Individual Assignment I: (15% of marks)

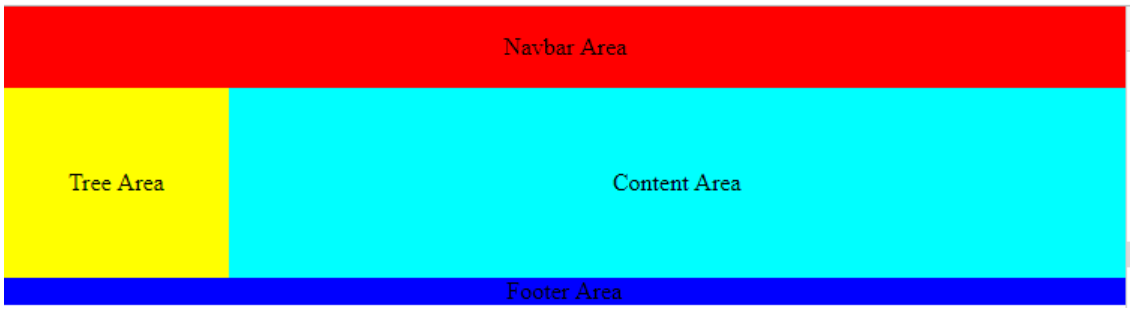
Complete WebAPP: Generating our individual portfolio

The objective of this work is to build a PHP-based website that shows the exercises from the individual portfolio produced so far (and the ones you will produce in next units) in a structured way. To do so you will need to complete a set of requirements described below:

1. **WebBuilder:** You have to create a PHP class (WebBuilder) that allows you to build web pages including different sections and to output the resulting HTML code.
   - 1.1. **NavigationBar:** You have to include a dropdown navigation bar that will be constructed to reflect the set of folders with the units and exercises. In the navigation bar the different units will appear. Then in the dropdown menu, the different exercises for that unit will appear.
   - 1.2. **Tree:** When one of the exercises is selected, a tree-like view will be used to show the different files included in that exercise.
   - 1.3. **Content area:** will show a set of buttons/links to perform different actions:
     - 1.3.1. **Show source:** When one of the files is selected, the content of the file will be shown in the content area.
     - 1.3.2. **Show output:** when one of the files is selected, the output produced will be shown in the content area.
     - 1.3.3. **Download:** to download the current file.
     - 1.3.4. **Download ALL:** to download the current exercise, all files of the exercise will be zipped before downloading.
2. **Extras:** Any extra feature that you may think of and has been "approved" by the teacher, such as:
   - 2.1. **Configure:** will allow you to change some parameters as the style used to show the code, the size of the left panel, or other you may think of.
   - 2.2. **Upload files:** will allow you to upload new exercises or files to the site.

1. WebBuilder

The WebBuilder class will be responsible of holding the whole webpage before being outputted. It should have methods to add HTML elements to the different areas of the webpage, **so it can be built in memory before outputting (echo) it.** The structure could be as follows (available on the PDU):



The class should have some arrays to hold the content of each Area while it is being created, it should have methods to allow the inclusion of new content to each of the areas and finally it should have a method that returns a string with the HTML code ready to be shown by the browser.

You need to consider that your class needs to be prepared to include styles and scripts, and that those can be included by linking the file or by adding the content into the same file, so you need to create methods to do so.

1.1. NavigationBar Area

Navigation bars are used as menus, in this practice we will create a two-level navigation bar that will follow the same structure as your units folders and exercises.

Navigation bars are built from a set of HTML unordered lists that contain links as elements, and then styled to look like a menu bar. You will need to adapt one of the existing examples on the internet[1] to work as you wish.

The class NavigationBar class should receive the path to the root of the units. Then, when constructed it will traverse the unit folders and exercise folders generating a data structure able to hold the information. It will contain also an outputHTML method that will return the string containing the HTML code needed to build the bar. Finally, it will contain also a method to get the CSS styles being used so they can be added to the web page. Consider that your root folder with the exercises may include files and/or empty folders, be prepared for those situations.

[1] E.g.: https://www.w3schools.com/css/css_dropdowns.asp

1.2. Tree Area

Trees have been used for decades to show the hierarchy among a set of files. There are no native trees in HTML, but they can be built using lists (again) and proper styling.

When one of the exercises is selected from the navigation bar, the tree will show a tree containing all the files of the exercise. Then the user will be able to select any of the files to do further actions over it.

To generate the tree you will need a tree class that gets the path to the folder to be constructed. It will traverse all the files contained in that folder (or any subsequent folder) an generate a structure of list items containing links. Then if you want to have a nice tree, you will apply some styles taken from the internet or even some JavaScript code to make the folders collapse and uncollapse when clicked.

I recommend adapting this:
http://web.archive.org/web/20181128113716/https://www.abeautifulsite.net/php-file-tree
This one can also be adapted: https://www.thecssninja.com/css/css-tree-menu & https://ryanseddon.com/demo/css_tree/

1.3. ContentArea:

The first part in the content area will be a set of buttons/links that will allow the user to select the action to be performed. This set of buttons will be built from the content class (using a similar strategy than with the previous ones) and will act as links, pointing to the proper URL. The buttons will allow you to perform different actions:

1.3.1. Show source:

It will present the contents of the file selected, without being processed. To do so, we just need to read the file (instead of accessing it through the HTTP) and escape it properly to make sure that is shown as it is an nothing is interpreted by the browser as HTML content (imaging how difficult is for a browser to show source HTML code without interpreting it into HTML elements).

In addition, we want our code highlighted, so we can read it faster and understand what it is doing (the same thing that is done by all IDEs, giving colours and styles to the different reserved words, functions, objects, etc…)
We will create a specific class for handling this that will receive as input the file being shown and will have a method to return the HTML code.
To do so we will use an existing library, that will make all the hard work for us: https://github.com/scrivo/highlight.php/tree/9.18

No points if you use the function highlight_file().

1.3.2. Show output:

This is the easiest one. We just need to show the output of calling a .php file from the browser. To do so we will be using the same method that we use to read whole files, but this time we will point to a file through the http scheme instead of pointing a file into our own computer. This will make it be pre-processed and we will get the output into a string, ready to be included into our webpage.

Some problems may happen depending on how the page is built and the information that includes but can be solved by filtering out the elements that we do not want before showing them (depending on your exercise you could end up with more than one body or head).

1.3.3. Download & Download ALL:

To download elements from the browser, the webpage where we are accessing needs to explain our browser that what will receive is a file to be stored in the computer. This is done through the headers, so our .php code will write some HTTP headers for the browsers, and will send the data inside the file to the clients computer.

Similarly, when a whole folder wants to be downloaded, we could repeat the download action several times (usually causes problems) or even better, create a .zip file including all the files that wants to be downloaded and then download that file.

Create a specific class with static methods to manage the downloads. As you will need to create a file and store it in the server (temporarily, you should delete it when you finish) you may have problems with the permissions, as you will need write permission in the folder where you store the file.

Tips:

- There is a single "page" for the whole project (index.php) and all the HTTP requests from the client-side will go to that page. You do not have to create more or to link to other pages. To make your page different each time is accessed you will need to provide parameters using GET, so you can tell PHP what file the user clicked, or if he wants to browse the source code or the output, etc.
- **Only 1 echo should be needed in the whole project!** It should be done from the index.php file to echo the output of the webBuilder (you can use more for debug if you want). Your functions should return strings, not print anything.
- For the zip file you will need to use headers (meta-info interchanged between browser and web server). The **headers must be sent before any other data**, so if you have any echo/print_r/var_dump executed before the header it will fail!
- All the navigation will be done using links (<a> HTML element), we do not need to create any JS code (you can if you wish but is not mandatory for any of the features). In order to work properly you will need to generate those links (from the PHP code) so they include the parameters needed in each case (so you should have some variables to represent the "current location" of the user and pass it back and forth encoded as part of the URL).
- You will have to handle some paths to files through the whole project, I recommend you to create a config.php file where all the things common to the project are specified (load of classes, declaration of constants with the root of the exercises, the root of the document, the url of the server, the name and values of the parameters, etc). Then you just need to include that single file in the files that use those constants.
- You have to go little by little, each part is independent and can be built and tested by separate, so do not expect to build the whole project from start to end without checking that each part is working by separate.
- Adding different styles from different sources that try to style the same elements (ul element for instance) will give you problems, be aware of those problems and solve them before is too late! (I am talking about the navigation bar and the tree).
- Be organized and divide the code into meaningful parts, being sure that they are complete and there is not too much dependency. If you create a class for the highlighting of text and that requires also the inclusion of a style, make sure that the same class is generating the link to the proper style. In my case I needed 8 different .php files that are below 50 lines of code each (except the webBuilder which is about 100).
- The behaviour in production server will be different to the one in localhost, do not wait until the last minute to upload it or it will fail. For instance, paths in unix systems use a different separator than in windows. \ vs /

Assessment Criteria:

- **Details about the assessment and extras will be discussed through the forums and in class**
- The web-app follows the requirements, the structure of the web-app follows the structure proposed and contains the elements described
- Quality and structure of the code. Include proper documentation of the code (comments and naming of functions and variables)
- Optional features implemented (if you think of different features that you would like to include, tell me before adding them).
- The division of the marks will be as follows:
    - WebBuilder ~ 2 points.
    - NavigationBar ~ 2 points.
    - Tree ~ 1.5 points.
    - Content area:
        - Show source: 1.5 point
        - Show output: 1 point
        - Downloads: 1 point.
    - Configure / Upload files / change highlight styles / other improvements: 1 point
- **To get the complete marks of each part you will need to explain it properly during the presentation, consisting of some slides with an overview of the project.**

Delivery:

- **Format:** A compressed folder including the project and the presentation. You can include an extra document explaining anything you wish but is not mandatory (the presentation is enough)
- **Deadline: 29 October.**
- **Presentation**: You will have to present the webapp, showing the features implemented, and explaining the technical aspects of the development. The teacher will place questions about the decisions taken during the development and the source code generated. **The webapp should be shown in the production environment! (in localhost won't be accepted).**