Report on the Project in Informatics of the 1st cycle degree in Informatics Engineering

Management and Retrieval of Digital Photographs

Alexandra de Carvalho
Anthony Pereira
Mariana Santos
Pedro Iglésias
Wei Ye

UNIVERSIDADE DE AVEIRO
DEPARTAMENTO DE ELETRÓNICA, TELECOMUNICAÇÕES E INFORMÁTICA

Authors

Alexandra de Carvalho
Anthony Pereira
Mariana Santos
Pedro Iglésias
Wei Ye

Place for Project

Departamento de Eletrónica Telecomunicações e Informática of Universidade de Aveiro
Aveiro, Portugal

Supervisors

Professor António Neves, Assistant Teacher
Ricardo Ribeiro, Grantee
Universidade de Aveiro

Acknowledgements

This work would never be made without continuous help from our advisers, and that's why we want to thank them for giving endless advice and helping us daily with our questions and insecurities. This project could not exist without them! And many thanks to our families, which always gave us the strength to keep going and never give up on our dreams.

Abstract

Imageable is a system that processes images and, through modules for extracting objects, places, text or even facial recognition, it associates each photograph with a set of tags that can be customizable by the user, tested to support a large amount of data. It supports two main features: the search for images by text associated with the data taken by the previous modules, or even the search for similar images.

Using Neo4j as a database to resort to the use of relationships between entities, and Elastic Search to improve search time, Imageable is a high-performance system with an intuitive and easy-to-use interface.

This interface consists of a search page, a folder submission page, a facial recognition page, and even a dashboard with multiple statistical indicators related to the data extracted from the submitted images.

Given the growing consumption of digital content and the accumulation of several photographs over the years, Imageable revolutionizes by being an image analysis and retrieval tool used locally on a device, having no data privacy issues, in contrast to its direct competitors.

Keywords

Object extraction, Text extraction, Places detection, Facial recognition, Search engine, Image processing

Contents

1 Introduction **1**

1.1 Background 1

1.2 Objectives 1

1.3 Personas/Scenarios/User stories 1

1.4 State of The Art / Related Work 7

1.5 Conceptual Modeling (architecture diagram) 11

2 Procedure **12**

2.1 Database and Domain Model 12

2.2 Elastic Search 13

2.3 Modules 15

2.4 Simple File System Manager 20

2.5 Natural Language Processing 23

3 Results and Discussion **28**

3.1 Presentation of the Interface 28

3.2 Quality Assurance 29

3.3 SonarQube 29

3.4 New Functionalities 30

3.5 Deployment 37

4 Conclusion **38**

References **39**

1 Introduction

This project consisted of the development of a web application that allows managing and recovering images using data extracted from them. It was developed in partnership with the coordinator Professor António Neves and the co-coordinator Ricardo Ribeiro.

1.1 Background

In a society increasingly focused on the digital, to which the fast technological development of smartphones is not indifferent, allowing more and more photos to be taken with better quality and less effort, it is essential to manage the entire amount of digital data we keep in our daily lives - through a system for searching and retrieving images from a large amount.

1.2 Objectives

The goal of this project was to develop a system capable of searching and retrieving images locally from the device, which might be considered a large database. We can highlight some features, such as searching for images similar to a given one – content-based image retrieval –, and by content extracted from them – concept-based image retrieval – such as people, objects or places. We also aimed to have a user-friendly application, ensuring that anyone can use it regardless of age or technological knowledge.

1.3 Personas/Scenarios/User stories

Figure 1.1 shows the use case diagram for our product. As we can see, we only planned for one user because our system is very general and can be used by anyone equally. Because it is an offline software, it does not need any type of account management. In Figure 1.1, we can see the tasks our users are able to do in the system, each of them leading to the development of a specific persona and user story.

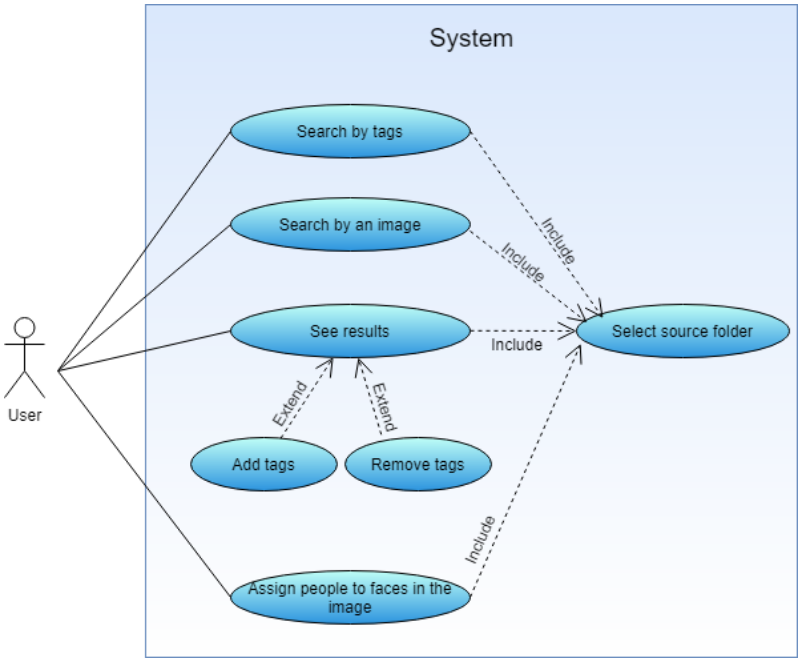

Figure 1.1: Imageable use case diagram

Alexandra Carvalho In Figure 1.2 we can discover more about the persona that we named Alexandra Carvalho.

Alexandra is a 21-year-old girl. She decides to enter Veterinary Medicine, where she can do her best to protect all animals. In parallel, she starts volunteering at the animal association "Senhores Bichinhos" in Vila Nova de Gaia, where she takes care of the cats that are in the shelter and holds adoption sessions. In charge of preparing leaflets to advertise the kittens that are in the shelter, to be published on the social media Facebook and Instagram, Alexandra intends to use the application in order to filter her photographs to obtain those of a specific cat.Alexandra intends to use an easy-to-use application that allows her to search for the name of a specific cat from the refuge.

User story: As Alexandra, I want to search for the name of a specific cat from the refuge so that I can make leaflets and share them on social media.

António Nogueira In Figure 1.3 we can discover more about the persona that we named António Nogueira.

An 84-year-old man, António has worked hard all his life. He lives in Porto, his homeland, and is looking forward to the dinner he will offer to his youngest

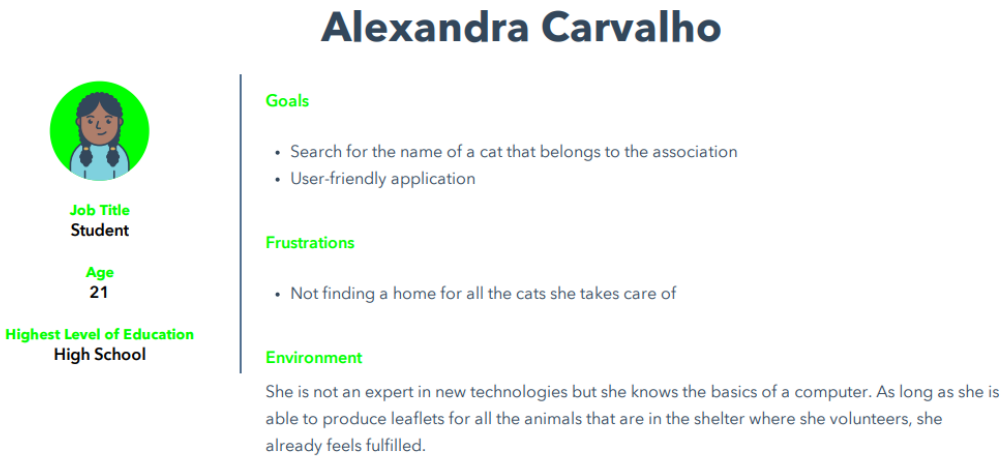

Figure 1.2: Persona 1 - Alexandra Carvalho

grandson, Tomás, in celebration of his admission to college. António prepared a surprise for his son António and grandson Tomás: he was taking pictures of the old albums he had kept in the attic of his house, picking up mold, to ensure that these memories are never lost for all eternity. Thus, António plans to show his grandson photographs of his father, at the same young age – so that everyone remembers that life is an endless cycle.

António wishes to be able to filter the photographs without much difficulty, as he does not have great technological literacy, through a simple interface that allows him to have constant feedback, namely through the appearance of messages with a font size that does not tire his eyes. He should also be able, using face recognition technology and when taking a picture of his son António, to collect all the photos of his son that are in the folder that is being analysed.

User story: As António, I want to be able to filter the photographs without much difficulty, recurring to face recognition, so that I can make a special surprise for my grandson.

Filipe Nogueira In Figure 1.4 we can discover more about the persona that we named Filipe Nogueira.

Filipe is a 25-year-old young man. He took a Representation course, where he

António Nogueira

Job Title
Retired

Age
84

Highest Level of Education
Fourth class

Goals

- Find photos of his grandson when he was younger through facial recognition
- Use of a user-friendly application

Frustrations

- Application not adapted to his vision problems
- Application with a high learning curve
- Slow processing times

Environment

Having only attended school for 4 years and worked all his life in the construction industry, he never showed curiosity in the area of technologies. He has a smartphone, but he can hardly do anything other than taking photos and copy them to his computer. He only uses his PC to play Solitaire and Candy Crush, because he doesn't have much patience to learn all the features that a computer can offer.

Figure 1.3: Persona 2 - António Nogueira

quickly caught the attention of national producers. He was part of the soap opera "A Casa Das Sete Donzelas", in which, being one of the protagonists, he gained national fame. Filipe is in a relationship with Rita, a 26-year-old woman with whom he made a romantic pair in the soap opera in which they both became famous. They married last year in Taipu de Fora, Bahia – an unforgettable moment, according to them. For the celebration of their first anniversary, they intend to make a compilation of the best moments and backstage of their wedding to be published on the social network Instagram, in the account shared by the couple and in which they are supported by a large legion of fans. That's why, Filipe and Rita intend to filter the photographs they have by location, thus obtaining all the incredible moments that passed on that beach in Brazil.

Filipe and Rita intend to use an easy-to-use application, thus being very intuitive, so that they can filter by a specific location.

User story: As Filipe, I want to filter my photos by location, so that I can make a great compilation of moments from my wedding and share it with my fans.

José Carlos In Figure 1.5 we can discover more about the persona that we named José Carlos.

A 43-year-old man, he lives in the city that saw him came to life and grow, Ovar.

Filipe Nogueira

Job Title
Actor

Age
25

Highest Level of Education
University (Bachelor degree)

Goals

- Filter photos by location to get the best photos of his wedding
- Intuitive and easy to use application

Frustrations

- Slow processing times
- Application crashes
- Not being able to make a compilation of the best moments of his wedding

Environment

Being a very active public figure on social networks like Instagram and Facebook, in addition to using his personal computer for his professional projects (such as preparing the scenes of the next episodes of the soap opera in which he participates), he finds new technologies and applications easy to handle. Since he is a very busy person with his professional projects, he looks for an application that is easy and quick to use that does not spend a lot of time to master.

Figure 1.4: Persona 3 - Filipe Nogueira

He started by studying Architecture Faculdade de Belas Artes, but dropped out of the course a few months later. He later entered Photography at Universidade Lusófona and, having completed his degree, opened a studio of his name – Estúdio Almeida – on the main avenue of the city of Ovar. Approaching the start of the new edition of Viagem Medieval em Terras de Santa Maria, an annual event held in the municipality of Feira around the castle where events and traditions of the Middle Ages are recreated, José intends to advertise photographs of medieval-themed weddings in which he participated in the showcase of his studio. So, having a photo of a medieval wedding, he intends to use the application to find similar images.

José Carlos is looking for an application that is easy to use and that is quick to perform its necessary functions in a vast cluster of photographs, in order to be able, through a photo indicated by him, to find all the related ones that are in the folder that is being used and inspected.

User story: As José, I want to get all photos similar to one indicated by me, so that I can decorate my showcase.

Viviana Nunes In Figure 1.6 we can discover more about the persona that we named Viviana Nunes.

José Carlos Almeida

Job Title
Photographer

Age
43

Highest Level of Education
University (Bachelor degree)

Goals

- Pick a photo and find similar images to decorate his shop window
- High performance application

Frustrations

- Slow processing times
- Application crashes
- Application with few features

Environment

He has no problem dealing with new technologies, since he is always up-to-date with the new technological devices that allow him to take photographs with better quality and less effort. For him, the important thing is that the application complies with the requirements that it proposed, allowing him to achieve his objectives, even if for that he has to learn all of its functionalities.

Figure 1.5: Persona 4 - José Carlos

Viviana is a 20-year-old student of Business Relations at the University of Coventry, England. She used to spend her summer holidays in Portugal, taking the opportunity to see her family and friends again. An unconditional fan of summer festivals, she used to attend Vodafone Paredes de Coura every year, accompanied by a group of friends. According to her, more than watching the concerts of her favourite artists, it was the coexistence with different people in the accompaniment that made the stay memorable. The barbecues, the conversations around the bonfire in the moonlight, the singing until dawn, everything was recorded in her memory. In times of pandemic, Viviana will not be able to leave England and appear at Vodafone Paredes de Coura for the first time. Feeling nostalgic, she decides to recall the moments experienced in previous editions. Thus, she uses the application to search for the images of the festival, and tries to edit them in order to identify the specific edition in which they were taken.

Viviana hopes to find an easy-to-use application that is intuitive and that supports a large number of photographs, in order to be able to edit each photo and tag the name of the festival and the year in which it was taken.

User story: As Viviana, I want to edit manually each photo, adding the name of the festival and the year in which it was taken, so that I can remember all the good times I spent.

Viviana Nunes

Job Title
Student

Age
20

Highest Level of Education
University

Goals

- Manually insert tags into photos to identify the festivals she attended
- An application that supports a large number of photographs

Frustrations

- Not being able to edit all the photos she has
- Slow processing times

Environment

Born in the 21st century, she does well with new technologies. She doesn't care if the application is not easy to use, as she believes she is able to use it anyway. For her, the important thing is that it allows an allocation of a large amount of data to be able to edit all of her photographs, so she would prefer that the application had a high-performance index.

Figure 1.6: Persona 5 - Viviana Nunes

1.4 State of The Art / Related Work

There are many image search engines available, each with its own set of features and functionality.

Some of them are presented in the list that follows:

TinEye Reverse Image Search is a reverse image search engine which means that the user can, using a URL of an image or the image file, search and find a similar or even the same image from all around the web.

In Figure 1.7 we can see it working and the results showing.

Google is a well-known engine which not only allows us to search using images or keywords as it gives us many filters for those results. But some competitors' browsers also have their image search engines, like Yahoo or Bing.

In Figure 1.8 we can see the search results.

Flickr is a cloud storage based image search engine, which means that the user can publicly or privately store images on the site and access its search engine for the images.

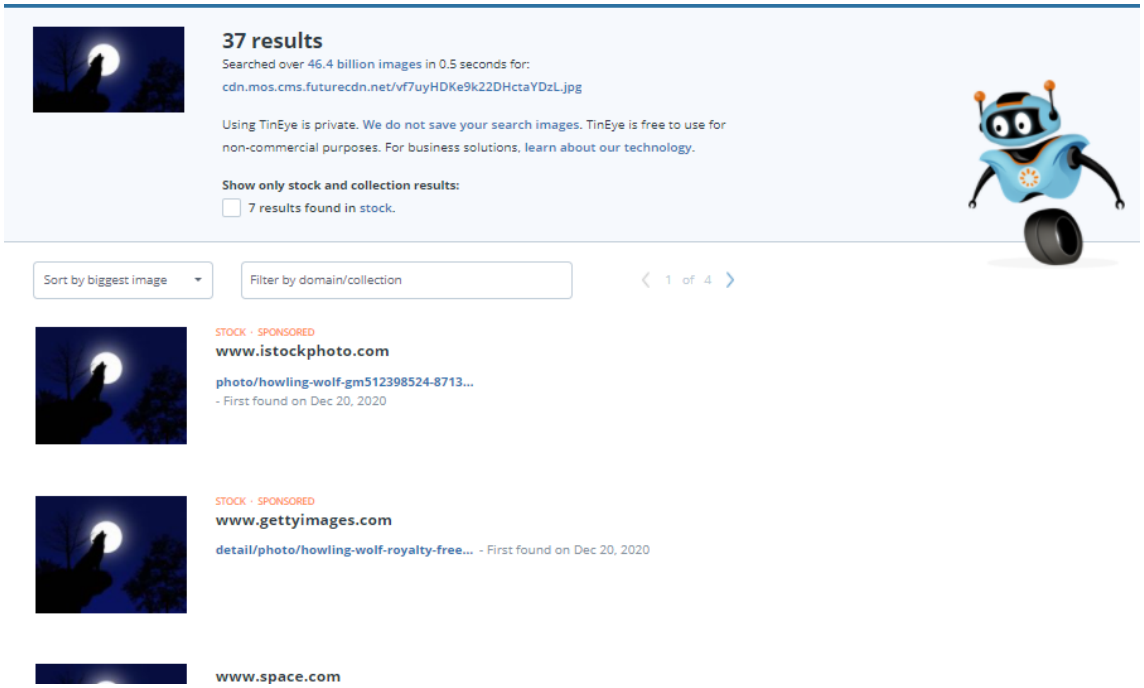

Figure 1.7: TinEye Reverse Image Search results page screenshot

In Figure 1.9 we can see the search results.

Google Photos is a cloud storage based search engine that automatically analyzes photos, identifying various visual features and objects. The user can search using any word and it will return results from three major categories: People, Places and Objects [13].

Google Photos also has a face recognition feature, that can not only identify human faces but pets as well, grouping similar ones together. Geographic landmarks, birthdays, buildings, animals, food, and more are also identified.

Different forms of machine learning in the Photos service allow recognition of photo contents, automatic generation of albums, animation of similar photos into quick videos, and improvement of the quality of photos and videos

Because of the optical character recognition (OCR) [16] technology also found in Google Lens [6], Google Photos can be used to search for text inside the images.

It also uses Vision AI [4] which is a machine learning program that, using photos and training models, can learn how to identify certain patterns in the photos.

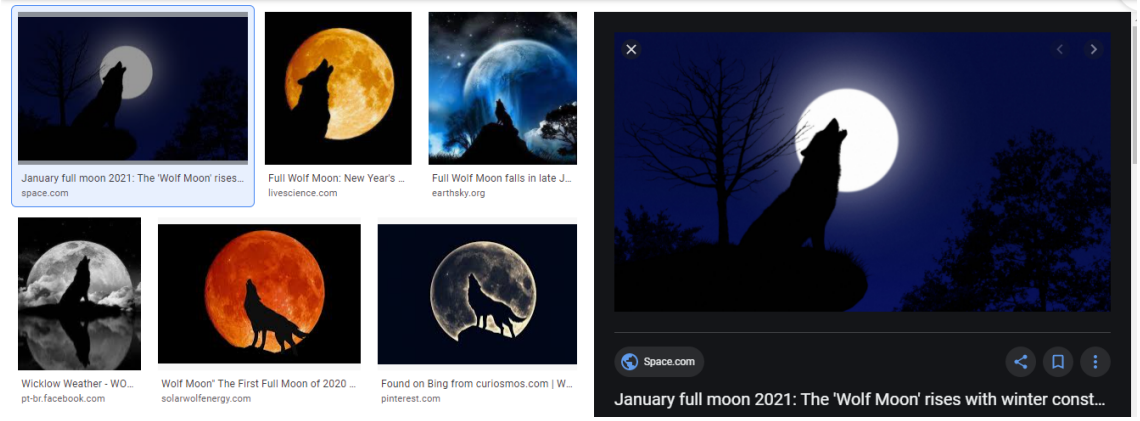

Figure 1.8: Google images search result page screenshot

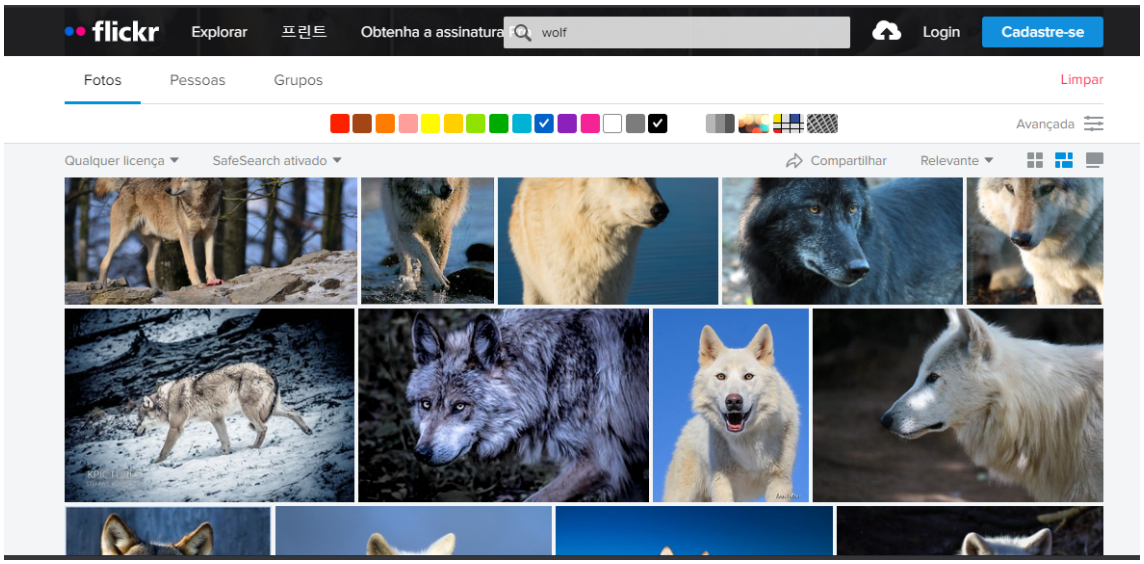

Figure 1.9: Flickr search result page screenshot

Amazon Photos uses Amazon Recognition [3] which can recognize scenery, objects and their tags and it also allows the implementation of personalized tags. It has text recognition and inappropriate photos filtration. It also recognizes faces and identifies their expression based on their eyes and mouth.

- **Amazon SageMaker Ground Truth**

  It is a labelling training program where the user can create a custom label and train it to automatically label what it should. Amazon also has its elastic search technology called Amazon Elasticsearch Service.

Loggy Our work has shared features with Loggy, a project that is being worked on at the moment by Ricardo Ribeiro.

Ricardo's work consists of recording the life of a person via smartwatch, wearable cameras, sensors and many other technologies and filtering and processing these images in the cloud.

Then Loggy performs the part that is the most similar to our work, the organizing and filtering of images.

Currently, the State-of-the-art approaches in this area have already reached considerable performance for detecting/recognizing objects using deep learning approaches, generally Convolutional neural networks (ConvNets).

The following technologies were investigated during the development of Loggy and recommended to us for investigating and even usage:

- ImageNet Large Scale Visual Recognition Challenge is a benchmark in object category classification and detection on 1000 object categories and millions of images.
- Microsoft COCO - Common Objects in Context is a dataset to improve the state-of-the-art in object recognition by placing the question of object recognition in the context of the broader question of scene understanding (containing more than 200,000 images and 80 object categories).
- Among the most used ConvNet architectures we can find the VGG, ResNet, Inception, ResNeXt, EfficientNet

Some of the technologies investigated for scenery recognition are the ones that follow.

- **MIT Indoor 67** - Indoor scene recognition benchmark dataset, a challenging open problem in high-level vision.
- **SUN** - A scene categorization benchmark dataset providing a large collection of scene categories.
- **Places** - Image database for scene recognition, allows learning of deep scene features for several scene recognition tasks, intending to establish new state-of-the-art performances on scene-centric benchmarks.

The Loggy web application was divided into different modules:

- Upload of multiple images into the database;
- Pre-processing the images using ConvNet pre-trained models on the three datasets, ImageNet, Microsoft COCO and Places365, to extract useful visual concepts and also EXIF metadata from the images;
- Analyse the extracted content and show the results into statistical charts;
- Visualization and summarization into image categories based on the date, scenes, environment and objects.

1.5 Conceptual Modeling (architecture diagram)

The development of our product involved both the development of a front-end and back-end component.

The front-end of our product was developed using HTML, CSS and JavaScript technologies, while the back-end was developed in Django.

Elastic Search is used as a non-persistent database for search purposes – and is explain in more depth in subsection 2.2 – and Neo4j, integrated with Neomodel, as a persistent database for all data resultant of image processing – more detailed in subsection 2.1.

Figure 1.10 shows a diagram representation of our Imageable's architecture.

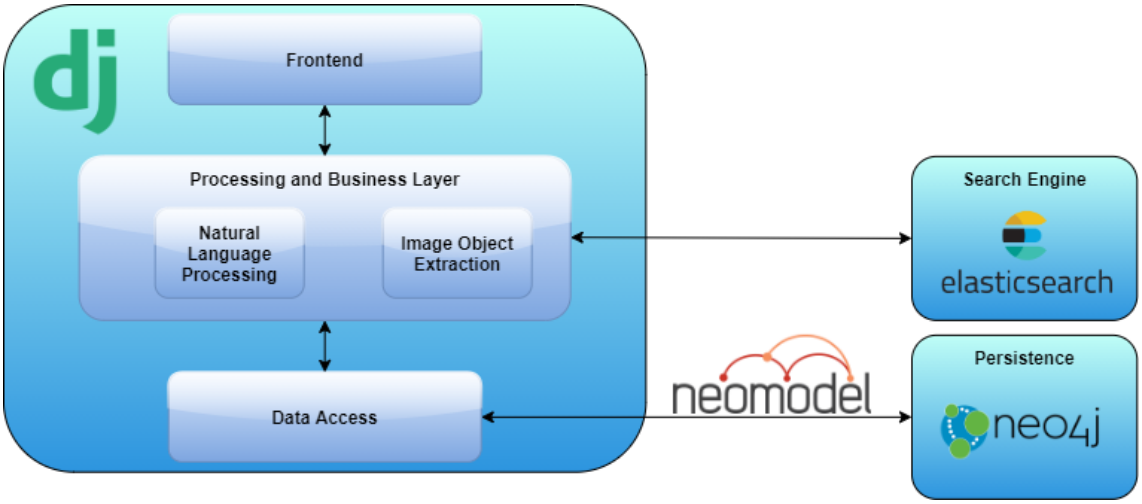

Figure 1.10: Architecture Diagram for Imageable

2 Procedure

2.1 Database and Domain Model

Being a graph database, Neo4j treats the relationships between entities as equally important as the entities themselves. Because this work consists of recognizing and connecting data to images based on their content and properties, this was our database of choice.

The data model for our project is shown on Figure 2.1. It is possible to see that the `Image` entity has a lot more attributes when compared to the other entities. This is because it is the central element of our project.

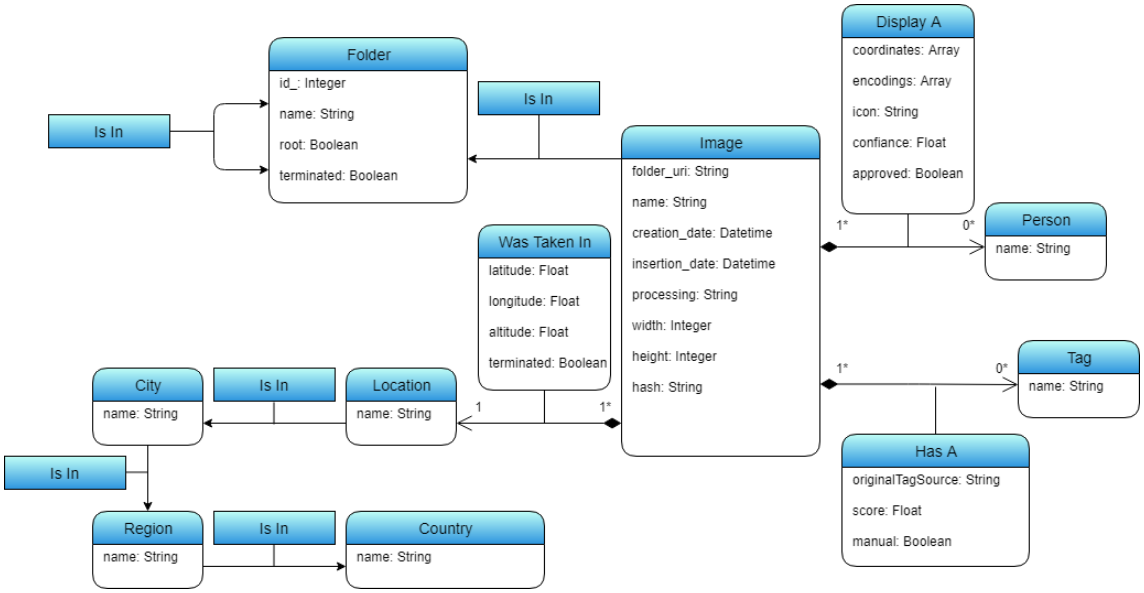

Figure 2.1: Domain Model

This `Image` entity has many relationships and entities alongside it. For example, it has a relationship `Was Taken In` with the `Location` entity – that will contain the coordinates where the picture was taken, if the image's metadata has this information. On the other hand, this entity also has a relationship with a node `City`, which in turn has one with the node `Region`, and finally `Country`, in a way to represent how it is in the real world.

The image will also have one or many relationships to the `Tag` entity. This relationship contains valuable information, such as its origin – for example, if the user added it manually, if it comes from one of the implemented algorithms,

specifying which one, etc –, the score if it is an algorithm-based tag, among others.

If the image contains people, then it can have one or several relationships to `Person` entities, which are only created by the face recognition component. The relationship between these two contains much information, such as the coordinates of the bounding boxes, the face thumbnail path that is shown on the faces page, and if the user checked its name, among others. The `Person` entity *per se* only contains information about its name.

2.2 Elastic Search

Neo4j is a powerful database that can quickly find nodes directly or indirectly connected – this is, with more than one level of connection – with a given relationship. However, because most of the searches that our end customers need to do are much more complex – for example, searching images by giving several keywords – or **tags** –, the results will have to be ordered according to their relevance related to the given tags. To give an example, if the client searches for the tags 'dog' and 'cat', the first results should be those related to both 'dog' and 'cat', and only then those with only one of the relations should appear. For this reason, Neo4j may not be appropriated for this type of search.

Performance has always been one of our main concerns. Therefore, we used Elastic Search (ES) – a NoSQL database, but that is better known as an efficient search engine – to implement the search. It is a distributed and "real-time" search and analysis engine, built on the full-text search engine Apache Lucene [9]. Like ES, Neo4j is also built on Lucene, but it is a less efficient and flexible solution.

Databases, Tables, Rows and Columns are basic concepts in relational databases. ES also has its similar structures: Indexes, Documents and Fields. To make it easier to understand, we can think of Indices as Databases, Documents as Rows, and Fields as Columns. Although there is a structure corresponding to Tables – Types –, it has been deprecated. [20]

For our project, we created an ES model –- `ImageES` –, which is a subclass of `Document`, using a library `elasticsearch-dsl`.

```
class ImageES(Document):
        uri = Text(required=True)
        hash = Text()
        tags = Text()

        class Index:
             name = 'image'
```

It has 3 fields:

- **uri** – required `Text` type field that is used to store the full path of an image.
- **hash** – `Text` type field used to store the unique hash code of an image. It is declared as text because the number types, such as `IntegerProperty`, would not allow the storage of such a large value.
- **tags** – even though it is declared as a `Text` type, it can also by an array of strings, and it is used to store all tags of an image, such as the detected objects and places, etc.

The inner class `Index` represents its Index, named `image`.

The `Keyword` type was considered for the field tags, but we found that the use of this type would bring unwanted consequences. For example, if the user searched by 'ocean', ES would not return images with the tag 'ocean deep'. Using the `Text` type solved this problem since when a new document is inserted into ES, it will split these fields by spaces, creating inverted indexes for them.

```
def search(tags,page):
    image_in_page = 20
    q = Q('bool', should=[Q('term', tags=tag) for tag in tags],
   minimum_should_match=1)
    s = Search(using=es, index='image').query(q).extra(from_=(page-1)*
   image_in_page, size=page*image_in_page)
    return s.execute()
```

Regarding searches, `Q` (which means query) is used to create the query, which is executed by `Search`. Both `Q` and `Search` are native classes of the library. The query is configured with `minimum_should_match = 1`, which specifies that each image found has at least one of the given tags. These tags – an array of keywords/strings

– are a parameter required by the function. Besides that, by default, no matter how many records ES finds, it only returns the first 10. To bypass this problem, it is necessary to use pagination. The variable `imagesInPage` has the purpose of indicating the size of each page. By experimenting with different values, we found that the number 20 would be a good compromise, and decided to use it.

2.3 Modules

2.3.1 Object Extraction

An important component of our project is to be able to search for images by objects that they contain. In order to achieve this, we need to use an algorithm that analyses the picture and returns a list of objects and the respective confidence that they are indeed present.

We decided to use the deep-learning Python's library PyTorch. From PyTorch Hub we loaded a pretrained YOLOv5s model [19]. Using this model we were able to extract the information we needed from each one of the images to be processed.

2.3.2 Places

Users do not only want to describe an image with the objects detected in it, sometimes they also want to be able to find images by the scene presented in it.

To implemented this feature, we used a model trained on a dataset called Places365 [22]. This dataset has 2 different versions: Places365-Standard, and Places365-Challenged. We used a ResNet18 PyTorch model pre-trained on the 1.8 million images in the first version of the dataset, making it capable of identifying a scene from 365 different ones – as the name suggests.

It recognizes scenes/places present in an image, returning a list containing all the scenes that could be identified in the image – all 365 of them – along with the confidence that that scene is depicted in the picture. This list is also ordered by this same confidence in descendent order – that is, the scenes with more probability of

having been correctly identified appear first in the list. This means that the results have to be filtered after the model returns its results.

We solved this by accepting only the first 10 items in the list, granted that they have a confidence value greater than a selected threshold, which we set to 0.1. This value was set after experimenting with different values.

2.3.3 Facial Recognition

Facial recognition – the system's ability to predict and assign the correct person's name to their face on a picture – was also a feature that we considered important since the beginning. This would allow the user to search pictures by the names of the people shown in them.

For this goal, we used a python library called face_recognition. This package is nothing more than a wrapper around Dlib's face recognition functionality, providing a simpler interface for us to use to implement this feature. [1]

The methodology we used to identify new faces on the images that were uploaded is as follows. When it is going through the processing phase, the algorithm finds and extracts the bounding boxes delimiting the faces found. With the coordinates, a small thumbnail of the profile of the person in that picture is cut and saved on the disk, while the encodings – the arrays that make up the images – related to that portion of the picture are saved in memory – for fast access – and also in the database.

The thumbnails are later used so that the user can see all faces detected by the system, allowing them to change their associated names.

On the other hand, the encodings will have a primary role in the prediction of the person's name, as explained below.

After this initial set-up, the encodings are used to predict the name of the face. The algorithm works as follows. For every name known to the system, the encodings related to it are going to be compared to the new unknown encoding, returning an array of decimal values between 0 and 1. These values correspond to the distances between each of the encodings in the array with the current one, and the higher the value is, the less alike the faces in their pictures. Each of the values in the

gotten array is then subtracted by 1 and the result multiplied by -1. This is to get an array of numbers between 0 and 1 with higher values if the images are closer to each other, opposite of what we had before.

Every encoding and name is associated with a score. Our goal is to calculate the score for this encoding for every name in the system and find out which name gives a higher score. To do this, we multiply every value of the previous array by 0.85, and also multiply every score from the current name's encodings by 0.15, creating an array with such values. Then, element-wise, we add both of these arrays and calculate an average.

The score we are looking for is this value we found subtracted by a penalization factor that weights 10%. This penalization factor is related to the number of examples that the system has associated with a certain name. If there are 50 or more encodings associated with a person's name, then this factor will be 0, meaning that there will be no penalization on the score. On the other hand, if there are less than this number of examples, then this penalization is going to be according to following the expression:

$$penalization = \frac{50 - \#examples}{50}$$

We can sum up the calculations in the following expressions.

$$score = avg((-distances - 1) \times 0.85 + array\, of\, existent\, scores \times 0.15) - penalization \times 0.1$$

$$penalization = \begin{cases} \frac{50 - \#examples}{50} & if\ \#examples\ \leq 50 \\ 0 & if\ \#examples\ \geq 50 \end{cases}$$

After doing this for all the names in the system, the one with the highest score is chosen, provided that it is higher or equal to 0.35. If this verifies, then the encoding is saved on the system as belonging to that name. If not, then a new person is created and assigned to a fixed-length string with random characters and the score associated is set to 1. The threshold value of 0.35 was found through

experimenting with different values and finding out which one provided the best balance.

Later, the user can change the name that the system defined for the faces. When this happens, the encoding and related information is set to the new name, and the score is also set to 1 since the user validated that value. After this change, all the other faces that the user has not confirmed are predicted again, with the system having more information, which in turn would be reflected on a better prediction.

2.3.4 Text Extraction

For text recognition and extraction we used Google's tesseract-OCR engine – an open-source OCR engine available under the Apache 2.0 license [16] – and the python package `pytesseract` [11] to integrate it into our project.

Through experimentation we concluded that the engine itself is not precise enough. To improve the results, we did some image manipulation and configuration of the engine. After this, we managed to get acceptable results.

In particular, 2 algorithms were applied. We will use Figure 2.8 to illustrate this process.

1. Based on tutorials [2] [15] we implemented this algorithm that checks for text and crops the image around the text. This will return several cropped images, such as the Figure 2.2 and 2.3, among others. All of these images will be passed into the mentioned OCR algorithm, which in turn will return the text extracted as a string.

Figure 2.2: Cropped image passed to the OCR engine in algorithm 1 - example 1

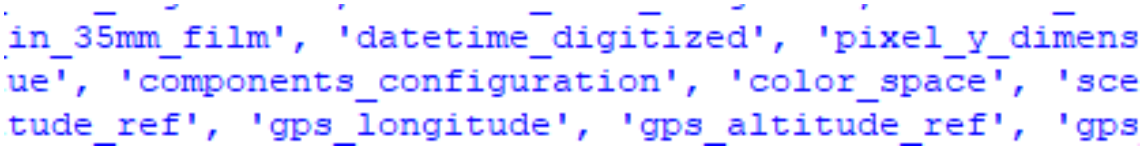

Figure 2.3: Cropped image passed to the OCR engine in algorithm 1 - example 2

2. Based on a tutorial [21] we implemented this algorithm that is composed of many phases:

   (a) The image is passed through a gray filter that transforms it into a gray scale version of it. The picture after the transformation can be seen on Figure 2.4.

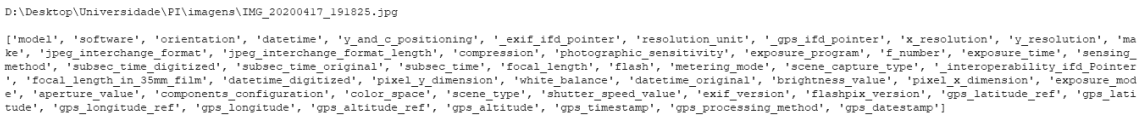

Figure 2.4: Figure 2.8 after grey filter

   (b) This gray scaled image goes through a blur filter that smoothes out the image, as shown on Figure 2.5.

Figure 2.5: Figure 2.4 after blur filter

   (c) Then this version is passed into a threshold filter – a technique where a threshold value is firstly defined. Then each pixel in the image is compared to it and converted to either 0 or 255, depending on if the pixel is lower or higher than this threshold. Passing the previous image through this filter generates Figure 2.6.

Figure 2.6: Figure 2.5 after threshold filter

   (d) The last filter that the image has to go through is a dilation filter. This will increase the boundaries of the regional foreground pixels, expanding the area around the text, facilitating its identification. The results are shown in Figure 2.7.

   (e) To finish this process, the coordinates of each area around the text will be extracted, allowing for us to go to the original image and extract the

Figure 2.7: Figure 2.6 after dilation filter

text to pass it through OCR.

2.3.5 Exchangeable Image File

Exchangeable Image File (EXIF) is a format for storing metadata in image and audio files as we can see in Figure 2.8.

Metadata, which is often present on images, can give a big amount of information about it, such as the location where it was taken and the date, data about the camera settings, etc. It would be useful for us to retrieve this information from the images, allowing the user to search for things such as the location where the picture was taken.

Using the Python library EXIF [18] we were able to extract some of this information from the pictures, provided that they have EXIF metadata. In particular, we extracted their creation date, height and width, and GPS coordinates.

Having the GPS coordinates, there was the problem of how to convert them to a city name. For this, we used a library called `reverse_geocoding` [17].

2.4 Simple File System Manager

Our `Folder` model in Neo4j stores only the name of the folder, not the full path to the folder. For example, for the path `C:/a/b/c`, Neo4j creates 4 nodes, and each of them is assigned the field name `C`, `a`, `b` and `c`, respectively.

Figure 2.9 shows a representation of this path in the database. We can see that it resembles a tree. However, because it is in the database, it is not easy to take advantage directly of this structure because it would involve complex operations. Besides, we can also point out another problem with this structure. When we have two folders with the same name, checking if one of them exists through a `Cypher` query would be complicated.

To solve the problem mentioned above, the class `SimpleFileSystemManager` was created. This class is dedicated to managing folders in the database, facilitating operations like inserting and removing them.

```
class SimpleFileSystemManager:
    ## Starting with an empty dictionary
    def __init__(self):
      self.trees = {}  # key: folder name, value: node.
```

This manager has only one attribute: `trees` – a dictionary, used to store the root folders of the paths. The keys of this dictionary are strings with the name of the folder, that are mapped to a Node object. For the previous example, `C:/a/b/c`, the key will be the root folder's name – `C`, and the value will be the Node loaded from the Neo4j that corresponds to the folder.

```
class Node:
    def __init__(self, name: str, id: int, %terminated=False):
        self.name = name  # folder name
        self.children = {}  # key: folder name, value: %node
        self.parent = None
        self.id = id
        self.deleting = False

        # if current node is the last one of a path, it is terminated
        # ex: for uri C:\User\abcd\ef\, node 'ef' is terminated
        self.terminated = terminated
```

This `Node` class has the important information present in the Folder entity of the Neo4j model – like folder name, id, and the `terminated` flag. This last field indicates if the node is the last one of a URI. There is, however, another case where a Node can be terminated –- when the corresponding folder has images. If the user tries to upload a folder without images it is ignored, without any new nodes having been created by Neo4j. Only the folders with images are processed, thus it is possible to say that all `Nodes` that have images are `terminated`, and that all `terminated Nodes` have images.

The `children` dictionary is similar to the `trees` of the manager. The only difference being that the `children` stores only the folders that are children of the current folder. These children folders will then do the same thing, and so on.

Using the `Node` class, the manager can build a Trie Tree – which is what is shown in Figures 2.9 and 2.10 – with the folders, each folder representing a node in the tree. Different folders that have the same *parent folders* can share those common nodes between them, which is a typical characteristic of a Trie Tree. This tree is built when starting the application when all `Nodes` are loaded to memory from the database.

Now it is easy to find, insert and delete folders. The time complexity for finding or inserting a folder is O(`#folders in the URI`) – also known as linear. For example, the path `C:/a/b` is O(3), because there are 3 folders in it – `C`, `a`, and `b`.

The process of adding folders is straightforward. First, it is verified if the root folder `C` exists in the dictionary trees. If this is not the case, then it means that the folder `C:/a/b` – which we are considering for this example – has not yet been processed. The manager will have to create new nodes and store them both in the memory tree and the database. Newly stored nodes are "connected". The parent nodes `C` and `a` should store their children nodes `a` and `b`, respectively, in their `children` dictionary. These children nodes also have a reference to their parent – stored in the variable `parent`.

On the other hand, if `C` exists, then we need to do the same for the folder `a` in the `children` dictionary of the `Node C`, and so on. If all the nodes `C`, `a` and `b` exist in the database, and `b` is terminated – has images –, then the path is considered to exist.

The process of deleting a folder is more complex. The manager first needs to find the last node of the given path. If it exists, then its children nodes – or child folders – need to be deleted. Before an image is deleted, the relationship between it and the current node – or folder – needs to be disconnected. Only then an image can be deleted, granted that it does not have more relationships with other `Folder Nodes` – an image can be in different folders! A `Folder Node` can only be disconnected after this process has been done for all the images in it.

We can sum up the process as follows:

1. Get the last node of a given path

2. Try to delete images of its child nodes
3. Delete child nodes
4. Try to delete images of the last node
5. Delete the last node
6. If parent nodes have no more children and is not `terminated`, delete them

2.5 Natural Language Processing

Natural Language Processing (NLP) is, according to International Business Machines Corporation, an American technology company from New York, the branch of computer science -— and more specifically, the branch of artificial intelligence or AI—concerned with giving computers the ability to understand text and spoken words in much the same way human beings can[8]. In other words, NLP tries to understand the human language, the relations between words and their meanings, using complex data and deep learning models.

For this project, NLP was implemented to make the searches for images more robust and efficient, allowing to search not only for photos with tags that are exactly one of the words entered by the user in the search bar, but also searching for words from the same family, or even synonyms. This module is defined in the `nlpFilterSearch.py` file.

Everyone has already experienced the feeling of searching for some topic on Google and, while putting more and more keywords in the search bar, the results obtained were less and less significant than what would be expected. Now suppose a similar case, where an Imageable user writes ”Are you going to show me photos of my cats?” in the search bar, trying to look for images of their cats. Analyzing this sentence, it appears that it contains a lot of unnecessary content to impact the quality of the search – words that, by themselves, do not define what the user wants to search and, therefore, do not contain any relevant content, and even punctuation. The user could have just entered the word ”cat” which would have been able to satisfy their demand. Below, processes for removing expendable content will be explained.

The Natural Language ToolKit (NLTK) from Python was used to develop this module from the project, because it provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning[12].

Consider the following sentence: ”loving someone is something beautiful, just like the nature. I love the world.”

Let’s process and filter this query to check what it will look like in the end.

First, we need to divide the sentence described above into words, and for that, we use the `tokenizeText()` function that uses `word\_tokenize()` from the `nltk.tokenize` package. The output of this function for this specific case is shown below:

```
{'world', 'just', 'nature', 'i', 'loving', 'like', 'love', 'something',
    ',', 'is', 'beautiful', '.', 'the', 'someone'}
```

Note that even the punctuation is divided and considered to be a word.

The next step is to remove all punctuation from the obtained list * of words, recurring to the `filterPunctuation()` method that iterates over all the elements from this list and checks if the element is punctuation, removing it if it is the case. The output is the following:

```
{'love', 'just', 'beautiful', 'loving', 'like', 'i', 'is', 'world', '
   someone', 'nature', 'something', 'the'}
```

The following function to make use of is the `filterStopWords()` one - the one that iterates over the list of tokens and removes all words that belong to `stopwords.words(‘english’)`, a list of some commonly used English words that ”do not contain any relevant content”. The output is:

```
{'something', 'loving', 'beautiful', 'love', 'world', 'like', 'nature',
    'someone'}
```

Now it’s time to normalize each word - and for that, there are two implemented methods: the `stemming\_method()` and the `lemmatization\_method()`, responsible for **Stemming** and **Lemmatization** processes, respectively. First,

an introduction to these concepts: stemming is the proceeding of producing morphological variants of a root/base word and does not care about the context of the specified word and is more used to get, for example, the infinitive of a verb; in other words, it removes suffixes (sometimes when it should not). It runs faster than lemmatization, which is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. It is similar to stemming but it cares about the context of the word, requires a dictionary look-up and tries to return its lemma - and that's why it runs slower than stemming.

In the `stemming\_method()` function there are two different stemmers - the `PorterStemmer()`, the most common used stemmer, but slower one, and the `LancasterStemmer()`, faster but sometimes too much aggressive, making some inconsistencies while removing the suffixes. This function iterates over each word and tries to stem it with one of the defined stemmers and then confirms if the generated output belongs to an English dictionary. If it does not, the original word is stemmed again, but this time using the other stemmer, and in the end, we check again if the outcome exists in the dictionary; if it does not, it means that both stemmers failed to stem the initial word, and that's why we return it instead. The example of output is the following:

```
{'beauty', 'like', 'something', 'world', 'nature', 'love', 'someone'}
```

The next step is to lemmatize each word, but first, we need the corresponding part-of-speech tag, which is its word class, because the `lemmatize()` function defined in the `lemmatization\_method()` one needs a part-of-speech tag as a second attribute - if we use the default POS-tag, it will always consider the word as a noun. This is possible using the posTagging() function:

```
{('like', 'IN'), ('something', 'NN'), ('love', 'VBP'), ('someone', 'NN
    '), ('beauty', 'NN'), ('nature', 'NN'), ('world', 'NN')}
```

[10]

Finally, we pass to the `lemmatization\_method()`, where is used a `WordNetLemmatizer()` to lemmatize each word, returning the original one if it could not - so, it's not necessary to confirm if the word exists in an English dictionary. So in the end, our test case looks like the following:

```
{'nature', 'love', 'something', 'world', 'like', 'someone', 'beauty'}
```

Let's add synonyms to increase the number of search results, using the `get\_synsets()` function, which uses `wordnet.synsets()`. To better understand this method, two practical examples for "love" and "nature":

```
wordnet.synsets("love")
```

```
[Synset('love.n.01'), Synset('love.n.02'), Synset('beloved.n.01'),
    Synset('love.n.04'), Synset('love.n.05'), Synset('sexual\_love.n
    .02'), Synset('love.v.01'), Synset('love.v.02'), Synset('love.v
    .03'), Synset('sleep\_together.v.01')]
```

```
wordnet.synsets("like")
```

```
[Synset('like.n.01'), Synset('like.n.02'), Synset('wish.v.02'), Synset
    ('like.v.02'), Synset('like.v.03'), Synset('like.v.04'), Synset('
    like.v.05'), Synset('like.a.01'), Synset('like.a.02'), Synset('
    alike.a.01'), Synset('comparable.s.02')]
```

So, `wordnet.synsets()`, as we can analyse from both examples above, returns a list of meanings/synonyms of a specific word, and we call it for each word from the list of existing tokens; in the end, we get the lemma for the first five elements from the returned list, or its size, if the list is smaller.

Although this module was used to filter the initial query introduced by the user, not only the filtered query is used to search for image results - the initial one is also considered.

*consider "list" in the broad sense of "a group of things", for language simplification. In the code sets were often used for performance improvement reasons.

```
D:\Desktop\Universidade\PI\imagens\IMG_20200417_191825.jpg

['model', 'software', 'orientation', 'datetime', 'y_and_c_positioning', '_exif_ifd_pointer', 'resolution_unit', '_gps_ifd_pointer', 'x_resolution', 'y_resolution', 'ma
ke', 'jpeg_interchange_format', 'jpeg_interchange_format_length', 'compression', 'photographic_sensitivity', 'exposure_program', 'f_number', 'exposure_time', 'sensing_
method', 'subsec_time_digitized', 'subsec_time_original', 'subsec_time', 'focal_length', 'flash', 'metering_mode', 'scene_capture_type', '_interoperability_ifd_Pointer
', 'focal_length_in_35mm_film', 'datetime_digitized', 'pixel_y_dimension', 'white_balance', 'datetime_original', 'brightness_value', 'pixel_x_dimension', 'exposure_mod
e', 'aperture_value', 'components_configuration', 'color_space', 'scene_type', 'shutter_speed_value', 'exif_version', 'flashpix_version', 'gps_latitude_ref', 'gps_lati
tude', 'gps_longitude_ref', 'gps_longitude', 'gps_altitude_ref', 'gps_altitude', 'gps_timestamp', 'gps_processing_method', 'gps_datestamp']
```

Figure 2.8: Exif metadata extracted from an image

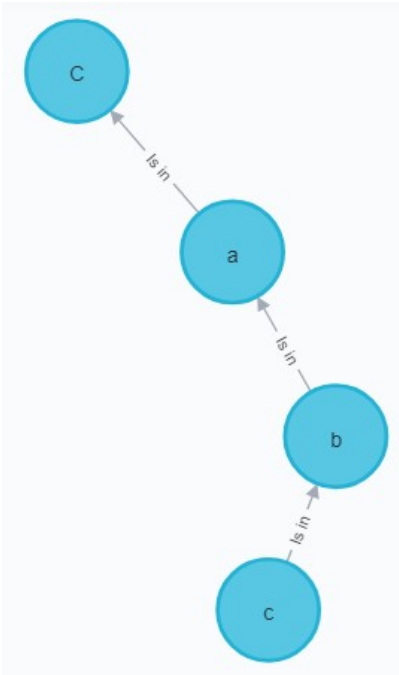

Figure 2.9: Path “C:/a/b/c” in the database

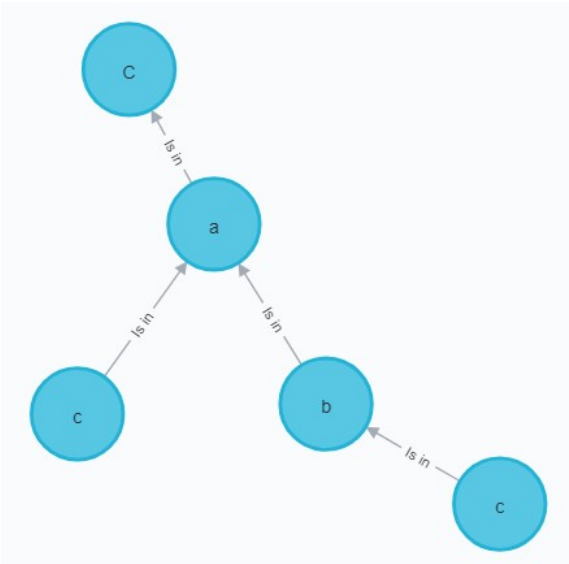

Figure 2.10: Representation of multiple folders with the same name (“c”) in the database

3 Results and Discussion

3.1 Presentation of the Interface

The user, when entering the application, is faced with a home page with a top bar containing many page navigation elements. First, it contains a search bar where the users can search for images by writing text. Clicking on the button on its right side allows the user to search for similar images to a given one. It also contains several buttons which redirect to different pages:

- To a page with list of all the people recognised in the processed photographs. In this page, the user is able to check the names that the system predicted for the people it recognized, and also validate or change them in order to help the system "learn" their names and make better predictions.
- To a page where the user can manage their folders. In particular, they can add new folders – by passing their absolute path –, reload all the inserted folders – which makes the system inspect all of them, processing the images that have not yet been uploaded (maybe because the user added them to these folders after they were uploaded to the system). Here they are also able to remove folders – with all the related images. Besides all of this, if the user clicks on one path, they are able to see all pictures present in that folder.
- To a dashboard with some statistical data and other information associated with the photos. More detailed information about this component can be consulted in sub-subsection 3.4.3.

When searching for an image by text – tags –, the user is redirected to a results page. Here they can see images that are associated with one or more tags previously inserted in the search. In case that the search returns no results, no images are shown on this page.

These images are sorted by relevance and can be filtered by the tags' origin – the modules that originated them –. by dimensions, among other criteria that are detailed more on sub-subsection 3.4.2.

Each image contains a set of information that can be seen in a modal. This includes

its name, dimensions, and the set of tags associated with it. Next to the tags, there is a button to edit them, allowing to remove or insert new instances as well.

In addition to this, the user can export the presented results in two formats - in a ZIP folder with a copy of the images, or an excel document with the information associated with each image. It is also possible to only export a sample of the results by individually selecting each desirable image.

At the top of this page, there is a bar where there are several buttons, which redirect to different pages - the first one to a list of all the people recognized in the processed photographs, the second to the ”Insert New Source Folder” page and, finally, a dashboard with some statistical data associated with the photos and some additional data.

3.2 Quality Assurance

3.2.1 Code Testing Techniques

In order to make development easier and foolproof, we decided to test our code.

The tests we created were mostly to check each algorithm and some fundamental functions of the system, as a way to guarantee that they return what they are supposed to. This is especially important to make sure that consequent changes to our code won’t affect the existing working functionalities.

3.3 SonarQube

Even though we started early in the development with using static analysis code tools – in particular SonarQube, which we had used before for a university course –, the earlier dashboards did not show great results. Figure 3.1 shows that this tool found many code smells and even a few bugs.

After revising our code and fixing the identified problems, the situation started to change for the better, as can be seen in Figure 3.2 and Figure 3.3.

At the end of the development of Imageable, after a big refactoring had gone through – involving the cleaning of code smells, the removal of unnecessary

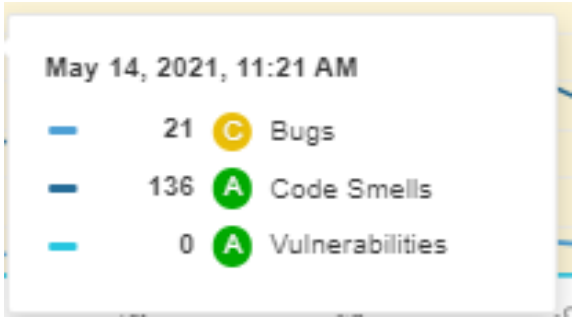

Figure 3.1: SonarQube first scan results

`prints` and implementation of logging, the results shown in the SonarQube dashboard – in Figure 3.4 – were much better.

As a way to sum up, in Figure 3.5 we can see the evolution of the number of code smells and bugs throughout our project construction.

3.3.1 Large Scale Test

As this project needs to be able to handle dealing with a large number of images, we needed to test the performance of our system. To do this, the "COCO 2017 Unlabeled images [123K/19GB]" dataset [5] was used. As the name indicates, this dataset contains 123000 images in a total of 19GB of space.

Of all the images available, our system managed to process 23000 of the images, until we stopped it. We noticed that the system could handle this large amount of data, creating more than 2336 different tags, and only using 70BB of space for the thumbnails.

This test also showed some not so good aspects of our application, such as the fact that the page to manage people takes too long to load when the system is dealing with such a heavy load of information. Even though it takes some time, it still works correctly.

3.4 New Functionalities

3.4.1 Export Results

In the first versions of our application, the search results were simply displayed. After noticing this, some professors suggested we should add new functionality –

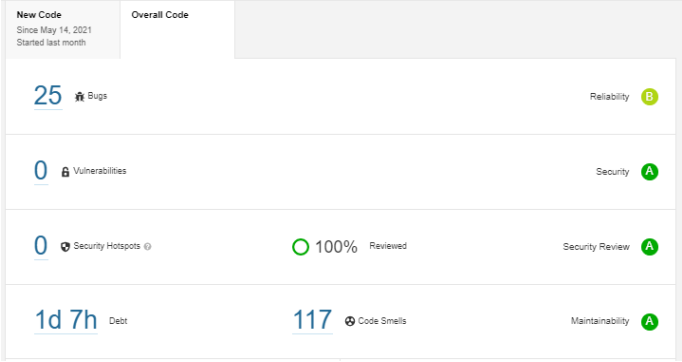

Figure 3.2: SonarQube dashboard two weeks later scan results

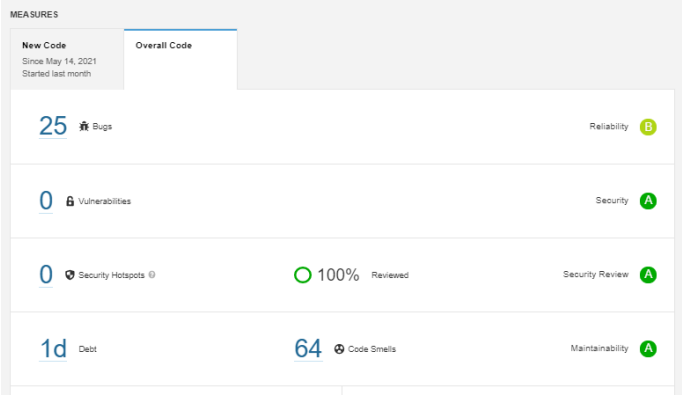

Figure 3.3: SonarQube dashboard four weeks later scan results

export, which allows users to store the search results, or information associated with it.

After a search is performed, and if some images were matched, then 2 buttons will be present on the page: `Export to excel` and `Export to zip`. Before clicking these buttons, the user can select some images by clicking on them. By doing this, and after they click on the button, only the selected images will be exported. An example of this is shown on Figure 3.6. If no image is selected, then all of the search results will, which we can see on Figure 3.7.

The two export options produce very different results. If the user is looking for a quick way to get a copy of the pictures, they can click the `Export to zip` button. The system will simply take the search images and put them in a zip file, that will be "downloaded" from the browser. The other button, `Export to excel` is for users that want a different kind of results. It does not export the images themselves, but rather the detailed information generated and stored during the processing phase. This includes the path to the image, its height and width, all the tags associated

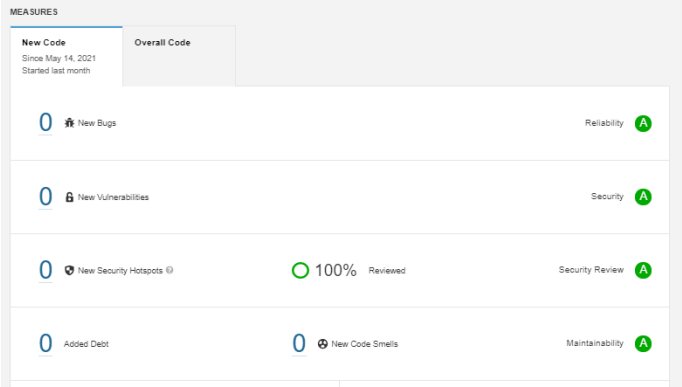

Figure 3.4: SonarQube dashboard five weeks later scan results

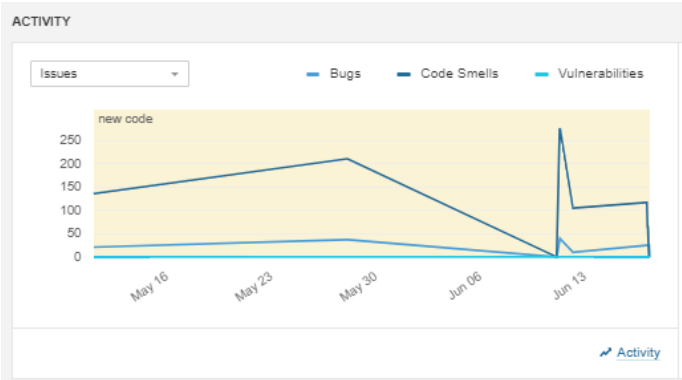

Figure 3.5: SonarQube scan results during the project's construction.

with it, the creation and insertion time, the people's names and the locations found in the picture. For every selected picture – or all if none is selected – in the results, this information will be included in a table in an excel file.

3.4.2 Filter Results

Because a simple search can return a large number of results, the ability to further reduce the search criteria to allow the user to get to what they want more directly is a necessity. In Figure 3.8 it is possible to see the interface where the user interacts with these filters.

The most simple thing we could do is to allow the user to activate/deactivate some components of our project since our search is based on tags from different sources. In particular, these filters are shown in Figure 3.8 in the checkboxes in the first two rows.

For example, if we disable the *Objects detected* checkbox and search for "chair",

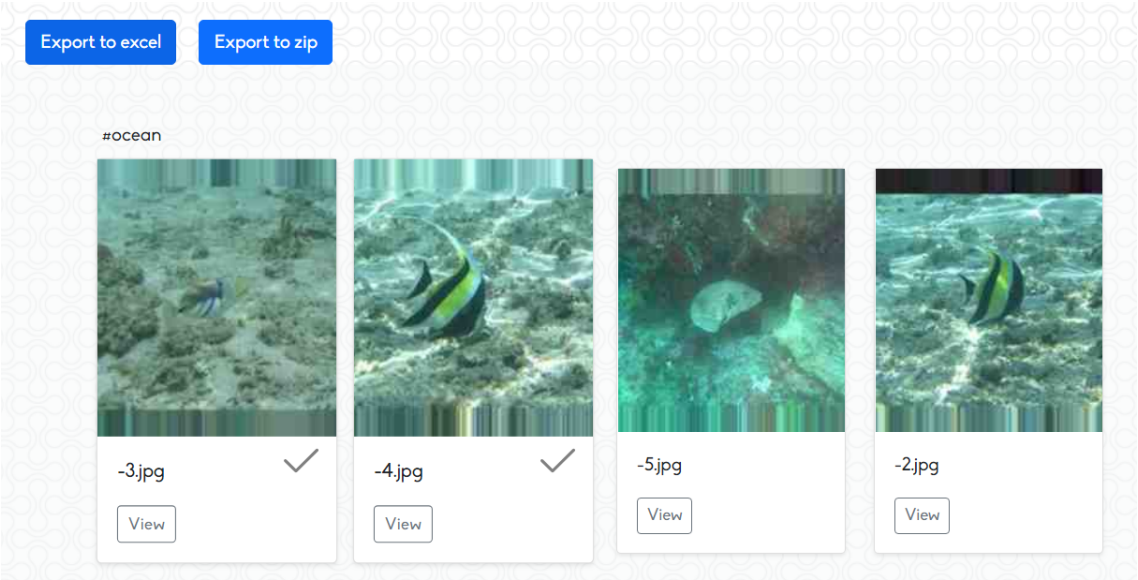

Figure 3.6: Some selected images to be exported

the images with "chair" as a tag produced by the object extraction algorithm will not appear in the results, except if those images have "chair" as a tag from any other source – provided that all the other checkboxes are ticked.

It is also possible to filter by the confidence – or score – of the algorithms, following a similar principle. If we increase the object extraction's minimum confidence to 0.5 and search for "chair", the images that have a tag "chair" with origin on the object extraction algorithm and with a score below 0.5 won't appear in the results. Again, if another algorithm identified a "chair" then the image would still appear – or if it is a manual tag.

In total, we can enable or disable 8 types of sources for our tags, 4 of which we can also specify the score range, just as explained. These last 4 are: the object extraction tags, the scenes/places, the identified people's names and the pet breeds tags. The sources from which we cannot specify a range are the text detection/OCR, the image metadata, the manual tags – added by the user –, and the folder name. The user is incapable of selecting a confidence range for these tags because these tags do not have scores associated with them.

The last row contains different kinds of filters. For example, the filter by insertion date of the images, where the user can specify a date range that will limit the results according to when the image was uploaded to the system. Similarly, there is a filter

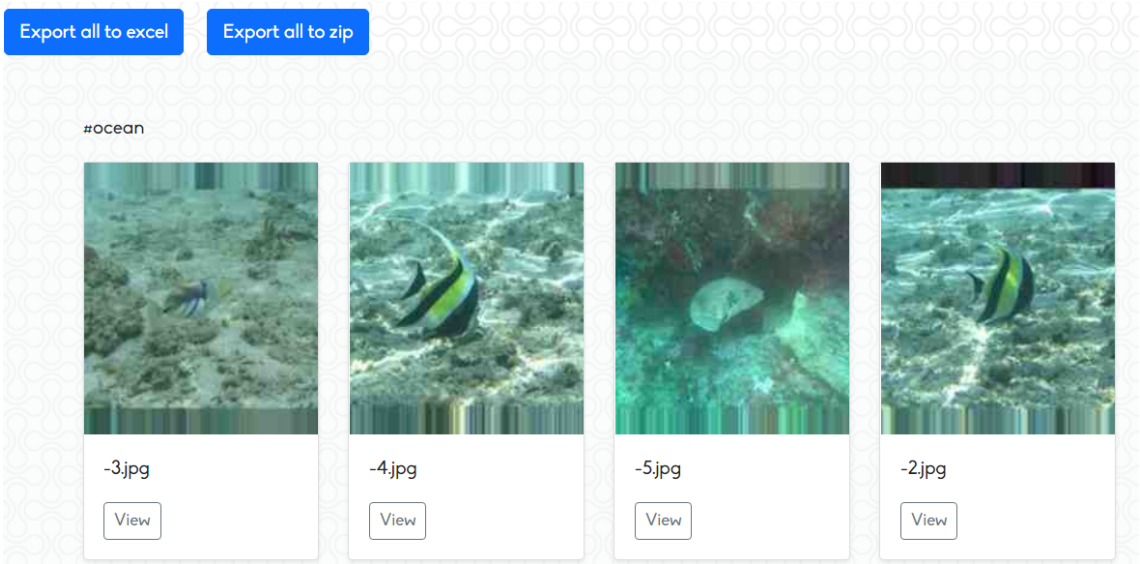

Figure 3.7: Export all if no selected images

by taken date, that verifies if the photo was taken within the date range. If the image does not contain metadata information, then when this filter is activated, the image will never appear.

To finish, we can also filter by the image size, allowing us to add or remove to the results images that are 'large", "medium" or "small". For these classifications, we used the number of pixels of the image to verify if it fitted the criteria.

3.4.3 Dashboard

In order to show some statistical data about the user's photos and the way our system behaves, we included a dashboard.

In particular, this page has information about the number of tags obtained from 4 of the methods that produce tags – object extraction, face recognition, places, and text extraction. Further down the page, there are two charts. One showing the 10 most common tags in the system, and other showing the number of tags in the system produced by each of the modules/components.

On the left side of the main dashboard page there is a sidebar with links to multiple pages, that show:

- All the tags created by the text extraction component, shown by alphabetical order.

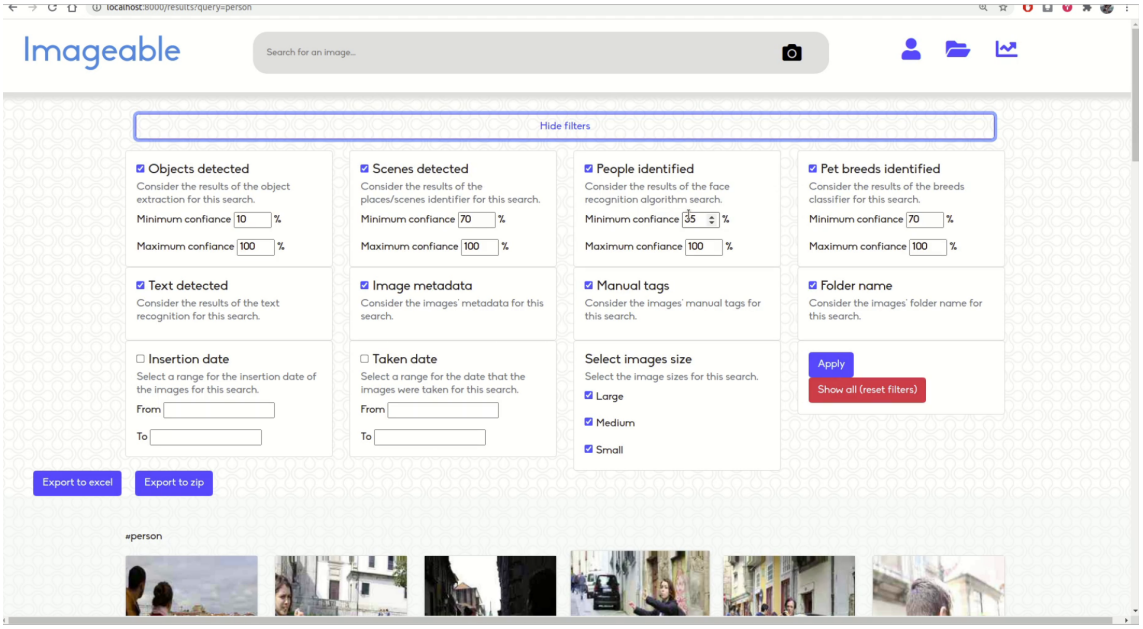

Figure 3.8: Filters available to filter searches.

- All the tags created by the scenes detection component, shown by alphabetical order.
- All the people's names that the user has validated in the system, shown by alphabetical order.
- All the objects detected by the system in the pictures, shown by alphabetical order.
- Two charts showing plots showing the number of pictures added to the system on each day, and also the number of images taking on each day.

All the tags shown in the first 4 points of the list above can be clicked, taking the user to a page showing the results of a search for that same clicked tag.

3.4.4 Dynamic Use of Threads

Because processing numerous pictures can take a long time, the user should feel free to use our program even while this process is happening. To tackle this problem, we automatically detect how many threads a user can hold – this is, the number of CPU cores – and create this number of thread instances in a thread pool.

Then a process to understand the best numbers of threads to use begins. This happens just before the system starts to analyse a folder, detecting how much free RAM and CPU usage the machine has at that moment. Then the following formulas are used sequentially to select the number of threads that will be used.

$$\#threads = ceil(min(\frac{freeRAM}{RAMUsedPerImage}, \frac{freeCPU}{CPUUsedPerImage}))$$

$$\#threads = min(\#threads, numThreadInstancesInPool)$$

This allows the performance of speed of the processing to be maximized while leaving enough resources freed so that the user can continue to use their machine as usual.

These threads were used mostly during the processing phase, but also when deleting the images and loading the different dependencies while the server is booting up, improving this process.

3.4.5 Implementation of the Pet Breeds Classifier

To develop even more our work, and to further differentiate our project from all the others available, we decided to implement a feature that allows the user to search by cats' and dogs' breeds. This component was different from the others we implemented since it used a Neural Network trained by one member of the team and one of our colleagues – namely, Mariana Santos and Pedro Silva – in the scope of another course in our degree. This model was developed for the first project of the course Tópicos de Aprendizagem Automática, lectured by professor Pétia Georgieva, and consists of a pet breed classifier that was trained on the dataset "Cats and Dogs Breeds Classification Oxford Dataset". Deep learning techniques were employed to develop it, and Keras was used for its implementation. It showed an accuracy of about 92% on the used test set of the mentioned dataset.

After the images went through the object extraction phase, if a dog or cat was identified, it was then fed into this model that would return the predicted race along with its respective confidence. If the value was higher than 0.7 – the

threshold we decided to use, discovered through various experiences – then the prediction would simply be discarded. If it was higher, then this data would be saved as a new tag, allowing the user to search for it.

3.5 Deployment

3.5.1 Creating an Executable

To make our application more easily available for everyone, we wanted to create an executable for it. The first tool we used to accomplish this was `pyinstaller` [14]. Because its utilization showed to be unreliable – showing many errors, such as not recognizing that certain packages were installed –, this tool was left aside.

After this first failed attempt, `cx_freeze` [7] was given an opportunity. However, this one also proved to be problematic, having trouble dealing with Django file paths – in particular, it was not able to find many Django files even though they were in the correct path and everything was set up correctly.

In the end, we weren't able to create an executable that is completely independent of python. As a poorer alternative, we created an executable that automatically initiates the Django server with a virtual environment Python.

4 Conclusion

Because of the large number of images stored in one's personal device nowadays, we wanted to create a web application that allowed the managing and recovering of images based on their content, using data extracted from them. The application we wanted to develop should work offline, so that it wouldn't suffer from privacy and security issues, much like the current solutions in the market. To this end, we implemented various algorithms and techniques that helped extract and retrieve information from images, creating a system capable of retrieving images from the device based on their content.

The usability test performed to our system revealed that the users were able to easily navigate within the system, without any help or previous experience. The test's results also revealed that all participants found the text and the overall information to be big enough and displayed adequately in the screen, which has always been a concern for the team. The main comments to the system praised the simple and intuitive interface and the feature to filter results. The feature to search by similar images has been considered very useful by some of the testers. The only negative aspect to be detected has been the system's performance, one of the main concerns of the team throughout the product's development and something we would have to improve as further work.

Working with several different algorithms and techniques has brought many challenges and although we are pleased with the outcome of the last months, we are aware that this project could be further enhanced. If we had more time, we would alter the page to edit people's names so that it performs well with a big dataset of images, we would enhance our solution by implementing more redundant algorithms, while allowing the filtering by such algorithms for academic purposes, we could also implement named-entity recognition for the Natural Language Processing module, and we would like to change how the executable runs, to be independent from Python or even create our own installer.

References

[1] ageitgey et al. *The world's simplest facial recognition api for Python and the command line - ageitgey/face$_recognition$*. [Online; accessed 23-June-2021]. 2021. URL: https://github.com/ageitgey/face_recognition.

[2] AnandhJagadeesan. *Text Detection and Extraction using OpenCV and OCR - GeeksforGeeks*. [Online; accessed 23-June-2021]. 2020. URL: https://www.geeksforgeeks.org/text-detection-and-extraction-using-opencv-and-ocr/.

[3] aws.amazon. *Amazon Rekognition – Video e Image – AWS*. [Online; accessed 23-June-2021]. 2021. URL: https://aws.amazon.com/pt/rekognition/?blog-cards.sort-by=item.additionalFields.createdDate&blog-cards.sort-order=desc.

[4] cloud.google. *Vision AI | Derive Image Insights via ML | Cloud Vision API*. [Online; accessed 23-June-2021]. 2021. URL: https://cloud.google.com/vision.

[5] cocodataset. *COCO - Common Objects in Context*. [Online; accessed 23-June-2021]. 2021. URL: https://cocodataset.org/#download.

[6] Harry Domanski. *Google Photos now allows you to search for text in your pics*. [Online; accessed 23-June-2021]. 2019. URL: https://www.techradar.com/news/google-photos-now-allows-you-to-search-for-text-in-your-pics.

[7] Marcelo Duarte. *Welcome to cx$_Freeze'sdocumentation!—cx_Freeze6.7b4documentation$*. [Online; accessed 23-June-2021]. 2021. URL: https://cx-freeze.readthedocs.io/en/latest/.

[8] IBM Cloud Education. *What is Natural Language Processing?* [Online; accessed 23-June-2021]. 2020. URL: https://www.ibm.com/cloud/learn/natural-language-processing.

[9] Elastic. *What is Elasticsearch?* [Online; accessed 23-June-2021]. 2021. URL: https://www.elastic.co/what-is/elasticsearch.

[10] guru99. *POS Tagging with NLTK and Chunking in NLP [EXAMPLES]*. [Online; accessed 23-June-2021]. 2021. URL: https://www.guru99.com/pos-tagging-chunking-nltk.html.

[11] Samuel Hoffstaetter et al. *pytesseract: PyPi*. [Online; accessed 23-June-2021]. 2020. URL: https://pypi.org/project/pytesseract/.

[12] nltk. *Natural Language Toolkit — NLTK 3.6.2 documentation*. [Online; accessed 23-June-2021]. 2020. URL: https://www.nltk.org/#.

[13] Google Photos. *Search by people, things places in your photos - Android - Google Photos Help*. [Online; accessed 23-June-2021]. 2021. URL: https://support.google.com/photos/answer/6128838?co=GENIE.Platform%3DAndroid&hl=en.

[14] pyinstaller. *PyInstaller Quickstart — PyInstaller bundles Python applications*. [Online; accessed 23-June-2021]. 2021. URL: https://www.pyinstaller.org/.

[15] Adrian Rosebrock. *OpenCV Text Detection (EAST text detector) - PyImageSearch*. [Online; accessed 23-June-2021]. 2018. URL: https://www.pyimagesearch.com/2018/08/20/opencv-text-detection-east-text-detector/.

[16] stweil et al. *tesseract-ocr/tesseract: Tesseract Open Source OCR Engine (main repository)*. [Online; accessed 23-June-2021]. 2021. URL: https://github.com/tesseract-ocr/tesseract.

[17] thampiman. *thampiman/reverse-geocoder: A fast, offline reverse geocoder in Python*. [Online; accessed 23-June-2021]. 2016. URL: https://github.com/thampiman/reverse-geocoder.

[18] Tyler N. Thieding. *exif · PyPI*. [Online; accessed 23-June-2021]. 2021. URL: https://pypi.org/project/exif/.

[19] ultralytics and Glenn Jocher. *YOLOv5 Pytorch*. [Online; accessed 23-June-2021]. 2021. URL: https://pytorch.org/hub/ultralytics_yolov5/.

[20] *What is an index in Elasticsearch*. [Online; accessed 23-June-2021]. 2013. URL: https://stackoverflow.com/questions/15025876/what-is-an-index-in-elasticsearch/15026433.

[21] Filip Zelic and Anuj Sable. *[Tutorial] OCR in Python with Tesseract, OpenCV and Pytesseract*. [Online; accessed 23-June-2021]. 2021. URL: https://nanonets.com/blog/ocr-with-tesseract/.

[22] zhoubolei et al. *CSAILVision/places365: The Places365-CNNs for Scene Classification*. [Online; accessed 23-June-2021]. 2020. URL: https://github.com/CSAILVision/places365.