Sumit Tiwari

Okta Developer Study Guide

Contents

[Authentication API (Transaction Model) 2](#_Toc84167172)

[Introduction 4](#_Toc84167173)

[Application Types 4](#_Toc84167174)

[Public Application 4](#_Toc84167175)

[Trusted Application 4](#_Toc84167176)

[Authentication Operations 4](#_Toc84167177)

[Primary Authentication 4](#_Toc84167178)

[Okta Auth SDK – JavaScript 4](#_Toc84167179)

[Installation 4](#_Toc84167180)

[Retrieve and Store an OIDC Token 4](#_Toc84167181)

[Client Configuration 4](#_Toc84167182)

[Retrieve ID token from Okta 4](#_Toc84167183)

[Parse the Token 4](#_Toc84167184)

[Store the parsed token 4](#_Toc84167185)

[Complete code to parse the token, display claim and store it in Token Manager 4](#_Toc84167186)

[Retrieve the stored token 4](#_Toc84167187)

[Complete OIDC Token Example 4](#_Toc84167188)

[Get Okta Session Cookie 4](#_Toc84167189)

[Complete Code OIDC + Session 4](#_Toc84167190)

Okta Developer Study Guide

# Authentication API (Transaction Model)

## Introduction

The Okta Authentication API provides operations to authenticate users, perform multifactor enrollment and verification, recover forgotten passwords, and unlock accounts. It can be used as a standalone API to provide the identity layer on top of your existing application, or it can be integrated with the Okta Sessions API to obtain an Okta session cookie and access apps within Okta.

The API is targeted for developers who want to build their own end-to-end login experience to replace the built-in Okta login experience and addresses the following key scenarios:

* Primary authentication allows you to verify username and password credentials for a user.
* Multifactor authentication (MFA) strengthens the security of password-based authentication by requiring additional verification of another Factor such as a temporary one-time password or an SMS passcode. The Authentication API supports user enrollment with MFA factors enabled by the administrator, as well as MFA challenges based on your Okta Sign-On Policy.
* Recovery allows users to securely reset their password if they've forgotten it, or unlock their account if it has been locked out due to excessive failed login attempts. This functionality is subject to the security policy set by the administrator.

## Application Types

The behavior of Authentication API varies based on application type and org policy (Sign on Policy, MFA enrollment policy and password policy). Policy evaluation is conditional on the client request context such as IP address, geo location, etc.

### Public Application

A public application is an application that anonymously starts an authentication or recovery transaction without an API token, such as the Okta Sign-In Widget. Public applications are aggressively rate-limited to prevent abuse and require primary authentication to be successfully completed before releasing any metadata about a user.

### Trusted Application

Trusted applications are backend applications that act as authentication broker or login portal for your Okta organization and may start an authentication or recovery transaction with an administrator API token. Trusted apps may implement their own recovery flows and primary authentication process and may receive additional metadata about the user before primary authentication has successfully completed.

Note: Trusted web applications may need to override the client request context to forward the originating client context for the user.

## Authentication Operations

### Primary Authentication

POST /api/v1/authn

Every authentication transaction starts with primary authentication which validates username and password. Policies are evaluated during primary authentication to determine password expiry, factor enrollment, etc. The transaction state of the response depends on the user’s status, group memberships and assigned policies.

#### Request parameters for primary authentication

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter | Description | Type | DataType | Required |
| Audience | App ID of the target app user is signing into | Body | String | False |
| Context | Additional context for authentication like user request context | Body | Context Object | False |
| Options | Optional features for authentication | Body | Options Object | False |
| Password | User’s password | Body | String | False |
| Username | User’s username  Note: User's non-qualified short-name (for example: dade.murphy) or unique fully-qualified sign in name (for example: dade.murphy@example.com) | Body | String | False |

#### Options Object

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Property | Description | DataType | Nullable | Unique | Readonly |
| multiOptionalFactorEnroll | Transitions transaction back to MFA\_ENROLL state after successful Factor enrollment when optional factors are available for enrollment | Boolean | True | False | False |
| warnBeforePasswordExpired | Transitions transaction to PASSWORD\_WARN state before SUCCESS if the user’s password is about to expire and within their password policy warning period | Boolean | True | False | False |

#### Context Object

The context object allows Trusted Web Application to pass additional context for the authentication or recovery transaction.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Property | Description | DataType | Nullable | Unique | Readonly |
| deviceToken | A globally uniqueID identifying the user’s client device or user agent | String | True | False | False |

No min length for this property. Max length is 32.

Overriding deviceToken is highly privileged operation and can only be performed using API token. It can only be used by Trusted application to perform authentication or recovery operation. If API token is not provided, Okta will ignore this property.

**Note**: You must always pass the same deviceToken for a user's device with every authentication request for per-device or per-session Sign-On Policy Factor challenges. If the deviceToken is absent or does not match the previous deviceToken, the user is challenged every-time instead of per-device or per-session. Similarly, you must always pass the same deviceToken for a user's device with every authentication request for new device security behavior detection. If the deviceToken is absent or doesn't match a recent deviceToken for the user, the request is considered to be from a new device.

#### Device Token best practices

Use the following recommendation for generating and storing a deviceToken for both web and native applications.

**Web apps**: Generate UUID or GUID for each client and persist the deviceToken using a secure, HTTP only cookie or browser local storage scoped to the okta tenant domain as the default implementation.

**Native apps**: Ask the device OS for a unique ID.

#### Response Parameters

Responses are authentication transaction object with the current state for the authentication transaction.

##### 401 Unauthorized

Invalid credentials, account locked, policy evaluation denied access.

HTTP/1.1 401 Unauthorized

Content-Type: application/json

{

"errorCode": "E0000004",

"errorSummary": "Authentication failed",

"errorLink": "E0000004",

"errorId": "oaeuHRrvMnuRga5UzpKIOhKpQ",

"errorCauses": []

}

##### 429 Too Many Requests

Rate limit has exceeded

HTTP/1.1 429 Too Many Requests

Content-Type: application/json

X-Rate-Limit-Limit: 1

X-Rate-Limit-Remaining: 0

X-Rate-Limit-Reset: 1447534590

{

"errorCode": "E0000047",

"errorSummary": "API call exceeded rate limit due to too many requests.",

"errorLink": "E0000047",

"errorId": "oaeWaNHfOyQSES34-a2Dw6Phw",

"errorCauses": []

}

#### Primary authentication with public application

Authenticate username/password via public application

##### Request

|  |  |  |
| --- | --- | --- |
| Endpoint | Header | Data |
| https://${yourOktaDomain}/api/v1/authn | Accept: application/json  Content-Type: application/json | {  "username": "dade.murphy@example.com",  "password": "somestrongpassword",  "options": {  "multiOptionalFactorEnroll": false,  "warnBeforePasswordExpired": false  }  } |

##### Response – Success, No Sign on Policy

Users with a valid password not assigned to a Sign-On Policy with additional verification requirements will successfully complete the authentication transaction.

{

"expiresAt": "2015-11-03T10:15:57.000Z",

"status": "SUCCESS",

"sessionToken": "00Fpzf4en68pCXTsMjcX8JPMctzN2Wiw4LDOBL\_9pe",

"\_embedded": {

"user": {

"id": "00ub0oNGTSWTBKOLGLNR",

"passwordChanged": "2015-09-08T20:14:45.000Z",

"profile": {

"login": "dade.murphy@example.com",

"firstName": "Dade",

"lastName": "Murphy",

"locale": "en\_US",

"timeZone": "America/Los\_Angeles"

}

}

}

}

##### Response – Invalid Credentials

HTTP/1.1 401 Unauthorized

Content-Type: application/json

{

"errorCode": "E0000004",

"errorSummary": "Authentication failed",

"errorLink": "E0000004",

"errorId": "oaeuHRrvMnuRga5UzpKIOhKpQ",

"errorCauses": []

}

##### Response – Locked out, based on Password Policy

Primary authentication requests for a user with LOCKED\_OUT status is based on password policy which defines whether to show or hide lockout failures which disclose a valid user identity to the caller.

###### Hide Lockout Failure

If password policy is configured to hide lockout failure a 401 Unauthorized error is returned preventing information disclosure of a valid user id. This is default config.

HTTP/1.1 401 Unauthorized

Content-Type: application/json

{

"errorCode": "E0000004",

"errorSummary": "Authentication failed",

"errorLink": "E0000004",

"errorId": "oaeuHRrvMnuRga5UzpKIOhKpQ",

"errorCauses": []

}

###### Show Lockout Failures

If the password policy is configured to show lockout failures then the authentication transaction completes with LOCKED\_OUT status.

{

"status": "LOCKED\_OUT",

"\_links": {

"next": {

"name": "unlock",

"href": "https://${yourOktaDomain}/api/v1/authn/recovery/unlock",

"hints": {

"allow": [

"POST"

]

}

}

}

}

##### Response – Password Expired

User must change their password to complete the authentication transaction. Users are challenged for MFA if MFA\_REQUIRED before PASSWORD\_EXPIRED if they have enrolled to a factor.

{

"stateToken": "007ucIX7PATyn94hsHfOLVaXAmOBkKHWnOOLG43bsb",

"expiresAt": "2015-11-03T10:15:57.000Z",

"status": "PASSWORD\_EXPIRED",

"\_embedded": {

"user": {

"id": "00ub0oNGTSWTBKOLGLNR",

"passwordChanged": "2015-09-08T20:14:45.000Z",

"profile": {

"login": "dade.murphy@example.com",

"firstName": "Dade",

"lastName": "Murphy",

"locale": "en\_US",

"timeZone": "America/Los\_Angeles"

}

},

"policy": {

"complexity": {

"minLength": 8,

"minLowerCase": 1,

"minUpperCase": 1,

"minNumber": 1,

"minSymbol": 0

}

}

},

"\_links": {

"next": {

"name": "changePassword",

"href": "https://${yourOktaDomain}/api/v1/authn/credentials/change\_password",

"hints": {

"allow": [

"POST"

]

}

},

"cancel": {

"href": "https://${yourOktaDomain}/api/v1/authn/cancel",

"hints": {

"allow": [

"POST"

]

}

}

}

}

##### Response – Factor Challenge

User is assigned to a Sign on Policy that requires additional factor verification to a previously enrolled factor by id to complete the authentication transaction.

{

"stateToken": "007ucIX7PATyn94hsHfOLVaXAmOBkKHWnOOLG43bsb",

"expiresAt": "2015-11-03T10:15:57.000Z",

"status": "MFA\_REQUIRED",

"\_embedded": {

"user": {

"id": "00ub0oNGTSWTBKOLGLNR",

"passwordChanged": "2015-09-08T20:14:45.000Z",

"profile": {

"login": "dade.murphy@example.com",

"firstName": "Dade",

"lastName": "Murphy",

"locale": "en\_US",

"timeZone": "America/Los\_Angeles"

}

},

"factors": [

{

"id": "rsalhpMQVYKHZKXZJQEW",

"factorType": "token",

"provider": "RSA",

"profile": {

"credentialId": "dade.murphy@example.com"

},

"\_links": {

"verify": {

"href": "https://${yourOktaDomain}/api/v1/authn/factors/rsalhpMQVYKHZKXZJQEW/verify",

"hints": {

"allow": [

"POST"

]

}

}

}

},

{

"id": "ostfm3hPNYSOIOIVTQWY",

"factorType": "token:software:totp",

"provider": "OKTA",

"profile": {

"credentialId": "dade.murphy@example.com"

},

"\_links": {

"verify": {

"href": "https://${yourOktaDomain}/api/v1/authn/factors/ostfm3hPNYSOIOIVTQWY/verify",

"hints": {

"allow": [

"POST"

]

}

}

}

},

{

"id": "sms193zUBEROPBNZKPPE",

"factorType": "sms",

"provider": "OKTA",

"profile": {

"phoneNumber": "+1 XXX-XXX-1337"

},

"\_links": {

"verify": {

"href": "https://${yourOktaDomain}/api/v1/authn/factors/sms193zUBEROPBNZKPPE/verify",

"hints": {

"allow": [

"POST"

]

}

}

}

},

{

"id": "clf193zUBEROPBNZKPPE",

"factorType": "call",

"provider": "OKTA",

"profile": {

"phoneNumber": "+1 XXX-XXX-1337"

},

"\_links": {

"verify": {

"href": "https://${yourOktaDomain}/api/v1/authn/factors/clf193zUBEROPBNZKPPE/verify",

"hints": {

"allow": [

"POST"

]

}

}

}

},

{

"id": "opf3hkfocI4JTLAju0g4",

"factorType": "push",

"provider": "OKTA",

"profile": {

"credentialId": "dade.murphy@example.com",

"deviceType": "SmartPhone\_IPhone",

"name": "Gibson",

"platform": "IOS",

"version": "9.0"

},

"\_links": {

"verify": {

"href": "https://${yourOktaDomain}/api/v1/authn/factors/opf3hkfocI4JTLAju0g4/verify",

"hints": {

"allow": [

"POST"

]

}

}

}

}

]

},

"\_links": {

"cancel": {

"href": "https://${yourOktaDomain}/api/v1/authn/cancel",

"hints": {

"allow": [

"POST"

]

}

}

}

}

##### Response – Factor Enroll

User is assigned to an enrollment policy that requires user to enroll to a factor during login and must select a factor to complete the authentication transaction.

{

"stateToken": "007ucIX7PATyn94hsHfOLVaXAmOBkKHWnOOLG43bsb",

"expiresAt": "2015-11-03T10:15:57.000Z",

"status": "MFA\_ENROLL",

"\_embedded": {

"user": {

"id": "00ub0oNGTSWTBKOLGLNR",

"passwordChanged": "2015-09-08T20:14:45.000Z",

"profile": {

"login": "dade.murphy@example.com",

"firstName": "Dade",

"lastName": "Murphy",

"locale": "en\_US",

"timeZone": "America/Los\_Angeles"

}

},

"factors": [

{

"factorType": "token",

"provider": "RSA",

"vendorName": "RSA",

"\_links": {

"enroll": {

"href": "https://${yourOktaDomain}/api/v1/authn/factors",

"hints": {

"allow": [

"POST"

]

}

}

},

"status": "NOT\_SETUP",

"enrollment": "OPTIONAL"

},

{

"factorType": "token:software:totp",

"provider": "OKTA",

"\_links": {

"enroll": {

"href": "https://${yourOktaDomain}/api/v1/authn/factors",

"hints": {

"allow": [

"POST"

]

}

}

},

"status": "NOT\_SETUP",

"enrollment": "OPTIONAL"

},

{

"factorType": "sms",

"provider": "OKTA",

"\_links": {

"enroll": {

"href": "https://${yourOktaDomain}/api/v1/authn/factors",

"hints": {

"allow": [

"POST"

]

}

}

},

"status": "NOT\_SETUP",

"enrollment": "OPTIONAL"

},

{

"factorType": "call",

"provider": "OKTA",

"\_links": {

"enroll": {

"href": "https://${yourOktaDomain}/api/v1/authn/factors",

"hints": {

"allow": [

"POST"

]

}

}

},

"status": "NOT\_SETUP",

"enrollment": "OPTIONAL"

},

{

"factorType": "push",

"provider": "OKTA",

"\_links": {

"enroll": {

"href": "https://${yourOktaDomain}/api/v1/authn/factors",

"hints": {

"allow": [

"POST"

]

}

}

},

"status": "NOT\_SETUP",

"enrollment": "OPTIONAL"

}

]

},

"\_links": {

"cancel": {

"href": "https://${yourOktaDomain}/api/v1/authn/cancel",

"hints": {

"allow": [

"POST"

]

}

}

}

}

#### Primary Authentication with Trusted Application

# Okta Auth SDK – JavaScript

It is built on top of Authentication API and OIDC API to enable users to create a fully branded login experience using JS. It is used by Okta Sign in Widget and it helps you build a JS custom app or SPA to control and customize beyond the possibility of widget.

## Installation

Include the following script tag in your web application.

<script src="https://global.oktacdn.com/okta-auth-js/4.5.0/okta-auth-js.min.js" type="text/javascript"></script>

For NPM: npm install @okta/okta-auth-js

## Retrieve and Store an OIDC Token

To retrieve and store OIDC token you must –

1. Configure OIDC app in Okta
2. Configure your Okta Auth SDK client
3. Redirect user to Okta Org Sign in page
4. Parse a token from the URL after redirect (Okta to app)
5. Store the parsed token inside SDK’s Token Manager
6. Retrieve the stored token from the Token Manager

### Client Configuration

To initialize the SDK create a new instance of the OktaAuth Object:

var authClient = new OktaAuth({

url: 'https://${yourOktaDomain}',

clientId: '${clientId}',

redirectUri: 'http://localhost:8080'

});

### Retrieve ID token from Okta

To retrieve an ID token from Okta, you will use the token.getWithRedirect method, specifying that you want an id\_token included in the response:

authClient.token.getWithRedirect({

responseType: 'id\_token' //Note: for authorization code flow use “code”

});

### Parse the Token

After the redirect, the URL will contain an ID token in the form of JWT, the token.parseFromUrl method can be used to parse the token from the URL:

authClient.token.parseFromUrl()

.then(res => {

const { idToken } = res.tokens;

})

To display a specific part of the token: console.log(`Hi ${idToken.claims.email}!`);

### Store the parsed token

Store the token you got from URL in Token Manager using the tokenManager.add method:

authClient.tokenManager.add('idToken', idToken);

### Complete code to parse the token, display claim and store it in Token Manager

authClient.token.parseFromUrl()

.then(res => {

const { idToken } = res.tokens;

console.log(`Hi ${idToken.claims.email}!`);

authClient.tokenManager.add('idToken', idToken);

})

### Retrieve the stored token

A token that is stored in the Token Manager can be retrieved using the tokenManager.get method:

authClient.tokenManager.get('idToken')

.then(function(token) {

if (token) {

// Token is valid

} else {

// Token has expired

}

})

## Complete OIDC Token Example

A complete code from the use cases above:

// Bootstrap the AuthJS Client

const authClient = new OktaAuth({

// Org URL

url: 'https://${yourOktaDomain}',

// OpenID Connect APP Client ID

clientId: '${clientId}',

// Trusted Origin Redirect URI

redirectUri: 'http://localhost:8080'

});

if (authClient.isLoginRedirect()) {

// Parse token from redirect url

authClient.token.parseFromUrl()

.then(data => {

const { idToken } = data.tokens;

console.log(`Hi ${idToken.claims.email}!`);

// Store parsed token in Token Manager

authClient.tokenManager.add('idToken', idToken);

console.log(idToken);

});

} else {

// Attempt to retrieve ID Token from Token Manager

authClient.tokenManager.get('idToken')

.then(idToken => {

console.log(idToken);

if (idToken) {

console.log(`Hi ${idToken.claims.email}!`);

} else {

// You're not logged in, you need a sessionToken

authClient.token.getWithRedirect({

responseType: 'id\_token'

});

}

})

}

## Get Okta Session Cookie

In the code example above, the ID Token is retrieved using a redirect to the Okta sign-in page. It is also possible to take a user-inputted username and password pair and pass them to the signIn method. This method then initiates an authentication process which returns an Okta session cookie. This Okta session cookie can then be used, along with the getWithRedirect method, to get back the ID Token. This means that there is no need to redirect the user to the Okta sign-in page.

else {

// You're not logged in, you need a sessionToken

var username = prompt('What is your username?');

var password = prompt('What is your password?');

authClient.signInWithCredentials({username, password})

.then(transaction => {

if (transaction.status === 'SUCCESS') {

authClient.token.getWithRedirect({

sessionToken: transaction.sessionToken,

responseType: 'id\_token'

});

}

});

}

## Complete Code OIDC + Session

// Bootstrap the AuthJS Client

const authClient = new OktaAuth({

// Org URL

url: 'https://${yourOktaDomain}',

// OpenID Connect APP Client ID

clientId: '${clientId}',

// Trusted Origin Redirect URI

redirectUri: 'http://localhost:8080'

});

if (authClient.isLoginRedirect()) {

// Parse token from redirect url

authClient.token.parseFromUrl()

.then(data => {

const { idToken } = data.tokens;

console.log(`Hi ${idToken.claims.email}!`);

// Store parsed token in Token Manager

authClient.tokenManager.add('idToken', idToken);

console.log(idToken);

});

} else {

// Attempt to retrieve ID Token from Token Manager

authClient.tokenManager.get('idToken')

.then(idToken => {

console.log(idToken);

if (idToken) {

console.log(`Hi ${idToken.claims.email}!`);

} else {

var username = prompt('What is your username?');

var password = prompt('What is your password?');

authClient.signInWithCredentials({username, password})

.then(transaction => {

if (transaction.status === 'SUCCESS') {

authClient.token.getWithRedirect({

sessionToken: transaction.sessionToken,

responseType: 'id\_token'

});

}

});

}

});

}