Design & Implementierung eines Echtzeit-Q&A-Systems als Erweiterung des IAmA-Subreddits

(Python) – Dokumentation von REST - Services

Benedikt Hierl

Version 1.0
Sonntag, den 10.07.2016

Table of Contents

Namespace Index

Packages

Here are the packages with brief descriptions (if available):

Class Index

Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

File Index

File List

Here is a list of all files with brief descriptions:

Namespace Documentation

r_rest_Crawl_N_Calculate_Data Namespace Reference

Classes

- class r_rest_Crawl_N_Calculate_Data

Variables

- mongo_db_client_instance = MongoClient('localhost', 27017)
- mongo_db_author_fake_iama_instance = mongo_db_client_instance['fake_iAMA_Reddit_Authors']
- mongo_db_author_fake_iama_collection_names = mongo_db_author_fake_iama_instance.collection_names()
- mongo_db_author_comments_instance = mongo_db_client_instance['fake_iAMA_Reddit_Comments']
- mongo_db_author_comments_collection = mongo_db_author_comments_instance.collection_names()
- reddit_instance = praw.Reddit(user_agent="University_Regensburg_iAMA_Crawler_0.001")
- reddit_submission = None
- int thread_created_utc = 0
- string thread_author = ""
- string thread_title = ""
- int thread_amount_questions = 0
- int thread_amount_unanswered_questions = 0
- int thread_duration = 0
- string thread_id = ""
- int thread_ups = 0
- int thread_downs = 0
- int thread_time_stamp_last_question = 0
- int thread_average_question_score = 0
- int thread_average_reaction_time_host = 0
- int thread_new_question_every_x_sec = 0
- int thread_amount_questions_tier_1 = 0
- int thread_amount_questions_tier_x = 0
- int thread_question_top_score = 0
- int thread_amount_questioners = 0
- list thread_unanswered_questions = []
- list thread_answered_questions = []
- list thread_answers_of_host = []
- list thread_questions_n_answers = []
- list thread_unanswered_questions_converted = []
- list json_object_to_return = []

Variable Documentation

list r_rest_Crawl_N_Calculate_Data.json_object_to_return = []

r_rest_Crawl_N_Calculate_Data.mongo_db_author_comments_collection = mongo_db_author_comments_instance.collection_names()

r_rest_Crawl_N_Calculate_Data.mongo_db_author_comments_instance = mongo_db_client_instance['fake_iAMA_Reddit_Comments']

r_rest_Crawl_N_Calculate_Data.mongo_db_author_fake_iama_collection_names = mongo_db_author_fake_iama_instance.collection_names()

r_rest_Crawl_N_Calculate_Data.mongo_db_author_fake_iama_instance = mongo_db_client_instance['fake_iAMA_Reddit_Authors']

r_rest_Crawl_N_Calculate_Data.mongo_db_client_instance = MongoClient('localhost', 27017)

r_rest_Crawl_N_Calculate_Data.reddit_instance = praw.Reddit(user_agent="University_Regensburg_iAMA_Crawler_0.001")

r_rest_Crawl_N_Calculate_Data.reddit_submission = None

int r_rest_Crawl_N_Calculate_Data.thread_amount_questioners = 0

int r_rest_Crawl_N_Calculate_Data.thread_amount_questions = 0

int r_rest_Crawl_N_Calculate_Data.thread_amount_questions_tier_1 = 0

int r_rest_Crawl_N_Calculate_Data.thread_amount_questions_tier_x = 0

int r_rest_Crawl_N_Calculate_Data.thread_amount_unanswered_questions = 0

list r_rest_Crawl_N_Calculate_Data.thread_answered_questions = []

list r_rest_Crawl_N_Calculate_Data.thread_answers_of_host = []

string r_rest_Crawl_N_Calculate_Data.thread_author = ""

int r_rest_Crawl_N_Calculate_Data.thread_average_question_score = 0

int r_rest_Crawl_N_Calculate_Data.thread_average_reaction_time_host = 0

int r_rest_Crawl_N_Calculate_Data.thread_created_utc = 0

int r_rest_Crawl_N_Calculate_Data.thread_downs = 0

int r_rest_Crawl_N_Calculate_Data.thread_duration = 0

string r_rest_Crawl_N_Calculate_Data.thread_id = ""

int r_rest_Crawl_N_Calculate_Data.thread_new_question_every_x_sec = 0

int r_rest_Crawl_N_Calculate_Data.thread_question_top_score = 0

list r_rest_Crawl_N_Calculate_Data.thread_questions_n_answers = []

int r_rest_Crawl_N_Calculate_Data.thread_time_stamp_last_question = 0

string r_rest_Crawl_N_Calculate_Data.thread_title = ""

list r_rest_Crawl_N_Calculate_Data.thread_unanswered_questions = []

list r_rest_Crawl_N_Calculate_Data.thread_unanswered_questions_converted = []

int r_rest_Crawl_N_Calculate_Data.thread_ups = 0

r_rest_Login_Behaviour Namespace Reference

Classes

- class r_rest_Login_Behaviour

Variables

- r = praw.Reddit(user_agent="University_Regensburg_iAMA_Crawler_0.001")
- client_id
- client_secret
- redirect_uri
- url_auth = r.get_authorize_url('uniqueKey', ['identity', 'submit'], True)

Variable Documentation

r_rest_Login_Behaviour.client_id

r_rest_Login_Behaviour.client_secret

r_rest_Login_Behaviour.r = praw.Reddit(user_agent="University_Regensburg_iAMA_Crawler_0.001")

r_rest_Login_Behaviour.redirect_uri

r_rest_Login_Behaviour.url_auth = r.get_authorize_url('uniqueKey', ['identity', 'submit'], True)

r_rest_Meta_Logger Namespace Reference

Classes

- class r_rest_Meta_Logger

r_rest_No_Cache Namespace Reference

Functions

- def nocache (view)

Function Documentation

def r_rest_No_Cache.nocache (*view*)

r_rest_Post_Behaviour Namespace Reference

Classes

- class r_rest_Post_Behaviour

r_rest_Service Namespace Reference

Functions

- def use_signin_key ()
- def crawl_n_calculate_data ()
- def write_meta_data_file ()
- def post_comment_to_reddit ()
- def return_font_files (font_file)
- def return_js_files (js_file)
- def return_js_map_files (map_file)
- def return_cs_map_files (map_file)
- def return_css_files (css_file)
- def return_img_files (img_file)

Variables

- app = Flask(__name__, static_url_path='')
- cData = r_rest_Crawl_N_Calculate_Data()
- tOverview = r_rest_Thread_Overview()
- iLogin = r_rest_Login_Behaviour()
- pBehaviour = r_rest_Post_Behaviour()
- mWriter = r_rest_Meta_Logger()
- string username_actually_logged_in = ""
- string thread_actually_used = ""
- r_object = None
- methods
- host
- threaded
- True
- debug

Function Documentation

def r_rest_Service.crawl_n_calculate_data ()

```
Crawls author data, writes them into databases and prepares questions and answers depending on given
parameters

    This route is active, whenever the user
    - clicked the refresh button on the (un)answered panel
    - initially selected a thread on the left side panel

    This route processes (sorting / filtering) settings for (un)answered questions panel

Args:
    request.args.get('t_id')  : The id of the thread being processed
    request.args.get('u_f_t') : The selected tier - filter for unanswered questions (all / 1 / X)
    request.args.get('u_s_e') : The selected score comparison - filter for unanswered questions
(eql / grt / lrt)
    request.args.get('u_s_n') : The selected score value used for filter for unanswered
questions(any int)
    request.args.get('u_s_d') : The selected sorting direction for unanswered questions (asc / des)
    request.args.get('u_s_t') : The selected type to sort the data to (author / creation / score
/ random)

    request.args.get('a_f_t') : The selected tier - filter for answered questions (all / 1 / X)
    request.args.get('a_s_e') : The selected score comparison - filter for answered questions (eql
/ grt / lrt)
    request.args.get('a_s_n') : The selected score value used for filter for answered questions(any
int)
    request.args.get('a_s_d') : The selected sorting direction for answered questions (asc / des)
    request.args.get('a_s_t') : The selected type to sort the data to (author / creation / score
/ random)

Returns:
    1. thread_over_view_data (whenever if will be entered) (dict):

        'title' (str):                  [The written title of the thread]
        'amount_answered' (str):        [The amount of questions already answered]
        'amount_of_questions' (str):    [The overall amount of questions]
        'duration' (str):               [The duration of the thread (in hours / days) depending
on internal calc]
        'thread_id' (str):              [The id of the thread]

    2. (un)answered question information sorted / filtered (dict):

        'extracted_an_filter_score_equals' (str):       [Answered q: The score comparison (eql /
grt / lrt)]
        'extracted_an_filter_score_numeric' (str):      [Answered q: The score value (int)]
        'extracted_an_filter_tier' (str):               [Answered q: The tier - filter (all / 1
/ Xx]
        'extracted_an_sorting_direction' (str):         [Answered q: The sorting direction (asc
/ des)]
        'extracted_an_sorting_type' (str):              [Answered q: The sorting type
                                                        (author / creation / score / random)]

        'extracted_thread_id' (str):                    [The ID of the processed thread]

        'extracted_un_filter_score_equals' (str):       [Unanswered q: The score comparison (eql
/ grt / lrt)]
        'extracted_un_filter_score_numeric' (str):      [Unanswered q: The score value (int)]
        'extracted_un_filter_tier' (str):               [Unanswered q: The tier - filter (all /
1 / Xx]
        'extracted_un_sorting_direction' (str):         [Unanswered q: The sorting direction (asc
/ des)]
        'extracted_un_sorting_type' (str):              [Unanswered q: The sorting type
                                                        (author / creation / score / random)]
```

def r_rest_Service.post_comment_to_reddit ()

```
Whenever the user clicked 'send' on the iAMA Experience prototype this route will be accessed and
the comment
will be posted to reddit

    This route is active, whenever the user clicks the "send" button within the unanswered questions
panel
    It works the following way:

    1. A REST-POST message, with the text inside its body to be uploaded to reddit will retreived
      1.1. That post will be uploaded to reddit

    2. The new information will be crawled from reddit and written into the database
      Crawling it live from reddit instead of directly writing it into the database is more precise
      (i.E. in cases of utc epoch timestamp)
    ----
    This route processes (sorting / filtering) settings for (un)answered questions panel

Args:
    request.args.get('c_id') (str) : The ID of the comment the author replied to
    request.json['text'] (str) : The answer text of the author

Returns:
    "Processed your posting request" (str) : The string, which will be given in return does not
matter.
```

After successful return of that string a new ajax - REST - Call triggering information recrawl will be done.

def r_rest_Service.return_cs_map_files (***map_file*****)**

```
Whenever the webpage tries to access .css.map files they will be returned to it

Args:
   map_file (str): The path to the requested .css.map- file

Returns:
```

(File): The requested .css.map - file

def r_rest_Service.return_css_files (***css_file*****)**

```
Whenever the webpage tries to access .css files they will be returned to it

Args:
   css_file (str): The path to the requested .css - file

Returns:
```

(File): The requested .css - file

def r_rest_Service.return_font_files (***font_file*****)**

```
Whenever the webpage tries to access font files they will be returned to it

Args:
    font_file (str): The path to the requested font file

Returns:
```

(File): The requested font file

def r_rest_Service.return_img_files (***img_file*****)**

```
Whenever the webpage tries to access image files they will be returned to it

Args:
    img_file (str): The path to the requested image file

Returns:
```

(File): The requested image file

def r_rest_Service.return_js_files (***js_file*****)**

```
Whenever the webpage tries to access javascript files they will be returned to it

Args:
    js_file (str): The path to the requested .js file

Returns:
```

(File): The requested .js file

def r_rest_Service.return_js_map_files (***map_file*****)**

```
Whenever the webpage tries to access map files they will be returned to it
```

```
.map files are required by jquery.min
Args:
    map_file (str): The path to the requested js.map file

Returns:
```

    (File): The requested js.map file

def r_rest_Service.use_signin_key ()

```
Handles the call, whenever the user clicked "allow access" on Reddit-OAUTH2 - website

    Whenever the user successfully logged on to reddit he will be redirect to this route.

    After redirection, the given sign_key will be extracted and authentification within PRAW will
be done with that
    key.

Args:
    request.args.get('code') (str) : The sign key returned by reddit

Returns:
    app.send_static_file('index.html'): If the authetification was successful the iAMA experience
prototype will be
```

    displayed

def r_rest_Service.write_meta_data_file ()

```
Handles the call, whenever the user clicked something on the webpage

    Whenever the user clicked something on the webpage (i.e. buttons) it will be written down into
a text file.

    This will collect meta data and help us analyzing and improving our iAMA experience

Args:
    request.json['author'] : The name of the currently logged on user
    request.json['text'] : The description of what the user actually clicked

Returns:
```

    'done' : Just some text to fulfill the return principles

Variable Documentation

r_rest_Service.app = Flask(__name__, static_url_path='')

r_rest_Service.cData = r_rest_Crawl_N_Calculate_Data()

r_rest_Service.debug

r_rest_Service.host

r_rest_Service.iLogin = r_rest_Login_Behaviour()

r_rest_Service.methods

r_rest_Service.mWriter = r_rest_Meta_Logger()

r_rest_Service.pBehaviour = r_rest_Post_Behaviour()

r_rest_Service.r_object = None

string r_rest_Service.thread_actually_used = ""

r_rest_Service.threaded

r_rest_Service.tOverview = r_rest_Thread_Overview()

r_rest_Service.True

string r_rest_Service.username_actually_logged_in = ""

r_rest_Thread_Overview Namespace Reference

Classes

- class r_rest_Thread_Overview

Variables

- reddit_instance = praw.Reddit(user_agent="University_Regensburg_iAMA_Crawler_0.001")
- mongo_db_client_instance = MongoClient('localhost', 27017)
- mongo_db_author_fake_iama_instance = mongo_db_client_instance['fake_iAMA_Reddit_Authors']
- mongo_db_author_fake_iama_collection_names = mongo_db_author_fake_iama_instance.collection_names()

Variable Documentation

r_rest_Thread_Overview.mongo_db_author_fake_iama_collection_names = mongo_db_author_fake_iama_instance.collection_names()

r_rest_Thread_Overview.mongo_db_author_fake_iama_instance = mongo_db_client_instance['fake_iAMA_Reddit_Authors']

r_rest_Thread_Overview.mongo_db_client_instance = MongoClient('localhost', 27017)

r_rest_Thread_Overview.reddit_instance = praw.Reddit(user_agent="University_Regensburg_iAMA_Crawler_0.001")

Class Documentation

r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data Class Reference

Public Member Functions

- def main_method (self, author_name, id_thread, un_filter_tier, un_filter_score_equals, un_filter_score_numeric, un_sorting_direction, un_sorting_type, an_filter_tier, an_filter_score_equals, an_filter_score_numeric, an_sorting_direction, an_sorting_type)

Static Public Member Functions

- def get_n_write_author_information (name_of_author)
- def clear_variables ()
- def get_thread_submission (id_of_thread)
- def fill_misc_thread_data ()
- def fill_left_n_top_panel_data (self)
- def fill_right_panel_data (self, id_of_thread)
- def calculate_question_stats (self)
- def calculate_down_votes ()
- def calculate_time_difference (time_value_1, time_value_2)
- def checker_comment_is_question (string_to_check)
- def checker_comment_is_question_on_tier_1 (string_to_check)
- def checker_comment_is_not_from_thread_author (author_of_thread, comment_author)
- def check_if_comment_has_been_answered_by_thread_author (self, author_of_thread, comment_acutal_id, comment_timestamp, comments_cursor)
- def sort_n_filter_questions (questions_to_be_sorted, filter_tier, filter_score_equals, filter_score_numeric, sorting_direction, sorting_type)
- def convert_epoch_to_time (timeAsString)
- def build_list_containing_q_n_a (self)
- def prepare_unanswered_questions (self)
- def uprint (objects, sep=' ', end='\n', file=sys.stdout)
- def test_calculated_values ()
- def create_json_object ()

Member Function Documentation

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.build_list_containing_q_n_a (*self*) `[static]`

```
Prepares data for display in the "answered questions" panel

    This method iterates over all answered questions and all answers the host made.
    Furthermore it merges them together into pairs for a easy display of it on the website

Args:
    self : Self reference - necessary to use methods within this class

Returns:
    -
```

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.calculate_down_votes ()`[static]`

```
Calculates the amount of down votes of a thread

    This is actually not necessary anymore but will be left inside, whenever downvotes will be
reimplemented
    to the website.

Args:
    -

Returns:
```

object (int): The amount of time difference between two values in seconds

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.calculate_question_stats (*self*)`[static]`

```
Calculates remaining question statistics, like average question score, reaction time and question
creation
    interval in seconds

Args:
    self:   Self representation of the class [necessary to use methods within the class itself]
Returns:
    -
```

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.calculate_time_difference (*time_value_1*, *time_value_2*)`[static]`

```
Calculates the time difference between two floats in epoch style and returns seconds

Args:
    time_value_1 (float): The first time value to be used for calculation
    time_value_2 (float): The second time value to be used for calculation
Returns:
    time_diff_seconds (int): The amount of time difference in seconds
```

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.check_if_comment_has_been_answered_by_thread_author (*self*, *author_of_thread*, *comment_acutal_id*, *comment_timestamp*, *comments_cursor*)`[static]`

```
Checks whether both strings are equal or not

1. A dictionary containing flags whether that a question is answered by the host with the appropriate
timestamp
    will be created in the beginning.
2. Then the method iterates over every comment within that thread
    1.1. Whenever an answer is from the iAMA hosts and the processed comments 'parent_id' matches
the iAMA hosts
comments (answers) id, the returned dict will contain appropriate values and will be returned
    1.2. If this is not the case, it will be returned in its default condition

Args:
    self:   Self representation of the class [necessary to use methods within the class itself]
    author_of_thread (str) : The name of the thread author (iAMA-Host)
    comment_acutal_id (str) : The id of the actually processed comment
    comment_timestamp (float): The timestamp of the currently processed comment
    comments_cursor (Cursor) : The cursor which shows to the amount of comments which can be iterated

Returns:
    True (bool): Whenever the strings do not match
```

False (bool): Whenever the strings do match

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.checker_comment_is_not_from_thread_author (*author_of_thread*, *comment_author*) `[static]`

```
Checks whether both strings are equal or not

1. This method simply checks wether both strings match each other or not.
    I have built this extra method to have a better overview in the main code..

Args:
    author_of_thread (str) : The name of the thread author (iAMA-Host)
    comment_author (str) : The name of the comments author

Returns:
    True (bool): Whenever the strings do not match
```

False (bool): Whenever the strings do match that given question

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.checker_comment_is_question (*string_to_check*) `[static]`

```
Simply checks whether a given string is a question or not

1. This method simply checks wether a question mark exists within that string or not..
    This is just that simple because messing around with natural processing kits to determine the
semantic sense
    would blow up my bachelor work...

Args:
    string_to_check (str) : The string which will be checked for a question mark

Returns:
    True (bool): Whenever the given string is a question
```

False (bool): Whenever the given string is not a question

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.checker_comment_is_question_on_tier_1 (*string_to_check*) `[static]`

```
Simply checks whether a given string is a question posted on tier 1 or not

1. This method simply checks whether a question has been posted on tier 1 by looking whether the
given
    string contains the substring "t3_" or not

Args:
    string_to_check (str): The string which will be checked for "t3_" appearance in it

Returns:
```

-

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.clear_variables () `[static]`

```
Resets all variables, to not return duplicate objects.
```

```
    Because the REST-Service won't destruct the objects by it self we have to reset them manually
here

Args:
    -

Returns:
    -
```

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.convert_epoch_to_time (***timeAsString*****)** `[static]`

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.create_json_object () `[static]`

```
Builds a JSON object consisting of all values which have been previously calculated

  Args:
      -

  Returns:
```

-

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.fill_left_n_top_panel_data (***self*****)** `[static]`

```
Fills data to the left and the top panel

Args:
    self:   Self representation of the class [necessary to use methods within the class itself]

Returns:
```

-

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.fill_misc_thread_data () `[static]`

```
Retrieves the creation time stamp and the thread author from the submission

Args:
    -

Returns:
```

-

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.fill_right_panel_data (***self*****,** ***id_of_thread*****)** `[static]`

```
Calculates various statistics for the left panel of the page

Args:
    self:   Self representation of the class [necessary to use methods within the class itself]

    id_of_thread: The id of the thread which is to be processed

Returns:
    -
```

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.get_n_write_author_information (*name_of_author*)`[static]`

```
Crawls data from the author into the mongodb

    At first all previously stored data will be dropped and then the new one will be crawled.
    This may be slow at some times but it enables us to give the user a better iAMA experience,
because
    he will immediately receive new data upon posting / requesting.

Args:
    name_of_author (str): The name of the author whose data is to be crawled

Returns:
```

-

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.get_thread_submission (*id_of_thread*)`[static]`

```
Receives the thread information live from Reddit via the Reddit-API

Args:
    id_of_thread (str): The id of the thread whose data are to be retrieved and stored globally

Returns:
```

-

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.main_method (*self*, *author_name*, *id_thread*, *un_filter_tier*, *un_filter_score_equals*, *un_filter_score_numeric*, *un_sorting_direction*, *un_sorting_type*, *an_filter_tier*, *an_filter_score_equals*, *an_filter_score_numeric*, *an_sorting_direction*, *an_sorting_type*)

```
Defines the main method which will be called by listening on a certain REST-Interface

Args:
    self:   Self representation of the class [necessary to use methods within the class itself]

    author_name(str): The name of the author who currently processed threads

    id_thread(str): The ID of the thread which will be searched for within the database


    un_filter_tier(str) : The kind of tier for which the questions will be filtered accordingly
(all / 1 / x)
     for unanswered questions
    un_filter_score_equals(str) : The kind of comparison the questions will be filtered on (eql
/ grt / lrt)
     for unanswered questions
    un_filter_score_numeric(str): The "number" of score / upvote which will be used to filter the
questions
     (int)for unanswered questions
    un_sorting_direction(str): The direction the questions will be filtered after (asc / desc)
     for unanswered questions
    un_sorting_type(str): The type of information the questions will be filtered after
     (author, creation, score, random) for unanswered questions

    an_filter_tier(str) : The kind of tier for which the questions will be filtered accordingly
(all / 1 / x)
     for answered questions
    an_filter_score_equals(str) : The kind of comparison the questions will be filtered on (eql
/ grt / lrt)
     for answered questions
    an_filter_score_numeric(str): The "number" of score / upvote which will be used to filter the
questions
     (int) for answered questions
    an_sorting_direction(str): The direction the questions will be filtered after (asc / desc)
     for answered questions
    an_sorting_type(str): The type of information the questions will be filtered after
    (author, creation, score, random) for answered questions

Returns:
    create_json_object (json): A complex json object containing

1. Information about various, thread related statistics
```

2. All (un)answered questions (& answers) sorted and filtered according to the parameters given

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.prepare_unanswered_questions (*self*)`[static]`

```
Re-prepares the unanswered questions for correct display on the website

    It is necessary to re-prepare and strip down information from the questions.
    If we would not do this there would be huge overhead in JSON - rest-transfer..
    (i.E. the website does not flags like "answered_by_host" == true, etc..)

Args:
    self : Self reference - necessary to use methods within this class

Returns:
```

-

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.sort_n_filter_questions (*questions_to_be_sorted*, *filter_tier*, *filter_score_equals*, *filter_score_numeric*, *sorting_direction*, *sorting_type*)`[static]`

```
Sorts and filters given question lists depending on parameters received via REST call

Args:
    questions_to_be_sorted (list): Contains all questions which will be processed later on

    filter_tier (str): Contains the information, which questions, depending on the tier, will be
sorted out
(all / 1 / X)
    filter_score_equals(str): Contains the information to filter a tier depending on a special score
(eql [equal] / grt [greather than] / lrt [lesser than])
    filter_score_numeric(str): The upvote score which will be used for filtering

    sorting_direction(str): The direction which will be used for sorting the questions
(asc [ascending] / des [descending])
    sorting_type(str): The kind of type which will be used for sorting
(author / creation / score / random)

Returns:
```

-

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.test_calculated_values ()`[static]`

```
This method is for debugging purpose only. It shows if all values have been calculated the correct
way.

Args:
    -
```

```
Returns:
```

    -

def r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data.uprint (***objects*****,** ***sep*** **= `' '`,** ***end*** **= `'\n'`,** ***file*** **= `sys.stdout`) `[static]`**

```
This method is also for debugging purpose only. It helps printing out questions which can not be
printed out
    the normal way because of errors displaying unicode characters (Windows has some problems with
it...)

Args:
    *objects(object) : The kind of object, which will be used for printing
    sep(str) : The seperator to seperated the printed text
    end(str) : Defines whenever the printing should stop
    file(object) : Defines where to print that object to

Returns:
```

    -

The documentation for this class was generated from the following file:

- r_rest_Crawl_N_Calculate_Data.py

r_rest_Login_Behaviour.r_rest_Login_Behaviour Class Reference

Static Public Member Functions

- def go_to_login_page ()
- def sign_in_with_returned_key (sign_key)

Member Function Documentation

def r_rest_Login_Behaviour.r_rest_Login_Behaviour.go_to_login_page () [static]

```
Whenever the REST - service gets initially started this method will be executed

    This method opens an authentification webpage, which will redirect to the route
    '/authorize_callback/' where the sign in key will be getting extracted and logon / posting
behaviour
    will be received

Args:
    -

Returns:
```

-

def r_rest_Login_Behaviour.r_rest_Login_Behaviour.sign_in_with_returned_key (*sign_key*) [static]

```
Logs on the user to reddit via using the transmitted sign_key.

    Additionally some user information gets extracted and the ability to post comments on reddit
will be
    achieved in here

Args:
    sign_key (str): The key which will be extracted from the authentification url callback

Returns:
    dict_to_return (dict) : Contains the extracted username and the PRAW (r) object, which is going
to be used
     within 'r_rest_Post_Behaviour' - class

    dict({
'username': authenticated_user.name,
'r_object': r
```

})

The documentation for this class was generated from the following file:

- r_rest_Login_Behaviour.py

r_rest_Meta_Logger.r_rest_Meta_Logger Class Reference

Static Public Member Functions

- def write_data_into_file (user, usage_text)

Member Function Documentation

def r_rest_Meta_Logger.r_rest_Meta_Logger.write_data_into_file (***user*****,** ***usage_text*****)** **`[static]`**

```
The mechanism to create text files containing usage data is defined here

    Whenever the user clicks something on the webpage it will be written down into a text file.
    That text file will be analyzed by a seperate method, which is not yet defined here

    This class works as described below:

    1. It receives the submission object for the given thread_id at first.
    2. Now it crawls all comments from reddit, by breaking up the hierarchy
    3. It iterates over all comments. Whenever the iterated comments id matches the one the author
replied to:
Post the answer of the author to reddit.

Args:
    user (str): The name of the author who has clicked something
    usage_text (str): The name of the behaviour he clicked / did

Returns:
    -
```

The documentation for this class was generated from the following file:

- r_rest_Meta_Logger.py

r_rest_Post_Behaviour.r_rest_Post_Behaviour Class Reference

Static Public Member Functions

- def post_comment_on_reddit (r_object, iama_thread_id, id_to_reply_to, comment_text)

Member Function Documentation

def r_rest_Post_Behaviour.r_rest_Post_Behaviour.post_comment_on_reddit (*r_object*, *iama_thread_id*, *id_to_reply_to*, *comment_text*) `[static]`

```
The mechanism to reply to questions on reddit is defined here

    This class works as described below:

    1. It receives the submission object for the given thread_id at first.
    2. Now it crawls all comments from reddit, by breaking up the hierarchy
    3. It iterates over all comments. Whenever the iterated comments id matches the one the author
replied to:
Post the answer of the author to reddit.

Args:
    r_object (PRAW.object): The prepared r-object, which is necessary to be able to post
    iama_thread_id (str): The thread the iAMA author is currently working on
    id_to_reply_to (str): The question id the author is replying to
    comment_text (str): The text the author has been posted

Returns:
```

-

The documentation for this class was generated from the following file:

- r_rest_Post_Behaviour.py

r_rest_Thread_Overview.r_rest_Thread_Overview Class Reference

Public Member Functions

- def get_n_return_thread_data (self, author_name)

Static Public Member Functions

- def get_live_thread_data (thread_id, thread_author_name)

Member Function Documentation

def r_rest_Thread_Overview.r_rest_Thread_Overview.get_live_thread_data (*thread_id*, *thread_author_name*)`[static]`

```
Retrieves fresh and live data for the given thread_id and given thread_author_name

    This method crawls thread data live from treddit, and does some minor calculation to fit the
requirements
    of the iAMA Experience prototype website on its left panel

Args:
    thread_id (str): The id of the thread beeing processed
    thread_author_name (str): The author name of the processed thread

Returns:
```

(File): The requested font file

def r_rest_Thread_Overview.r_rest_Thread_Overview.get_n_return_thread_data (*self*, *author_name*)

```
Retrieves live data for all threads the author has ever created

Args:
    author_name (str): The name of the author which threads are to be processed

Returns:
    json_data (json): Contains various little information for every thread the author has created
            The structure can be seen down below...

    {
"threads_information": [
    {"title":
    "amount_of_questions":
    "amount_answered":
    "duration":
    "thread_id":}
]
```

}

The documentation for this class was generated from the following file:

- r_rest_Thread_Overview.py

File Documentation

r_rest_Crawl_N_Calculate_Data.py File Reference

Classes

- class r_rest_Crawl_N_Calculate_Data.r_rest_Crawl_N_Calculate_Data

Namespaces

- r_rest_Crawl_N_Calculate_Data

Variables

- r_rest_Crawl_N_Calculate_Data.mongo_db_client_instance = MongoClient('localhost', 27017)
- r_rest_Crawl_N_Calculate_Data.mongo_db_author_fake_iama_instance = mongo_db_client_instance['fake_iAMA_Reddit_Authors']
- r_rest_Crawl_N_Calculate_Data.mongo_db_author_fake_iama_collection_names = mongo_db_author_fake_iama_instance.collection_names()
- r_rest_Crawl_N_Calculate_Data.mongo_db_author_comments_instance = mongo_db_client_instance['fake_iAMA_Reddit_Comments']
- r_rest_Crawl_N_Calculate_Data.mongo_db_author_comments_collection = mongo_db_author_comments_instance.collection_names()
- r_rest_Crawl_N_Calculate_Data.reddit_instance = praw.Reddit(user_agent="University_Regensburg_iAMA_Crawler_0.001")
- r_rest_Crawl_N_Calculate_Data.reddit_submission = None
- int r_rest_Crawl_N_Calculate_Data.thread_created_utc = 0
- string r_rest_Crawl_N_Calculate_Data.thread_author = ""
- string r_rest_Crawl_N_Calculate_Data.thread_title = ""
- int r_rest_Crawl_N_Calculate_Data.thread_amount_questions = 0
- int r_rest_Crawl_N_Calculate_Data.thread_amount_unanswered_questions = 0
- int r_rest_Crawl_N_Calculate_Data.thread_duration = 0
- string r_rest_Crawl_N_Calculate_Data.thread_id = ""
- int r_rest_Crawl_N_Calculate_Data.thread_ups = 0
- int r_rest_Crawl_N_Calculate_Data.thread_downs = 0
- int r_rest_Crawl_N_Calculate_Data.thread_time_stamp_last_question = 0
- int r_rest_Crawl_N_Calculate_Data.thread_average_question_score = 0
- int r_rest_Crawl_N_Calculate_Data.thread_average_reaction_time_host = 0
- int r_rest_Crawl_N_Calculate_Data.thread_new_question_every_x_sec = 0
- int r_rest_Crawl_N_Calculate_Data.thread_amount_questions_tier_1 = 0
- int r_rest_Crawl_N_Calculate_Data.thread_amount_questions_tier_x = 0
- int r_rest_Crawl_N_Calculate_Data.thread_question_top_score = 0
- int r_rest_Crawl_N_Calculate_Data.thread_amount_questioners = 0
- list r_rest_Crawl_N_Calculate_Data.thread_unanswered_questions = []
- list r_rest_Crawl_N_Calculate_Data.thread_answered_questions = []
- list r_rest_Crawl_N_Calculate_Data.thread_answers_of_host = []
- list r_rest_Crawl_N_Calculate_Data.thread_questions_n_answers = []
- list r_rest_Crawl_N_Calculate_Data.thread_unanswered_questions_converted = []
- list r_rest_Crawl_N_Calculate_Data.json_object_to_return = []

r_rest_Login_Behaviour.py File Reference

Classes

- class r_rest_Login_Behaviour.r_rest_Login_Behaviour

Namespaces

- r_rest_Login_Behaviour

Variables

- r_rest_Login_Behaviour.r = praw.Reddit(user_agent="University_Regensburg_iAMA_Crawler_0.001")
- r_rest_Login_Behaviour.client_id
- r_rest_Login_Behaviour.client_secret
- r_rest_Login_Behaviour.redirect_uri
- r_rest_Login_Behaviour.url_auth = r.get_authorize_url('uniqueKey', ['identity', 'submit'], True)

r_rest_Meta_Logger.py File Reference

Classes

- class r_rest_Meta_Logger.r_rest_Meta_Logger

Namespaces

- r_rest_Meta_Logger

r_rest_No_Cache.py File Reference

Namespaces

- r_rest_No_Cache

Functions

- def r_rest_No_Cache.nocache (view)

r_rest_Post_Behaviour.py File Reference

Classes

- class r_rest_Post_Behaviour.r_rest_Post_Behaviour

Namespaces

- r_rest_Post_Behaviour

r_rest_Service.py File Reference

Namespaces

- r_rest_Service

Functions

- def r_rest_Service.use_signin_key ()
- def r_rest_Service.crawl_n_calculate_data ()
- def r_rest_Service.write_meta_data_file ()
- def r_rest_Service.post_comment_to_reddit ()
- def r_rest_Service.return_font_files (font_file)
- def r_rest_Service.return_js_files (js_file)
- def r_rest_Service.return_js_map_files (map_file)
- def r_rest_Service.return_cs_map_files (map_file)
- def r_rest_Service.return_css_files (css_file)
- def r_rest_Service.return_img_files (img_file)

Variables

- r_rest_Service.app = Flask(__name__, static_url_path='')
- r_rest_Service.cData = r_rest_Crawl_N_Calculate_Data()
- r_rest_Service.tOverview = r_rest_Thread_Overview()
- r_rest_Service.iLogin = r_rest_Login_Behaviour()
- r_rest_Service.pBehaviour = r_rest_Post_Behaviour()
- r_rest_Service.mWriter = r_rest_Meta_Logger()
- string r_rest_Service.username_actually_logged_in = ""
- string r_rest_Service.thread_actually_used = ""
- r_rest_Service.r_object = None
- r_rest_Service.methods
- r_rest_Service.host
- r_rest_Service.threaded
- r_rest_Service.True
- r_rest_Service.debug

r_rest_Thread_Overview.py File Reference

Classes

- class r_rest_Thread_Overview.r_rest_Thread_Overview

Namespaces

- r_rest_Thread_Overview

Variables

- r_rest_Thread_Overview.reddit_instance = praw.Reddit(user_agent="University_Regensburg_iAMA_Crawler_0.001")
- r_rest_Thread_Overview.mongo_db_client_instance = MongoClient('localhost', 27017)
- r_rest_Thread_Overview.mongo_db_author_fake_iama_instance = mongo_db_client_instance['fake_iAMA_Reddit_Authors']
- r_rest_Thread_Overview.mongo_db_author_fake_iama_collection_names = mongo_db_author_fake_iama_instance.collection_names()

Index