FaceMe SDK HTTP API
API Document

V2.1.0

Table of Contents

1. Interface

1.1. Health Monitor

Health check API for load balancer to monitor CyberLink APP server status

Request

HTTP request

```
GET http://app-server/mp/api/v1.0/health
```

Request body

Do not supply a request body with this method.

Response

HTTP Status Codes

- 200 OK : APP server works correctly
- 503 Service Unavailable : Unable to service

Body

- “FACEME IS OK” if APP server works correctly

1.2. Enrollment

Insert a new face

This method accepts images with following characteristics:

- **Protocol: HTTP/2**
- **Format: PNG/BMP/JPG/JPEG**

Request

HTTP request

POST http://app-server/mp/api/v1.0/records

Headers

- Content-Type: multipart/form-data

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

Request body

Parameter Name	**Type**	**Description**
imageMetadata	JSON	Contains information of enroll face: imageID
imageID	String	Unique ID of a face image
image	Binary	Binary of face image
features	JSON	Apply features on enrollment: qualityCheck, showDetail, enableStrictMode
qualityCheck	String	Setting of quality check
showDetail	String	Setting of detail information
enableStrictMode	Boolean	Enable/disable Strict Mode. The strict mode provide by CyberLink, it will use CyberLink internal encryption algorithm to decrypt input data. **This mode only support for web component.**

Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error

- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method
- 503 Service Unavailable : Unable to service

Result

Property Name	Type	Description
error	JSON	Contains result of execution: code, message
code	String	Detail error code. Would be empty if execution finishes correctly
message	String	Detail error message.
image_info	JSON	Contains information of input image
width	String	Width of the image
height	String	Height of the image
imageMetadata	JSON	Contains information of enroll face: imageID
imageID	String	Unique ID of a face image
face_info	JSON	Contains the detail information of face: boundingBox, faceLandmark, qualityCheck
boundingBox	JSON	The bounding box of a face in query image
left	String	Left position of a bounding box
top	String	Top position of a bounding box
right	String	Right position of a bounding box
bottom	String	Bottom position of a bounding box
faceLandmark	JSON	Feature landmark set of the face: left_eye, right_eye, nose, mouth_left, mouth_right
left_eye	JSON	The position of left eye with x and y value
right_eye	JSON	The position of right eye with x and y value
nose	JSON	The position of nose with x and y value
mouth_right	JSON	The position of mouth right with x and y value
mouth_left	JSON	The position of mouth left with x and y value
qualityCheck	JSON	The result of qualityCheck
exposureThreshold	String	The threshold of over exposure and under exposure: “over_threshold/under_threshold”
exposureValue	String	The exposure value of the face
blurThreshold	String	The threshold of a blur face
blurValue	String	The blue value of the face
faceAngle	JSON	Detail information of face angle contains: yaw, pitch, roll
yaw	Double	Angle of swivels to left or right
pitch	Double	Angle of tilting forward or backward

roll	Double	Angle of rolling to left or right

Detail error message

Code	Message	HTTP Status Code
1000	Bad argument in imageMetada	400
1100	Image format not support	400
1101	Image ID exists in database	400
1102	Image ID is not allowed	400
1103	License not support	400
1200	Quality check failed: Not focused	400
1201	Quality check failed: Eyes are closed	400
1202	Quality check failed: Covered by things	400
1203	Quality check failed: Too close between eyes	400
1204	Quality check failed: Over exposure	400
1205	Quality check failed: More than one face	400
1206	Unable to detect face	400
1207	Quality check failed: Head turned	400
1208	Quality check failed: Gray scale detected	400

Example

```
# Request
curl -X POST http://localhost:8080/mp/api/v1.0/records --user
cyberlink:377339aa107ae58abe8e3c7fd30218b6 -F
"imageMetadata={\"imageID\":\"00001\"};type=application/json" -F
"features={\"showDetail\":\"true\"};type=application/json" -F "image=@face.jpg"
# Response
HTTP/2.0 200 OK
{
   "error": {"code":"", "message": "success"},
   "imageMetata": { "imageID": "00001"},
   "image_info": {
      "width": "413",
      "height": "531"
   },
   "face_info": {
      "boundingBox": {
         "left": 91,
         "top": 138,
         "right": 318,
          "bottom": 451
        },
        "faceLandmark": {
          "left_eye": {
            "x": "153",
            "y": "261"
          },
          "right_eye": {
            "x": "259",
            "y": "264"
          },
          "nose": {
            "x": "205",
            "y": "334"
          },
          "mouth_right": {
            "x": "253",
            "y": "379"
          },
          "mouth_left": {
            "x": "154",
            "y": "376"
          }
        },
        "qualityCheck": {
          "exposureThreshold": "0.1568627506494522/0.7058823704719544",
          "exposureValue": "0.501960813999176",
          "blurThreshold": "0.7165354490280151",
          "blurValue": "0.01516"
        },
        "faceAngle": {
          "yaw": 5.241986274719238,
          "pitch": -2.735459804534912,
          "roll": -7.189086437225342
        }
    }
}
```

1.3. Delete record

Remove a face record

Request

HTTP request

POST http://app-server/mp/api/v1.0/withdraw

Headers

- Content-Type: application/json

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

Request body

Parameter Name	Type	Description
imageID	String	Unique ID of a face image

Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error
- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method
- 503 Service Unavailable : Unable to service

Result

Property Name	Type	Description
error	JSON	Contains result of execution: code, message
code	String	Detail error code. Would be empty if execution finishes correctly
message	String	Detail error message.

Detail error message

Code	Message	HTTP Status Code
1000	Bad argument	400

Example

```
# Request
curl -X POST http://localhost:8080/mp/api/v1.0/withdraw --user
cyberlink:377339aa107ae58abe8e3c7fd30218b6 -H "Content-Type:
application/json" -d {\"imageID\":\"00001\"}

# Response
HTTP/2.0 200 OK
{
   "error": {"code":"", "message": "success"}
}
```

1.4. 1-to-1 Comparison

Compare similarity between two face images

Request

HTTP request

POST http://app-server/mp/api/v1.0/comparison

Headers

- Content-Type: multipart/form-data

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

Request body

Parameter Name	Type	Description
features	JSON	Apply features on comparison: qualityCheck, showDetail, enableStrictMode
qualityCheck	String	Setting of quality check
showDetail	String	Setting of detail information
enableStrictMode	Boolean	Enable/disable Strict Mode. The strict mode provide by CyberLink, it will use CyberLink internal encryption algorithm to decrypt input data. **This mode only support for web component.**
image1	Binary	Binary of first face image
Image2	Binary	Binary of second face image

Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error
- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method

- 503 Service Unavailable : Unable to service

Result

Property Name	Type	Description
error	JSON	Contains result of execution: code, message
code	String	Detail error code. Would be empty if execution finishes correctly
message	String	Detail error message.
image_info	JSON	Contains information of input image
width	String	Width of the image
height	String	Height of the image
image_info_2	JSON	Contains information of input image 2, same format as image_info
comparison_threshold	JSON	Contains thresholds of different precision level
10e-6	String	Ultra precision level
10e-5	String	Advanced precision level
10e-4	String	Basic precision level
result	JSON	Contains the result of 1-to-1 comparson, face_info, face_info_2
isMatch	String	Indicates if two face belong to same person
confidence	String	The confidence of comparison result
face_info	JSON	Contains the detail information of face1: boundingBox, faceLandmark, qualityCheck
boundingBox	JSON	The bounding box of a face in query image
left	String	Left position of a bounding box
top	String	Top position of a bounding box
right	String	Right position of a bounding box
bottom	String	Bottom position of a bounding box
faceLandmark	JSON	Feature landmark set of the face: left_eye, right_eye, nose, mouth_left, mouth_right
left_eye	JSON	The position of left eye with x and y value
right_eye	JSON	The position of right eye with x and y value
nose	JSON	The position of nose with x and y value
mouth_right	JSON	The position of mouth right with x and y value
mouth_left	JSON	The position of mouth left with x and y value
qualityCheck	JSON	The result of qualityCheck
exposureThreshold	String	The threshold of over exposure and under exposure: “over_threshold/under_threshold”

exposureValue	String	The exposure value of the face
blurThreshold	String	The threshold of a blur face
blurValue	String	The blue value of the face
faceAngle	JSON	Detail information of face angle contains: yaw, pitch, roll
yaw	Double	Angle of swivels to left or right
pitch	Double	Angle of tilting forward or backward
roll	Double	Angle of rolling to left or right
face_info_2	JSON	Contains the detail information of face2, same format as face_info_2

Detail error message

Code	Message	HTTP Status Code
1000	Bad argument	400
1100	Image format not support	400
1103	License not support	400
1200	Quality check failed: Not focused	400
1201	Quality check failed: Eyes are closed	400
1202	Quality check failed: Covered by things	400
1203	Quality check failed: Too close between eyes	400
1204	Quality check failed: Over exposure	400
1205	Quality check failed: More than one face	400
1206	Unable to detect face	400
1207	Quality check failed: Head turned	400
1208	Quality check failed: Gray scale detected	400

Example

```
# Request
curl -X POST http://localhost:8080/mp/api/v1.0/comparison --user
cyberlink:377339aa107ae58abe8e3c7fd30218b6 -F
"features={\"qualityCheck\":"true",\"showDetail\":\"true\"};type=application/json" -
F "image1=@face1.jpg" -F "image2=@face2.jpg"

# Response
HTTP/1.1 200 OK
{
  "error": {
    "code": "",
    "message": "Success"
  },
  "image_info": {
    "width": "87",
    "height": "102"
  },
  "image_info_2": {
    "width": "106",
    "height": "130"
  },
  "comparison_threshold": {
    "10e-4": "0.75",
    "10e-5": "0.7732314",
    "10e-6": "0.794489"
  },
  "result": {
    "isMatch": "false",
    "confidence": "0.49556762",
    "face_info": {
      "boundingBox": {
        "left": 4,
        "top": 2,
        "right": 75,
        "bottom": 96
      },
      "faceLandmark": {
        "left_eye": {
          "x": "19",
          "y": "37"
        },
        "right_eye": {
          "x": "51",
          "y": "40"
        },
        "nose": {
          "x": "30",
          "y": "59"
        },
        "mouth_right": {
          "x": "47",
          "y": "75"
```

```
            },
            "mouth_left": {
               "x": "19",
               "y": "73"
            }
         },
         "qualityCheck": {
            "exposureThreshold": "0.1568627506494522/0.7058823704719544",
            "exposureValue": "0.5098039507865906",
            "blurThreshold": "0.7165354490280151",
            "blurValue": "0.3151261"
         }
      },
      "face_info_2": {
         "boundingBox": {
            "left": 15,
            "top": -12,
            "right": 99,
            "bottom": 122
         },
         "faceLandmark": {
            "left_eye": {
               "x": "34",
               "y": "46"
            },
            "right_eye": {
               "x": "75",
               "y": "48"
            },
            "nose": {
               "x": "51",
               "y": "76"
            },
            "mouth_right": {
               "x": "71",
               "y": "97"
            },
            "mouth_left": {
               "x": "35",
               "y": "96"
            }
            },
            "qualityCheck": {
                "exposureThreshold": "0.1568627506494522/0.7058823704719544",
                "exposureValue": "0.6509804129600525",
                "blurThreshold": "0.7165354490280151",
                "blurValue": "0.45612"
            },
            "faceAngle": {
                "yaw": 5.241986274719238,
                "pitch": -2.735459804534912,
                "roll": -7.189086437225342
            }
        }
    }
}
```

1.5. 1-to-1 Face Templates Comparison

Compare similarity between two faces according to their face templates.

The two faces for comparing should use the same feature type.

If the feature type is different, the API will respond error because it's meaningless.

Request

HTTP request

POST http://app-server/mp/api/v1.0/face/compare11

Headers

- Content-Type: multipart/form-data

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

Request body

Parameter Name	Type	Description
facesInfo	JSON	The face template detail information for compare. *The json contains attributes listed in table 1.*
face1Template	Binary	Face template in binary format of the 1st face.
face2Template	Binary	Face template in binary format of the 2nd face.

Table 1 - facesInfo

Parameter Name	Type	Description
face1FeatureType	Integer	The feature type of 1st face. *The valid value listed in table 2.*
face2FeatureType	Integer	The feature type of 2nd t face. *The valid value listed in table 2.*
face1FeatureSubType	Integer	The feature sub type of 1st face. *The valid value listed in table 3.*
face2FeatureSubType	Integer	The feature sub type of 2nd face. *The valid value listed in table 3.*
face1ByteOrder	String	The face template byte order of 1st face. *The valid value listed in table 4.*

face2ByteOrder	String	The face template byte order of 2nd face. *The valid value listed in table 4.*

Table 2 - FeatureType

Name	Value	Description
High Precision	2	High Precision (H1 or H2)
Ultra High Precision	3	Ultra High Precision (UH)
Very High Precision	4	Very High Precision (VH)
Ultra High 3 Precision	5	Ultra High 3 Precision (UH3)
High 3 Precision	6	High 3 Precision (H3)

Table 3 - FeatureSubType

Name	Value	Description
None	0	Use this value for most feature type.
Asian	1	Use this value if using H2 model. You must combine it with Feature Type (2).

Table 4 - ByteOrder

If you are using Windows/Linux/iOS FaceMe SDK, please use "little"

If you are using Android FaceMe SDK, please use "big".

Name	Value	Description
Big-endian	"big"	Byte order about uploading face template. **It will be default value if identified value is neither "big" nor "little".**
Little-endian	"little"	Byte order about uploading face template.

Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error
- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method
- 406 Not Acceptable: Invalid parameters.

Result

Property Name	Type	Description
confidence	Float	The confidence of comparison result

thresholds	JSON	Contains thresholds of different precision levels. *The json contains attributes listed in table 5.*
isMatch	Boolean	Indicates if two faces belong to same person. It will return true If the confidence higher than Ultra precision level. We also provide other thresholds. You can compare the confidence of other precision level threshold by your own.

Table 5 - Thresholds

Property Name	Type	Description
10e-6	Float	Ultra precision level threshold.
10e-5	Float	Advanced precision level threshold.
10e-4	Float	Basic precision level threshold.

Example

# Request

```
curl -X POST http://localhost:8080/mp/api/v1.0/face/compare11 --user
cyberlink:377339aa107ae58abe8e3c7fd30218b6 -F
'facesInfo={"face1FeatureType":3,"face1FeatureSubType":0,"face1ByteOr
der":"big","face2FeatureType":3,"face2FeatureSubType":0,"face2ByteOrd
er":"big"}' -F face1Template=@face1.bin -F face2Template=@face2.bin
```

# Response

```
HTTP/1.1 200 OK
{
    "isMatch": true,
    "confidence":0.82512351,
    "thresholds": {
        "10e-6": 0.794489,
        "10e-5": 0.7732314,
        "10e-4": 0.75
    }
}
```

1.6. 1-to-N Comparison

Search similar faces from enrolled face dataset

Request

HTTP request

POST http://app-server/mp/api/v1.0/comparison

Headers

- Content-Type: multipart/form-data

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

Request body

Parameter Name	Type	Description
searchCriteria	JSON	Contains criteria of search request: returnCount
returnCount	String	Number of similar faces in result
features	JSON	Apply features of comparison: qualityCheck, showDetail, enableStrictMode
qualityCheck	String	Setting of quality check
showDetail	String	Setting of show detail information
enableStrictMode	Boolean	Enable/disable Strict Mode. The strict mode provide by CyberLink, it will use CyberLink internal encryption algorithm to decrypt input data. **This mode only support for web component.**
image1	Binary	Binary of face image

Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error

- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method
- 503 Service Unavailable : Unable to service

Result

Property Name	Type	Description
error	JSON	Contains result of execution: code, message
code	String	Detail error code. Would be empty if execution finishes correctly
message	String	Detail error message.
image_info	JSON	Contains information of input image
width	String	Width of the image
height	String	Height of the image
comparison_threshold	JSON	Contains thresholds of different precision level
10e-6	String	Ultra precision level
10e-5	String	Advanced precision level
10e-4	String	Basic precision level
result	JSON	Contains the result of 1-to-m comparson: bounding_box, similar_face
face_info	JSON	Contains the detail information of face: boundingBox, faceLandmark, qualityCheck
boundingBox	JSON	The bounding box of a face in query image
left	String	Left position of a bounding box
top	String	Top position of a bounding box
right	String	Right position of a bounding box
bottom	String	Bottom position of a bounding box
faceLandmark	JSON	Feature landmark set of the face: left_eye, right_eye, nose, mouth_left, mouth_right
left_eye	JSON	The position of left eye with x and y value
right_eye	JSON	The position of right eye with x and y value
nose	JSON	The position of nose with x and y value
mouth_right	JSON	The position of mouth right with x and y value
mouth_left	JSON	The position of mouth left with x and y value
qualityCheck	JSON	The result of qualityCheck
exposureThreshold	String	The threshold of over exposure and under exposure
exposureValue	String	The exposure value of the face
blurThreshold	String	The threshold of a blur face

blurValue	String	The blue value of the face
faceAngle	JSON	Detail information of face angle contains: yaw, pitch, roll
yaw	Double	Angle of swivels to left or right
pitch	Double	Angle of tilting forward or backward
roll	Double	Angle of rolling to left or right
similar_face	JSON	Contains a list of similar face
imageID	String	Unique ID of a face image
confidence	String	The confidence of comparison result

Detail error message

Code	Message	HTTP Status Code
1000	Bad argument	400
1100	Image format not support	400
1103	License not support	400
1200	Quality check failed: Not focused	400
1201	Quality check failed: Eyes are closed	400
1202	Quality check failed: Covered by things	400
1203	Quality check failed: Too close between eyes	400
1204	Quality check failed: Over exposure	400
1205	Quality check failed: More than one face	400
1206	Unable to detect face	400
1207	Quality check failed: Head turned	400
1208	Quality check failed: Gray scale image	400

Example

# Request

```
curl -X POST http://localhost:8080/mp/api/v1.0/comparison --user
cyberlink:377339aa107ae58abe8e3c7fd30218b6 -F
"searchCriteria={\"returnCount\":\"2\"};type=application/json" -F
"features={\"qualityCheck\":\"true\",\"showDetail\":\"true\"};type=application/json
" -F "image1=@face1.jpg"
```

# Response

```
HTTP/1.1 200 OK
{
   "error": {
      "code": "",
    "message": "Success"
  },
  "image_info": {
    "width": "87",
    "height": "102"
  },
  "comparison_threshold": {
    "10e-4": "0.75",
    "10e-5": "0.7732314",
    "10e-6": "0.794489"
  },
  "result": [
    {
      "face_info": {
        "boundingBox": {
          "left": 97,
          "top": 100,
          "right": 338,
          "bottom": 443
        },
        "faceLandmark": {
          "left_eye": {
            "x": "155",
            "y": "235"
          },
          "right_eye": {
            "x": "266",
            "y": "230"
          },
          "nose": {
            "x": "211",
            "y": "298"
          },
          "mouth_right": {
            "x": "257",
            "y": "362"
          },
          "mouth_left": {
            "x": "169",
            "y": "364"
          }
```

```
            },
            "qualityCheck": {
              "exposureThreshold": "0.1568627506494522/0.7058823704719544",
              "exposureValue": "0.5803921818733215",
              "blurThreshold": "0.7165354490280151",
              "blurValue": "0.4634420871734619"}
            },
            "faceAngle": {
              "yaw": 5.241986274719238,
              "pitch": -2.735459804534912,
              "roll": -7.189086437225342
            },
            "similiar_face": [ {"imageID": "1", "confidence": "0.998"},
                               {"imageID": "3", "confidence": "0.964"},
                               {"imageID": "752", "confidence": "0.960"}]
          },
          {
            "face_info": {
              "boundingBox": {
                "left": 882,
                "top": 100,
                "right": 338,
                "bottom": 443
              },
              "faceLandmark": {
                "left_eye": {
                  "x": "155",
                  "y": "235"
                },
                "right_eye": {
                  "x": "266",
                  "y": "230"
                },
                "nose": {
                  "x": "211",
                  "y": "298"
                },
                "mouth_right": {
                  "x": "257",
                  "y": "362"
                },
                    "mouth_left": {
                       "x": "169",
                       "y": "364"
                    }
                 },
                 "qualityCheck": {
                    "exposureThreshold": "0.1568627506494522/0.7058823704719544",
                    "exposureValue": "0.5803921818733215",
                    "blurThreshold": "0.7165354490280151",
                    "blurValue": "0.4634420871734619"
                 },
                 "faceAngle": {
                    "yaw": 5.241986274719238,
                    "pitch": -2.735459804534912,
                    "roll": -7.189086437225342
                 }
              },
              "similiar_face": [ {"imageID": "2399021", "confidence": "0.824"},
                                 {"imageID": "2641", "confidence": "0.818"},
                                 {"imageID":  "11992", "confidence": "0.522"}
               ]
           }
        ]
}
```

1.7. 1-to-1 ID Comparison

Compare similarity between input image and one specific image ID

Request

HTTP request

`POST http://app-server/mp/api/v1.0/comparison`

Headers

- Content-Type: multipart/form-data

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

Request body

Parameter Name	Type	Description
searchCriteria	JSON	Contains criteria of search request: imageID
returnCount	String	Number of similar faces in result
features	JSON	Apply features of comparison: qualityCheck, showDetail, enableStrictMode
qualityCheck	String	Setting of quality check
showDetail	String	Setting of show detail information
enableStrictMode	Boolean	Enable/disable Strict Mode. The strict mode provide by CyberLink, it will use CyberLink internal encryption algorithm to decrypt input data. **This mode only support for web component.**
image1	Binary	Binary of face image

Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error
- 401 Not Authorized: Lack of valid authentication credentials

- 405 Method Not Allow: Only support POST by this method
- 503 Service Unavailable : Unable to service

Result

Property Name	Type	Description
error	JSON	Contains result of execution: code, message
code	String	Detail error code. Would be empty if execution finishes correctly
message	String	Detail error message.
image_info	JSON	Contains information of input image
width	String	Width of the image
height	String	Height of the image
comparison_threshold	JSON	Contains thresholds of different precision level
10e-6	String	Ultra precision level
10e-5	String	Advanced precision level
10e-4	String	Basic precision level
result	JSON	Contains the result of 1-to-1 ID comparison: isMatch, confidence, face_info
isMatch	String	Indicates if two face belong to same person
confidence	String	The confidence of comparison result
face_info	JSON	Contains the detail information of face
boundingBox	JSON	The bounding box of a face in query image
left	String	Left position of a bounding box
top	String	Top position of a bounding box
right	String	Right position of a bounding box
bottom	String	Bottom position of a bounding box
faceLandmark	JSON	Feature landmark set of the face: left_eye, right_eye, nose, mouth_left, mouth_right
left_eye	JSON	The position of left eye with x and y value
right_eye	JSON	The position of right eye with x and y value
nose	JSON	The position of nose with x and y value
mouth_right	JSON	The position of mouth right with x and y value
mouth_left	JSON	The position of mouth left with x and y value
qualityCheck	JSON	The result of qualityCheck
exposureThreshold	String	The threshold of over exposure and under exposure
exposureValue	String	The exposure value of the face
blurThreshold	String	The threshold of a blur face

blurValue	String	The blue value of the face
faceAngle	JSON	Detail information of face angle contains: yaw, pitch, roll
yaw	Double	Angle of swivels to left or right
pitch	Double	Angle of tilting forward or backward
roll	Double	Angle of rolling to left or right

Detail error message

Code	Message	HTTP Status Code
1000	Bad argument	400
1100	Image format not support	400
1103	License not support	400
1104	Image ID is not exist	400
1200	Quality check failed: Not focused	400
1201	Quality check failed: Eyes are closed	400
1202	Quality check failed: Covered by things	400
1203	Quality check failed: Too close between eyes	400
1204	Quality check failed: Over exposure	400
1205	Quality check failed: More than one face	400
1206	Unable to detect face	400
1207	Quality check failed: Head turned	400
1208	Quality check failed: Gray scale image	400

Example

```
# Request
curl -X POST http://localhost:8080/mp/api/v1.0/comparison --user
cyberlink:377339aa107ae58abe8e3c7fd30218b6 -F
"searchCriteria={\"imageID\":\"00001\"};type=application/json" -F
"features={\"qualityCheck\":\"true\",\"showDetail\":\"true\"};type=application/json
" -F "image1=@face1.jpg"

# Response
HTTP/1.1 200 OK
{
   "error": {
      "code": "",
      "message": "Success"
   },
   "image_info": {
    "width": "87",
    "height": "102"
 },
 "comparison_threshold": {
    "10e-4": "0.75",
    "10e-5": "0.7732314",
    "10e-6": "0.794489"
 },
 "result": [
    {
    "isMatch": "false",
    "confidence": "0.55156762",
    "face_info": {
       "boundingBox": {
          "left": 97,
          "top": 100,
          "right": 338,
          "bottom": 443
       },
       "faceLandmark": {
          "left_eye": {
             "x": "155",
             "y": "235"
          },
          "right_eye": {
             "x": "266",
             "y": "230"
          },
          "nose": {
             "x": "211",
             "y": "298"
          },
          "mouth_right": {
             "x": "257",
             "y": "362"
          },
          "mouth_left": {
             "x": "169",
             "y": "364"
          }
       },
            "qualityCheck": {
                "exposureThreshold": "0.1568627506494522/0.7058823704719544",
                "exposureValue": "0.5803921818733215",
                "blurThreshold": "0.7165354490280151",
                "blurValue": "0.4634420871734619"}
            },
            "faceAngle": {
                "yaw": 5.241986274719238,
                "pitch": -2.735459804534912,
                "roll": -7.189086437225342
            }
        }
    }
}
```

1.8. Anti-Spoofing without Interactions

Verify if there is a spoofing attack in input images.
For how to integrate the anti-spoofing API, please leverage the Web Component and the Web Sample code.

Request

HTTP request

```
POST http://app-server/mp/api/v1.0/spoofingcheck
```

Headers

- Content-Type: multipart/form-data

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

Request body

Parameter Name	Type	Description
detail	JSON	Contains detail of capture device: cameraInfo, status, still, precisionLevel
cameraInfo	String	Detail information of camera
status	Array	Status value from browser plugin
still	Int	Indicator of which image that face is stayed still in front of camera
precisionLevel	String	Three precision levels of spoofing check: fast, standard, high
features	JSON	Apply features of spoofing check: enableStrictMode
enableStrictMode	Boolean	Enable/disable Strict Mode. The strict mode provide by CyberLink, it will use CyberLink internal encryption algorithm to decrypt input data. **This mode only support for web component.**
enable2Stage	Boolean	*The default value is true.* Enable or disable second stage anti-spoofing. If the value is false , "LivenessScore**"** will not

		response “Second_Stage”.
image1	Binary	Binary of face image
image2	Binary	Binary of face image
…		
Image20	Binary	Binary of face image
…		

Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error
- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method
- 503 Service Unavailable : Unable to service

Result

Property Name	Type	Description
error	JSON	Contains result of execution: code, message
code	String	Detail error code. Would be empty if execution finishes correctly
message	String	Detail error message.
result	JSON	Contains the result of spoofing verification and confidence
isSpoofing	String	Indicates if spoofing attack is detected
LivenessScore	String	The confidence of verification result: Liveness, Second_Stage, Spoofing
image	Binary	Image with quality in the sequence , only response when the result is Liveness

Detail error message

Code	Message	HTTP Status Code
1000	Bad argument	400
1100	Image format not support	400

Example

```
# Request
curl -X POST http://localhost:8080/mp/api/v1.0/spoofingcheck \
--user cyberlink:377339aa107ae58abe8e3c7fd30218b6 \
-F 'detail={"precisionLevel":"standard","cameraInfo":"Vimicro USB2.0 PC Camera
(0ac8:3410)","status":[0.648,0.593,0.552,0.534,0.496,0.536,0.539,0.565,0.6136,0.6245,
0.625,0.6070,0.6367,0.648,0.645],"still":6,"enable2Stage":false}' \
-F 'image1=@face1.jpg' \
-F 'image2=@face2.jpg' \
-F 'image3=@face3.jpg' \
-F 'image4=@face4.jpg' \
-F 'image5=@face5.jpg' \
-F 'image6=@face6.jpg' \
-F 'image7=@face7.jpg' \
-F 'image8=@face8.jpg' \
-F 'image9=@face9.jpg' \
-F 'image10=face10.jpg' \
-F 'image11=@face11.jpg' \
-F 'image12=@face12.jpg' \
-F 'image13=@face13.jpg' \
-F 'image14=@face14.jpg' \
-F 'image15=@face15.jpg' \
-F 'image16=@face16.jpg' \
-F 'image17=@face17.jpg' \
-F 'image18=@face18.jpg' \
-F 'image19=@face19.jpg' \
-F 'image20=@face20.jpg'

# Response
HTTP/1.1 200 OK
{
   "result": {"isSpoofing": "Second_Stage", "confidence": "0.52726555" },
   "error": {"code":"", "message": "success"}
}
[image]
```

1.9. Anti-Spoofing with Interactions

Second stage that to verify if there is a spoofing attack in input images.
For how to integrate the anti-spoofing API, please leverage the Web Component and the Web Sample code.

Request

HTTP request

```
POST http://app-server/mp/api/v1.0/spoofingcheckV2
```

Headers

- Content-Type: multipart/form-data

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

Request body

Parameter Name	Type	Description
detail	JSON	Contains detail of capture device: cameraInfo, dir, previous
cameraInfo	String	Detail information of camera
dir	String	Direction of interaction
previous	String	Liveness score from the result of 1.6
features	JSON	Apply features of spoofing check: enableStrictMode
enableStrictMode	Boolean	Enable/disable Strict Mode. The strict mode provide by CyberLink, it will use CyberLink internal encryption algorithm to decrypt input data. **This mode only support for web component.**
image2	Binary	Binary of face image
…		
Image20	Binary	Binary of face image
…		

Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error
- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method
- 503 Service Unavailable : Unable to service

Result

Property Name	Type	Description
error	JSON	Contains result of execution: code, message
code	String	Detail error code. Would be empty if execution finishes correctly
message	String	Detail error message.
result	JSON	Contains the result of spoofing verification and confidence
isSpoofing	String	Indicates if spoofing attack is detected
LivenessScore	String	The confidence of verification result: Liveness, Spoofing
image	Binary	Image with quality in the sequence, only response when the result is Liveness

Detail error message

Code	Message	HTTP Status Code
1000	Bad argument	400
1100	Image format not support	400

Example

# Request

```
curl -X POST http://localhost:8080/mp/api/v1.0/spoofingcheckV2 \
--user cyberlink:377339aa107ae58abe8e3c7fd30218b6 \
--form 'detail={"precisionLevel":"standard","cameraInfo":"Vimicro USB2.0 PC
Camera (0ac8:3410)","dir":"left","previous":0.52726555}' \
-F 'image1=@face1.jpg' \
-F 'image2=@face2.jpg' \
-F 'image3=@face3.jpg' \
-F 'image4=@face4.jpg' \
-F 'image5=@face5.jpg' \
-F 'image6=@face6.jpg' \
-F 'image7=@face7.jpg' \
-F 'image8=@face8.jpg' \
-F 'image9=@face9.jpg' \
-F 'image10=face10.jpg' \
-F 'image11=@face11.jpg' \
-F 'image12=@face12.jpg' \
-F 'image13=@face13.jpg' \
-F 'image14=@face14.jpg' \
-F 'image15=@face15.jpg' \
-F 'image16=@face16.jpg' \
-F 'image17=@face17.jpg' \
-F 'image18=@face18.jpg' \
-F 'image19=@face19.jpg' \
-F 'image20=@face20.jpg'
```

# Response

```
HTTP/1.1 200 OK
{
   "result": {"isSpoofing": "Spoofing", "LivenessScore": "0.15" },
   "error": {"code":"", "message": "success"}
}
```

1.10. Anti-Spoofing with 2 stages

Two stage anti-spoofing detection that compose previous two APIs into one request.
For how to integrate the anti-spoofing API, please leverage the Web Component and the Web Sample code.

Request

HTTP request

POST http://app-server/mp/api/v1.0/spoofingcheckV3

Headers

- Content-Type: multipart/form-data

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

Request body

Parameter Name	Type	Description
detail	JSON	Contains detail of capture device: cameraInfo, status, dir, features.
cameraInfo	String	Detail information of camera
status	Array	Status value from browser plugin
still	Int	Indicator of which image that face is stayed still in front of camera
precisionLevel	String	Three precision levels of spoofing check: fast, standard, high
dir	String	Direction of interaction
features	JSON	Apply features of spoofing check: enableStrictMode
enableStrictMode	Boolean	Enable/disable Strict Mode. The strict mode provided by CyberLink, it will use CyberLink internal encryption algorithm to decrypt input data. **This mode only support for web component.**
image1	Binary Array	Binary of face images for the 1st stage detection.

image2	Binary Array	Binary of face images for the 2nd stage detection.

Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error
- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method
- 503 Service Unavailable : Unable to service

Result

Property Name	Type	Description
error	JSON	Contains result of execution: code, message
code	String	Detail error code. Would be empty if execution finishes correctly
message	String	Detail error message.
result	JSON	Contains the result of spoofing verification and confidence
isSpoofing	String	Indicates if spoofing attack is detected
LivenessScore	String	The confidence of verification result: Liveness, Spoofing
image1Index	Integer	Image index in image1 parameter with quality in the sequence, only response when the result is Liveness

Detail error message

Code	Message	HTTP Status Code
1000	Bad argument	400
1100	Image format not support	400

Example

```
# Request
curl -X POST 'http://localhost:8080/mp/api/v1.0/spoofingcheckV3' --user
cyberlink:377339aa107ae58abe8e3c7fd30218b6 -F
'detail={"precisionLevel":"standard","cameraInfo":"Vimicro USB2.0 PC Camera
(0ac8:3410)","status":[0.59953,0.6023,0.603,0.5909,0.5887,0.5333],"still":3,"dir":"left"}'
\
-F 'image1=@face1.jpg' \
-F 'image1=@face2.jpg' \
-F 'image1=@face3.jpg' \
-F 'image1=@face4.jpg' \
-F 'image1=@face5.jpg' \
-F 'image2=@face6.jpg' \
-F 'image2=@face7.jpg' \
-F 'image2=@face8.jpg' \
-F 'image2=@face9.jpg' \
-F 'image2=@face10.jpg'

# Response
HTTP/1.1 200 OK
{
   "result": {"isSpoofing": "Spoofing", "LivenessScore": "0.15", "image1Index": 4 },
   "error": {"code":"", "message": "success"}
}
```

1.11. Face image quality check

Check quality of face image

Request

HTTP request

`POST http://app-server/mp/api/v1.0/faceimagequalitycheck`

Headers

- Content-Type: multipart/form-data

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

Request body

Parameter Name	Type	Description
image	Binary	Binary of face image
features	JSON	Apply features of spoofing check: enableStrictMode
enableStrictMode	Boolean	Enable/disable Strict Mode. The strict mode provide by CyberLink, it will use CyberLink internal encryption algorithm to decrypt input data. **This mode only support for web component.**

Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error
- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method
- 503 Service Unavailable : Unable to service

Result

Property Name	Type	Description
error	JSON	Contains result of execution: code, message
code	String	Detail error code. Would be empty if execution finishes correctly
message	String	Detail error message.
image_info	JSON	Contains information of input image
width	String	Width of the image
height	String	Height of the image
face_info	JSON	Contains the detail information of face
boundingBox	JSON	The bounding box of a face in query image
left	String	Left position of a bounding box
top	String	Top position of a bounding box
right	String	Right position of a bounding box
bottom	String	Bottom position of a bounding box
faceLandmark	JSON	Feature landmark set of the face: left_eye, right_eye, nose, mouth_left, mouth_right
left_eye	JSON	The position of left eye with x and y value
right_eye	JSON	The position of right eye with x and y value
nose	JSON	The position of nose with x and y value
mouth_right	JSON	The position of mouth right with x and y value
mouth_left	JSON	The position of mouth left with x and y value
qualityCheck	JSON	The result of qualityCheck
exposureThreshold	String	The threshold of over exposure and under exposure
exposureValue	String	The exposure value of the face
blurThreshold	String	The threshold of a blur face
blurValue	String	The blue value of the face
faceAngle	JSON	Detail information of face angle contains: yaw, pitch, roll
yaw	Double	Angle of swivels to left or right
pitch	Double	Angle of tilting forward or backward
roll	Double	Angle of rolling to left or right

Detail error message

Code	Message	HTTP Status Code
1100	Image format not support	400
1103	License not support	400
1200	Quality check failed: Not focused	400
1201	Quality check failed: Eyes are closed	400

1202	Quality check failed: Covered by things	400
1203	Quality check failed: Too close between eyes	400
1204	Quality check failed: Over exposure	400
1205	Quality check failed: More than one face	400
1206	Unable to detect face	400
1207	Quality check failed: Head turned	400
1208	Quality check failed: Gray scale image	400

Example

```
# Request
curl -X POST http://localhost:8080/mp/api/v1.0/faceimagequalitycheck --user
cyberlink:377339aa107ae58abe8e3c7fd30218b6 -F "image=@face1.jpg"

# Response
HTTP/1.1 200 OK
{
   "error": {
      "code": "1201",
      "message": "Quality check failed: Eyes are closed"
   },
   "image_info": {
      "width": "463",
      "height": "455"
   },
   "face_info": {
      "boundingBox": {
         "left": 122,
         "top": 101,
         "right": 323,
         "bottom": 358
      },
      "faceLandmark": {
         "left_eye": {
            "x": "179",
            "y": "200"
         },
         "right_eye": {
            "x": "272",
            "y": "211"
         },
            "nose": {
               "x": "217",
               "y": "252"
            },
            "mouth_right": {
               "x": "264",
               "y": "289"
            },
            "mouth_left": {
               "x": "171",
               "y": "278"
            }
         },
         "qualityCheck": {
            "exposureThreshold": "0.11764705926179886/0.7843137383460999",
            "exposureValue": "0.4745098054409027",
            "blurThreshold": "0.4803149700164795",
            "blurValue": "0.0"
         },
         "faceAngle": {
            "yaw": 5.241986274719238,
            "pitch": -2.735459804534912,
            "roll": -7.189086437225342
         }
      }
}
```

1.12. Renew License

Renews license key.

This API requires internet connection.

For security concern this API only can access from localhost (127.0.0.1).

Request

HTTP request

```
POST http://app-server/mp/api/v1.0/license/renew
```

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error
- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method
- 406 Not Acceptable: Invalid parameters.

Result

Property Name	Type	Description
license	JSON	Contains license date information.
activationType	String	License activation type. It would be “Daily activation” and “First time activation”.
expireDate	String	Expire date for “First time activation”
deadline	String	Deadline date for “First time activation”.
startDate	String	Start date for “Daily activation”.
endDate	String	End date for “Daily activation”.
isExpired	Boolean	If date is out of service period.
features	JSON	Contains license features list.
maxDeviceCount	String	How many devices is allowed to run FaceMe services.

detectionModel	JSON Array	Face detection model list, it would contain “DNN”
extractionModel	JSON Array	Face template extraction model list, it would contain “UH”, “UH3”, “VH”, “H1”, “H2” and “H3”
inputMode	JSON	Input Control
imageModeEnabled	Boolean	Constraint number of frames into to FaceMeService.
imageModeFps	Integer	Maximum frames per second can input into FaceMeService.
databaseConstraint	JSON	Database constraint list
maxPeopleEnabled	Boolean	Constraint maximum database size.
maxPeopleCount	Integer	Number of people.
qualityCheck	JSON Array	Image quality check options.
faceSize	Boolean	Check if the face is too small.
occlusion	Boolean	Check if the face is occluded.
lighting	Boolean	Check if the photo is under exposure.
blurriness	Boolean	Check if the photo is blurred.
grayScale	Boolean	Check if the image is grayscale.
faceAngle	Boolean	Check face angle.

Example

# Request
```
curl -X POST http://localhost:8080/mp/api/v1.0/license/renew \
     --user cyberlink:377339aa107ae58abe8e3c7fd30218b6
```

# Response
```
HTTP/1.1 200 OK
{
    "license": {
        "activationType": "Daily activation",
        "startDate": "2020-04-01",
        "endDate": "2021-03-31",
        "isExpired": false
    },
    "features": {
        "maxDeviceCount": "1",
        "detectionModel": [ "DNN" ],
        "extractionModel: [ "UH", "UH3", "VH", "H3" ],
        "inputMode": {
            "imageModeEnabled": true,
            "imageModeFps": 96
        },
        "databaseConstraint": {
            "maxPeopleEnabled": true,
            "maxPeopleCount": 1000
        },
        "qualityCheck": {
            "faceSize": true,
            "occlusion": true,
            "lighting": false,
            "blurriness": true,
            "grayscale": false,
            "faceAngle": true,
        }
    }
}
```

1.13. Deactivate License

Deactivate API key.

This API requires internet connection.

After deactivated, you need to use **Chapter 1.16 - Register License** to input new API key.

For security concern this API only can access from localhost (127.0.0.1).

### Request

HTTP request

POST http://app-server/mp/api/v1.0/license/deactivate

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

### Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error
- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method
- 406 Not Acceptable: Invalid parameters.

Example

# Request

```
curl -X POST http://localhost:8080/mp/api/v1.0/license/deactivate \
     --user cyberlink:377339aa107ae58abe8e3c7fd30218b6
```

# Response

```
HTTP/1.1 200 OK
```

1.14. Check License Expiration

Check license expiration.

This API requires internet connection for daily activation type.

For security concern this API only can access from localhost (127.0.0.1).

### Request

HTTP request

POST http://app-server/mp/api/v1.0/license/checkExpire

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error
- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method
- 406 Not Acceptable: Invalid parameters.

Result

Property Name	Type	Description
license	JSON	Contains license date information.
activationType	String	License activation type. It would be "Daily activation" and "First time activation".
expireDate	String	Expire date for "First time activation"
deadline	String	Deadline date for "First time activation".
startDate	String	Start date for "Daily activation".
endDate	String	End date for "Daily activation".
isExpired	Boolean	If date is out of service period.

Example

# Request

```
curl -X POST http://localhost:8080/mp/api/v1.0/license/checkExpire \
     --user cyberlink:377339aa107ae58abe8e3c7fd30218b6
```

# Response

```
HTTP/1.1 200 OK
{
    "license": {
        "activationType": "Daily activation",
        "startDate": "2020-04-01",
        "endDate": "2021-03-31",
        "isExpired": false
    }
}
```

1.15. Query License Info

Query license information.

For security concern this API only can access from localhost (127.0.0.1).

Request

HTTP request

```
POST http://app-server/mp/api/v1.0/license/info
```

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error
- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method
- 406 Not Acceptable: Invalid parameters.

Result

Property Name	Type	Description
license	JSON	Contains license date information.
activationType	String	License activation type. It would be “Daily activation” and “First time activation”.
expireDate	String	Expire date for “First time activation”
deadline	String	Deadline date for “First time activation”.
startDate	String	Start date for “Daily activation”.
endDate	String	End date for “Daily activation”.
isExpired	Boolean	If date is out of service period.
features	JSON	Contains license features list.
maxDeviceCount	String	How many devices is allowed to run FaceMe services.
detectionModel	JSON Array	Face detection model list, it would contain

		"DNN"
extractionModel	JSON Array	Face template extraction model list, it would contain "UH", "UH3", "VH", "H1", "H2" and "H3"
inputMode	JSON	Input Control
imageModeEnabled	Boolean	Constraint number of frames into to FaceMeService.
imageModeFps	Integer	Maximum frames per second can input into FaceMeService.
databaseConstraint	JSON	Database constraint list
maxPeopleEnabled	Boolean	Constraint maximum database size.
maxPeopleCount	Integer	Number of people.
qualityCheck	JSON Array	Image quality check options.
faceSize	Boolean	Check if the face is too small.
occlusion	Boolean	Check if the face is occluded.
lighting	Boolean	Check if the photo is under exposure.
blurriness	Boolean	Check if the photo is blurred.
grayScale	Boolean	Check if the image is grayscale.
faceAngle	Boolean	Check face angle.

Example

# Request
```
curl -X POST http://localhost:8080/mp/api/v1.0/license/info \
     --user cyberlink:377339aa107ae58abe8e3c7fd30218b6
```

# Response
```
HTTP/1.1 200 OK
{
    "license": {
        "activationType": "Daily activation",
        "startDate": "2020-04-01",
        "endDate": "2021-03-31",
        "isExpired": false
    },
    "features": {
        "maxDeviceCount": "1",
        "detectionModel": [ "DNN" ],
        "extractionModel: [ "UH", "UH3", "VH", "H3" ],
        "inputMode": {
            "imageModeEnabled": true,
            "imageModeFps": 96
        },
        "databaseConstraint": {
            "maxPeopleEnabled": true,
            "maxPeopleCount": 1000
        },
        "qualityCheck": {
            "faceSize": true,
            "occlusion": true,
            "lighting": false,
            "blurriness": true,
            "grayscale": false,
            "faceAngle": true,
        }
    }
}
```

1.16. Register License

Register license with API key.

This API requires internet connection.

For security concern this API only can access from localhost (127.0.0.1).

Request

HTTP request

```
POST http://app-server/mp/api/v1.0/license/register
```

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

Request body

Parameter Name	Type	Description
licenseKey	String	License key.

Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error
- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method
- 406 Not Acceptable: Invalid parameters.

Result

Property Name	Type	Description
license	JSON	Contains license date information.
activationType	String	License activation type. It would be “Daily activation” and “First time activation”.
expireDate	String	Expire date for “First time activation”
deadline	String	Deadline date for “First time activation”.
startDate	String	Start date for “Daily activation”.
endDate	String	End date for “Daily activation”.

isExpired	Boolean	If date is out of service period.

Example

# Request

```
curl -X POST http://localhost:8080/mp/api/v1.0/license/register \
     --user cyberlink:377339aa107ae58abe8e3c7fd30218b6 \
     -F 'licenseKey=REPLACE_WITH_YOUR_LICENSE_KEY'
```

# Response

```
HTTP/1.1 200 OK
{
    "license": {
        "activationType": "Daily activation",
        "startDate": "2020-04-01",
        "endDate": "2021-03-31",
        "isExpired": false
    }
}
```

1.17. Setup database connection

Setup database configuration with IP, port, account and password.

For security concern this API only can access from localhost (127.0.0.1).

### Request

HTTP request

POST http://app-server/mp/api/v1.0/setup/database/connection/update

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

Request body

Parameter Name	Type	Description
databaseIp	String	Database IP.
databasePort	String	Database port.
databaseType	Integer	0: Microsoft SQL Server 1: MySQL
databaseAccount	String	Database account.
databasePassword	String	Database password.

Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error
- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method
- 406 Not Acceptable: Invalid parameters.

Example

# Request

```
curl -X POST
http://localhost:8080/mp/api/v1.0/setup/database/connection/update \
     --user cyberlink:377339aa107ae58abe8e3c7fd30218b6 \
     -F 'databaseIp=127.0.0.1' \
     -F 'databasePort=1433' \
     -F 'databaseType=0' \
     -F 'databaseAccount=faceme' \
     -F 'databasePassword=12345678'
```

# Response

```
HTTP/1.1 200 OK
```

1.18. Check setup status

Check setup status.

This API requires internet connection.

For security concern this API only can access from localhost (127.0.0.1).

### Request

HTTP request

`POST http://app-server/mp/api/v1.0/setup/check`

Authorization

This request requires HTTP basic authentication with access code provided by CyberLink.

### Response

HTTP Status Codes

- 200 OK : The request has succeeded
- 400 Bad Request: Unable to process due to client error
- 401 Not Authorized: Lack of valid authentication credentials
- 405 Method Not Allow: Only support POST by this method
- 406 Not Acceptable: Invalid parameters.

Result

Property Name	Type	Description
license	Boolean	If FaceMe SDK license registered successfully.
license_failure_reason	String	Optional. Reason about license failure.
Database	Boolean	If database schema and tables established.
database_failure_reason	String	Optional. Reason about database failure.

Example

# Request

```
curl -X POST http://localhost:8080/mp/api/v1.0/setup/check \
     --user cyberlink:377339aa107ae58abe8e3c7fd30218b6
```

# Response

```
HTTP/1.1 200 OK
```

```
{
    "license": true,
    "database": true
}
```

2. Database Schema

Table: face_info

Column Name	Data Type	Allow Nulls
face_id	int, Primary Key, IDENTIY(1, 1)	X
create_date	date_time	V
collection_id	int	X
outer_id	vchar(100), UNIQUE	X
face_feature	binary (2064)	X
extra_data	binary (2048)	V
feature_size	int	X
feature_type	int	V
feature_sub_type	int	V

Table: modified_item

Column Name	Data Type	Allow Nulls
id	int, Primary Key, IDENTIY(1, 1)	X
create_date	date_time	V
action	int	X
item_id	int	X
table_index	int	X

3. Snapshot Setting

Snapshot is the file cache for fast search to increase speed of initialization

Setting in

Redhat ：

/var/opt/faceme/config.yaml

Windows：

C:\ProgramData\CyberLink\FaceMeSDK\FaceMeSdkHttpApi\config.yaml

```
database:
   server: 127.0.0.1
   username: faceme
   password: 12345678
   dbname: FaceMeServerDB

index:
   addr: 0.0.0.0
   port: 6666
   snapshot: '0 0 15 * *'
```

The value of snapshot means 'minute hour mday month wday'

For example: '0 0 15 * *' means fast search service will save snapshot at 0:00 on 15th every month.