EFRONT PRO 4

REST JSON API DOCUMENTATION

(Version 1.0)

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Documentation Index

Quick Start

In this quick start we provide you with some very basic information about how to access the API. More information and examples can be found in the following chapters.

1. Access through the command line (cURL):

```
curl -u <MY_API_KEY>: http://my-efront-pro.com/API/v1.0/System/Info
```

   - ✓ Replace <MY_API_KEY> with your API key.
   - ✓ Keep the : symbol after your API key without a space before.
   - ✓ Provide your domain and the API location. In the above example we use the API version 1.0 and request information about the system.

2. Access through the SDK:

```
$eFrontProSDK->GetAPI('System')->GetInfo();
```

   - ✓ See more on how to initialize $eFrontProSDK, here.

Note: To use JSONP, append to the endpoint "?callback=<myCallbackName>" without the quotation marks.

API Introduction

eFront PRO provides a comprehensive REST JSON API that allows interaction with remote services. Authentication is based on an API key that is defined under your installation's system settings. The functionality provided focuses on performing tasks meaningful for a remote service, such as user creation, course assignment, listing courses etc. In addition, one can use the API to provide basic SSO for users.

To ease implementation of services, we provide a SDK library for PHP that automates the tasks of communicating with the API.

The first part of this guide provides a detailed description of the available API calls, as well as information on authentication and error handling.

The second part of the guide demonstrates the use of the PHP SDK, providing information on setting it up and performing some basic tasks.

You can find the latest version of this guide here.
For comments and suggestions, visit here.

The Efront PRO team.

API Requirements

API doesn't have or require different technical requirements than eFront PRO. If your system meets the eFront PRO's requirements, then it also meets the requirements of the API.

At this point, you may want to properly configure your web server to achieve 2 goals, if your web server is different than Apache. If your web server is Apache there already exists an .htaccess configuration file inside the www/API folder, but you have to enable the mod_rewrite module (if not already enabled) on your own or inform your network/system administrator to do it for you.

1. The API recognizes only "pretty - SEO friendly" URLs, which means that you have to use a rewrite engine (for example Apache mod_rewrite) and set the minimum required conditions and rules.

2. Make sure that if the PHP interacts with your web server through the Fast-CGI protocol, usually there exists a problem with the HTTP authorization headers, so you have to pass the HTTP authorization headers to an environment variable.

API Calls Reference

Entity	HTTP	Call	Purpose
Account	POST	/Account/Status	Checks whether a user exists or not by providing the login name and the password.
Branch	GET	/Branches	Returns the branch list.
	GET	/Branch/:Id	Returns information about the branch with the associated Id.
	POST	/Branch	Creates a branch with the requested attributes.
Branch + User	PUT	/Branch/:Id/AddUser	Adds a user to the specified branch. The Id in the URL, refers to the branch. The Id of the user is defined as a PUT field, in the request.
Category	GET	/Categories	Returns the category list (tree structured).
	GET	/Category/:Id	Returns information about the category with the associated Id.
Course	GET	/Courses	Returns the course list.
	GET	/Course/:Id	Returns information about the course with the associated Id.
Course + User	GET	/CourseUserStatus/:CourseId,:UserId	Returns information about the status of a specified user in the specified course.
	POST	/CourseUserStatus/:CourseId,:UserId	Updates a user's information in the specified course.
	PUT	/Course/:Id/AddUser	Adds a user to the specified course. The Id in the URL, refers to the course. The Id of the user is defined as a PUT field, in the request.
	PUT	/Course/:Id/RemoveUser	Removes a user from the specified course. The Id in the URL, refers to the course. The Id of the user is defined as a PUT field, in the request.
Curriculum + User	PUT	/Curriculum/:Id/AddUser	Adds a user to the specified curriculum. The Id in the URL, refers to the curriculum. The Id of the user is defined as a PUT field, in the request.

	PUT	/Curriculum/:Id/RemoveUser	Removes a user from the specified curriculum. The Id in the URL, refers to the curriculum. The Id of the user is defined as a PUT field, in the request.
Group	GET	/Groups	Returns the entire group list.
	GET	/Group/:Id	Returns information about the group with the associated Id.
Group + User	PUT	/Group/:Id/AddUser	Adds a user to the specified group. The Id in the URL, refers to the group. The Id of the user is defined as a PUT field, in the request.
	PUT	/Group/:Id/RemoveUser	Removes a user from the specified group. The Id in the URL, refers to the group. The Id of the user is defined as a PUT field, in the request.
Plugin	GET	/Plugins	Returns a list of the available plugins and their information.
	GET	/Plugin/:pluginName	Same as the above, but for the requested plugin.
	POST	/Plugin/:pluginName	Posts data to be used by the requested plugin.
User	GET	/Autologin/:loginName	Returns the auto-login URL for the requested user.
	GET	/Logout/:loginName	Logouts the requested user.
	POST	/User	Creates a new user given some registration information. The information are defined as POST data fields. The required information to create successfully a user are the “login”, “name”, “surname”, “email” and “password”. In the future we plan to add some more fields.
	GET	/Users	Returns the entire user list.
	GET	/Users/:eMail	Returns a list of users which they have the requested e-mail address.
	GET	/User/:Id	Returns information about the user with the associated Id.
	PUT	/User/:Id	Edits the specified user.
	PUT	/User/:Id/Activate	Activates the specified user.

	PUT	/User/:Id/Deactivate	Deactivates the specified user.
System	GET	/System/Info	Returns information about the system.

API Error Handling

The error state handling of the API is very easy. In each response a key "**success**" is always included. This key contains the value "**true**" in case of success or "**false**" when something has gone wrong. That way you can easily check if the call to the API was succeed or failed.

Except that, we offer an additional methodology to be informed about the error states, and this is by the HTTP response status codes. Anything different than **200**, MUST be considered as an error state.

When an error has been occurred you can find into the body of the response some other useful information such as its code, a generic message and optionally a more technical reason. For a live demonstration of the API calls and their responses (succeed and failed) you can use this tool.

HTTP Status Code	Reason
503	✓ Service unavailable. API status is disabled.
405	✓ Unsupported HTTP method. Only GET, POST and PUT are currently supported.
404	✓ The requested API call does not exist. ✓ The requested entity (User, Course, etc) which is specified for example by an Id, does not exist.
401	✓ Authentication required.
400	✓ General HTTP status code, if the action can't be processed.

API Authentication

eFront PRO API doesn't offer any call that it's public, in other words that it isn't require authentication. So you MUST authenticate your requests in order to use the API. But before this step, you have to enable its status and find out your personal private API key.

To enable the API, just navigate to your "**System Settings**" through the eFront PRO administration control panel and proceed to the "**API**" tab. Check the "**Enable API**" checkbox and click on the "**Save**" button.

Once you have enabled the API, copy your personal private API key to use it later on the SDK. In case that someone finds out this key, you can always generate a new one by clicking on the "**refresh**" icon.

More information about how you MUST authenticate your requests (API calls) with your personal private key, you can read here.

SDK Introduction

With the eFront PRO SDK, you will be able to use its API easily and efficiently without advanced programming knowledge.

Programming Language: **PHP** | Version: **2.0.0** | API Support: **1.0**

SDK Requirements

If your system meets the eFront PRO's requirements, then it also meets the requirements of the SDK.

SDK Installation

In this chapter, we will analyze how to install your SDK. You MUST follow the below steps in order to work with the SDK:

1. Download the SDK (ZIP Archive).

2. Extract its contents anywhere you want inside your web server's document root. The document root is the folder where the website files for a domain name are stored. You SHOULD contact your administrator in case that you aren't sure about this action. It's RECOMMENDED to extract the contents inside the **www** folder of your eFront PRO web application.

3. Create a php file inside the Source folder (the folder that the AutoLoader.php file is located). There is no restriction about the name of this file, but it's RECOMMENDED to name it **index.php**.

4. Now paste the below code in the file you just create in order to start making calls:

```php
<?php

include 'AutoLoader.php';

use Epignosis\eFrontPro\Sdk\eFrontProSDK as eFrontProSDK;
use Epignosis\eFrontPro\Sdk\Factory\Handler\API as Api;
use Epignosis\eFrontPro\Sdk\Request\Handler\cURL as cUrl;

$apiVersion   = '1.0';
$apiLocation  = 'my-domain.com/API';
$apiKey       = '0123456789abcdef';
$eFrontProSDK = new eFrontProSDK(new Api(new cUrl));

$eFrontProSDK->Config($apiVersion, $apiLocation, $apiKey);
```

SDK Examples

In the previous chapter you learn how you can install the SDK. Moreover on step 4, you initialize the SDK with its dependencies, the version and of course your API key and its location.

So far you did a lot, which means that your requests now will be automatically authenticated and you won't have to worry about URL construction for each unique call of the API. That's the responsibility of the SDK.

In the below use cases, **each method of the GetAPI method** returns a string in JSON encoded format. You MUST decode it (json decode), in order to access the properties of the response. SDK doesn't decode automatically these responses/strings, because sometimes it's useful to store immediately this string into a database or create an array of multiple JSON encoded strings and do another work with it.

Finally it's always RECOMMENDED as a good practice, to use the SDK inside a try/catch block. For example:

```
try {
    // various SDK commands ..
} catch (\Exception $e) {
    echo 'Oops! An error occurred. [', $e->getMessage(), ', ', $e->getCode(), ']';
}
```

BASIC EXAMPLES

Check the status of an account.

```
$eFrontProSDK->GetAPI('Account')->Exists($loginName, $password);
```

Get all the branches.

```
$eFrontProSDK->GetAPI('BranchList')->GetAll();
```

Get information about a branch. GetInfo method, accepts a positive integer as the branch Id.

```
$eFrontProSDK->GetAPI('Branch')->GetInfo($branchId);
```

Create a branch. Create method, accepts an associative array as the branch information to be created. The required information consisted of the "name" and "url"; "parent_ID" and "public_ID" are optional.

```
$eFrontProSDK->GetAPI('Branch')->Create([

      'name' => 'foo', 'url' => 'foo', 'parent_ID' => 10, 'public_ID' => 'abc123'

]);
```

Add a user in a branch. AddRelation method, accepts 2 parameters which both are positive integers. The 1st one refers to the user's Id and the 2nd to the branch's Id.

```
$eFrontProSDK->GetAPI('BranchUser')->AddRelation($userId, $branchId);
```

Get all the categories (tree structured).

```
$eFrontProSDK->GetAPI('CategoryList')->GetAll();
```

Get information about a category. GetInfo method, accepts a positive integer as the category Id.

```
$eFrontProSDK->GetAPI('Category')->GetInfo($categoryId);
```

Get all the courses.

```
$eFrontProSDK->GetAPI('CourseList')->GetAll();
```

Get information about a course. GetInfo method, accepts a positive integer as the course Id.

```
$eFrontProSDK->GetAPI('Course')->GetInfo($courseId);
```

Get all the groups.

```
$eFrontProSDK->GetAPI('GroupList')->GetAll();
```

Get information about a group. GetInfo method, accepts a positive integer as the group Id.

```
$eFrontProSDK->GetAPI('Group')->GetInfo($groupId);
```

Get all the plugins.

```
$eFrontProSDK->GetAPI('Plugin')->GetAll();
```

Get information about a plugin. GetInfo method, accepts a string as the plugin name.

```
$eFrontProSDK->GetAPI('Plugin')->GetInfo($pluginName);
```

Notify the specified plugin by sending some data. Notify method, accepts a string as the plugin name (1st parameter) and an array (2nd parameter) with the custom notification data.

```
$eFrontProSDK->GetAPI('Plugin')->Notify($pluginName, $data);
```

Get all the users.

```
$eFrontProSDK->GetAPI('UserList')->GetAll();
```

Get all the users by their e-mail address. GetAllByMail method, accepts a string as the e-mail address of a user.

```
$eFrontProSDK->GetAPI('UserList')->GetAllByMail($mailAddress);
```

Get information about a user. GetInfo method, accepts a positive integer as the user Id.

```
$eFrontProSDK->GetAPI('User')->GetInfo($userId);
```

Activate a user. Activate method, accepts a positive integer as the user Id.

```
$eFrontProSDK->GetAPI('User')->Activate($userId);
```

Deactivate a user. Deactivate method, accepts a positive integer as the user Id.

```
$eFrontProSDK->GetAPI('User')->Deactivate($userId);
```

Create a user. Create method, accepts an associative array as the user's information to be created. The required information consisted of the login, name, surname, email and password fields.

```
$eFrontProSDK->GetAPI('User')->Create ([
    'login' => 'foo', 'name' => 'bar', 'surname' => 'baz',
    'email' => 'foo@bar.buz', 'password' => 'blackWhale'
]);
```

Edit a user. Edit method, accepts 2 parameters. The 1st parameter is a positive integer as the user Id and the 2nd an associative array as the user's information to be edited. The keys of the array are the same as the above method (Create) but aren't required all of them, so you can edit only the information which you want.

```
$eFrontProSDK->GetAPI('User')->Edit (
    $userId, ['login' => 'foo1', 'password' => 'blackWhale123']
);
```

Add a user in a group. AddRelation method, accepts 2 parameters which both are positive integers. The 1st one refers to the user's Id and the 2nd to the group's Id.

```
$eFrontProSDK->GetAPI('UserGroup')->AddRelation($userId, $groupId);
```

Remove a user from a group. RemoveRelation method, accepts 2 parameters which both are positive integers. The 1st one refers to the user's Id and the 2nd to the group's Id.

```
$eFrontProSDK->GetAPI('UserGroup')->RemoveRelation($userId, $groupId);
```

Add a user in a course. AddRelation method, accepts 3 parameters which. The 1st one refers to the user's Id (positive integer), the 2nd to the course's Id (positive integer) and 3rd to whether you want to force the operation or not, if the requested course belongs to a curriculum. The last parameter is set to false by default.

```
$eFrontProSDK->GetAPI('CourseUser')->AddRelation($userId, $courseId, $force);
```

Add a user in a curriculum. AddRelation method, accepts 3 parameters which. The 1st one refers to the user's Id (positive integer), the 2nd to the curriculum's Id (positive integer) and 3rd to whether you want to force the operation or not. The last parameter is set to false by default.

```
$eFrontProSDK->GetAPI('CurriculumUser')->AddRelation($userId, $curriculumId, $force);
```

Check the status of a user in a course. CheckStatus method, accepts 2 parameters which both are positive integers. The 1st one refers to the user's Id and the 2nd to the course's Id.

```
$eFrontProSDK->GetAPI('CourseUser')->CheckStatus($userId, $courseId);
```

Update the status of a user in a course. UpdateStatus method, accepts 3 parameters. The first 2 are positive integers. The 1st one refers to the user's Id and the 2nd to the course's Id. The last is an array which contains the update info.

```
$eFrontProSDK->GetAPI('CourseUser')->UpdateStatus (
      $userId, $courseId,
      ['score' => 100, 'to_timestamp' => 1418893082, 'status' => 'completed']
);
```

Remove a user from a course. RemoveRelation method, accepts 2 parameters which both are positive integers. The 1st one refers to the user's Id and the 2nd to the course's Id.

```
$eFrontProSDK->GetAPI('CourseUser')->RemoveRelation($userId, $courseId);
```

Remove a user from a curriculum. RemoveRelation method, accepts 2 parameters which both are positive integers. The 1st one refers to the user's Id and the 2nd to the curriculum's Id.

```
$eFrontProSDK->GetAPI('CurriculumUser')->RemoveRelation($userId, $curriculumId);
```

Get information about the system.

```
$eFrontProSDK->GetAPI('System')->GetInfo();
```

Autologin a user. Autologin method, accepts a string as the user's login name.

```
$eFrontProSDK->GetAPI('User')->AutoLogin($loginName);
```

Logout a user. Logout method, accepts a string as the user's login name.

```
$eFrontProSDK->GetAPI('User')->Logout($loginName);
```

ADVANCED EXAMPLES

Logout all the users:

```
try {
    // See page 11 ..

    // Fetch all the users:
    $userList = json decode (
        $eFrontProSDK->GetAPI('UserList')->GetAll(), true
    );

    // Check if the call was succeed:
    if (!$userList['success']) {
        throw new \Exception (
            $userList['error']['message'], $userList['error']['code']
        );
    }

    // Iterate through the user list:
    foreach ($userList['data'] as $user) {
        $logoutResponse = json decode (
            $eFrontProSDK->GetAPI('User')->Logout($user['login']),
            true
        );

        echo 'User <b>', $user['login'], '</b> was ';

        // Check whether the logout process was succeed or not:
        if ($logoutResponse['success']) {
            echo 'logout with success.<br>';
        } else {
            echo 'not possible to logout.<br>';
        }
    }
} catch (\Exception $e) {
    echo $e->getMessage();
}
```

Activate all the users with odd Id and deactivate these with even Id:

```
try {
    // See page 11 ..

    // Fetch all the users:
    $userList = json decode (
        $eFrontProSDK->GetAPI('UserList')->GetAll(), true
    );

    // Check if the call was succeed:
    if (!$userList['success']) {
        throw new \Exception (
            $userList['error']['message'], $userList['error']['code']
        );
    }

    // Iterate through the user list:
    foreach ($userList['data'] as $user) {
        $apiUser    = $eFrontProSDK->GetAPI('User');
        $evenNumber = $user['id'] % 2 == 0;

        $response =
            ($evenNumber)
                ? json decode (
                      $eFrontProSDK->GetAPI('User')->Deactivate($user['id']),
                      true
                  )
                : json decode (
                      $eFrontProSDK->GetAPI('User')->Activate($user['id']),
                      true
                  );

        echo 'User <b>', $user['login'], '</b> was ';

        if ($response['success']) {
            echo $evenNumber ? 'deactivated' : 'activated', ' with success.<br>';
        } else {
            echo 'not possible to ', $evenNumber ? 'deactivated' : 'activated', '<br>';
        }
    }
} catch (\Exception $e) {
    echo $e->getMessage();
}
```

Create a user (assuming a male), assign him to a course and get the login URL by auto login him:

```
try {
    // See page 11 ..

    // Create the user:
    $userInfo = [
        'login'    => 'efront',
        'name'     => 'efront',
        'surname'  => 'efront',
        'email'    => 'xarhsdev@efrontlearning.net',
        'password' => 'foobarbuz'
    ];

    $userCreation = json_decode (
        $eFrontProSDK->GetAPI('User')->Create($userInfo), true
    );

    // Throw an exception if the creation was failed:
    if (!$userCreation['success']) {
        throw new \Exception (
            $userCreation['error']['message'], $userCreation['error']['code']
        );
    }

    // Get the course list:
    $courseList = json_decode (
        $eFrontProSDK->GetAPI('CourseList')->GetAll(), true
    );

    // Throw an exception if the call was failed:
    if (!$courseList['success']) {
        throw new \Exception (
            $courseList['error']['message'], $courseList['error']['code']
        );
    }

    // Assign our user to the 1st course of the course list:
    $course = reset($courseList['data']);

    $courseAssignResult = json_decode (
        $eFrontProSDK->GetAPI('CourseUser')->AddRelation (
            // The Create method was returned his Id:
            $userCreation['data']['id'], $course['id']
        ),
        true
    );
```

```
    // Throw an exception if the call was failed:
    if (!$courseAssignResult['success']) {
        throw new \Exception (
            $courseAssignResult['error']['message'],
            $courseAssignResult['error']['code']
        );
    }

    // Fetch the auto-login URL:
    $autoLoginResult = json decode (
        $eFrontProSDK->GetAPI('User')->Autologin($userInfo['login']), true
    );

    // Throw an exception if the call was failed:
    if (!$autoLoginResult['success']) {
        throw new \Exception (
            $autoLoginResult['error']['message'], $autoLoginResult['error']['code']
        );
    }

    $autoLoginURL = $autoLoginResult['data'];

    // ..
} catch (\Exception $e) {
    echo $e->getMessage();
}
```

For each registered user, print information about courses:

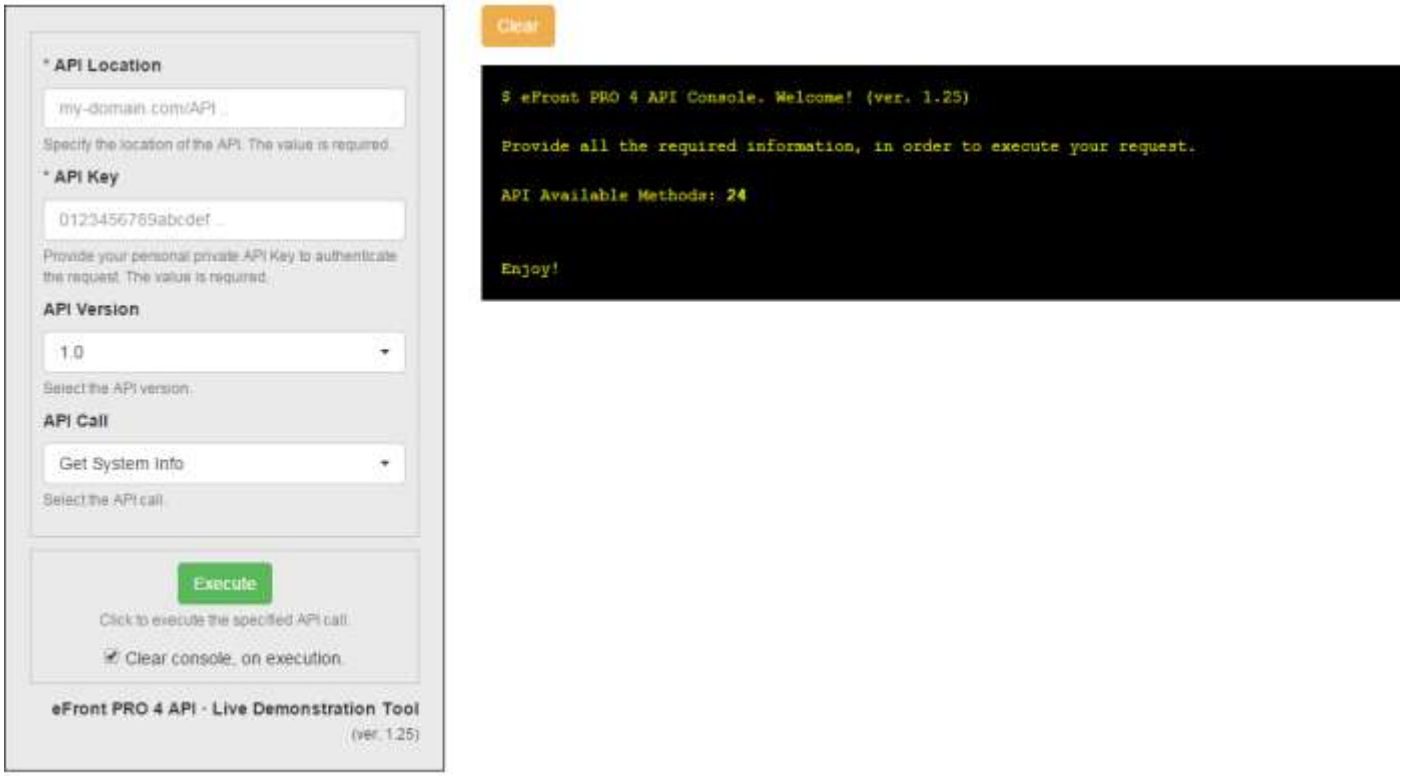
```php
<?php

try {
    // See page 11 ..

    // Fetch the user list:
    $userList = json_decode(
        $eFrontProSDK->GetAPI('UserList')->GetAll(), true
    );

    if (!$userList['success']) {
        throw new \RuntimeException (
            $userList['error']['message'],
            $userList['error']['code']
        );
    }

    echo
        '<table border=3 cellpadding=10><tbody><tr>',
        '<th>Id</th>',
        '<th>Login</th>',
        '<th>Name</th>',
        '<th>Surname</th>',
        '<th>e-Mail</th>',
        '<th>Courses [Name, Status]</th>',
        '<th>Avg. Score</th></tr>';

    foreach ($userList['data'] as $user) {
        // For the current user fetch the courses that he/she is
        // registered to:
        $userInfo = json_decode (
            $eFrontProSDK->GetAPI('User')->GetInfo($user['id']), true
        );

        if (!$userInfo['success']) {
            throw new \RuntimeException (
                $userInfo['error']['message'],
                $userInfo['error']['code']
            );
        }

        echo
            '<tr><td>', $user['id'], '</td>',
            '<td>', $user['login'], '</td>',
            '<td>', $user['name'], '</td>',
            '<td>', $user['surname'], '</td>',
            '<td>', $user['email'], '</td><td>';

        $courseList = $userInfo['data']['courses']['list'];

        if (empty($courseList)) {
            $avgScore = '-';

            echo '-';
        } else {
            $avgScore = 0.0;
            $c = 0;

            foreach ($courseList as $info) {
                $avgScore += $info['score'];
                $c++;
```

```
                echo
                    sprintf (
                        '[%s, %s]<br>',
                        $info['formatted_name'],
                        $info['status']
                    );
            }

            if ($c > 0) {
                $avgScore /= $c;
                $avgScore = round($avgScore, 2);
            }
        }

        echo '</td><td>' . $avgScore . '</td></tr>';
    }

    echo '</tbody></table>';
} catch (\Exception $e) {
    echo $e->getMessage();
}
```

API Live Demonstration Tool

There is also a folder named "**API-Live**". This folder contains a live demonstration tool, and can be accessed through your web browser. This tool can help you to:

- ✓ Test before you start to develop (or during the development of) your application.
- ✓ Understand the API philosophy.
- ✓ Have a detailed overview of any of its responses.



Please keep in mind that you MUST secure this folder with additional methods (htaccess, etc), to prevent unauthorized access or completely remove the folder on a live/production environment.