User API Design Document

Version 0.3

Mind2Mobile

3 June 2019

Document Revision History			
Revision	Date	Description of Change	Originator
0.1	06/02/19	Created	Charles Stack
0.2	06/03/19	Initial Release	Charles Stack
0.3	06/09/19	Added the OAS_UserAPI.yaml reference	Charles Stack

Approvals		
Role	Approval	Date
Author		

Table of Contents

License and Copyright

The content of this document is published and licensed in accordance with the MIT License. The terms and conditions are outlined below:

Introduction

Document Purpose

This document describes the design and usage of the User Service API. The User Service provides a set of REST based APIs intended to provide a general-purpose service for managing the users of a more complex system.

Audience

This document is intended for use by Software Development and Software Testing Teams.

Related Documents

a) Swagger_UserAPI.yaml – A Swagger 2.0 document describing the User Management Service API.

b) OAS_UserAPI.yaml – An OAS 3.0 document describing the User Management Service API.

Conventions

N/A

Acronyms & Definitions

Term	Definition
JSON	Javascript Object Notation
MIT License	A permissive free software license originating at the Massachusetts Institute of Technology (MIT)
OTP	One-Time-Password. A unique code generated either by an application or device the user holds or sent to the user via an Out-Of-Band (OOB) method (such as SMS).
REST	Representational State Transfer is a software architecture style that defines a set of constraints to be followed when creating web services. Web services that confirm to the REST conventions are known as RESTful web services and provide interoperability with other systems on the Internet.
SWAGGER/OAS	An open-source software framework that helps developers, design, build, document and consume RESTful Web Services. See https://swagger.io/ There are two specifications: Swagger (Version 2.0) and Swagger 3.0 (now known as the Open API Specification (OAS).
UUID	Universally Unique Identifier
YAML	Yet Another Markup Language

Design Considerations

Technology Stack

The technology stack chosen is highly scalable and open-source. The Swagger and its success, Open API Standard (OAS), provide a mechanism to clearly define the API in a manner that is both human and machine readable.

Implementation Language

The User Management Service API is programming language and platform neutral. While Swagger/OAS is used to define the API, the API can be implemented using the developer's language and platform of choice. Swagger/OAS provides tools to generate server and client code for a variety of languages and is recommended only because of the tooling available.

Authentication

The User Management Service API requires that Two Factor Authentication be implemented in any concrete implementation of the API. It supports OTP via the Google Authenticator app as well as via an SMS message.

Database

The design is database agnostic. It is up to the development team implementing this API to select a persistent data storage mechanism.

Security

HTTPS is the preferred security for any public facing services. HTTP is permitted only during development.

Source and Version Control

The API source document and related documentation is maintained on GitHub.

User Management Service API Details

Overview

The User Management Service API centers around the concept of handling and managing a User of one or more systems. The four functional areas are:

- User Management
- User Access
- Password Management
- Account Security

User Management deals with the creation, updating, viewing, and closing a User account.

User Access is responsible for obtaining access (login), ending access (logout) and refreshing a user session to prevent logout.

Password Management is responsible for changing a User's password or resetting it in the event the user forgets their password.

Account Security is responsible for enabling or disabling Two Factor Authentication (TFA) on a User account.

Two One-Time-Password (OTP) methods are supported:

Google Authenticator – A OTP code is generated by an application supporting the Google Authenticator protocol.
SMS – A time-based code is sent to the user's registered cell phone.

User Roles and User Role Management are not handled by the API at this time. However, roles can be stored in the user tables or retrieved and inserted into a response.

Communication Protocol

Clients shall interact with the service via the RESTful API over HTTP/HTTPS.

When the service is implemented in a production environment, only HTTPS shall be used.

All API calls require at least the following two HTTP headers to be populated:

APIClientKey

This header value shall represent the specific implementation of the service and identifies the application itself. The header name is: X-ApiClientKey

APIClientSecret

This header value shall represent the equivalent of a password for the APIClientKey.

Two (2) other headers represent a specific user session. They are shown below.

UserAuth

The first of these is the UserAuth header which assumes the X-UserAuth header name and contains the user’s session id. This value will expire and no longer be valid after the user session expires due to non-use or the user logs out.

AdminAuth

The second user session header is the AdminAuth header which assumes the X-AdminAuth header name. When present, it designates the user has administrative level priviledges. And, like the UserAuth header, it will expireafter a period of non-use or the user logs out.

Each API call shall be made using the appropriate HTTP method (GET, POST, PUT, PATCH, DELETE) against an endpoint designated by the concatenation of the following:

Protocol
Host Name
BasePath
Endpoint

This combination is referred to as the Request URL.

Example:

Protocol: https://
Host Name: users.mind2mobile.net
Port: 8080
BasePath: /v1
Endpoint: users

Request URL: https://users.mind2mobile.net:8080/v1/users

Depending upon the call, a payload may be passed in the body. These payloads are defined as JSON objects and can be found in the ***definitions*** section of the Swagger specification file.

Each request also returns an HTTP response code. Some API calls will return an JSON object as defined in the ***definitions*** section of the Swagger specification file. Please refer to the Responses section for each call which describes the applicable return code.

Additionally, when an API call returns an error, it may return the **ErrorResponse** object to provide additional detail on why the API call failed. For clients that are unable to process this request, it may be ignored.

The Curl application can be used to submit a request to the service for processing. The following is an example curl call for the GET /users/{userid} endpoint:

```
curl -X GET "https://users.mind2mobile.net/v1/users/ef34a11c-
a3db-4a8a-b4d2-8d8aeb723c3b" -H "accept: application/json" -H
"X-UserAuth: e7667243-99c6-4f95-97a9-d397d62e8547" -H "X-
ApiClientKey: e7667243-99c6-4f95-97a9-d397d62e8547" -H "X-
ApiClientSecret: MyClientSecret"
```

Date and Time Values

All Date and Time values shall be expressed per RFC 3339, section 5.6

Dates shall be expressed using the following format:

CCYY-MM-DD

Where:

CC is the century (i.e. 19, 20, 21, etc.)

YY is the current year within the century (i.e. 19)

MM is the numerical month with leading zero.

January = 01, February = 02…December = 12

Hence, the date of 02 June, 2019 would be represented as: 2019-06-02

Times shall be expressed as concatenation of the Date and the time expressed in 24-hour Greenwich time using the following format:

CCYY-MM-DDTHH:NN:SSZ

The date component shall follow the rules stated above. The TIME component shall be expressed as:

HH is the hour (00…23)

NN is the minute (00…59)

SS is the second (00…59)

Assuming the time is 10:22 PM on 02 June, 2019, the result would be represented as: 2019-06-02T22:19:00Z

Components:

The following definitions are defined for the User Management Service within the Swagger file.

ErrorResponse

Description: The Error Response object is returned in the event of an error while making an API call. It provided additional information that can assist in diagnosing why an API call failed.

Name	Type	Description	Attributes
code	Integer	**Required**. The HTTP Response code.	Default: 200
subCode	Integer	**Optional**. Application error code.	Default: 0
timestamp	String	**Required**. Formatted Date & Time	
message	String	**Optional**. An optional message conveying the error code.	

User

Description: The User Object represents a user account.

Name	Type	Description	Attributes
userId	String	**Conditional**. A UUID assigned to the user. It is generated and supplied by service. Should a value be passed in this field while creating an account, the value will be ignored.	Default: N/A
userName	String	**Optional**. A user supplied identifier indicating the preferred name for the user. If not supplied, the email address will be used as the username.	Default: User's Email Address
email	String	**Required**. The User's email address.	
firstName	String	**Optional**. User's first name.	
lastName	String	**Optional.** User's last name.	
Salutation	String	**Optional.** User's salutation such as Mr, Mrs, Ms, Dr.	
phone	String	**Conditional.** User's cell phone number.	

		Required if the SMS TFA method is specified or desired. Format: +C-AAA-PPP-XXXX C = Country Code AAA = Area Code PPP – Prefix XXXX – Post Fix	
password	String	**Mandatory.** The User's Password hashed using a slow hash such as BCrypt. This value is NEVER returned in the clear.	
tfaEnabled	Boolean	**Conditional.** Indicates whether Two-Factor Authentication (TFA) is enabled.	Default: false
tfaMethods	String[]	**Mandatory.** An array of strings indicating the methods configured for Two-Factor Authentication (TFA).	Default: []
roles	String[]	**Conditional.** A read-only array containing the names of the roles assigned to the user for this application. **It is returned only for users with Admin privileges.**	Default: []
status	AccountStatusInfo	**Conditional.** Contains information regarding the current status of the account. **It is returned only for users with Admin privileges.**	

AccountStatusInfo

Description: The AccountStatusInfo object is returned as part of the User object. It conveys information regarding the status of the User account.

Name	Type	Description	Attributes
isActive	Boolean	**Required**. Flag indicating whether the User account is active.	Default: false
isSuspended	Boolean	**Required**. Flag indicating whether the User account has been suspended.	Default: false
suspendedCode	Integer	**Conditional**. A code indicating the reason the account was suspended. **This value is returned only when the isSuspended flag is true.**	Default: 0
suspendedReason	String	**Conditional**. A message conveying the reason the account was suspended and any additional information. **This value is returned only when the isSuspended flag is true.**	
suspendedTS	String	**Conditional.** A timestamp value indicating when the account was suspended. **This value is returned only when the isSuspended flag is true.**	
unsuspendTS	String	**Conditional.** A timestamp value indicating when a suspended account may be unsuspended. If omitted, the account must be manually unsuspended. This value, when present, must always reflect a time greater than the suspendTS value. **This value is returned only when the isSuspended flag is true.**	

Request Objects

UserLoginRequest

Description: The UserLoginRequest is passed when the user wishes to sign-on/log-on to the remote system.

Name	**Type**	**Description**	**Attributes**
userName	String	**Required**. The username associated with the User account. This may be the value supplied by the user (if set) OR it may be the user's email address.	Default: ""
password	String	**Required**. This is the user's password. The value will always be converted to a BCrypted value prior to storing. Care must be taken to ensure this value is never stored or exposed in plaintext (i.e. in log files).	Default: ""
twoFactorCode	String	**Required**. This is a two-factor code generated either by an authenticator or sent via SMS. By default, the service will assume that the value is generated by an authenticator UNLESS the smsMethods array contains "SMS" and user has requested the code be sent via SMS.	Default: ""

UserChangePasswordRequest

Description: The UserLoginRequest is passed when the user wishes to sign-on/log-on to the remote system.

Name	Type	Description	Attributes
userName	String	**Required**. The username associated with the User account. This may be the value supplied by the user (if set) OR it may be the user’s email address.	Default: “”
currentPassword	String	**Required**. This is the current user password. The value will always be converted to a BCrypted value prior to storing. Care must be taken to ensure this value is never stored or exposed in plaintext (i.e. in log files).	Default: “”
newPassword	String	**Required**. This is the desired new password. It is recommended that the plaintext version of the password be checked against a known corpus of cracked passwords. Additionally, this service should support a password history to prevent the reuse of an existing password. The value will always be converted to a BCrypted value prior to storing. Care must be taken to ensure this value is never stored or exposed in plaintext (i.e. in log files).	Default: “”

Responses Objects

UserLoginResponse

Description: The UserLoginRequest is passed when the user wishes to sign-on/log-on to the remote system.

Name	Type	Description	Attributes
id	String	**Required**. This is the User’s UUID in the system following a successful login request.	Default: “”
userName	String	**Required**. This is the User’s userName or email address (if userName is not not set) in the User account.	Default: “”
token	String	**Required**. This is a JWT created for the session.	Default: “”
expiration	String	**Required.** A timestamp indicating when the session will be closed. It mirrors the expiration time of the JWS.	

TFAInfoResponse

Description: The TFAInfoResponse is the object returned when Two-Factor Authentication is requested prior to a login.

Name	Type	Description	Attributes
methods	String[]	**Required**. This an array of strings containing the TFA methods the service will use when authenticating the user.	Default: [“Authenticator”]

uri	String	**Required**. This is the URI generated for use with the Google authenticator.
		It follows the pattern shown below.
		`otpauth://TYPE/LABEL?PARAMETERS`
		See the following or specific details on the construction of this URI:
		https://github.com/google/google-authenticator/wiki/Key-Uri-Format
			`Example:`
			`otpauth://totp/ACME%20Co:john.doe@email.com?`
			`secret=HXDMVJECJJWSRB3HWIZR4IFUGFTMXBOZ`
			`&issuer=ACME%20Co&algorithm=SHA1`
			`&digits=6&period=30`

image	String	**Optional**. A MIME (aka Base64) encoded string containing the image data.	
imageFmt	String	**Conditional.** This value is required when the image field is populated.	

User Management API Methods

Use Case: Create a User Account

Description: User desires to create a new account. Creating an account is a two-step process. The first step is this call. It creates an inactive User record.

The second step is to call the GET method which includes the UserKey header and the username query parameter.

Only after the second step, if the UserKey and username are correct, is the account activated and will permit the user to access the account.

Step 1 – Create an Account

HTTP Method: POST
Path: /users

Preconditions:

- SMTP server configured to send email
- Email address set in the CreateUserRequest object
- APIClientKey value has been assigned and provided to the caller.
- APIClientSecret value has been assigned and provided to the caller.

Headers:

APIClientKey: X-ApiClientKey=”key”
APIClientSecret: X-APiClientSecret=”secret”

Parameters:

Path: N/A
Query: N/A
Body: ***CreateUserRequest*** object configured and passed in the request body.

The CreateUserRequest object must have the following fields populated:

- userName – User Supplied userName or email
- email – valid email
- password – a BCrypted Hash of the User’s password.

Responses:

Code	Reason	Response Body
201	**Successful Account Creation**	User
400	**Bad Request** One or more headers and/or the body are invalid or missing.	ErrorResponse
401	**Unauthorized** One or more of the headers are invalid.	ErrorResponse
409	**Duplicate User** The userName or email already exist.	ErrorResponse
501	**Not Implemented**	ErrorResponse

Step 2 – Confirm and Activate the Account

HTTP Method: GET
Path: /users

Description:
The call is made to activate the previous created account.

If following the Step 1, and email was sent to the supplied email address, a link to this call may be embedded in that email.

Preconditions:

- Email address was previously set in the CreateUserRequest object
- APIClientKey value has been assigned and provided to the caller.
- APIClientSecret value has been assigned and provided to the caller.

Headers:
APIClientKey: X-ApiClientKey=”key”
APIClientSecret: X-APiClientSecret=”secret”

Parameters:

Path: N/A

Query: ***userName*** – Username or email address

userId – value assigned by the server and returned in the User record.

Body: N/A

Responses:

Code	Reason	Body
201	Successful Account Creation	User
400	Bad Request	ErrorResponse
401	Unauthorized	ErrorResponse
409	Duplicate User	ErrorResponse
501	Not Implemented	ErrorResponse

Use Case – Log a User Into the application

HTTP Method: POST
Path: /users/login

Description:
This call attempts to sign a user into the application specified by the APIClientKey.

Preconditions:

- User account has been created and activated
- APIClientKey value has been assigned and provided to the caller.
- APIClientSecret value has been assigned and provided to the caller.
- A Two-Factor authentication code has been either generated or received.

Headers:
APIClientKey: X-ApiClientKey=”key”
APIClientSecret: X-APiClientSecret=”secret”

Parameters:

Path: N/A

Query: ***useSMS*** – If true, the TFA value was previously sent by SMS.
If false or omitted, the TFA value is expected to have been generated by an authenticator application.

Body: ***UserLoginRequest***

Responses:

Code	Reason	Body
201	**Successful Login** The value for subsequent UserAuth header will be returned with the response body as the token field.	UserLoginResponse
400	**Bad Request**	ErrorResponse
401	**Unauthorized**	ErrorResponse
501	**Not Implemented**	ErrorResponse

Use Case – Log a User Out

HTTP Method: POST
Path: /users/{***userId***}/logout

Description:
The call is made to log a user out of the application.

Preconditions:

- The user has logged in and has an active session.
- UserAuth value is has been assigned. This is a JWT provided when the user was logged in.
- APIClientKey value has been assigned and provided to the caller.
- APIClientSecret value has been assigned and provided to the caller.

Headers:

UserAuth: X-UserAuth=”key”
APIClientKey: X-ApiClientKey=”key”
APIClientSecret: X-APiClientSecret=”secret”

Parameters:

Path: N/A

Query: ***userId*** – value assigned by the server and previously returned in the User record.

Body: N/A

Responses:

Code	Reason	Body
204	Successfully Logged Out	
400	Malformed or Bad Request	ErrorResponse
401	Unauthorized	ErrorResponse
501	Not Implemented	ErrorResponse

Use Case – Refresh a User Session

HTTP Method: POST
Path: /users/{***userId***}/refresh

Description:
This call attempts to refresh a User Session.

Preconditions:

- User has successfully logged in.
- APIClientKey value has been assigned and provided to the caller.
- APIClientSecret value has been assigned and provided to the caller.
- A Two-Factor authentication code has been either generated or received.

Headers:
UserKey: X-UserKey="JWT previously returned during login"
APIClientKey: X-ApiClientKey="key"
APIClientSecret: X-APiClientSecret="secret"

Parameters:
Path: ***userId*** – The userId of the user from the previously return User record.

Query: N/A
Body: N/A

Responses:

Code	Reason	Body
201	**Successful Refesh** The value for subsequent UserAuth header will be returned with the response body as the token field.	UserLoginResponse
400	**Bad Request**	ErrorResponse
401	**Unauthorized** One or more of the headers are invalid or the JWT in the UserAuth header is expired.	ErrorResponse
403	**Forbidden** The user can't refresh the session. Most likely caused by the user being logged out.	ErrorResponse
501	**Not Implemented**	ErrorResponse

Use Case – Retrieve a User’s account record

HTTP Method: GET
Path: /users/{***userId***}

Description:
The call is made to retrieve a User’s account record by their userId.

Preconditions:

- The User has previously logged in and is still logged in.
- The JWT in the User.token field has not expired.
- APIClientKey value has been assigned and provided to the caller.
- APIClientSecret value has been assigned and provided to the caller.

Headers:

UserAuth: X-UserAuth=”key”
APIClientKey: X-ApiClientKey=”key”
APIClientSecret: X-APiClientSecret=”secret”

Parameters:

Path: ***userId*** – value assigned by the server and returned in the User record.

Query: N/A

Body: N/A

Responses:

Code	Reason	Body
200	**Success** Successfully retrieved the User record	User
400	**Bad Request**	ErrorResponse
401	**Unauthorized**	ErrorResponse
404	**Not Found**	ErrorResponse
501	**Not Implemented**	ErrorResponse

Use Case – Update a User's account record

HTTP Method: PUT
Path: /users/{***userId***}

Description:
The call is made to update a User's account record by their userId. It is possible to update the userName and email fields provided a duplicate error condition doesn't exist.

Preconditions:

- The User has previously logged in and is still logged in.
- The JWT in the User.token field has not expired.
- APIClientKey value has been assigned and provided to the caller.
- APIClientSecret value has been assigned and provided to the caller.

Headers:
UserAuth: X-UserAuth="key"
APIClientKey: X-ApiClientKey="key"
APIClientSecret: X-APiClientSecret="secret"

Parameters:
Path: ***userId*** – value assigned by the server and returned in the User record.

Query: N/A

Body: ***User*** record with the userId field set.
Other fields set as necessary.

Responses:

Code	Reason	Body
201	**Success** Successfully updated the User record	User
304	**Record Not Modified** This is returned should the record not be modified.	ErrorResponse
400	**Bad Request**	ErrorResponse
401	**Unauthorized**	ErrorResponse
404	**Not Found**	ErrorResponse
409	**Duplicate User** There was an attempt to change the User.username field and it is taken. There was an attempt to change the User.email address and it is taken by another User.	ErrorResponse
501	**Not Implemented**	ErrorResponse

Use Case – Update a User's account record via a patch

HTTP Method: PATCH
Path: /users/{***userId***}

Description:
The call is made to update specific fields of a User's account record by their userId. It is possible to update the userName and email fields provided a duplicate error condition doesn't exist.

Preconditions:

- The User has previously logged in and is still logged in.
- The JWT in the User.token field has not expired.
- APIClientKey value has been assigned and provided to the caller.
- APIClientSecret value has been assigned and provided to the caller.

Headers:

UserAuth: X-UserAuth="key"
APIClientKey: X-ApiClientKey="key"
APIClientSecret: X-APiClientSecret="secret"

Parameters:

Path: ***userId*** – value assigned by the server and returned in the User record.

Query: N/A

Body: ***User*** record with the userId field set.
Other fields set as necessary.

Responses:

Code	Reason	Body
201	**Success** Successfully updated the User record	User
304	**Record Not Modified** This is returned should the record not be modified.	ErrorResponse
400	**Bad Request**	ErrorResponse
401	**Unauthorized**	ErrorResponse
404	**Not Found**	ErrorResponse
409	**Duplicate User** There was an attempt to change the User.username field and it is taken.There was an attempt to change the User.email address and it is taken by another User.	ErrorResponse
501	**Not Implemented**	ErrorResponse

Use Case – Delete a User's Account

HTTP Method: DELETE
Path: /users/{***userId***}

Description:
The call is made delete an open User account.

Preconditions:

- The User has previously logged in and is still logged in.
- The JWT in the User.token field has not expired.
- APIClientKey value has been assigned and provided to the caller.
- APIClientSecret value has been assigned and provided to the caller.

Headers:

UserAuth: X-UserAuth="key"
APIClientKey: X-ApiClientKey="key"
APIClientSecret: X-APiClientSecret="secret"

Parameters:

Path: ***userId*** – value assigned by the server and returned in the User record.

Query: N/A

Body: N/A

Responses:

Code	Reason	Body
204	**Success** Successfully deleted the User record	
400	**Bad Request**	ErrorResponse
401	**Unauthorized**	ErrorResponse
404	**Not Foiund**	ErrorResponse
501	**Not Implemented**	ErrorResponse

Password Management API Methods

Use Case – Request a Password Token

HTTP Method: GET
Path: /passwords

Description:
The call is made request a password reset in the event the user has forgotten their password. A reset token is generated and sent via email to the specified email account. This token is NOT in the form of a URL.

Preconditions:

- APIClientKey value has been assigned and provided to the caller.
- APIClientSecret value has been assigned and provided to the caller.

Headers:
APIClientKey: X-ApiClientKey=”key”
APIClientSecret: X-APiClientSecret=”secret”

Parameters:
Path: N/A

Query: ***email*** – the registered email for the User

Body: N/A

Responses:

Code	Reason	Body
204	**Success** Successfully deleted the User record	
400	**Bad Request**	ErrorResponse
401	**Unauthorized** One or more of the headers is invalid or missing.	ErrorResponse
404	**Not Found** No account was found for the specified email address.	ErrorResponse
501	**Not Implemented**	ErrorResponse

Use Case – Reset the User's Password

HTTP Method: POST
Path: /passwords/{***userId***}/{***resetToken***}

Description:
The call is made reset the password. The ***resetToken*** was requested and sent, out-of-bound, to the user via email.

Preconditions:

- APIClientKey value has been assigned and provided to the caller.
- APIClientSecret value has been assigned and provided to the caller.
- A resetToken has been sent to the user's email and retrieved by the user from a previous call to request the reset token.

Headers:

APIClientKey: X-ApiClientKey="key"
APIClientSecret: X-APiClientSecret="secret"

Parameters:

Path:

userId – The userId field of the User record for the user whose password is to be reset.

resetToken – The token sent via email from a previous call to request the password reset.

Query: N/A

Body: ***newPassword*** – This is the new password encrypted using BCrypt.

Responses:

Code	Reason	Body
204	**Success** Successfully reset the User’s password.	
400	**Bad Request**	ErrorResponse
401	**Unauthorized** One or more of the headers is invalid or missing.	ErrorResponse
403	**Forbidden** A password reset request token was not previously requested.	ErrorResponse
404	**Not Found** UserId not found	ErrorResponse
501	**Not Implemented**	ErrorResponse

Use Case – Change the User's Password

HTTP Method: POST
Path: /passwords/{***userId***}

Description:
The call is made change the User's password.

Preconditions:

- APIClientKey value has been assigned and provided to the caller.
- APIClientSecret value has been assigned and provided to the caller.
- The User is signed in and UserAuth token has not expired.

Headers:

UserKey: X-UserAuth = "JWT token previously returned in the UserLoginResponse"
APIClientKey: X-ApiClientKey="key"
APIClientSecret: X-APiClientSecret="secret"

Parameters:

Path:

userId – The userId field of the User record for the user whose password is to be changed.

Query:

email – the registered email for the User

Body:

newPassword – This is the new password.

Care must be taken to never expose or store the newPassword value in plaintext. It should be hashed using BCrypt at the earliest opportunity.

Responses:

Code	Reason	Body
204	**Success** Successfully reset the User's password.	
400	**Bad Request**	ErrorResponse
401	**Unauthorized** One or more of the headers is invalid or missing.	ErrorResponse
404	**Not Found** UserId not found	ErrorResponse
501	**Not Implemented**	ErrorResponse

Two-Factor Authentication Management API Methods

Use Case – Request a Two-Factor Authentication Code via SMS

HTTP Method: GET
Path: /users/{***userId***}/tfa

Description:
The call will generate a unique one-time code that is to be sent via SMS to the registered user device (i.e. their cellphone number). The SMS method for TFA must be enabled for the account.

Preconditions:

- APIClientKey value has been assigned and provided to the caller.
- APIClientSecret value has been assigned and provided to the caller.
- The User account must have a registered cellphone.
- An SMS gateway must be configured to send the code to the registered device.

Headers:

APIClientKey: X-ApiClientKey=”key”
APIClientSecret: X-APiClientSecret=”secret”

Parameters:

Path: ***userId*** – The userId of the User

Query: N/A

Body: N/A

Responses:

Code	Reason	Body
203	**Success**	
400	**Bad Request**	ErrorResponse
401	**Unauthorized** One or more of the headers is invalid or missing.	ErrorResponse
403	**Forbidden**	ErrorResponse
405	**Not Allowed**	ErrorResponse
501	**Not Implemented**	ErrorResponse

Use Case – Enable or Disable Two-Factor Authentication

HTTP Method: POST
Path: /users/{***userId***}/tfa/{***enable***}

Description:
This API call enables or disables two-factor authentication for the user.

When initially enabling TFA, no credential payload is required. The response, if successful, includes the URL that can be imported by an authenticator app as well as an QR Code image, mime-encoded, and in the desired format.

The credential payload is required when disabling TFA.

If the **useSMS=true** was used, the code supplied in the SMS is what should be entered in the payload.

If **useSMS=false** or omitted, the code supplied should be that generated by an authenticator.

Only after the a valid TFA code is submitted via the payload will the desired action be taken.

Note (1) - when disabling TFA, ALL methods for TFA are disabled and removed from the account.

Note (2) - When enabling TFA, the authenticator link is supplied only once and is not recoverable by the user. You must disable TFA and reenable it to get a new authenticator link.

Preconditions:

- The UserAuth value has been assigned and has not expired.
- APIClientKey value has been assigned and provided to the caller.
- APIClientSecret value has been assigned and provided to the caller.
- The User account must have a registered cellphone.
- An SMS gateway must be configured to send the code to the registered device.

Headers:

UserAuth: X-UserAuth = “JWT Token”
APIClientKey: X-ApiClientKey=”key”
APIClientSecret: X-APiClientSecret=”secret”

Parameters:

Path: ***phase*** – The userId of the User

Enabling TFA is a two-phase process.

In phase=1, the desired action is specified as a query. The response includes the URL for an authenticator app to use.
In phase=2, the credentials are passed and include the a TFA code generated by the authenticator app to activate TFA on the account.

Disabling TFA is, a one or two step process.
If the TFA code is generated by an authenticator, it can be included in the body and the phase query param can be omitted or set to 1.
If the TFA code is being sent via SMS (**phase=1** and **useSMS=true**) then, upon receiving the SMS code, the code must be specified in the payload and the query parameters, **phase=2** and **useSMS=true**, must be set.

userId – The userId for the user. This is a UUID.

enable – A Boolean used to indicate whether Two-Factor Authentication should be enabled for the specified method.

Query: ***useSMS*** – A Boolean value used to indicate whether to generate the one time code using SMS.

Body: ***UserLoginRequest*** – a valid set of Login credentials.

Responses:

Code	Reason	Body
201	**Success** Two-factor authentication enabled.	TFAInfoResponse
400	**Bad Request**	ErrorResponse
401	**Unauthorized** One or more of the headers is invalid or missing.	ErrorResponse
403	**Forbidden**	ErrorResponse
501	**Not Implemented**	ErrorResponse