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

TA
Patrick Vogel

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

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.

Developers. As well as for the Administrators of web services/API's, the Developers will be able to take advantages of the system to develop a dashboard. The dashboard will offer the developers to measure the effectiveness of their websites. Developers will be able to create a tool that will help them to improve their websites. So, as well as for Administrators, the system will help Developers to correct errors but also, to develop a effective system for their dashboards.

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.
- Allow users to enable/disable the actual monitoring of specific endpoints. ***Since the flask-monitoring-dashboard API does not yet provide any way to achieve this, we will not yet be able to implement this feature.***

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 + Privacy
    - We want to use an Secure Socket Layer connection, both to connect the users to the application, as well as to connect to the different instances of the *flask-monitoring-dashboard*. The reasoning behind this, is that when we do not do this, sensitive information might be shared or stolen with third parties.
    - We want to hash the user-account passwords in the application, to ensure that in the unfortunate event that PyDash.IO's information might be compromised, that the user accounts are relatively safe.
    - We want to prevent the resources used in the application to be susceptible to an *enumeration* attack: That is, authorization to access a certain resource will be checked before returning it, and inside URLs, identifiers that are non-incrementing integers will be used. This will stop people from both getting insight into the data we have stored by looking at the identifiers (for example how many dashboards we have) and using the URL syntax to access new endpoints they would otherwise not know about.
- Tooling
    - Usage of the programming language Python and the micro web framework Flask is required to build the PyDash.IO application. The reason behind this is to be able to use the *flask-monitoring-dashboard* to monitor the PyDash.IO web application itself at some point.

Low priority

- Responsive Design: We want the application to function properly (do not have lacking functionality) and not contain obvious graphical bugs or oversights. We are going to test this by accessing our web tool on different systems. We want our tool to work on at least the following display sizes:
  - A Full-HD screen (1920 x 1080)px
  - A medium-sized computer or tablet screen (1280 x 768)px
  - A small portrait-mode smartphone screen (320 x 550)px
- Scalability
  - In the hopes that PyDash.IO will be used by a reasonable group of users at some point, we want to ensure that the application can handle at least fifty *flask-monitoring-dashboard* instances side-by-side without the application being hampered in its primary purpose, which is to have users browse the distilled information already extracted from those *flask-monitoring-dashboards.* An endpoint response is too slow for a user if it takes longer than 0.5 seconds to respond, measured in server execution time.
- Etiquette of performing external API calls
  - When calling into an external *flask-monitoring-dashboard*, it is mandatory to not stress (overload) that application too much. This means that we need to rate-limit the calls made to these external applications (Do not perform a request to the same application more often than once every three minutes), as well as ensure that we do not request too much data at one time from them. ***(Managing this second requirement is difficult unless the flask-monitoring-dashboard API specification is changed to reduce the amount of data being sent in one single request!)***

Flask-monitoring-dashboard API Specification

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

1. get_json_details

Returns, in JSON format, some information about the *flask-monitoring-dashboard* deployment. At the moment, the version is returned.

2. get_json_data/<time_from>

Returns, encoded using JSON Web Token with the dashboard's security token, a list of objects containing endpoint access data of the *flask-monitoring-dashboard*. Optionally, a *time_from* UNIX timestamp can be provided to get all data since this timestamp. The objects returned are structured as follows:

- 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 requested the endpoint.
- ip   \<string\>
  IP address of the client making the request.

3. get_json_monitor_rules

Returns, encoded using JSON Web Token with the dashboard's security token, a list of objects describing *monitoring rules* for all endpoints on the website. The objects are structured as follows:
- endpoint   \<string\>
  Name of the endpoint, corresponds to the endpoint above.
- last_accessed   \<timestamp\>
  When this endpoint was last accessed.
- monitor   \<boolean\>
  Describes whether the endpoint is monitored or not.
- time_added   \<timestamp\>
  Time the endpoint was added to the web service.
- 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>

Meeting Log

For communication with the customer it was decided that we solely rely on using Slack. We therefore did not meet in person regarding issues the customer should be notified of.

Changelog

Date	Iteration	Changes	Author
2018-02-27	First delivery. (Sprint #0)	None. Initial document	*Shared Effort*
2018-03-09	Sprint #1	Improve non-functional requirements	Wiebe-Marten Wijnja
2018-03-09	Sprint #1	Added enabling/disabling monitoring of specific endpoints to nice to haves.	Tom Apol
2018-03-12	Sprint #1	Updated the *flask-monitoring-dashboard* API specification to the latest version.	Koen Bolhuis
2018-03-23	Sprint #2	Made changes based on TA suggestions	Jeroen Langhorst