PyDash.io

Requirements Document

PyDash 2018
J. G. S. Overschie
T. W. E. Apol
A. Encinas-Rey Requena
K. Bolhuis
W. M. Wijnja
L. J. Doorenbos
J. Langhorst
A. Tilstra

Customers
Patrick Vogel
Mircea Lungu

VERSION 2018.02.25

Table of contents

Introduction

For implementing web services using Python, one of the most used solutions is called Flask. This however does not include a way to monitor which parts of the web service are used at what rate. A dashboard able to collect this data was developed by a group of computing science students of the University of Groningen. There are still a few shortcomings in this dashboard, most notably in the case of monitoring a large number of Flask apps.

To solve this, PyDash.io can be used. PyDash.io is a dashboard containing the information of multiple deployed Flask monitoring dashboards. The user (with the right credentials) now has a better overview of his deployed dashboards, and does not have to maintain multiple credentials for each one.

Target Users

Developers & Administrators of web services/API's. This users are a potential target as it will be very interested to make possible that other companies, apart from RUG, make use of the system. It would be useful for them to be able to measure with the analytic system the number of hits in their website. The system can be used as a global measurement or more specific. This will also benefit companies as they will be able to measure a certain part of the website and view what customers are interested when entering in their pages. So, this will mean that they will be able to correct errors of their websites or even improve and make emphasis on what their customers like.

Computing Science master students. As contrary of Developers & Administrators of web services/API´s, this user target will not benefit for a professional environment but for a preparation to a professional future. As we are doing right now, this analytic system will allow master students to develop dashboards. They can be implemented for a bigger and heavier project. With this, they will be able to build a complete application for their assignments and final projects of the master. For them to be able to control this system of measurement of hits on websites will provide a more specific and precise study of their own websites, that will lead to a more professional project.

Requirements

High Priority

- Users need to be able to create an account.
- Users need to be able to log in to their account.
- Users need to be able to register flask-monitoring-dashboards.
- Users need to be able to see an overview of all their registered dashboards.
- The PyDash service needs to receive data from each registered dashboard in regular intervals, for example, on an hourly basis.
- For each registered dashboard: Functionality for users to change which endpoints are being monitored.
- For each registered dashboard: visualising (the performance of) said dashboard (similar to how a dashboard is currently visualised).
    - General / Over all endpoints:
        - Overview of endpoints:
            - Endpoint name
            - Flag telling whether or not the endpoint is being monitored
            - Number of executions
            - Last accessed (date & time)
            - Average execution time
        - Heatmap of endpoint executions (per hour)
        - A set of horizontal bar-plots of how many times Endpoints are executed per day
        - Execution time - all endpoints and versions:
            - Per version:
              Show a horizontal box-plot of the execution times (over all Endpoints) per version
            - Per endpoint:
              Show a horizontal box-plot of the execution times per Endpoint (over all versions)
    - Endpoint specific:
        - Endpoint summary:
            - Endpoint name
            - Added since app version (version nr.)
            - Date added to app (date & time)
            - Link to endpoint
            - Last accessed (date & time)
        - Execution time:
            - Per day:

Show a stacked-vertical-bar-plot with the average execution time per day.
- Per user, per version:
  Show a dot-plot for the average execution time per user per version.
- Per IP-address, per version:
  Show a dot-plot for the average execution time per IP-address per version.
- Per version:
  Show a box-plot for the execution time per version.
- Per user:
  Show a box-plot for the execution time per user.

- Hits per day:
  - Vertical-bar-plot:
    Show a vertical-bar-plot with the number of hits per day.
  - Heatmap:
    Show a heatmap (per hour) when the specific Endpoint was executed.

Low Priority

- For each registered dashboard visualising (the performance of) said dashboard:
  - Endpoint specific:
    - Debug info about potential outliers:
      - Date (date & time)
      - Execution time (in ms)
      - Request values
      - Request headers
      - Request environment
      - Request url
      - Cpu percent
      - Memory
      - Stacktrace
- A number of static tutorial pages, detailing how to use the flask-monitoring-dashboard.
- An administrator overview for PyDash.io, where an administrator can see visualizations on the usage of PyDash.io itself.
  - In order to do this, we deploy a flask-monitoring-dashboard on PyDash.io itself.
- Allow users to change scope of monitored endpoints in case there are many. e.g. hiding or showing endpoints or groups of endpoints.

Nice to have

- In the future, we might want to support other types of dashboards, such as, for example, a django-monitoring-dashboard.

Not going to build

- Add more visualizations. (We're not going to reinvent the wheel and it's not necessary)

Non-functional requirements

High priority

- Security
  - Use https
  - Hash passwords

Low priority

- Responsive web design

API Specification

We are set to use the API from the *flask-monitoring-dashboard*. We are going to use two endpoints, which are described as follows:

Endpoint 1

```
get_json_data/<security_token>/<time_from>
```

The endpoint needs a *security_token* passed along with any request. Additionally a *time_from* in unix timestamp can be provided to get endpoint data since the time given. A list of objects are returned in JSON:

- `Endpoint` `<string>`
  The name of the endpoint.

- `Execution_time` `<float>`
  The execution time of the endpoint in milliseconds.

- `Time` `<timestamp>`

When the endpoint has been triggered, returns a string containing the time (something like: '%Y-%m-%d %H:%M:%S.%f').

- `Version` `<float>`
  Version of the website.

- `Group_by` `<string>`
  Which user called the function

- `Ip` `<string>`
  IP-address of the client that makes the request.

Endpoint 2

`get_json_monitor_rules/<security_token>`

The endpoint needs a *security_token* passed along with any request. Furthermore, a list of objects is returned in JSON:

- `last_accessed` `<timestamp>`
  When endpoint was last hit.

- `monitor` `<boolean>`
  Describes whether the endpoint is monitored or not. If false, no data is collected in FunctionCall-table.

- `time_added` `<timestamp>`
  Time the endpoint was added to webservice.

- `version_added` `<string>`
  Describes the version of the webservice at the time this endpoint was added. This is a git commit hash.

Customer Contact

Patrick Vogel <p.p.vogel@student.rug.nl>
Mircea Lungu <m.f.lungu@rug.nl>

Changelog

Date	Iteration	Changes
27-02-2018	First delivery.	None. Initial document
