BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INTELLIGENT SYSTEMS
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

DETECTION OF FRAUDULENT REAL ESTATE ADVERTISEMENTS

DETEKCE PODVODNÝCH INZERÁTŮ NEMOVITOSTÍ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **TOMÁŠ FRÁTRIK**
AUTOR PRÁCE

SUPERVISOR **Ing. ANTON FIRC,**
VEDOUCÍ PRÁCE

BRNO 2023

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

152809

Institut:	Department of Intelligent Systems (DITS)
Student:	**Frátrik Tomáš**
Programme:	Information Technology
Title:	**Detection of fraudulent real estate advertisements**
Category:	Security
Academic year:	2023/24

Assignment:

1. Study automated fraud detection in the online environment focusing on fraudulent advertising.
2. Learn about image content comparison methods.
3. Design a solution to detect fraudulent advertisements based on the search for similar images. The solution will be functional for at least two property ad portals and extendable to others.
4. Implement the proposed solution.
5. Test the solution's functionality, reliability and usability and discuss possible future improvements.

Literature:

- F. Kanei, D. Chiba, K. Hato and M. Akiyama, "Precise and Robust Detection of Advertising Fraud," *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, Milwaukee, WI, USA, 2019, pp. 776-785, doi: 10.1109/COMPSAC.2019.00115.
- Fumihiro KANEI, Daiki CHIBA, Kunio HATO, Katsunari YOSHIOKA, Tsutomu MATSUMOTO, Mitsuaki AKIYAMA, Detecting and Understanding Online Advertising Fraud in the Wild, IEICE Transactions on Information and Systems, 2020, Volume E103.D, Issue 7, Pages 1512-1523, Released on J-STAGE July 01, 2020, Online ISSN 1745-1361, Print ISSN 0916-8532, https://doi.org/10.1587/transinf.2019ICP0008, https://www.jstage.jst.go.jp/article/transinf/E103.D/7/E103.D_2019ICP0008/_article/-char/en
- N. Vasconcelos and A. Lippman, "A unifying view of image similarity," *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, Barcelona, Spain, 2000, pp. 38-41 vol.1, doi: 10.1109/ICPR.2000.905271.
- Russakoff, D.B., Tomasi, C., Rohlfing, T., Maurer, C.R. (2004). Image Similarity Using Mutual Information of Regions. In: Pajdla, T., Matas, J. (eds) Computer Vision - ECCV 2004. ECCV 2004. Lecture Notes in Computer Science, vol 3023. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-24672-5_47

Requirements for the semestral defence:
Points 1 to 3

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Firc Anton, Ing.**
Head of Department:	Hanáček Petr, doc. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	9.5.2024
Approval date:	26.4.2024

Abstract

The advent of the Internet has led to a rise in the number of Internet users. Internet ad fraud refers to a form of online crime where individuals or businesses publish deceptive real estate advertisements to defraud unsuspecting victims. The purpose of this thesis is to provide an overview of real estate ad frauds found on websites that allow anyone to post real estate ads on their websites. And also provide a tool, that determines the probability of fraudulent advertisement of real estate chosen by user from supported web portals.

Abstrakt

Nástup internetu viedol k nárastu počtu používateľov internetu. Internetové reklamné podvody predstavujú formu online zločinu, pri ktorej jednotlivci alebo firmy zverejňujú klamlivé inzeráty na nehnuteľnosti, aby oklamali nič netušiace obete. Účelom tohto príspevku je poskytnúť prehľad podvodov s inzerciou nehnuteľností, ktoré sa vyskytujú na webových stránkach, ktoré umožňujú komukoľvek uverejňovať inzeráty s nehnuteľnosťami na ich webových stránkach. A tiež poskytnúť nástroj, ktorý zisťuje pravdepodobnosť podvodnej inzercie nehnuteľnosti vybranej užívateľom z podporovaných webových portálov.

Keywords

advertisement, fraud, real estate, reverse image search engine, API, similarity measures

Klíčová slova

reklama, podvod, nehnuteľnosti, spätný vyhľadávač obrázkov, API, merania podobnosti

Reference

FRÁTRIK, Tomáš. *Detection of fraudulent real estate advertisements.* Brno, 2023. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Anton Firc,

Rozšířený abstrakt

Zvýšenie pokroku, prístupu a dostupnosti internetovej technológie v poslednom desaťročí intenzifikovalo rast internetových používateľov. To z internetovej reklamy urobilo populárne miesto pre mnohé spoločnosti na propagáciu svojich produktov a služieb. Dnes je internetová reklama jedným z najdôležitejších zdrojov príjmu, ktorý ovplyvňuje ekonomiku mnohých veľkých podnikov.

Avšak napriek mnohým výhodám nie je internetová reklama bez svojich výziev. Jedným z najnaliehavejších problémov, s ktorými sa táto odvetvie stretáva, je reklamný podvod. Reklamný podvod označuje podvodné aktivity, ktoré manipulujú alebo oklamávajú systém internetovej reklamy za osobný zisk. Tento stav predstavuje vážnu hrozbu pre ďalší rast a integritu internetovej reklamy.

Hoci môže reklamný podvod nastať v rôznych sektoroch, podvod s nehnuteľnosťami sa stal rastúcim problémom v digitálnom veku. Keď ľudia začnú hľadať domy na prenájom alebo kúpu, internet je prvým miestom, kde začnú svoje vyhľadávanie. A, rovnako ako v prípade iných typov klasifikovaných webových stránok, podvodníci tiež využívajú inzeráty s nehnuteľnosťami na získanie peňazí od potenciálnych kupujúcich a nájomcov. Problém spočíva v tom, že podobne ako v prípade mnohých podvodov na online trhoch, nie sú vždy ľahko rozpoznateľné. Tieto podvodné aktivity zahŕňajú zverejňovanie zavádzajúcich alebo falošných inzerátov s nehnuteľnosťami s cieľom oklamať a podviesť nedbanlivých jednotlivcov alebo podniky. Takéto podvody môžu mať miesto na rôznych online platformách, vrátane klasifikovaných webových stránok, sociálnych médií a online trhovísk.

V našej práci sme mali za cieľ analyzovať podvody s inzerciou nehnuteľností, ktoré sa nachádzajú predovšetkým na portáloch s otvoreným prístupom, a vytvoriť nástroj na ich detekciu na základe podobnosti obrázkov. Na tento účel sme navrhli nástroj vo forme webovej aplikácie.

Na záver budeme diskutovať o tom, ako presne dokáže náš nástroj detekovať podvody. Zistili sme, že náš nástroj dokáže detekovať podvody, aspoň v niektorých scenároch. Avšak existuje mnoho scenárov, kde podvod nie je možné odhaliť bez zásahu používateľa. Používateľovi sme poskytli nájdené inzeráty a ak sme nedokázali detekovať podvod, aspoň sme mu poskytli nástroje na ďalšie preskúmanie. Veríme, že žiadny nástroj nemôže detegovať podvody s 100% presnosťou, ale veríme, že sa môžeme vždy priblížiť k tomuto číslu, aj s naším nástrojom, a preto sme sa snažili vytvoriť škálovateľný a modulárny nástroj, kde môže byť implementovaná ďalšia analýza inzercie.

Detection of fraudulent real estate advertisements

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. Anton Firc I have listed all the literary sources, publications, and other sources, which were used during the preparation of this thesis.

.....
Tomáš Frátrik
May 7, 2024

Acknowledgements

I would like to thank my supervisor, Ing. Anton Firc, for his guidance and ideas. I would also like to thank all the people who participated in testing the application.

Contents

List of Figures

Chapter 1

Introduction

Increasing advancement, access, and availability of Internet Technology have intensified the growth of Internet users over the past decade. This has made online advertising a popular place for many companies to market their products and services. Today, online advertising is one of the most important sources of revenue that impact the economy of many large enterprises. [13]

However, despite its many benefits, online advertising is not without its challenges. One of the most pressing issues facing this industry is advertising fraud. Advertisement fraud refers to fraudulent activities that manipulate or deceive the online advertising system for personal gain. This poses a serious threat to the continued growth and integrity of online advertising. Although ad fraud can occur in various sectors, real estate ad fraud has become a growing concern in the digital era. [13]

When people start looking for houses to rent or buy, the Internet is the first place they begin their search. And, as with other types of classified sites, scammers are also using real estate listings to extract money from potential buyers and renters. The problem, as with many online marketplace scams, is that they are not always that easy to detect. [32]

These fraudulent activities involve the posting of misleading or false real estate advertisements to deceive and defraud unsuspecting individuals or businesses. Such scams can take place on a wide range of online platforms, including classified websites, social media platforms, and online marketplaces.[13]

In our thesis, our objective was to analyze the real estate advertising fraud found primarily on open-access portals and create a tool to detect them based on the similarity of the images. For this, we have proposed a tool in the form of a Web application. This tool was implemented in a scalable way in which we can add modules for additional analysis of advertisements.

The rest of the thesis is organized as follows. Chapter 2 describes the problem of fraudulent advertisements and tactics used by those who make them. In Chapter 3 we take a look at some reverse image search engines and pick the best one that suits our needs. The same choice based on our needs for our tool is made in Chapter 4 where we take a look at some similarity measures. The design of our tool is discussed in Chapter 5. Implementation of our tool starts in Chapter 6. In Chapter 7 we make an experimental analysis of our tool. Finally, Chapter 8 concludes the thesis.

Chapter 2

Fraudulent Advertisement

As the online advertising market has grown, advertising fraud has become a serious problem. Countermeasures against advertisement fraud are evaded because they rely on noticeable features (e.g., burstiness of ad requests) that attackers can easily change.[13]

Over the past decade, researchers have been investigating the mechanism of ad fraud by analyzing network traffic originating from online ads and client applications (e.g., browser extensions and mobile applications) that display advertisements to users. Some researchers also focused on detecting advertising fraud. They proposed anomaly detection of click-spam and correlation analysis to detect crowd-sourced ad fraud.[25] There are various types of advertisement fraud in the wild. The most sophisticated attack is a distributed attack that abuses many clients and publisher websites. In this case, malicious activities launched from various clients and publisher websites are combined into legitimate activities. This type of attack cannot be detected using a simple anomaly-based method that leverages the burstiness of advertisement requests; thus, such a method causes non-negligible false positives and false negatives.[25]

2.1 Real Estate Advertisement Frauds

Real estate scams have been steadily on the rise recently, unfortunately with the help of technology. It has possibly never been easier for a con artist to post an advertisement online ostensibly seeking to purchase or sell a home but searching for illegally obtained funds.

You would like to think that the average web user is sufficiently well-informed to recognize a scam when he or she encounters one, but it is not true. Scammers would not continue to post these advertisements if they did not work to some extent, so fraudulent listings appear everywhere. The typical scam works like this: Con people find online property listings on sites like Craigslist or Zillow, and then use the same photos and information to create their real estate advertisement for a property they don't own. They will price it attractively, remove the address and contact information, and wait for someone to message them through the site's messaging portal.

These scammers usually make their money by requesting a payment for something seemingly minor, like a credit check or a security deposit. Too many people give in without recognizing their potential mistakes. In most cases, there is nothing you can do to get your money back because the scammer cannot be tracked, and/or the money is wired to an overseas account without any chance of a refund.[27]

2.2 How to Spot Potential Fraud

When buying real estate, it is important to keep an eye out for possible fraud. Things like super-cheap prices, poor-quality photos, and promises that sound too good to be true should raise some suspicion. Here, we list some common signs that could indicate potential fraud when buying real estate.

The Price is Too Good to Be True

In the current national rental economy, rent prices have been climbing. The housing market is also booming, which means that prices are considerably higher than they were a few years ago. Competition keeps the market alive, and real estate investors will not make much of a profit if they do not keep their prices competitive.[27] If the price listed for a property seems significantly lower than other similar properties in the area, it could be a red flag. Fraudsters often use attractive prices to lure unsuspecting victims.

Poor Quality Photos

If the property photos appear blurry or seem to have been taken from different sources, it could indicate that the listing is not genuine. Also, if there is only one photo of the property, or you can see that there has been tempered with the photo, it can indicate fraud. Be wary of listings with only exterior shots or pictures that do not match the description.[27]

Unrealistic Claims

Be cautious of ads that make exaggerated claims or promises, such as guaranteed high rental income, extremely low maintenance fees or unrealistic amenities. Fraudsters may use these tactics to attract attention.[27]

The Email Sounds Strange

Some listings hide the email address when you send a message, so you might not be able to see the address if you respond to the listing. But if you are allowed to see the email, evaluate it for authenticity. First of all, scammers usually use free email servers such as Hotmail or Yahoo. In addition, they will often go by a series of random letters to make them less easily traceable. So, if you see that the poster has an email like axoiehips@hotmail.com, that is one sign to be wary of.[27]

Requests for Payment in Advance

Be cautious if the listing requests upfront payment or personal financial information without any opportunity to view the property or meet in person. Scammers often use this tactic to extract money or sensitive data.

The Seller Won't Show You the Property

Naturally, scammers will not let you see a property they do not own. If you ask to see the property the next time you are in town and they say it is impossible, it is probably a fake listing. The ostensible seller or broker might even use a venerable excuse, such as being out of town or visiting a sick relative. But most home sellers will make time for

people who are interested in the property. If possible, never buy a home that you have not walked through first.[27]

Pressure to Act Quickly

If the ad creates a sense of urgency, such as stating that the property is in high demand or pressuring you to make a quick decision, it might be a sign of fraud. Take your time and do thorough research before committing to anything.

There's No Address

Sometimes the absence of an address may be an oversight of the seller. But more often it is an indication of foul play. You might ask for an address, and the seller will respond by saying that he does not want to send it to you for security purposes. At other times, if you insist on an address, the person may send you a fake one to appease your demands. However, if you plug it into Google Maps, you will find that the random address does not belong to the house you were considering.[27]

2.2.1 Tips For Safe Online Transactions

While our thesis does not go into this topic in-depth, we offer two essential points for safe online transactions.

- Use secure payment methods, such as escrow services[1] or online payment platforms that offer buyer protection. Advise against making payments through wire transfers or sending money directly to individuals without any guarantees or safeguards.
- Verify the legitimacy of a listing before proceeding with any transactions. Verify points from Section 2.2.

2.3 Automatic Detection

The manual search for fraudulent real estate advertisements can be a burden, often requiring extensive human intervention. Going through all the points of Section 2.2 would be a time-consuming task. With the exponential growth of online real estate platforms, it has become increasingly challenging to manually identify and flag fraudulent listings effectively. Automatic detection of fraudulent real estate advertisements offers a solution to this burden.

Daiki, Kunio, Yoshioka, and Matsumoto propose [18] method for precisely detecting fraudulent advertisement requests observed within an advertisement network. The proposed method is a machine-learning-based method that leverages features extracted from advertisement requests and classifies them as either benign or malicious. To design robust features against attacker evasion, they introduce the assumption that attackers cannot know the statistics of ad requests from legitimate users because they cannot observe advertisement requests from the standpoint of the ad network. Based on this assumption, they statistically calculate the value of features from both client and publisher information and

[1] Escrow is a legal concept describing a financial agreement whereby an asset or money is held by a third party on behalf of two other parties that are in the process of completing a transaction.[15]

use them to distinguish advertisement frauds from a large number of legitimate advertisement requests.[18]
To accomplish the automatic detection of fraudulent real estate advertisements, our work will involve the development of a tool that should automatically determine the probability of fraudulation of a given advertisement based primarily on the origin of the country of the advertisement.

Chapter 3

Reverse Image Search API

The big part of the tool will be taking the provided image by the user and finding sources or similar images on the Internet. The reason why we need this is given in the Chapter 5. But on the Internet, there are almost a trillion photos present. For this problem, we need to use some of the provided reverse image search engines.

3.1 What is a Reverse Search

A reverse search is useful if you have ever tried to find information about an individual or business. A reverse search can involve a phone number, email address, physical address, name, etc. Running a reverse search means searching for something based on a particular piece of information. The idea is to use those data to find more information about what it is you are looking for. For example, if you do a reverse image search, you are running the search using a picture instead of text. A reverse phone number search is the search using a phone number instead of something else, such as a name. Along the same lines, a reverse email search, reverse address search, etc., uses an email address or physical address as the search query. [33]

3.1.1 Visual Search Engine

A visual search engine is a search engine designed to search for information on the World Wide Web through a reverse image search. Information may consist of web pages, locations, other images, and other types of documents. This type of search engine is mostly used to search the mobile Internet through an image of an unknown object (unknown search query). Examples are buildings in a foreign city. These search engines often use techniques for Content-Based Image Retrieval.

A visual search engine searches for images and patterns based on an algorithm that it could recognize and gives relative information based on the selective or applied pattern-matching technique. [35]

3.2 Reverse Search Engines

There are not many robust reverse image search engines available on the Internet due to the high costs or the limit of depth they search for. Some we cannot properly test because of their paid plans, like TinEye. Most of them do not provide any public APIs. Google

Search by Image, Bing Search, and Yandex Search are the most popular [16]. To find their relevance for our search needs, we need to test them.

3.3 Testing Search Engines

We will use an image [17] taken from one of our supported portals in Subsection 6.2.7.

3.3.1 Yandex

Yandex Reverse Image Search is a search engine feature that allows you to discover visually similar images by inputting an existing image. This tool is essential for several tasks, such as finding similar images and recognizing unidentified objects. [8]

The result is scrollable, but there were no really similar images. The similarity is taken to the depth of a house of yellow and pink colors.

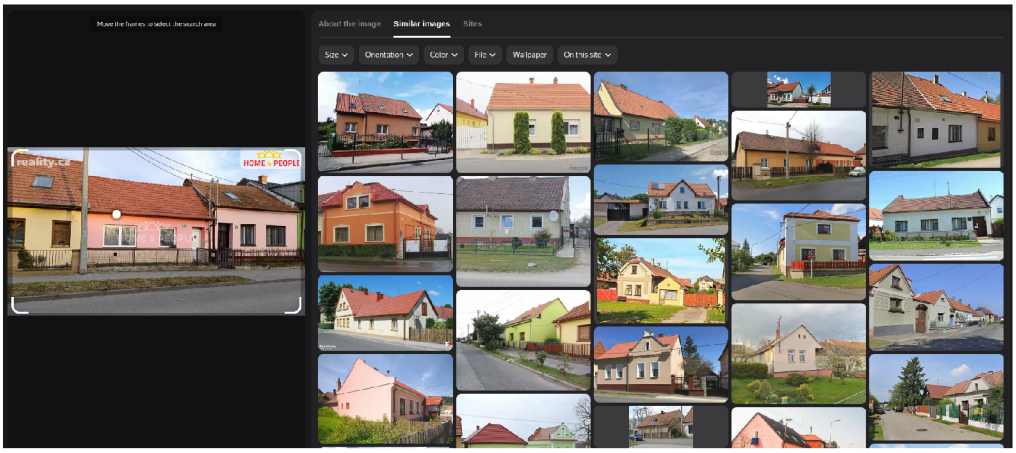

Figure 3.1: Yandex search engine result

3.3.2 Bing

Bing is an Internet search engine that is powered by Microsoft. As a search engine, it helps users find different types of information on the Internet and works the same way as other search engines. [9]

Again the result is scrollable, but the results are similar to Yandexe's. There are visually similar houses, but there is not the one we are looking for.

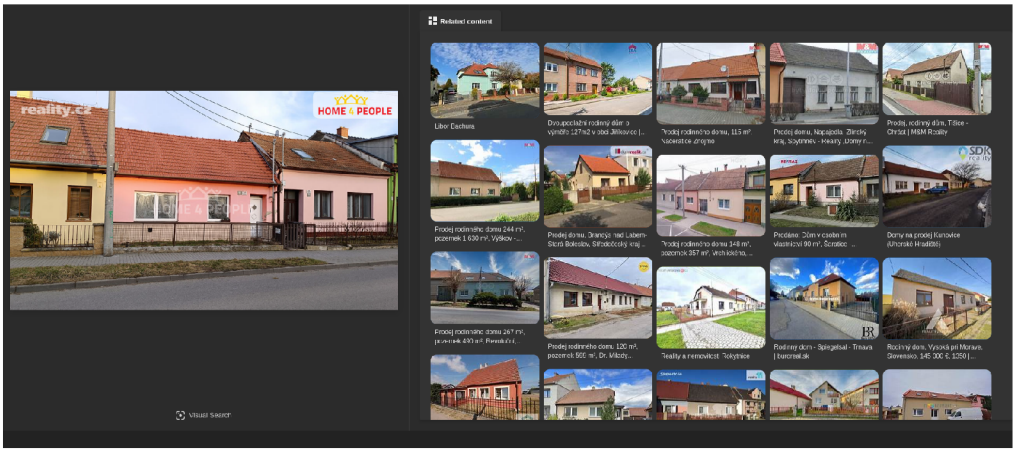

Figure 3.2: Bing search engine result

3.3.3 Google

First, we are shown similar results. From our observation, 60 results are shown. And, as we can see from the figure 3.3, the first result of the search is the house we searched for. But the more interesting and important thing is that Google provides us with the button 'Find image source'. When we go to this page we will see the exact images or even sometimes if found from different angles. The number of results there varies. From zero up to 20 max most of the time. These results can be seen in the figure 3.4. It also provides us with other information, such as the website name and direct link to the page where that image was taken. But still results on the `similar` tab are still important to us. These results can still find the same photos as `source` page images, or they can even find the property in a different state (e.g., differently painted).

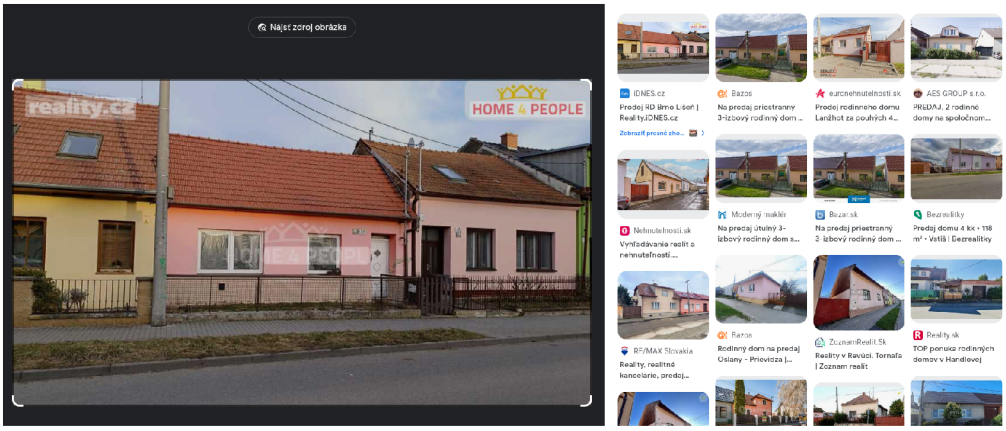

Figure 3.3: Google search engine result of similar page

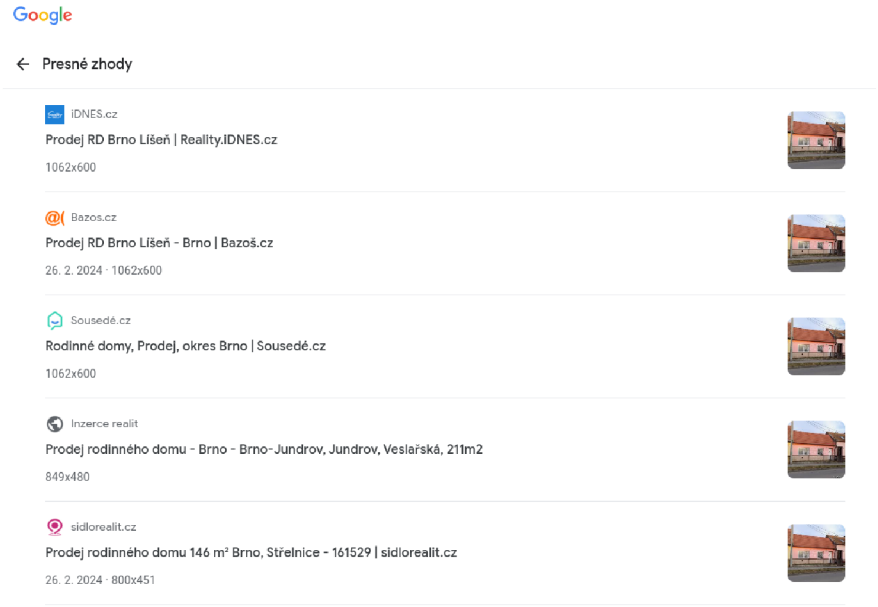

Figure 3.4: Google search engine result of source page

3.3.4 Google Search By Image

Since this engine provides us with found sources of images and similar results, we have chosen Google Search By Image as our search engine. One major issue is the lack of an official API, which prompted us to create our implementation. This API utilizes Google's reverse image engine, by using the Python library *selemnium* [6] as an automatic bot, to import the image to Google's reverse image search engine, and `BeautifulSoup4` to scrape those results that the Google engine provided. This provides us with a cheap method with fairly good results. More detailed information on this implementation can be found in Chapter 6 in Section 6.1.

How Does It Work

Google's reverse image search, officially called Google Search by Image, is a service provided by Google that allows a user to search for images using an image as the starting point, rather than a written or spoken search query. [22]

You simply upload an image or provide a link to an image that can be found online, and Google will try to find related images. These will typically be similar images or a mix of similar images and exact copies.[22]

Search by image analyzes an image to find its most distinctive colors, points, lines, and textures; it uses these features to generate a query; this query is sent to Google's backend for matching against billions of images. Their algorithm returns matching and visually similar images. [23]

Chapter 4

Image Similarity

With our chosen reverse image search described in Section 3.3.4, we obtain a lot of images. As mentioned in Section 3.2 there should be 60 results on the `similar` page. In Section 6.4 we received feedback that there are too many results. To address this, we will filter some images based on their image similarity. How we implemented this can be found in Section 6.3.

In a world overwhelmed by images, the ability to measure and quantify the similarity between images has become a critical task. Whether for image retrieval, content recommendation, or visual search, image similarity approaches play a pivotal role in modern applications. [28]

Image similarity measures play an important role in image fusion algorithms and applications, such as duplicate product detection, image clustering, visual search, change detection, quality evaluation, and recommendation tasks. These measures essentially quantify the degree of visual and semantic similarity of a pair of images. In other words, they compare two images and return a value that tells you how visually similar they are.[28]

The input images may be images obtained from the same scene or object taken from different environmental conditions, angles, lighting conditions, or edited transformations of the same image. In most use cases, the degree of similarity between the images is significant. In other use cases, the aim is to consider whether two images belong to the same category.[28]

The resulting measure can be different depending on the types of features. Typically, the feature space is assumed to be **Euclidean**[1]

In image comparison, we have two input images I_A and I_B, and our goal is to measure their similarity $S(I_A, I_B)$. First, we must realize that the concept of similarity is not strictly defined and can be interpreted in many ways. Specifically, two images I_A and I_B can be considered similar if [14]:

- They differ only in terms of contrast, brightness, and rotation.[14]
- They are semantically identical, meaning that they depict the same objects.[14]

[1] Euclidean is a space in any finite number of dimensions (originally three-dimensional), in which points are defined by coordinates (one for each dimension).[28]

4.1 Similarity Measures

Given a pair of images each described by a set of characteristics, image similarity is defined by comparing the set of characteristics based on a similarity function[29]. There are many measures to choose from. It is almost impossible to test all of them under different conditions. But in general, there are two larger groups, global image comparison, where we have popular measurements such as `MSE, PSNR, SSIM`, and local feature-based techniques, where we have again chosen popular options such as `SIFT, ORB`. Now we will introduce them, and in the next Section 4.2 we will test them.

4.1.1 Mean Squared Error (MSE)

A naive approach would be to use a metric that takes as input the raw pixels of the images and outputs their similarity. In this case, we can use the classical mean square error to calculate the mean value of the square of the differences between the pixels of the two images:[28]

$$\mathbf{MSE}(\mathbf{I_A}, \mathbf{I_B}) = \frac{1}{m * n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |I_A(i,j) - I_B(i,j)|^2$$

We can assume that the lower the MSE value, the higher the similarity. Despite its speed and simplicity, this method has many problems. The large Euclidean distance [2] between the pixel intensities does not necessarily mean that the image content is different.[28]

Peak Signal-to-Noise Ratio (PSNR)

Peak Signal-to-Noise Ratio (PSNR) measures the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. PSNR is usually expressed in terms of the logarithmic decibel scale. Typical values for PSNR in lossy image and video compression in 8-bits are between 30 and 50 dB, where higher values are more desirable. For 16-bit data, typical values for PSNR are between 60 and 80 dB.[28] We can use `MSE` to compute `PSNR`.

$$\mathbf{PSNR}(\mathbf{I_A}, \mathbf{I_B}) = 10 \log_{10} \frac{R^2}{MSE}$$

In the PSNR calculation, R is the maximum possible pixel value of the image. Please note that the logarithmic transform is applied to have the value in decibels.[28]

4.1.2 Structural Similar Index Measure (SSIM)

SSIM is based on visible structures in the image. In other words, SSIM measures the perceptual difference between two similar images. The algorithm does not judge which of the two is better. However, that can be inferred from knowing which is the original image and which has been subjected to additional processing, such as data compression. The SSIM value is between -1 and 1 with 1 indicating perfect structural similarity. SSIM examines similarities between their luminance, contrast, and structure.[28]
Given two N-dimensional signals/images (or corresponding patches), $x = (x_1, \cdots, x_N)$

[2] The Euclidean distance formula gives the distance between two points (or) the straight line distance[26].

and $y = (y_1, \cdots, y_N)$,. The ϵ_i are numerical stability coefficients to prevent zero nominators. Additionally, they can model the deviation from Weber's law as intensities approach 0.[21]

$$S(x,y) = S_1(x,y)S_2(x,y) = \left[\frac{2\bar{x}\bar{y} + \epsilon_1}{\bar{x}^2 + \bar{y}^2 + \epsilon_1}\right]\left[\frac{2s_{xy} + \epsilon_2}{s_x^2 + s_y^2 + \epsilon_2}\right]$$

Note: $-1 \leq S(x,y) \leq 1$ and $S(x,y) = 1$ if $x = y$ If $\bar{x} = \bar{y}$ then $S_1(x,y) = 1$

4.1.3 Scale-Invariant Feature Transform (SIFT)

The scale-invariant feature transform (SIFT) is an algorithm used to detect and describe local features in digital images. It locates certain key points and then furnishes them with quantitative information (so-called descriptors), which can, for example, be used for object recognition. The descriptors are supposed to be invariant against various transformations which might make images look different although they represent the same object(s). [20]

4.1.4 ORB

The algorithm used for the detection of features from the given image along with the orientation and descriptors for the image is called the ORB algorithm and it is a combination of the FAST keypoint detector and the BRIEF descriptor where the rotation performed by the BRIEF is rotated by the ORB according to the orientation of the key point to find the rotation matrix and in terms of computation cost, performance, extraction of features, and patents.[28]

4.2 Testing Measures

Just to be sure, we will use real data on the properties of the portals we support that can be found in Subsection 6.2.7. `SSIM` and `SIFT` have normalized the result numbers from -1 to 1 with 1 being 100% similar. `MSE` has 100% similarity with result of `0`, `PSNR` and `ORB` have higher similarity with a larger result number. When we get the result `100` of the orb, we don't know how good or bad that result is unless we compare it to other images. For this, with a comparison of house, we will also try to compare it with house that is not similar at all, at least in terms of houses. And to have some correlation, we will do this test twice. The result images of the real estate are also often taken from different angles, at lower resolution, or both. So, for each real estate, we will have one original photo, one with a much lower resolution, one taken from a different angle, and one taken from a different angle with low resolution.

We will be testing these measurements using Python's[34] *cv2*[4] and *skimage*[5] libraries, which make implementing these measures easier.

4.2.1 Chosen Images

These are the images that we have chosen.

Figure 4.1: House 1 [12]

Figure 4.2: House 1 (different angle)

Figure 4.3: House 2 [11]

Figure 4.4: House 2 (different angle)

Figure 4.5: House 3 [10]

Lower-resolution images are the same images but just 40% of the original resolution.

4.3 Running Tests

Image 1	**Image 2**	**MSE**	**PSNR**	**SSIM**	**SIFT**	**ORB**
House 1	House 1 (low resolution)	34.34	29.76	0.92	0.55	153
House 1	House 1 (different angle)	102.86	10.32	0.18	0.02	132
House 1	House 1 (diff. angle, low res.)	102.08	10.65	0.13	0.04	125
House 1	House 3	107.62	10.48	0.18	0.03	129
House 1	House 3 (low resolution)	107.21	10.83	0.12	0.03	112

Table 4.1: Image Comparison Metrics for House 1

Table 4.2: Image Comparison Metrics for House 2

Image 1	**Image 2**	**MSE**	**PSNR**	**SSIM**	**SIFT**	**ORB**
House 2	House 2 (low resolution)	40.40	27.60	0.86	0.50	114
House 2	House 2 (different angle)	100.35	11.91	0.24	0.04	127
House 2	House 2 (diff. angle, low res.)	100.40	12.22	0.24	0.05	121
House 2	House 3	107.42	11.08	0.19	0.02	131
House 2	House 3 (low resolution)	107.54	11.49	0.15	0.02	142

4.4 Test Results

From the data shown, we can see that `ORB` considered different houses to be too similar and performed poorly with the same house with low resolution. `SIFT` was overall bad, and seeing that `SSIM` was much better with lower resolution, we can move on to another measure. For others, we can see that they did similar. They had a good score for the low-resolution house and a bad score for the same image of a house taken from a different angle. But with `SSIM` this is not at least a relative value, but a normalized value, and we do not need another image to judge its result. For example, `PSNR's` result similarity number can go to infinity. And since Google processes images and, therefore, they are of lower resolution, `SSIM` seems perfect for that. As we can see, they all performed badly with different angles images. This will not be a really big obstacle since we will use a trick for this. For this, we will use images from the `source` tab, more on this in Section 6.3.

Chapter 5

Design

Our main goal of this thesis is to design a tool that is focused on detecting fraudulent advertisements regarding real estate based on image similarity. And support with this tool real estate advertisement portals.

During the design of this tool, we need to keep track of multiple things. First, we need to think about the target audience, and who will be using it. How does it change the format of this application? Then some questions target the main focus of the application. What does it mean to support advertisement portals? How to get a user's image and search for that image on the Internet? When we get matching or similar images and their advertisements, how can I determine which one is fraudulent and which one is not? Can our application be 100% sure that that particular advertisement is fraudulent? We will try to answer these questions in the following sections.

What is the Best Format

There were many formats of application taken into account, such as console, desktop, and web application. We wanted to find the best fit for a person with average knowledge of technology.

We believe that those requirements are met by using a web application. There was another idea of a web extension, but this would need to depend on the type of web browser. We believe that a web application is a powerful and flexible enough way to create an interface for our tool.

Why Web Application ?

If a user wants to see if some real estate advertisement is fraudulent, he shouldn't have to download a desktop application to see if that one particular advertisement is legitimate or not. So it is essentially aimed at people who don't have to do much work to get the result. In addition, using a web application means that the application is accessible on multiple operational systems and devices, such as mobile phones. More about the tools we will be using for the implementation of frontend and backend can be found in Chapter 6

5.1 Supporting Advertisement Portals

This web application should support real estate advertising platforms and should be extensible for more. We will be searching image that is taken from the advertisement portal, so

for us supporting the advertisement portal means that the user can pick whatever image of real estate he wants from that advertisement, to be later further analyzed by our tool. Subsection 6.2.8 describes which portals are supported.

Answering another question about how we will get the user's image, we decided to let the user paste the `URL` link into our tool. This means that users can copy an advertisement link from a supported portal and paste the link into the search bar in the application. Then they should be prompted with the images from that advertisement where they pick one, and then this image will be processed.

5.2 Getting Images From the Internet

Now that we have the image user selected, we can search for this image on the Internet to find similar images. Now the plan is for the backend to import this image into our Google Rever Image Search API, more about the development of this API is given in Section 6.1. This will provide us with two sets of images, as said in Subsection 3.3.4, we will receive `source` and `similar` images.

5.2.1 Point System

Our design will be point-based; points are meant to represent „red flags“, so they are meant to be given negatively. If the point is given, that means that the advertisement is suspicious in that category. We plan to take all of the advertisements we have gained from our Google Reverse Image Search API (`GRISA`), whose implementation can be found in Section 6.1, and create a baseline. From many `URLs` we can find the origin of the country the portal is based in and if the country differs, we will give points. Also, from `URL` we get the domain name of the portal, so if that domain is known to have many fraudulent advertisements, we will give points. `Source` images tab, provides us also with resolution, so we can also create a baseline from this and give points if the resolution of the image is below the baselines. And again the same thing for the image extension we get. We were also planning to use metadata from the image, e.g. to know the location from where the image was taken. Unfortunately due to the processing of images from Google, there is no metadata present.

These are the points we come up with:

- *Top Level Domain* (`TLD`): A scammer posting fake real estate listings might use images from one portal and post them on a portal from a different country to appear legitimate, aiming to deceive potential victims. This helps the scammer avoid the need for local images, thinking this will evade detection through reverse image searches. Additionally, it could be a good strategy to overcome language barriers, where the scammer thinks that by using a different language this listing will be harder to find.

- *Suspicious portal*: This point is useful for detecting portals that contain a lot of fraudulent advertisements. It's also suitable for portals lacking secure posting features, such as email, ID, phone verification, or even verification of the property.

- *Image resolution*: If the scammer is downloading an image they want to use for his fraudulent advertisement, there is a chance that they will maybe download this in lower resolution. For example, this can happen if they download an image in some page view that limits the resolution of the image.

- *Image extension*: Someone might use a different image extension with the belief that it can manipulate search engine results. This point is not that useful since Google's image procession makes most of the images `jpegs`, but from our experience, we have encountered different extensions in the reverse engine result, and that's why we will still include this point.

More detailed information on how we implemented them can be found in Section 6.2 of Chapter 6.

We want points to be flexible, so adding/removing points shouldn't be difficult, and we also want points to have a weight system. For example, if an advertisement gets a point for an extension and a top-level domain, the point for that domain is considerably more important than that for the extension of an image file. Therefore, by the number and weight of the points, if all points are given, the result should always be within the percentage range of $0\% - 100\%$. The percentage and points will be displayed to the user for each advertisement.

Setting up weights for points is subjective. But since real estate are properties that are within one state, because they are not movable, the `TLD` should get many more points than the extension of the image, which can be modified just by uploading it to the portal. This also can happen with resolution, where the portal can limit the resolution of the image.

But now comes the big question of whether the application can correctly determine fraudulent advertisements. We will try to answer this question at the end of this thesis in Chapter 8.

Chapter 6

Implementation

This Chapter will look into implementation details and hosting of our tool. Implementation of this tool can be found op publicly on GitHub[1]

6.1 GRISA (Google Reverse Image Search API)

At the moment, there is no official Google API available. Our implementation tries to create a suitable Python [34] interface for Google's reverse image search engine. If you visit Google's engine[2] and upload an image, you will notice that the `URL` is very long and consists of random symbols. This was the first obstacle for us because this meant that the nature of the web content was dynamic. The `UML` representation of the implementation can be found in the in figure 6.1.

6.1.1 Selenium

We solved this using the Python package *Selenium*[6] With Python, it is used to carry out automated test cases for browsers or web applications. You can easily use it to simulate tests such as tapping on a button, entering content into the structures, skimming the entire site, etc. [30]. So, *Selenium* provides us with the tools necessary to simulate users' behavior on the website. This means that if the user of the engine was able to reverse search some images, so should this tool.

For navigation on a web page, *Selenium* has to know which element we want to interact with and when. The part of „what element" is done throughout our `identificator` class `PageNav`. If an element is not found, results will not be found, or, in the worst case, the program crashes. For example, this can be done by not clicking on the first element with `cookies` which is mandatory to decline or accept when not logged in, but if the user found a way to log in even while using Selenium, then there would not be this element with cookies, and this would cause this problem. Therefore, we had to allow the user of this API to choose whether or not to use this element. Another thing we had to allow users to choose from was usage during local development or when deployed. Our Web application is deployed on *Heroku*[1], more information about this is in Section 6.5, and we had to change some things for this to work, specifically how we interact with elements. We cannot ensure that this will work with every deployment platform, but this was tested to work on *Heroku* and can be further expanded to support other deployment platforms. With the 'when'

[1] **The implementation of tool** - https://github.com/tomasfratrik/bachelor-thesis
[2] **Google reverse image search einge** - https://www.google.com/imghp?hl=en

part, it does not look like there should be any obstacle, but in reality there was a problem. When we tested this program, it would work 50 out of 51 times. After further testing, we found that this problem was caused by imperfect timings of web page element loading, so if the website loaded element just a second later, the program would crash, this could be affected by many factors, but mostly by internet speed. This was solved by implementing fixed wait using the *Seleniums* functions until the presence of an element on the website was found.

6.1.2 What do We Scrape

Now that we have discussed the navigation through the pages, let us dive into the scraping process and the specific content we extract from these pages.

As we mentioned in Subsection 3.3.4, we have two types of results that Google provides us with. The first results displayed to us are similar results. These contain images similar to our uploaded image. Google will display images that are visually similar to the one we provided. Here, we utilize our `Scraper` class, which uses the Python library *BeautifulSoup*[2] to scrape HTML content from the website. On this page, with each item of the result, we provide a picture, website name, short description, and link that takes you to the place where that picture was taken. We scrape those data into the `JSON` format. Then, if the user of this API decides, he can navigate with the button `Find sources of images`, which takes us to the page with results of exact matches. There we scratch the same things as we did with `similar` images. The `source` images there also have written the original resolution of the image.

At first glance, one might question the necessity of having the resolution information scratched from the text Google provided when it could be extracted directly from the image. However, it is important to note that platforms such as Google often compress images, resulting in a reduction in resolution. That is why we cannot get the exact photo on the website. But this is exactly why we have chosen `SSIM` low-resolution image comparison.

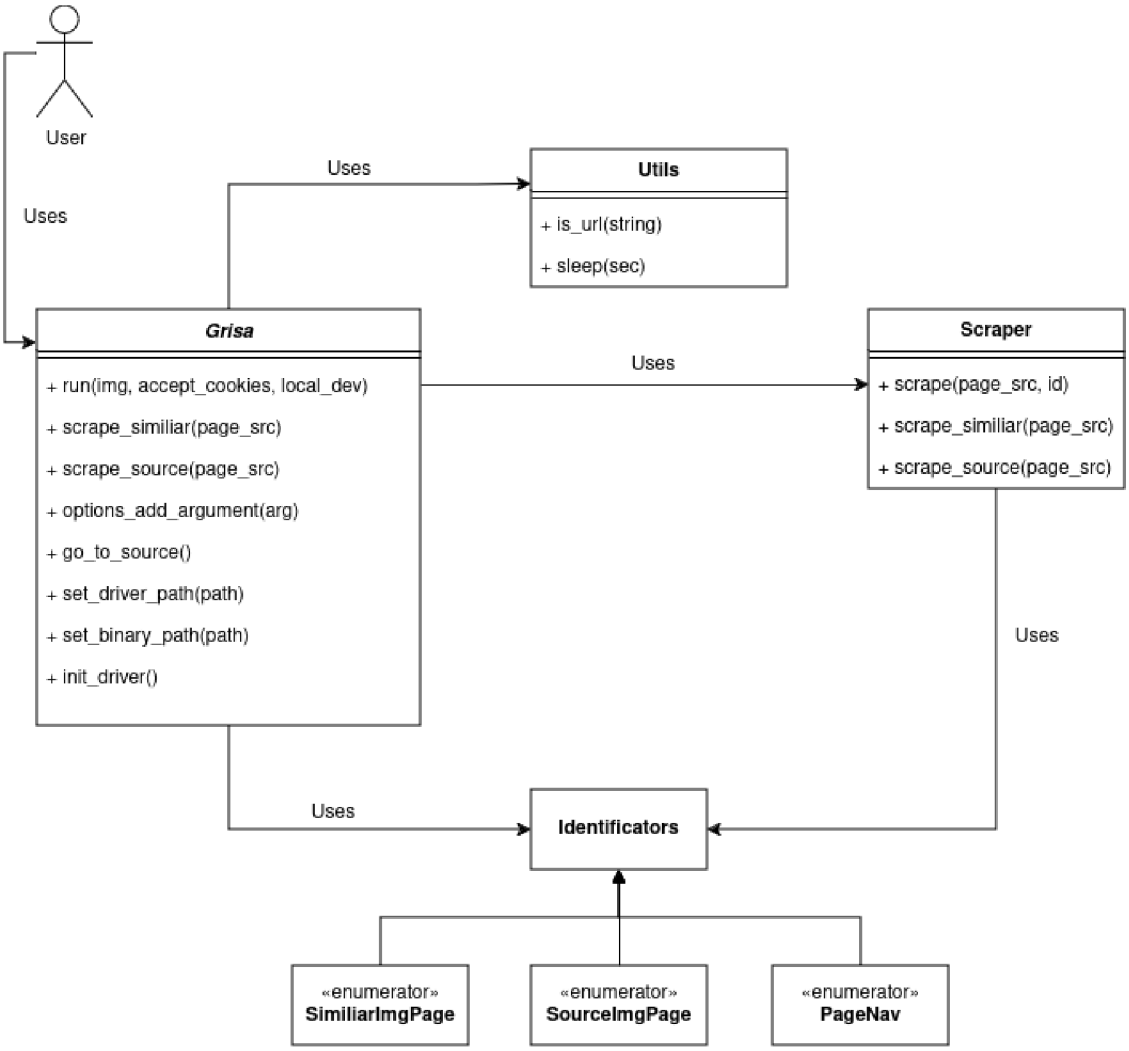

Figure 6.1: UML diagram of `GRISA's` implementation, with the most notable methods.

6.2 Backend

In designing this image system, scalability has been a key focus. The aim has been to create a framework that can easily adapt and grow, supporting future enhancements such as additional evaluation factors or support for new website portals.

The main application is built using *Flask* [3], a Web framework for Python. It exposes several endpoints, main entry endpoints are `/grisa/upload`, which is the primary `POST` endpoint for running image analysis using `GRISA`, figure 6.3 depicts UML diagram of its implementation.

Endpoint `/get_images_from_url`, is a `POST` endpoint to retrieve images from a given URL. Its UML diagram can be found in the figure 6.2.

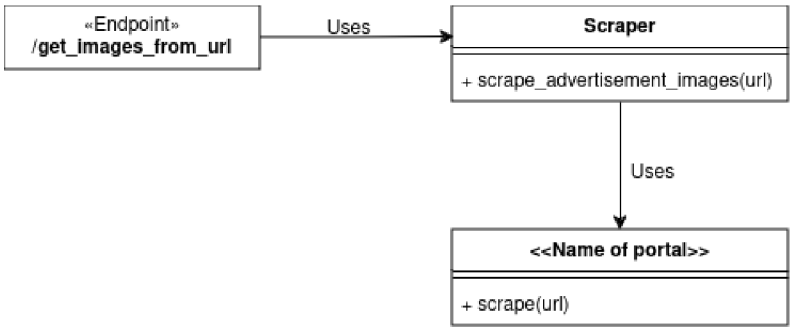

Figure 6.2: UML diagram of `/get_images_from_url` endpoint, with the most notable methods.

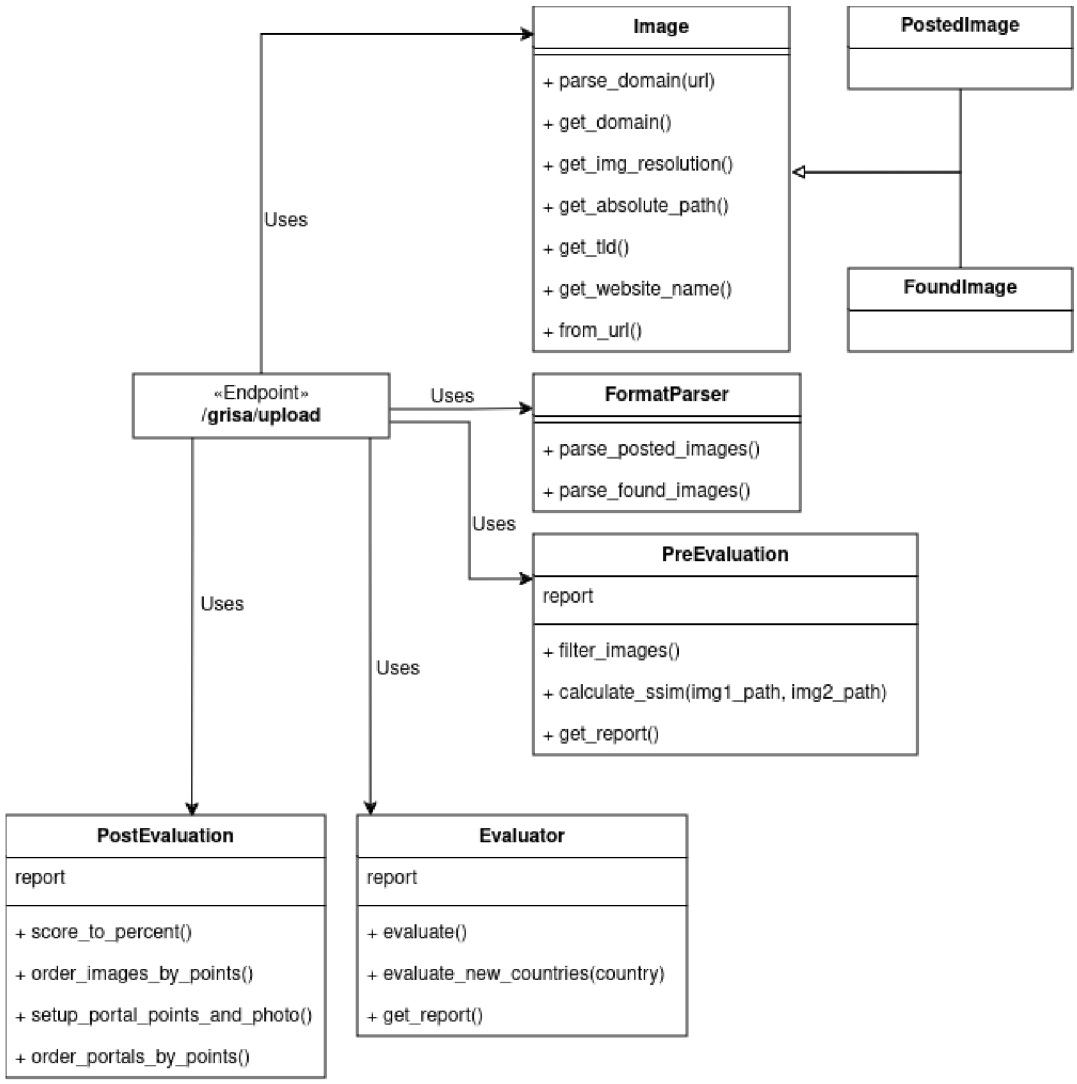

Figure 6.3: UML diagram of `/grisa/upload` endpoint, with the most notable methods.

6.2.1 Getting Images From Supported Web Portals

The first step is to retrieve the given URL, find out if the portal is supported by us, and return the images found in the estate. Each website portal has different types of image retrieval. It can differ just in element names for scraping or completely different methods. Therefore, the important thing for us was to make it possible to scale this to any portal if possible. More about this can be found in Subsection 6.2.7.

6.2.2 Retriving Selected Image

This is the part where we take the image, evaluate it, and send the result back.

For now, we retrieve the direct `URL` to the image and initialize the class `PostedImage` with it. Now we have 2 types of images, posted by the user represented by class `PostedImage` and those found throughout the API, represented by class `FoundImages`. Both are derived from class `Image`. They contain many useful methods for parsing those images so that we can later create a **report**.

After we obtain a selected image from the user, we parse it into `PostedImage`, and then we run `GRISA` with this posted image. From the output of `GRISA`, we get two lists of objects, one for `similar` images and the other for `source` images. These lists of objects are now parsed into the `FormatParser` class.

Problem With Saving Images

For image comparison that we will discuss in Section 6.3, we need to temporarily save images. This is done using the methods implemented within the image classes mentioned above. But to temporarily save them, we have to make `GET` request to their `URL`. The problem with this is that each request takes about a third of the second, so with 80 images and other processing the whole process took more than a minute. This was a big problem since no user would want to wait minutes for this result. Here, we utilized multithreading for requests, and we were able to fasten this request section to only 3 seconds. Now, the whole process usually takes around 15 to 20 seconds with a stable Internet connection.

6.2.3 Making a Format of Report

In the following stages, we will work with all images, and it is important to keep track of the context. For this, `FormatParser` creates a `JSON` format **report**, with which we will work from now on. In this **report**, we keep track of the current `status`, if there was no error present, we keep track of all enabled point modules and with how many points they are associated with, max points, baselines for those modules, and images itself. These images are objects with many attributes that can differ depending on the type of image. But in general, they will have their display photo URL, website URL, domain, and points gained with corresponding modules detected.

6.2.4 Pre-evaluation Stage

This is the stage where we can make any changes before evaluation. This is carried out by class `PreEvaluation`.

Portals that host advertisements have their main page for the individual advertisement with that picture, but we probably will get many more results from the same picture, which will probably occur at different places on the same website portal. This was the reason why

we initially removed redundant images in this pre-evaluation stage because when they were displayed in the frontend, the result was not clear. This was done by picking an advertisement with the highest resolution or points and removing others. But this approach is not good because this way we do not account for the scenario where there can be 2 different advertisement frauds with the same images on the same portal. This is why now we group them based on the portal in which they occurred. More about how we display them is in the frontend Section 6.4.

As we said, this change makes the result more clear, but there are still just too many images in the `similar` category from `GRISA`, that are visibly not the same real estate as the original. For this, we also do image comparison and filtering. This image comparison implementation can be found in Subsection 6.3.1.

6.2.5 Evaluation Stage

In this stage, we check every advertisement and calculate its points. This is being done by the `Evaluator` class. We wanted this to be modular, so adding, modifying, and removing modules is easy. For this, we have the Python package, which contains `points_map`. You can add the module to this map, create a file with the corresponding function name with the module, and implement its functionality. This `points_map` can look something like this:

```
points_map = {
    "top_level_domain": {
        "name": "Top Level Domain",
        "points": 3,
        "needs_url": True,
        "description": "Top level domain of the~website is probably "
                       "from a~different country than expected"
    },
    "suspicious_portal": {
        "name": "Suspicious portal",
        "points": 2,
        "needs_url": True,
        "description": "Portal is known for having many fraudulent advertisements"
    },
    "resolution": {
        "name": "Resolution",
        "points": 1,
        "needs_url": False,
        "description": "Resolution of the~image found on the~website "
                       "is lower than expected"
    },
    "extension": {
        "name": "Extension",
        "points": 1,
        "needs_url": False,
        "description": "Image extension is not common"
    },
}
```

So far there are 4 point modules. The reason behind the weight of the modules was explained in Subsection 5.2.1 of the Chapter 5. We will briefly discuss their implementation.

Top Level Domain

We go through all advertisements and create a baseline of the most occurring Top Level Domains. At least that was our first implementation, but, a lot of times the results were not accurate, so we decided to do baseline based on `source` images, if `source` images are not present, `similar` will be used.

For this, we have a map of all `TLDs` of countries. All `TLDs` that do not belong to the country we do not count. After this, we put the baseline of the country with the highest occurrence, and this baseline is then added to the **report**. Then we go through all advertisements and if the `TLD` differs, points are added. If more countries have the highest count of occurrences, all of them are added to the baseline and we notify the user with an alert that more origin countries were found in the frontend of the application.

In our implementation process, we've discovered that users might be dissatisfied with our results if they know that the country we've identified as the origin doesn't match the actual location of the real estate. This could occur, for instance, if they're the property owner. To address this issue, we've added another endpoint, `/grisa/set/country`, which accepts the existing **report** and country as input. It then recalculates the `TLD` point for each advertisement, assuming that the chosen country is the true origin. This change of country can be done in the frontend of the application.

Extention

Similarly, as with Top Level Domain, we create a baseline of all image extensions, and if some differ, we add points.

Resolution

This is only being done for the `Source` images, as we know they have an exact resolution. We create a baseline for all image resolutions. This is done by computing the pixel count for the image. The images are then compared with this baseline, and if they are lower, points are added.

Suspicious portal

Here we do not create any baseline. For this, we have a map of suspicious portals. And if the advertisement domain matches this, points are added. This map of suspicious portals was made from our testing, which can be found in Chapter 7. We found some portals [19, 31, 24] that have many fake listings and have added them to our list of suspicious portals. In the case, with *Airbnb* we found out that the same company can have multiple domains differing only in just `TLD` or have special domains like *.com.co*. For this reason, we have also implemented a way to include multiple domain types of the same portal if they exist.

6.2.6 Post-evaluation Stage

In this stage, any final adjustment to **report** can be made before returning. Here we order portal groups based on most points found within them, and set display image and points for the group that will be shown in the fontend and we order advertisements inside these portal groups based on points. This ordering is being done.

6.2.7 Supporting Portals

For us now, supporting the portal means that we can parse images from the portal based on `URL`. URL request is received throughout `/get_images_from_url` endpoint. Here, we use our `Scraper` class. Using the Python library *urllib*[7], we parse the `URL` and obtain the domain of the website. Then `second-level` and `top-level` domains of the website are compared with our `websites_map`, this is where we try to find the corresponding class for the website. For this, we have `portals_map` that has domains of portals with corresponding classes. If found, now we scrape `URL` links of images on the website based on the site's need. The difficulty of this now depends on whether the content is dynamically loaded or not; If not, the use of *BeautifulSoup* should be enough; if it is dynamically loaded, then it is necessary to use *Selenium* again with the cost of some extra time. After image `URLs` are retrieved, we send them back to the client, where frontend displays them to the user, who chooses one and sends it back to the backend. So, since the functionality of each scraping portal is separate, adding a new portal means that we can adjust the functionality as we want.

6.2.8 Which Ones are Supported?

Many portals have dynamic content, and scraping them requires *selenium*. But some don't. For demonstration, we decided to support static ones. We have chosen 2 portals, *reality.cz* and *nehnutelnosti.sk*. There are no other special reasons why we have chosen these two.

6.3 Filtering Images Based on Similarity

For this, we initially implemented filtering images, based on some `SSIM` constant, and if the images were below this similarity constant, they were removed. But for the user, this is not flexible, and we have also experienced a small number of cases where the image containing the picked by the user real estate had a smaller similarity than some other ones. In this particular case, this was caused by an extremely high difference in resolution or by Google Image Processing. The way we solved this was to calculate the similarity of advertisements compared to the original image and advertisements will be dynamically displayed in the frontend based on the similarity the user chooses.

6.3.1 Setting Images for Comparision

As mentioned, advertisements will be compared to the original image the user selected. But this is where we can utilize `source` images from `GRISA`. Since these images display the source of the image or even the same real estate from different angles, we can also use them with the original image for comparison.

6.3.2 Image Comparision

We go through `similar` images and compare them to our set of images consisting of the original image and `source` images. This is being done in the `Pre-evaluation` stage.

```
1 def calculate_ssim(img1_path, img2_path):
2     img1 = cv2.imread(img1_path, 0)
3     img2 = cv2.imread(img2_path, 0)
4
```

```
    if img1.shape[0] * img1.shape[1] > img2.shape[0] * img2.shape[1]:
        img1, img2 = img2, img1

    resized_img1 = resize(img1, (img2.shape[0], img2.shape[1]), anti_aliasing=True
    , preserve_range=True)

    return ssim(resized_img1, img2, data_range=255)
```

Listing 6.1: Python function to calculate SSIM

We save the best similarity result of that image into that advertisement in the **report**. In the end, we save best the similarity to **report** as a default threshold value, which the user later can change in the frontend of the Web application.

6.4 Frontend

For development, we used the JavaScript framework *Vue.js*. In addition, for faster development, we used a component library *Naive UI*, which provides us with useful components, such as buttons, drawers, and many others. In the implementation, we tried to modularize components as much as possible; this ensures that this frontend will be flexible, and scalable, and the code more readable. CSS was used for styles.

During implementation, we had five individuals, from different age groups, who evaluated various aspects of user experience. Their feedback and findings played a crucial role in refining and improving the overall user experience of our application. In this Chapter, we will refer to the feedback gathered during the implementation as „From the feedback...“ or any other suitable phrase.

As for our color palette, we used green and white, for which there were no complaints from the feedback.

In our context, you can refer to `View` as a page with different content.

6.4.1 Navigation

Navigation component `Nav` is present in the `App.vue`, so it is visible in every `View`.

Figure 6.4: Navigation of application

Navigation contains a header and navigation menu with a button that reroutes the user to different `View`. For mobile support, the navigation is adjusted with the width of the screen.

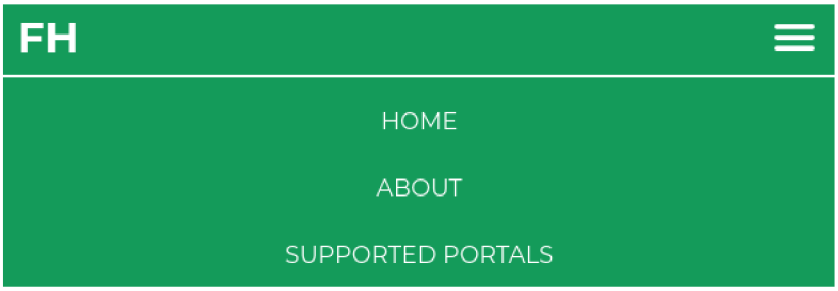

Figure 6.5: Navigation of application

6.4.2 Home View

This is the home page of the Web application. The routing entry point for this is at „/". Here, the user sees the URL input field, with a header that guides him on what to insert into the input field. From the feedback, we got our idea to add the option to support uploading images, without the need for a URL. With this option, the user can search for properties on the portal that we do not support, the user has to download the photo to do this, so it is inconvenient. This is where we utilized *Naive UI's* `Tab` component and also added the tab for uploading images.

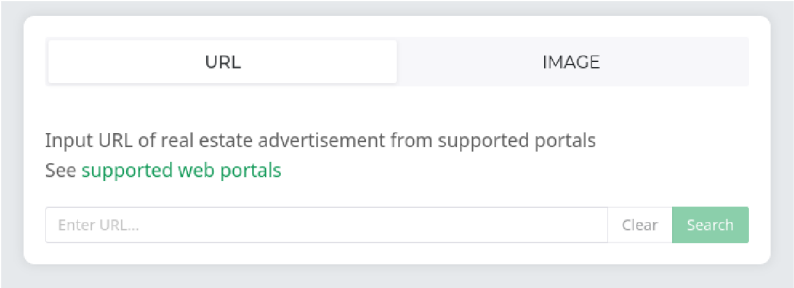

Figure 6.6: URL input of the Home View

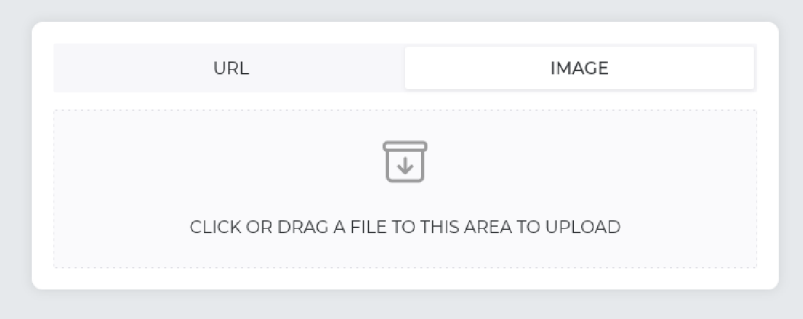

Figure 6.7: Image input of the Home View

Next to the input field, there are buttons to clear the input field and search field. The user cannot click these buttons unless there is something written in the input field. It is also important to give user feedback if possible; this is a feature that many users requested in the feedback of the user experience that was missing. So, we are trying to

utilize the notification component. For example, if the backend cannot fetch images, we will display a proper notification message.

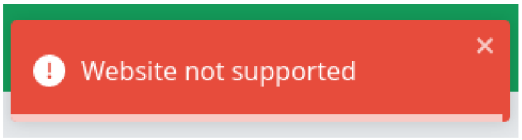

Figure 6.8: Alert of the application

After correct URL input, we utilize component `Drawer` and display to him all images we have fetched. Here, the user is asked to pick an image that best depicts the property and click the `Search` button, or the storno process with the `Close` button.

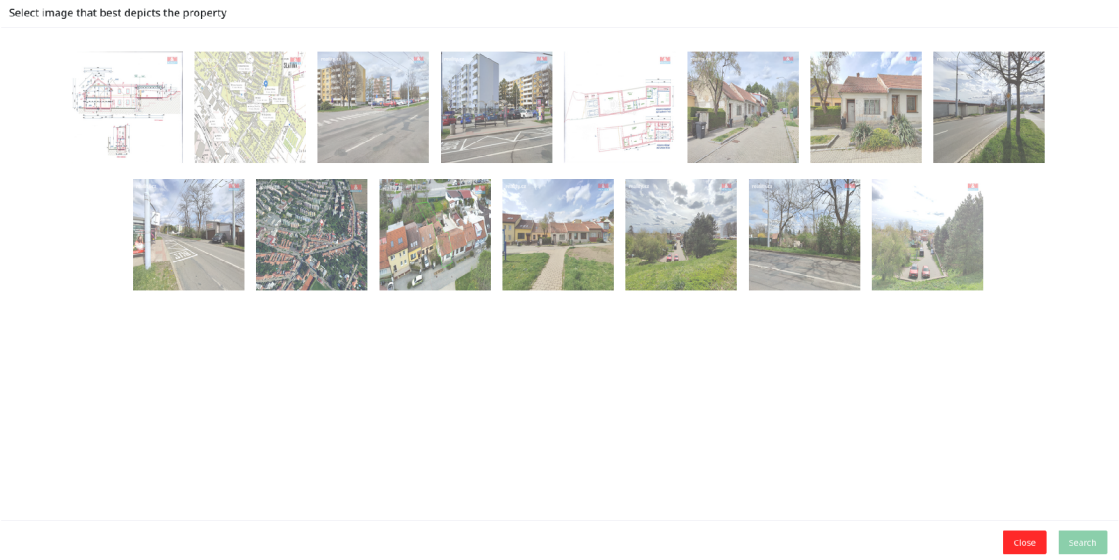

Figure 6.9: Select image drawer

After selecting the image and pressing the search button, the component with the loading circle will be displayed until the result is displayed. This result is called **report**. The user is then presented with 3 tabs `Your image`, `Source images`, and `Similar images`. These are the types of images we get from GRISA mentioned in Section 6.1. In the picture below 6.10, you can see what the final **report** result looks like on the Home View page.

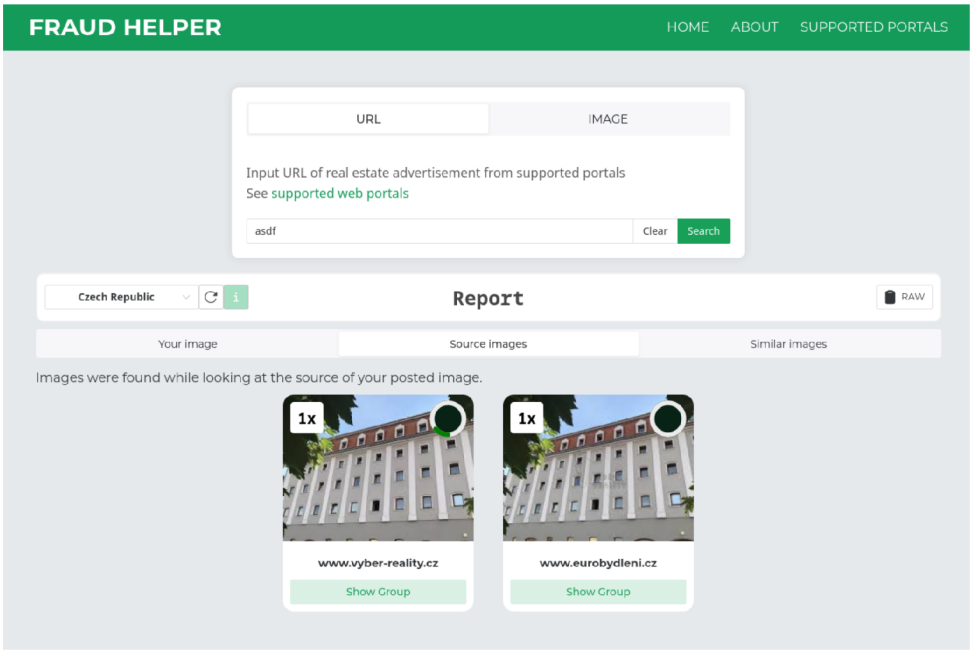

Figure 6.10: Home View with **report**

In **report** heading there is a component to change country, which sends requests to the backend. Users can also reset the country to its original country/countries.

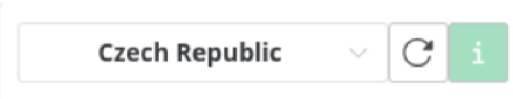

Figure 6.11: Component to change and reset origin of the country

In the **report** tabs, there are image group card components. These group cards are sorted, and the ones with the most points are displayed first. Component card `ImgGroupCard` is a card that groups the portal.

This card contains a photo, the number of advertisements in the group, the portal domain name, the button to show this group, and the circle with the percentage of how many points were given from total points.

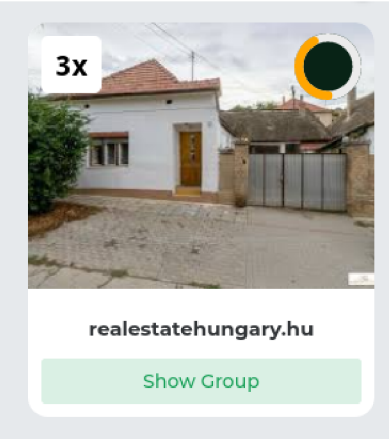

Figure 6.12: Image group card

The higher the percentage, the more red the color of the bar in the color becomes. The bar is changing color from green to yellow, orange, and red. This image and percentage are taken from the first advertisement in the group, which has the most points. The group contents are displayed in the modal that displays the cards of the advertisements, again in descending order by points. This card of a particular advertisement has an image, a redirection button to that advertisement, and a circle with a bar that shows the percentage of points with the percentage written in the middle.

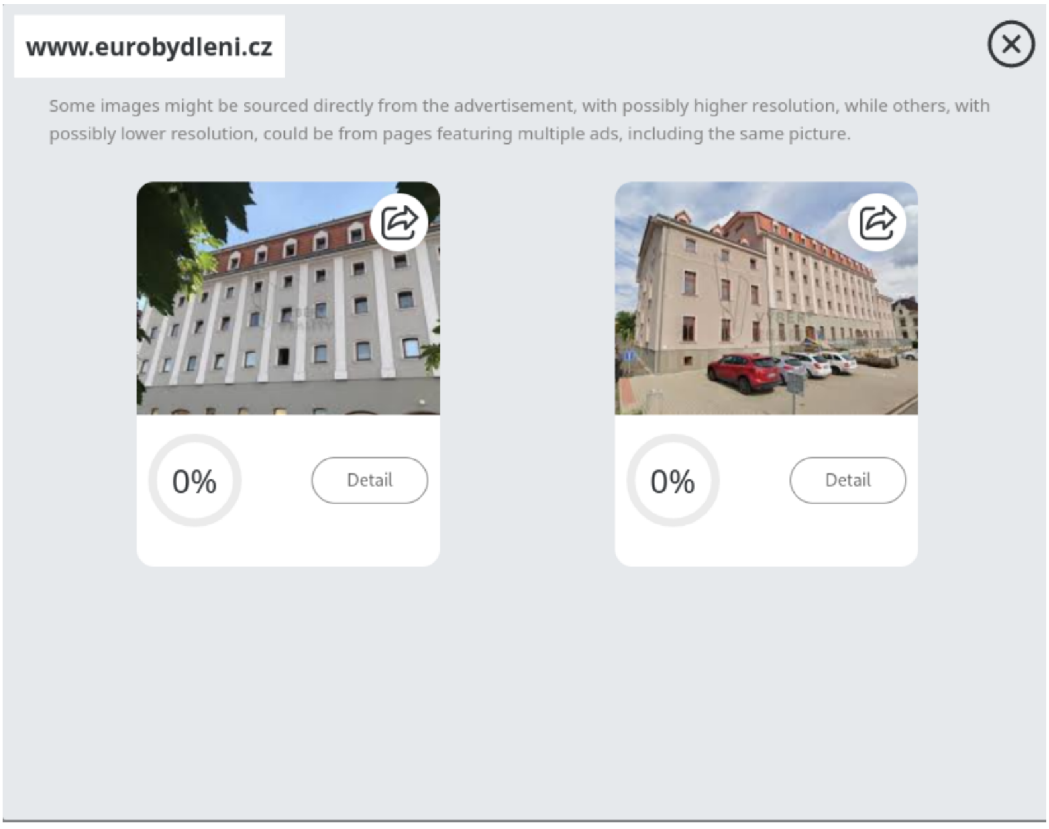

Figure 6.13: Image group modal

If the user also wants to see some details from the **report**, he can click on the detail button, which opens the modal with details, where there is the full image, general information, which contains the exact number of points, link to the advertisement, portal name, and domain. But more importantly, there is information that displays which point modules were detected with short descriptions of them.

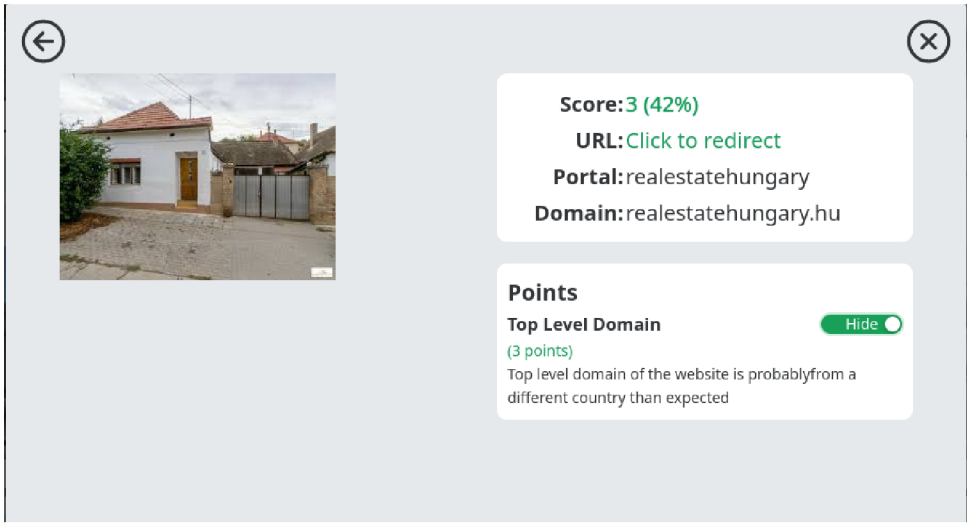

Figure 6.14: Advertisement detail modal

All of this information is taken always from the **report** which was obtained from the backend. If we change this module description inside our point modules in the backend, it will be updated in the frontend as well. From this modal, there are 2 buttons. One is to exit both modals and see other groups, and the second is to go back to see the modal of the portal group.

Our first implementation displayed all source images and all similar images under each other. Feedback on this was very negative. The data displayed showed too much information and users said that the displayed **report** is disorganized. This is why we separated those image types into tabs and grouped them. However we did not group those portals that had only one advertisement, so we skipped the group portal card and showed the advertisement card. Even though this made sense for us, we found out from the feedback when there were mixed group cards and cards of advertisements that this was still disorganized and the result did not look consistent, and therefore we decided to keep her only cards for groups. With those changes from the feedback, we kept in mind that our sample of individuals testing user experience is small. So, therefore, these changes are always easily revertable.

The source tab portals are displayed as a whole, because there often are not many results (10 to 20 most of the time) that are almost always displaying the same real estate. We say almost because this part is not tested, but in the other testing we have done on outside real estate, like houses, we have never experienced a case where there would clearly be different real estate.

But the similar images tab is different. It contains a lot more advertisements and some images clearly have not the same real estate, but they are still similar. As mentioned in Subsection 6.3.1, our first implementation filtered images in the backend, based on `SSIM` constant, and later we changed this so that the frontend receives all the images in the **re-**

port. By this, the user can change the similarity index themselves. This is done by the component `Slier`. The default value for this slider is taken from the **report**.

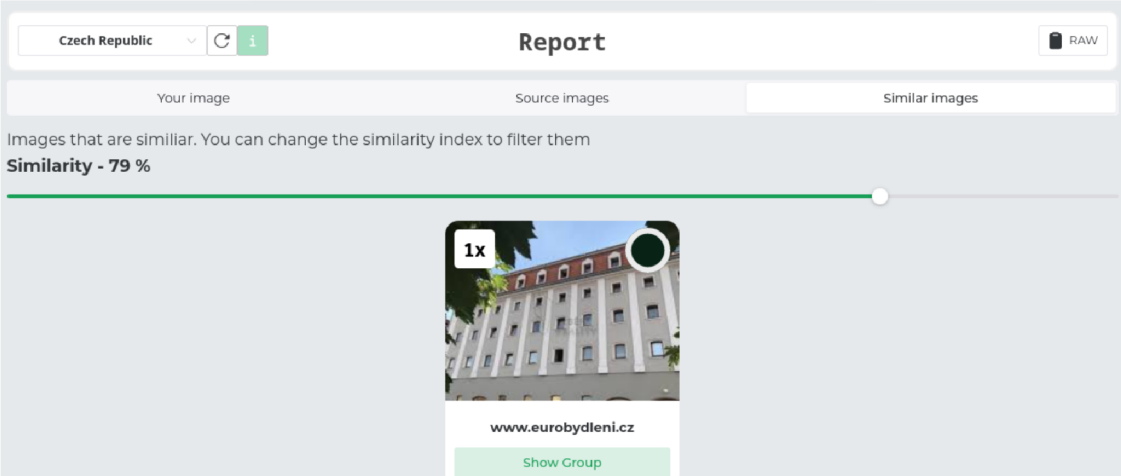

Figure 6.15: Similar images tab with slider

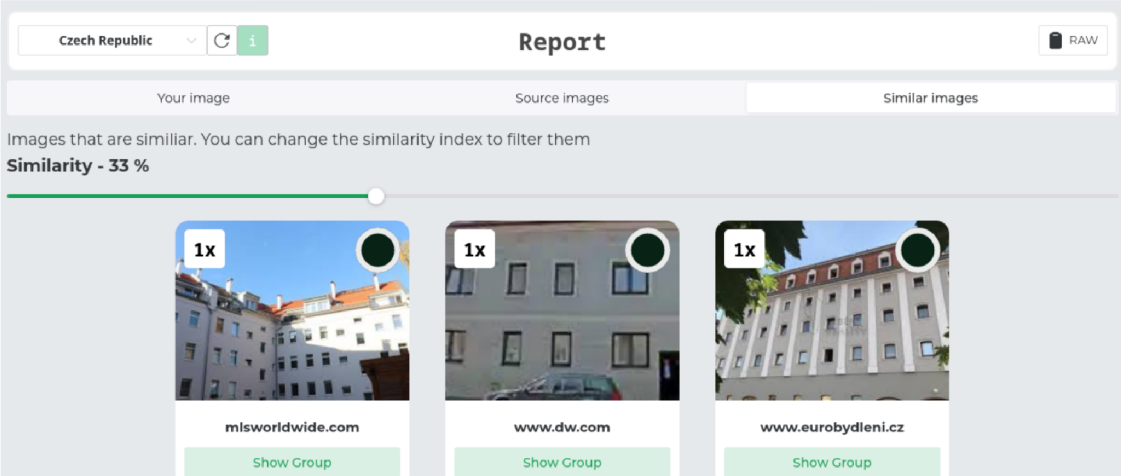

Figure 6.16: Similar images tab with adjusted slider

If a user wants to see the raw **report** data, there is a button for it next to the **report** header. Clicking on this button will show the `drawer` component with the **report** in a `JSON` format. At the top is a button to copy the **report** data.

6.4.3 About View

This page informs the user about this project being a work of bachelor's thesis. And presents documentation for it.

6.4.4 Supported Portals View

This page displays the currently supported portals in a table format that contains the domain name, link to the portal, the origin country of the portal, and the tutorial button.

Clicking on this button shows the component `Drawer` that has the tutorial that corresponds to that portal. This tutorial can contain any information on which `URL` link to choose from that portal. Also, any issues found with the portal can be written here. Adding such a tutorial was made easy and modular, by adding a portal to the table and creating a separate file for the tutorial.

6.5 Deployment

The `URL` link for the Web application can be found in the footnote[3]. For the deployment platform, we have decided to use *Heroku* [1]. The reason for this was our familiarity with the platform, relatively low-cost servers, and the ease of deployment. The mention of deployment is important for understanding the architecture of this entire Web application. During almost the entire development time, we had this web application divided into 2 parts, frontend, and backend. But at the end of the implementation, we have encountered a problem with the deployment. *Heroku* offers `500 Megabytes` limit to *Slug size*. *Slug size* is a bundle that contains code and its dependencies. And after our project grew more and more we exceeded the limit for our backend. This is where we had to split the backend into 2 parts. We split apart `GRISA` and created for it a separate endpoint API and the rest of the logic, which now communicate with their `API` endpoints. After this, now our entire tool / Web application has 3 parts, frontend, backend, and `GRISA`.

The application was developed and tested with *Firefox* and *Chrome* Web browsers. However, we cannot guarantee its compatibility with others.

[3] **Link to Web application** - https://bt-frontend-6397f2521b23.herokuapp.com/

Chapter 7

Experimental Evaluation

In Chapter 6 Section 6.4 we have explained the reasoning of some changes by feedback on the frontend. This was done by over-the-shoulder testing. There, we focused on how the result information is displayed. However, we did not focus on the actual results. In this Chapter, we will try to test the functionality of our tool.

7.1 Experiments

In this Section, we will try to do two experiments. The first experiment 7.1.1 will try to find our posted image on the website which is meant for posting images publicly. The second experiment 7.1.2 will try to find actual real estate fraud.

7.1.1 Finding Image

In this experiment, we want to see that if someone uploads an image on the Internet it will be found by Google's indexation. For this, we have uploaded image of building on a public German website that allows users to post publicly any images. We have also removed any watermarks from this image because scammers may also. Within a day, the image was successfully found. This is depicted in the figure 7.1.

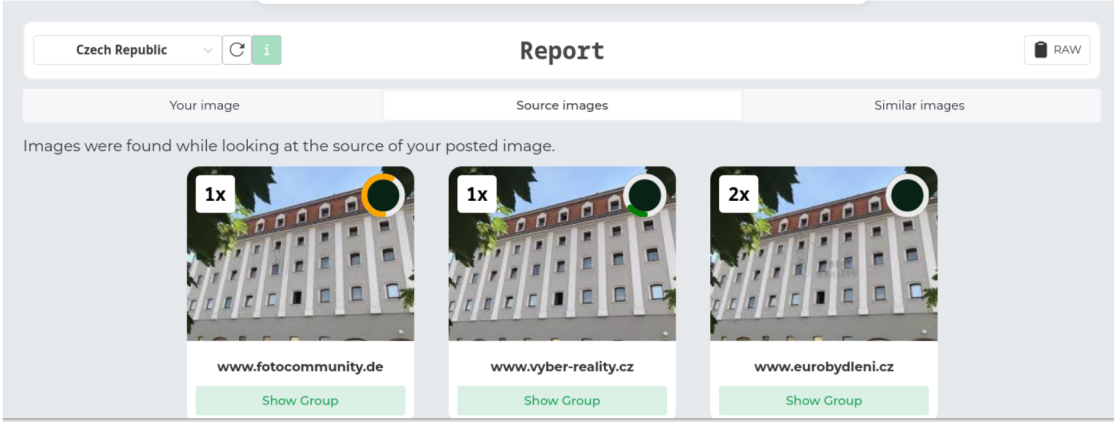

Figure 7.1: Finding image on public image uploader website

7.1.2 Finding Fraudulent Advertisement

In this Section, we want to test if our tool can correctly find and detect fraudulence in advertisements. The best approach for testing would be to find an actual fraudulent advertisement. This could take a long time with no promise of finding one. However, at the start of the thesis, we had some listings for potential fraud saved, but unfortunately, listings from one portal were already taken down. That is why we thought that therefore we could not use them. However, we found out that Google's indexing of images sometimes keeps the images indexed, even if they were removed. We cannot guarantee for how long this is or what metrics must be met for this to be kept indexed but thanks to this we can search for one. Figure 7.2 shows the result from **report**, where *Airbnb* received points for suspicious portal and *bezrealitky* received points for resolution, and `TLD`. This image no longer works.

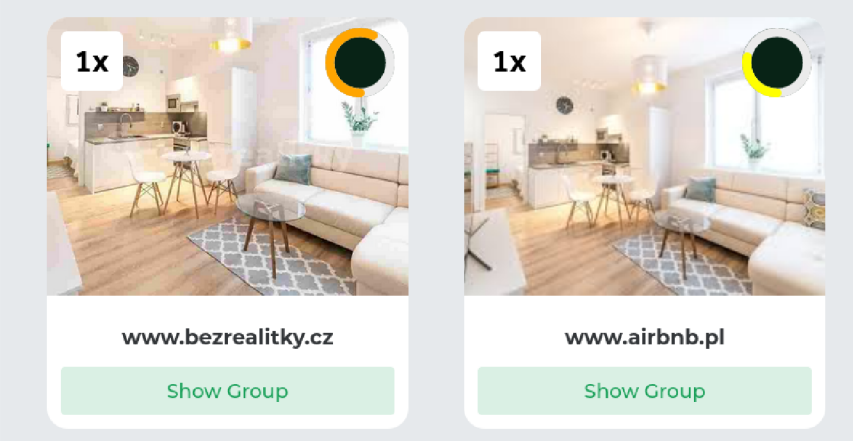

Figure 7.2: Found fraud with origin in Poland.

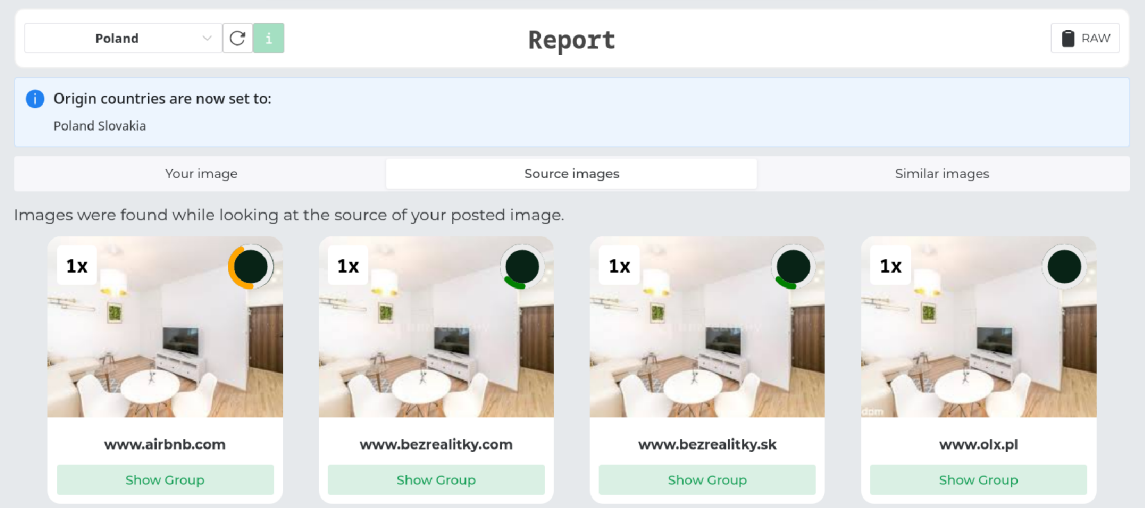

Figure 7.3: Result of **report** from an image of a different angle, with two countries as the origin.

Result from different angle can be seen in figure 7.3. There, our tool determined the origin country to be either Slovakia or Poland. As of the time of writing this thesis, this image

is still working and will be present in the GitHub[1] repository, also in this repository you can find a demo video of us finding the fraud with this tool.

In the listing on *Airbnb.pl*, we see a real estate that is located in a Polish city, which is the same for *Airbnb.com* and *olx.pl*. When we open the other advertisement on *bezrealitky.cz*, the page says that the listing was removed, but there we can still see that they claimed the property was from a Czech city, and as for *bezrealitky.sk* they also claimed the Czech city.

The listing on *Airbnb* has already been present there for 6 years, we have checked reviews and reviewers. We found multiple listings for the same Poland city from different portals. Advertisements on *bezrealitky* were taken down. On the Slovak version of *bezrealitky*, we found a listing for the Czech country. Thanks to this knowledge gained we can assume that the fraud was present in *bezrealitky*, and the original real estate is from Poland.

Investigation

There is no need for our thesis to 100% confirm our assumption, but we can try to simulate the next steps of the user. From the listing on the Polish site, we found two clues that confirm our assumption. These two clues are in two different rooms of the flat and from all present images we can deduce that they are part of the same flat. In the figure 7.4 we can see TV with a channel that has written in the top-right corner NOWA. We found out that it is a Polish television channel *Nowa*.

Figure 7.4: Polish news channel *Nowa*

Another clue is in figure 7.5 where we can see a building in the background. After investigating streets that were present in the listing. We found the same building on the Polish street, that is present in figure 7.6.

[1]**The implementation of tool** - https://github.com/tomasfratrik/bachelor-thesis

Figure 7.5: Found building in the background

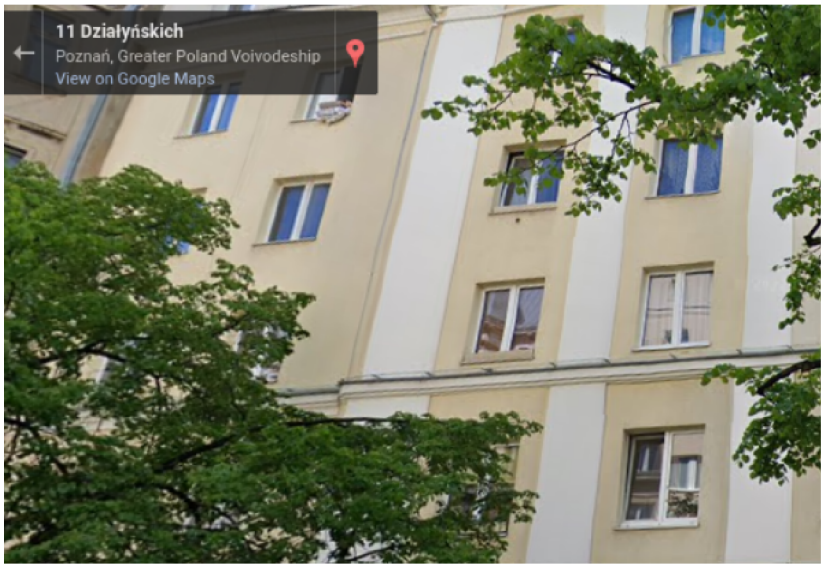

Figure 7.6: The same building found on the street that is mentioned in a Polish advertisement

7.1.3 Results of the Experiments

In the first experiment 7.1.1 we found out that if someone posts an image on the Internet this image will be indexed after a while, and no special actions are needed. In the second experiment 7.1.2 we found an actual fraudulent advertisement. The first image of the property detected fraud correctly, with results in Poland and the Czech Republic. The second one showed fraud being present either in Poland or Slovakia.

When someone sees the result of *bezrealitky* they would maybe immediately assume that it is fraud because one is in Slovakia and the other in Czechia and they both point at the same address. However, when going through some advertisements, we have once encountered a similar situation. In the figure 7.7, there are 3 listings again from *bezrealitky*. Two are from the Czech Republic and one is from Slovakia. We have checked this and found out this is not a fraud. This is due to the property being close to the border of these countries, and since the countries have very similar language, someone decided to post this also on the Slovak portal. This assumption was only possible thanks to our knowledge of these two countries, and this is why user touch and thinking are needed.

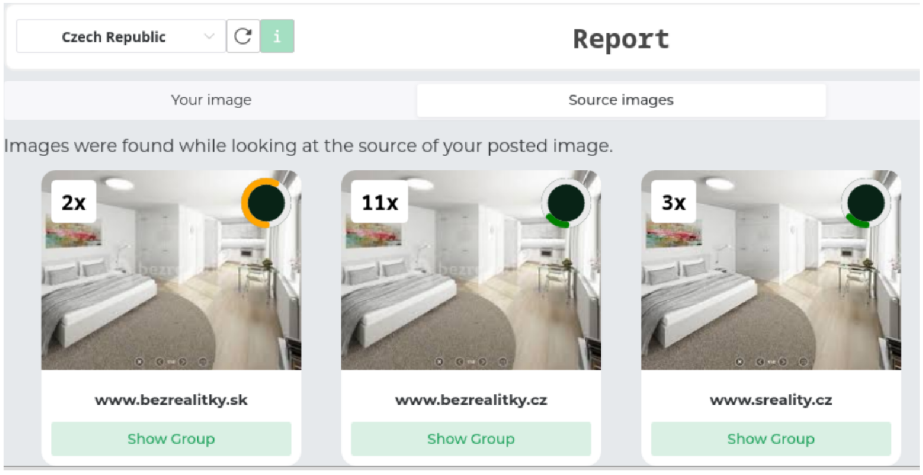

Figure 7.7: Suspicious listing that is not a fraud

In addition, **report** showed us that all instances of fraud images in the experiment had lower resolution. Based on all of this we could say that this experiment was successful, but this was only possible thanks to suitable conditions. For example, if there were more fraud advertisements, than the real ones present from the same country then it would detect the origin country wrongly. In our case, we have given points *Airbnb* for the portal even though the fraud was present on another portal. But thanks to the provided tools and results, with our human touch, we were able to find which one was the real fraud with certainty.

Chapter 8

Conclusion

Our task in this thesis was to develop a tool to detect fraudulent real estate advertising based on image comparison. In-kind a way, we have created 3 tools that create one fraud detection system.

First, we have delved into the topic of real estate advertising frauds and why and how they are done. For us to be able to find similar real estate images on the Internet, we had to take a look at reverse image search engines. From the presented engines, we have made a pick. This is where our first tool was created, a `Google Reverse Image Search API`, which was an essential part of our system. This provided us with many results. In fact, too many results. Here, in the thesis, we examine some similarity measures. We have performed some tests for similarity measures and picked the best that suits us. This created a way for us to filter those results. The second tool that we created is the tool that evaluates results. Here we have created a scalable backend connected with `GRISA` for our application. After evaluating all the results, we return **report**. But for this to have some use case for common users of the Internet, we had to create an interface to easily import images from advertisements to our tool, and to also display **report** in a visually pleasant way, which was partially designed thanks to user experience experiments. This is where our third tool was created. It is a frontend for our application. These three tools essentially together create a Web application.

To answer the question of whether our application can 100% detect fraud, we can look at the results of Chapter 7. Here we mention that we successfully found a fraudulent advertisement, but only thanks to suitable conditions. And this is essentially the reason why our application is not able to detect fraudulent advertisements for sure. That part now depends on the user. We have provided them with found advertisements and, if we could not detect fraud, we have at least provided tools for them to further investigate. And this was the exact way how we confirmed which one was a fraud with certainty. We believe that no tool can detect fraud with 100% precision, but we believe that we can always get closer to this number, even with our tool, which is why we tried to create a scalable and modular tool, where further advertisement analysis can be implemented.

Bibliography

[1] *Heroku* [https://www.heroku.com/]. Accessed: February 4, 2024.

[2] *Python Package Index - Beautifulsoup4*. Python Software Foundation. Available at: https://pypi.org/project/beautifulsoup4/.

[3] *Python Package Index - Flask*. Python Software Foundation. Available at: https://pypi.org/project/Flask/.

[4] *Python Package Index - opencv-python*. Python Software Foundation. Available at: https://pypi.org/project/opencv-python/.

[5] *Python Package Index - scikit-image*. Python Software Foundation. Available at: https://pypi.org/project/scikit-image/.

[6] *Python Package Index - Selenium*. Python Software Foundation. Available at: https://pypi.org/project/selenium/.

[7] *Python Package Index - urllib3*. Python Software Foundation. Available at: https://pypi.org/project/urllib3/.

[8] *What Is Yandex Reverse Image Search? How Does It Work?* 2011. Available at: https://zenserp.com/what-is-yandex-reverse-image-search-how-does-it-work/.

[9] *How to Do an Image Search on Bing*. 16. July 2016. Available at: https://www.wikihow.com/Do-an-Image-Search-on-Bing.

[10] *Třešňová, Plzeň 4 - Doubravka*. 19. October 2023. Available at: https://www.reality.cz/prodej/domy/mesto-Plzen/BYI-030943/?c=17.

[11] *Nepomucká, Starý Plzenec*. 24. January 2024. Available at: https://www.reality.cz/prodej/domy/mesto-Plzen/EXN-BG0ZZS/?c=14.

[12] *Střelnice, Líšeň*. 2. February 2024. Available at: https://www.reality.cz/prodej/domy/mesto-Brno/BYI-031123/?c=3.

[13] Abhishek Agarwal, M. *Automatic Detection of Click Fraud in Online Advertisements*. 2012. Dissertation. Faculty of Texas Tech University. Available at: https://ttu-ir.tdl.org/server/api/core/bitstreams/97196924-d275-4868-a245-53821be43537/content.

[14] Antoniadis, P. *Algorithms for Image Comparison*. 2023. Available at: https://www.baeldung.com/cs/image-comparison-algorithm.

[15] Banton, C. *How Escrow Protects Parties in Financial Transactions*. 17. August 2023. Available at: https://www.investopedia.com/terms/e/escrow.asp.

[16] Beck, B. *The Top 7 Reverse Image Search Tools and How to Use Them*. 27. February 2024. Available at: https://www.clearvoice.com/resources/reverse-image-search-tools/.

[17] Camella. *Residential Real Estate's Role on the Thriving Digital Economy*. May 2023. Available at: https://www.camella.com.ph/residential-real-estates-role-on-the-thriving-digital-economy/.

[18] Daiki, C., Kunio, H., YOSHIOKA, K., MATSUMOTO, T., et al. Detecting and Understanding Online Advertising Fraud in the Wild. *IEICE Transactions on Information and Systems* [online]. 2020. Available at: https://www.jstage.jst.go.jp/article/transinf/E103.D/7/E103.D_2019ICP0008/_pdf.

[19] Dimitriou, N. Real Estate Fraud and How We Stopped a Property Listings Scammer. *Seon* [online]. Available at: https://seon.io/resources/real-estate-fraud/.

[20] Edmund, P. D. *How Search by Image works*. Available at: http://weitz.de/sift/.

[21] Glew, D. and Vrscay, E. R. *Max and min values of the structural similarity (SSIM) functionon the L2 sphere*. 2012. Department of Applied Mathematics. Available at: https://www.math.uwaterloo.ca/~ervrscay/talks/post206_iciar12.pdf.

[22] Golowczynski, M. *Google reverse image search: Everything you need to know*. 13. November 2012. [Online; posted 13-November-2012]. Available at: https://smartframe.io/blog/google-reverse-image-search-everything-you-need-to-know/.

[23] Google. *How Search by Image works*. Available at: https://www.youtube.com/watch?v=keTZaJg0784.

[24] Hunter atum. Scam apartment listings are everywhere. Here's how to spot them. *Washingtonpost.* [online]. October 2022. Available at: https://www.washingtonpost.com/technology/2022/10/25/avoid-apartment-rental-scams/.

[25] Kanei, F., Chiba, D., Hato, K. and Akiyama, M. Precise and Robust Detection of Advertising Fraud. In: *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*. 2019, vol. 1, p. 776–785. DOI: 10.1109/COMPSAC.2019.00115.

[26] Khurma, M. *Euclidean Distance Formula*. Available at: https://www.cuemath.com/euclidean-distance-formula/.

[27] Luis, R. *10 Signs You Could Be Dealing with a Real Estate Scammer* [online]. Green Residential, july 2016 [cit. 2016-7-27]. Available at: https://www.lifewire.com/how-to-reverse-search-logic-3482669.

[28] Niku Ekhtiari, P. *Comparing ground truth with predictions using image similarity measures*. 2021. Available at: https://up42.com/blog/image-similarity-measures.

[29] Rui, T. M. . Y. *Image Similarity* [online]. SpringerLink, 2009 [cit. 2009]. Available at: https://doi.org/10.1007/978-0-387-39940-9_1014.

[30] Simplilearn. *An Introduction to Selenium with Python.* 28. February 2023. Available at: https://www.simplilearn.com/tutorials/python-tutorial/selenium-with-python.

[31] Spittel, M. Fake Craigslist Houses for Rent. *Sellbuymdhomes.* [online]. Available at: https://sellbuymdhomes.com/real-estate-blog/craigslist-rental-scams/.

[32] Strandell, B. J. Real Scams on Real Estate Sites. *Basedo* [online]. June 2023.

[33] Tim, F. *How to Do a Reverse Search to Find Something Online* [online]. Lifewire, december 2022 [cit. 2022-12-31]. Available at: https://www.lifewire.com/how-to-reverse-search-logic-3482669.

[34] Van Rossum, G. and Drake, F. L. *Python 3 Reference Manual.* Scotts Valley, CA: CreateSpace, 2009. ISBN 1441412697.

[35] *Reverse image search* [online]. September 2023 [cit. 2023-9-3]. Available at: https://en.wikipedia.org/wiki/Reverse_image_search.

Appendix A

Contents of the included storage media

This appendix lists the contents of the attached memory media with descriptions.

- `/` — Storage media
  - `README.md` — Contains URL link to the application, with local installation manual
  - `xfratr01.pdf` — This thesis in the PDF
  - `text/` — LaTeX source code of thesis
  - `demo/` — Video demonstrations and assets used for them
  - `src/` — Source code of the application
    - `backend/` — Backend of the application
    - `frontend/` — Frontend of the application
    - `grisa-api/` — Endpoint API for `GRISA`