ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ
към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ

ДИПЛОМНА РАБОТА

по професия код 481020 „Системен програмист“
специалност код 4810201 „Системно програмиране“

Тема: Приложение за автоматично качване и тагване на изображения във фотобанка.

Дипломант:
Стела Тихомирова Касабова

Дипломен ръководител:
инж. Мартин Стоянов

СОФИЯ

2 0 2 2

УВОД

Може би фотографията не е най-старата форма на изкуство, но е една от най-бързо развилите се сфери. Рисуването и скулптурата, например, са имали много повече време да се усъвършенстват, но въпреки това фотографията може да се сравнява с тяхната изящност и красота.

В днешно време всеки може да се занимава с фотография – нужен е само телефон с камера. Превърнало се е от интересно хоби в сложна и трудоемка професия. С набирането на популярност се появят и нови опции за продаване на снимки – фотобанки, където потребители могат да качат своите фотографи или да купят правата за ползване на нечии чужди.

Създадени са приложения, чиято цел е да улеснят и забързат процеса на продаване - качването на множество снимки наведнъж не е приятно, особено ако става за дума за стотици или хиляди файлове. За съжаление част от тези решения на проблема просто създават нов – добавяне на ключови думи. Те помагат на потребителите да намират снимки по-бързо или да са сигурни, че работата им има по-голям шанс да бъде намерена. Не всички приложения предлагат добавянето им към снимките под формата на метаданни, което улеснява самото качване, но има не малък шанс да навреди на доходите, идващи от продаването във фотобанки.

Целта на дипломната работа е да предостави интерфейс в който да може да се избере директория от машината на потребителя, след което да визуализира снимките и да покаже предложените ключови думи. Потребителят има възможността да архивира директорията и да качи снимките, както и техните тагове под формата на метаданни, към FTP сървър.

1. ПЪРВА ГЛАВА - Проучване

1.1. Съществуващи реализации

1.1.1 Dropstock

Целта на Dropstock е да служи като посредник между потребителя и сървърите. Приложението е направено за тези, които вече използват услугите на Dropbox за да съхраняват снимките си. То работи като се свърже с Dropbox акаунт, след което потребителят трябва да обозначи папка, в която да действа Dropstock. Добавянето на файлове в тази папка ги праща на предварително избрани FTP сървъри. Има услуга, която предлага ключови думи (тагове), описания и заглавия за снимките, но опцията за тяхното качване все още не съществува.

1.1.2. Wirestock

Това не е приложение, а по-скоро услуга. Подадените снимки не се качват в профила на потребителя, а в този на компанията, която е създала Wirestock. Тази функционалност има плюсове и минуси – по-вероятно е повече хора да намерят и купят снимките през вече известният профил на приложението, но по този начин се затруднява създаването на портфолио от потребителите. Тук, също като в Dropstock, няма опция за добавяне на метаданни.

1.1.3. StockSubmitter и Microstock Plus

StockSubmitter е един от най-добрите варианти за масово качване към фотобанки. Въпреки че приложението работи чудесно само, за още по-добри резултати създателите на StockSubmitter са добавили Microstock Plus. С помощта на този уеб базиран софтуер се дава възможност за организиране на файловете, следене на продажбите, свързани с профила на потребителя, както и добавяне на метаданни като ключови думи.

2. ВТОРА ГЛАВА -

2.1. Функционални изисквания

Приложението трябва да предостави интерфейс на потребителя през който да се визуализират и разглеждат снимки от избрана директория. Заедно с файловете се зареждат и ключови думи и описание, предоставени от Azure Computer Vision. След това може да се премине към архивиране, което се запазва в Azure Data Lake (по-точно Azure Blob Storage) или качване чрез FTP. При пращането към сървър разширенията на снимките се променят на .JPEG, тъй като повечето сайтове работят с този тип файлове. Потребителската активност се запазва в .log файл, ако няма такъв при инициализиране на програмата се създава нов.

2.2. Технологии за разработване

2.2.1. Език и библиотеки

Избрах да работя по приложението на езика Python. Като един от най-използваните и универсални програмни езици разполага с много библиотеки и източници на информация за разучаване на функционалностите. Използвах библиотеките logging, os, ftplib, PIL, zipFiles, azure, iptcinfo3.

Iptcinfo3 е библиотека с отворен код, разработена първоначално от Джон Картър за езика Perl, след което се адаптира да работи с Python от Тамаш Гулачи[1]. Позволява създаването и модифицирането на метаданни за изображения. Една снимка може да има различни типове метаданни, като най-използваните са Exif и IPTC. Exif се фокусира на техническите детайли, като например тип камера или използвани настройки, докато IPTC набляга на съдържанието на самата снимка[6]. Тъй като е необходимо

да се добавят ключови думи и описание към файл, смятам, че данни от типа IPTC ще са най-подходящи. С помощта на тази библиотека изпращането на допълнения към съдържанието на снимките се улеснява.

Освен тези библиотеки използвах и рамката Tkinter. Това е един от най-използваните инструменти за разработване на графичен интерфейс в Python, който разполага с множество функционалности за създаването на стабилно приложение. Произлиза от Tcl/Tk, добавяйки обектна ориентираност [2]. Това прави кода по-лесен за четене и разбиране, особено за програмисти вече свикнали с обектно-ориентирани езици.

За реализирането на проекта съм използвала PyCharm Community Edition. Тази развойна среда е направена специално за Python и позволява лесна работа с много файлове едновременно.

2.2.2. Azure

Архивирането на файлове, както и анализирането на снимки, се случват с помощта на Microsoft Azure. С абонамент се получава достъп до много услуги, но за този проект ни интересуват най-вече две от тях.

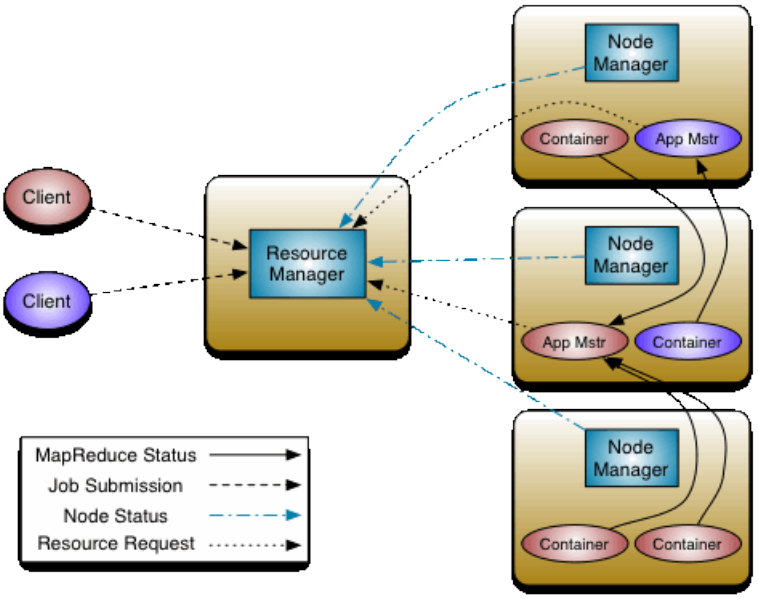
Azure Computer Vision е API, което има възможността да анализира снимки или видео записи и да връща данни за тях, включително и увереност в отговора [3]. Изключително полезно е за разработването на проекта, но идва с ограничения. При работата с изображения трябва да се има в предвид, че фотографът не може да надвишава размер от 4MB. Височината и ширината на снимката трябва да имат стойност по-голяма от 50 пиксела. Computer Vision разполага с много функционалности, някои от които са: намиране на обекти, генериране на ключови думи и описания, засичане на маркови продукти и други [8]. Разпознаването на обекти зависи

от това дали предметите са достатъчно големи (поне 5% от снимката), дали са струпани заедно и дали се различават само по производителя. Ако някое от тези условия е изпълнено е възможно да доведе до неадекватен отговор от страна на Computer Vision [9].

Data Lake е облачна услуга, предлагана от Azure, която е базирана на моделът на работа на Apache Hadoop YARN (Yet Another Resource Negotiator) [10]. Действа с имплементирането на клъстери, които са групи от машини, действащи заедно за да съхраняват и анализират данни. В системата тези машини се наричат „nodes“/”възли”. Структурата на Hadoop YARN съдържа в себе си 6 части [11]:

- Client – подава заявки към компонентите.
- Container – тук се изпълняват заявките. Съдържа Container Launch Context (CLC), в което са записани нужните команди и ресурси за извършването на заявка.
- Application Master – взима CLC от Node Manager и комуникира с Resource Manager.
- Node Manager – следи действията, който се извършват във възложения му възел. Главната му функционалност е да се грижи за контейнерите. Работи заедно с Resource Manager, за да се контролира употребата на ресурси между възлите. Всички действия, извършени от Node Manager, се пазят в логове.
- Resource Manager – получава заявките от клиента и се грижи за разпределянето на свободните ресурси към клъстерите, които се нуждаят от тях.
- Application Manager – отговаря за следенето на група заявки, подадени на възел. Проверя дали данните са валидни, както и

статуса на вече инициализирани действия. Може да откаже изпълнението на заявка ако няма достатъчно свободни ресурси.



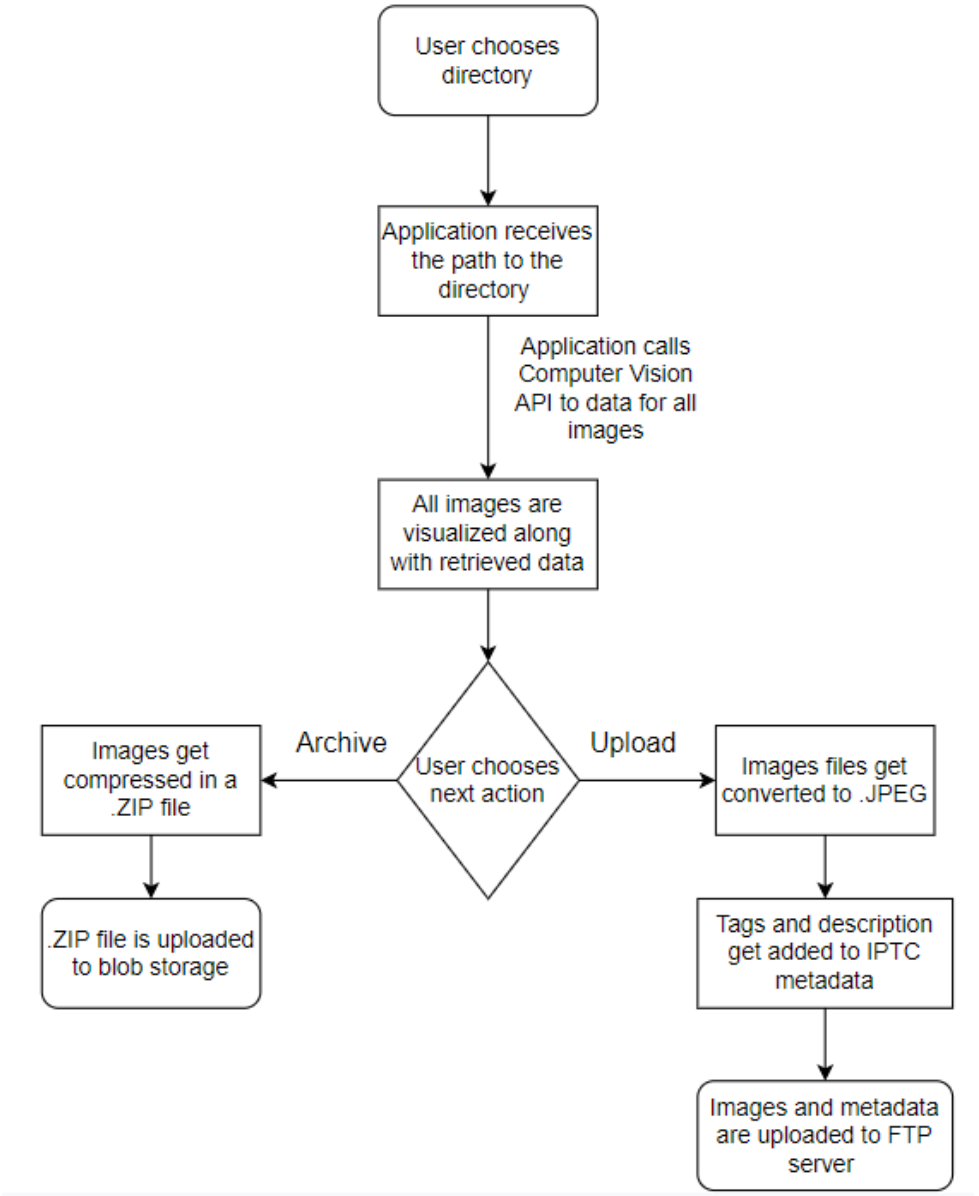
Фиг. 2.2. Структура на работа на Apache Hadoop YARN

Data Lake се фокусира върху съхраняването на огромни количества данни и тяхното анализиране. Една от възможните опции за облачно хранилище е Azure Blob Storage, което набляга на пазенето на файлове. Тези файлове се записват под формата на бинарни данни [4].

2.3. Описание на алгоритъма

При стартиране на приложението потребителят може да избере директория, от която да се визуализират снимките. Със заявка към Computer Vision се получават ключовите думи и описанието. Всичко се зарежда в прозорец, като навигирането между фотографи се случва с бутони. След това може да се архивират файловете или да се качат към сървър. Архивирането се осъществява като се обходи директорията и всички снимки се добавят в компресиран ZIP файл, който се изпраща към създадено от мен блоб хранилище. Ако потрбителят избере някоя от

опциите за качване се прави опит за свързване със сървъра, който при успех позволява да изпратим снимките и създадените към тях файлове, съдържащи метаданните. Тъй като част от фотобанките работят само с JPEG файлове, включително и двете, използвани в проекта, преди самото качване директорията със снимките се обхожда и всички файлове на изображения се модифицират да са с JPEG разширение.



Фигура 2.1. Блок схема на принципа на работа

3. ТРЕТА ГЛАВА – Имплементация на приложение за автоматично тагване и качване във фотобанка

3.1. Структура на проекта

Приложението може да се раздели на четири главни части:

- Графичен потребителски интерфейс
- Извличане на метаданни
- Архивиране
- Качване към FTP сървър

3.2. Графичен потребителски интерфейс

За реализирането на тази част от приложението използвах Python рамката Tkinter. Първата стъпка е инициализиране и пускане на прозорец, което се случва във файла mainWindow.py:

```
root = Tk()
root.title('Application')
root.state("zoomed")
```

```
dir_button = Button(root, text="Choose directory", command=lambda: getImages(filedialog.askdirectory()))
dir_button.place(anchor=CENTER, relx=.5, rely=.5)

root.mainloop()
```

В този код се създава инстанция на Tk, което служи като основа за поставяне на елементи, и след което се добавя бутон за избиране на директория. Пътят до тази директория се подава на функцията getImages(path_param), която приема път до директория и се разделя на три части:

```
def getImages(path_param):
    logging.basicConfig(level=logging.DEBUG, filename='logs.log', format='%(asctime)s %(levelname)s:%(message)s')
    logger = logging.getLogger(__name__)
    logger.info("Visualizing directory: " + path_param)

    image_paths = []
    images = []
    tag_arr = []

    extensions = ('.JPG', '.jpg', '.JPEG', '.jpeg')
    for file in os.listdir(path_param):
        if file.endswith(extensions):
            path = os.path.join(path_param, file)
            # Azure API only works with images 4MB and under
            if os.path.getsize(path) / (1024 * 1024) < 4:
                image_paths.append(path)

    # Resizes photos for consistent image display
    for image in image_paths:
        temp_image = PIL.Image.open(image)
        temp_image = temp_image.resize((500, 500), PIL.Image.ANTIALIAS)
        temp_resized = ImageTk.PhotoImage(temp_image)
        images.append(temp_resized)
```

Първата част на функцията обикаля подадения път и намира всички изображения. Заради ограниченията на Computer Vision само снимките по-малки от 4MB могат да се подават за получаването на метаданните, съответно само те ще бъдат използвани. Тъй като повечето фотографи са с прекалено голям размер, за да се визуализират удобно преди добавянето на екрана се смаляват до размер 500x500 пиксела.

```
# Adds frames for structured display of elements
image_frame = Frame(root, width=700, height=700)
image_frame.place(anchor=CENTER, relx=.4, rely=.5)
tag_frame = Frame(root)
tag_frame.place(anchor=CENTER, relx=.65, rely=.5)

tag_arr = getTagLabels(image_paths[0], tag_frame)
for g in range(0, len(tag_arr)):
    tag_arr[g].grid(row=g, column=0)

image_label = Label(image_frame, image=images[0])
image_label.grid(row=0, column=0, columnspan=3)

caption_label = Label(image_frame, text=getCaption(image_paths[0]))
caption_label.grid(row=1, column=1)
```

Във втората част се добавят рамки, за да може елементите да не се препокриват и да си пречат. Първата снимка от масива се слага в съответната рамка заедно с ключовите думи и описанието, чието

извличане ще бъде описано в точка 3.3. Функцията getTagLabels се намира във файла getImageData.py и връща всички ключови думи под формата на етикети, които могат да бъдат сложени на екрана.

```
button_back = Button(image_frame, text="<<", command=cycleImages, state=DISABLED)
button_back.grid(row=1, column=0)
button_forward = Button(image_frame, text=">>", command=lambda: cycleImages(2,
                                                                             caption_label,
                                                                             image_label,
                                                                             image_frame,
                                                                             images,
                                                                             image_paths,
                                                                             tag_frame,
                                                                             tag_arr))
button_forward.grid(row=1, column=2)

archive_button = Button(tag_frame, text="Archive directory", command=lambda: archive(path_param))
archive_button.grid(row=15)

upload_zoonar_button = Button(tag_frame, text="Upload to Zoonar",
                              command=lambda: logInTopLevel(path_param, 'ftp.zoonar.com'))
upload_zoonar_button.grid(row=16)

upload_alamy_button = Button(tag_frame, text="Upload to Alamy",
                             command=lambda: logInTopLevel(path_param, 'upload.alamy.com'))
upload_alamy_button.grid(row=17)

dir_button.destroy()
```

Последната част добавя бутони за навигиране между снимките, както и за архивиране и качване.

3.2.1. Функционалност на бутоните за сменяне между снимките

```
def cycleImages(image_number, caption_label, image_label, image_frame, images, image_paths, tag_frame, tag_arr):
    # Delete old tags
    for i in range(0, len(tag_arr)):
        tag_arr[i].grid_forget()

    image_label.grid_forget()
    image_label = Label(image_frame, image=images[image_number - 1])
    image_label.grid(row=0, column=0, columnspan=3)

    caption_label.grid_forget()
    caption_label = Label(image_frame, text=getCaption(image_paths[image_number - 1]))
    caption_label.grid(row=1, column=1)

    # Display new tags
    tag_arr = getTagLabels(image_paths[image_number - 1], tag_frame)
    for j in range(0, len(tag_arr)):
        tag_arr[j].grid(row=j, column=0)
```

```
button_back = Button(image_frame, text="<<", command=lambda: cycleImages(image_number - 1,
                                                                          caption_label,
                                                                          image_label,
                                                                          image_frame,
                                                                          images,
                                                                          image_paths,
                                                                          tag_frame,
                                                                          tag_arr))
button_forward = Button(image_frame, text=">>", command=lambda: cycleImages(image_number + 1,
                                                                             caption_label,
                                                                             image_label,
                                                                             image_frame,
                                                                             images,
                                                                             image_paths,
                                                                             tag_frame,
                                                                             tag_arr))

if image_number == len(images):
    button_forward = Button(image_frame, text=">>", state=DISABLED)

button_back.grid(row=1, column=0)
button_forward.grid(row=1, column=2)
```

Извикването на cycleImages, която се намира във файла buttonFunctions.py, визуализира снимката с подадения индекс, което е параметърът image_number, и заличава предишната информация от екрана.

3.2.2. Прозорец за архивиране

```
def archive(path):
    archive_window = Toplevel(height=300, width=300)

    zip_name_label = Label(archive_window, text="What name should the directory be archived under? (Please separate "
                                                "words with underscores, not spaces)")
    zip_name_label.pack()
    zip_name_input = Entry(archive_window)
    zip_name_input.pack()

    submit_button = Button(archive_window, text="Submit", command=lambda: uploadBlob(path, zip_name_input.get()))
    submit_button.pack()
```

Натискането на бутона с текст „Archive” извиква функцията archive(path), също намираща се във файла buttonFunctions.py и приемаща път към директория като параметър. Изкарва нов прозорец на екрана, където потребителят може да въведе името, под което да се архивира директорията, в която приложението работи в момента.

3.3. Извличане на метаданни

Всички функции за извличане на данни се намират във файла getImageData.py. Преди функциите е създаден клас Tag, в чийто инстанции се записват имената на таговете, както и увереността на Computer Vision дали думата се отнася за снимката.

```
def getTags(path):
    decrypted_data = str(decryptKeys())
    decrypted_data.replace("'", '"')

    result = json.loads(decrypted_data)
    subscription_key = result["subscription"]

    with open('sas_keys.json', 'w') as file:
        file.write(decrypted_data)
        file.close()

    encryptKeys()

    endpoint = 'https://cvazureapi.cognitiveservices.azure.com/'
    analyze_url = endpoint + "vision/v3.2/analyze?"

    headers = {'Ocp-Apim-Subscription-Key': subscription_key, 'Content-Type': 'application/octet-stream'}
    params_tags = {'visualFeatures': 'Tags'}
    with open(path, 'rb') as image:
        data = image.read()
```

```
# Connect to Computer Vision API and get tags
response_tags = requests.post(analyze_url, headers=headers, params=params_tags, data=data)
tags = response_tags.json()

tag_arr = []
for i in range(0, len(tags["tags"])):
    temp_tag = Tag(tags["tags"][i]["name"], tags["tags"][i]["confidence"])
    if temp_tag.confidence > 0.5:
        tag_arr.append(temp_tag)

return tag_arr
```

Във функцията getTags(path) се осъществява връзка с Computer Vision с помощта на POST заявка. В резултатът от тази заявка могат да се намерят ключовите думи за снимката, чийто път е подаден като аргумент.

```python
def getCaption(path):
    decrypted_data = str(decrypt_keys())
    decrypted_data.replace("'", '"')

    result = json.loads(decrypted_data)
    subscription_key = result["subscription"]

    with open('sas_keys.json', 'w') as f:
        f.write(decrypted_data)
        f.close()

    encrypt_keys()

    endpoint = 'https://cvazureapi.cognitiveservices.azure.com/'
    analyze_url = endpoint + "vision/v3.2/analyze?"

    headers = {'Ocp-Apim-Subscription-Key': subscription_key, 'Content-Type': 'application/octet-stream'}
    params_tags = {'visualFeatures': 'Description'}
    with open(path, 'rb') as image:
        data = image.read()

    # Connect to Computer Vision API and get tags
    response_captions = requests.post(analyze_url, headers=headers, params=params_tags, data=data)
    caption = response_captions.json()
    return caption["description"]["captions"][0]["text"]
```

getCaption(path) е близъко по логика на getTags(path), като главната разлика е какво се иска и как се връща крайният резултат. Когато се получи отговор от заявката за вземане на описание всъщност е масив, в който се съдържат няколко възможни описания за снимката. Тъй като са подредени по увереност, а не по азбучен ред, може да се вземе първото възможно описание.

Използването на услугите на Azure става с помощта на ключ, който идва с абонамент. Един от начините за позволяване на достъп до профил, който вече има абонамент е със Shared Access Signature, или накратко SAS ключ. За да се избегне злоупотреба от потребител с този ключ, както и с връзката към блоб хранилището, те са записани в JSON формат и са криптирани. Съхраняват се във файла sas_keys.json.

3.3.1. Криптиране и декриптиране на данни

Защитата на важните данни е направена с помощта на библиотеката Cryptography. От нея се извлича Fernet, което предоставя симетрично криптиране на данни. Това означава, че с помощта на специален ключ може да се манипулират данните, като при всяко криптиране и декриптиране ще получава един и същи резултат всеки път[5]. Служещите за това функции се намират в encrypt.py.

```
def encryptKeys():
    key = Fernet.generate_key()
    with open('key.key', 'wb') as file:
        file.write(key)
        file.close()

    with open('key.key', 'rb') as fkey:
        key = fkey.read()
        fkey.close()

    with open('sas_keys.json', 'rb') as file:
        data = file.read()
        file.close()

    fernet = Fernet(key)
    encrypted = fernet.encrypt(data)

    with open('sas_keys.json', 'wb') as file:
        file.write(encrypted)
        file.close()
```

encryptKeys() създава нов уникален ключ всеки път когато се извика, като въпросният ключ се записва във файла key.key. След като прочете данните от sas_keys.json криптира съдържанието и го записва на мястото на съществуващия текст.

```
def decryptKeys():
    with open('key.key', 'rb') as fkey:
        key = fkey.read()
        fkey.close()

    with open('sas_keys.json', 'rb') as file:
        encrypted_data = file.read()
        file.close()

    fernet = Fernet(key)
    return fernet.decrypt(encrypted_data).decode()
```

За декриптирането на файла се извиква decryptKeys(), което връща оригиналния текст във формата на String променлива.

3.4. Архивиране

Една функционалност, която липсва на продуктите в точка 1.1 е опцията за съхраняване на снимките в облачно хранилище. Възможността потребителите да подсигурят файловете си направо от приложението, което използват за качване във фотобанка, е по-удобна от това да търсят алтернативни варианти. С помощта на облачните хранилища на Azure това е възможно. Функцията, реализираща архивирането, е uploadBlob(path, zip_name) и се намира във файла uploadToBlob.py.

```
def uploadBlob(path, zip_name):
    decrypted_data = str(decryptKeys())
    decrypted_data.replace("'", '"')

    result = json.loads(decrypted_data)
    connection_string = result["connection"]

    with open('sas_keys.json', 'w') as file:
        file.write(decrypted_data)
        file.close()

    encryptKeys()

    blob_container = "blob"

    zipFiles(path, zip_name)
    file_name = zip_name+'.zip'
    blob_service_client = BlobServiceClient.from_connection_string(connection_string)
    blob_client = blob_service_client.get_blob_client(container=blob_container, blob=file_name)
```

Библиотеката azure позволява достъпа до блоб хранилище, в което да се качва компресиран файл, съдържащ всички снимки от избраната при отварянето на приложението директория. Тъй като се изпраща само един файл няма проблем да се добавят и снимките, чиито размер е повече от 4MB. Името, избрано през прозореца в точка 3.2.2., се подава на функцията за компресиране и след установяване на връзка с хранилището новосъзданеният файл се изпраща.

3.4.1. Компресиране на файлове

С цел избягването на безразборното архивиране на множество снимки реших, че компресирането на изображенията в дадена директория е по-добър подход. По този начин възстановяването на файловете е по-бързо и лесно. Функцията за компресиране е zipFiles(path, name) - минава през подадения път и добавя изображенията от него в ZIP файл, който може да се намери в същата директория.

```
def zipFiles(path, name):
    extensions = ('.JPG', '.jpg', '.JPEG', '.jpeg', '.PNG', '.png')

    with ZipFile(path+'/'+name+'.zip', 'w') as zip_object:
        for file in os.listdir(path):
            if file.endswith(extensions):
                file_path = os.path.join(path, file)
                zip_object.write(file_path, basename(file_path))
    zip_object.close()
```

3.5. Качване към FTP сървър

Повечето фотобанки имат критерии, по които определят кой може да качва снимки. Най-често срещаният подход е преди да се качват файлове потребителят да предостави портфолио. Ако отговаря на изискванията профилът получава одобрение и публикуваните снимки не подлежат на проверка, освен ако съдържанието не нарушава правилата. За реализирането на проекта беше нужно да се намерят платформи, които не

изискват предварително одобрение, а разглеждат снимките една по една. Освен това тези фотобанки трябва да поддържат качване през FTP. Двете платформи, използвани в проекта, са Zoonar и Alamy.

Изпращането към FTP сървърите се реализира в upload(username_input, password_input, path, server_address), което се намира във файла upload.py.

```
def upload(username_input, password_input, path, server_address):
    logging.basicConfig(level=logging.DEBUG, filename='logs.log', format='%(asctime)s %(levelname)s:%(message)s')
    logger = logging.getLogger(__name__)

    # Connect to FTP server
    session = ftplib.FTP(server_address)
    session.login(username_input.get(), password_input.get())

    if server_address == 'upload.alamy.com':
        session.cwd('/Stock')
```

В началото на функцията се инициализира връзка със сървъра. В зависимост от това кой от двата сайта се сменя директорията, в която се изпращат снимките, защото при Alamy има няколко възможни категории.

```
logger.info("Image upload started")
try:
    # Gets images from directory
    img_extensions = ('.JPG', '.jpg', '.PNG', '.png')
    for file in os.listdir(path):
        if file.endswith(img_extensions):
            filename, ext = os.path.splitext(os.path.join(path, file))
            os.rename(os.path.join(path, file), filename+'.JPEG')

    for file in os.listdir(path):
        if file.endswith(('.JPEG', '.jpeg')):
            img_path = os.path.join(path, file)

            # Azure API only works with images 4MB and under
            if os.path.getsize(img_path) / (1024 * 1024) < 4:
                addIPTCInfo(img_path)
                session.storbinary('STOR ' + file, open(img_path, 'rb'))
except EOFError:
    logger.warning("Image upload failed")
finally:
    logger.debug("Image upload done")
```

При качването на снимките се променят файловите разширения, причината за което е обяснена в точка 2.1., както и добавянето на

метаданни към изображенията, алгоритъмът за което се намира в точка 3.5.1. Всички снимки с размер над 4MB се изключват от качването, защото няма как да получат ключови думи или описание. Тъй като библиотеката ftplib предлага изпращане само под бинарна форма или тази на текст[7], методът storbinary() е използван за предаването към сървъра.

```
# Upload IPTC data to FTP server
logger.info("Metadata upload started")
try:
    iptc_extensions = ('.JPEG~', '.jpeg~')
    for file in os.listdir(path):
        if file.endswith(iptc_extensions):
            iptc_path = os.path.join(path, file)
            session.storbinary('STOR ' + file, open(iptc_path, 'rb'))
except EOFError:
    logger.warning("Metadata upload failed")
finally:
    logger.debug("Metadata upload done")
    messagebox.showinfo("Done", "Upload done!")

session.quit()
```

В последната част на функцията се изпращат метаданните, които се записват във формата на JPEG~ файл.

3.5.1. Добавяне на метаданни към изображения

Всички фотобанки се нуждаят от ключови думи и описания на снимките, по този начин се улеснява намирането и продаването на изображения. При качването на файлове през сайтовете или приложенията на фотобанките включването на тагове е лесно, но нещата се усложняват ако вместо това се избере изпращане по FTP сървър. Най-лесното решение на този проблем е добавянето на метаданни към снимките, предвидени за изпращане.

```python
def addIPTCInfo(img_path):
    temp_tag_arr = []

    tag_arr = getTags(img_path)
    for t in tag_arr:
        temp_tag_arr.append(t.name)

    img_data = IPTCInfo(img_path, force=True)
    img_data['keywords'] = []
    img_data['keywords'] = temp_tag_arr

    img_data['caption/abstract'] = []
    img_data['caption/abstract'] = [getCaption(img_path), "stock photo"]

    img_data.save()
```

В getImageData.py се намира и функцията addIPTCInfo(img_path), която добавя нова информация към файла, съдържащ метаданни за снимката. Ако няма такъв файл ще се създаде нов, в който да се налеят данните.

4. ГЛАВА ЧЕТВЪРТА – Ръководство на потребителя

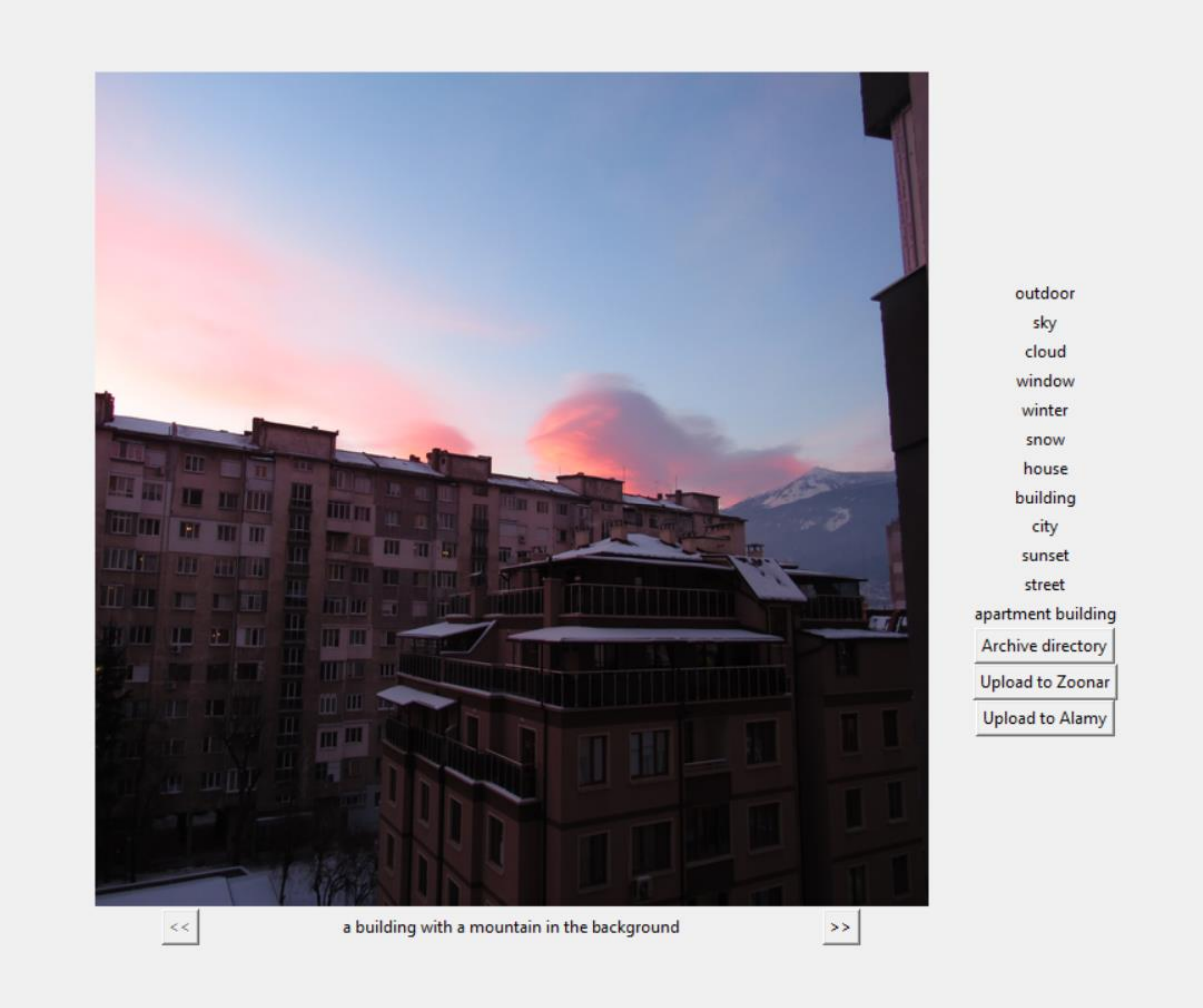
4.1. Инсталиране на библиотеки

За да функционира проектът се нуждае от версия на питон 3.9, както и няколко външни библиотеки, които могат да се инсталират по следния начин:

- pip install IPTCInfo3, което е нужно за промяната и добавянето на метаданни
- pip install azure, което е нужно за работата с Azure Blob Storage
- pip install requests, което е нужно за комуникация с Azure Computer Vision
- pip install Pillow, което е нужно за визуализиране и манипулиране на изображения
- pip install cryptography, което е нужно за разчитане и криптиране на SAS ключа и адреса на хранилището

4.2. Използване на приложението

След стартиране на приложението се отваря Tk прозорец, където може да се избере директорията, с която ще се работи. Това се случва с натискането на бутонът с текст “Choose directory”. Когато действието приключи се появява първата снимка, заедно с описание, ключови думи и бутони за операция.



Фиг. 4.1. Начален екран след зареждане на директория

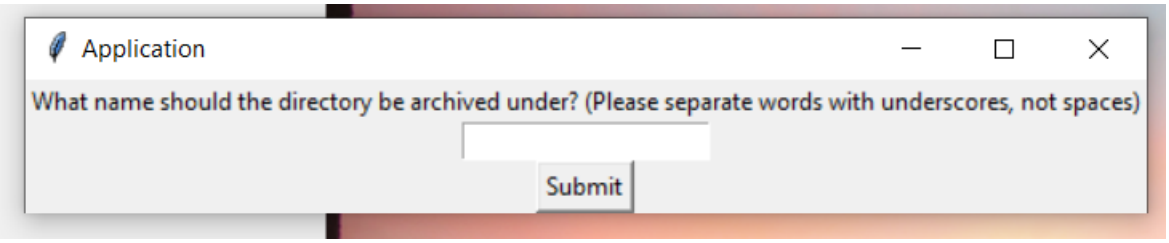
4.2.1. Навигиране между снимки

За да се минава през изображенията се използват бутоните от двете страни на описанието, маркирани с “<<” и “>>”. Ако сегашната снимка е първата в заредената директория, бутонът за минаване назад (“<<”) е

блокиран. Същото важи и в случая, че се разглежда последното възможно изображение, но тогава се блокира минаването напред.

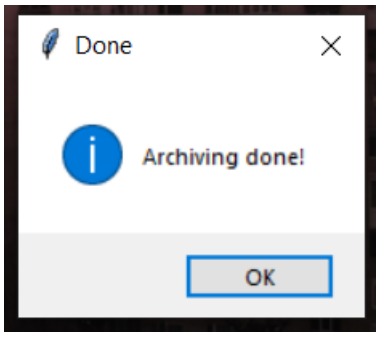
4.2.2. Архивиране

Точно под таговете за снимката седи бутонът “Archive directory”. При неговото натискане на екранът се появява прозорецът за избиране на името на архива.



Фиг. 4.2. Прозорец за именуване на архив

Натискането на бутонът “Submit” инициализира архивирането. Необходимо е малко да се изчака, за да се извърши функцията. При успешно приключване се появява уведомление, след което потребителят може спокойно да затвори всички прозорци, с които няма да работи повече.

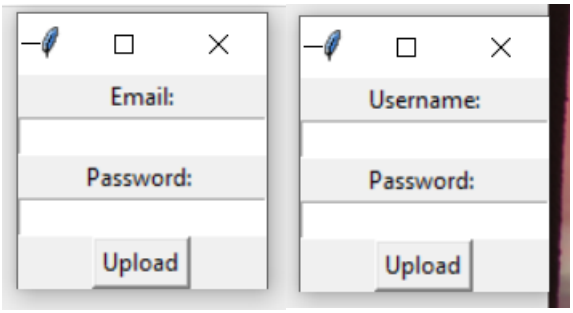


Фиг. 4.3. Уведомление за успешно архивиране

4.2.3. Качване към FTP сървър

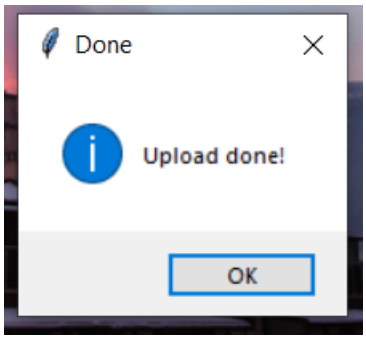
Освен архивиране, потребителят има възможност да качи снимките и метаданните от директорията към FTP сървъра на Zoonar или Alamy. С

зависимост от избраната платформа ще се покаже различен прозорец за влизане в системата.



Фиг. 4.4. Прозорец за влизане в Alamy (ляво) и Zoonar (дясно)

Процесът по пращане на файловете и добавянето на метаданните отнема повече време от архивирането, тъй като е по-сложен. Възможно е приложението да отиде в режим на Not Responding, но това не е притеснително. Ако потрбителят изчака функцията да се изпълни докрай на екрана ще се появи известие.



Фиг. 4.5. Известие за успешно качване на файловете

Заключение

Едно от нещата, които се нуждаят от подобрение, е съхраняването на ключовете. Използването само на криптиране, и то такова от библиотека, не пази данните достаъчно добре. Този проблем може да бъде опрваен с помощта на Azure Key Vault, което е облачно хранилище, специално направено за пазенето на данни.

Използвана литература

[1] Tamás Gulácsi, IPTCInfo Documentation, https://github.com/20minutes/iptcinfo/blob/master/iptcinfo.py, June 2012

[2] John E. Grayson, Python and Tkinter Programming, Manning Publications, January 2000, p. 14

[3] What is Computer Vision?, https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/overview

[4] Introduction to Azure Blob storage, https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction

[5] Symmetric Cryptography, https://www.sciencedirect.com/topics/computer-science/symmetric-cryptography

[6] What is IPTC metadata? Everything you need to know, https://smartframe.io/blog/what-is-iptc-metadata-everything-you-need-to-know/

[7] ftplib – FTP protocol client, https://docs.python.org/3/library/ftplib.html

[8] What is Image Analysis?, https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/overview-image-analysis

[9] Object detection - Computer Vision, https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/concept-object-detection

[10] Microsoft Azure Data Lake, https://www.techtarget.com/searchcloudcomputing/definition/Microsoft-Azure-Data-Lake

[11] What Is Hadoop Yarn Architecture & It's Components, https://www.upgrad.com/blog/what-is-hadoop-yarn-architecture-its-components/