VeriDas

/Multimodal system

/Operation manual

Index

/Introduction

Multimodal system permits authentications with a series of factors:

- **face authentication (1FA)** for **regular** users: in this scenario, user **embedding** (which is a biometric vector obtained from the face characteristics) is stored in the database during the registration process and it is propagated to the terminal. Thus, the user is being authenticated as soon as it approaches the terminal.
- **biometric QR authentication (2FA)** for **occasional** users (such as visitors): in this scenario, user embedding is stored in the QR code shared with the user during the registration process. Thus, the user is being authenticated as soon as the user scans the QR code and the embedding from the QR code coincides with the biometric vector of the detected face.
- **non-biometric QR authentication (1FA)** for **occasional** users (such as visitors): in this scenario, user QR code is shared with the user during the registration process. Thus, the user is being authenticated as soon as the user scans a valid QR code .

/Registration of regular users with face authentication

The Credential Service Provider (from now we will refer to it as **Welcome**) is a web API that provides credential management for physical access control. **Welcome** implements the following main use cases, such as user enrolment, accreditation and distribution or replication.

An individual user registration process requires the following calls:

1. Obtaining an authentication token by following the process outlined in section Authentication.
2. Sending the registry data via a POST to the /singles resource outlined in section POST /singles.

Authentication

Authentication within the Welcome API is based on **OpenID Connect (OIDC)**, which is an authentication protocol implemented on top of the OAuth 2.0 authorization framework.

In the following sections, the authentication process for API calls is specified in more detail with examples and the available resources are also described in a detailed way.

Token generation

Veridas will provide the client with two pieces of information that identify the client and are necessary to obtain the access token: *client_id* and *client_secret*. With these two pieces of information an access token must be obtained by calling the API endpoint /token as follows:

```
POST /auth/realms/{TENANT_ID}/protocol/openid-connect/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Accept: */*
Host: iam.work-srv.das-gate.com
content-length: XX

grant_type=client_credentials&scope=roles&client_id=CLIENT_ID&client_secret=CLIENT_SECRET
```

The obtained response is a json with the following format:

```
{'access_token': 'eyJhbGc....',
 'expires_in': 300,
 'token_type': 'bearer',
 'not-before-policy': 0,
 'session_state': '3f9a7a76-cd4a-4082-b3e0-95686bac24ee',
 'scope': 'email profile roles'
}
```

From this response the *access_token* shall be extracted for use in API requests as follows. The *expires_in* field indicates the validity period of the token in seconds.

Access to protected API resources

The access token can be used to perform the necessary operations against the API. The API will respond with HTTP 403 error code if the authenticated operator does not have privileges for the operation, and HTTP 401 if the access token is expired or the operator is anonymous. The propagation of the access token, so that it can be processed and validated by the API, is done via the http Authorization header and bearer token, as described in RFC 6750, Authorization Request Header Field. Example of HTTP request:

```
GET /resource HTTP/1.1
Host: server.example.com
Authorization: Bearer ACCESS_TOKEN
```

Refreshment of tokens

When the token expires, it is necessary to generate a new one. To do so, it is necessary to carry out the operation indicated in the Token generation section again.

It is also possible to use the **Refresh Token grant**, although it involves sending practically the same as in the section Token generation.

```
POST /auth/realms/{TENANT_ID}/protocol/openid-connect/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Accept: */*
Host: iam.work-srv.das-gate.com
content-length: XX

grant_type=refresh_token&refresh_token=REFRESH_TOKEN&client_id=CLIENT_ID&client_secret=CLIEN
T_SECRET
```

POST /singles

Once the access token is obtained, the POST /singles request should be used to register a single user by sending a selfie and the user identification data. It creates a user with the submitted access control system identifier (from now we will refer to it as **acs_id**).

```
POST /singles HTTP/1.1
Content-Type: application/json
Accept: */*
Host:
Authorization: Bearer ACCESS_TOKEN
x-api-key: APIKEY
x-dasgate-tenant-id: TENANT
content-length: XX

{
    "acs_id_schema": "string64",
    "deliver_by_email": "false",
    "acs_id": "ACS_ID",
    "portrayal_type": "selfie",
    "selfie": "SELFIE_IMAGE_BASE64",
}
```

Request parameters

- POST /singles
- Headers:

    - **x-dasgate-tenant-id** (*string*): Veridas tenant id. Provided for each customer.
    - **x-api-key** (*string*): Veridas api key. Provided for each customer.
- Body scheme :
    - **acs_id_schema** (*string*): This field indicates the format of the acs_id field.
        - Mandatory field
        - Possible values: wiegand26, string64
    - **acs_id** (*string*): This field corresponds to the user identifier in the external system.
        - Mandatory field
        - maxLength: 64
    - **portrayal_type** (*string*): This field indicates the type of registration.
        - Mandatory field
        - Possible values: selfie
    - **selfie** (*string*): User photo encoded in base64 format (RFC2397).
        - Mandatory field
        - Supported image types: image/jpg and image/png.
        - Max size: 4MB
    - **deliver_by_email** (*boolean*): This field indicates if the credential should be delivered by given email
        - Mandatory field
        - Value: False
    - **username** (*string*): This field allows to introduce a name that identifies the user.
        - Optional field
    - **show_username** (*boolean*): This field indicates whether the username field will appear on the terminal screen once the user has been authenticated.
        - Optional field
    - **email** (*string*): This field indicates user email
        - Optional field

Request example

In this example case the objective is to create a new user in the Veridas system using the endpoint POST /singles:

Content of the request:

```
{
    "acs_id_schema": "string64",
    "acs_id": "75588821R",
    "show_username": "true",
    "username": "test_name",
    "portrayal_type": "selfie",
```

```
    "deliver_by_email": false
    "selfie":
"url('
V5Zd2XPu9wCS0vwn0GJNoNfq1LQkVWWSQLi92+Y/YIAHA2Sm9vb//fX/UdnaAp2aTvBolxlOalsgwwS3tkBboC1bW0Cl
aZuVU99MjzmAGE8xg9oW6LKnKZe201lZW....)”
}
```



```
Successful response: 201
```

```
{
    "onboarding_id": "1ed02748-5347-6db8-98a3-0242ac110004",
    "acs_id": "72673821P",
    "status": "finished",
    "result": {
        "outcome": "accepted",
        "reason": null
    },
    "user_id": "cc75f09e-0828-4602-81c4-a84239cd845b",
    "username": null
}
```


This request may not be successful for a variety of reasons:

- token not handled correctly (answer 401 - 403, non authorized)
- invalid photo (insufficient resolution, photo not corresponding to one side)
- system overload due to too many requests per minute
- incorrect acs_id_schema format for tenant used
- existing acs_id entered

It is **important** to consider these situations when implementing the integration, and that negative feedback is correctly fed back to the third system.

/Registration of occasional users with biometric or non biometric QR codes

das-FaceQR is used for QR code generation. **das-FaceQR** is a cloud-based service which allows the authentication of a person via face biometry. As a result, a face biometric credential is provided, which is a mathematical descriptor obtained from the face characteristics. The biometric credential is bundled in a QR code. The official das-FaceQR API documentation can be found in our documentation hub .

API Specification

Although the documentation hub includes multiple forms of obtaining credentials, for the purpose of this integration, the two queries to generate biometric and non-biometric QR codes should be used:

Biometric QR

- POST /dasfaceqr/v2/{hashOrTag}/credential/{codeType}
  - **{hasOrTag}** (string): Model identifier, which could be given either as a tag or a hash. For example: *"20210913"*.
  - **{codeType}** (string): *"qr-code"*.
- Body scheme (both fields are **required**):
  - **selfieImage** (string <*binary*>): A picture with a selfie. The maximum admitted size for the image is 2 MB.
  - **contextualData** (string <*binary*>): This is a field that can include contextual information.
- Parameters:
  - **qrVersion** (integer): QR version to generate
  - **qrErrorCorrection** (string): Error correction level to use, as specified by the QR standard
  - **boxSize** (string): Controls how many pixels each box of the QR code is
  - **binary** (boolean): If set to true or 1, forces the raw QR content to be returned, regardless of what's been said in the Accept header

  Here goes the list of parameters with the values that must be applied to permit a sufficient size margin for contextual data:

Parameter	Value
qrVersion	28
qrErrorCorrection	M
boxSize	4
binary	false

Non-biometric QR

- POST /dasfaceqr/v2/credential/{codeType}
  - **{codeType}** (string): *"qr-code"*.
- Body scheme (the field below is **required**):
  - **contextualData** (string <*binary*>): This is a field that can include some contextual information.
- Parameters:
  - **qrVersion** (integer): QR version to generate

- **qrErrorCorrection** (string): Error correction level to use, as specified by the QR standard
- **boxSize** (string): Controls how many pixels each box of the QR code is
- **binary** (boolean): If set to true or 1, forces the raw QR content to be returned, regardless of what's been said in the Accept header

Here goes the list of parameters with the values that must be applied to permit a sufficient size margin for contextual data:

Parameter	Value
qrVersion	18
qrErrorCorrection	M
boxSize	4
binary	false

POSTMAN for testing environment

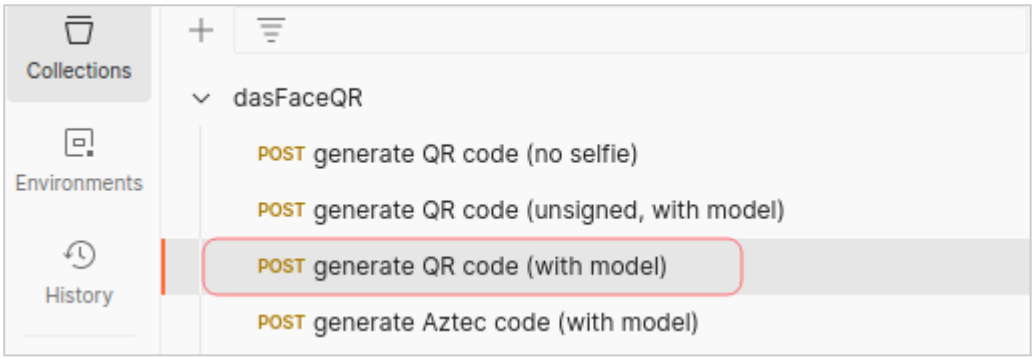
Postman is a popular API testing tool and can be used to verify a provided call to das-FaceQR endpoint. The QR codes can be generated via Postman application. In order to do so, the following steps should be applied:

- Use dasFaceQR.postman_collection
  - Download this compressed file, and extract the collection file named dasFaceQR.postman_collection.json.
  - Open Postman application.
  - Click on Import, and select the dasFaceQR.postman_collection.json file.
- Set environment variables. For instance, variables of the WORK environment have been configured:
  - apikey (in the final integration, this field is specific per each customer)
  - protocol: https
  - url:

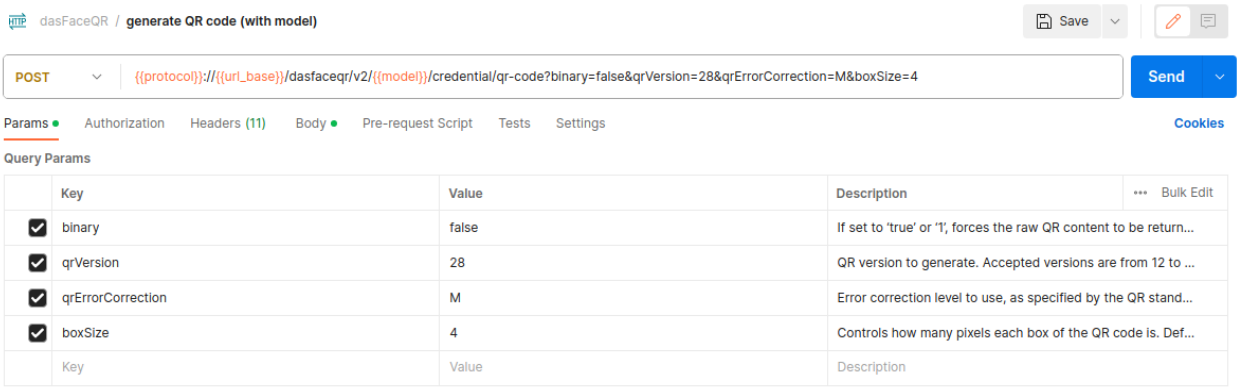
WORK	api-work.eu.veri-das.com
LIVE	api.eu.veri-das.com

  - model: 20210913 (can be configured depending on the current model development stage)

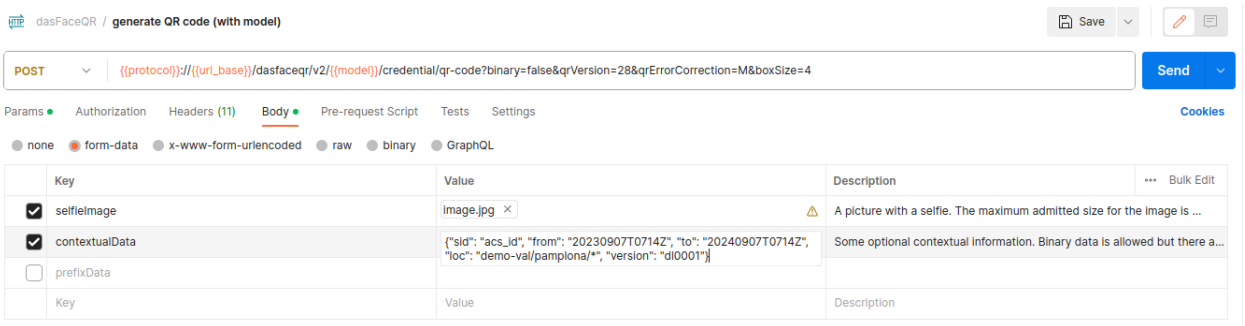
1. Select from Collections the option to generate QR code (with model)



2. Set query parameters according to QR code type.



3. Set body

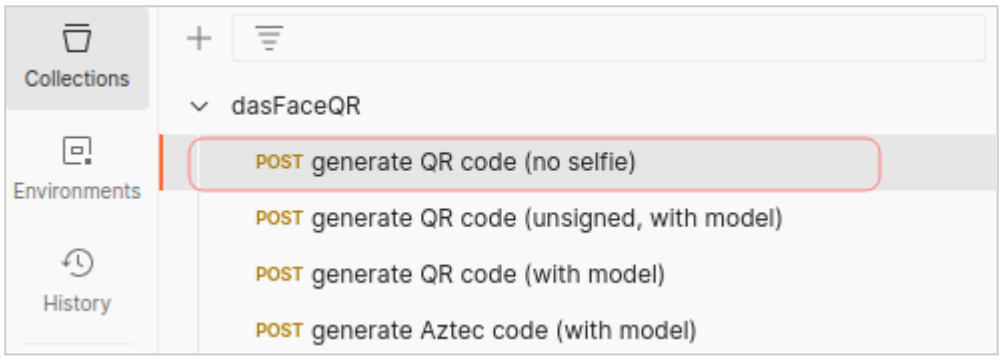


In this step, the selfie image should be selected (*selfieImage* key). Moreover, according to the scenario necessities, the contextual data could be set in *contextualData* key.

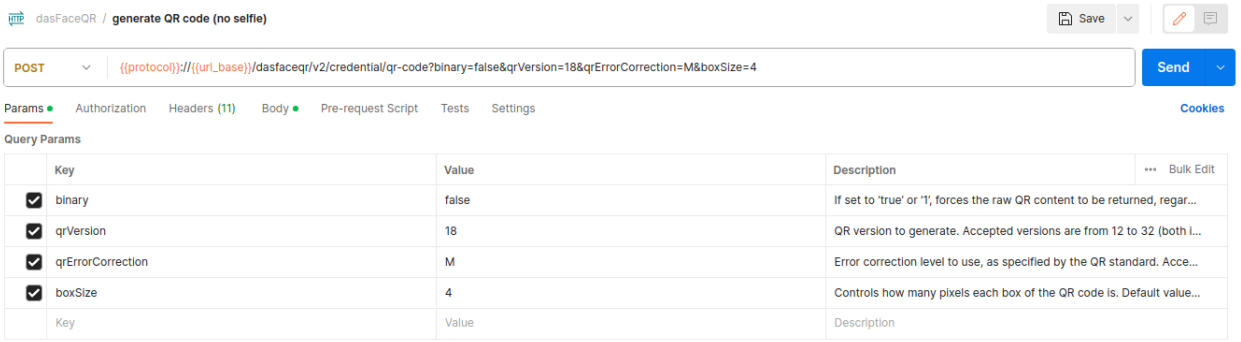
4. Click on the Send button. As a response, a QR code will be generated.

Non-biometric QR

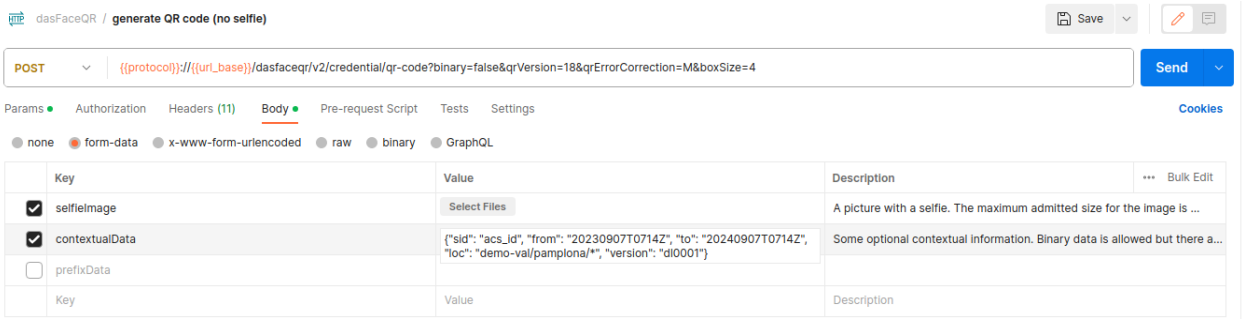
1. Select from Collections the option to generate QR code (no selfie)



2. Set query parameters according to the tenant configuration



3. Set body



In no-biometric QR case, **no image file** should be selected in the *selfieImage* key. Nevertheless, depending on the scenario necessities, the contextual data could be set in *contextualData* key.

4. Click on the Send button. As a response, a QR code will be generated.

Contextual data

The QR code used for biometric and non-biometric authentications includes contextual data which is information that provides context on a user and/or temporal/geographical restrictions of the scanned credential. The json schemas define required and optional properties of the contextual data as well as its format. Currently, two schemas for contextual data are supported:

- se0001 (requires user identifier information)
- dl0001 (requires user identifier as well as temporal/geographical information)

Given json schemas define a list of required fields, nevertheless if any **additional property** is added to the json file, the file is considered as **valid** (By default any additional properties are allowed).

se0001

The format of the contextual data given by json schema se0001 includes the following properties:

Property	Description	Format	Requirement state	Example
sid	single identifier for credential (access control system identifier acs_id)	string	required	username0001
version	json schema version	const	optional	se0001

The **sid** property specified in the schema is **required**, while **version** property is **optional** in this case. Nevertheless, if no scheme is specified in contextual data, the se0001 schema is applied **by default**. Examples of correct contextual data with json schema se0001:

- *{"sid": "username0001"}*
- *{"sid": "username0001", "version": "se0001"}*

dl0001

The format of the contextual data given by json schema dl0001 includes the following properties:

Property	Description	Format	Example
sid	single identifier for credential (access control system identifier acs_id)	string	username0001
from	starting datetime range for authorization	string: date-time	20230907T0714Z
to	ending datetime range for authorization	string: date-time	20240907T0714Z
loc	location for access point authorization	string	demo-val/pamplona/*
version	json schema version	const	dl0001

The date-time format should be YYYYMMDDTHHmmZ where,

- YYYYMMDD – is the date: year-month-day.
- The character "T" is used as the delimiter.
- HH:mm – is the time in UTC: hours, minutes.
- The 'Z' part denotes that the time zone used is UTC.

All the properties specified in the schema are **required**. An example of correct contextual data with json schema dl0001:

- *{"sid": "username0001", "from": "20230907T0714Z", "to": "20240907T0714Z", "loc": "demo-val/pamplona/*", "version": "dl0001"}*

Location range

It is important to stress that location can be restricted at different levels based on the access point structure:

- "loc": "tenant/*"
- "loc": "tenant/scope/*"
- "loc": "tenant/scope/facility/*"
- "loc": "tenant/scope/facility/boundary/*"
- "loc": "tenant/scope/facility/boundary/lane"

In order to validate the credential, the location restriction included in the QR code is compared with the terminal access point.

Use of contextual data

The contextual data is used at the authorization step. Thus, once the QR code is scanned, the user is being first authenticated. Once the authentication is successfully completed then the contextual data is sent to the authorization service via **authentication_succedeed** event of Phacex protocol (in the *data* field). The authorization service makes a decision based on the validity of the contextual data.

Format error

Although the access control system makes use of the contextual data when making authorization decisions, the contextual data format is validated against the selected schema during the authentication process. Thus, if the contextual data does not fulfill one of the given schemas or another schema version is introduced, the *FormatError* will be obtained as soon as the QR code is scanned. Examples of **incorrect** contextual data:

- incorrect schema version: *{"sid": "username0001", "version": "dl0002"}*
- missing fields: *{"version": "dl0001"}*
- missing fields: *{"sid": "username0001", "from": "20230907T0714Z", "to": "20240907T0714Z", "version": "dl0001"}*
- missing fields: *{"sid": "username0001", "loc": "demo-val/pamplona/*", "version": "dl0001"}*

Additionally, if any of the keys included in the contextual data is from the following list: ["*loc*", "*from*", "*to*"], the contextual data should also include the schema version. Otherwise, *FormatError* will be obtained. Example of **incorrect** contextual data:

- {"*sid*": "*username0001*", "*from*": "*20230907T0714Z*", "*to*": "*20240907T0714Z*", "*loc*": "*demo-val/pamplona/**"}

In both cases, the authentication attempt ends with the **authentication_failed** event (reason: *incorrect_format*) of Phacex protocol.

On the other side, as **additional properties are allowed**, the following contextual data dictionaries are considered as **correct** from the json schema point of view:

- {"*sid*": "*username0001*", "*from*": "*20230907T0714Z*", "*to*": "*20240907T0714Z*", "*loc*": "*demo-val/pamplona/**", "*version*": "*se0001*"}
- {"*sid*": "*username0001*", "*additionalProperty*": "*additionalProperty*"}
- {"*sid*": "*username0001*", "*from*": "*20230907T0714Z*", "*to*": "*20240907T0714Z*", "*loc*": "*demo-val/pamplona/**", "*additionalProperty*": "*additionalProperty*", "*version*": "*dl0001*"}
- {"*sid*": "*username0001*", "*from*": "*20230907T0714Z*", "*to*": "*20240907T0714Z*", "*loc*": "*demo-val/pamplona/**", "*additionalProperty*": "*additionalProperty*", "*version*": "*se0001*"}

Supported QR versions and maximum contextual data lengths

Although certain flexibility is permitted when adding contextual data, we should encourage the client to use json format as close as possible to one of the given schemas. This is related to the **size limit** of the contextual data length, which depends on the selected QR parameters (*qrVersion* and *qrErrorCorrection*). The QR parameters selected in the API Specification section gives us a certain margin for the contextual data. If the available size is not enough for given contextual data, the following error response is obtained:

```
{
   "code": "ContextualDataTooLargeError",
   "message": "Contextual data will not fit into QR code (length: _ bits = _ bytes)"
}
```

To find more information about QR parameters and available contextual data length, visit the following link:

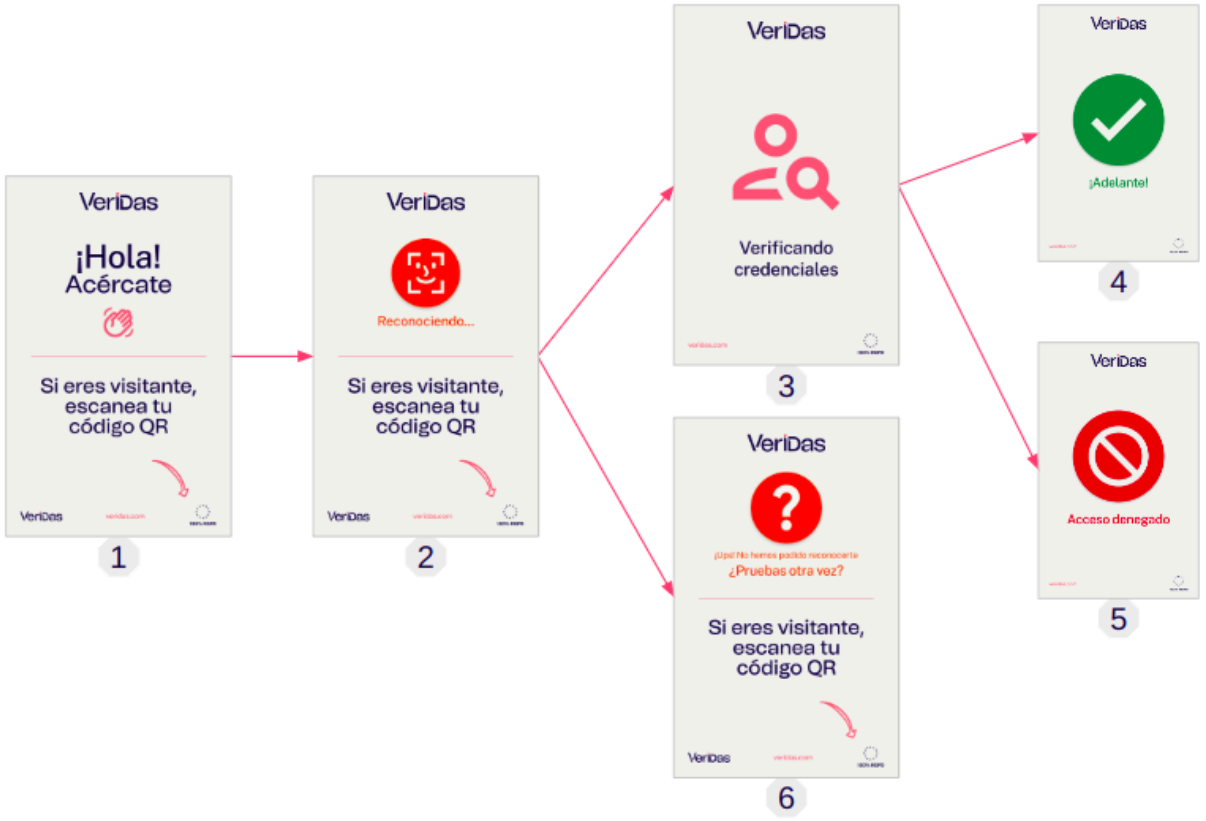
https://docs.veridas.com/das-faceqr/cloud/v2.6/annexes/supported-qr-versions/?h=contextual+data

/User experience with multimodal terminal

From the UX point of view, several improvements in the terminal interface have been made to provide clear and concise instructions for all groups of users. For instance, the screen has been divided into two areas. In the upper area, offers a step-by-step guidance through the identification process for common terminal users using face authentication. On the other hand, the bottom part of the screen shows instructions for occasional users who need to scan a QR code to start the authentication process.

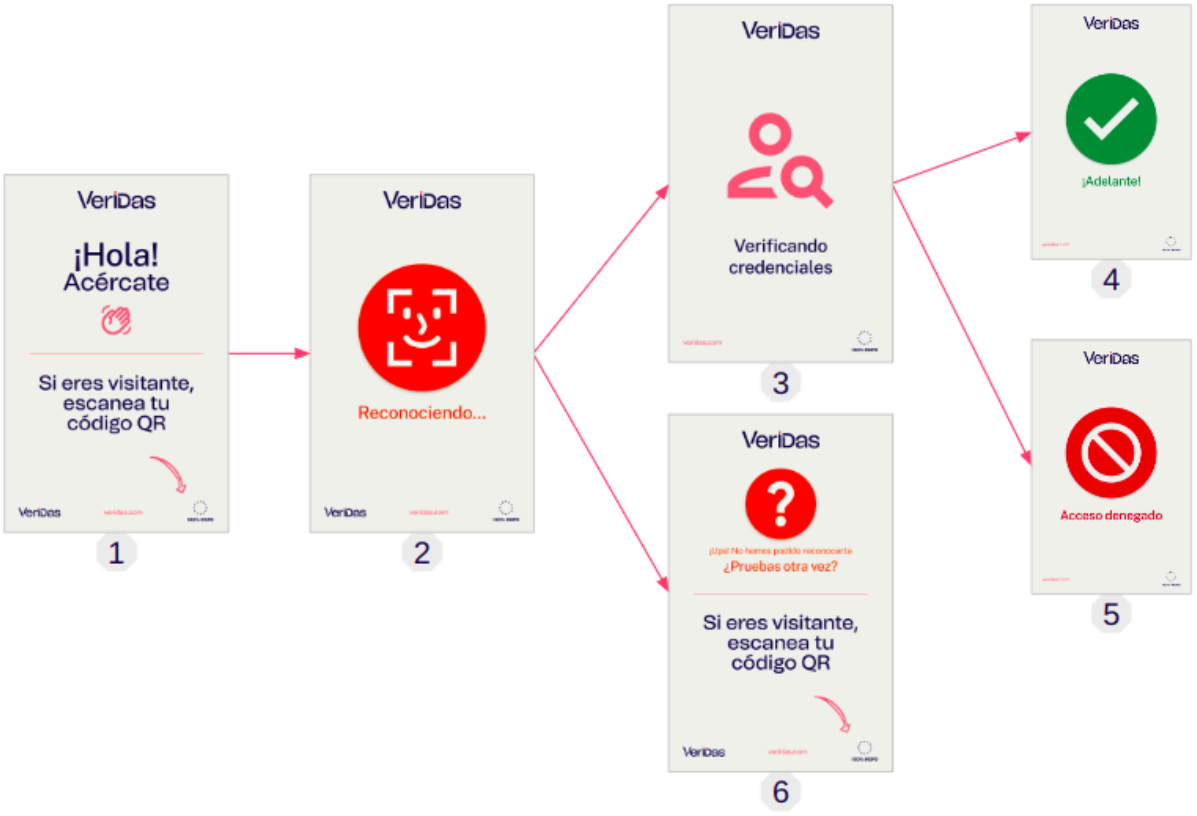
In the following, the schematic screen flows have been included to explain the user experience in case of different authentication modes.

Face authentication (1FA)



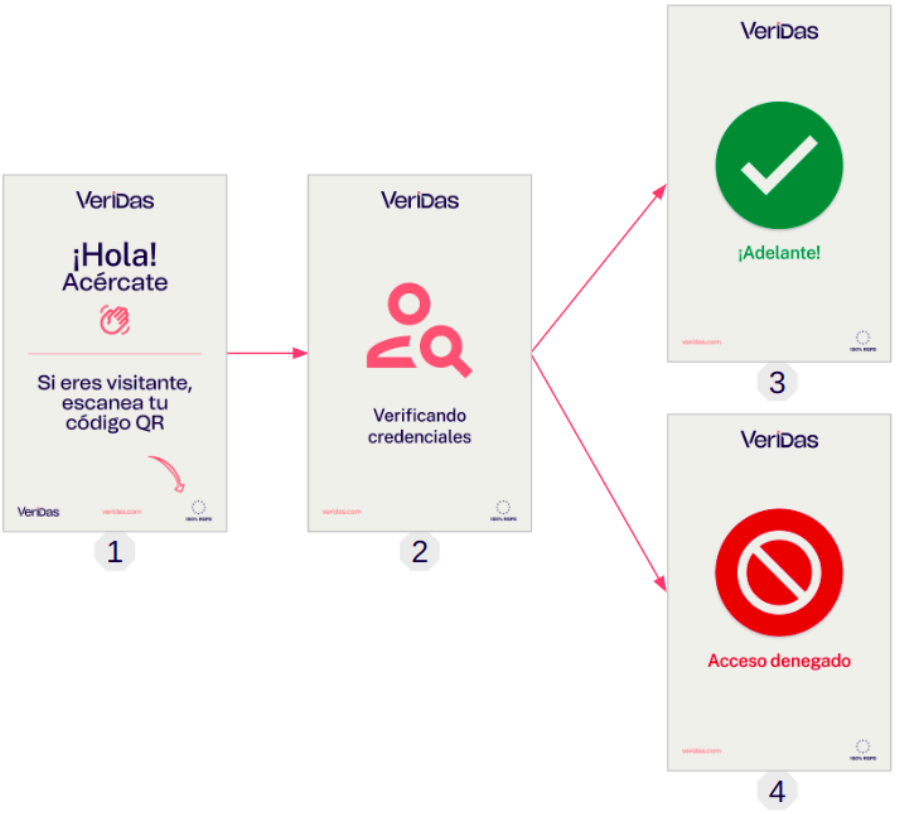
1. The user is being invited to get close to the terminal.
2. The authentication starts as soon as the face is detected.
3. Once the user is authenticated, the authorization process starts.
4. Access rights are being granted.
5. Access rights are being denied (more details about the denial reason can be found in logs).
6. The authentication time is out and the user is not recognized.

Biometric QR authentication (2FA)



1. The user is being invited to scan a QR code.
2. The authentication starts as soon as the valid QR code is read.
3. Once the user is authenticated, the authorization process starts.
4. Access rights are being granted.
5. Access rights are being denied (more details about the denial reason can be found in logs).
6. The authentication time is out and the user is not recognized.

Non-biometric QR authentication (1FA)



1. The user is being invited to scan a QR code.
2. The authorization process starts as soon as the valid QR code is read (authentication is completed).
3. Access rights are being granted.
4. Access rights are being denied (more details about the denial reason can be found in logs).