

ДОКУМЕНТАЦИЯ РАЗРАБОТЧИКА ПРИЛОЖЕНИЯ “PROFILE ANALYZER”

Исходные коды приложения лежат в репозитории https://github.com/neonestea/profile_analyzer. Для того, чтобы выкачать проект, необходимо в терминале ввести строку:

```
git clone  
https://github.com/neonestea/profile\_analyzer.git
```

Предварительная подготовка

Разместите архив `saved_models.zip` на диске `C:\` и разархивируйте во избежание ошибок с русскими символами в пути к файлу.

В случае, если не хватает нужных пакетов, их необходимо дополнительно установить при помощи `pip` (или другого менеджера):

```
>pip install django  
>pip install django-crispy-forms  
>pip install crispy_bootstrap4  
>pip install requests  
>pip install vk_api  
>pip install "pandas<2.0.0"  
>pip install nltk  
>pip install pymystem3  
>pip install gensim  
>pip install emoji  
>pip install autocorrect  
>pip install razdel  
>pip install fasttext-wheel  
>pip install category_encoders  
>pip install scikit-learn==1.2.2  
>pip install catboost
```

Запуск приложения

Перед запуском приложения нужно подготовить и применить миграции для базы данных.

```
>python manage.py makemigrations
```

```
>python manage.py migrate
```

Для запуска необходимо выполнить:

```
>python manage.py runserver
```

При успешном запуске приложения в консоль выведется примерно такой код:

```
System check identified 1 issue (0 silenced).
```

```
March 25, 2024 - 14:16:31
```

```
Django version 5.0.3, using settings  
'web_project.settings'
```

```
Starting development server at http://127.0.0.1:8000/
```

```
Quit the server with CTRL-BREAK.
```

Архитектура

Данный проект создан на основе фреймворка Django 5.0.3.
Документация сгенерирована при помощи Sphinx 7.2.6.

Root

root

```
root.db.sqlite3 # database
root.manage # main function
```

Модуль web_project

web_project

```
web_project.urls #main urls
web_project.asgi #settings of Django project
web_project.wsgi #settings of Django project
web_project.settings #settings of Django project
```

Модуль data_collector

data_collector

```
data_collector.custom_logger
data_collector.forms
data_collector.models
data_collector.result_builder
data_collector.urls
data_collector.views
data_collector.vk api processor
data_collector.static.data_collector #data_collector
templates
```

custom_logger.py

Logger

`data_collector.custom_logger.configure_logger()`

Creates new logger.

Returns:

Returns new logger.

Return type:

logger

`data_collector.custom_logger.set_stdout_handler(custom_logger)`

Configures stdout handler for logger.

Parameters:

custom_logger (*logger*) – Input Logger.

Returns:

Returns logger.

Return type:

logger

forms.py

Data collector forms

`class data_collector.forms.SearchForm(**kwargs)`

Form to create search

Form fields:

- `name`: Name (**CharField**)
- `link`: Link (**CharField**)

`__init__(**kwargs)`

property **media**

Return all media required to render the widgets on this form.

`save(commit=True)`

Save this form's self.instance object if commit=True. Otherwise, add a `save_m2m()` method to the form which can be called after the instance is saved manually at a later time. Return the model instance.

`class data_collector.forms.SearchFormUpdate(data=None, files=None, auto_id='id_%s', prefix=None, initial=None, error_class=<class 'django.forms.utils.ErrorList'>, label_suffix=None, empty_permitted=False, instance=None, use_required_attribute=None, renderer=None)`

Form to update search

Form fields:

- `name`: Name (**CharField**)

property **media**

Return all media required to render the widgets on this form.

`data_collector.forms.validate_name(obj)`

Validates name

models.py

Data collector models

`class data_collector.models.ProfileInfo(*args, **kwargs)`

ProfileInfo model

Parameters:

- **id** (*UUIDField*) – Primary key: Id
- **link** (*CharField*) – Link
- **first_name** (*CharField*) – First Name
- **last_name** (*CharField*) – Last Name
- **bdate** (*IntegerField*) – Birth Date
- **interests** (*TextField*) – Interests
- **books** (*TextField*) – Books
- **tv** (*TextField*) – TV
- **games** (*TextField*) – Games
- **movies** (*TextField*) – Movies
- **activities** (*TextField*) – Activities
- **music** (*TextField*) – Music
- **status** (*TextField*) – Status
- **military** (*IntegerField*) – Military
- **university_name** (*TextField*) – University Name
- **faculty** (*TextField*) – Faculty
- **home_town** (*CharField*) – Home Town
- **relation** (*TextField*) – Relation
- **sex** (*IntegerField*) – Sex
- **about** (*TextField*) – About
- **country** (*CharField*) – Country
- **city** (*CharField*) – City
- **friends_count** (*IntegerField*) – Friends count
- **followers_count** (*IntegerField*) – Followers count
- **groups** (*IntegerField*) – Groups count
- **group_infos** (*TextField*) – Groups
- **posts** (*TextField*) – Posts
- **comments** (*TextField*) – Comments
- **comments_of_other_users** (*TextField*) – Comments of other users

- **alcohol** (*CharField*) – Alcohol
- **life_main** (*CharField*) – Life Main
- **people_main** (*CharField*) – People Main
- **political** (*CharField*) – Political
- **religion** (*CharField*) – Religion
- **smoking** (*CharField*) – Smoking
- **photos_count** (*IntegerField*) – Photos count
- **posts_count** (*IntegerField*) – Posts count

Relationship fields:

Parameters:

connected_search (**ForeignKey** to **Search**) – Connected search (related name: **profileinfo**)

exception **DoesNotExist**

exception **MultipleObjectsReturned**

about

Type: **TextField**

About

activities

Type: **TextField**

Activities

alcohol

Type: **CharField**

Alcohol

bdate

Type: **IntegerField**

Birth Date

books

Type: **TextField**

Books

city

Type: **CharField**

City

comments

Type: **TextField**

Comments

comments_of_other_users

Type: **TextField**

Comments of other users

connected_search

Type: **ForeignKey** to **Search**

Connected search (related name: **profileinfo**)

connected_search_id

Internal field, use **connected_search** instead.

country

Type: **CharField**

Country

faculty

Type: **TextField**

Faculty

first_name

Type: **CharField**

First Name

followers_count

Type: **IntegerField**

Followers count

friends_count

Type: **IntegerField**

Friends count

games

Type: **TextField**

Games

group_infos

Type: **TextField**

Groups

groups

Type: **IntegerField**

Groups count

home_town

Type: **CharField**

Home Town

id

Type: **UUIDField**

Primary key: Id

interests

Type: **TextField**

Interests

last_name

Type: **CharField**

Last Name

life_main

Type: **CharField**

Life Main

link

Type: **CharField**

Link

military

Type: **IntegerField**

Military

movies

Type: **TextField**

Movies

music

Type: **TextField**

Music

objects = *<django.db.models.Manager object>*

people_main

Type: **CharField**

People Main

photos_count

Type: **IntegerField**

Photos count

political

Type: **CharField**

Political

posts

Type: **TextField**

Posts

posts_count

Type: **IntegerField**

Posts count

relation

Type: **TextField**

Relation

religion

Type: **CharField**

Religion

sex

Type: **IntegerField**

Sex

smoking

Type: **CharField**

Smoking

status

Type: **TextField**

Status

tv

Type: **TextField**

TV

university_name

Type: **TextField**

University Name

`class data_collector.models.Result(*args, **kwargs)`

Result model

Parameters:

- **id** (*UUIDField*) – Primary key: Id
- **first_name** (*CharField*) – First Name
- **last_name** (*CharField*) – Last Name

- **age** (*CharField*) – Age(s)
- **city** (*TextField*) – City(s)
- **country** (*TextField*) – Country(s)
- **open** (*CharField*) – Openness
- **cons** (*CharField*) – Cons
- **neur** (*CharField*) – Neur
- **agree** (*CharField*) – Agree
- **extr** (*CharField*) – Extr
- **interests** (*TextField*) – Interests

Relationship fields:

Parameters:

connected_search (**ForeignKey** to **Search**) – Connected search (related name: **result**)

exception **DoesNotExist**

exception **MultipleObjectsReturned**

age

Type: **CharField**

Age(s)

agree

Type: **CharField**

Agree

city

Type: **TextField**

City(s)

connected_search

Type: **ForeignKey** to **Search**

Connected search (related name: **result**)

connected_search_id

Internal field, use **connected_search** instead.

cons

Type: **CharField**

Cons

country

Type: **TextField**

Country(s)

extr

Type: **CharField**

Extr

first_name

Type: **CharField**

First Name

id

Type: **UUIDField**

Primary key: Id

interests

Type: **TextField**

Interests

last_name

Type: **CharField**

Last Name

neur

Type: **CharField**

Neur

objects = <django.db.models.Manager object>

open

Type: **CharField**

Openness

class data_collector.models.**Search**(*args, **kwargs)

Search model

Parameters:

- **id** (*UUIDField*) – Primary key: Id
- **name** (*CharField*) – Name
- **link** (*TextField*) – Link
- **date** (*DateTimeField*) – Date of search
- **ready** (*IntegerField*) – Ready

Relationship fields:

Parameters:

created_by (**ForeignKey** to **User**) – Created by (related name: **search**)

Reverse relationships:

Parameters:

- **profileinfo** (Reverse **ForeignKey** from **ProfileInfo**) – All ProfileInfos of this Search (related name of **connected_search**)
- **result** (Reverse **ForeignKey** from **Result**) – All results of this Search (related name of **connected_search**)

exception **DoesNotExist**

exception **MultipleObjectsReturned**

created_by

Type: **ForeignKey** to **User**

Created by (related name: **search**)

created_by_id

Internal field, use **created_by** instead.

date

Type: **DateTimeField**

Date of search

get_absolute_url()

get_next_by_date(**, field=<django.db.models.DateTimeField: date>, is_next=True, **kwargs*)

Finds next instance based on **date**. See **get_next_by_FOO()** for more information.

get_previous_by_date(**, field=<django.db.models.DateTimeField: date>, is_next=False, **kwargs*)

Finds previous instance based on **date**. See **get_previous_by_FOO()** for more information.

id

Type: **UUIDField**

Primary key: **Id**

link

Type: **TextField**

Link

link_lines()

name

Type: **CharField**

Name

objects = *<django.db.models.Manager object>*

profileinfo_set

Type: Reverse **ForeignKey** from **ProfileInfo**

All ProfileInfos of this Search (related name of **connected_search**)

ready

Type: **IntegerField**

Ready

result_set

Type: Reverse **ForeignKey** from **Result**

All results of this Search (related name of **connected_search**)

save_model(*request, obj, form, change*)

result_builder.py

Module to build result

data_collector.result_builder.**build_result**(*search, first_name, last_name, age, country, city, open, cons, neur, agree, extr, interests*)

Builds result of the search.

Parameters:

- **search** (*Search*) – Search to show.
- **first_name** (*str*) – First name of the person.
- **last_name** (*str*) – Last name of the person.
- **age** (*str*) – Age or ages.
- **country** (*str*) – Country or countries.
- **city** (*str*) – City or cities.
- **open** (*str*) – Openness or openness percentage.
- **cons** (*str*) – Conscientiousness or conscientiousness percentage.
- **neur** (*str*) – Neurotism or neurotism percentage.
- **agree** (*str*) – Agreeableness or agreeableness percentage.
- **extr** (*str*) – Extraversion or extraversion percentage.
- **interests** (*str*) – Interests.

data_collector.result_builder.**logger** = <Logger data_collector.custom_logger (DEBUG)>

logger

urls.py

Data collector urls

```
data_collector.urls.urlpatterns = [<URLPattern '' [name='search_home']>, <URLPattern 'create' [name='create_search']>, <URLPattern '<uuid:pk>' [name='search_detail']>, <URLPattern '<uuid:pk>/delete' [name='search_delete']>, <URLPattern '<uuid:pk>/update' [name='search_update']>, <URLPattern '<uuid:pk>/info' [name='profile_info']>, <URLPattern '<uuid:pk>/result' [name='result']>]
```

urls for redirection

```
[<URLPattern '' [name='search_home']>,<URLPattern 'create' [name='create_search']>,<URLPattern '<uuid:pk>' [name='search_detail']>,<URLPattern '<uuid:pk>/delete' [name='search_delete']>,<URLPattern '<uuid:pk>/update' [name='search_update']>,<URLPattern '<uuid:pk>/info' [name='profile_info']>,<URLPattern '<uuid:pk>/result' [name='result']>,]
```

analyzer.py

Module to analyze information

List of preloaded classifiers, vectorizers and encoders.

```
data_collector.analyzer.add_count_sentiment_columns(df)
```

Adds sentiment count columns.

Parameters:

df (*DataFrame*) – Input Dataframe.

Returns:

Returns Dataframe with new columns.

Return type:

DataFrame

```
data_collector.analyzer.analyze(search)
```

Analyze search.

Parameters:

search (*Search*) – Input search.

```
data_collector.analyzer.count_sentiments(texts)
```

Counts sentiment texts.

Parameters:

texts (*list*) – Texts to check sentiments.

Returns:

Returns count of sentiments

Return type:

[tuple](#)

```
data_collector.analyzer.create_line(profile)
```

Creates dictionary from one profile to pass to personality traits classifiers.

Parameters:

profile (*ProfileInfo*) – Profile to create dictionary.

Returns:

Returns dict from profile to add to dataset

Return type:

[dict](#)

```
data_collector.analyzer.encode_with_hashing(x_df, trait)
```

Classifies extraversion.

Parameters:

- **x_df** (*DataFrame*) – Input dataframe.
- **trait** (*str*) – Trait short name.

Returns:

Returns label

Return type:

[int](#)

```
data_collector.analyzer.get_agree(df)
```

Classifies agreeableness.

Parameters:

df (*DataFrame*) – Input dataframe.

Returns:

Returns label

Return type:

[int](#)

`data_collector.analyzer.get_cons(df)`

Classifies cons.

Parameters:

df (*DataFrame*) – Input dataframe.

Returns:

Returns label

Return type:

[int](#)

`data_collector.analyzer.get_extr(df)`

Classifies extraversion.

Parameters:

df (*DataFrame*) – Input dataframe.

Returns:

Returns label

Return type:

[int](#)

`data_collector.analyzer.get_interests(profile)`

Defines interests of a person.

Parameters:

profile (*ProfileInfo*) – Input Profile.

Returns:

Returns interests.

Return type:

[list](#)

`data_collector.analyzer.get_neur(df)`

Classifies neurotism.

Parameters:

df (*DataFrame*) – Input dataframe.

Returns:

Returns label

Return type:

[int](#)

```
data_collector.analyzer.get_open(df)
```

Classifies oppenness.

Parameters:

df (*DataFrame*) – Input dataframe.

Returns:

Returns label

Return type:

[int](#)

```
data_collector.analyzer.get_topic_by_id(id)
```

Returns topic by id.

Parameters:

id (*int*) – Input id.

Returns:

Returns name of topic.

Return type:

[str](#)

```
data_collector.analyzer.logger = <Logger data_collector.custom_logger (DEBUG)>
```

logger

```
data_collector.analyzer.make_prepr(df)
```

Preprocesses texts in dataframe.

Parameters:

df (*DataFrame*) – Input Dataframe.

Returns:

Returns Dataframe with preprocessed texts.

Return type:

DataFrame

`data_collector.analyzer.preprocess_df(new_df)`

Cleanes dataset and processes columns.

Parameters:

new_df (*DataFrame*) – Input Dataframe.

Returns:

Returns Dataframe with new columns.

Return type:

DataFrame

`data_collector.analyzer.preprocess_text(text)`

Preprocesses text.

Parameters:

text (*str*) – Input text.

Returns:

Returns preprocessed text.

Return type:

[str](#)

`data_collector.analyzer.russian_stopwords`

stopwords

`data_collector.analyzer.spell = <autocorrect.Speller object>`

speller

views.py

Data collector views

`class data_collector.views.SearchDeleteView(**kwargs)`

View to delete search

model

alias of **Search**
success_url = '/searches/'
template_name = 'data_collector/delete.html'

class data_collector.views.**SearchDetailView**(**kwargs)
 View for search details

context_object_name = 'search'
model
 alias of **Search**
template_name = 'data_collector/details_view.html'

class data_collector.views.**SearchUpdateView**(**kwargs)
 View to update search

form_class
 alias of **SearchFormUpdate**
model
 alias of **Search**
template_name = 'data_collector/update.html'

data_collector.views.**create_search**(request)
 Creates search

data_collector.views.**profile_info**(request, pk)
 Renders profile info.

Parameters:
pk (*UUID*) – UUID of search.

data_collector.views.**result**(request, pk)
 Renders results.

Parameters:
pk (*UUID*) – UUID of search.

data_collector.views.**search_home**(request)
 Renders search home page

data_collector.views.**update_search**(request, pk)
 Updates search.

Parameters:
pk (*UUID*) – UUID of search.

data_collector.views.**validate_links**(request)
 Validates links in request

vk_api_processor.py

Module to get information by VK API

data_collector.vk_api_processor.**check_group**(*url*, *vk*)

Gets group from VK API by url.

Parameters:

url (*str*) – Input url.

vk (*vk_api*) – Input vk_api session.

Returns:

Returns group from vk api.

Return type:

[dict](#)

data_collector.vk_api_processor.**check_user**(*url*, *vk*)

Gets user from VK API by url.

Parameters:

url (*str*) – Input url.

vk (*vk_api*) – Input vk_api session.

Returns:

Returns user from vk api.

Return type:

[dict](#)

data_collector.vk_api_processor.**date_format** = '%d.%m.%Y'

date formatter

data_collector.vk_api_processor.**get_all_user_comments**(*user_id*, *vk*)

Gets all comments per user.

Parameters:

user_id (*str*) – Input user id.

vk (*vk_api*) – Input vk_api session.

Returns:

Returns user posts, user comments, comments from other users.

Return type:

[tuple](#)

`data_collector.vk_api_processor.get_photo_comments(user_id, vk)`

Gets photo comments of user.

Parameters:

user_id (*str*) – Input user id.

vk (*vk_api*) – Input vk_api session.

Returns:

Returns user photo comments count and comments from other people count.

Return type:

[tuple](#)

`data_collector.vk_api_processor.get_user_followers(user_id, vk)`

Gets user followers count from VK API user id.

Parameters:

user_id (*str*) – Input user id.

vk (*vk_api*) – Input vk_api session.

Returns:

Returns user followers count.

Return type:

[int](#)

`data_collector.vk_api_processor.get_user_friends(user_id, vk)`

Gets user friends count from VK API user id.

Parameters:

user_id (*str*) – Input user id.

vk (*vk_api*) – Input vk_api session.

Returns:

Returns user friends count.

Return type:

int

data_collector.vk_api_processor.**get_user_groups**(*user_id*, *vk*)

Gets user groups and user_groups count from VK API user id.

Parameters:

user_id (*str*) – Input user id.

vk (*vk_api*) – Input vk_api session.

Returns:

Returns user group infos and count.

Return type:

tuple

data_collector.vk_api_processor.**get_user_photos**(*user_id*, *vk*)

Gets number of user photos from VK API user id.

Parameters:

user_id (*str*) – Input user id.

vk (*vk_api*) – Input vk_api session.

Returns:

Returns number of user photos.

Return type:

dict

data_collector.vk_api_processor.**get_user_posts**(*user_id*, *vk*)

Gets number of posts of user.

Parameters:

user_id (*str*) – Input user id.

vk (*vk_api*) – Input vk_api session.

Returns:

Returns number of posts.

Return type:

int

data_collector.vk_api_processor.**get_user_wall**(*user_id*, *vk*)

Gets user wall.

Parameters:

user_id (*str*) – Input user id.

vk (*vk_api*) – Input vk_api session.

Returns:

Returns user posts, user comments, comments from other users.

Return type:

[tuple](#)

data_collector.vk_api_processor.**parse_profile**(*url, search*)

Initiates profile processing.

Parameters:

- **url** (*str*) – Input url.
- **search** (*Search*) – Input Search.

data_collector.vk_api_processor.**parse_profile_info**(*profile, url, search*)

Creates new ProfileInfo.

Parameters:

- **profile** (*dict*) – Input Profile from VK API.
- **url** (*str*) – Input url.
- **search** (*Search*) – Input search.

data_collector.vk_api_processor.**session** = <vk_api.vk_api.VkApi object>

vk api session

data_collector.vk_api_processor.**start_collecting_info**(*search, links*)

Starts collecting info.

Parameters:

- **search** (*Search*) – Input Search.
- **links** (*str*) – Input links as text.

data_collector.vk_api_processor.**get_vk**(*token_id*)

Starts collecting info.

Parameters:

- **token_id** (*int*) – Id of the token to take

Returns:

- Returns vk_api

- **Return type:**
- [vk_api](#)

Модуль main

main

```
main.apps  
main.urls  
main.views  
main.templates.main #main templates  
main.static.main #main static
```

apps.py

Main apps

```
class main.apps.MainConfig(app_name, app_module)  
  
    main module config  
  
    default_auto_field = 'django.db.models.BigAutoField'  
  
    name = 'main'
```

urls.py

Main urls

```
main.urls.urlpatterns = [<URLPattern " [name='home']>, <URLPattern 'about' [name='about']>, <URLPattern 'contacts' [name='contacts']>]
```

main module urls

```
[  
    <URLPattern " [name='home']>,  
    <URLPattern 'about' [name='about']>,  
    <URLPattern 'contacts' [name='contacts']>,  
]
```

views.py

Main views

```
main.views.about(request)
```

Function to open page about project
`main.views.contacts(request)`

Function to open page with contacts
`main.views.index(request)`

Function to open main page

Модуль users

users

users.apps

users.forms

users.urls

users.views

users.templates.users #html templates

apps.py

Users apps

```
class users.apps.UsersConfig(app_name, app_module)
```

Users module config

default_auto_field = 'django.db.models.BigAutoField'

name = 'users'

forms.py

Users forms

```
class users.forms.NewUserForm(*args, **kwargs)
```

Form to create user

Form fields:

- **username**: Username (**CharField**)
- **password1**: Password (**CharField**)
- **password2**: Password confirmation (**CharField**)

property **media**

Return all media required to render the widgets on this form.

save(commit=True)

Save this form's `self.instance` object if `commit=True`. Otherwise, add a `save_m2m()` method to the form which can be called after the instance is saved manually at a later time. Return the model instance.

urls.py

Users urls

```
users.urls.urlpatterns = [<URLPattern 'register' [name='register']>, <URLPattern 'logout' [name='logout']>, <URLPattern 'login' [name='login']>]
```

user urls

```
[  
    <URLPattern 'register' [name='register']>,  
    <URLPattern 'logout' [name='logout']>,  
    <URLPattern 'login' [name='login']>,  
]
```

views.py

Users views

```
users.views.login_request(request)
```

Login view

```
users.views.logout_view(request)
```

```
users.views.profile(request)
```

```
users.views.register_request(request)
```

Register user view