Content

JavaScript Client API

The Client API is the interface that is provided by the vaccinator JavaScript API for client software developers.

Include the vaccinator JavaScript API and all needed classes into your existing application using this in the <head> section:

```
<!-- start vaccinator include -->
<script src="localforage.min.js"></script>
<script src="forge-sha256.min.js"></script>
<script src="jschacha20.js"></script>
<script src="vaccinator_api.js"></script>
<!-- end vaccinator include -->
```

Now, there is a new class `vaccinator` available.

Please note that the *localforage.min.js* is for local database access, the *jchacha20.js* is for encryption, the *forge-sha256.min.js* is for providing hash algorithm and *vaccinator.js* is the final class code you want to use.

All class functions except the validateAppId() function are asynchronous and return a promise. Don't forget to wrap a try/catch block around your class calls to handle potential errors thrown by the vaccinator class like in this example:

```
try {
  var a = new vaccinator();
  a.init("https://serviceprovider.com/service.php", "username", "appid", "password", false)
  .then(function() {
    console.log("Successfully initialized vaccinator class");
  }
} catch (e) {
  // catch any vaccinator class errors from here. e is vaccinatorError class.
  console.error(e);
}
```

The vaccinator class offers the following functions:

init

Description:	Initialize a new vaccinator session.
Parameters:	(string) serviceURL, (string) user-identifier, (optional string) app-id, (optional string) password, optional boolean) debugMode
Return value:	(promise) (boolean) true = success
Info:	`serviceURl` is the URL where the API endpoint at the service provider is located. For example: "https://service-provider.tld/protocol". All POST calls of the API will be sent to this destination. Please note that "same origin" policy might affect this. In the best case, this is the same domain than your app is running at. `user-identifier` is some mandatory identifier that the class is using to handle different aspects of saving user related information (like app-id). Also, the class is submitting this information as `uid` parameter in all protocol calls to the service provider. We suggest to use the name of the user who is using your application (eg email address). `app-id` is the end users application password for additional encryption. The vaccinator class expects some app-id known. If not submitted or undefined, the class is trying to get it from previous calls (local database). It throws an error if this fails. `password` is used to encrypt the app-id in the local storage database. If not submitted or undefined, the app-id will get stored without any encryption (not recommended). We recommend to submit the password the user entered for log-in to your application. By this, the local database will not leak the app-id in case someone is trying to read the browser database. Set `debugMode` to true in order to activate debug output to browser console. Mainly for finding bugs by the developer of vaccinator service class but maybe also helpful for you.

userNew

Description:	Create a new user entry.
Parameters:	(string) payload
Return value:	(promise) (string) PID
Info:	The `payload` is some JSON encoded dataset. It may contain personal information of a person. This is then returned later by `userGet`.

userUpdate

Description:	Update an existing user entry.
Parameters:	(string) PID, (string) payload
Return value:	(promise) (string) PID
Info:	The `PID` is the identifying person ID (for example, previously returned by userNew). The `payload` is some JSON encoded dataset. It may contain personal information of a person.

userGet

Description:	Retrieve the payload of a given user entry.
Parameters:	(array) multiple PIDs or (string) PID
Return value:	(promise) (object array) payload

Info:	The submitted `PID` is the identifying person ID (previously returned by userNew). Multiple PIDs can be submitted as array with multiple PIDs or a string with multiple PIDs divided by blank.
	The returned payload is an associative object array with the `PID` as key and some object as value. The value object is having two fields: `status` (OK or NOTFOUND) and `data` (the payload). If `status` is NOTFOUND, data is false.
	This is a typical object array response like displayed in Firefox console:
	`0d52f1b0a314fba7d45e87ca5bf5e654:`
	`  Object { status: "OK",`
	`  data: "{\"fn\":\"Spongebob\",\"ln\":\"Squarepants\"}"`
	`  }`
	`1d52f1b0a314fba7d45e87ca5bf5e654:`
	`  Object { status: "NOTFOUND",`
	`  data: false`
	`  }`
	`fb9a6fd4c504878b2a76d9e78af795bb:`
	`  Object { status: "OK",`
	`  data: "{\"fn\":\"Patrick\",\"ln\":\"Star\"}"`
	`  }`
	Access the results like this:
	`var status = result['0d52f1b0a314fba7d45e87ca5bf5e654']['status];`

userDelete

Description:	Delete the given user entry.
Parameters:	(array) multiple PIDs or (string) PID
Return value:	(promise) (array) PID(s)
Info:	The `PID` is the identifying person ID (for example, returned by userNew). Multiple PIDs can be submitted as array with multiple PIDs or a string with multiple PIDs divided by blank.

userWipe

Description:	Wipe the given user entry from the local cache (does not delete data from vaccinator service!)
Parameters:	(array) multiple PIDs or (string) PID
Return value:	(promise) (array) PID(s)
Info:	The `PID` is the identifying person ID (for example, returned by userNew). Multiple PIDs can be submitted as array with multiple PIDs or a string with multiple PIDs divided by blank. Please note that, if the `PID` is requested after this was called, the system will request it again from the vaccinator service and will update the cache. A possible use case is, if you know that the local cache is outdated for this `PID`, you can force the system to refresh its cache by wiping the user with this function.

wipeCache

Description:	Wipe all locally cached information.
Parameters:	(string) token (optional, unset or empty string to force wipe)
Return value:	(promise) (boolean) true = cache was wiped, false = cache stayed untouched
Info:	This wipes all local cached information. In case the given token (eg time stamp) is different to the one used before, or even unset or empty, it will wipe the cache. There are two use-cases: 1) If the service provider is sending a time stamp (refer to "update person" vaccinator protocol function). In this case, call wipeCache() with the given time stamp as token. If the token differs from last time, this function will wipe the whole cache. New requests will restore the cache step by step. By this, your local cache is always up to date. 2) If the application was used in Internet café or other security concerns are against permanent local caching (please note that the caching massively increases speed of the whole system). After the cache was wiped, all data has to become requested from the vaccinator service again if requested. Thus, please call this function (if needed) with no token regularly after logout (in this situation).

changeAppId (not yet implemented)

Description:	This is trying to re-encode all payloads after the app-id has changed.
Parameters:	(array) PIDs, (string) old app-id, (string) new app-id, (reference) progressCallback
Return value:	(promise) (int) 0 = success, >0 = error code
Info:	The app-id is used to encrypt the payload in identity management. For whatever reason, if the app-id is changing for a user, then all entries in identity management need to become re-encrypted. Obviously, this is not to be done on identity management place to protect the data. So it must be done locally. For this, the API class downloads and decrypts all payloads. Then it logs out initializes again with the new app-id. Then, all payloads are getting encrypted and updated. `PIDs` is one or more PIDs. Please submit as array. This list has to be complete! In doubt, make sure you have the list of ALL PIDs for this app-id. `old app-id` and `new app-id` are the old and new app-id to use for re-encryption. `progressCallback` is some JS function reference. It will get called if the re-encryption is finished. In this case, you can bring back functionality and remove any "please wait" notifications. Submit false in case you do not want (not recommended). The whole process may take a long time, depending on the number of people affected. Until the promise is fulfilled you should show some "please wait" dialogue to tell the user that something is going on in the background. **NOTE:** It is important that this call contains ALL PIDs assigned to the given app-id. If not, some data in vaccinator service may stay encrypted with the old app-id. In the worst case, this would cause serious data loss. **NOTE:** In case this function was interrupted, there is a chance that some entries in vaccinator service may be encrypted with the new app-id and other still with the old one. The API is making sure that only payloads encrypted with the old app-id get re-encrypted (by using the cs value from the payload). By this, it is possible to call this function multiple times (with exactly the same parameters) to fix any previous interruption.

getAppId

Description:	Returns the app-id that is currently in use.
Parameters:	-
Return value:	(promise) (string) app-id
Info:	If no app-id is available, it throws an error!

validateAppId

Description:	Validates the checksum of the given app-id
Parameters:	(string) app-id
Return value:	(boolean) validity
Info:	Returns true if the given app-id contains a valid checksum. Returns false if not.

Error handling

The vaccinator class throws error of type `vaccinatorError` in case something goes wrong. The `vaccinatorError` inherits the JavaScript Error class and adds two additional values:

reason: It is one of the following reasons of the error:

`VACCINATOR_SERVICE` = The vaccinator service is the reason for the problem. Check vaccinatorCode value for more details.

`VACCINATOR_INVALID` = You very likely submitted some invalid or missing parameter. Not vaccinator related but related to your input.

`VACCINATOR_UNKNOWN` = Error with no further specification.

vaccinatorCode: In case the reason was VACCINATOR_SERVICE, this code contains the return code from vaccinator service.

In general, if you get an error of reason VACCINATOR_SERVICE, you have to validate the vaccinatorCode and maybe inform the user about some issues that may go away in some time (try later). If you get some VACCINATOR_INVALID, you very like passed in some parameter or values that do either not fit to the rules or are invalid or of wrong type.