SMILE – Website Documentation

Last edited: May 3rd, 2014
Author: Radu Marian Tutunaru

The following document will present the documentation for the SMILE Website. The website was written in PHP, Javascript/jQuery, HTML5 and CSS3. It uses the Bootstrap 3 UI framework. The full documentation of the Bootstrap 3 elements is available at http://getbootstrap.com/.

1.Installation:

In order to install the website, you would to have a server which has PHP installed on it. In the PHP installation you would need to manually enable the curl extension in order for the REST functions to work.

If you already have a version of this website installed on your server, you would need just to drag and drop any new version over the old one in order to overwrite it.

2.Structure:

The website structure is pretty simple to gasp:

- The "**core**" folder holds the core back-end functions of the website and contains one folder "**inc**" and a one file "**init.core.php**".
    - The "**init.core.php**" file, starts the user session and acts as a controller for the entire website. It must be included in all php pages you create before anything else.
    - The "**inc**" folder contains the libraries developed for this project. One is the dynamic form library, which allows server retrieval of information from the dynamic forms used by this website. The second contains the REST functions used by this project.
- The second folder "**ext**" contains front-end scripts, styles, fonts and images of the website.
    - In order to edit any CSS styles please use the "**style.css**" available in the "**ext/styles/**" folder. If you add other CSS style files please add them in this folder and then import them at the beginning of the "**style.css**" document.
    - If you want to add any Javascript libraries please add them in "**ext/js/**" folder and remember to include them at the end of "**footer.php**" right before the "**</body>**" tag.
    - If you want to add other custom fonts, please add them to the "**ext/fonts**" folder, but remember to include them in your "**style.css**" using the font-face css rule. Also you will have to target each element that you want to change manually and apply the font CSS3 property.
    - Any images added to the project must be added to the "**ext/img**" folder.
- Finally all the other files the main "**smile**" folder are the pages created in the website.

3.REST & PHP:

In order to call the smile-server API, REST functions must be used. The website has the following REST functions converted and ready to be used:

REST verbose	Function name:	Parameters:	Returns:	Information:
GET	rest_get	$url, $headers	$response	The REST GET call function. Used to get information from the API.
POST	rest_post	$url, $data, $headers	$response	The REST POST call function. Used to add information to the API and return the added JSON object for further use. Not to be used with rest_post_status.
POST	rest_post_status	$url, $data, $headers	$code	The REST POST call function. Used to add information to the API and return the added status code of the operation. Not to be used with rest_post.
PUT	rest_put	$url, $data, $headers	$response	The REST PUT call function. Used to update information about an entity in the API.
DELETE	rest_delete	$url, $headers	$response	The REST DELETE call function. Used to delete all inforamtion about a specific entity in the API.

Parameters:

$url – String – The API url (ex: http://smile.abdn.ac.uk/smile-server/api-1.1/users)

$headers – String[] - The array of headers. Each header is a string.

$data – JSON - The JSON object passed to the API. See the API documentation for the format.

$response – JSON - The JSON object that that comes as a response after calling a REST function.

$code – Integer – The response code. Useful when you want to POST something to the API but return no JSON object.

Sending information to the API (steps):

1. Collect your data either from the form or from your global paremeters. Make sure you escape the data before submitting it to the API to avoid database attacks.
2. Create a variable that will hold the url of the rest call.
3. Create a string array containing all the needed headers.
4. Encode the array of information as JSON.
5. Call the needed rest methods (see above). Store the $response / $code in a variable.
6. Break the $response object in order to do future manipulation or to verify if specific conditions are matched.

Retrieving / delete information from the API (steps):

1. Create a variable that will hold the url of the rest call.
2. Create a string array containing all the needed headers.
3. Call the needed rest methods (see above). Store the $response in a variable.
4. Break the $response object in order to get access to each individual field or to see if the delete procedure was successful.

4. Dynamic forms:

Dynamic forms are created using javascript. Currently the project holds functionality in order to create up to 20 dynamic fields for each of the following fields: properties, products, categories, tags, identifiers. The scripts holding those are available in the "**ext/js/**" folder.

In order to correctly match the fields and be able to parse them the library two special functions are used. These functions are available in "**dynamic-form.inc.php**" inside the "**core/inc**" folder.

Function name:	Parameters:	Returns:	Information:
get_form_data_kv	$inputLabel, $inputValue	$formArray	Matches together input / value dynamic combinations. To be used in order to match properties, identifiers.

get_form_data_v	$inputValue	$values	Creates an array of dynamic values. To be used with tags, products and categories.

Parameters:

$inputLabel – String – The name of the label input.

$inputValue – String – The name of the value input.

$formArray – String[] - Returns nominal Key / value array.

$values – String [] - Returns an Array containing just the values.

5. Session:

The following values are stored in the session object:

Name:	Type:	Information:
email	String	The e-mail of the user.
name	String	The name of the user.
accountType	String	The type of account (business/user)
apiKey	String	The user API key.
businessApiKeys	String[]	Array containing all the business keys assigned to this user.
currentBusinessKey	String	The current business API key.

6. Geolocation service:

The geolocation service (using the geolocation API) is added automatically as long as the person completing the form allows it. The script is located in the "**ext/js/**" folder, called "**geolocation.js**".

It automatically completes the values of longitude and latitude in the create item form. To be used only with modern browsers: Chrome, Opera, IE9+, Firefox, Safari. This will provide the exact location of the device as long as the person is using a laptop, tablet or a smartphone. Cannot be used with a desktop.

7. QR code printing service:

The QR code printing service is available in "**ext/js**" under "**printQR.js**". It allows individual printing of QR codes, on an id that's passed as a parameter. The print styles are setup using CSS3 print media queries.

8. Searching:

Dynamic searching is available in "**ext/js**" under "**search.js**". The functionality is currently implemented for property labels, identifier labels and products. Although the REST call is made it still needs a small update on the API. The url of the API must be changed in order to support GET parameters.

Example:

Instead of:

http://smile.abdn.ac.uk/smile-server/api-1.1/search/property/key/{search word}

Use the GET url structure:

http://smile.abdn.ac.uk/smile-server/api-1.1/search/property?key={search word}

Once this is complete searching will work, but css styling of the list of results will be needed.

9. Status of the website:

Section:	Status:
User account system:	Complete
Items:	Complete
Collection:	Partially complete. Missing update collection page.
Product:	Partially complete. Missing update product page.
Activities:	Missing
QR printing:	Complete
Geolocation service:	Complete
Dynamic forms:	Complete
Searching:	Almost complete. Missing small changes to the API and list of results CSS style. (see above)

10. Future work:

1. Create update collection / update product. Follow the example I have created for update items.
2. Implement activites.
3. Fix the search issue.

For any further questions please e-mail me at: radu.tutunaru.10@aberdeen.ac.uk