SWitCH - Computing systems and networks (SCOMRED) 2019/2020 Laboratory Script - Week 07

BASH based CGI applications.

Backend and web services testing - postman.

Team project development.

1. The Hash Calculator – deployment, testing and frontend

Under lecture six a hash calculator backend application has been introduced, it's a BASH based CGI application.

```
#!/bin/bash
M_CONTENT_FILE=/tmp/.scomred-hash-calculate.$$.tmp
### I'm file /var/www/cgi-bin/hashCalculate with execute permission
error_response() {
 echo "Status: 400 Bad Request"
 echo "Content-type: text/plain"
 echo ""
 echo "ERROR: ${1}"
 rm -f $M_CONTENT_FILE
 exit
}
###
if [ -n "$CONTENT_LENGTH" ]; then cat > $M_CONTENT_FILE
else error_response "No content found on the request"; fi
if [ "$CONTENT_LENGTH" == "0" ]; then error_response "No content found on the request"; fi
###
if [ "$REQUEST_METHOD" != "POST" ]; then
       error_response "Invalid method. The only supported method is POST."
       fi
if [ -z "$QUERY_STRING" ]; then error_response "No query-string"; fi
ALG=${QUERY_STRING#algorithm=}
if [ "$ALG" == "$QUERY_STRING" ]; then error_response "Bad query-string: $QUERY_STRING"; fi
case "$ALG" in
        MD4|MD5|RIPEMD160|SHA1|SHA224|SHA256|SHA384|SHA512);;
        *) error_response "Invalid hash algorithm: $ALG";;
esac
HASHCODE="$(openssl dgst -$ALG $M_CONTENT_FILE)"
rm -f $M_CONTENT_FILE
echo "Content-type: text/plain"
echo ""
echo "${HASHCODE#*= }"
###
```

If the request has a content (body), as it's supposed to, then the CONTENT_LENGTH variable is defined (not empty), the content is available through STDIN. The cat command is used to read all available data in STDIN and copy it to a temporary file. The $$ sequence in the temporary filename is expanded to the current process's PID, if two instances of this application are running at the same time in parallel each will have a unique PID, thus used filenames will be unique as well.

The only supported HTTP method is POST, if the received HTTP request is not a POST, then a compliant HTTP error message is sent by calling the error_response() function.

The HTTP request's URI is supposed to carry a query-string with the algorithm parameter, otherwise an error message is sent back.

If the hash algorithm is supported (actually supported by the openssl command), then the openssl's dgst command is used to calculate the hash code using data in the temporary file. The result is sent as response's content after removing a prefix included in the openssl's output.

1.1. Deployment

To be treated by the web server as a CGI application some settings and conditions are required on the server side, students are expected to use a VS created from the **Apache on Ubuntu 18.04 LTS - VNET1** template.

CGI support is already enabled on this template, executable files (with execute permission) that are stored in folder **/var/www/cgi-bin/** are handled by the Apache web server as being CGI applications and not documents (stored in folder /var/www/html).

The **/var/www/html/** folder is the **document root**, meaning under URI point of view it's the root folder (**/**). This means the URI **/index.html** is in fact file **/var/www/html/index.html**. The entire directories tree and documents starting at folder **/var/www/html/** is available as URI starting at **/**.

Beyond this basic tree of documents starting from the **document root**, the web server may be configure to provide other resources that will be mapped into the basic tree as **aliases**. In this configuration folder **/var/www/cgi-bin/** is mapped to URI **/cgi-bin/**, thus the directories tree starting from **/var/www/cgi-bin/** is available as URI starting from **/cgi-bin/**.

Activities:

a) Deploy the Hash Calculator CGI application into the **/var/www/cgi-bin/** folder of your VS, you may name the file as **hashCalculate** (**/var/www/cgi-bin/hashCalculate**).

Don't forget granting it the execute permission to the file.

b) It should be now available as **http://vsX.dei.isep.ipp.pt/cgi-bin/hashCalculate**, you may test it in your web browser.

If everything went as expected, you will get the following plain text:

ERROR: No content found on the request

This is because this URI is a web service and it's not intended for direct user interaction.

c) Look at the application code to understand why and where this response message was generated.

1.2. Testing with Postman

Clearly, testing web services with a web browser and nothing else is not a solution.

As reported in lecture six, one of the most popular tools to test web services is Postman (https://www.getpostman.com/).

Activities:

a) Install on your workstation the Postman application, it's not mandatory to install the full desktop version, a Postman plugin for your web browser will do perfectly.

b) Refer to lecture six and execute yourself the sequence of tests presented there as an example.

c) Try other different tests over your web service. See if you can find a bug, e.g. some missing input data validation.

1.3. Developing a frontend for the backend

A web service like for instance our **hashCalculate** is not specifically designed to be used by a frontend, a web service is designed to be used by applications named as consumers the same way applications call local functions available on the local operating system.

Such web services consumer applications may or not be a frontend, for instance a backend application may as well be a web services consumer, for instance if at some point a hash code calculation is required our **hashCalculate** could be called. The fact is, by using AJAX a browser based frontend application is able to become a web services consumer application.

Now the backend (the **hashCalculate** web service) is tested, consumers may be developed.

Activity/challenge:

Develop a web browser based frontend (HTML+JavaScript) to use the **hashCalculate** web service and make it directly available to users.

- The frontend UI can be as simple as one input text field to specify the hash algorithm name, a text area box to place data whose hash code will be calculated, and a calculate button for action.
- No significant input validations are required, those are already enforced on the backend, and under all points of view that's the right place to validate input data.

2. On line voting application – example from lecture seven (this week)

Activity

Deploy into your VS the example AJAX application available on lecture 7 – on line voting, it's made of three files.

Test the application.