitir acesso remoto através do AJAX.

Com os navegadores ficando cada dia mais sofisticados, o JavaScript tem ganho grande importância. Parte da lógica de uma aplicação *web* que se costumava residir no lado servidor, hoje em dia reside no lado cliente, fazendo com que a separação das responsabilidades fique bem mais clara e que o acoplamento diminua.

AJAX (*Asynchronous JavaScript And XML*) - conceito implementado pelos navegadores que permite chamadas assíncronas a serviços *web*, inicialmente utilizando como comunicação o formato XML. Atualmente é muito mais comum utilizar-se o formato JSON para comunicar dados e eventualmente até o formato HTML para transferir trechos de uma página.

JSON (*JavaScript Object Notation*) - formato de comunicação de dados que se utiliza de elementos da sintaxe do JavaScript e surgiu como uma alternativa ao XML. É amplamente suportado por servidores e navegadores. E ganhou popularidade por ser bem menos verboso do que o XML e por interagir melhor com JavaScript.

Backbone.js - biblioteca de JavaScript que tem o intuito de prover recursos para facilitar o desenvolvimento de aplicações do lado cliente, utilizando-se de padrões arquiteturais como o MVC. Além disso provê recursos para controlar-se o histórico do navegador do usuário, e também para mapear rotas (quando o usuário digita algo na barra de endereços) para ações do JavaScript.

Mustache.js - biblioteca que visa prover a funcionalidade de gerar trechos de HTML, tendo como entrada um *template* e dados (JSON).

A aplicação cliente se comunica com a aplicação do lado servidor, utilizando-se da linguagem JavaScript, fazendo chamadas AJAX para as APIs disponibilizadas pela aplicação servidor.

5.1.2 Aplicação servidor

REST (Representational State Transfer) - é um estilo arquitetural para serviços *web* que visa prover funcionalidade de criação, atualização, busca e remoção de entidades através de métodos HTTP sem que se mantenha estado no lado do servidor. Ou seja, cada chamada deve conter todas as informações necessárias para o seu processamento. Esta abordagem é uma alternativa a SOAP, CORBA e RPC, tendo como vantagem sobre estas o fato de ser mais fácil de escalar, justamente porque ela prega que não se deve guardar estado. No contexto desta aplicação, este conceito foi utilizado com a ajuda do *framework* Play para a linguagem de programação Scala.

Banco de Dados - conjunto organizado de registros que se relacionam de alguma forma. A aplicação em questão utiliza o banco de dados MySQL.

Além dos conceitos citados acima é importante falar que a aplicação do lado servidor foi implementada utilizando-se a linguagem de programação Scala, que é uma linguagem que provê tanto o paradigma orientado a objetos como o paradigma funcional. A comunicação entre a aplicação cliente e a aplicação servidor ocorre através de chamadas AJAX do lado cliente a API disponibilizada do lado servidor, e elas trocam informações através do formato JSON. A partir de então a camada de *web service* propaga a informação para a camada de negócio que por sua vez propaga a informação para a camada de persistência que então se comunica com o banco de dados.

5.2 Modelo de dados

O modelo de dados da aplicação é bem simples. São apenas três entidades representadas nas tabelas: report, result e ftr. A tabela “ftr” representa um indivíduo da base de dados GEDCOM com seus campos (ou evidências) todas armazenadas no campo “fields“ no formato JSON. A tabela “result” representa a combinação entre dois indivíduos e a previsão se eles se referem a mesma pessoa. A tabela “report” representa os resultados de uma

requisição de previsões em cima de uma base GEDCOM, contendo uma lista de resultados, tabela “result”. A Figura 5.2 mostra um diagrama das tabelas do sistema.

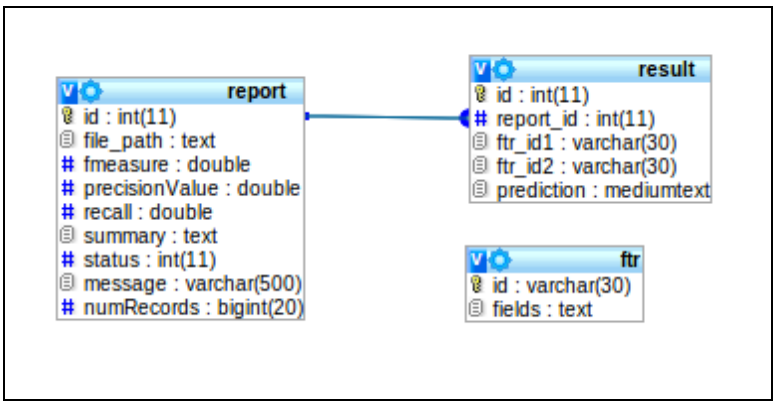

Figura 5.2: Representação do modelo de dados

5.3 Diagramas de classes

Afim de aprofundar o entendimento do sistema, nesta seção serão detalhados seus componentes internos, chegando a granularidade de classe. Em cada uma das duas aplicações, serão abordadas separadamente cada camada.

Aplicação web - por simplicidade, a aplicação web será tratada aqui como uma camada. Ela pode ser pensada como a camada de apresentação do sistema como um todo. Mas sendo vista como uma aplicação independente, e se utilizando do padrão arquitetural MVC, ela tem as suas camadas internas.

Camada de *web service* - a camada de *web service* é a fachada da aplicação do lado servidor. Ela disponibiliza funcionalidades na forma de API, que pode ser utilizadas por aplicações externas, através de chamadas remotas se utilizando do protocolo HTTP, juntamente com o conceito REST.

Camada de negócios - a camada de negócios contém a lógica de cada entidade do sistema, fazendo a mediação entre a camada de *web service* e a camada de persistência de dados.

Camada de persistência - camada responsável por representar os dados conforme eles estão no banco de dados e de persisti-los. Ela detém o conhecimento de características específicas do banco de dados que estão sendo utilizado.

Esta arquitetura utilizada, que promove a separação das responsabilidades, propicia que se tenha ganhos em manutenibilidade do código, legibilidade, tornando mais fácil fazer mudanças que são inerentes ao desenvolvimento de *software*. Se fosse preciso trocar o banco de dados a ser utilizado, bastaria fazer alterações na camada de persistência para suportá-lo.

Além disso, o fato de a aplicação do lado cliente ser totalmente desacoplada da aplicação servidor, faz com que seja possível escalar o sistema com pouco esforço.

A Figura 5.3 mostra o diagrama de classes da aplicação web, que como foi mencionado em seções anteriores, é uma aplicação que roda inteiramente no lado cliente, sem se utilizar de nenhuma tecnologia/linguagem do lado servidor para gerar HTML.

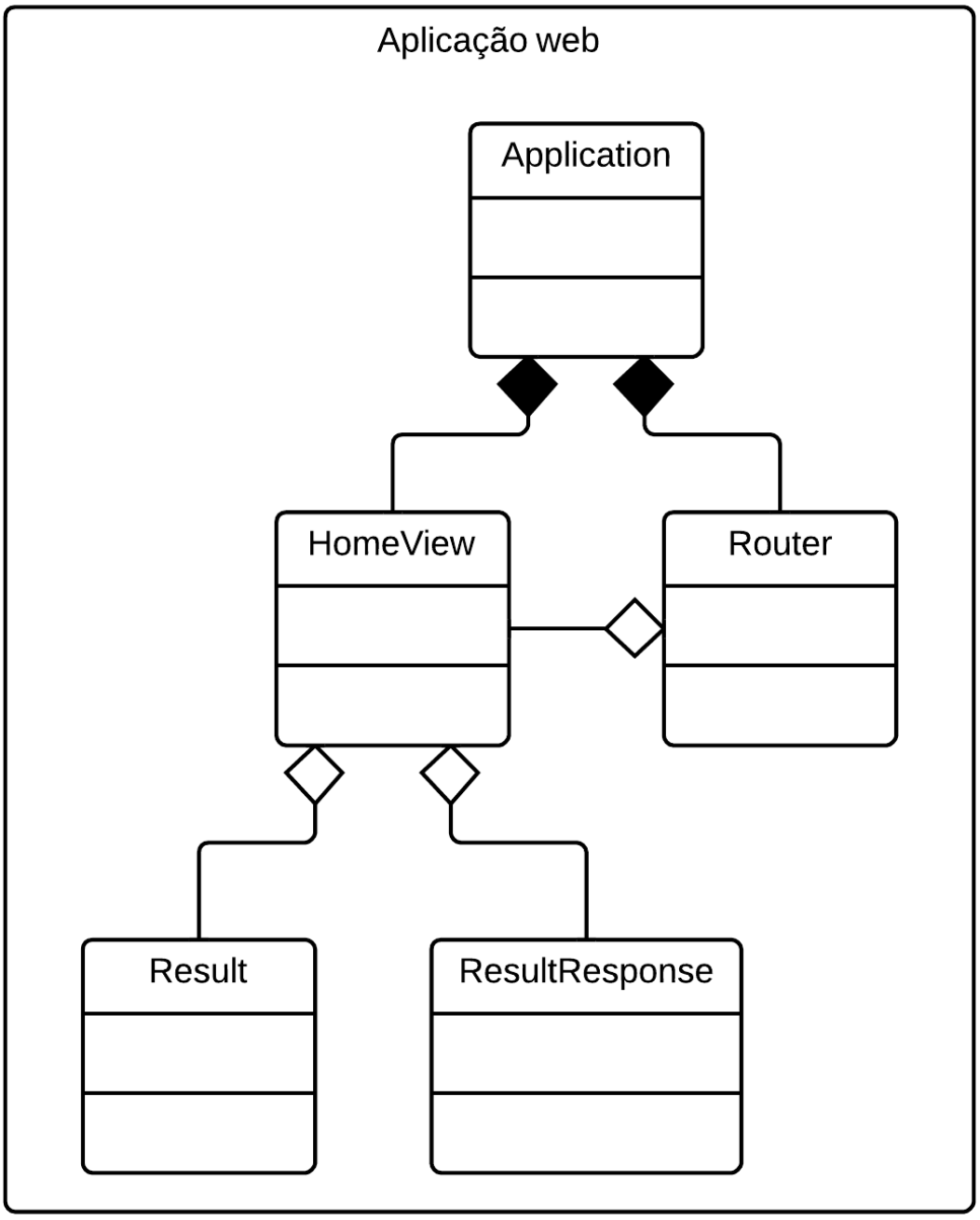

Figura 5.3: Diagrama de classes da aplicação web

As classes da aplicação cliente, refletem o uso do padrão arquitetural MVC. Elas se utilizam de classes da biblioteca Backbone.js (http://backbonejs.org/) herdando funcionalidades já implementadas nestas classes. Na Figura 5.3 estão expressas apenas as relações entre as classes desenvolvidas para a aplicação cliente.

Application - é uma classe que implementa a funcionalidade de VIEW e o padrão SINGLETON. Sendo a primeira a ser instanciada quando a aplicação cliente é iniciada. Sendo a primeira classe da aplicação a rodar, ela fica responsável por gerenciar as demais classes que serão instanciadas durante o período em que a aplicação estiver rodando. Como pode ser visto na Figura 5.3, há uma relação hierárquica entre ela e as classes HomeView e Router.

HomeView - esta é outra classe que implementa a funcionalidade de VIEW, tendo o papel de renderizar código de marcação (HTML) com base nas classes que fazem o papel de MODEL, e nas interações do usuário.

Router - esta classe faz o papel de CONTROLLER, capturando eventos de interação com usuário (assim como as classes de VIEW). A diferença é que ela captura interações do usuário com a barra de endereços. Ou seja, quando o usuário navega através da barra de endereços para um caminho interno a aplicação, esta ação é capturada e mapeada para uma função contida nela.

Result - classe que faz o papel de MODEL e é responsável por representar as entidades do sistema que tem reflexo na interface gráfica. Também tem a responsabilidade de fazer a comunicação com a aplicação servidor para se atualizar. No contexto da aplicação, ela representa um registro da lista de resultados.

ResultResponse - esta classe também é uma MODEL e ela representa a resposta da aplicação do lado servidor a uma consulta por uma página de um relatório. Ou seja, ela possui uma coleção de MODELs do tipo “Result”, além de outras informações como o número total de registros.

A aplicação do lado servidor é composta por três camadas, como já foi referido nas seções acima. A camada que expõe funcionalidades na forma API é a camada de *web service.* Na Figura 5.4 pode ser visto o diagrama de classes desta camada.

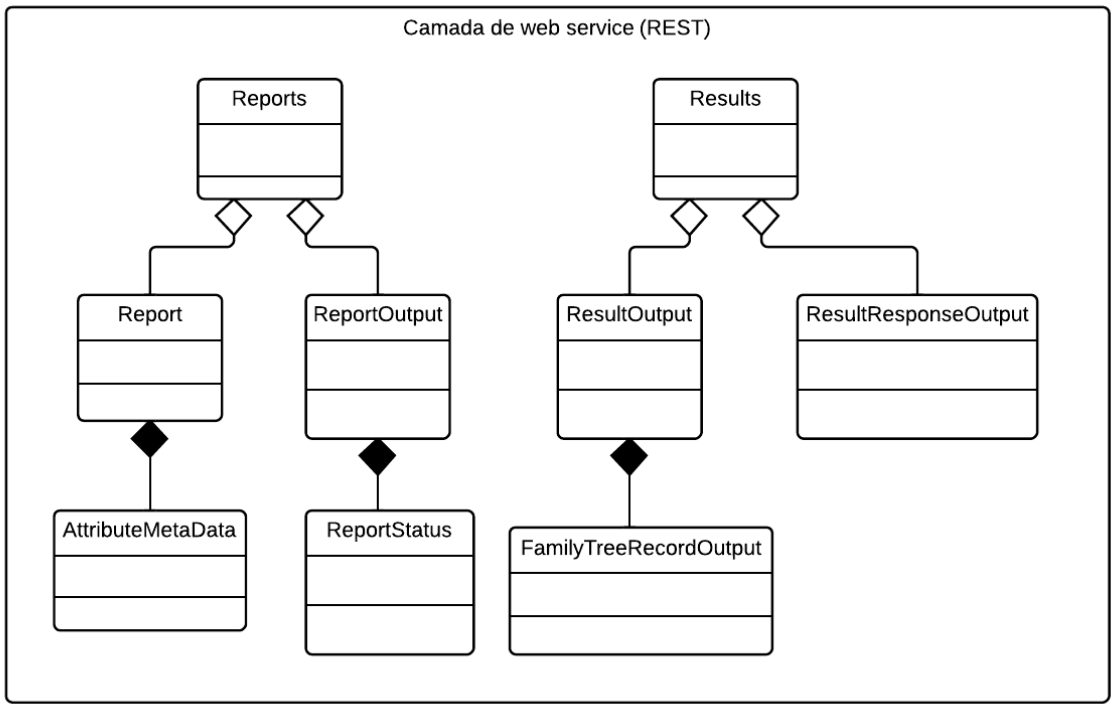

Figura 5.4: Diagrama de classes da camada de web service

As classes da camada de *web service*, seguem o conceito REST, provendo uma API que pode ser acessada por aplicações externas, utilizando-se o protocolo HTTP. Esta camada foi desenvolvida com a ajuda do PLAY FRAMEWORK (http://www.playframework.com/), que é uma biblioteca que tem o objetivo de ajudar a desenvolver aplicações *web*.

Reports - esta classe modela a API da entidade de relatórios, disponibilizando funcionalidades relacionadas a ela. Algumas das funcionalidades disponibilizadas são a criação e a consulta, que podem ser utilizadas através de requisições HTTP. Conforme prega o conceito REST, o URI tem a funcionalidade de identificar um recurso (neste caso o relatório) e para se informar a API a operação que se quer fazer sobre o recurso, deve-se passar o método HTTP. Por exemplo, o método POST é utilizado para criar um recurso e o método GET é utilizado para se fazer a consulta a um recurso.

Results - esta classe, assim como a classe "Reports" provê acesso a aplicações externas às suas funcionalidades. No caso, esta classe provê a acesso aos resultados, que representam um conjunto de registros que compõe um relatório. Eles podem ser selecionados por página ou por outro critério como palavras-chave.

Report - é a classe que modela os dados de entrada do relatório. Ela contém os dados que serão utilizadas pela aplicação para gerar o relatório.

ReportOutput - é a classe que modela os dados de saída do relatório. Ela contém os dados do relatório após o processamento da aplicação.

AttributeMetaData - esta classe modela uma evidência juntamente com o algoritmo escolhidos pelo usuário. Como mostrado na Figura 5.4 existe uma relação hierárquica entre esta classe e a classe “Report”. A classe “Report” tem uma lista de instâncias desta classe.

ReportStatus - esta classe modela o estado do relatório, que é consultado eventualmente com o intuito de verificar se ele está pronto.

ResultOutput - modela os dados de um registro que são mostrados quando o a API do resultado é consultada.

ResultResponseOutput - modela os dados de um conjunto de resultados que são mostrados quando a API de resultado é consultada.

FamilyTreeRecordOutput - modela os dados de um indivíduo da base GEDCOM com seus dados.

As classes da camada de negócios são responsáveis por cuidar da lógica de negócios das entidades da aplicação e também por fazer a mediação entre a camada de *web service* e a camada de persistência. A Figura 5.5 mostra o diagrama de classes desta camada.

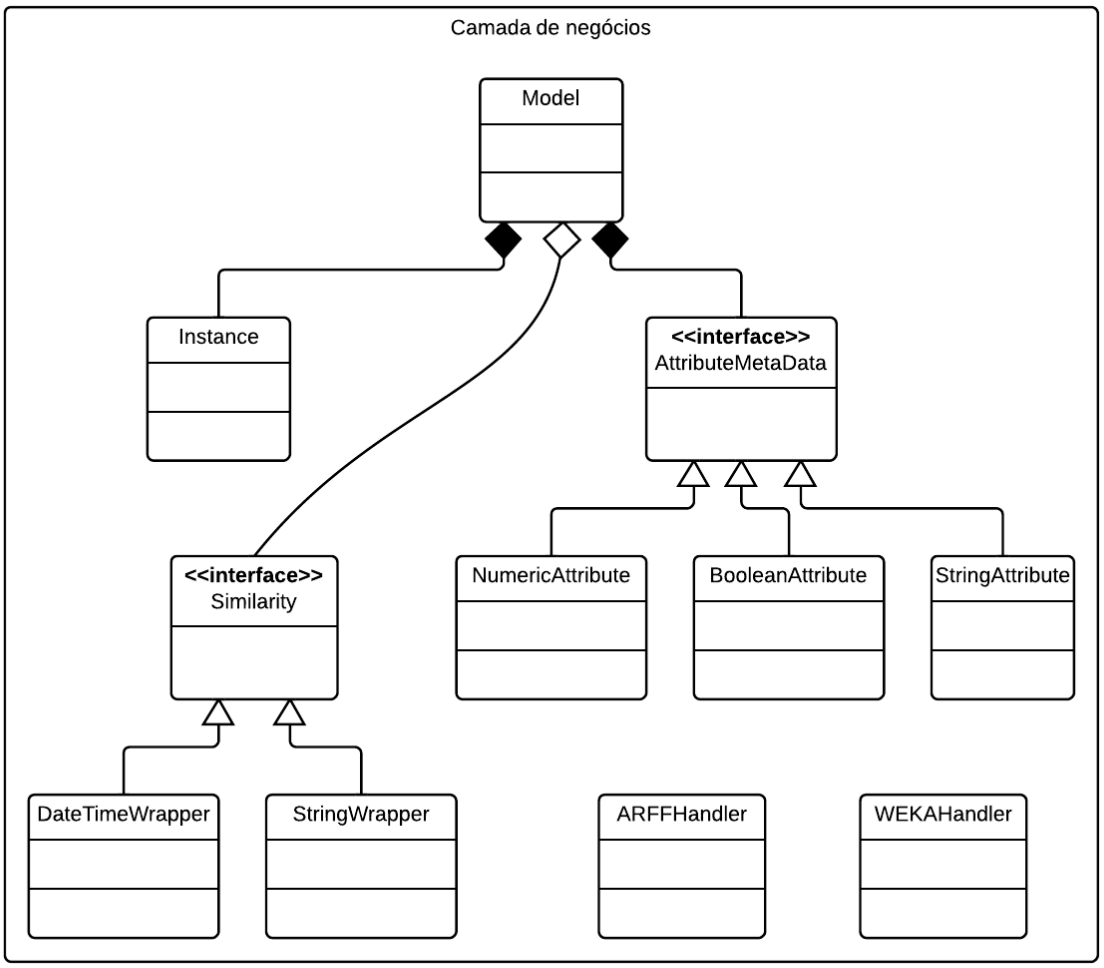

Figura 5.5: Diagrama de classes da camada de negócios

Model - esta classe contém toda a lógica de negócio de pré-processamento dos dados da base GEDCOM e do cálculo das similaridades das evidências.

Instance - esta classe modela um indivíduo da base de dados GEDCOM e contém a lógica de extração dos dados que serão utilizados para gerar os registros que serão utilizados para passar para a API do WEKA.

Similarity - esta é uma interface que define os protótipos para o cálculo de similaridade.

DateTimeWrapper - esta classe implementa os métodos definidos na interface "Similarity" para o cálculo de similaridade entre campos do tipo data.

StringWrapper - esta classe implementa os métodos definidos na interface "Similarity" para o cálculo de similaridade entre campos do tipo string.

AttributeMetaData - esta interface define abstratamente que campos uma evidência deve ter.

NumericAttribute - classe que modela evidências com valores numéricos.

BooleanAttribute - classe que modela evidências com valores booleanos.

StringAttribute - classe que modela evidências com valores do tipo string.

ARFFHandler - classe responsável por conter a lógica de geração de arquivos ARFF que podem ser utilizados pela API do WEKA.

GEDCOMHandler - classe responsável por conter a lógica de extração de dados de arquivos GEDCOM.

As classes da camada de persistência tem uma relação direta com o banco de dados escolhido e com as entidades modeladas no banco na forma de tabelas. A Figura 5.6 apresenta o diagrama de classes desta camada.

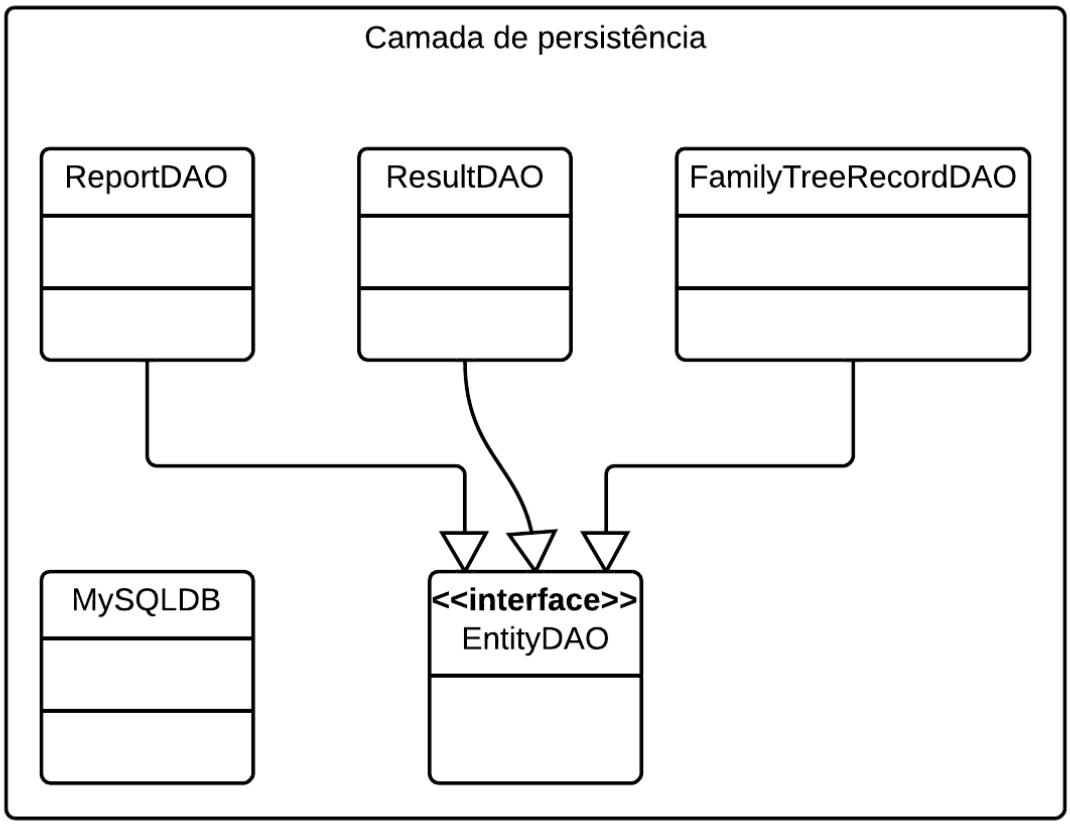

Figura 5.6: Diagramas de classe da camada de persistência

MySQLDB - esta classe abstrai a conexão com o banco de dados MySQL. Além disso ela oferece outras utilidades como a abstração das operações básicas e utilização de transação.

EntityDAO - esta interface define as operações que cada entidade deve ter, sendo elas: criação, atualização, busca e remoção.

ReportDAO - esta classe implementa as operações definidas na interface “EntityDAO” para o contexto da entidade de relatório, especificando os campos que esta entidade contém.

ResultDAO - esta classe implementa as operações definidas na interface “EntityDAO” para o contexto da entidade de resultado, especificando os campos que esta entidade contém.

FamilyTreeRecordDAO - esta classe implementa as operações definidas na interface “EntityDAO” para o contexto da entidade indivíduo (registro que vem da base GEDCOM), especificando os campos que esta entidade contém.

5.4 Diagramas de sequência

Diagramas de sequência são utilizados para ilustrar a interação entre os componentes de uma aplicação, evidenciando a sequência em que os eventos acontecem. Nesta seção serão mostrados um diagrama de sequência para cada caso de uso, afim de dar uma visão mais profunda de como a arquitetura da aplicação funciona. A Figura 5.7 mostra o diagrama de sequência do caso de uso “Solicita previsões para uma base de dados GEDCOM”.

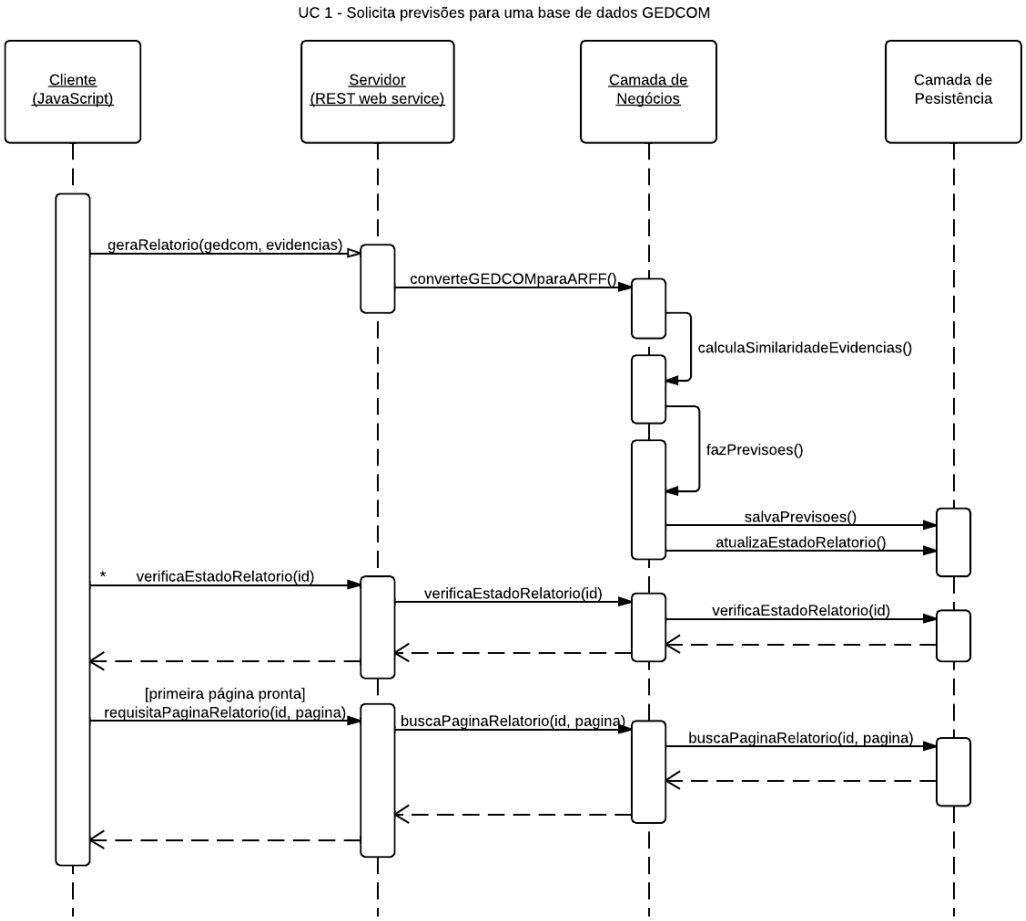

Figura 5.7: Diagrama de sequência do caso de uso “Solicita previsões para uma base de dados GEDCOM”

A Figura 5.7 detalha a interação que ocorre entre os componentes da aplicação. Após o usuário fazer a solicitação de previsões para uma base GEDCOM, a aplicação cliente dispara uma chamada assíncrona para a aplicação servidor solicitando que ser crie um relatório com estas previsões e passando como entradas uma base GEDCOM juntamente com a lista de evidências a serem consideradas. A camada de *web service* solicita por sua vez que a camada de negócios extraia os dados da base GEDCOM e calcule a similaridade entre os seus registros levando em consideração as evidências informadas. Com estes valores de similaridade calculados a camada de negócios salva iterativamente, utilizando a camada de persistência, os valores de similaridade entre cada registro (indivíduo) juntamente com a previsão fornecida pela API do WEKA.

Este processo é computacionalmente custoso, por isso foi escolhida a abordagem assíncrona combinada com uma consulta periódica ao estado do relatório. Se o relatório tem pelo menos uma página pronta ele é mostrado pela aplicação cliente.

Tendo pelo menos uma página pronta, a aplicação permite ao usuário navegar nas páginas (que vão sendo disponibilizadas conforme ficam prontas). Esta funcionalidade foi descrita no caso de uso “Navega nos resultados através de paginação” e é detalhada do ponto de vista das interações dos componentes da aplicação no diagrama de sequência da Figura 5.8.

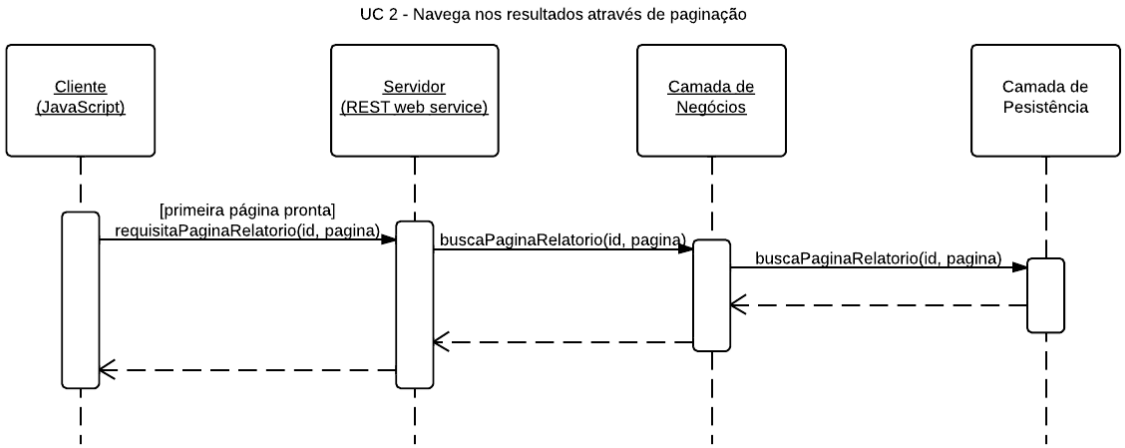

Figura 5.8: Diagrama de sequência do caso de uso “Navega nos resultados através de paginação”

A Figura 5.8 detalha o caso de uso “Navega nos resultados através de paginação”. As interações que ocorrem entre os componentes nesta funcionalidade, são bem simples. O que acontece de fato é que a chamada que se origina na aplicação cliente se propaga diretamente para a camada de persistência. Não sendo necessário nenhum processamento nas camadas intermediárias.

Outra funcionalidade que o sistema disponibiliza é a possibilidade de busca por palavra-chave. Ela também tem como requisito que pelo menos uma página do relatório já tenha sido gerada.

De forma semelhante à funcionalidade supra mencionada, o sistema também permite que o usuário faça busca de registros por valor da classe (ou resultado da previsão). Estas duas funcionalidades, descritas respectivamente pelos casos de uso “Busca resultados por palavra-chave” e “Busca resultados pelo valor da classe”, por questão de simplicidade e também por serem intimamente ligadas do ponto de vista da implementação, foram postas no mesmo diagrama de sequência que pode ser visto na Figura 5.9.

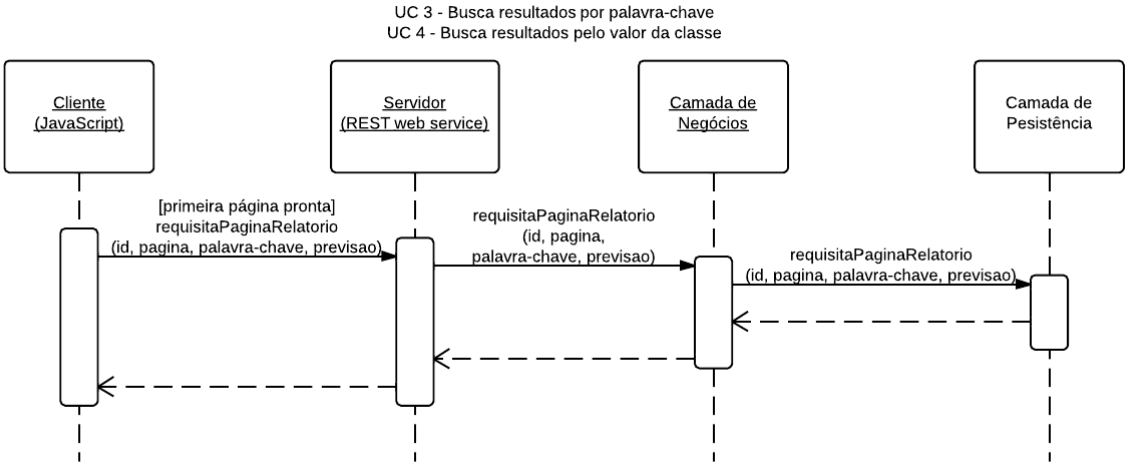

Figura 5.9: Diagrama de sequência dos casos de uso “Busca resultados por palavra-chave” e “Busca resultados pelo valor da classe”

Como pode ser visto na Figura 5.9, do ponto de vista da implementação, os casos de uso detalhados nela, se diferenciam apenas nos parâmetros de entrada do caso de uso da Figura 5.8. A palavra-chave e a previsão são acrescentados aos critérios de busca, mantendo-se a identificação do relatório assim como o número da página escolhida.

6 CONCLUSÃO

Este trabalho apresentou uma ferramenta para ajudar a identificar duplicatas em bases de dados genealógicas. Para tanto, fez-se uso de conceitos de aprendizado de máquina implementados pela biblioteca WEKA, do padrão de bases genealógicas GEDCOM e de diversas tecnologias que auxiliaram na geração dos componentes básicos que se interligaram para gerar o sistema final.

O sistema permite que se obtenha sugestões de registros de indivíduos de uma base genealógica que possam se referir a mesma pessoa. As entradas da aplicação são uma base de dados GEDCOM e uma lista de evidências que serão consideradas para comparar os registros. Com estas informações, a aplicação calcula a similaridade das evidências para cada registro e, com a ajuda da biblioteca WEKA, faz as previsões que são mostradas ao usuário na forma de um relatório paginado.

Para os próximos passos, poderia ser implementado na aplicação uma forma de eliminar a necessidade de se especificar as evidências, que poderiam ser inferidas utilizando alguma heurística em dados históricos da aplicação ou até mesmo em uma junção dos dois. Isto seria mais um passo na direção da automatização total do processo de busca e eliminação de duplicatas.

Outra possibilidade de complementação deste trabalho seria a integração da ferramenta a aplicações de genealogia existentes. Isto permitiria um uso mais adequado da funcionalidade, já que o usuário não precisaria sair do contexto da aplicação de genealogia de sua preferência para utilizar da funcionalidade proposta pela ferramenta do presente trabalho.

REFERÊNCIAS

WEKA. Disponível em: http://weka.sourceforge.net/doc.stable/. Acesso em: jun. 2014.

FOWLER, Martin. **UML Distilled: A Brief Guide to the Standard Object Modeling Language**. 3. ed. Boston: Addison-Wesley Professional, 2005.

WITTEN, Ian H.; FRANK, Eibe; HALL, Mark A. **Data Mining: Practical Machine Learning Tools and Techniques**. 3. ed. San Francisco: Morgan Kaufmann, 2011. 664 p. (The Morgan Kaufmann Series in Data Management Systems).

OSMANI, Addy. Developing **Backbone.js Applications: Building Better JavaScript Applications**. 1. ed. Sebastopol: O'Reilly Media, 2013. 374p.

HALL, Mark; FRANK, Eibe; HOLMES, Geoffrey; PFAHRINGER, Bernhard; REUTEMANN, Peter; WITTEN, Ian H.; The WEKA Data Mining Software: An Update. **SIGKDD Explorations**, v. 11, n 1, p.10-18, jul. 2009.