tu Latch

API Documentation

CONTENT

1. API Documentation

1.1. Authentication

This page describes how authentication works in the Latch service.

All requests to the Latch API must be signed. The signing process is a simplified two-way Oauth protocol.

Each HTTP request to the API must be accompanied by two authentication headers: Authorization and Date.

1.1.1. The Authorization Header

The **Authorization** header must have the following format:

```
11PATHS requestId requestSignature
```

- **requestId** can be either an applicationId or a userId, depending on which API is being accessed by the request.
- **11PATHS** is a constant that determines the authentication method.
- **applicationId** is an alphanumeric identifier obtained when registering a new application.
- **requestSignature** is a signature derived from the URL, parameters, custom headers, and date of the current request, all hashed using a **Secret** that is also obtained through the applicationId when registering the application.

The request signature is a Base64-encoded, HMAC-SHA1 signed string.

The string must be created by following this process:

1. Start with an empty string.
2. Append the method name in uppercase. Currently, only GET, POST, PUT, and DELETE values are supported.
3. Append a line separator, "\n" (Unicode code point U+000A).
4. Append a string with the current date in the exact format yyyy-MM-dd HH:mm:ss. This format should be self-explanatory, but note that everything is numeric and should be zero-padded as needed to ensure

all numbers have two digits (except for the year, which has four digits). You must retain this value to use it in the **Date** header. The signature check will fail if they do not match.

5. Append a line separator, "\n" (Unicode code point U+000A).

6. Serialize all headers specific to this application (not every HTTP header of the request). All names of these headers start with X-11paths-.
   - Convert all header names to lowercase.
   - Sort the headers by header name in ascending alphabetical order.
   - For each header value, convert multi-line headers into a single line by replacing newline characters "\n" with single spaces " ".
   - Create an empty string. Then, for each header after sorting and transforming, add to the newly created string the header name followed by a colon ":" and then the header value. Each name:value must be separated from the next by a single space " ".
   - Trim the string to ensure it does not contain any leading or trailing spaces.

7. Append a line separator, "\n" (Unicode code point U+000A).

8. Append the URL-encoded query string, which consists of the path (starting from the first slash) and the query parameters. The query string must not include the hostname or the port, and it must not contain any leading or trailing spaces.

9. **Only for POST or PUT requests**: append a line separator, "\n" (Unicode code point U+000A).

10. **Only for POST or PUT requests**: serialize the request parameters as follows. The parameter name and its value are the UTF-8 representation of their URL encoding.
    - Sort the parameters by parameter name in ascending alphabetical order and then by parameter value.
    - Create an empty string. Then, for each parameter after sorting, add to the newly created string the parameter name followed by an equals sign "=" and the parameter value. Each name=value pair must be separated from the next by an ampersand "&".
    - Trim the string to ensure it does not contain any leading or trailing spaces.

Once the string has been created following the process described above, it must be signed with the HMAC-SHA1 algorithm using the **Secret** obtained when registering the application. After signing, the raw binary data must be Base64-encoded. The resulting string is the `requestSignature` that should be added to the **Authorization** header.

1.1.2. The X-11Paths-Date Header

The **X-11Paths-Date** header contains the current UTC date value and must have the following format:

yyyy-MM-dd HH:mm:ss

Where yyyy is the year, MM is the month number, dd is the day number, HH is the hour in 24-hour format, mm is the minute within the hour, and ss is the second within the minute. All values must be zero-padded so that they are all two-digit values, except for the year, which is four digits.All values must be zero-padded to ensure two digits, except for the year, which has four digits.

It is very important that the value and format of this header are exactly the same as those used in the process of creating requestSignature for the **Authorization** header, as explained above.

Note that you can continue to use the standard HTTP **Date** header in any format you wish (for example, RFC 1123). Just make sure you do not mix them up and always use the value in **X-11Paths-Date** in the signing process. The API will ignore the standard **Date** header.

Errors

Below is a list of possible error conditions that can occur during authentication:

- **Error 101:** Invalid Authorization header format
- **Error 102:** Invalid application signatura
- **Error 103:** Authorization header missing
- **Error 104:** Date header missing
- **Error 108:** Invalid date format

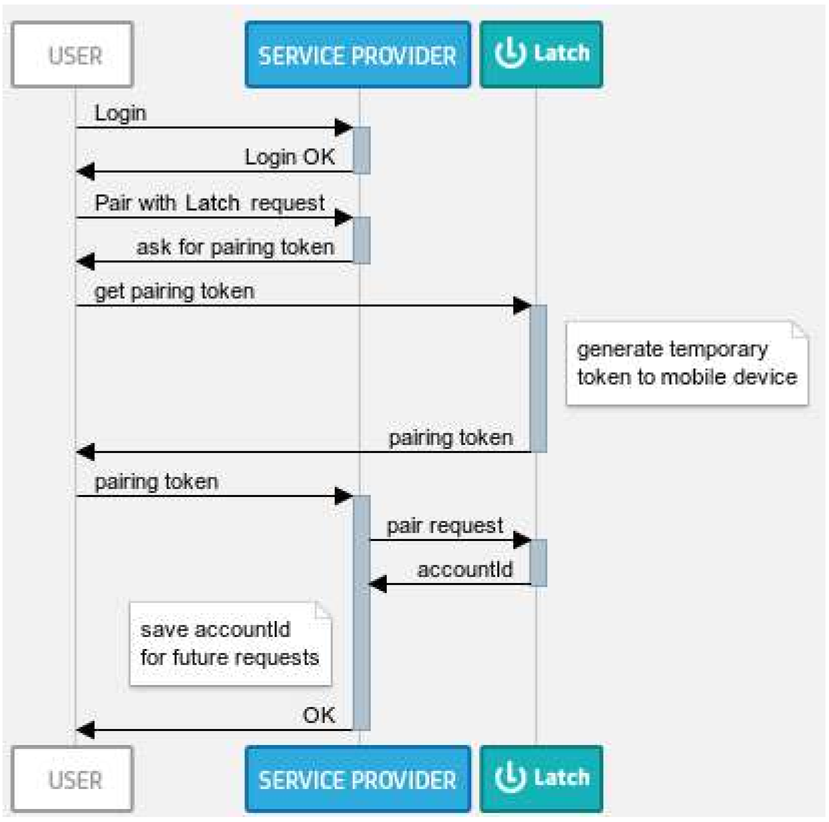
- **Error 109:** Request expired, date is too old
- **Error 111:** User not authorized
- **Error 112:** Invalid user signatura
- **Error 113:** Secret signing this request is not authorized to perform this operation

1.2. Aplication API

En esta sección se detalla la información relacionada con la interacción de la aplicación con la API de Latch.

1.2.1. Pair Account

This page describes the preferred method to pair accounts using a pairing token that users can obtain through the mobile apps.



Authentication

To use this request, you must authenticate your app using your applicationId and sign it as explained in the Authentication section.

Request

To pair with the token API, make an HTTP GET request to the following endpoint:

```
https://latch.tu.com/api/2.0/pair/{token}
```

where token is a six-character alphanumeric token that should be obtained from the user, who in turn retrieved it through the mobile app.

Additionally, and completely optionally, you can include a parameter commonName where you include an identifier of the user in your system. This identifier will be shown in the support tool (LST) and help you identify the user more easily.

```
https://latch.tu.com/api/2.0/pair/{token}?commonName=YOUR_COMMON
_NAME
```

Response

If everything goes well, the API will return a response like this:

```
{"data":{"accountId":"..."}}
```

where account is a 64-character alphanumeric token that uniquely identifies the account being paired with this application. You should store this value for later use in subsequent calls such as **status** or **unpair**.

Errors

Below is a list of possible error conditions that can occur when attempting to use this API:

- **Error 205: Account and application already paired**
  This error will be returned when a user whose account is already paired with this application has generated the specified token.

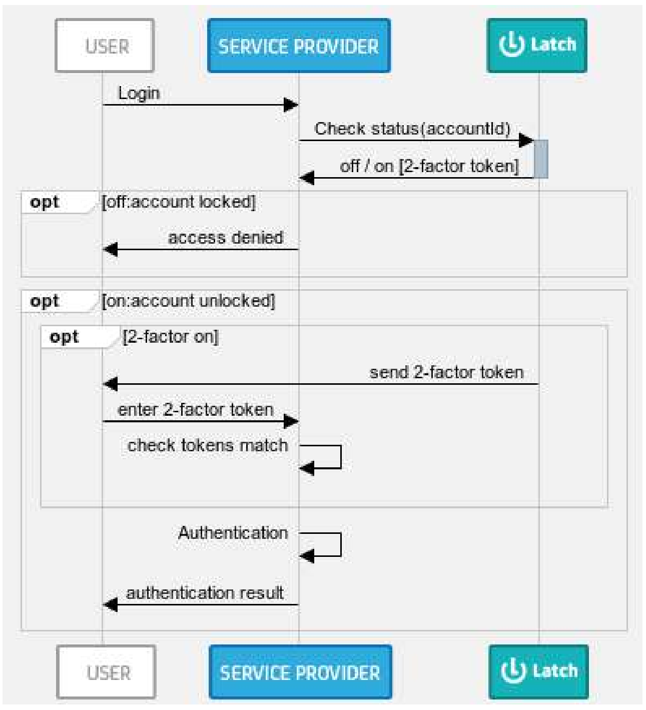
- **Error 206: Pairing token not found or expired**
  The most likely cause of this error is the use of an expired token. Tokens must be used to perform the pairing within 60 seconds of their creation; after that period, they expire. Another obvious reason for this error is that the request includes a fabricated token.

- **Error 401: Missing parameter in API call**
  The cause of this error is that a required parameter is missing or empty.

- **Error 406: Invalid parameter length**
  This error will be returned when the commonName parameter has a length greater than 100 characters.

1.2.2. Check Status

This page describes how to check the status of an account and the different possible responses.



Authentication

To use this request, you must authenticate your app using your applicationId and sign it as explained in the Authentication section.

Request

To use the status API, make an HTTP GET request to the following endpoint:

```
https://latch.tu.com/api/2.0/status/{accountId}
```

where accountId is a 64-character alphanumeric token that uniquely identifies the account whose status you are checking, obtained using the pairing method.

If you have created one or more operations to be controlled by individual Latches, you can check the status of each operation using its operation identifier:

```
https://latch.tu.com/api/2.0/status/{accountId}/op/{operationId}
```

IMPORTANT: If you have created operations for an application, you should NOT need to check the status using the application ID. The same is true if you have nested operations. Once an operation has child operations, the application ID of the parent operation should not be used for status calls.
You can do it, and it will work, but in the mobile apps, applications with operations or operations with children don't have the options of One-Time-Passwords, Scheduler setup and Autolock. Instead, they act as a master switch for all children operations, allowing to lock/unlock them all at once

Optionally, if you want to check the status:

- Without including any one-time password in the response, it is possible by adding the `/nootp` suffix to any of the two endpoints.
- Without sending push notifications to mobile apps, it is possible by adding the `/silent` suffix to any of the two endpoints.

```
https://latch.tu.com/api/2.0/status/{accountId}/nootp

https://latch.tu.com/api/2.0/status/{accountId}/silent

https://latch.tu.com/api/2.0/status/{accountId}/nootp/silent
```

```
https://latch.tu.com/api/2.0/status/{accountId}/op/{operationId}
/nootp/silent
```

Response

If everything goes well, the API will return a JSON response that looks like this:

```
{"data":{"operations":{"applicationId":{"status":"...",
"two_factor":{...}, "operations":{...}}}}}
```

There are two possible values for the status field: on and off.

This two_factor field is optional and when present, it has the following format:

```
"two_factor":{"token":"...","generated":...}
```

where token is a six-character alphanumeric value that is also sent to the user's mobile device and must be checked to complete the operation. The generated field is the timestamp at which the token was generated, and it can be used to control the time window during which the token can be validated.

The operations object, if present, contains the child operations and is recursively defined by the same fields: status, two_factor and operations.

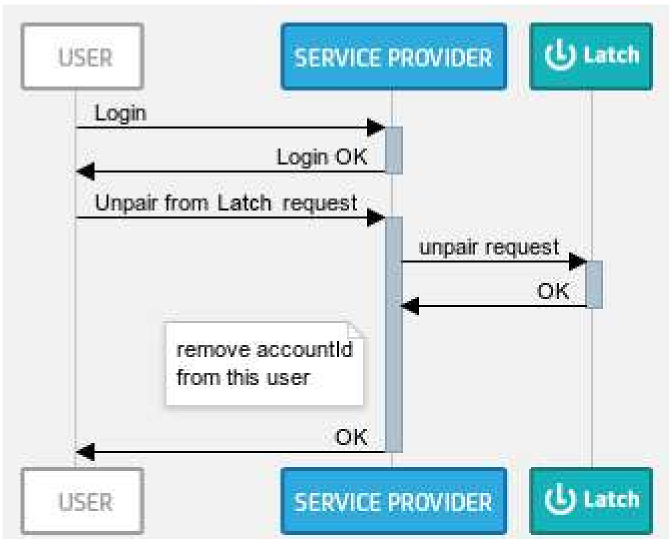
Errors

The following is a list of possible error conditions that can occur when trying to use this API:

- **Error 201:** Account not paired
  This error occurs when the account identifier is incorrect or is not currently paired with the application requesting the status check.
- **Error 401:** Missing parameter in API call
- **Error 704:** Silent notification is not applied due to subscription limits

1.2.3. Unpair Account

This page describes how to unpair a Latch account.



Authentication

To use this request, you must authenticate your app using your applicationId and sign it as explained in the Authentication section.

Request

To unpair, make an HTTP GET request to the following endpoint:

```
https://latch.tu.com/api/2.0/unpair/{accountId}
```

where accountId is a 64-character alphanumeric token that uniquely identifies the account to be unpaired, which was obtained through the pairing method.

Response

If everything goes well, the API will return an empty JSON response that looks like this:

```
{}
```

Errors

Below is a list of possible error conditions that may occur when attempting to use this API:

- **Error 201:** Account not paired (API > 0.6)
- **Error 204:** Error unpairing account (API <= 0.6)
  This error occurs when the account identifier is incorrect or is not currently paired with the application requesting the unpair.
- Error 401: Missing parameter in API call

1.2.4. Modify Status

This page describes how to modify the status of a Latch account.

Authentication

To use this request, you must authenticate your app using your applicationId and sign it as explained in the Authentication section.

Request

To use the external modifications API, make an HTTP POST request to the following endpoints depending on whether you want to lock or unlock the latch for an application or operation:

```
https://latch.tu.com/api/2.0/lock/{accountId}
https://latch.tu.com/api/2.0/unlock/{accountId}
```

where `accountId` is a 64-character alphanumeric token that uniquely identifies the account whose status is being modified, obtained using the pairing method.

If you have created one or more operations to be controlled by individual Latches, you can modify the status of each operation using its operation identifier:

```
https://latch.tu.com/api/2.0/lock/{accountId}/op/{operationId}
https://latch.tu.com/api/2.0/unlock/{accountId}/op/{operationId}
```

Response

If everything goes well, the API will return an empty JSON response that looks like this:

```
{}
```

Errors

Below is a list of possible error conditions that may occur when attempting to use this API:

- **Error 111:** User not authorized
  This error occurs when the current subscription plan of the user does not include the support tools.
- **Error 201:** Account not paired
  This error occurs when the account identifier is incorrect or is not currently paired with the application requesting the status check.
- **Error 401:** Missing parameter in API call

1.2.5. Manage Operations

This page describes how to manage the operations of an application, allowing you to add, update, or delete operations.

Authentication

To use this request, you must authenticate your app using your applicationId and sign it as explained in the Authentication section.

Requests

Add an operation

To add an operation, make an HTTP PUT request to the following endpoint:

`https://latch.tu.com/api/2.0/operation`

Request parameters:

- parentId: The identifier of the application or operation to which we want to add the new operation to be created.
- name: A new name for the operation.
- two_factor (optional): Value for two-factor authentication.
  Possible values: MANDATORY, OPT_IN, DISABLED. Default value is DISABLED.
- lock_on_request (optional): Value for locking after querying.
  Possible values: MANDATORY, OPT_IN, DISABLED. Default value is DISABLED.

Response

If everything goes well, the API will return a JSON response that looks like this, where `operationId` is the identifier of the newly created operation:

```
{data:{"operationId":"..."}}
```

Modify an operation

To modify an operation, make an HTTP POST request to the following endpoint:

`https://latch.tu.com/api/2.0/operation/{operationId}`

'OperationId' will be the identifier of such operation.

Request parameters:

- name: A new name for the operation.

- two_factor (optional): Value for two-factor authentication.
  Possible values: MANDATORY, OPT_IN, DISABLED. Default value is DISABLED.
- lock_on_request (optional): Value for locking after querying.
  Possible values: MANDATORY, OPT_IN, DISABLED. Default value is DISABLED.

Response

If everything goes well, the API will return an empty JSON response that looks like this:

```
{}
```

Remove an operation

To remove an operation, make an HTTP DELETE request to the following endpoint:
https://latch.tu.com/api/2.0/operation/{operationId}

where operationId is the identifier of that operation.

Response

If everything goes well, the API will return an empty JSON response that looks like this:

```
{}
```

Get operations

To get your existing operations, perform an HTTP GET request in the following endpoint:

```
https://latch.tu.com/api/2.0/operation
https://latch.tu.com/api/2.0/operation/{operationId}
```

Response

If everything goes well, the API will return a response that looks like this:

```
{"data":{"operations":{"operationId":{"name":"...","two_factor":
"...","lock_on_request":"...","operations":{...}}}}}
```

Errors

Below is a list of possible error conditions that can occur when attempting to use this API:

- **Error 301:** Application or Operation not found
  This error occurs when the operation identifier used does not match a valid operation identifier.
- **Error 401:** Missing parameter in API call
- **Error 703:** Application or Operation not created due to subscription limits
  This error occurs when trying to create an operation outside the subscription limits.

1.2.6. User History

This page describes how to query the history of an account.

Authentication

To use this request, you must authenticate your app using your applicationId and sign it as explained in the Authentication section.

Request

To use the history API, make an HTTP GET request to the following endpoint:

```
https://latch.tu.com/api/2.0/history/{accountId}/{from}/{to}
```

where `accountId` is a 64-character alphanumeric token that uniquely identifies the account whose history is being queried, obtained using the pairing method.

The parameters `from` and `to`, which are optional, allow filtering the history between two dates. Both parameters are numeric and correspond to the number of milliseconds elapsed since January 1, 1970, 00:00:00 GMT. (Epoch Time).

Response

If everything goes well, the API will return a JSON response that looks like this:

```
{ "data": { "applicationId": { ... }, "count": ...,
"clientVersion": { ... } }, "lastSeen": ..., "history": [ { "t":
..., "action": "...", "what": "...", "value": "...", "was":
"...", "name": "...", "userAgent": "...", "ip": "..." } ] } }
```

where the information contained is:

- **applicationId**: Information about the application and its operations.
- **count**: The number of elements contained in the `history` array.
- **clientVersion**: Information about the mobile clients used by the user.
- **lastSeen**: The time at which the user last activity was recorded.
- **history**: An array containing the actions performed on the account, either by the user or by the developer.
    - **t**: The time at which the action was performed.
    - **action**: Whether it is an action performed by the user (`USER_UPDATE`), by the developer (`DEVELOPER_UPDATE`) or a status query (`get`).
    - **what**: The parameter on which the action was performed (`status`, `two_factor`, `from`, etc.).
    - **was**: The previous value of the action if it was a modification.
    - **value**: The current value of the action.
    - **name**: The name of the application or operation on which the action was performed.
    - **ip / userAgent**: The IP and User-Agent of whoever performed the action.

Errors

Below is a list of possible error conditions that can occur when attempting to use this API:

- **Error 111:** User not authorized
  This error occurs when the current subscription plan of the user does not include support tools.

- **Error 201:** Account not paired
  This error occurs when the account identifier is incorrect or is not currently paired with the application requesting the status check.

- **Error 401:** Missing parameter in API call

- **Error 405:** History response is limited to 1000 entries for the selected date range
  This non-fatal error occurs when the response should contain more items than the maximum that is returned (1000).

1.2.7. Manage instances

This page describes how to manage the instances of an application or operation associated with a specific accountId, as well as how to interact with them.

Authentication

To use this request, you must authenticate your app using your applicationId and sign it as explained in the Authentication section.

Requests

Add an instance

To add an instance to an application, make an HTTP PUT request to the following endpoint:

```
https://latch.tu.com/api/2.0/instance/{accountId}
```

To add instances to an operation, make an HTTP PUT request to the following endpoint:

```
https://latch.tu.com/api/2.0/instance/{accountId}/op/{operationId}
```

where accountId is a 64-character alphanumeric token that uniquely identifies the account whose instances are being modified. This would have been previously obtained using the pairing method.

Request parameters:

- instances: The name of the instances to create in the format instances=name, and you can create several simultaneously separated by &.

Example: `instances=Name_1&instances=Name_2&instances=Name_N`

Response

If everything goes well, the API will return a JSON response that looks like this, where the various `instancesId` are the identifiers of the newly created instances that must be stored to interact with them:

```
{"data":{"instances":{"instanceId_1":"Name_1","instanceId_2":"Name_2","instanceId_N":"Name_N", ...}}}
```

Delete an instance

To delete an instance from an application, make an HTTP DELETE request to the following endpoint:

```
https://latch.tu.com/api/2.0/instance/{accountId}/i/{instanceId}
```

To delete an instance from an operation, make an HTTP DELETE request to the following endpoint:

```
https://latch.tu.com/api/2.0/instance/{accountId}/op/{operationId}/i/{instanceId}
```

where instanceId is the identifier of that instance and operationId is the identifier of the operation to which it belongs.

Response

If everything goes well, the API will return an empty JSON response that looks like this:

```
{}
```

Get instances

To get instances from an application, perform an HTTP GET request in the next endpoint:

```
https://latch.tu.com/api/2.0/instance/{accountId}
```

To get the instances of an operation, make an HTTP GET request to the following endpoint:

```
https://latch.tu.com/api/2.0/instance/{accountId}/op/{operationI
d}
```

where `instanceId` is the identifier of that instance and `operationId` is the identifier of the operation to which it belongs.

Response

If everything goes well, the API will return a response that looks like this:

```
{"data": {"instanceId_1": {"name": "MyInstance1", "two_factor":
"MANDATORY|DISABLED|OPT_IN", "lock_on_request":
"MANDATORY|DISABLED|OPT_IN"}, "instanceId_2": {"name": "...",
"two_factor": "...", "lock_on_request": "..."}}}
```

Edit an instance

To edit an instance of an application, make an HTTP POST request to the following endpoint:

```
https://latch.tu.com/api/2.0/instance/{accountId}/i/{instanceId}
```

To edit an instance of an operation, make an HTTP POST request to the following endpoint:

```
https://latch.tu.com/api/2.0/instance/{accountId}/op/{operationI
d}/i/{instanceId}
```

where `instanceId` is the identifier of that instance and `operationId` is the identifier of the operation to which it belongs.

Request parameters:

- name (Optional): A new name for the instance.
- two_factor (optional): Value for two-factor authentication.
  Possible values: MANDATORY, OPT_IN, DISABLED. Default value is DISABLED.
- lock_on_request (optional): Value for locking after querying.
  Possible values: MANDATORY, OPT_IN, DISABLED. Default value is DISABLED.

Response

If everything goes well, the API will return an empty JSON response that looks like this:

```
{}
```

Check the status of an instance

To check the status of an instance, make an HTTP GET request to one of the following endpoints (for instances of an application or instances of an operation):

```
https://latch.tu.com/api/2.0/status/{accountId}/i/{instanceId}
```

`https://latch.tu.com/api/2.0/status/{accountId}/op/{operationId}/i/{instanceId}`

Both the response and the other additional options such as silent or nootp are detailed in the **Account Status** section.

Modify the status of an instance

Modify the status of an instance, make an HTTP POST request to one of the following endpoints (for instances of an application or instances of an operation):

`https://latch.tu.com/api/2.0/lock/{accountId}/i/{instanceId}`

`https://latch.tu.com/api/2.0/lock/{accountId}/op/{operationId}/i/{instanceId}`

`https://latch.tu.com/api/2.0/unlock/{accountId}/i/{instanceId}`

`https://latch.tu.com/api/2.0/unlock/{accountId}/op/{operationId}/i/{instanceId}`

The 'Modify status' section contains the response and additional information

Errors

Below is a list of possible error conditions that can occur when attempting to use this API:

- **Error 301:** Application or Operation not found
  This error occurs when the operation identifier used does not match a valid operation identifier.
- **Error 302:** Instance not found
  This error occurs when the instance identifier used does not match a valid instance identifier.
- **Error 401:** Missing parameter in API call

- **Error 703:** Application or Operation not created due to subscription limits
  This error occurs when trying to create an instance outside the subscription limits.

1.2.8. Set CommonName

This page describes how to edit a common name for a paired user.

Authentication

To use this request, you must authenticate your app using your applicationId and sign it as explained in the Authentication section.

Request

Set a common name

To set a common name, make an HTTP POST request to the following endpoint:

```
https://latch.tu.com/api/3.0/commonName/{accountId}/
```

Request parameters:

- commonName: The service provider identifier for the paired user

Response

If everything goes well, the API will return an empty JSON response that looks like this:

```
{}
```

Errors

Below is a list of possible error conditions that can occur when attempting to use this API:

- **Error 201:** Account not paired
  This error occurs when the account identifier is incorrect or is not currently paired with the application requesting the status check.

- **Error 401:** Missing parameter in API call
  The cause of this error is that a required parameter is missing or empty.

- **Error 406:** Invalid parameter length
  This error will be returned when the `commonName` parameter has a length greater than 100 characters.

1.3. User API

1.3.1. Manage Applications

This page describes how to manage a user's applications, allowing you to add, update, or delete applications.

Authentication

To use this request, you must authenticate your app using your userId and sign it as explained in the Authentication section.

Requests

Add an application

To add an application, make an HTTP PUT request to the following endpoint:

`https://latch.tu.com/api/2.0/application`

Request parameters:

- `name`: The new name for the application.
- `contactEmail`: Contact email for the application.
- `contactPhone`: Contact phone number for the application.
- `two_factor` (Optional): Value for two-factor authentication. Possible values: `MANDATORY`, `OPT_IN`, `DISABLED`. Default value is `DISABLED`.
- `lock_on_request` (Optional): Value for locking after querying. Possible values: `MANDATORY`, `OPT_IN`, `DISABLED`. Default value is `DISABLED`.

Response

If everything goes well, the API will return a JSON response that looks like this, where applicationId is the identifier of the newly created application:

```
{data:{"applicationId":"...", "secret":"..."}}
```

Modify an application

To modify an application, make an HTTP POST request to the following endpoint:

```
https://latch.tu.com/api/2.0/application/{applicationId}
```

where applicationId is the identifier of that application.

Request parameters:

- name: The new name for the application.
- contactEmail: Contact email for the application.
- contactPhone: Contact phone number for the application.

- two_factor (Optional): Value for two-factor authentication.
  Possible values: MANDATORY, OPT_IN, DISABLED. Default value is DISABLED.

- lock_on_request (Optional): Value for locking after querying.
  Possible values: MANDATORY, OPT_IN, DISABLED. Default value is DISABLED.

Response

If everything goes well, the API will return an empty JSON response that looks like this:

```
{}
```

Remove an application

To delete an application, make an HTTP DELETE request to the following endpoint:

```
https://latch.tu.com/api/2.0/application/{applicationId}
```

where `applicationId` is the identifier of that application.

Response

If everything goes well, the API will return an empty JSON response that looks like this:

```
{}
```

View your applications

To view your applications, make an HTTP GET request to the following endpoint:

```
https://latch.tu.com/api/2.0/application
```

Response

If everything goes well, the API will return a response that looks like this:

```
{"data":{"operations":{"applicationId":{"name":"...","two_factor
":"...","lock_on_request":"...","operations":{...}}}}}
```

Errors

Below is a list of possible error conditions that can occur when attempting to use this API:

- **Error 111:** User not authorized
  This error occurs when the current subscription plan of the user does not include the user API.
- **Error 301:** Application or Operation not found
  This error occurs when the application identifier used does not match a valid application identifier.
- **Error 401:** Missing parameter in API call
- **Error 402:** Invalid parameter value
  This error occurs when some parameters have an incorrect format, such as email addresses, phone numbers, or names.
- **Error 703:** Application or Operation not created due to subscription limits
  This error occurs when trying to create an application outside the subscription limits.

1.3.2. Developer Subscription

This page describes how to obtain the subscription information of a user.

Authentication

To use this request, you must authenticate your app using your userId and sign it as explained in the Authentication section.

Requests

View subscription

To obtain the subscription information, make an HTTP GET request to the following endpoint:

```
https://latch.tu.com/api/2.0/subscription
```

Response

If everything goes well, the API will return a JSON response that looks like this:

```
{"data":{"subscription":{"id":"subcriptionName","applications":{
"inUse":X,"limit":X},"operations":{"appName1":{"inUse":X,"limit"
:X},"users":{"inUse":X,"limit":X}}}}
```

where the information contained is:

- **subscriptionName**: The name of the current subscription.
- **inUse**: The number of applications, users, or operations currently in use.
- **limit**: The maximum number allowed for the current subscription plan. `-1` for unlimited.

Errors

Below is a list of possible error conditions that may occur when attempting to use this API:

- **Error 111: User not authorized**
  This error occurs when the current subscription plan of the user does not include the user API.

1.4. TOTP Server

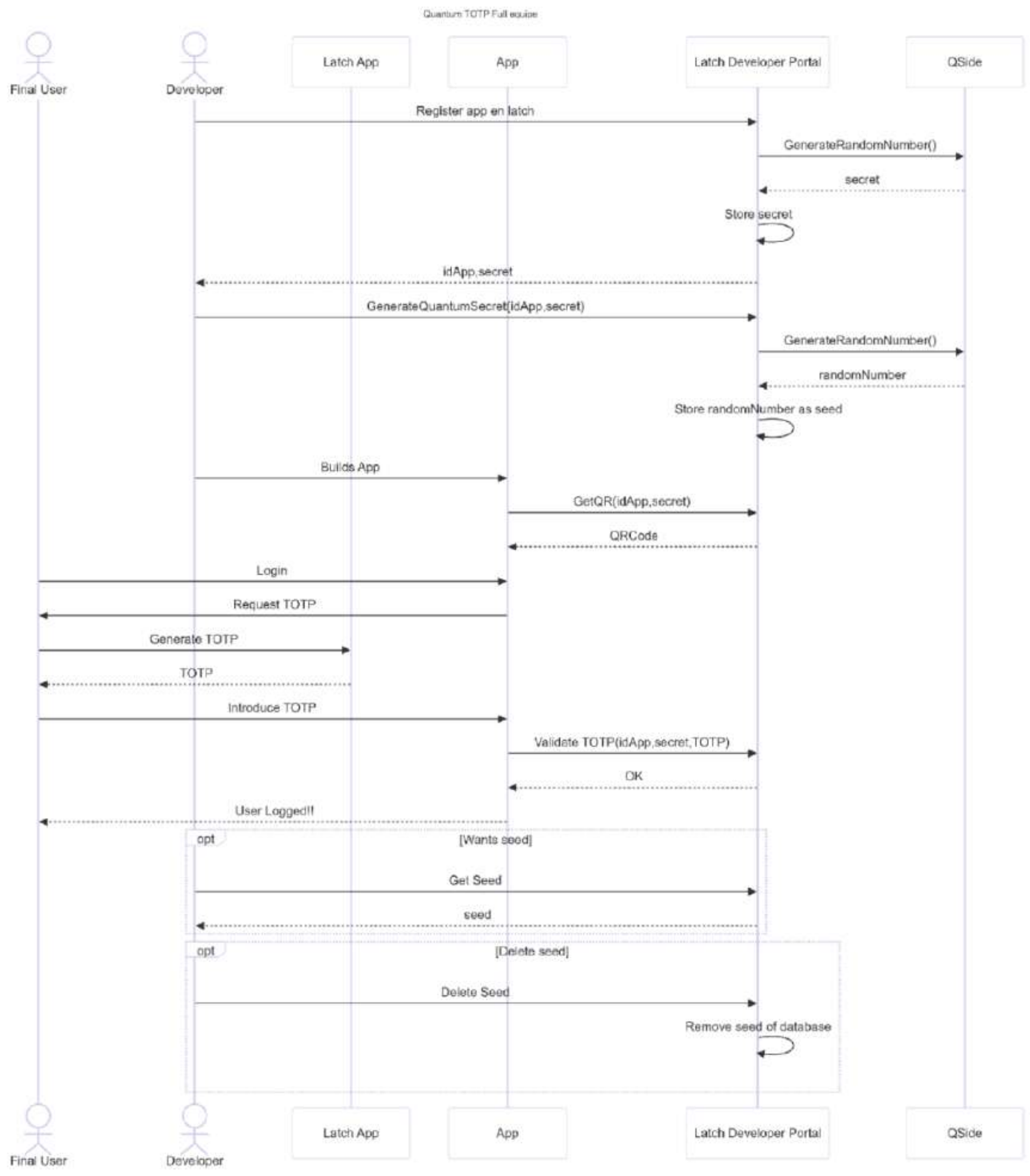
1.4.1. Create TOTP

This service acts as a TOTP (Time-based One-Time Password) server that enables you to create, manage, and validate time-based one-time passwords.

It provides an API to create and manage these TOTPs and secrets associated with different users. The main functionalities include:

- **Create a TOTP**: Allows users to create a TOTP for authentication.
- **Delete a TOTP**: TOTPs created can be deleted when no longer needed.
- **Get TOTP information**: You can query information about a specific TOTP.
- **Validate a TOTP**: Allows validating a TOTP code provided by a user.

The sequence of operations and their use is represented in the following diagram:



Authentication

To use this request, you must authenticate your app using your applicationId and sign it as explained in the Authentication section.

Request

Create a new TOTP for a given user. userId (the user identifier) and commonName (the common name of the TOTP) are required.

```
POST https://latch.tu.com/api/3.0/totps
```

Parameters:

- `userId`: The identifier that will be associated with the user (provider-dependent).
- `commonName`: The name associated with the user (provider-dependent).

Response

If everything goes well, the API will respond with the information related to the TOTP:

```
{
    "data": {
        "totpId": "identificador del TOTP",
        "secret": "secreto del TOTP",
        "appId": "identificador de la app de Latch",
        "identity": {
            "id": "identificador del usuario",
            "name": "nombre del usuario"
        },
        "issuer": "nombre de la app",
        "algorithm": "algoritmo del TOTP",
        "digits": "dígitos del TOTP",
        "period": "período del TOTP",
        "createdAt": "fecha de creación",
        "qr": "imagen del código QR en base64",
        "uri": "URI del TOTP, formato otpauth"
    }
}
```

Errors

Below is a list of possible error conditions that can occur when attempting to use this API:

- **Error 301:** Application or Operation not found
  This error occurs when the application identifier used does not match a valid application identifier.

- **Error 304:** No totp server configured
  No TOTP Server configured for this app.

- **Error 401:** Missing parameter in API call
  Invalid parameters.

- **Error 307:** Unable to generate totp
  Could not generate the secret for the TOTP. Internal error.

1.4.2. Delete TOTP

Authentication

To use this request, you must authenticate your app using your applicationId and sign it as explained in the Authentication section.

Request

Delete a specific TOTP using its identifier totpId.

```
DELETE https://latch.tu.com/api/3.0/totps/{totpId}
```

Parameters:

- totpId: Identifier of the TOTP you want to delete.

Response

If everything goes well, an HTTP RESPONSE 204 will be returned.

Errors

Below is a list of possible error conditions that can occur when attempting to use this API:

- **Error 304: No totp server configured**
  No TOTP Server configured for this app.

- **Error 305: App totp not found**
  TOTP not found.

1.4.3. Get TOTP

Authentication

To use this request, you must authenticate your app using your applicationId and sign it as explained in the Authentication section.

Request

Returns the information of a specific TOTP based on its totpId.

```
GET https://latch.tu.com/api/3.0/totps/{totpId}
```

Parameters:

- totpId: Identifier of the TOTP you want to retrieve.

Response

If everything goes well, the API will respond with the information related to the TOTP:

```
{
  "data": {
    "totpId": "identificador del TOTP",
    "secret": "secreto del TOTP",
    "appId": "identificador de la app de Latch",
    "identity": {
      "id": "identificador del usuario",
      "name": "nombre del usuario"
    },
    "issuer": "nombre de la app",
    "algorithm": "algoritmo del TOTP",
    "digits": "dígitos del TOTP",
    "period": "período del TOTP",
    "createdAt": "fecha de creación",
    "qr": "imagen del código QR en base64",
    "uri": "URI del TOTP, formato otpauth"
  }
}
```

Errors

Below is a list of possible error conditions that can occur when attempting to use this API:

- **Error 304:** No totp server configured
  No TOTP Server configured for this app.
- **Error 305:** App totp not found
  TOTP not found.

1.4.4. Validate TOTP

Authentication

To use this request, you must authenticate your app using your applicationId and sign it as explained in the Authentication section.

Request

Validates a specific TOTP code provided by the user. The totpId identifies the TOTP, and code is the generated code you want to validate.

```
POST https://latch.tu.com/api/3.0/totps/{totpId}/validate
```

Parameters:

- totpId: Identifier of the TOTP used to validate the algorithm against the provided code.
- code: The code provided by the end user, to be validated using the algorithm specified by totpId.

Response

In both cases, an HTTP code of 200 is returned:

Correct code:

```
{}
```

Incorrect Code:

```
{"error": {
 "code": 306,
 "message": "Invalid totp code"
   }
}
```

Errors

Below is a list of possible error conditions that can occur when attempting to use this API:

- **Error 304:** No totp server configured
  No TOTP Server configured for this app.
- **Error 305:** App totp not found
  TOTP not found.
- **Error 306:** Invalid totp code
  The code is invalid according to the stored parameters and the current timestamp.
- **Error 307:** Unable to generate totp
  Error retrieving TOTP generation parameters.
- **Error 401:** Missing parameter in API call
  Some of the input parameters do not exist.
- **Error 402:** Invalid parameter value
  Invalid parameters.
- **Error 708:** TOTP not validated due to subscription limits
  TOTP not valid due to developer subscription limits.

1.5. Authorization Controls

Introduction

The Authorization Controls feature in the Latch product allows you to establish a mechanism for managing the approval of actions within a system, based on the intervention of one or more users. This feature is useful in scenarios where explicit approval from certain users is required before a process or flow can proceed.

Creating an Authorization Control

Authorization controls are created via the tool called LST. From LST, administrators or users with permissions can define a new rule or control that manages approvals for a specific group of users.

Steps to create an authorization control

1. Access the LST tool.
2. Select the option to create a new Authorization Control.
3. Specify the users involved in the control.
4. Define whether the control should be based on:
    - **Unanimous Approval (TODOS)**: The flow or action is only allowed if all selected users approve the request.
    - **Partial Approval (ALGUNO)**: The flow or action is allowed if at least one of the selected users approves the request.
5. Save the authorization control.

Identification of the Authorization Control

Each authorization control created has a unique identifier called controlID. This controlID is essential for interacting with this feature through the system's API, enabling subsequent queries or modifications to that specific control.

Using the API for Authorization Controls

Once an authorization control has been created, it can be managed or queried through the Latch APIs. The controlID will be the main identifier used to perform any operation on that control.

Available operations via API:

- **Consult an authorization control**: Retrieve the details of the control, including the associated users and approval rules (TODOS or ALGUNO).
- **Update an authorization control**: Modify the users or the approval conditions.
- **Delete an authorization control**: Remove an authorization control if it is no longer needed.

Example Use Scenario

Suppose a business flow requires the approval of three users before completing a transaction. You can configure an authorization control in Latch where:

- All three users involved are added to the control.
- It is defined that approval must be unanimous—i.e., all users must approve before the transaction is executed.

In another case, if it is required that only one of the three users is able to approve in order to proceed, you would configure the control with the `ALGUNO` option.

Conclusion

The Authorization Controls feature in Latch offers a flexible and powerful way to manage user approvals, enabling configurations based on both **all** users involved or **any** user from the group. Integration with LST and the availability through an API makes it easy to use both via a graphical interface and through automation in external applications.

Authentication

To use this request, you must authenticate your app using your applicationId and sign it as explained in the Authentication section.

Request

Query the status of a given authorization control.

```
GET https://latch.tu.com/api/3.0/control-status/{controlId}
```

Parameters:

- controlId: Identifier of the authorization control to query.

Response

If everything goes well, the API will respond with information about the status of the authorization control controlId:

```
{
    "data": {
        "status": "on" | "off"
    }
}
```

Errors

Below is a list of possible error conditions that can occur when attempting to use this API:

- **Error 113:** Secret signing this request is not authorized to perform this operation
  The appId/secret signature is not authorized to perform this operation.
- **Error 301:** Application or Operation not found
  This error occurs when the operation identifier used does not match a valid operation identifier.
- **Error 302:** Instance not found
  This error occurs when the instance identifier used does not match a valid instance identifier.
- **Error 1100:** Authorization control not found
  The authorization control identifier was not found.
- **Error 1104:** Error checking authorization control status
  Error querying the authorization control.

2. Webhooks

WebHooks allow developers to receive real-time notifications when their paired users perform actions through the mobile app.

Once a URI is registered to receive WebHooks, Latch will start sending HTTP requests to that URI each time there are changes for any of your users.

2.1. Add a Webhook

To add a new WebHook, go to your application page in **'My Applications'**.

Then, add the full URI of your WebHook without including a query (e.g., https://www.example.com/hook) in the **"WebHooks"** section. Note that this URI needs to be publicly accessible on the Internet; i.e., URIs like 127.0.0.1 or localhost are not valid because Latch will not be able to communicate with your local network.

Once you have entered a URI for a WebHook, a verification request will be sent to that URI. This verification is an HTTP GET request that includes a challenge parameter. Your WebHook must echo this parameter in order to be verified. This verification is performed to ensure that your application indeed wants to receive notifications at that URI. If you accidentally entered an incorrect URI (or if someone maliciously added your server as a WebHook), your application will not respond correctly to the challenge, and Latch will not send notifications to that URI.

Example:

```
GET https://www.example.com/hook?challenge=ABC123
```

If your WebHook successfully responds to the above challenge, Latch will start sending notifications to the WebHook URI with changes for your users (see the **WebHook Notifications** tab for more information). Otherwise, you will see an error message indicating the problem encountered.

NOTE: Your URI only has ten seconds to

2.2. Webhook Notifications

When a WebHook URI is added, you start receiving **notification requests** each time one of your users makes a change in your application.

A notification request is an HTTP POST request that includes JSON in its body. Note that multiple simultaneous changes for multiple users can be included in the same notification. You can find the specific JSON format for each type of notification below.

Notification Signature

Each notification request will include an HTTP header X-11paths-Authorization consisting of the HMAC-SHA1 signature of the request body (the application's **secret** is used as the signing key). This allows your application to verify that the notification comes from Latch. It is advisable to check the signature validity for each notification received.

Lock / unlock of a latch

This notification is sent each time a user modifies the latch of your application or any of its internal operations.

```
{
  "t":1457913600,
  "accounts":{
    "ACCOUNT_ID1":[
                 {
                  "type":"UPDATE",
                  "id":"APP_OR_OPERATION_ID | GLOBAL_LATCH",
                  "source":"USER_UPDATE | DEVELOPER_UPDATE",
                  "new_status":"on | off | interval"
                 },
                 { ... }
```

```
      ],
      "ACCOUNT_ID2":[
        ...
      ]
    }
  }
```

Information:

- **t**: The UTC timestamp when Latch sends this notification.
- **ACCOUNT_ID**: The accountId of your users, obtained during the pairing phase.
- **id**: The identifier of the application or operation of the latch that the user just modified. If the user modifies the master latch of all applications, the value will be GLOBAL_LATCH.
- **source**: If the change was made by the user (USER_UPDATE) or by the developer using the support API or the LST tool (DEVELOPER_UPDATE).
- **new_status**: The new latch status: on for unlocked, off for locked, or interval if the user has activated the "scheduled lock" option for that latch.

3. Limited Secrets

Each Latch application provides an **application identifier** (appId) and a **secret**. This pair of credentials allows requests to the API to be signed to ensure they are made by the legitimate owner of that application. However, there are multiple scenarios in which the secret can be compromised (desktop applications, mobile apps, etc.).

In these scenarios, a compromised secret could allow a potential attacker to perform unwanted API operations. To avoid this, you can create up to **3 limited**

secrets for each application, each with restricted scope. These secrets are used in the same way as the master secret, but can only be used to sign those requests for which they have been configured.

A **limited secret** can contain one or more of the following permissions:

- **Status**: Allows making status calls to the API, to check the status of a Latch.
- **Pairings**: Allows using the calls to pair and unpair a user.
- **Support**: Allows using the support calls of the API to lock and unlock latches.
- **History**: Allows querying the history of actions for a service.
- **Operations**: Grants access to perform the calls that manage operations. This management allows creating, editing, querying, and deleting operations.
- **Instances**: Grants access to perform the calls that manage instances. This management allows creating, editing, querying, and deleting instances. To check the status of an instance, the STATUS permission is used, and to modify its status, the SUPPORT permission is required.
- **TOTP Server**: Grants access to perform the calls that manage TOTPs.
- **Number of Quantum Secrets**: Grants access to perform calls that manage quantum secrets.
- **Authorization Controls**: Grants access to perform calls that manage authorization controls.

4. Deep Linking

4.1. What is deep linking?

'Deep linking' is a technique that allows launching native mobile applications through a link.

Deep linking makes it possible to connect a unique URL with a specific action in a mobile application; depending on the mobile platform, the URIs required to launch the application may vary.

These are the functionalities of our applications currently available through deep linking. At the moment, they are only available for Android and iOS:

- **Launch the Latch application.**

4.2. Deep Linking in Latch Applications

You can use any of these URI resources from either a web browser or a native mobile application:

Launch the Latch application

Android

```
latch://launch
```

iOS

```
latch://launch
```