ForVis - Doc2 - Design of Visualization System

Klimek

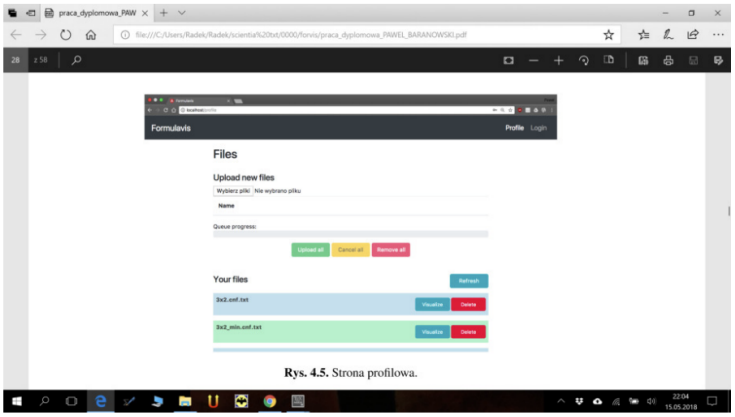

Figure 1: The screenshot of the system – formula management.

The implemented system, see Fig. 1, is briefly described. The server application was made using Python language together with the Django REST Framework application [1]. For operation of asynchronous tasks, Celery task queue [2], together with RabbitMQ [3] message broker, are responsible. Data is stored in a PostgreSQL database server. The client application, as well as the user interface, was created using the TypeScript language with the Angular 4 framework. An effective visualization library based on JavaScript, named vis.js [4], was used. The Nginx [5] server proxy is responsible for managing the communication between the user interface and the server application. The entire project is containerized using Docker technology [6].

Nginx plays the role of a Reverse-Proxy or Forward-Proxy server, see Fig. 2. A Reverse-Proxy server helps in:

- hiding the current system behind the proxy server,
- distributing traffic between application instances of the server,
- directing traffic towards appropriate applications,
- compressing data flow content,

- manipulating requests and responses.

Celery is an asynchronous task queue based on task distribution using messages. It focuses on operations performed in the shortest possible time while also supporting scheduled tasks. Tasks are executed concurrently using one or multiple worker instances with multiprocess transformation. These tasks can be performed asynchronously (allowing the application to continue working) or synchronously (requiring the application to wait for the result). In order to use Celery, a message broker is required to provide data to workers. The message broker recommended by Celery is RabbitMQ. It is an open-source and easy-to-implement system written in Erlang. It supports and monitors asynchronous message transmission and deployment using clusters, enabling system scalability.

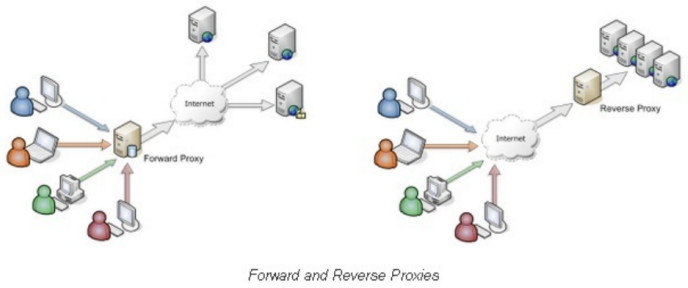

Figure 2: Difference between Forward-Proxy and Reverse-Proxy server. Source: Vivek Srivastav: Proxy Pass [7].

The created client application serves as the system's interface, providing access to the end user. This allows the user to log in and out, upload and delete data from the server, and most importantly, display visualizations and save them as files. The implemented application operates as a browser-based application. The separately implemented server application organizes data processing according to user expectations. A detailed analysis of this issue exceeds the scope of this work; see also Fig. 3.

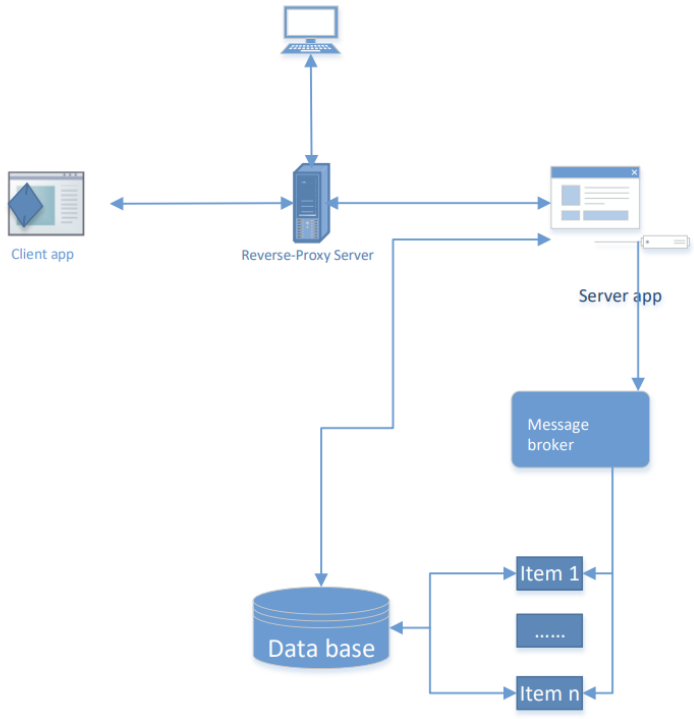

Figure 3: Logical structure of the visualization system.

References

[1] Django REST Framework. Available: `https://www.django-rest-framework.org/`

[2] Celery Task Queue. Available: `http://www.celeryproject.org/`

[3] RabbitMQ Message Broker. Available: `https://www.rabbitmq.com/`

[4] Vis.js Visualization Library. Available: `https://visjs.org/`

[5] Nginx Web Server. Available: `https://www.nginx.com/`

[6] Docker Containerization. Available: `https://www.docker.com/`

[7] Vivek Srivastav: Proxy Pass. Available: `http://viveksrivastv.blogspot.com/p/apacheadministration.html`. Accessed on 1 May 2018.