rssFeed

Jorge Crespo Sueiro

August 30, 2019

Contents

1 rssFeed

Briefing the rssFeed provider does the job of retrieving all the avaliable data as rssFeed from the URL passed, to do this, Python is used to resolve such task.

1.1 Code at once

```
import feedparser
import json
import logging
import traceback
from datetime import datetime
from flask import Flask, request, jsonify

history = []
responses = []

HEADERS = {'Content-Type': 'application/json'}


app = Flask(__name__) # Declaring aplication

@app.route('/parse/history', methods=['GET'])
def rssParserHistory():
    return jsonify(history)

@app.route('/parse/responses', methods=['GET'])
def rssParserResponses():
    return jsonify(responses)


# port is 8050
@app.route('/parse', methods=['POST'])
def rssParser():

    #Extract URL, DATE, HEADING, FEEDCHANNEL"title"
    # Here will go the url coming from the JSON request


    url = request.json.get('url', 0)
    # HERE DATE
    storedTimestamp = request.json.get('from_time', 0)


    print("timestamp in request: {}".format(storedTimestamp))
    history.append(url)
    print("url " + url)
    if (url != 0 and storedTimestamp != 0):
        pfeed = feedparser.parse(url)
```

```
    print(pfeed)


#DATA IN CHANNEL
    if 'title' in pfeed.feed:
        channelTitle = pfeed.feed.get('title', 'no title')
        print("title " + channelTitle)
    if 'link' in pfeed.feed:
        channelLink = pfeed.feed.get('link', 'no link')
        print("link " + channelLink)
    if 'description' in pfeed.feed:
        channelDescription = pfeed.feed.get('description', 'no
            description')
        print("Description " + channelDescription)
    if 'updated' in pfeed.feed:
        channelPublished = pfeed.feed.get('updated', 'no date')
        print("Published " + channelPublished)

    entries = pfeed.entries



#HERE HEADING
    #HAHA JOKES, we don't do that over here

#HERE FEEDCHANNEL"title"


    data = None
    print("DEBUG: start to catch info")
    try:
        payload = {}
        try:
            payload['title'] = channelTitle
        except Exception:
            payload['title'] = 'none'
        try:
            payload['description'] = channelDescription
        except Exception:
            payload['description'] = 'none'
        try:
            payload['url'] = channelLink
        except Exception:
            payload['url'] = 'none'
        try:
            payload['published'] = channelPublished
        except Exception:
            payload['published'] = 'none'
        n = 0
        print ("DEBUG: Start to catch subEntries")
```

```
            for entry in entries:


                formattedTime = datetime.fromisoformat(entry.updated)
                updatedStamp = datetime.timestamp(formattedTime)

                if (updatedStamp > storedTimestamp):
                    oneEntry = {}
                    try:
                        oneEntry['entryId'] = entry.id
                    except Exception:
                        oneEntry['entryId'] = 'no id'
                    try:
                        oneEntry['entryTitle'] = entry.title
                    except Exception:
                        oneEntry['entryTitle'] = 'no entryTitle'
                    try:
                        oneEntry['entryDescription'] = entry.description
                    except Exception:
                        oneEntry['entryDescription'] = 'no
                            entryDescription'
                    try:
                        oneEntry['url'] = entry.link
                    except Exception:
                        oneEntry['url'] = 'no entryLink'
                    try:
                        oneEntry['entryPublished'] = entry.updated
                    except Exception:
                        oneEntry['entryPublished'] = 'no entryDate'
                    payload['entry' + str(n)] = oneEntry

                    n = n + 1

                print("building response")
                responses.append(payload)
                #response = requests.request('POST', url,
                    data=json.dumps(payload), headers=HEADERS)
                print ("RESPONSE SENT")
            except Exception as e:
                logging.error(traceback.format_exc())
                logging.error(str(e))

            print(payload)
            print("DEBUG End of building")
            return json.dumps(payload)
        print("RETURNING 404")
        return 'not found'


if __name__=='__main__':
```

```
    app.run(host='0.0.0.0', debug=True, port=8050)
```

1.2 Code analyzed

The objective of this subsection is to make easier to understand the code above.

Imports and aux. methods FeedParser is a library made for the job at hand, it allows parsing directly from rss/atom to JSON, all the others are related either to the REST service or to the HTTP requests.

rssParserHistory and rssParseResponses are methods to check history and responses arrays, those are filled with each new request

```
import feedparser
import json
import logging
import traceback
from datetime import datetime
from flask import Flask, request, jsonify

history = []
responses = []

HEADERS = {'Content-Type': 'application/json'}


app = Flask(__name__) # Declaring aplication

@app.route('/parse/history', methods=['GET'])
def rssParserHistory():
    return jsonify(history)

@app.route('/parse/responses', methods=['GET'])
def rssParserResponses():
    return jsonify(responses)
```

main parse method Once defined the url and the kind of request, we retrieve from the body of the request both timestamp and the url to check, the objective of this is fetching new data only.

The url is added to the history, both fields are validated and every possible field from the rssFeed is tried, to cover the possibility of one field not existing, some default values are defined (those are the second argument in each feed attempt). This piece of code contents the main .rss data, then the entries.

```
# port is 8050
@app.route('/parse', methods=['POST'])
def rssParser():

    #Extract URL, DATE, HEADING, FEEDCHANNEL"title"
    # Here will go the url coming from the JSON request


    url = request.json.get('url', 0)
    # HERE DATE
    storedTimestamp = request.json.get('from_time', 0)


    print("timestamp in request: {}".format(storedTimestamp))
    history.append(url)
    print("url " + url)
    if (url != 0 and storedTimestamp != 0):
        pfeed = feedparser.parse(url)
        print(pfeed)


    #DATA IN CHANNEL
        if 'title' in pfeed.feed:
            channelTitle = pfeed.feed.get('title', 'no title')
            print("title " + channelTitle)
        if 'link' in pfeed.feed:
            channelLink = pfeed.feed.get('link', 'no link')
            print("link " + channelLink)
        if 'description' in pfeed.feed:
            channelDescription = pfeed.feed.get('description', 'no
                description')
            print("Description " + channelDescription)
        if 'updated' in pfeed.feed:
            channelPublished = pfeed.feed.get('updated', 'no date')
            print("Published " + channelPublished)

        entries = pfeed.entries
```

response building Before catching all the data, we need the container for it, this JSON object is the one that will hold both main rssFeed data and each entry.

```
data = None
       print("DEBUG: start to catch info")
       try:
           payload = {}
           try:
```

```
    payload['title'] = channelTitle
except Exception:
    payload['title'] = 'none'
try:
    payload['description'] = channelDescription
except Exception:
    payload['description'] = 'none'
try:
    payload['url'] = channelLink
except Exception:
    payload['url'] = 'none'
try:
    payload['published'] = channelPublished
except Exception:
    payload['published'] = 'none'
n = 0
print ("DEBUG: Start to catch subEntries")
```

fetch and send for each existing entry, an empty JSON object is created and populated with fetched DATA, once done, it's appended to the final response and sent as a response. For debugging purposes the same data is shown in the console of the machine running this service.

At the end, 0.0.0.0 is set to adopt the same IP as the hosting machine and the port is set.

```
for entry in entries:


    formattedTime = datetime.fromisoformat(entry.updated)
    updatedStamp = datetime.timestamp(formattedTime)

    if (updatedStamp > storedTimestamp):
        oneEntry = {}
        try:
            oneEntry['entryId'] = entry.id
        except Exception:
            oneEntry['entryId'] = 'no id'
        try:
            oneEntry['entryTitle'] = entry.title
        except Exception:
            oneEntry['entryTitle'] = 'no entryTitle'
        try:
            oneEntry['entryDescription'] = entry.description
        except Exception:
            oneEntry['entryDescription'] = 'no
                entryDescription'
        try:
```

```
                    oneEntry['url'] = entry.link
                except Exception:
                    oneEntry['url'] = 'no entryLink'
                try:
                    oneEntry['entryPublished'] = entry.updated
                except Exception:
                    oneEntry['entryPublished'] = 'no entryDate'
                payload['entry' + str(n)] = oneEntry

                n = n + 1

            print("building response")
            responses.append(payload)
            #response = requests.request('POST', url,
                data=json.dumps(payload), headers=HEADERS)
            print ("RESPONSE SENT")
        except Exception as e:
            logging.error(traceback.format_exc())
            logging.error(str(e))

        print(payload)
        print("DEBUG End of building")
        return json.dumps(payload)
    print("RETURNING 404")
    return 'not found'


if __name__=='__main__':
    app.run(host='0.0.0.0', debug=True, port=8050)
```