Poznań University of Technology

Faculty of Control, Robotics and Electrical Engineering

Institute of Robotics and Machine Intelligence

Division of Control and Industrial Electronics

Server IoT
PHP / Python script and REST architecture

Mobile and embedded applications for Internet of Things

Teaching materials for laboratory

Dominik Łuczak, Ph.D.; Adrian Wójcik, M.Sc.

Dominik.Luczak@put.poznan.pl
Adrian.Wojcik@put.poznan.pl

I. Goal

Knowledge

The aim of the course is to familiarize yourself with:

- processing of arguments of PHP script, passed to with GET method,
- REST architecture style (Representational State Transfer),
- data formats used in IT systems

Skills

The aim of the course is to acquire skills in:

- creating simple and complex scripts written in PHP with processing arguments passed using the GET method,
- developing a simple data processing in IT systems.

Social competences

The aim of the course is to develop proper attitudes:

- strengthening the understanding and awareness of the importance of non-technical aspects and effects of the engineer's activities, and the related responsibility for the decisions taken,
- proper technical communication in context of software development.
- choosing of the right technology and programming tools for a given problem,

II. Laboratory report

Complete laboratory tasks as per the instructor's presentation. Work alone or in a team of two. **Keep safety rules while working!** Prepare laboratory report documenting and proving the proper execution of tasks. Editorial requirements and a report template are available on the *eKursy* platform. The report is graded in two categories: tasks execution and editorial requirements. Tasks are graded as completed (1 point) or uncompleted (0 points). Compliance with the editorial requirements is graded as a percentage. The report should be sent as a *homework* to the *eKursy* platform by Sunday, March 28, 2021 by 23:59.

III. Prepare to course

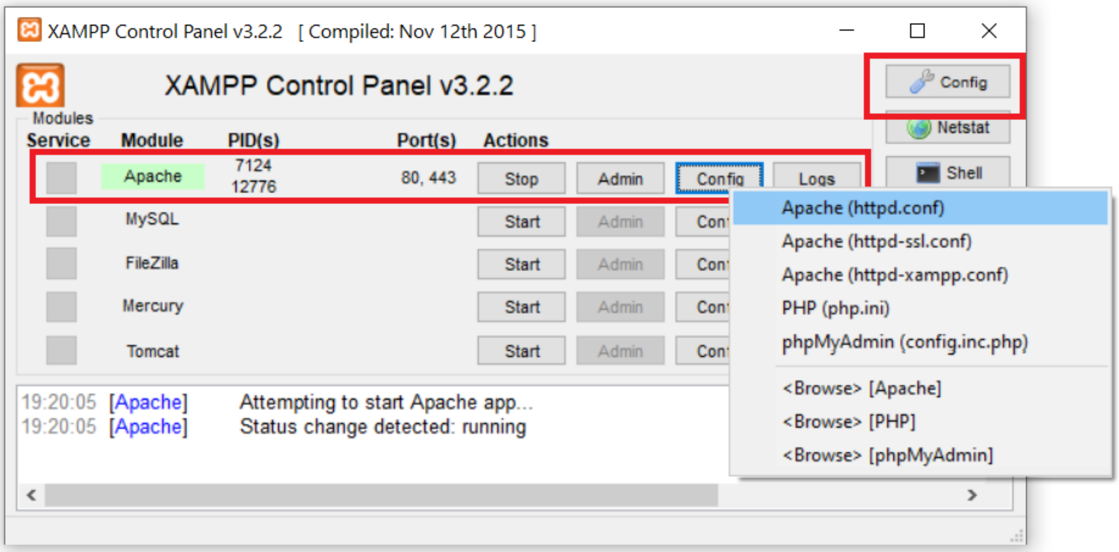
a) Know the safety rules

All information on the laboratory's safety instructions are provided in the laboratory and on Division website [1]. All inaccuracies and questions should be clarified with the instructor. It is required to be familiar with and apply to the regulations.

Attend the class prepared. Knowledge from all previous topics is mandatory.

b) Apache HTTP Server

XAMPP [2] is a free Apache HTTP Server distribution including PHP [3]. It's easy to install and configure web server. Apache is an open source HTTP server for modern operating systems including Linux and Windows [4]. The fig. 1 shows the main control panel of the XAMPP application with the Apache module running.



```
19:20:05  [Apache]   Attempting to start Apache app...
19:20:05  [Apache]   Status change detected: running
```

Fig. 1. XAMPP control panel.

Server can be turn on and off with the **Start/Stop** button. The list of server configuration files is available under the Apache module **Config** button. Main **Config** button (common menu on the right) opens a menu that allows, among others, choice of text editor for configuration files. Support for PHP scripts is enabled by default and requires no further configuration. Python scripting requires additional configuration by adding the content of Listing 1. to the *httpd.conf* file. This command allows you to handle *.py* files as CGI scripts [5] using the interpreter defined in first line of a script.

Listing 1. Apache configuration: .py files as CGI scripts.

```
# Enable Python CGI scripts
AddHandler cgi-script .py
ScriptInterpreterSource Registry-Strict
```

c) Introduction to the PHP language

The PHP language syntax (PHP Hypertext Preprocessor) [3] is similar to the C/C++ language with operations on variables specific to scripting languages (e.g. *bash* Unix shell). All instructions are normally placed between `<?php ?>` tags. Basic control instructions and loops write in the same way for PHP as C/C++. The most important differences are related to the use of variables, which are the basic (String, Integer, Float etc.), array and object variables. Each variable name begins with '$'. In addition, basic variables do not require the user to explicitly specify their type beforehand. Importantly, the value of a variable can be easily presented without first formatting it to text using the `echo` command. An examples of basic and array variables are presented in Listing 2. and Listing 3., respectively. Array variables, unlike C/C++, can have numeric and *text* indexes. The `print\_r` command allows you to display the contents of the selected table, which significantly facilitates the analysis of the code.

Listing 2. Basic variables operations.

```php
<?php
  /*
   * Multiline comment
   */
  $a=1;
  $b=2;
  $c=$a+$b;
  echo $a." + ".$b." = $c"; // Inline comment
?>
```

Listing 3. Array variables operations.

```php
<?php
  $d[0] = "value_1";
  $d['nazwa'] = "value_2";
  print_r($d);
?>
```

In PHP, global system variables (superglobal variables) are also available from anywhere in the script. The basic ones are presented in Listing 4. Parameters passed to the server using the **GET** method will be visible in the `$_GET` variable. Before using a given array variable, make sure using the `isset` command that its value from the given index is set.

Listing 4. PHP superglobal variables.

```php
<?php
  print_r($_SERVER);  // Server and execution environment info
  print_r($_GET);     // HTTP GET variables
  print_r($_POST);    // HTTP POST variables
  print_r($_SESSION); // Session variables
?>
```

Listing 5 shows a simple HTML form. The form contains basic information about the type of data entered, contained in the type field. It should be remembered that this is information for the web browser necessary for the correct formatting of data, however it is not important for the server side. Data from the form is sent to the server as plain text, regardless of its meaning. Therefore, data validation must be done on the server side. Each input element of the form must have a specific **name** field, which will be the index in the `$_GET` array. The value of the table for the specified index will depend on the data entered by the user in the form.

Listing 5. HTML form.

```html
<form action="<?php echo $_SERVER['PHP_SELF']?>" method="get">
  <input name="zmienna1" type="text" maxlength="10" value="Wpisz">
    Nazwa<br>
  <input name="zmienna2" type="password" maxlength="10" value="Wpisz">
    Haslo<br>
  <input name="zatwierdz" type="submit" value="Wprowadz">
</form>
```

Web interface without the use of additional mechanisms is stateless. This means that the server has no information about previous queries made by the user and related variables. The basic element that allows uto remember user and associated variables are the information saved in cookies. The cookie is created on the server side and is send to the client who must remember it and send it back to the server with each new query. A session is server-side information intended to exist only throughout the visitor's interaction with the website. Only a unique identifier is stored on the client side. This token is passed to the web server when the visitor's browser requests HTTP address. An example of using the session mechanism is Listing 6.

Listing 6. SESSION superglobal varialbe.

```php
<?php
  session_start();
  $_SESSION['zmienna']=$_SESSION['zmienna']+1;
  print_r($_SESSION);
?>
```

D) Introduction to CGI scripts in Python language

Python is a general-purpose programming language. One of the popular uses of Python is CGI scripting. Writing a CGI script in Python requires the `cgi` module [6]. Listing 7. presents a simple template of a CGI script in Python that returns a HTML message. Parameters transferred to the server using the HTTP GET or POST method are available in the script via `cgi.FieldStorage()` method (example on listing 8). JSON support requires the import of an additional `json` module [7].

Listing 7. A simple Python CGI script that responds with a HTML document.

```python
#!C:\Users\[USER]\AppData\Local\Programs\Python\Python37-32\python.exe
import cgi
import cgitb

cgitb.enable() # Enable error logging

print("Content-Type: text/html")    # HTML is following
print()                             # blank line, end of headers

print("Hello, world!<br>(but in Python!)")
```

Listing 8. Python CGI script that responds with a JSON object based on the input parameters.

```python
#!C:\Users\[USER]\AppData\Local\Programs\Python\Python37-32\python.exe
import cgi
import cgitb
import json

cgitb.enable() # Enable error logging

class SimpleColorDataType:
  R = 0
  G = 0
  B = 0

color = SimpleColorDataType()

color.R = 0
color.G = 0
color.B = 0

params = cgi.FieldStorage()

if "R" in params:
  color.R = int(params["R"].value)

if "G" in params:
  color.G = int(params["G"].value)

if "B" in params:
  color.B = int(params["B"].value)

colorStr = json.dumps(color.__dict__)

print("Content-Type:␣application/json")    # JSON is following
print()                                     # blank line, end of headers
print(colorStr)
```

E) REST architecture

REST (Representational State Transfer) is a style of network systems architecture. The concept of the RESTful interface (i.e. based on the REST architecture) breaks client-server communication to create a series of small modules. Each module applies to a specific part of the system. This modularity gives developers a lot of flexibility, but it can be a difficult task for designers from scratch. The RESTful interface uses HTTP methodology: uses **GET** to download the resource; **PUT** to change the state or update the resource, which may be an object, file or block; **POST** to create this resource; and **DELETE** to delete it. Thanks to REST, network components are a resource you need access to - a black box whose implementation details are unclear. All requests are assumed to be stateless.

A properly designed RESTful interface should have the following features [8]:

- **Use of a uniform interface (UI)**. Resources should be uniquely identifiable through a single URL, and only by using the underlying methods of the network protocol, such as DELETE, PUT and GET with HTTP, should it be possible to manipulate a resource.
- **Client-server based**. There should be a clear delineation between the client and server. UI and request-gathering concerns are the client's domain. Data access, workload management and security are the server's domain. This loose coupling of the client and server enables each to be developed and enhanced independent of the other.
- **Stateless operations**. All client-server operations should be stateless, and any state management that is required should take place on the client, not the server.
- **RESTful resource caching**. All resources should allow caching unless explicitly indicated that caching is not possible.
- **Layered system**. REST allows for an architecture composed of multiple layers of servers.
- **Code on demand**. Most of the time a server will send back static representations of resources in the form of XML or JSON. However, when necessary, servers can send executable code to the client.

IV. Scenario for the class

A) Teaching resources

Hardware	• computer,
Software	• web server with PHP / Python scripts support (e.g. XAMPP), • text editor (e.g. Notepad++, VS Code), • **(optional)** PHP IDE (e.g. Aptana Studio) or Python IDE (e.g. IDLE).

B) Tasks

Every task can be optionally implemented in Python using `cgi` module.

1. Run the XAMPP application control panel.
   (a) Start HTTP server - Apache.
   (b) Place in the folder indicated in **DocumentRoot** (file *httpd.conf*) your script *test.php* with content from the listing 9.

Listing 9. Test PHP script.

```
<?php
  echo "Hello, World!";
```

   (c) Launch your web browser and enter the URL: http://localhost/test.php.
   (d) If `Hello world` appeared in the browser, go to the next task.

2. Even number printing.
   (a) Add a new file *even_numbers.php*.
   (b) Display even numbers from the range given by the instructor (e.g. from 0 to 100).
   (c) Use the `for` or `while` loop.
   (d) Display consecutive numbers on the next line using `<br>`.

3. Printing even numbers from the given range.
   (a) Add new file *even_numbers_range.php*.
   (b) Send to the script by **GET** method the beginning and end of the interval from which the numbers will be printed.
   (c) Check if the array element `$_GET` has been set before using it.

4. Printing even numbers from the given range in JSON format.
   (a) Add a new file *even_numbers_range_json.php*.
   (b) Create a new empty array `$even_numbers` with the `array()` command. In the loop, set the values of the array.
   (c) Get the JSON format using `json_encode()`.
   (d) Print only the JSON array.
   (e) Test the script, if necessary - add an integer cast.

5. Using the JSON format parser for PHP `json_decode()`, create and print object variable from data presented in Listing 10.

Listing 10. JSON data.

```
{
  "device": 10,
  "analog_input": [ 10, 20 ],
  "other_info": "description"
}
```

6. Simple PHP calculator.
   (a) Create separate PHP scripts for factorial functions (e.g. *factorial.php*) and a simple calculator (e.g. *calculator.php*). Attach the factorial file to the calculator file with the command `include nazwa_pliku.php`.
   (b) Write the basic arithmetic operations: *addition*, *subtraction*, *multiplication* and *division* for 2 arguments. Add the factorial operation (1 argument) with the previously written script.
   (c) Before using the passed parameters, validate them. Print the result, arguments, and operation name in JSON format.
7. Read from file.
   (a) Add the files folder and create 3 non-empty text files with any content (.txt or .json) in it.
   (b) Using the `scandir()` function and the `foreach(array_expression as $ key => $ value)` loop, display all file names in the newly added folder.
   (c) Eliminate files'. 'and '...'.
   (d) Add another loop. Display the contents of the files using the `file_get_contents()` command.
8. Save to file.
   (a) Create the class Parameter (with *id*, *name*, *value*, *type* fields) and class Type (const type_foat = 0, const type_int = 1).
   (b) Create an object of the Parameter class and complete it with data (e.g. id = 0001, name = "test", value = 2.5, type = Type::type_foat).
   (c) Obtained in the JSON format from an object, save the file *id.txt* using the `file_put_contents()` function.
   (d) Read the data from the file, change the value field and then overwrite the file. Display data before and after modifications.
9. **(*)** Develop a class describing the input and output of the embedded device.
   (a) Create a class for representing the digital I/O state.
   (b) Create a class for representing the analog I/Ostate.
   (c) Extend the digital I/O class with additional information (e.g. device name, unique channel number, channel name, type of sensor/actuator, frequency measurement, date of last reading, activation level: low, high, slope).
   (d) Extend the analog I/O class with additional information (e.g. device name, unique channel number, channel name, type of sensor/actuator, resolution, accuracy, frequency of refreshing, date of last update of input, unit of measured value, range: minimum and maximum value, type of input signal filtering).
   (e) Read the sensor data from manually processed text files. Includes 4 digital and analog I/O.
   (f) Add read and write procedure for selected value using the GET method (e.g. `http://localhost/parameter.php?id=0001&operation=read` or `http://localhost/parameter.php?id=0001&operation=write&value=1`).

References

1. *Regulations, health and safety instructions* [online]. [N.d.] [visited on 2019-09-30]. Available from: http://zsep.cie.put.poznan.pl/materialy-dydaktyczne/MD/Regulations-health-and-safety-instructions/.
2. *XAMPP Installers and Downloads for Apache Friends* [online] [visited on 2021-03-12]. Available from: https://www.apachefriends.org/pl/index.html.
3. *PHP: Documentation* [online]. [N.d.] [visited on 2020-03-10]. Available from: https://www.php.net/docs.php.
4. *Welcome! - The Apache HTTP Server Project* [online] [visited on 2021-03-12]. Available from: https://httpd.apache.org/.
5. Common Gateway Interface. In: *Wikipedia, wolna encyklopedia* [online]. 2020 [visited on 2021-03-12]. Available from: https://pl.wikipedia.org/w/index.php?title=Common_Gateway_Interface&oldid=61707259. Page Version ID: 61707259.
6. *cgi — Common Gateway Interface support — Python 3.9.2 documentation* [online] [visited on 2021-03-12]. Available from: https://docs.python.org/3/library/cgi.html.
7. *json — JSON encoder and decoder — Python 3.9.2 documentation* [online] [visited on 2021-03-12]. Available from: https://docs.python.org/3/library/json.html.
8. ÖZLÜ, Ahmet. *Mastering REST Architecture — REST Architecture Details* [online]. 2019-01 [visited on 2020-03-10]. Available from: https://medium.com/@ahmetozlu93/mastering-rest-architecture-rest-architecture-details-e47ec659f6bc. Library Catalog: medium.com.