Aprilage

April Web API 2.0

Table of Contents

1. Overview

This document describes the design and usage patterns of April Web API 2.0, from now on referred to simply as April API 2.0 or the API.

April API 2.0 is a RESTful Web Service hosted in the cloud that supports the operation of the new Age-Me (presently, AgeMe) Website and enables programmatic access into the April Aging Infrastructure. This API enables the creation of social and mobile apps, ad-hoc solutions for corporate users.

1.1 What's new?

The initial version of the April API introduced only a small, yet effective set of operations an API client could perform. All of these operations focused on aging functionality and obtaining results. In contrast, April API 2.0 implements a wide range of features, which can be classified under the following groups (please consult Figure 2—"Map of April API 2.0" for more detail):

- Account (e.g., create user, retrieve user info, change user's password)
- Images (e.g., upload image, download image)
- Aging (e.g., create aging document, initiate matching, retrieve aging results)

In addition, the new version of the API has introduced enhanced security mechanisms for accessing the API, including granular permissions and guest access. These mechanisms are described in detail further in this section of the document. Another important change in the second iteration of the API is the shift toward resource-oriented Web Service architecture, as opposed to the verb-oriented architecture previously employed. A quick overview of resource-oriented Web Services can be found at the IBM® developerWorks® portal[1].

1.2 Compatibility with April API 1.0

Because of the radical changes in the URL paths and JSON data schemata, it appeared to be infeasible to maintain compatibility with April API 1.0. The initial version of the API still does exist, but it has been deprecated in favor of the new, more extensive, feature-rich, and technologically beautiful April API 2.0. Currently, the build process is configured to generate April API 2.0 Web archive by default; in order to generate a Web archive for April API 1.0, minor modifications will be required in web.xml.

1.3 Aging Process

While the account-related functionality is common to most Web sites and APIs and should be trivial to comprehend, the operations comprising the aging process can be challenging to wrap one's head around. Below you will find a brief overview of this peculiar process.

[1] http://www.ibm.com/developerworks/webservices/library/ws-restful/

The aging process begins with the user submitting an image of a face and concludes with the user viewing a generated sequence of aged images with various lifestyle effects applied to them. What happens in between is the curious part, comprised of three sub-processes. First sub-process is *feature point detection*, which is the process of identifying a human face in an image with a number of points describing the face's prominent parts. Second sub-process is *2D-to-3D matching*, which is the process of overlaying a two-dimensional face from the uploaded image onto a three-dimensional reference face model. Finally, the third sub-process is the actual *aging* of the face in the image. Please note that the order of these three sub-processes must always be maintained. Once an image has been matched, it can be aged multiple times.

All three processes mentioned above are computationally intensive and can only be performed in an asynchronous mode. The status of each operation can be obtained by retrieving the status of the aging document in question. The illustration below demonstrates a typical aging process.

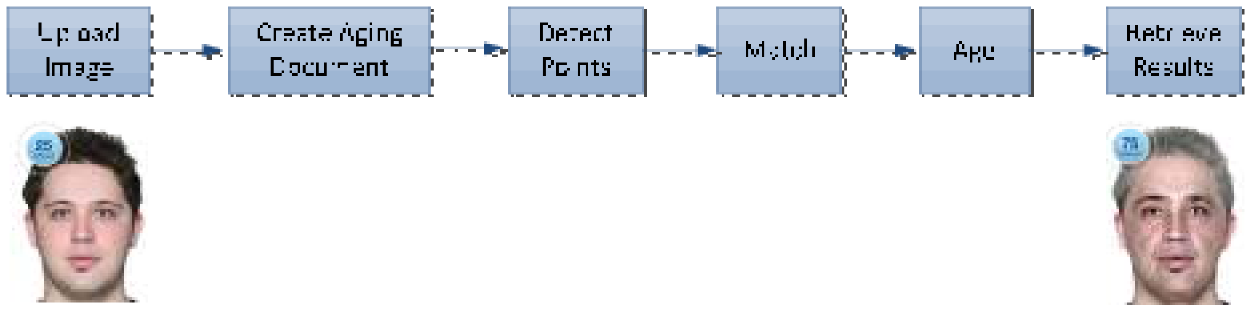

Figure 1. The aging process.

The last point that is crucial for a basic understanding of the aging API is the notion of an *aging document*. An aging document is a collection of meta-data describing the person depicted in the image that has been previously uploaded (and referenced in that document). This meta-data is comprised of the person's name and physical (height, weight) and demographic (age, gender, ethnicity) characteristics. An aging document will have aging results associated with it, which in turn will contain one or more aging sequences. Eventually, aging sequences will encapsulate aged images. Thus, the hierarchy of this information model can be described as follows: Aging Document => Aging Results => Aging Sequences => Aged Images. As you will observe, the API respects this hierarchy and makes logical use of it. For example, you will be able to retrieve aging results both per aging document and per user.

1.4 Authentication

In order to access April API 2.0, the user must be authenticated, with an exception of a number of publicly available methods—such as accessing an image, shared aging result, creating a new user, and several others. The permissions hierarchy consists of the following roles, listed from highest to lowest: *Admin*, *User*, *Guest*, and *Anonymous*. Anonymous simply signifies a user who is not authenticated. It must be well noted that a role includes access to all resources available to roles standing lower in the hierarchy.

April API 2.0 offers two strategies to authenticating a user. First strategy is HTTP Basic Access Authentication[2], which is carried out by including HTTP header `Authorization` in all requests to the API. This authentication strategy is best suited for fully stateless API clients, such as mobile and social apps. An important aspect to keep in mind with relation to the Basic Authentication is that it should only be used together with HTTPS, as the credentials are transmitted as base64-encoded plaintext. Below is an example of the header field:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

The text after the word "Basic" is a base64 encoding of "username:password".

The second authentication strategy is Form-based Authentication, which breaks the RESTful premise of the API, but it does simplify the life of Web app developers by providing a notion of state through maintaining server-side sessions. Most importantly, this was a requirement of the new Age-Me Website. Using form-based authentication, the user has an ability to login, logout, and enjoy the convenience of the "remember-me" feature. It must be taken into account that this strategy relies heavily on cookies for identifying the user within a session and, also, when remember-me is selected. Below is an example of the login and logout requests:

```
POST /login HTTP/1.1
Host: ageme.com/AprilAPI
Content-Type: application/x-www-form-urlencoded
Content-Length: 53

email=joe@example.com&password=secret&rememberMe=true
```

```
POST /logout HTTP/1.1
Host: ageme.com/AprilAPI
```

The cookies that are set after a successful login request are named JSESSIONID and REMEMBER_ME. The session times out in 30 minutes, given no requests are received by the API server in that period of time.

1.5 Common Responses

Most API calls return a specific JSON representation of the resource being requested. However, when an error occurs or when a simple descriptive message is returned, the response format is uniform (further called Uniform Response), as shown in the example below:

[2] http://en.wikipedia.org/wiki/Basic_access_authentication

```
{
    "result_code": 0,
    "message": "Unauthorized"
}
```

Parameter	Scope	Type	Description
result_code	Required	Number	Result code, signifying whether an operation was successful (1) or not (0). In future, other error codes may be supported to communicate a more specific error.
message	Optional	String	Either an error message or descriptive message explaining the outcome of the request.

1.6 Testing Guidelines

In order to carry out testing of the API, one needs a RESTful client to perform the actual HTTP requests. Personal recommendation of the author of this document is restclient-tool[3]; however, to much regret, this little tool only supports 32-bit JVMs. It must be noted, though, that any software capable of issuing HTTP requests with content-type of multipart/form-data will be suitable. For example, one can choose to use a browser plugin instead of a standalone software program.

In addition to a RESTful client, one will need a high-quality passport photo for testing the aging process. Also, one will require the URL of the API endpoint available for testing purposes; at the time of this writing this URL is http://api-stage.age-me.com/AprilAPI.

[3] http://code.google.com/a/eclipselabs.org/p/restclient-tool/

1.7 Map of April API 2.0

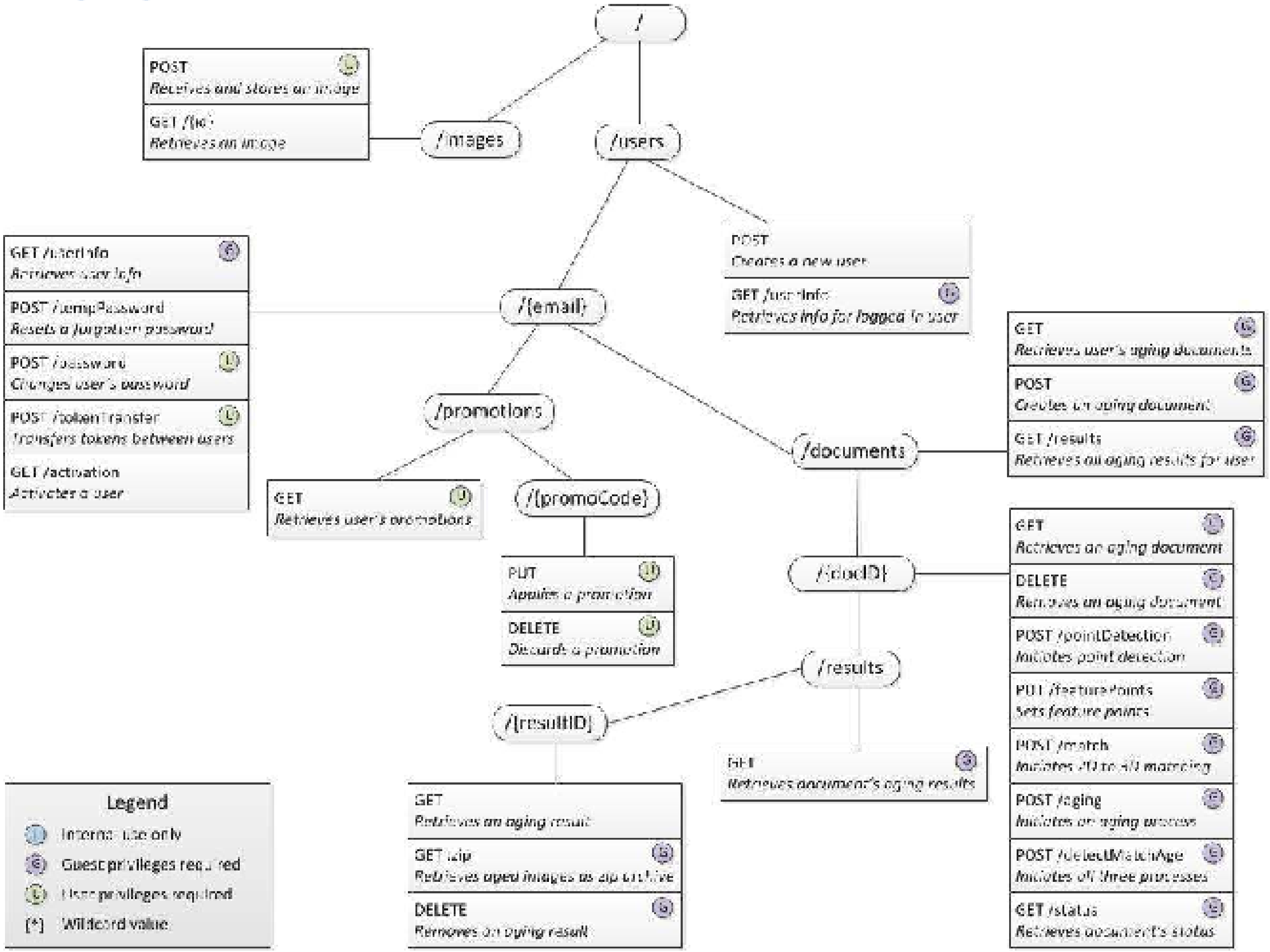

Figure 2. Map of April API 2.0.

2. April API 2.0 Methods

This section of the document describes each API method in detail by providing its request and response formats, required authorization level, and possible error messages. For convenience of the reader, all methods are divided into several categories. Certain paths include a segment enclosed in curly braces (e.g., /users/{email}); such a segment denotes a wildcard substituted at runtime for the actual value (e.g., /users/joe@example.com). JSON representations are described in Appendix A. All `GET` methods communicate parameters via the query string.

2.1 Account Methods

2.1.1 Create User

Creates a new user in the API and serves as a conceptual synonym to registration. This method allows creation of both regular users and guest users. Guest users are temporary and are deleted one week after creation. To create a guest user, only *email* and *isGuest=true* are required. To transform a guest user to a regular user, one should issue a new request to this method with the same email and other relevant parameters.

Path: /users
HTTP Method: `POST`
Content-Type: `application/x-www-form-urlencoded`
Authorization: N/A
Parameters:

Parameter	Scope	Type	Description
email	Required	String	User's email in valid format
password	Required, for regular users	String	User's password, minimum six symbols long
termsAccepted	Required, for regular users	Boolean	Indicates whether a user agrees to the terms and condition of the service. Can only be `true`.
promoCode	Optional	String	Promotional code
isGuest	Optional	Boolean	Indicates whether the request is for creation of a guest user or not. If not provided, defaults to `false`.

Errors:

HTTP Status Code	Condition
400	If a validation error occurs, if a promotion code cannot be applied, if activation email cannot be sent (only for regular user creation)
409	If a regular user with such email already exists (guest users overwritten)

Successful Response:

HTTP Status Code	Content-Type	Format
201	`application/json`	Uniform Response

2.1.2 Retrieve User Info

Retrieves a summary of the user information. This method is accessible at two paths: /users/userInfo and /users/{email}/userInfo. The former variant is a convenience shortcut.

Path: /users/userInfo and /users/{email}/userInfo

HTTP Method: `GET`

Content-Type: N/A

Authorization: `Guest`, `User`, `Admin`

Parameters: N/A

Errors:

HTTP Status Code	Condition
401	If user is not authorized to access this method
404	If user is not found

Successful Response:

HTTP Status Code	Content-Type	Format
200	`application/json`	User Info Representation

2.1.3 Reset Password

Resets a forgotten password for a user and sends an e-mail containing the temporary password.

Path: /users/{email}/tempPassword

HTTP Method: `POST`

Content-Type: `application/x-www-form-urlencoded`

Authorization: N/A

Parameters: N/A

Errors:

HTTP Status Code	Condition
400	If a validation error occurs, if no user with such email exists

Successful Response:

HTTP Status Code	Content-Type	Format
200	`application/json`	Uniform Response

2.1.4 Change Password

Changes a user's password. Please note that remember-me authentication does not grant access to this method.

Path: /users/{email}/password

HTTP Method: `POST`

Content-Type: `application/x-www-form-urlencoded`

Authorization: `User, Admin`

Parameters:

Parameter	Scope	Type	Description
oldPassword	Required	String	User's current password
newPassword	Required	String	New password, minimum six symbols long

Errors:

HTTP Status Code	Condition
400	If a validation error occurs, if no user with such email exists
401	If user is not authorized to access this method

Successful Response:

HTTP Status Code	Content-Type	Format
200	`application/json`	Uniform Response

2.1.5 Transfer Tokens

Transfers tokens between users. Once a token transfer is completed, a new promotion will be created and added to the recipient user. The recipient user will then have liberty to either apply this promotion or discard it.

Path: /users/{email}/tokenTransfer

HTTP Method: `POST`

Content-Type: `application/x-www-form-urlencoded`

Authorization: `User, Admin`

Parameters:

Parameter	Scope	Type	Description
emailFrom	Required	String	Sender's email
emailTo	Required	String	Recipient's email
tokens	Required	Number	Number of tokens to be transferred

Errors:

HTTP Status Code	Condition
400	If a validation error occurs, if no user with such email exists
401	If user is not authorized to access this method

Successful Response:

HTTP Status Code	Content-Type	Format
200	`application/json`	Uniform Response

2.1.6 Retrieve Promotions

Retrieves currently available promotions for a user. Often a promotion represents a pending token transfer to be accepted or declined by the recipient.

Path: /users/{email}/promotions

HTTP Method: `GET`

Content-Type: N/A

Authorization: `User, Admin`

Parameters: N/A

Errors:

HTTP Status Code	Condition
401	If user is not authorized to access this method

Successful Response:

HTTP Status Code	Content-Type	Format
200	`application/json`	Array of Promotion Representations

2.1.7 Apply Promotion

Applies a promotion to the user's account.

Path: /users/{email}/promotions/{promoCode}

HTTP Method: `PUT`

Content-Type: N/A

Authorization: `User, Admin`

Parameters: N/A

Errors:

HTTP Status Code	Condition
400	If a validation error occurs
401	If user is not authorized to access this method

Successful Response:

HTTP Status Code	Content-Type	Format
200	`application/json`	Uniform Response

2.1.8 Discard Promotion

Discards a promotion and removes it from the user's account.

Path: /users/{email}/promotions/{promoCode}

HTTP Method: `DELETE`

Content-Type: N/A

Authorization: `User, Admin`

Parameters: N/A

Errors:

HTTP Status Code	Condition
400	If a validation error occurs
401	If user is not authorized to access this method

Successful Response:

HTTP Status Code	Content-Type	Format
200	`N/A`	N/A

2.1.9 Activate User

Activates a user within the system. This method is provided to support user activation emails (thus the usage of `GET`, and not `POST`).

Path: /users/{email}/activation
HTTP Method: `GET`
Content-Type: N/A
Authorization: N/A
Parameters:

Parameter	Scope	Type	Description
accessKey	Required	String	Access key that is used to activate the user

Errors:

HTTP Status Code	Condition
400	If a validation error occurs

Successful Response:

HTTP Status Code	Content-Type	Format
200	`application/json`	Uniform Response

2.2 Image Methods

2.2.1 Upload Image (Multipart)

Receives an image from the user and stores it in the system. This method uses content-type of `multipart/form-data` for transferring the image and its metadata. Please note that the metadata must include the image's file name.

Path: /images
HTTP Method: `POST`
Content-Type: `multipart/form-data`
Authorization: `User`, `Admin`
Parameters:

Parameter	Scope	Type	Description
image	Required	Binary	Image to be received and stored. Currently only JPEG images are supported with the minimum resolution of 128x128 pixels.

Errors:

HTTP Status Code	Condition
400	If a validation error occurs
401	If user is not authorized to access this method

Successful Response:

HTTP Status Code	Content-Type	Format
201	`application/json` `text/html` (for MSIE user-agent)	Image Representation

2.2.2 Upload Image (Octet-Stream)

Receives an image from the user and stores it in the system. This method uses content-type of `application/octet-stream` for transferring the image and header field `X-File-Name` for communicating the image's file name. The content-type can, as well, be set to any other value, but `multipart/form-data` (e.g., `image/jpeg`).

Path: /images
HTTP Method: `POST`
Content-Type: `application/octet-stream, image/jpeg, image/*`
Authorization: `User`, `Admin`
Parameters:

Parameter	Scope	Type	Description
N/A	Required	Binary	Image to be received and stored. Currently only JPEG images are supported with the minimum resolution of 128x128 pixels.

Errors:

HTTP Status Code	Condition
400	If a validation error occurs
401	If user is not authorized to access this method

Successful Response:

HTTP Status Code	Content-Type	Format
201	`application/json` `text/html` (for MSIE user-agent)	Image Representation

2.2.3 Retrieve Image

Retrieves an image, optionally resizing it on the fly. This method may return either actual binary stream or a redirect to the image stored in an Amazon S3 bucket. In case the image appears to be registered in the system, but it cannot be retrieved, a default JPEG image will be returned with text "Missing Image" written in the middle of it.

Path: /images/{id}
HTTP Method: `GET`

Content-Type: N/A
Authorization: N/A
Parameters:

Parameter	Scope	Type	Description
width	Optional, if height is not provided	Number	Resized width of the image
height	Optional, if width is not provided	Number	Resized height of the image
padded	Optional	Boolean	Indicates whether the image should be padded when resized or not. If not provided, it will be defaulted to `false`.

Errors:

HTTP Status Code	Condition
400	If a validation error occurs
401	If user is not authorized to access this method
404	If image is not found

Successful Response:

HTTP Status Code	Content-Type	Format
200	`image/jpeg, image/*`	Binary
307	N/A	N/A

2.3 Aging Methods

2.3.1 Create Aging Document

Creates a new aging document for future processing using the provided parameters. An aging document represents meta-data associated with a certain image. This method can optionally perform asynchronous feature point detection.

Path: /users/{email}/documents
HTTP Method: `POST`
Content-Type: `application/json`
Authorization: `Guest`, `User`, `Admin`
Parameters: New Document Request Representation
Errors:

HTTP Status Code	Condition
400	If a validation error occurs, if image is not found, if user does not have any tokens (for non-sample agings)
401	If user is not authorized to access this method

403	If a guest user attempts to create an aging document with non-sample image

Successful Response:

HTTP Status Code	Content-Type	Format
201	`application/json`	Aging Document Representation

2.3.2 Retrieve Aging Documents

Retrieves a collection of user's aging documents. The collection can optionally be paginated, filtered, and sorted. If no pagination parameters are provided, the maximum number of returned aging documents is limited to one hundred.

Path: /users/{email}/documents

HTTP Method: `GET`

Content-Type: N/A

Authorization: `Guest`, `User`, `Admin`

Parameters:

Parameter	Scope	Type	Description
pageNumber	Optional, if pageSize is not provided	Number	Page number, starting at 1
pageSize	Optional, if pageNumber is not provided	Number	Number of items per page
orderBy	Optional	String	Field to be used for ascending ordering. Allowed values are: `status`, `age`, `ethnicity`, `gender`, and `name`.
filter	Optional	String	Expression to be used for filtering of aging documents. Filtering expression has the form of `field[<|=|>]value`. For instance, `age>8`. Allowed filtering fields are: `status`, `age`, `ethnicity`, `gender`, and `name`. Multiple filters are not supported.

Errors:

HTTP Status Code	Condition
400	If a validation error occurs
401	If user is not authorized to access this method

Successful Response:

HTTP Status Code	Content-Type	Format
200	`application/json`	Paginated Result Representation (`items` are Aging Document Representations)

2.3.3 Retrieve Aging Results

Retrieves a collection of user's aging results. Each aging result may contain multiple sequences of aged images and is guaranteed to contain the relevant aging document representation. The collection can optionally be paginated, filtered, and sorted. If no pagination parameters are provided, the maximum number of returned aging results is limited to one hundred.

Path: /users/{email}/documents/results

HTTP Method: `GET`

Content-Type: N/A

Authorization: `Guest`, `User`, `Admin`

Parameters:

Parameter	Scope	Type	Description
pageNumber	Optional, if pageSize is not provided	Number	Page number, starting at 1
pageSize	Optional, if pageNumber is not provided	Number	Number of items per page
orderBy	Optional	String	Field to be used for ascending ordering. Allowed values are: `status` and `sequenceType`.
filter	Optional	String	Expression to be used for filtering of aging results. Filtering expression has the form of `field[<|=|>]value`. For instance, `smoking>0.7`. Allowed filtering fields are: `status`, `smoking`, `sunExposure`, `bmi`, `multiplier`, `bmifunc`, and `sequenceType`. Multiple filters are not supported.

Errors:

HTTP Status Code	Condition
400	If a validation error occurs
401	If user is not authorized to access this method

Successful Response:

HTTP Status Code	Content-Type	Format
200	`application/json`	Paginated Result Representation (`items` are Aging Result Representations)

2.3.4 Retrieve Aging Document

Retrieves an aging document for a user.

Path: /users/{email}/documents/{docID}
HTTP Method: `GET`
Content-Type: N/A
Authorization: `Guest`, `User`, `Admin`
Parameters: N/A
Errors:

HTTP Status Code	Condition
401	If user is not authorized to access this method
403	If aging document does not belong to the user
404	If aging document is not found

Successful Response:

HTTP Status Code	Content-Type	Format
200	`application/json`	Aging Document Representation

2.3.5 Remove Aging Document

Removes an aging document along with all associated aging results.

Path: /users/{email}/documents/{docID}
HTTP Method: `DELETE`
Content-Type: N/A
Authorization: `Guest`, `User`, `Admin`
Parameters: N/A
Errors:

HTTP Status Code	Condition
401	If user is not authorized to access this method
403	If aging document does not belong to the user
404	If aging document is not found

Successful Response:

HTTP Status Code	Content-Type	Format
200	N/A	N/A

2.3.6 Detect Feature Points

Performs automatic feature point detection on the image referenced by the aging document in question. This is an asynchronous process and, thus, it will return immediately. The status of the point detection can be obtained by sending a `GET` request to /users/{email}/documents/{docID}/status.

Path: /users/{email}/documents/{docID}/pointDetection

HTTP Method: `POST`

Content-Type: N/A

Authorization: `Guest`, `User`, `Admin`

Parameters: N/A

Errors:

HTTP Status Code	Condition
401	If user is not authorized to access this method
403	If aging document does not belong to the user
404	If aging document is not found

Successful Response:

HTTP Status Code	Content-Type	Format
202	N/A	N/A

2.3.7 Set Feature Points

Applies feature points provided by the user as input parameters to the image referenced by the aging document in question. This method should be used when the automatic point detection fails or when the user expects it to fail due to poor quality of the image. Even though the process is normally instantaneous, it is still an asynchronous process. The status of the process can be obtained by sending a `GET` request to /users/{email}/documents/{docID}/status.

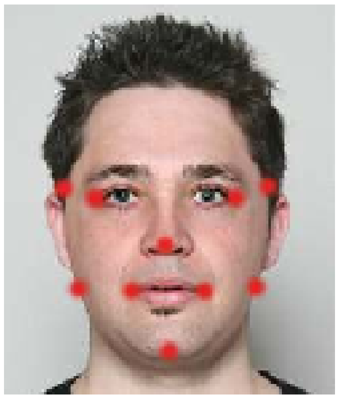

Figure 3. Location of feature points on a person's face.

Path: /users/{email}/documents/{docID}/featurePoints

HTTP Method: `PUT`

Content-Type: `application/json`

Authorization: `Guest`, `User`, `Admin`

Parameters: Feature Points Representation

Errors:

HTTP Status Code	Condition
400	If a validation error occurs
401	If user is not authorized to access this method

403	If aging document does not belong to the user
404	If aging document is not found

Successful Response:

HTTP Status Code	Content-Type	Format
202	N/A	N/A

2.3.8 Match Image

Performs 2D-to-3D matching of the image referenced by the aging document in question. This is an asynchronous process and, thus, it will return immediately. It is expected that with an average load the matching will take from 30 to 90 seconds. The status of the matching process can be obtained by sending a `GET` request to /users/{email}/documents/{docID}/status.

Path: /users/{email}/documents/{docID}/match
HTTP Method: `POST`
Content-Type: N/A
Authorization: `Guest`, `User`, `Admin`
Parameters: N/A
Errors:

HTTP Status Code	Condition
400	If called before feature points are set/detected
401	If user is not authorized to access this method
403	If aging document does not belong to the user
404	If aging document is not found

Successful Response:

HTTP Status Code	Content-Type	Format
202	N/A	N/A

2.3.9 Age Image

Performs an aging of the image referenced by the aging document in question. This method can be called multiple times with different sequence parameters for the same matched aging document. This is an asynchronous process and, thus, it will return immediately. It is expected that with an average load an aging will take from 15 to 45 seconds. The status of the process can be obtained by sending a `GET` request to /users/{email}/documents/{docID}/status.

Path: /users/{email}/documents/{docID}/aging
HTTP Method: `POST`
Content-Type: `application/json`
Authorization: `Guest`, `User`, `Admin`
Parameters: Aging Request Representation

Errors:

HTTP Status Code	Condition
400	If a validation error occurs, If called before matching has succeeded
401	If user is not authorized to access this method
403	If aging document does not belong to the user
404	If aging document is not found

Successful Response:

HTTP Status Code	Content-Type	Format
202	N/A	N/A

2.3.10 Detect Feature Points + Match Image + Age Image

Performs a chained set of all three image processing routines (automatic feature point detection, 2D-to-3D matching, and aging) on the image referenced by the aging document in question. This method is provided for convenience of stateful API clients. This is an asynchronous process and, thus, it will return immediately. It is expected that with an average load this set of processes will take from 60 to 120 seconds. The status of the aging document can be obtained by sending a `GET` request to /users/{email}/documents/{docID}/status.

Path: /users/{email}/documents/{docID}/detectMatchAge
HTTP Method: `POST`
Content-Type: `application/json`
Authorization: `Guest`, `User`, `Admin`
Parameters: Aging Request Representation
Errors:

HTTP Status Code	Condition
400	If a validation error occurs
401	If user is not authorized to access this method
403	If aging document does not belong to the user
404	If aging document is not found

Successful Response:

HTTP Status Code	Content-Type	Format
202	N/A	N/A

2.3.11 Retrieve Status of Aging Document

Retrieves the status of an aging document.

Path: /users/{email}/documents/{docID}/status
HTTP Method: `GET`
Content-Type: N/A
Authorization: `Guest`, `User`, `Admin`

Parameters: N/A

Errors:

HTTP Status Code	Condition
401	If user is not authorized to access this method
403	If aging document does not belong to the user
404	If aging document is not found

Successful Response:

HTTP Status Code	Content-Type	Format
200	`application/json`	Status Representation (See `status` in Aging Document Representation)

2.3.12 Retrieve Aging Results per Aging Document

Retrieves all aging results for an aging document.

Path: /users/{email}/documents/{docID}/results

HTTP Method: `GET`

Content-Type: N/A

Authorization: `Guest`, `User`, `Admin`

Parameters: N/A

Errors:

HTTP Status Code	Condition
401	If user is not authorized to access this method
403	If aging document does not belong to the user
404	If aging document is not found

Successful Response:

HTTP Status Code	Content-Type	Format
200	`application/json`	Array of Aging Result Representations

2.3.13 Retrieve Aging Result

Retrieves an aging result.

Path: /users/{email}/documents/{docID}/results/{resultID}

HTTP Method: `GET`

Content-Type: N/A

Authorization: N/A

Parameters: N/A

Errors:

HTTP Status Code	Condition
403	If aging result does not belong to the user
404	If aging result is not found

Successful Response:

HTTP Status Code	Content-Type	Format
200	`application/json`	Aging Result Representation

2.3.14 Retrieve Aging Result as Zip Archive

Retrieves an aging result as a Zip archive containing aged images for all relevant sequences.

Path: /users/{email}/documents/{docID}/results/{resultID}.zip

HTTP Method: `GET`

Content-Type: N/A

Authorization: `Guest`, `User`, `Admin`

Parameters: N/A

Errors:

HTTP Status Code	Condition
401	If user is not authorized to access this method
403	If aging result does not belong to the user
404	If aging result is not found

Successful Response:

HTTP Status Code	Content-Type	Format
200	`application/zip`	Zip Archive

2.3.15 Remove Aging Result

Removes an aging result.

Path: /users/{email}/documents/{docID}/results/{resultID}

HTTP Method: `DELETE`

Content-Type: N/A

Authorization: `Guest`, `User`, `Admin`

Parameters: N/A

Errors:

HTTP Status Code	Condition
401	If user is not authorized to access this method
403	If aging result does not belong to the user
404	If aging result is not found

Successful Response:

HTTP Status Code	Content-Type	Format
200	N/A	N/A

Appendix A: JSON Representations

User Info Representation

Field	Scope	Type	Description
uri	Required	String	URI of the user resource
email	Required	String	User's email
tokens	Required	Number	Amount of tokens
numOfAgings	Required	Number	Number of aging results
role	Required	String	User's role: `admin`, `user`, or `guest`

```
{
    "uri": "http://ageme.com/AprilAPI/users/joe@example.com",
    "email": "joe@example.com",
    "tokens": 3,
    "numOfAgings": 78,
    "role": "user"
}
```

Paginated Result Representation

Field	Scope	Type	Description
uri	Required	String	URI of this paginated result representation
pageNumber	Optional, if pageSize is not provided	Number	Page number, starting at 1
pageSize	Optional, if pageNumber is not provided	Number	Number of items per page
totalItems	Required	Number	Total number of items that meet the query conditions (filter)
orderBy	Optional	String	Soring order that was used for this result
filter	Optional	String	Filter that was used to retrieve this result
prev	Optional	String	URI for retrieval of the previous batch of results
next	Optional	String	URI for retrieval of the next batch of results
items	Required	Array of Objects	An array of items that meet the request criteria. These items are in the form of various representations, such as User Info, Document, or Aging Result.

```
{
    "uri": "http://ageme.com/AprilAPI/users/j@ex.com/documents?pageNum
ber=2&pageSize=10&orderBy=status&filter=age=8",
    "totalItems": 12,
    "pageNumber": 2,
    "pageSize": 10,
    "orderBy": "status",
    "filter": "age=8",
    "items": [...],
    "prev":
"http://ageme.com/AprilAPI/users/j@ex.com/documents?pageNumber=1&pageS
ize=10&orderBy=status&filter=age=8",
}
```

Promotion Representation

Field	Scope	Type	Description
uri	Required	String	URI of the promotion resource
sender	Required	String	Sender's email
tokens	Required	Number	Amount of tokens

```
{
    "uri":
"http://ageme.com/AprilAPI/users/joe@example.com/promotions/69",
    "sender": "joe@example.com",
    "tokens": 1
}
```

Image Representation

Field	Scope	Type	Description
uri	Required	String	URI of the image
id	Required	Number	ID of the image

```
{
    "uri": "http://ageme.com/AprilAPI/images/1j2rn9Ng",
    "id": 11209
}
```

New Document Request Representation

Field	Scope	Type	Description
imageId	Required, if imageUri is not provided	Number	ID of the previously uploaded image that will be aged
imageUri	Required, if imageId is not provided	String	URI of the previously uploaded image that will be aged

detectPoints	Optional	Boolean	Indicates whether automatic feature point detection should be performed or not. If not provided, will be default to `false`.
document	Required	Aging Document Object	Aging Document Representation that carries the actual meta-data payload to be used for future aging processing.

```
{
    "imageId": 12490
    "document": {
        "name": "Joe",
        "gender": "male",
        "age": 21,
        "ethnicity": "Caucasian"
    }
}
```

Aging Document Representation

Field	Scope	Type	Description
uri	Required, unless within New Document Request	String	URI of this aging document
id	Required, unless within New Document Request	Number	ID of this aging document
status	Required, unless within New Document Request	String	Status of this aging document. Values are: • `uploaded` • `point_detection_pending` • `points_set` • `point_detection_failed` • `match_pending` • `matched` • `match_failed` • `aging_pending` • `aging_complete` • `aging_failed`
isSample	Required, unless within a	Boolean	Indicates whether this aging document is a sample or not.

	New Document Request		
originalImage	Required, unless within a New Document Request	String	URI of the image referenced by this aging document. If original image was cropped, this URI will point to the cropped image.
name	Optional	String	Name of the person in the image referenced by this aging document
gender	Required	String	Gender of the person in the image referenced by this aging document. Accepted values are `male` and `female`.
age	Required	Number	Age of the person in the image referenced by this aging document. Minimum age is 6 and maximum age is 65.
ethnicity	Optional	String	Ethnicity of the person in the image referenced by this aging document. Accepted values are: `Caucasian`, `Asian`, `African`, `Latino-Hispanic`, and `South-Asian`. If not provided, defaults to `Caucasian`.
height	Optional, if weight is not provided	Number	Height of the person in the image referenced by this aging document. This is used for BMI calculation and is needed with weight. Height can either be in centimetres or inches. Measurement units are identified by field `measurement`. Minimum height is 92cm or 36in. Maximum height is 213 cm or 84in.
weight	Optional, if height is not provided	Number	Weight of the person in the image referenced by this aging document. This is used for BMI calculation and is needed with height. Weight can either be in kilograms or pounds. Measurement units are identified by field `measurement`. Minimum weight is 23kg or 50 lbs. Maximum weight is 136kg or 300lbs.
measurement	Optional	String	Unit of measurement. Accepted values are `metric` and `imperial`. If not provided, defaults to `imperial`.
bounds	Optional	Array of Numbers	Bounds of cropping rectangle around the face. Values are relative, in the following order: top,

			right, bottom, left. For instance, [0.131, 0.067, 0.845, 0.900].

```
{
    "uri": "http://ageme.com/AprilAPI/users/j@ex.com/documents/62",
    "id": 62,
    "status": "uploaded",
    "originalImage": " http://ageme.com/AprilAPI/images/sample_ben",
    "name": "Ben",
    "gender": "male",
    "age": 15,
    "ethnicity": "Caucasian",
    "isSample": true
}
```

Aging Result Representation

Field	Scope	Type	Description
uri	Required	String	URI of this aging result
status	Required	String	Status of this aging result. Possible values are: `not_done`, `done`, and `failed`.
sequenceType	Required	String	Sequence type identifies a predefined sequence of ages, which can be fixed or relative to the starting age. Possible values are: `Max72` and `Rel40`. Additional sequence types can be defined on request.
sequences	Required	Array of Aging Sequence Objects	An array of aging sequences that specify various aging parameters. For completed results, aging sequences will contain aged image representations.
document	Optional	Aging Document Object	Aging Document that is associated with this aging result. This field will only be set when the aging result is retrieved per user, and not per document.

```
{
    "uri":
"http://ageme.com/AprilAPI/users/j@ex.com/documents/69/results/52101",
    "status": "done",
    "sequenceType": "Max72",
    "sequences": [{
        "smoking": 0,
        "sunExposure": 0,
        "multiplier": 1,
        "images": [{
            "age": 26,
            "uri": "http://ageme.com/AprilAPI/images/G6xqcnBt"
        }, ...]
    }]
}
```

Aging Request Representation

Field	Scope	Type	Description
sequenceType	Required	String	Sequence type identifies a predefined sequence of ages, which can be fixed or relative to the starting age. Possible values are: `Max72` and `Rel40`. Additional sequence types can be defined on request.
sequences	Required	Array of Aging Sequence Objects	An array of aging sequences that specify various aging parameters. Maximum number of submitted sequences is two.

```
{
    "sequenceType": "Max72",
    "sequences": [{
        "smoking": 0,
        "sunExposure": 0,
        "multiplier": 1
    }]
}
```

Aging Sequence Representation

Field	Scope	Type	Description
smoking	Optional	Number	Indicates to what extent smoking effects should be applied to the aged images. Values can range from 0.0 to 1.0. If not provided, default to 0.0.
sunExposure	Optional	Number	Indicates to what extent sun exposure effects should be applied to the aged images. Values can range from 0.0 to 1.0. If not provided, default to 0.0.
bmi	Optional	Number	Indicates to what extent weight gain effects should be applied to the aged images. BMI values can range from 16.0 to 39.0. If not provided, the person's initial BMI will not be changed.
bmifunc	Optional	String	BMI function that controls how the weight gain effects will be applied to the aged images.

			whether the weight gain will be constant through the aging sequence or it will increase linearly toward the BMI value. The accepted values are `linear` and `constant`. If not provided, the default is `linear`.
multiplier	Optional	Number	Aging multiplier that can be used to simulate lifestyles that cause aging to occur quicker. Values can range from 0.0 to 1.5. If not provided, the default is 1.0.
images	Optional	Array of Aged Image Objects	An array of aged images that can be accessed publicly using the provided URLs.

```
{
    "smoking": 0,
    "sunExposure": 0,
    "multiplier": 1
    "images": [{
        "age": 26,
        "uri": "http://ageme.com/AprilAPI/images/G6xqcnBt"
    }, ...]
}
```

Aged Image Representation

Field	Scope	Type	Description
uri	Required	String	URI of the aged image
age	Required	Number	Person's age that the image corresponds to

```
{
    "uri": "http://ageme.com/AprilAPI/images/lj2rn9Ng",
    "age": 72
}
```

Feature Points Representation

Field	Scope	Type	Description
templeLeft	Required	Feature Point Object	Point location of the left side of the face
templeRight	Required	Feature Point Object	Point location of the right side of the face
eyeLeft	Required	Feature Point Object	Point location of the left eye
eyeRight	Required	Feature Point Object	Point location of the right eye

nose	Required	Feature Point Object	Point location of the nose
mouthLeft	Required	Feature Point Object	Point location of the left side of the mouth
mouthRight	Required	Feature Point Object	Point location of the right side of the mouth
cheekLeft	Required	Feature Point Object	Point location of the left cheek
cheekRight	Required	Feature Point Object	Point location of the right cheek
chin	Required	Feature Point Object	Point location of the chin

```
{
    "templeLeft": {"x": 0.220, "y": 0.642},
    "templeRight": {"x": 0.765, "y": 0.646},
    "eyeLeft": {"x": 0.313, "y": 0.611},
    "eyeRight": {"x": 0.673, "y": 0.613},
    "nose": {"x": 0.486, "y": 0.484},
    "mouthLeft": {"x": 0.393, "y": 0.384},
    "mouthRight": {"x": 0.616, "y": 0.380},
    "cheekLeft": {"x": 0.254, "y": 0.354},
    "cheekRight": {"x": 0.751, "y": 0.374},
    "chin": {"x": 0.498, "y": 0.227}
}
```

Feature Point Representation

Field	Scope	Type	Description
x	Required	Number	X-coordinate of the feature point. The value is relative, starting from left.
y	Required	Number	Y-coordinate of the feature point. The value is relative, starting from bottom.

```
{
    "x": 0.751,
    "y": 0.374
}
```