Userservice API

Userservice API
Morten Fæster

Latest published version of the Userservice API dokumentation is always available here: https://userservice.jppol.dk/doc/Userservice%20API.pdf

Document history

Version	Date	Comment
Current Version (v. 3)	**Feb 06, 2018 11:05**	**Morten Fæster** : Link to changelog iterator example inserted.
v. 2	Nov 24, 2017 15:36	**Morten Fæster** : Showing PUT implementation for authenticate. Mentions swagger
v. 1	Aug 25, 2017 14:31	**Morten Fæster** : Reservation of user profiles described. Added "reserve" endpoint and extended description of Create, Lookup and

2014-05-02: Custom email function uses POST, not PUT.
2014-05-19: Added "IsLockedOut" in response from validate and authenticate methods.
2014-07-31: Added appendix on request signing.
2014-08-07: Added sections "Querying users", "Access user history" and "Test client"
2015-03-10: Describing the "created()" lookup 2015-04-09: !isnull(any(property)) clearification
2015-10-13: Http status codes described
2016-03-08: Mentioned the possibility of sending different custom mails as part of a branding.
2016-08-23: Added Event time to change log result.
2016-10-13: Added validateEmail-method.
2016-10-29: Create users with password added (under create).
2016-11-02: Updated description of status codes from password validation.
2017-01-24: Added mention of usernameDomain() search function. Mentioning new behaviour on null updates.

At a glance

Medielogin consists of three main components, namely the OpenID based authentication system https://polid.jppol.dk, the user and group handling site https://medielogin.dk and the Userservice API, which exposes all the major user features to external applications.

All Userservice requests must be signed an application specific key. The application will be allowed to perform a subset of the features described in this document based on the application credentials.
The Userservice enables basic CRUD functionality (create, read, update and delete users), as

well as specialized functionality specific to the medielogin. When reading and writing user data the application key given will determine which user attributes the caller is permitted to read and manipulate.
In order to avoid race conditions all CRUD request to the user service must contain a "Version" timestamp indicating the latest user state known to the application.

Swagger / OpenAPI spec

While this document continues to be the authoritative documentation for the Userservice API it is also possible to get an OpenAPI 3.0.0 definition at /userservice-swagger.json or to access the Swagger UI at /swagger

Please note that this spec also describes the OAuth2 implementation available.

Maintaining user state

The main user update operations are shown with examples below.

Request/response format

Http request headers

The following headers are a necessary part of the request/response structure.

Authorization

Contains a specially crafted signature string. Please see the appendix about request signing.

Accept

The accept header instructs the service about the desired response format. The choices are limited to json, but the response comes in two flavours. "application/json" yields dates in a long and rightfully forgotten format invented by Microsoft where dates are serialized as ms since epoch January 1st 1970 inserted into \/Date(msSinceEpock)\/.

Use the value **"application/json+jsondate"** to get a formatting ISO8601.

Content-Type:

Specifies payload encoding. Again **"application/json+jsondate"** is likely the right choice.

Payload

Userservice request/response payload


```
{
    "Identifier": "9c6c3b24-8688-43f1-aa2f-b31b806dfdbd",
    "Properties": {
        "a": null,
        "Activated": "2017-08-10T09:52:16.131Z",
        "ActivationCode": null,
        "Activations": [
            "2017-01-31T14:06:56.791Z",
            "2017-02-07T10:44:08.314Z",
            "2017-04-21T09:50:22.369Z",
            "2017-05-11T08:59:29.084Z"
        ],
        "Birthdate": "1978-10-05T12:00:00.000Z",
        "Branding": "MLI.mli",
        "CareOfName": null,
        "City": "København N",
        "Country": "DK",
        "EbNewsletterSubscriptions": [
            "ekstra-samfund-og-politik",
            "ekstra-servicemail-tvungen"
        ],
        "JppolPermission": 68,
        "JppolPermissionUrl":
"https://medielogin.dk/ekstra-bladet/UpdateConsent",
        "SubscriptionID_POL": [
            "57505299"
        ],
        "Username": "faester@gmail.com",
    },
    "Version": "636379555361411897",
    "Created": "2017-01-31T14:04:55.681Z",
    "PendingDelete": null,
 "State": "LoginCreated"
}
```


The CRUD requests and responses from the userservice all share the format shown here. Please consult the chapter "User state" to get a definition of the "State" property.

The property bag

User attributes that can be manipulated by the caller are all contained within the "Properties" dictionary. The individual properties are constrained on data types (dates, ints, doubles and string and arrays of any of these data types). An application may be allowed to read more values than it is allowed to update thus receiving values it cannot change.

Pending delete

If a user has requested a deletion of the account, the PendingDelete is set. The value will then describe at which point in time the account is deleted. (The user can cancel the deletion by logging in until this period.)

Create time

The user creation time is contained within the Created property.

Latest change

In the Version property the latest change time for the user is stored. The value is a .NET tick value registered at the server at the time of the latest change. The Version must be given in the If-Not-Modified header for all subsequent updates.

Dates

All dates take the rather unusual format "/Date(milli seconds since 1970-01-01)/". The format was proposed by Microsoft when the JSON specification had not yet ripened and is an unfortunate legacy part of the user service.

API methods

Read

Reading user data happens one user at a time. The user is identified using the users id as the last component in the URL.

Endpoint	/user/{userid} https://userservice.jppol.dk/SSOUser.svc/user/c3131858-67ec-4a96-a06d-d5e65

HTTP method	GET
Request header	HTTP/1.1 AuthorizationDate: Mon, 17 Feb 2014 23:16:13 GMT Authorization: MIDS 39a2fc1e-9761-45fa-a96c-8d85d013f2d2 08MuWITB9vbWyulbqx2+4n3Vu9w= If-None-Match: Host: userservice.jppol.dk
Request body	*<empty>*
Response header	a06d-d5e65d30d691 HTTP/1.1 AuthorizationDate: Mon, 17 Feb 2014 23:16:13 GMT Authorization: MIDS 39a2fc1e-9761-45fa-a96c-8d85d013f2d2 08MuWITB9vbWyulbqx2+4n3Vu9w= If-None-Match: Host: userservice.jppol.dk

Response body	**Userservice get response**
	{
	"Identifier": "c3131858-67ec-4a96-a06d-d5e65d30d691",
	"Properties": {
	"Username": "faester@gmail.com",
	"Gender": null,
	"Activated": "2017-01-31T14:06:56.791Z",
	"SubscriptionID_POL": [
	"57505299"
	],
	"Activations": [
	"2017-01-31T14:06:56.791Z",
	"2017-02-07T10:44:08.314Z",
	"2017-04-21T09:50:22.369Z",
	"2017-05-11T08:59:29.084Z"
	],
	},
	"Version": 635282413468023718,
	"Created": "2017-01-30T14:06:56.791Z",
	"PendingDelete": null,
	"State": "Activated"
	}

Create

When users are created in the userservice, an email based on the template named in the "Branding" property is sent. The user is assigned a temporary password, and the system will create a one time password (an "ActivationCode") that is communicated to the user in the email, allowing the user to choose a real password of the users own liking. The "ActivationCode" can be defined in the request if allowed by the client application permissions. If unset it will always be defined by the user service as a 10 character value.

Endpoint	/user/{userid} https://userservice.jppol.dk/SSOUser.svc/user/b1fcbf2e-0109-4b1c-8955-da30a0
HTTP method	POST
Request header	HTTP/1.1 AuthorizationDate: Thu, 13 Mar 2014 09:54:44 GMT Authorization: MIDS 39a2fc1e-9761-45fa-a96c-8d85d013f2d2 TB4pMjktKFNj7qpZGJnVSB2l05g= If-Match: 635303011306457500 Host: userservice.jppol.dk Content-Length: 310

Request body	**Userservice POST request**
	`{"Identifier": "b1fcbf2e-0109-4b1c-8955-da30a09b1d49",`
	`"Properties":{`
	`"Branding":"POL2.prod",`
	`"Username": "faester+example@gmail.com",`
	`"FirstName": "Morten",`
	`"Birthdate":"1978-10-05T12:00:00.000Z",`
	`"SubscriptionID_POL": [ "57505299" ],`
	`"LastName": null`
	`},`
	`”Password”: ”qwerty1234isbad”`
	`}`

Response header	HTTP/1.1 200 OK Server: Microsoft-IIS/7.5 X-Powered-By: ASP.NET Date: Thu, 13 Mar 2014 09:54:44 GMT Content-Length: 0
Response body	*<empty>*

All properties writeable to the client application can be set in the create.
All applications allowed to create users can write to the "Username" field in the create

request. The field must contain a valid email address. Changing this field during updates are usually not possible due to permission settings.
If defined in the "Branding", the user will receive an activation reminder after a certain period if unactivated. This mail body is also defined in the branding. Likewise the branding can specify to have unactivated accounts defined after a certain period from creation time.
The password field is optional and requires a specific privilege on the system.

User profiles vs logins

Please note that a "Create" will always cause a login to be created. If no Branding property is set, a default value will be assigned. If a userprofile without an accompanying login is desired the endpoint "reserve" should be used.

Creating with password

If a user is created with password login can be performed using that password. But please be aware that the account is not considered activated until the user clicks the received activation link. In most cases login is not allowed for an unactivated account. This is a decision of the receiver though.
Please note that the password given must conform to the password requirements of the system.

Reserving a user identifier

If a user profile without a login is desired the "reserve" endpoint can be used. This endpoint will always return a user identifier on either a new user or an existing account that matches the input parameters.

This method takes a dictionary as input. This dictionary can be empty or contain the property "Username". If the input dictionary is empty a new user profile will be created for each invokation. If the Username is given the ID of any user profile with the same username will be returned and a new profile is only created if no other profiles matches the Username.

Endpoint	/reserve/ https://userservice.jppol.dk/SSOUser.svc/user/b1fcbf2e-0109-4b1c-8955-da30a0
HTTP method	POST
Request header	HTTP/1.1 AuthorizationDate: Thu, 13 Mar 2014 09:54:44 GMT Authorization: MIDS 39a2fc1e-9761-45fa-a96c-8d85d013f2d2 TB4pMjktKFNj7qpZGJnVSB2l05g= If-Match: 635303011306457500 Host: userservice.jppol.dk Content-Length: 310

Request body	**Userservice POST request** `{"Username": "john@doe.com"}`
Response header	HTTP/1.1 200 OK Server: Microsoft-IIS/7.5 X-Powered-By: ASP.NET Date: Thu, 13 Mar 2014 09:54:44 GMT Content-Length: 0
Response body	**reserve response** `{ "Identifier": "49341154-539f-4d56-bf60-0998b3fa3aa8", "IsNewUser": true, "HasLogin": false, "Version": 0, "State": "Reserved" }`

The reserve endpoint will always return an identifier and could be compared to a lookup-and-create-non-existing endpoint.

The response from the "reserve" endpoint contains the identifier of the user profile that was just created or which already matched the input parameters. Additionally a set of meta data is returned allowing to make logic branching on whether a matching user profile already existed etc. "IsNewUser" will only be true for the invokation where a new user profile was created. "Version" contains the Version property of the user, and HasLogin is true when State is either "LoginCreated" or "Activated". (Please see user profile states for details.)

To make user profiles transition from "Reserved" states the update method should be used.

Update

All fields writeable to the application can be set during updates. The update must always contain a value in the If-Match http header. This is used to make a simple guarantee that client applications do not enter into a race condition. Since the value must match the users current state as obtained by the retrieving the user and read the Version property. Users always have the version 0 immediately following a create.

Endpoint	/user/{userid} https://userservice.jppol.dk/SSOUser.svc/user/b1fcbf2e-0109-4b1c-8955-da30a0

HTTP method	PUT
Request header	PUT https://userservice.jppol.dk/SSOUser.svc/user/b1fcbf2e-0109-4b1c-8955-da30a0 HTTP/1.1 AuthorizationDate: Thu, 13 Mar 2014 10:03:09 GMT Authorization: MIDS 39a2fc1e-9761-45fa-a96c-8d85d013f2d2 QJ1NqGMCS1+DljU9soYTZsNQSvw= If-Match: 0 Host: userservice.jppol.dk Content-Length: 322 Expect: 100-continue

Request body	**Userservice PUT request**
	
	`{"Identifier": "b1fcbf2e-0109-4b1c-8955-da30a09b1d49",`
	`"Properties":{`
	`"Branding":"POL2.prod",`
	`"Username": "faester+example@gmail.com",`
	`"FirstName": "Morten",`
	`"Birthdate":"1978-10-05T12:00:00.000Z",`
	`"SubscriptionID_POL": [ "57505299","111222999" ],`
	`"LastName": null`
	`}`
	`}`

header	HTTP/1.1 200 OK Server: Microsoft-IIS/7.5 X-Powered-By: ASP.NET Date: Thu, 13 Mar 2014 09:54:44 GMT Content-Length: 0
Response body	**Userservice PUT response** `{"UpdateResult":635303017900900449}`

The "UpdateResult" in the response contains the new "Version" to be specified in the next update.

> **Empty updates**
> In case an update does not alter any fields, the update will be accepted, but no internal messages will be displayed. That is, if a user has "FirstName" set to "John", and the update does not alter the value (or any other field), the update will be accepted, but the

response will be contain the original "Version" value and no notifications will be distributed internally in Medielogin; effectively the operation will be a null-operation, but status will be 200 Ok. Clients can detect that an update is considered a null-update by comparing the Version value sent in the If-Match header and the Version from UpdateResult. Or they should know before sending the update...

Update can be used to transition profiles from "Reserved" to "LoginCreated" states. This is done by ensuring that both "Username" and "Branding" is set. It is not possible to modify the password using update; this can be accomplished with the password endpoint.

Delete

User deletion is usually not permitted to applications. A delete will cause the user to be immediately removed from the database.
Hence it is also possible to request a delete, which will cause the user to be put in a pending state with deletion following after 30 days. Listening applications are notified about the delete request and can choose to contact the user in the mean time. If the user attempts to login during this period the user is prompted if the pending delete should be cancelled and login is only allowed when delete is cancelled.
The actual delete request types are discerned from each other using the query string parameter "action".
If no "action" is specified, a delete is requested. If "action=complete" the user is deleted immediately. If "action=cancel" any pending delete request is removed.
The example below has an empty "action" query string parameter and will cause a delete request to be scheduled.

Endpoint	/user/{userid} https://userservice.jppol.dk/SSOUser.svc/user/b1fcbf2e-0109-4b1c-8955-da30a0
HTTP method	DELETE
Request header	DELETE https://userservice.jppol.dk/SSOUser.svc/user/b1fcbf2e-0109-4b1c-8955-da30a0 HTTP/1.1 AuthorizationDate: Thu, 13 Mar 2014 10:03:09 GMT Authorization: MIDS 39a2fc1e-9761-45fa-a96c-8d85d013f2d2 QJ1NqGMCS1+DljU9soYTZsNQSvw= If-Match: 0 Host: userservice.jppol.dk Content-Length: 322
Request body	*<empty>*

Response header	HTTP/1.1 200 OK Content-Length: 2 Content-Type: application/json; charset=utf-8 Server: Microsoft-IIS/7.5 X-Powered-By: ASP.NET Date: Thu, 13 Mar 2014 10:07:43 GMT
Response body	{}

The time where a requested delete is performed can be determined using the retrieved users "PendingDelete" property.

Emailing user

Users can be emailed by sending a reset password link or by starting a change username flow. Additionally a custom mail can be sent. All mails are based on templates in the branding.

Emails are sent asynchronously but usually within 5 seconds from the time the service request completes.

Starting username changes

Endpoint	/user/{userid} https://userservice.jppol.dk/SSOUser.svc/newemail/b1fcbf2e-0109-4b1c-8955-da
HTTP method	PUT
Request header	AuthorizationDate: Thu, 13 Mar 2014 10:03:09 GMT Authorization: MIDS 39a2fc1e-9761-45fa-a96c-8d85d013f2d2 QJ1NqGMCS1+D If-Match: 0 Host: userservice.jppol.dk Content-Length: 322
Request body	*<empty>*

Response header	HTTP/1.1 200 OK Content-Length: 2 Content-Type: application/json; charset=utf-8 Server: Microsoft-IIS/7.5 X-Powered-By: ASP.NET Date: Thu, 13 Mar 2014 10:07:43 GMT
Response body	*<empty>*

An email is sent to the requested-email@example.com based on the email template described in the branding "POL2.prod".
The user must know the old credentials to be able to complete the username change.

Resetting password

Password reset is simply a question og obtaining a link with a token that allows password reset for a specific user. The web service request will sent the user an email with such a link using the template in the specified branding.

Endpoint	/user/{userid} https://userservice.jppol.dk/SSOUser.svc/pwdreset/faester@gmail.com?brandinç
HTTP method	PUT
Request header	AuthorizationDate: Thu, 13 Mar 2014 10:03:09 GMT Authorization: MIDS 39a2fc1e-9761-45fa-a96c-8d85d013f2d2 QJ1NqGMCS1+DljU9soYTZsNQSvw= If-Match: 0 Host: userservice.jppol.dk Content-Length: 322
Request body	*<empty>*
Response header	HTTP/1.1 200 OK Content-Length: 2 Content-Type: application/json; charset=utf-8 Server: Microsoft-IIS/7.5 X-Powered-By: ASP.NET Date: Thu, 13 Mar 2014 10:07:43 GMT
Response body	*<empty>*

Password resets will fail if user is in state "Reserved".

Custom e-mail

A single custom email can be included in a branding. The mail can be sent using the

Endpoint	/mail/{userid}?branding={brandingidentifier} https://userservice.jppol.dk/SSOUser.svc/mail/ b1fcbf2e-0109-4b1c-8955-da30a09b1d49?branding=POL2.prod
Endpoint specifying mail number	/mail/{userid}/{mailnumber}?branding={brandingidentifier} https://userservice.jppol.dk/SSOUser.svc/mail/ b1fcbf2e-0109-4b1c-8955-da30a09b1d49?branding=POL2.prod
HTTP method	POST
Request header	AuthorizationDate: Thu, 13 Mar 2014 10:03:09 GMT Authorization: MIDS 39a2fc1e-9761-45fa-a96c-8d85d013f2d2 QJ1NqGMCS1+DljU9soYTZsNQSvw= If-Match: 0 Host: userservice.jppol.dk Content-Length: 322
Request body	*<empty>*
Response header	HTTP/1.1 200 OK Content-Length: 2 Content-Type: application/json; charset=utf-8 Server: Microsoft-IIS/7.5 X-Powered-By: ASP.NET Date: Thu, 13 Mar 2014 10:07:43 GMT
Response body	*<empty>*

A branding can contain several custom e-mails identified by a number. Can be given as a url component. If unspecified the mail number will default to one.

No email can be set if user is in state "Reserved".

E-mail validation

This endpoint can be used for validating endpoints. Validation includes pattern validation, validation of blacklisted domains and a lookup of an MX record corresponding to the domain

given.
A user cannot be created with an email that does not validate against this method. Existing users can have Usernames that no longer validates. This can happen if an MX record is no longer associated with a domain or if a domain has been intentionally blacklisted.
We blacklist certain domains if they are known to be used for anonymized email services.

Endpoint	/validateEmail/email%40domain.com https://userservice.jppol.dk/validateEmail/email%40domain.com
HTTP method	POST
Request header	Accept:application/json+jsondate Accept-Encoding:gzip, deflate Accept-Language:en-US,en;q=0.8,sv;q=0.6 Authorization:MIDS d0be05e3-937a-4d04-9aeb-df94a0841616 uz09QFLjMp6vCfz8/dO8GWrSuUg=
Request body	*<empty>*
Response header	HTTP/1.1 200 OK Content-Type:application/json; charset=utf-8 Date:Thu, 13 Oct 2016 14:04:13 GMT
Response body	**Email validation response** `{ "validatedEmail": "faester@get2net.com", "isValidEmail": false, "validationResultCode": "NoMxRecordValidationError", "validationResultMessage": "Could not find an MX record for the given domain" }`

The response repeats the input email in "validatedEmail". "isValidEmail" contains true or false corresponding to the emails validity.

The validationResultCode can be used to determine why an email is rejected. The possible values are "NoMxRecordValidationError", "EmailPatternValidationError", "BlacklistedEmailDomainValidationError" and "Ok".

Please be aware that the RFC's defining valid e-mails are in many circumstances more permissive than the Userservice. An e-mail like john.doe@localhost is rejected due to the missing top level domain. Formally legal e-mails like {*}"fobar@thisisjustforfun.com" @bar.dom* (mailto:) **are also rejected although they are considered valid from an**

RFC-point-of-view.
And of course domains without an MX-record would disallow bye-bye-blackbird@get2net.com which is a good thing as the user intended to write bye-bye-blackbird@get2net.dk

User lookup

The userservice provides a simple user id lookup based on e-mail address.

Endpoint	/lookup/{e-mail} https://userservice.jppol.dk/SSOUser.svc/lookup/faester%40gmail.com
HTTP method	GET
Request header	Accept: application/json AuthorizationDate: Mon, 17 Mar 2014 08:32:14 GMT Authorization: MIDS 39a2fc1e-9761-45fa-a96c-8d85d013f2d2 k4ZEZJ9hbZk0TWHDEtb7Gece0TI= If-None-Match: 0 If-Match: 0 Host: userservice.jppol.dk
Request body	*<empty>*
Response header	HTTP/1.1 200 OK Content-Length: 72 Content-Type: application/json; charset=utf-8 Server: Microsoft-IIS/7.5 X-Powered-By: ASP.NET Date: Mon, 17 Mar 2014 08:32:14 GMT Connection: close
Response body	**Userservice lookup response** `{ "Identifier": "cfcc8101-2207-4512-a364-aabd71233cc1", "IsActivated": false, "IsLockedOut": true, "State": "Activated" }`

The "Identifier" is of course the id of the user that can be used to retrieve or update information in the userservice.
The "IsActivated" property indicates if the user has followed the activation link and chosen a password.
If "IsLockedOut" is *true* the account is currently locked out due to too many wrong password attempts. The account can be unlocked using regular password reset by the user.

Credential verification

Although the method is discouraged and OpenID generally should be used to validate a users credentials, it is possible to validate a username/pwd pair in the user service. (This is generally not allowed by any applications.)

Endpoint	/authenticate/{username} https://userservice.jppol.dk/SSOUser.svc/authenticate/{username}
HTTP method	PUT
Request header	Content-Type: application/json AuthorizationDate: Mon, 17 Mar 2014 08:42:06 GMT Authorization: MIDS 39a2fc1e-9761-45fa-a96c-8d85d013f2d2 48zYIMZPcuB2Reb8O3gae+1/8Nc= Host: userservice.jppol.dk Connection: Keep-Alive
Request body	`{ "password": "somepassword123" }`
Response header	HTTP/1.1 200 OK Content-Length: 137 Content-Type: application/json; charset=utf-8 Server: Microsoft-IIS/7.5 X-Powered-By: ASP.NET Accept: application/json+jsondate Date: Mon, 17 Mar 2014 08:42:06 GMT

Response body	**Userservice password validation response**
	
	{
	"IsAuthorized": true,
	"IsActivated": true,
	"Identifier": "cf67ceb3-5e90-47ac-88af-e09fc09c70e3",
	
	"UserCreated": "2015-12-02T09:28:36.452Z",
	"IsLockedOut": false,
	"FoundMatchingUser": true,
	"State": "Activated"
	}

If the response "IsAuthorized" is true, the username/password combination is valid and the "Identifier" will contain the corresponding users id. The user should only be considered Authorized when this value is set to true. True when valid uid/pwd is given and account his not locked out.
"IsActivated" returns true if the user has followed the activation link and confirmed his e-mail. Users can be Authorized even when not activated at the discretion of the client.
"UserCreated" is the user creation time. It is only returned when IsAuthorized is true.
"IsLockedOut" indicates whether the user has had too many subsequent wrong passwords attempts in either the service method or through the user interface in the OpenID provider. Users that has "IsLockedOut" *true*, must not be considered authenticated and the identifier will be empty. The user can unlock the account by resetting her password.
FoundMatchingUser: True if email was matched not considering other parameters.
In other words the user id is only given for a valid password, but the other meta information is returned when the e-mail is known. There is intentionally no password validation status in the response as I would rather not verify it if login is prohibited due to locked accounts etc. (Password hashing is expensive, so doing it for locked out accounts is not desirable.)

"State" will report the status of the found underlying user if any.

Deprecated version of authenticate endpoint

This endpoint is still active for backwards compatibility, but shoule be closed as soon as possible. The password in the query string makes logging etc hard internally and does generally impose a security risk as password can be sniffed easier from a query string than the request body.

Endpoint	/authenticate/?uid={e-mail}&pwd={password} https://userservice.jppol.dk/SSOUser.svc/authenticate/?uid=faester%40gmail.co
HTTP method	GET

Request header	Content-Type: application/json AuthorizationDate: Mon, 17 Mar 2014 08:42:06 GMT Authorization: MIDS 39a2fc1e-9761-45fa-a96c-8d85d013f2d2 48zYIMZPcuB2Reb8O3gae+1/8Nc= Host: userservice.jppol.dk Connection: Keep-Alive
Request body	*<empty>*
Response header	HTTP/1.1 200 OK Content-Length: 137 Content-Type: application/json; charset=utf-8 Server: Microsoft-IIS/7.5 X-Powered-By: ASP.NET Accept: application/json+jsondate Date: Mon, 17 Mar 2014 08:42:06 GMT
Response body	**Userservice password validation response** `{ "IsAuthorized": true, "IsActivated": true, "Identifier": "cf67ceb3-5e90-47ac-88af-e09fc09c70e3", "UserCreated": "2015-12-02T09:28:36.452Z", "IsLockedOut": false, "FoundMatchingUser": true, "State": "Activated" }`

Accessing changelog

The change log system is a means of obtaining the ids and change operations that has happened on users since a logical time stamp. This makes it possible for listeners to know which users has been changed by other applications, has requested to be deleted, has been deleted, created etc.
See the document "UserService change log.pdf" for more information.
Request/response flow is

Endpoint	/changes/{maxKnownChangeNumber}/ https://userservice.jppol.dk/ssouser.svc/changes/2048/

HTTP method	GET
Request header	Accept: application/json AuthorizationDate: Mon, 17 Mar 2014 08:48:23 GMT Authorization: MIDS 39a2fc1e-9761-45fa-a96c-8d85d013f2d2 TsLQJSljCFoOSj1pEqpMVBz7XLY= If-None-Match: 0 If-Match: 0 Host: userservice.jppol.dk Connection: Close
Request body	<*empty*>
Response header	HTTP/1.1 200 OK Content-Length: 26115 Content-Type: application/octet-stream Server: Microsoft-IIS/7.5 X-Powered-By: ASP.NET Date: Mon, 17 Mar 2014 08:48:22 GMT Connection: close

Response body (Truncated)	**Userservice changelog response**
	[
	{
	"OperationNumber": 2049,
	"UserIdentifier":
	"b54aef87-d66c-8071-3561-0a0c678bd467",
	"Operation": "Create",
	"EventTime": "2016-06-23T08:58:30.822"
	},
	{
	"OperationNumber": 2050,
	"UserIdentifier":
	"9779689b-1ccf-777f-e314-f2af28b99b34",
	"Operation": "Update",
	"EventTime": "2016-06-23T10:46:53.238"
	},
	<…>
	{
	"OperationNumber": 2304,
	"UserIdentifier":
	"4b9cd00a-ca34-2db8-ba21-76b9fe4551c2",
	"Operation": "Delete",
	"EventTime": "2016-06-23T10:47:41.059"
	},
	{
	"OperationNumber": 2304,
	"UserIdentifier":
	"4b9cd00a-ca34-2db8-ba21-76b9fe4551c2",
	"Operation": "Create",
	"EventTime": "2016-06-23T11:20:59.366"
	}
	]

When requesting the change log a maximum of 256 operations are returned. The operation number given is the exclusive lower bound of the change array.
Each operation describes an action that has happened on the user at the logical time specified. (Operation 2307 has happened *after* operations 2001 and 2306 and *before* 2308 and 123987. The event times returned will be in UTC time.)
If less than 256 operations are returned the current last change has been seen.
Basically the change log can be used to deduce which users exists at a given logical time and check if the state of any known user has changed since last time the change log was consulted.
The operation types are "Create", "Update", "Delete", "RequestDelete" and "CancelDelete".

There is a simple iterator for the change log written in node.js available at https://github.com/faester/changelogiterator - The iterator needs userservice credentials setup at auth.jppol.dk

Querying users

The user service can answer queries about user states in a specific query syntax. The result set will contain a number indicating the number of matching users, and a list of ids of matching users. The list of user ids will be paged, so at most 128 ids are returned at a time. The response will contain the total number of pages as well as the current zero-based page number.
The query expression is sent as a json encoded string in the request body. The example below retrieves ids of users with LastName equal to "Engdal".

Endpoint	/query/?page=0 https://userservice.jppol.dk/ssouser.svc/query/?page=0
HTTP method	PUT
Request header	Accept:*/* Authorization: MIDS d0be05e3-937a-4d04-9aeb-df94a0841616 PUvd9gvcbFPfK970Ec7hIe4zTdQ= AuthorizationDate: Thu, 07 Aug 2014 13:13:08 GMT Content-Length:24 Content-Type: application/json Host:userservicetest.jppol.dk
Request body	**Userservice query request** `"LastName == \"Engdal\""`
Response header	HTTP/1.1 200 OK Content-Length: 26115 Content-Type: application/octet-stream Server: Microsoft-IIS/7.5 X-Powered-By: ASP.NET Date: Mon, 17 Mar 2014 08:48:22 GMT Connection: close

Response body	**userservice query response**
	
	{
	"MatchCount": 2,
	"IdsOfMatchingUsers": [
	"5580a0b2-9ce5-5878-9064-577d90bb56a2",
	"8ef04e70-abd0-6b33-b3fa-9066f71db0b5"
	],
	"Page": 0,
	"NumberOfPages": 1
	}

The response indicates that two matching users has been found in the user population. The ids of these two users are contained within IdsOfMatchingUsers properties.
The page number is 0, and since the total NumberOfPages is 1, no further pages can be retrieved.
See appendix "User query syntax" for an in-depth specification of the query syntax.

Accessing user history

It is possible to see the different states a user has had over time. The returned result will contain a property state snapshot for the user create and each update . The properties will be limited to the values readable to the requesting application. Hence two state snapshot may be identical as the modifications can be in properties hidden for the application or the update request can have set properties to their previous values.

Endpoint	/history/{userid} https://userservice.jppol.dk/ssouser.svc/history/63ef4bc7-fc6d-4f5b-a912-20469
HTTP method	GET
Request header	Accept:***/*** Authorization:MIDS d0be05e3-937a-4d04-9aeb-df94a0841616 ET0ydVwGmgDkAPtJv/GKXNRSjiI= AuthorizationDate:Thu, 07 Aug 2014 13:25:21 GMT Content-Type:application/json Host:userservicetest.jppol.dk
Request body	1. *Empty*

Response header	Content-Length:35792 Content-Type:application/octet-stream Date:Thu, 07 Aug 2014 13:25:21 GMT Server:Microsoft-IIS/7.5

Response body (truncated)	**Userservice user history response**
	{
	"UserStates": [
	{
	"Version": 0,
	"Properties": {
	"ActivationCode": "D1aeXtX0TZoj",
	"Branding": "medielogin.standard",
	"FirstName": null,
	"LastName": null,
	"PhonePrimary": null,
	"telmoreVasToken": null,
	"Username": "faester@gmail.com"
	}
	},
	{
	"Version": 635315177227736600,
	"Properties": {
	"ActivationCode": "0lZEKGJ5RHsQ",
	"Branding": "medielogin.standard",
	"FirstName": null,
	"LastName": null,
	"PhonePrimary": null,
	"telmoreVasToken": null,
	"Username": "faester@gmail.com"
	}
	},
	{
	"Version": 635315178750567300,
	"Properties": {
	"ActivationCode": "fXebJmKag8l3",
	"Branding": "medielogin.standard",
	"FirstName": null,
	"LastName": null,
	"PhonePrimary": null,
	"telmoreVasToken": null,
	"Username": "faester@gmail.com"
	}
	},
	{
	"Version": 635368788380969100,
	"Properties": {
	"ActivationCode": null,
	"Branding": "medielogin.standard",
	"FirstName": "Morten",
	"LastName": "Fæster",

```
 "PhonePrimary": null,
 "telmoreVasToken":
"1C57557A26502BC9F08ED2467B19EB3BAB500C8CC54845003F640451E3C8D
 "Username": "faester@gmail.com"
 }
 }
```

```
  ],
  "UserId": "63ef4bc7-fc6d-4f5b-a912-204695c25b9b"
}
```

User state

The user profiles contained in user service can be in three states: "Reserved", "LoginCreated" and "Activated"

Reserved user profiles

Reserved user profiles indicates that an identifier has been stored, but no login has been created. The identifier can be completely anonymous or it can contain any number of the known properties in Userservice. A reserved user may have a Username associated, but there will not be a login connected to the profile. If the user attempts to create a profile through self service on either medielodin.dk or polid.jppol.dk, the experience will be identical to the experience of a completely unknown e-mail address. Trying to reset password or change username will also be impossible as no login exists.

Reserved user profiles are intended for scenarios where the person behind the information does not necessarily want a login but where we need to store information about the person either as a customer of print news papers, as participant in competition, for marketing activity, for newsletter registration etc. It is of course the responsibility of the clients to ensure that the no data registration happens without proper consent any time.

As long as a user is in Reserved state no emails or other communication will happen from Medielogin.

Login created

A reserved user profile can transition into a login by updating the user properties. A login is created when both the "Username" and the "Branding" property is set. If users are created using the "Create" method a login will always be created. If no "Branding" is set when invoking "Create" a default value will be assigned.

When a user profile is in state "Login created" the user cannot register a new account on either polid.jppol.dk or medielogin.dk and the user can reset password and Medielogin can sent email to the user.

At the time of setting both "Branding" and "Username" a welcome email will be sent to the user. If either of the values are later removed and even later restored to non-null values the system will not sent welcome messages.

Although the login has been created the user will not be able to perform logins at this point, as the ownership of the account has not been established.

Activated

User profiles become activated when a value in the property "Activated" has been set. This

happens when the user has received a welcome email and confirmed ownership of the account by clicking the activation link.

When a user is Activated the profile is considered a full login.

Pending delete

If a user requests profile deletion a value will be set in the PendingDelete property of the User object. While user is in pending delete state login is disabled. Instead the user is prompted if the pending delete should be cancelled.

Test client

The user service exposes a test client if you go to https://userser.jppol.dk/ . (This is of course also available in the test environment at https://userservicetest.jppol.dk).
The test client is intended to be a developer tool, where a developer can get familiarized with the functionality and get a reference for the request signing prior to making real integrations against the system.
As the screenshow from the UI below shows, all method calls are group in boxes that can be expanded and shrunk by clicking on the headers.
First step should always be setting the application id and application key in the "Setup" box. (The fields "Protocol" and "Host" are predefined with the correct values and should not be touched.)
All requests and responses are executed using JavaScript and can be examined using the browsers regular developer tools. The signing relevant parameters from the request are copied to the "Request structure" box in the UI.
Results from the service are displayed with a generic wrapping of the properties in ul/li-tags with correct nesting.

Error responses

All errors are communicated by a http status code as well as a json object describing the error.
The error object contains the error code, an error message and an exception name.

```
{
"Code": 412,
"exceptionName": "UserNotInSyncException",
"message": "Exception of type 'UserService.Exceptions.UserNotInSyncException' was thrown."
}
```

The possible error codes are

Http status code	Description	Exception name
400	The value of a property is not allowed. (This can be caused by violation of the properties data type or using a single value where an array is required or vice versa)	InvalidPropertyValueException
400	The request is ill-formed.	FormatException
400	A query expression was not allowed.	InvalidQueryExpressionException
400	An attempt to update an undefined property was made.	UnknownPropertyException
401	No authorization header in the request.	InvalidAuthorizationHeaderException
401	Ill-formed authorization header.	AuthorizationHeaderFormatException
401	The authorization header contains illegal values or is not signed with a correct signature for the application.	InvalidAuthorizationHeaderException
401	The request time is invalid.	InvalidRequestTimeException

403	The calling application is not allowed to set the value of a specific property.	InvalidPropertyModificationException
404	Unknown user id.	UserNotFoundException
405	The application with specified id is not allowed to perform the requested operation.	InvalidApplicationAccessException
406	The user identifier in the request can not be parsed as a valid GUID.	InvalidUserIdentifierException
406	Different user ids in the request body and the request URI.	UserIdsMismatchException
409	The Username is already in use.	DuplicateUsernameException
409	The user id matches a known user.	DuplicateUserException
412	No If-Match header in request.	InvalidIfMatchHeaderException
412	No If-None-Match header in request.	InvalidIfNoneMatchHeaderException
412	User has been altered by another applicaiton. (The Version in the If-Match header is not the current verison in the database.)	UserNotInSyncException
415	Content-Type or Accept was not recognized. Use "application/json" (for legacy MS-style datetimes) or "application/json+jsondate" (for more current datetime strings).	UnknownMediaTypeInAcceptHeaderException
415	Invalid e-mail address in the request.	CouldNotParseMailIdentifierException
500	General error.	Exception

Invalid properties and error reports

POST and PUT operations may attempt to set property to illegal values. In this case the response will inform about the first illegal property.
The format is

```
{
"Code": 400,
"errorProperty": "a",
"exceptionName": "InvalidPropertyValueException",
"message": "Invalid property value in property \"a\"",
"validationStatus": "DatatypeValidationError"
}
```

The code matches the http status code. "errorProperty" contains the name of the first illegal property. The message is an informal description of the error. "validationStatus" is an enum describing the error in details:
RangeValidationError
The property value is outside the range for the specified value.
RegexValidationError
The property does not conform to the pattern for the property.
DatatypeValidationError
An invalid data type has been given. (Ex: Strings were an int is expected or arrays for single valued properties.)
NoMxRecordValidationError
There was no valid mx record for the domain of an email.
EmailPatternValidationError
An email did not follow pattern matching rules for e-mail addresses.
BlacklistedEmailDomainValidationError
The domain of the email was blacklisted.
ValueIsNull
Null was given to a mandatory property.
EmailHasInvalidTopLevelDomain
The top level domain of the e-mail was not allowed.
InvalidPasswordValue
When an attempt to create a user fails because the password given does not conform to the complexity requirements of the passwords.

Appendix A - Request signing

This section is an excerpt from the "Politiken mobile infrastructure documentation". It is included here for convenience.

HTTP Request format

The authorization of all request are bas ed on the raw HTTP request. Notably both the "Date", "Authorization" and "Host" headers must be set in order to be allowed to invoke the service.

Date header

The "Date" header should contain a date time string representation of the request time. Valid date string representations are those directly translateable by .NET's System.DateTime.TryParse(string dateString) method. Eligible date formats are among others ISO8601. The date should be the current time when performing the request - if the date time differs significantly from the time on the server the request will be rejected. This is in order to avoid reuse of old requests.

Authorization header

The authorization header consists of the following elements:

Authorization: AuthorizationValue

where AuthorizationValue is

MIDS \[Application ID\] \[Encrypted string\]

where

MIDS is a constant describing that the following header is a Politiken formatted security string.

Application ID contains an application id delivered by JP/Politikens Hus.

Encrypted string contains a hashed and encrypted version of the remaining request parameters.

Construction of signed string

The encrypted string is a concatenation of the HTTP request metod/HTTP verb, the sorted parameters and the Date string. Each element is separated with a newline.
The string obtained from the operation is HMAC-SHA1 encrypted with the applications secret key and Base64-encoded.
One small example: An application named "test" wants to perform a request

```
http://userservice.local/ssouser.svc/authenticate/?uid=faester@gmail.com&pwd=0263Ff
```

This translates into the following string to sign:

```
GET
userservice.local
/ssouser.svc/authenticate/
pwd=0263Ff
uid=faester@gmail.com
Thu, 31 Jul 2014 07:19:05 GMT
```

Each line should be separated with a newline, which - in this context - means the characters "\r\n".

The application ID corresponds to the secret key "privatekey".
The string to sign is encrypted using the secret key and a corresponding encrypted string is created.

```
JF4iRvFVr9R3cSI01g3JlhQMco4=
```

When the values are used the following http request is created:

```
GET
http://userservice.local/ssouser.svc/authenticate/?uid=faester@gmail.com&pwd=0263Ff
HTTP/1.1
Host: userservice.local
Connection: keep-alive
Date: Thu, 31 Jul 2014 07:19:05 GMT
Accept: */*
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/36.0.1985.125 Safari/537.36
Authorization: MIDS d0be05e3-937a-4d04-9aeb-df94a0841616
JF4iRvFVr9R3cSI01g3JlhQMco4=
Content-Type: application/json
Referer:
http://userservice.local/client.html


Accept-Encoding: gzip,deflate,sdch
Accept-Language: da,en;q=0.8,en-US;q=0.6,nb;q=0.4
```

Alternative date header

In some frameworks (amongst others WCF) it is not possible to control the date header. In this case the date header can be replaced with a non-standard http header named AuthorizationDate. The request would then become:

```
GET
http://userservice.local/ssouser.svc/authenticate/?uid=faester@gmail.com&pwd=0263Ff

HTTP/1.1
Host: userservice.local
Connection: keep-alive
AuthorizationDate: Thu, 31 Jul 2014 07:19:05 GMT
Date: Thu, 31 Jul 2014 07:26:03 GMT
Accept: */*
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/36.0.1985.125 Safari/537.36
Authorization: MIDS d0be05e3-937a-4d04-9aeb-df94a0841616
JF4iRvFVr9R3cSI01g3JlhQMco4=
Content-Type: application/json
Referer:
http://userservice.local/client.html


Accept-Encoding: gzip,deflate,sdch
Accept-Language: da,en;q=0.8,en-US;q=0.6,nb;q=0.4
```

If the standard Date header is present it will take precedence over the AuthorizationDate and should be used for constructing the encrypted string.

User query syntax

The query syntax is a simple expression syntax, that allows for value comparison, lookup of values in arrays, testing for isnull and some capabilities to look at current and past values of properties.
The main comparisons are the well-known operators from say JavaScript or C#:

Comparison: "==", "!=","<", ">"

It is also possible to test if a property is either of an array of values using the "in" operator:

Any of the values: Firstname in [“Jens”, “Peter”]

String properties can also be queried using MS SQL-based "like" syntax:

FirstName like "M_rt_n"
Username like "%@mailinator%"

Logical operators

Binary: "&&" and "||"

Unary (negation): "!"

Constant values are expressed as

Strings "constant string value"
Datetimes: 2014/02/28, 2014/02/24-12:32:12, 2014/02/24-12:32:12.1312
Integers: 9312
Floating point values use dots as decimal separator: 1.2231

Properties are referenced through their case-sensitive names.
And of course expressions can be grouped using parenthesis.
To test if a property contains a null, use the syntax "isnull(PropertyName)".

Test for null: "isnull(Firstname)"

Creation time

To query user creation time the function "created()" is used

Email domain

To query domains of the username function "usernameDomain()" is used

Finally it is possible to define if the current, previous or any property value should be used. (If no time operator is specified only the current value will be considered.)

Current value of property: current(Propertyname) Previous value of property: previous(Propertyname) Any value for property: any(Propertyname)

Given these primitives it is possible to construct query expressions like

All users with first name ("FirstName" equal to "Jens").	FirstName == "Jens" && Birthdate < 1945/04/05
Firstname "Jens" and Birthdate before April 5th 1945	FirstName == "Jens" && Birthdate < 1945/04/05
Firstname is either “Jens” or “Peter” (with inclause)	FirstName in ["Jens", "Peter"]
All users with last name “Jensen” where Birthdate is either null or before April 5th 1945	(isnull(Birthdate) || Birthdate < 1945/04/05) && LastName == "Jensen"
Users that previously had a value in the field accessToken but where the value is currently null:	isnull(accessToken) && !isnull(previous(accessToken))
Previous first name was "Jens" but current is "Wilhelmine"	previous(FirstName) == "Jens" && current(FirstName) == "Wilhelmine"
The property “accessToken” currently has a value or has had a value in the past. This will be changed since it does in reality match the query *!(isnull(any(accessToken)) && isnull(current(accessToken)))* using OR between the two statements *!isnull(any(acce ssToken))* But the consistent interpretation should be *!(isnull(any(accessToken))* || *isnull(current(accessToken)))* Please use these more verbose but easier to understand statements instead.	!isnull(any(accessToken))
User is created after March 10 2014 with a current value in accessToken	created() > 2014/03/10 && !isnull(accessToken)

Arrays

Property containing arrays are queried like single-value properties. The operators applied will match any value within the array. So if we have a user U1 with the property SubscriptionID containing the values \["1", "2"\] and another user U2 with SubcriptionID \["2", "3"\] the

expression "SubscriptionID == "2"" will match both users. "SubscriptionID != "1"" will only match user 2. If you want to know if a specific array position contains a value, the zero-based array index can be appended to the property reference with a dot separator. Thus the expression "SubscriptionID.0 == "2"" will match U2 only, since U1 has "1" in the first array position.