Notes on the Development for a PHP SDK for the eTilbudsavis API

Let me start by expressing appreciation towards the API documentation that was provided to me. While it contains some incorrect information (more on that soon), it is indispensable for being able to complete my task in a timely manner.

Skipping the research details, the first implementation problem revolved around which library to use for the HTTP Requests. I have always used the *pecl_http* and sometimes sockets to get the job done, but in this case sockets seemed archaic, so I tried using *pecl_http* but, to my surprise, I found out it was no longer a basic DLL in my PHP installation. This led me to research the matter for a while and I discovered that it is actually not used that much and the default now seems to be *cURL*, which is indeed a basic library for my PHP installation as well. Since I did not want to add new library dependencies to the SDK, and I assume you as well as app developers have *cURL* installed, I stuck with it even if the syntax is, in my opinion, clumsier than *pecl_http*.

I then proceeded to study the library as I had never used it before and creating an interface and a class for useful pre-defined requests (*GET, POST, PUT, DELETE*). I think that having a separate class with these function is tidier and allows for better code reuse. Some of the options of each request are the same for all four functions but I still refrained from putting them all together as private default variables, just in case one might need to change just one option in just one request type (for example setting `CURLOPT_Header` to `0` in just `curl_put()`), so that is my choice.

Then I stumbled to the first documentation-related problem. No matter how I constructed my function to obtain an API session I stumbled on a *404* page. After several attempts, I even tried with the *Postman REST* client and still ended up with the same error. I was almost ready to send an email to Henrik when I decided to try with a different Host (*https://api.etilbudsavis.dk* instead of *eta.dk*) and it started working. Sort of. I was now getting a different error message, which was resolved by using the full URL for the request (regardless of what it was written in the Host field) and ignoring local SSL certificates with "`CURLOPT_SSL_VERIFYPEER => false`," to bypass a SSL error that persisted even after installing different/new certificates on localhost. This option should probably be removed when using the final version of the SDK.

Other design choices were adding a third parameter to the `initialize()` function, `$v1` which is considered an array of credentials in case someone wants to authenticate with a v1 API key (as the documentation says) as well as creating a separate class for user credentials so that they are private members and can only be accessed with getter methods.

A choice I am thinking about is to also pass the current token to `initialize()` so I can try to renew the session if it already exists, before creating a new one, but then "*initialize*" would not be semantically correct anymore, so maybe it is better to do that somewhere else, I just wanted you to know I thought about it. I am also perplexed on why the API Secret is passed to `initialize()`, from what I can see it is only used to sign requests, and you need a token to do that, which implies you already have a session going. I have therefore removed it.

Before implementing optional functions I took my time to sort out code structure and comments.

Public methods that have an interface are commented in the interface file, private methods are commented in the implementation. I tried to avoid repeating comments too much where it is not needed.

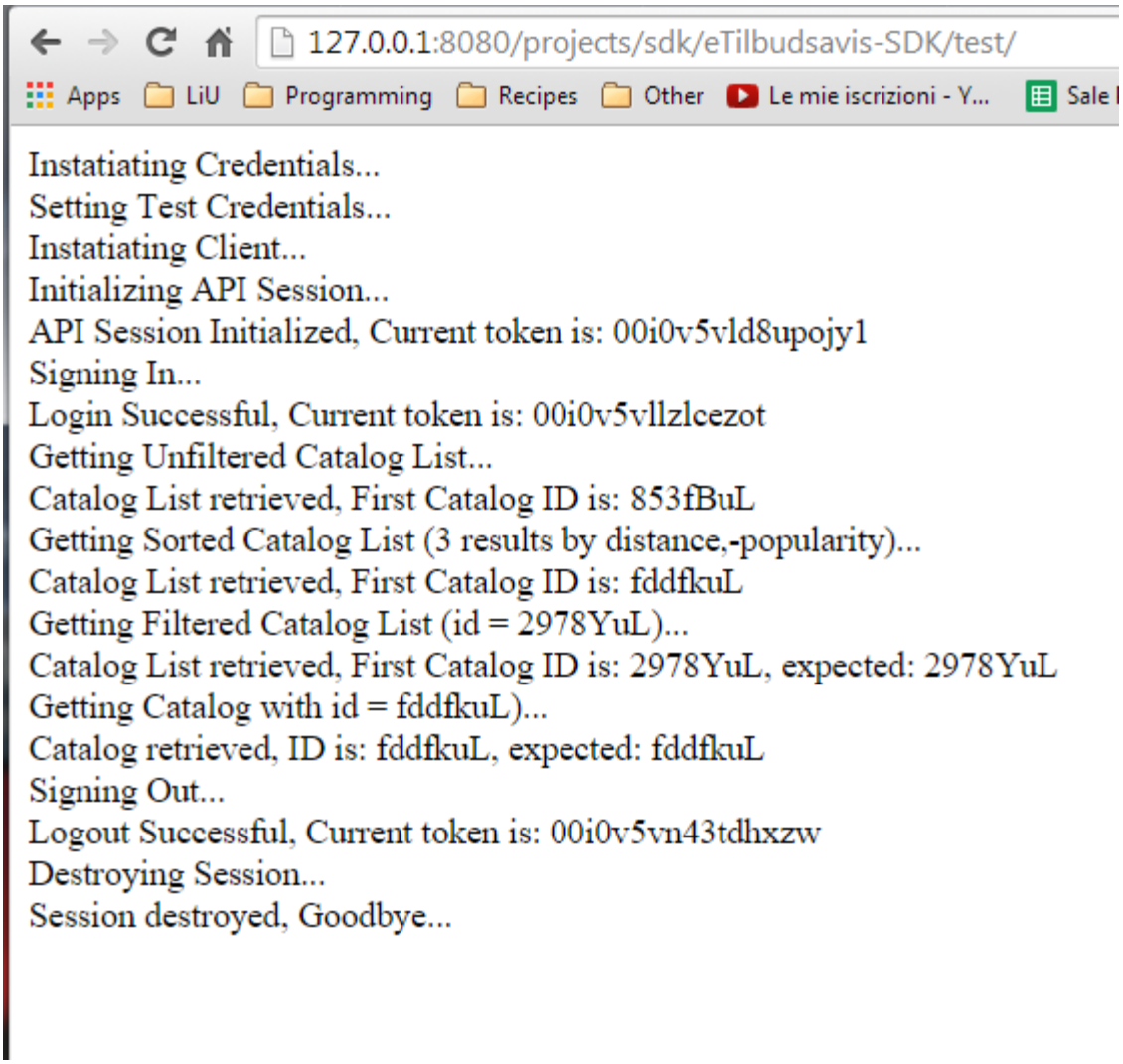
I also decided to have a separate class for credential values with getters and setters for each and a general `getCredentials()` method to get all of them at once. I still define my own credentials in the test file (*test/index.php*) but I think it is tidier to have a separate class handling them so that an end user is pushed towards using an OOP design for his application.

test/index.php defines the testing of the SDK. Each implemented function is run at least once, instantiation of classes is checked and errors are reported. If you set the global variable `$verbose` to `1` you will get messages for every step of the testing process, otherwise a blank page is the result of a successful test. The strings echoed give enough insight about what is going on that I decided additional comments would be superfluous.

Regarding the optional functions to implement I decided to start from getting a catalog list or a single catalog. `getCatalogList()` checks for all the required fields (as per the documentation) and builds a query based on those and valid optional fields if they are present (especially, strings for sorting options are checked against an array containing only valid ones). The return value is an array of objects containing the catalog properties. I returned this as a PHP resource for consistency with other functions (which return 1 object), but there might be a more elegant solution. `getCatalog()` is treated as a special case of `getCatalogList()` where all optional fields are stripped (if they were present in the call) but 1 *catalogs_id* is required, if more than one is present, only the first one is used to call `getCatalogList()` and retrieve only the one wanted catalog. The only thing I do not like about this section is the sheer length of function comments in the interface as `$options` needs to be explained properly, but at the moment I could not come up with a better solution (maybe external documentation, but so far that was the only thing that required extensive explanation). These 2 functions are tested a total of 4 times in the test file. In the beginning only required fields are set and an unfiltered, unsorted, full catalog list is retrieved. Then sorting and limiters are defined and only the first 3 sorted results are retrieved. Finally filters are added (*catalog_ids* makes the others pointless, but it can be commented out to check that the other filters also work), and only the filtered result is displayed and compared to the expected one (from the id that the user defined). In the same fashion `getCatalog()` is tested with 1 expected ID compared to the retrieved one.

It is Saturday evening and I will leave the last day to just re-test and polish my code. I have started working on the project later than I wanted to, but apartment-related things kept me from programming. Anyways from what I have seen in the documentation, the other optional functions should not be too different or require different skills and knowledge from what I already implemented, so, hopefully, you should have a clear idea of what I can bring to the team.

Finally I will add a screenshot of the current, successful, verbose testing page as I see it (hopefully nothing gets broken after committing files and whatnot).


127.0.0.1:8080/projects/sdk/eTilbudsavis-SDK/test/
Apps
LiU
Programming
Recipes
Other
Le mie iscrizioni - Y...
Instatiating Credentials...
Setting Test Credentials...
Instatiating Client...
Initializing API Session...
API Session Initialized, Current token is: 00i0v5vld8upojy1
Signing In...
Login Successful, Current token is: 00i0v5vllzlcezot
Getting Unfiltered Catalog List...
Catalog List retrieved, First Catalog ID is: 853fBuL
Getting Sorted Catalog List (3 results by distance,-popularity)...
Catalog List retrieved, First Catalog ID is: fddfkuL
Getting Filtered Catalog List (id = 2978YuL)...
Catalog List retrieved, First Catalog ID is: 2978YuL, expected: 2978YuL
Getting Catalog with id = fddfkuL)...
Catalog retrieved, ID is: fddfkuL, expected: fddfkuL
Signing Out...
Logout Successful, Current token is: 00i0v5vn43tdhxzw
Destroying Session...
Session destroyed, Goodbye...