ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Projektbericht

an der Hochschule für Technik und Wirtschaft des Saarlandes
im Studiengang Praktische Informatik
der Fakultät für Ingenieurwissenschaften

Persönliches Informationsportal „Elastifeed“ Backend

vorgelegt von
Jannik Schäfer
Matthias Riegler
Jorge Crespo Sueiro
Mark Martinussen

betreut und begutachtet von
Prof. Dr. Klaus Berberich

Saarbrücken, 28. September 2019

Inhaltsverzeichnis

1 Content retrieval

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. [**bentley:1999**]

1.1 test blab

Hier ein Test der Abkürzungen: Kubernetes (K8S). Beim ersten Mal wird noch der volle Name angegeben, danach nur die Abkürzung. K8S oder auch Kubernetes (K3S)

2 RssFeed

Jorge Crespo Sueiro

2.1 Introduction

In order to have a simpler representation of the info that a webpage has, there is a resource available called RSS or Really Simple Syndication, it's main use is to distribute updated info to subscribed users (of that page).
In our case, the idea is to have a simpler version in our system and keep it updated with news, this module has the responsability of retreiving those feeds by petition and checking if the info is new or not.

2.2 Solution

To do this job, I chosed Python over other programming languages mainly because of past experience working with this language, plenty of libraries to choose from and the readibility of the final code.

Python counts with a library that allows the deployment of a service very fast, called Flask. Even if it's not recomended to be used in production, it deals very well with reasonable amounts of traffic, wich is why in this case it can stay. Along with it, the "feedParser"library was made with the unique objective of retreiving any kind of data done with this paradigm in mind, RSS or Atom.
Having this 2 tools and the concept clear, the first prototype was done, with it and some additional research 2 more versions were made:

2.2.1 Version 2

This is the currently implemented version, is the first prototype but upgraded to handle better exceptions and to log every request that was made and every answer retreived, it stills runs in Flask but has an improved error handling so it's ready to work. The code below is this version.

2.2.2 Version 3

This is the alternative code containing the additional software, the reasons behind the making of this version are several, all coming from the same initial research.

As the first version had a lack of proper error handling, the unique sign of failure was that several HTTP request were being missed. I didn't find any logical reason for that so I thought it was low performance. Trying to find a way to improve the speed of this module, I found a paper blaming Python GIL (Global Interpreter Lock) that basically mess the way Python treats resources used by different threads, it lacks proper resource management and creating new threads it's a very demanding task.

This reading didn't said much to me related to my program as the GIL lack of performance affects mainly programs using diffferent cores at same time but made me wonder how to use several threads in an optimal way.
The solution I found had exactly the focus of managing several HTTP requests at same time, it consist of:

gunicorn A HTTP server implementing the pre-fork worker model, premade threads waiting and a controller that assign task to them, without killing each thread at the end of each work and reutilizing the pre-existing threads, the performance increases quite a bunch.

nginx Another server that can be used as a reverse Proxy, using it to buffer both request sending and receiving, allows gunicorn to work asyncronously from the interaction with the network. This is like this because the HTTP enviroment is lock from the moment the request is received to the moment it has being answered, if the network is slow, this time increases and the thread can't work in anything else. Having a reverse proxy buffering means that the worker threads will always be occupied with the business logic and leaving the work of receiving/sending to nginx.

Having this 2 tools, the efficiency of any HTTP server increases a lot compared with only using one thread running in flask. This version has not been implemented because to this project, version 2 works fine enough and because the lack of knowledge to make everything runnable as a whole(without having to first install gunicorn, then nginx, configure both etc...).

2.3 Code

```
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)
```

Referenzen

[1] Google Chrome. *Chrome Dev Tools Protocol Viewer*. 2019. URL: https://chromedevtools.github.io/devtools-protocol/ (besucht am 25. 09. 2019).

[2] Google Chrome. *Puppeteer*. 2019. URL: https://pptr.dev/ (besucht am 27. 09. 2019).

[3] *GitHub cdproto-gen*. 2019. URL: https://github.com/chromedp/cdproto-gen (besucht am 27. 09. 2019).

[4] Sean Leonard. *The text/markdown Media Type*. RFC 7763. März 2016. DOI: 10.17487/RFC7763. URL: https://rfc-editor.org/rfc/rfc7763.txt.

[5] flaggerdoot LLC. *browserless*. 2013. URL: https://docs.browserless.io/docs/start.html (besucht am 27. 09. 2019).

[6] Postlight Labs LLC. *Custom Parsers README*. 2019. URL: https://github.com/postlight/mercury-parser/blob/master/src/extractors/custom/README.md (besucht am 19. 09. 2019).

[7] Postlight Labs LLC. *Mercury Web Parser*. 2019. URL: https://mercury.postlight.com/ (besucht am 19. 09. 2019).

[8] Andrey Lushnikov. *Using Chrome DevTools Protocol*. 2019. URL: https://github.com/aslushnikov/getting-started-with-cdp/blob/master/README.md (besucht am 24. 09. 2019).

[9] Charlie Robbins u. a. *Winston Transports*. 2019. URL: https://github.com/winstonjs/winston/blob/HEAD/docs/transports.md (besucht am 20. 09. 2019).

[10] Go Team. *Package context*. 2019. URL: https://golang.org/pkg/context/ (besucht am 20. 09. 2019).

[11] World Wide Web Consortium. *HTML Standard*. 2019. URL: https://html.spec.whatwg.org/ (besucht am 18. 09. 2019).

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

K8S Kubernetes

K3S Kubernetes

Anhang