1 ARCHITECTURE

1.1 OVERALL SYSTEM

1.2 MVC

1.3 WEB INTERFACE

1.4 WEB SERVICE

2 INTRODUCTION

This project contains some backgrounds which are not part of computer science. Here introduces adequate information in order to help getting start.

2.1 MOTIVATION

The first civil registration in Ireland was performed on 1864 [19]. Before that time census meterials were mostly lost or incomplete. So genealogical researches need to rely on parish records and also some 'census substitute' documents, such as land ownership and tenancy records[1].

However, for these documents, each of them usually does not contain enough information to indentify individuals. Some of them contains name and address, whereas others might contain only name. In order to fulfil missing information of one individual that scattered among many documents, *Record linkage* is one method to do so.

"Record linkage is used in historical research, social studies, marketing, administration and government as well as in genealogy"
– Winstanley [22] section 2.2

Record linkage uses a person's name as a basis to link that person's information between many documents. Together with other coherant attributes to ensure the link is correct, a more complete information about that person can be obtained.

In addition, this is not only just for one person. We can assume the relationship of the person to others that might be close to, and apply the information to those people as well. For example, if we know that there is a record that is believed to consist of people from the same area in each page [12] (but no area or address is mentioned, or some is missing in the page). And we can find one or more person's addresses in that page by using record linkage. We might be able to apply those addresses to all people in that page as well.

Apparently linking or matching person's name is important in the process. Unfortunately, in the 19th century, in Ireland, there was no standard of the spelling of names, handwriting could be difficult to read and contractions or abbreviations were often used. Many people were not literate, so they asked literate people to write their names. This way even names with the same pronounciation and for the same individual could be written in many different ways, depending on who wrote them.

1 [22] section 1.1

In addition to the various ways of spelling one's name, people from this time also often use Irish names which equivalent to modern names, for example, Irish version of *Smith* could be *Gowan*. There are also some Irish prefixes like *O'*, *M'*, *Mac*, etc. When combined together this would result in *O'Gowan* or *M'Gowen*, and so on.

An example list of possible equivalent Irish names of *Smith* could be as follow.

> Smith, Smyth, Smythe, Smeeth, Going, Gowing, Maizurn, McGhoon, MaGough, M'Ghoon, MacGivney, MacGivena, M'Givena, MacGhoon, M'Evinie, McGivney, MacEvinie, McGivena, M'Givney, McEvinie, MacAvinue, M'Avinue, McAvinue, McCona, MaGowen, MaGowan, MaGovern, MaGeown, McGowan, McGoween, McGown, M'Cona, MeCowan, MeGowan, MacGown, MacGoween, MacGowan, MacCona, M'Gowan, M'Gowen, M'Gown, Ogowan, O'Gowan, Gowen, Gowan, Gow, Goan

At present time, when historical researchers try to trace people back using historical records, they would encounter this problem of name variations.

Various solutions have been created to find matching different names that refer to the same person. However, for our extent knowledge, there is yet no public system which encodes those solutions together and provides a service of name matching. This project is to create one system to archieve this.

2.2 RESEARCH QUESTIONS

From the motivation, we address our research questions as follow.

1. Can we provide a web service to match names, where matching can be a complicated process because of the way people record their names.
2. Can the web service act as a platform system for general names or words matching system so that it can be extended to other languages as well.

The first question derives directly from the motivation. The second question is an enhancement for the system. It can be designed as a more general purpose matching system rather than just specified only for Irish names. Therefore it should be extensible for any further matching methods to be developed in the future.

In addition to the web service, web interface is to be introduced as well for the purpose of user friendly usage, individual usage, and demonstration.

2.3 OBJECTIVE AND AIMS

The objective of this project is to provide a web service that encodes several of matching methods and produces matching results between two list of names.

The project aims to be a part of a bigger system, such as genealogy research. These client systems, at some point, they might need a service of a name matching on demand, so then they can use this web service, providing their lists of name, methods be be used, and threshold as inputs, and get matching results for their further usage.

We would start by focusing on Irish *surname* first. For any further kind of names we would leave it for future works.

2.4 REPORT STRUCTURE

This report is separated into four parts, The Background, The Solution, and Appendix.

THE BACKGROUND: Current part, states about background of this project. Introduces the initial problem, also some historical situations and terms which are not resident to computer science. Also related works that are involved in the project.

THE SOLUTION: The implementation to solve the problem. Details about algorithm, tools, language, frameworks, etc. which being used in the project.

THE FOREWARD: Evaluation of its performance, conclusion of the outcome of the project. encountered problems, and future works for extending and improvements.

APPENDIX: The 'user manual' of the project. Presents technical aspects, for example, how to use the web service in real world situation, or how to create an environment to host this project.

3 RELATED WORK

From research questions on section 2.2, there are three aforementioned terms that will be core research fields of this project. These fields are *name matching*, *web service*, and *extensible platform*.

3.1 NAME MATCHING

There are many methods for matching names. This project encodes various of them at the starting state.

3.1.1 *Edit distance*

> Edit distance is a way of quantifying how dissimilar two strings (e.g., words) are to one another by counting the minimum number of operations required to transform one string into the other. – Edit distance, Wikipedia [20]

An direct string operation way of comparing two string could work with name matching too. One of the edit distance variant, *Levenshtein distance* [21] is chosen to be implemented in this project.

3.1.2 *Soundex*

Soundex [1] encodes a name (or any string) into a 4 character code which represents an essence of its sound as pronounced in English. The idea is to encode letters with similar sound into the same group, and ignore vowels (unless it is the first letter). For example, *Smith* is translated to `S530`, and *Simon* is translated to `S550`.

Irish Soundex[1] is a modified version of Soundex, aims to improve capability of a traditional one upon Irish surnames. By applying rules accroding to the langauage characteristics and make some adjustment to distinguish names properly.

Both Soundex variants are also implemented in the project.

1 [22] Appendix 3.

3.1.3 *Lookup Table*

In 1901, Robert Edwin Matheson, the assistant registrar-general in Dublin, developed a name classification system [8] for an aid of register indexing and searching. He used a report on surnames in Ireland extracted from civil registers [7] in 1894 as a base of his system[2].

He gathered information from registry offices, focusing on people or members of close families. When these people made official register records with the office, they might use different variant of their surnames. For example, Mr. Green can be registered as dead by his son using the name Huneen.

With these information, Matheson classified the surnames in Ireland into 2091 groups. For example, group 753 consists of these names.

> Green, Greenan, greenaway, greene, grene, Guerin, Houneen, Huneen, MacAlasher, MacAlesher, MacGlashan, MacGlashin, MacIllesher, M'Alasher, M'Alesher, McAlasher, McAlesher, McGlashan, McGlashin, McIllesher, M'Glashan, M'Glashin, M'Illesher, Oonin.

This classification also includes multiple mapping between names. One name can belong to one or more group. For example, *Green* belongs to groups 753, 754, 768, and 1350.

By using this classification information, we can construct a lookup table for Irish names by having names in the same group hold the same reference number.

3.2 WEB SERVICE

One convenient way to bring this service to public is to create a *web service*. A web service is a tool or function that can be accessed by other programs over the web (via http) [11]. A result from web service is designed to be used by computer programs rather than humans.

There are many ways to implement web services. Two famous ones are *Simple Object Access Protocol (SOAP)* and *Representational State Transfer (REST)*. Both has their own advantages [16]. We decided to implement our service using REST due to its simplicity and scalability [9][5].

At this initial state, data resulting from our web service is in `JSON` [6] format. Since it is widely used in web development and becoming more and more popular [4]. However, our service can be extended into any other format easily as well, such as traditional `XML`.

2 [22] section 2.3.

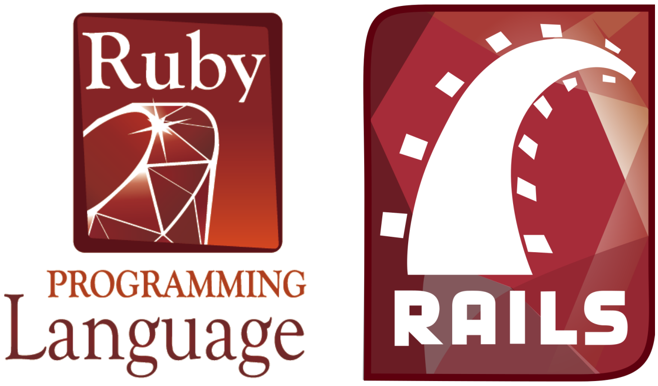

Figure 1: Ruby programming language (left) and Ruby on Rails framework (right).

3.3 EXTENSIBLE FRAMEWORK

"Ruby is designed to make programmers happy."
– Venners [18]

Our system is implemented in *Ruby* [13] programming language. Ruby is a well-balanced language, it can be used as an traditional object-oriented language [17] and also capable of performing functional programming [14], thus making it very flexible and versatile.

The system sits on top of *Ruby on Rails* (or *Rails*, in short) [15] framework. Rails is a mature and stable framework that has been in web development for decades [3]. So it has a great support and a large community bebind. A great choice for building a sustainable system.

Rails is capable of both web service and web interface. By sharing the same algorithm we could provide a service for both programs (targeted by web service) and humans (targeted by web interface).

4 NAME MATCHING ALGORITHMS

This chapter describes the details how the project is implemented. Note that algorithms and codes listed here are written in Ruby programming language, which is the main language of the project.

We will start off by detailing bundled matching algorithms here. Each matching algorithm calculates the *similarity score* between two strings.

The score is ranging between 0.0 to 1.0, where 0.0 means two strings are completely different and 1.0 means both are extactly matched.

Also note that string inputs here is in all in the uppercase format, in order to prevent letter-case difference.

4.1 LEVENSHTEIN DISTANCE

This algorithm measures the difference between two strings. It tells the minimum number of opearations needed to change string to another. These opeations are insertions, deletions, or substitutions. Consider these following examples.

- SM**I**TH → SM**Y**TH
  the minimum operation to change is 1, which is to substitute *I* to *Y*, therefore Levenshtein distance for these two strings is 1.
- GOWAN → **MC**GOWAN
  2 insertions of *M* and *C* is required.
- SM**I**TH**E** → SM**Y**TH
  1 deletion of *e* and 1 substitution of *i* to *y* are required.

The implementation used in the project is done by Battley [2][1]. Once the distance is calculated, it will be compared to the length of the longer string between the two (or if they are the same length, use that length).

For example, Levenshtein distance between *SMYTH* and *SMITHE* is 2, compare 2 to length of the longer string, *SMITHE*, which is 6. So the *similarity score* of these two strings are $6 - (2/6) = 0.667$.

The code of this algorithm is as in listing 1, note that `name` and `@base_name.name` are two strings to be matched.

1 levenshtein.rb

GROUP	LETTERS
1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M, N
6	R
-	A, E, I, O, U, H, W, Y

Table 1: Soundex letter group.

```
def cal_score
  @value = Text::Levenshtein.distance(@name, @base_name.name)
  size = [@name.size, @base_name.name.size].max
  @score = ((size - @value).to_f / size)
end
```

Listing 1: Levenshtein distance implementation.

4.2 SOUNDEX

Soundex encodes a string into a 4 character code representing an essence of its sound as pronounced in English. It operates in the following steps.

1. Take the first letter of a string.
2. Encode each remaining letters into a group following table 1. Discards A, E, I, O, U, H, W, and Y
3. Remove two adjacent same characters.
4. If a group of a first letter is the same as the second letter, remove the second letter.
5. Trim or pad with zeros as necessary, making the result 4 characters long.

Let us follow these steps by step, consider we are going to encode the string *PFISTTER*.

1. Take first letter of *PFISTTER*.
   PFISTTER → P
2. Encode remaining letter *FISTTER*.
   PFISTTER → P1-233-6 → P12336

3. Remove two adjacent same characters.
   PFISTTER → P12336 → P1236
4. P is also in group 1, so remove the second 1 letter.
   PFISTTER → P1236 → P236
5. P236 is 4 characters long, so no need to be trimmed or padded.
   PFISTTER → P1236 → P236

Therefore, soundex of *PFISTTER* is `P236`.

The implementation of soundex (listing 2) in this project is adapted from Winstanley's *Irish soundex* implementated in Visual Basic[2]. The code is commented following the same aforementioned steps.

```
def self.soundex(name)
  # Take the first letter of a string.
  result = name.first

  # Encode remaining letters
  name[1..name.length].split('').each do |n|
    result = result + category(n).to_s
  end

  # Remove two adjacent same characters
  result.gsub!(/([0-9])\1+/, '\1')

  # If category of 1st letter equals 2nd character, remove 2nd
      character
  if result.size >= 2 && category(result[0]).to_s == result[1]
    result.slice!(1)
  end

  # Trim or pad with zeros as necessary
  result = if result.size == 4
             result
           elsif result.size > 4
             result[0..3]
           else
             result.ljust(4, '0')
           end
end
```

Listing 2: Soundex implementation.

The `category` function implements soundex grouping table (table 1) as in listing 3.

2 [22] Appendix 3.

```ruby
def self.category(c)
  if c.match(/[AEIOUHWY]/).present?
    ""
  elsif c.match(/[BPFV]/).present?
    1
  elsif c.match(/[CSKGJQXZ]/).present?
    2
  elsif c.match(/[DT]/).present?
    3
  elsif c.match(/[L]/).present?
    4
  elsif c.match(/[MN]/).present?
    5
  elsif c.match(/[R]/).present?
    6
  else
    ""
  end
end
```

Listing 3: Soundex grouping table implementation.

Now that we encode two strings to be matched in soundexes. We then calculate the *similarity score* of these two soundexes using these steps.

- Compare first characters of each soundex, if they are different, *similarity score* is 0, otherwise move to next step.
- Compare the rest 3 digits by using Levenshtein distance (section 4.1) to calculate the distance between them.

For example, *similarity score* between *SMITH* and *SPEED*, which soundexes are `S530` and `S130` respectively, is 0.75 (1 substitution from *5* to *1*, so 1 difference of length 4).

The code of this soundex *similarity score* is as in listing 4.

```
def soundex_distance_score(s1, s2)
  if s1.first != s2.first
    0 # Different category, so they suppose to be completely
        different
  else
    (s1.size - Text::Levenshtein.distance(s1, s2).to_f ) / s1.size
  end
end
```

Listing 4: Soundex similarity score implementation.

4.3 IRISH SOUNDEX

Irish soundex is another variant of traditional soundex. It determines characteristics of Irish names and normalised them to modern names. This algorithm also follows Winstanley's *Irish soundex*[3].

Irish names might contain some prefix, e.g. *Mc* or *O*, which are obstructive to soundex result. These prefixes are to be discarded. Moreover, there is no initial soft *C* in Irish names, instead *K* is used. So the first letter *C* is changed to *K*. It is implemented as in listing 5.

3 [22] Appendix 3.

```
def self.soundex(name)
  # Change initial ST. to SAINT
  name = name.match(/^ST\./).present? ? "SAINT
      #{name[3..name.length]}" : name

  # Discard Irish prefixes
  name = if name.match(/^O /).present?
           name[1..name.length].gsub(' ', '')
         elsif name.match(/^O'/).present?
           name[2..name.length].gsub(' ', '')
         elsif name.match(/^MC/).present?
           name[2..name.length].gsub(' ', '')
         elsif name.match(/^M'/).present?
           name[2..name.length].gsub(' ', '')
         elsif name.match(/^MAC/).present? && name != 'MAC'
           name[3..name.length].gsub(' ', '')
         else
           name
         end

  # Change initial C to K
  name = name.strip.gsub(/^C/, 'K')

  # Call to traditional soundex.
  return {
    :label => name,
    :soundex => Soundex.soundex(name)
  }
end
```

Listing 5: Irish soundex implementation.

Irish soundex algorithm in this project calls traditional soundex described in section 4.2 to minimise repeated code. It also calculates *similarity score* the same way soundex does, as in listing 4.

4.4 LOOKUP TABLE

Lookup table is constructed from Robert Edwin Matheson's classification of Irish names. All classification information is stored in a Database, using PostgreSQL, which is a powerful, open source object-relational database system [10].

Matheson classified the surnames in Ireland into 2091 groups. Each group has one or more names, and on the other hand, each name belongs to one or more group.

In this section, we will consider the names as strings input. So the term *string* will be used in consistent with previous sections.

For example, considering the string *ACHESON*, this string belongs to two groups, 4 and 42. So *ACHESON* will have 2 records in the database. One with reference to group 4 and another with reference to group 42.

Next, let us consider group 4 and 42.

> 4 → ACHESON, ACHISON, AITCHISON, ATCHESON, ATCHIESON, ATCHISON, ATKINSON
> 42 → ACHESON, ARKESON, ATKINS, ATKINSON

By combining two groups together, these are all possible strings that match *ACHESON* according to Matheson's classification.

Now is the process to match two strings using Lookup table, suppose two strings are *ACHESON* and *ATKINS*.

1. Find references of *ACHESON*. We get references for group 4 and 42. Note the use of `pluck` method to select reference attribute (`ref`) here[4].

```
LookupTableRecord.where(:name => 'ACHESON').pluck(:ref)
=> [4, 42]
```

2. Find reference to matching string *ATKINS* and also specify the reference groups from step 1. If there is no match `where` method will return empty array. `present?` method is used to check the result if it is not empty[5].

```
LookupTableRecord.where(:ref => [4, 42], :name =>
    'ATKINS').present?
=> true
```

   By specifying both matching string and references group, we can ensure the matching name is also in the one of the same reference group of the base name. In this case, we conclude that there is a match between *ACHESON* and *ATKINS* via group 4 or 42 (or more specifically, 42, because *ATKINS* belongs to group 41 and 42).

---

4 Use pluck as a shortcut to select one or more attributes without loading a bunch of records just to grab the attributes you want. http://api.rubyonrails.org/classes/ActiveRecord/Calculations.html#method-i-pluck

5 http://api.rubyonrails.org/classes/Object.html#method-i-present-3F

If a reference is found on both steps, *similarity score* for lookup table of the two strings is 1.0. If the system fails to find any reference on any step, consider a no match and the score is 0.0.

By following these steps, the implementation of lookup table is as in lisiting 6.

```
def cal_score
  base = LookupTableRecord.where(:name => @base_name.name)

  @score = if base.nil? # Could not find reference for base name,
      no matches
         0
        else
          # Find any reference that has 1) same name 2) same
              reference
          base = base.map(&:ref)
          refs = LookupTableRecord.where(:ref => base, :name =>
              @name)

          if refs.present?
            @label = (base & refs.map(&:ref)).join(', ')
            @value = "Matched"
            1
          else # Could not find reference for matching name, no
              matches
            0
          end
        end
end
```

Listing 6: Lookup table implementation.

BIBLIOGRAPHY

[1] National Archives and Records Administration. *The Soundex Indexing System*. May 2007. URL http://www.archives.gov/research/census/soundex.html. accessed May 4th, 2015.

[2] Paul Battley. *Text, Collection of text algorithms*. 2015. URL https://github.com/threedaymonk/text. accessed May 9th, 2015.

[3] Kresimir Bojcic. *What are the Benefits of Ruby on Rails? After Two Decades of Programming, I Use Rails*. 2013. URL http://www.toptal.com/ruby-on-rails/after-two-decades-of-programming-i-use-rails. accessed May 6th, 2015.

[4] Hussain Fakhruddin. *JSON or XML – Which Data Format Is Better For Developers?* April 2015. URL http://teks.co.in/site/blog/json-or-xml-which-data-format-is-better-for-developers/. accessed May 5th, 2015.

[5] Steve Francia. *REST Vs SOAP, The Difference Between Soap And Rest*. Jan 2010. URL http://spf13.com/post/soap-vs-rest. accessed May 5th, 2015.

[6] Introducing JSON. *json.org*. 2015. URL http://www.json.org. accessed May 5th, 2015.

[7] Robert Edwin Matheson. *Special report on surnames in Ireland*. 1894. URL https://archive.org/details/cu31924029805540. accessed May 4th, 2015.

[8] Robert Edwin Matheson. *Varieties and synonymes of surnames and christian names in Ireland*. 1901. URL https://archive.org/details/varietiessynony00math. accessed May 4th, 2015.

[9] John Mueller. *Understanding SOAP and REST Basics*. Jan 2013. URL http://blog.smartbear.com/apis/understanding-soap-and-rest-basics/. accessed May 5th, 2015.

[10] postgresql.org. *PostgreSQL*. 2015. URL http://www.postgresql.org. accessed May 11th, 2015.

[11] Vincent Ramdhanie. *What is a 'web service' in plain English?* Oct 2008. URL http://stackoverflow.com/a/226159/459794. accessed May 5th, 2015.

[12] Christopher Ridgway. *The Morpeth Roll: Ireland identified in 1841*. 2013.

[13] ruby lang.org. *Ruby, a programmer's best friend*. 2015. URL https://www.ruby-lang.org/en/. accessed May 6th, 2015.

[14] ruby lang.org. *About Ruby*. 2015. URL https://www.ruby-lang.org/en/about/. accessed May 6th, 2015.

[15] rubyonrails.org. *Web development that doesn't hurt*. 2015. URL http://rubyonrails.org. accessed May 6th, 2015.

[16] The Java EE 6 Tutorial. *Types of Web Services*. 2013. URL http://docs.oracle.com/javaee/6/tutorial/doc/giqsx.html. accessed May 5th, 2015.

[17] tutorialspoint.com. *Object Oriented Ruby*. 2015. URL http://www.tutorialspoint.com/ruby/ruby_object_oriented.htm. accessed May 6th, 2015.

[18] Bill Venners. *The Philosophy of Ruby*. Sep 2003. URL http://www.artima.com/intv/rubyP.html. accessed May 6th, 2015.

[19] welfare.ie. *History of Registration in Ireland*. 2015. URL https://www.welfare.ie/en/downloads/GRO-History.pdf. accessed May 8th, 2015.

[20] Wikipedia. *Edit distance*. 2015. URL http://en.wikipedia.org/wiki/Edit_distance. accessed May 4th, 2015.

[21] Wikipedia. *Levenshtein distance*. 2015. URL http://en.wikipedia.org/wiki/Levenshtein_distance. accessed May 4th, 2015.

[22] Adam Winstanley. *Identifying People on the Morpeth Roll*. July 2014. Postgraduate Diploma in Genealogical, Palaeographic & Heraldic Studies 2013-14.