

# API Design

## Components

Designing APIs, also known as the surface area or API contract. We will use the Twitter REST API as an example, which will help us understand the considerations, restrictions, versioning, and the various functions that an API can encompass.

APIs typically provide general CRUD (Create, Read, Update, Delete) functions. In the case of Twitter, these functions may include `createTweet()`, `getTweets()`, `editTweet()` and `deleteTweet()`.

These operations are performed on entities, which can be thought of as nouns or resources (e.g. tweets, users, etc). The operations themselves, such as create, delete, etc., are equivalent to different HTTP/REST methods, namely GET, POST, PUT, and DELETE. As developers, we need not be concerned with the internal implementation of the API, but rather focus on the method signature and the return value.

By defining the API contract, we establish a clear interface for developers to interact with the API, ensuring consistent behavior and seamless integration into their application.

---

## Why bother designing good APIs?

When designing good APIs, it is crucial to consider their public-facing nature. These APIs can be relied upon by hundreds, if not thousands, of applications, which heavily depend on their function calls. This restriction limits API developers from making excessive changes to avoid potential crashes in the apps that rely on the Twitter API.

## Creating a Tweet

Let's consider the example of the `createTweet(userId, content)` method, which is used to create a tweet. Suppose we want to introduce a reply feature. Modifying the original

method signature to include a **parentId** parameter, like **createTweet(userId, content, parentId)**, would potentially disrupt a large number of applications. To address this, a better approach would be to make the **parentId** parameter optional.

By making the **parentId** parameter optional, we acknowledge that not every tweet is a reply and may not have a parent tweet. This design choice also ensures **backward compatibility**, which means that newer additions or changes to the software application should still be compatible with older versions of the same application. Therefore, applications utilizing the **createTweet(userId, content)** method would not be required to make any changes to their existing code.

Considering backward compatibility helps maintain a smooth transition and avoids breaking existing functionality for applications that rely on the original method.

A tweet object might look like the following and creating the tweet might have the following URL:

<https://api.twitter.com/v1.0/tweet>

userId: string, // The creator of the tweet, passed by the client

tweetId: string, // Uniquely identifies each tweet, created server side

content: string, // Passed in by the client

createdAt: date, // The timestamp, handled server side

likes: int, // handled by the server side

## Retrieving a Tweets

Retrieving a single tweet can be accomplished using an endpoint like

<https://api.twitter.com/v1.0/tweet/:userId>,

Which fetches all the tweets for a specific creator. However, when it comes to retrieving a list of tweets, a different approach is needed. This situation often arises in Higher Education institutions' websites, where a segment on the homepage displays their twitter activity. Using the same endpoint for retrieving a single tweet is not suitable in this case.

To address this, let's recall the concept of pagination discussed in the previous chapter. To fetch only 10 tweets, we can structure our URL as follows:

<https://api.twitter.com/v1.0/users/:id/tweets?limit=10&offset=0>. Here, the **limit** parameter indicates the number of tweets to be fetched per page, while the **offset** parameter determines the starting point. It's important to recall that GET requests are idempotent, meaning that regardless of how many times we call this URL. It should consistently return the same data without side effects. (Though, in twitter's case the homefeed data itself may change over time.)

By incorporating **limit** and **offset** parameters in the URL, we can implement pagination effectively, enabling the retrieval of a specific number of tweets at a time while maintaining consistency in the returned data.

We could technically design a server to write data to a database in handling of a get request, but it is against HTTP best practices. GET requests are supposed to be read-only.

## API Versions

When significant changes are made to an API, such as adding new parameters, methods, or completely changing how things work, companies typically update the API version. They announce that the previous version is deprecated, and developers are encouraged to update their API calls accordingly.

By versioning the API, it becomes easier to distinguish between different iterations and helps developers adapt their code to accommodate any changes or improvements. Deprecating the older. Version prompts developers to transition to the latest version, which often offers better functionality, improved performance, and bug fixes.

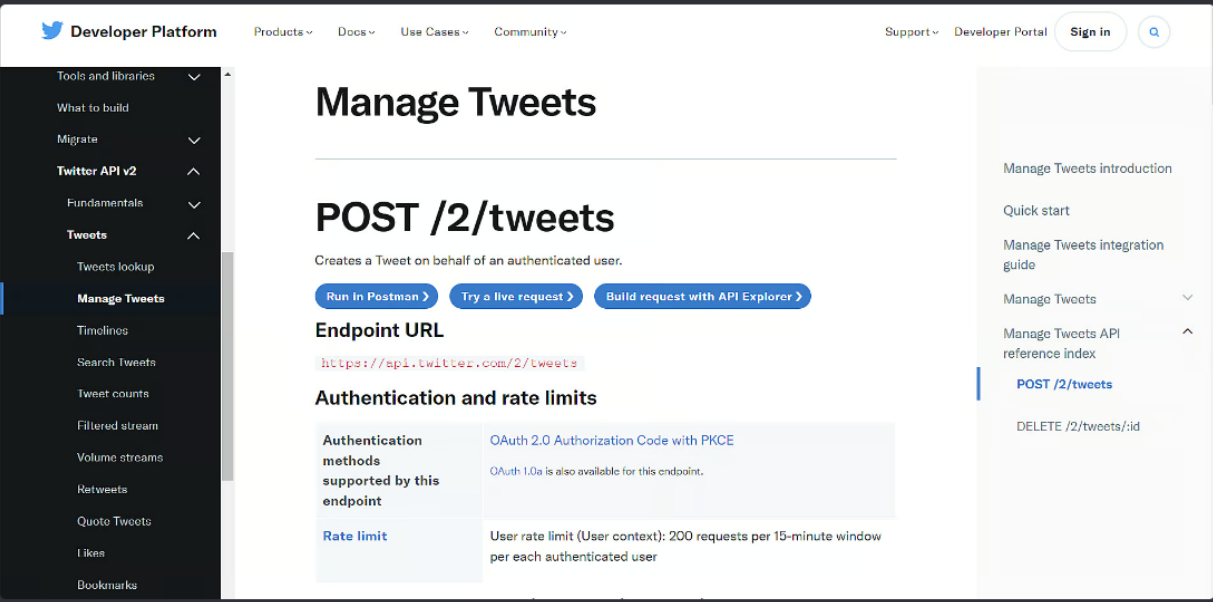
The deprecation announcement serves as a notification for developers to update their code and align it with the latest API specifications. This ensures compatibility and a smooth transition while embracing the advancements and updates in the API design.

Sometimes, to address the security vulnerabilities, API providers may release new versions as well. For this, they may deprecate old API keys and issue and generate new ones for the client. API keys are like a secret token, which is included in the API requests to verify the identity and permissions of the requesting application or client.

# Creating a Tweet

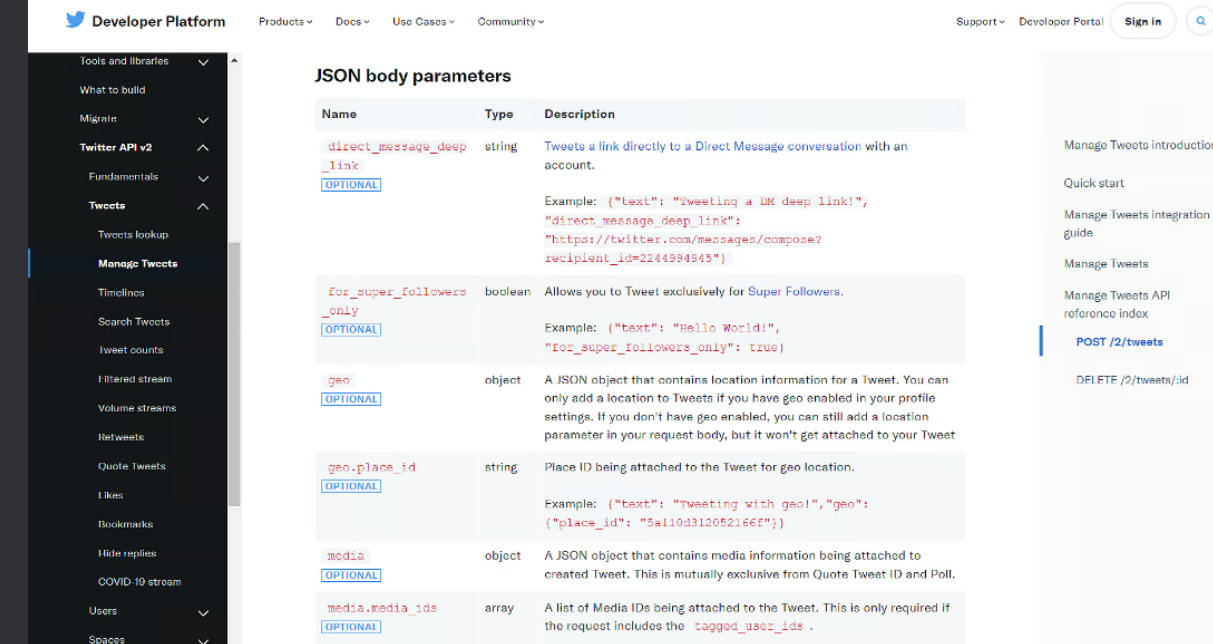
This section demonstrates how the REST API looks like on the Developer Platform on Twitter.

1. First, let's look at how a tweet is created.



The screenshot shows the Twitter Developer Platform interface. The left sidebar contains a navigation menu with categories like 'Tools and libraries', 'What to build', 'Migrate', 'Twitter API v2', 'Fundamentals', and 'Tweets'. The 'Tweets' category is expanded, showing options like 'Tweets lookup', 'Manage Tweets', 'Timelines', 'Search Tweets', 'Tweet counts', 'Filtered stream', 'Volume streams', 'Retweets', 'Quote Tweets', 'Likes', and 'Bookmarks'. The main content area is titled 'Manage Tweets' and features the endpoint 'POST /2/tweets'. Below the endpoint name, it states 'Creates a Tweet on behalf of an authenticated user.' and provides three buttons: 'Run in Postman', 'Try a live request', and 'Build request with API Explorer'. The 'Endpoint URL' is listed as 'https://api.twitter.com/2/tweets'. Under 'Authentication and rate limits', it specifies 'Authentication methods supported by this endpoint' as 'OAuth 2.0 Authorization Code with PKCE' and 'OAuth 1.0a is also available for this endpoint.' The 'Rate limit' is 'User rate limit (User context): 200 requests per 15-minute window per each authenticated user'. A right sidebar contains a list of links: 'Manage Tweets introduction', 'Quick start', 'Manage Tweets integration guide', 'Manage Tweets', 'Manage Tweets API reference index', 'POST /2/tweets' (highlighted), and 'DELETE /2/tweets/:id'.

Above we see the API call (POST) to create a tweet on behalf of the authenticated user.



The screenshot shows the 'JSON body parameters' section of the Twitter Developer Platform. It contains a table with columns 'Name', 'Type', and 'Description'. The parameters listed are: 'direct\_message\_deep\_link' (string, optional), 'for\_super\_followers\_only' (boolean, optional), 'geo' (object, optional), 'geo.place\_id' (string, optional), 'media' (object, optional), and 'media\_ids' (array, optional). Each parameter has a description and an example JSON snippet. The 'geo' parameter description states: 'A JSON object that contains location information for a Tweet. You can only add a location to Tweets if you have geo enabled in your profile settings. If you don't have geo enabled, you can still add a location parameter in your request body, but it won't get attached to your Tweet'. The 'media' parameter description states: 'A JSON object that contains media information being attached to created Tweet. This is mutually exclusive from Quote Tweet ID and Poll.' The 'media\_ids' parameter description states: 'A list of Media IDs being attached to the Tweet. This is only required if the request includes the tagged\_user\_ids.' The right sidebar is the same as in the previous screenshot, with 'POST /2/tweets' highlighted.

The payload that would be sent in the POST request. Notice how these are all optional.

Developer Platform

Products Docs Use Cases Community

Support Developer Portal Sign in

Fundamentals

Tweets

Twotools lookup

Manage Tweets

Timelines

Search Tweets

Twotools counts

Filtered stream

Volume streams

Retweets

Quote Tweets

Likes

Bookmarks

Hide replies

COVID-19 stream

Users

Spaces

### Example responses

Successful response

```
1 {
2   "data": {
3     "id": "1445888648472328192",
4     "text": "Are you excited for the weekend?"
5   }
6 }
```

### Response fields

Name	Type	Description
id	string	The ID of the newly created Tweet.
text	string	The text of the newly created Tweet.

The result of a successful response. The response fields in the Twitter API contain information returned by the API server after making a request.

2. Another example is a `GET` request for fetching a given user's tweets.

Quick start

Manage Tweets integration guide

Manage Tweets

Manage Tweets API reference index

POST /2/tweets

DELETE /2/tweets/:id

Developer Platform

Products Docs Use Cases Community

Support Developer Portal Sign in

Twitter API v2

Fundamentals

Tweets

Twotools lookup

Manage Tweets

Timelines

Search Tweets

Twotools counts

Filtered stream

Volume streams

Retweets

Quote Tweets

Likes

Bookmarks

Hide replies

COVID-19 stream

Users

Spaces

Direct Messages

Lists

Compliance

## Timelines

### GET /2/users/:id/tweets

Returns Tweets composed by a single user, specified by the requested user ID. By default, the most recent ten Tweets are returned per request. Using pagination, the most recent 3,200 Tweets can be retrieved.

The Tweets returned by this endpoint count towards the Project-level Tweet cap.

[Run in Postman](#) [Try a live request](#) [Build request with API Explorer](#)

#### Endpoint URL

`https://api.twitter.com/2/users/:id/tweets`

#### Authentication and rate limits

Authentication methods supported by this endpoint	OAuth 2.0 Authorization Code with PKCE OAuth 2.0 App-only OAuth 1.0a is also available for this endpoint.
Rate limit	App rate limit (Application only): 1500 requests per 15-minute window shared among all users of your app User rate limit (User context): 900 requests per 15-minute window per each

Above we see the end-point URL format we discussed before hand. Given an ID, we can fetch the tweet. Our pagination concept from earlier also comes into play here. The `2` in the URL represents the version of the API.

Timelines introduction

Timelines quick start guide

Timelines integration guide

Timelines version comparison

API reference

GET /2/users/:id/tweets

GET /2/users/:id/mentions

GET /2/users/:id/tweets/reverse...

Developer Platform

Products
Docs
Use Cases
Community

Support
Developer Portal
Sign in

Twitter API v2
Fundamentals
Tweets
Tweets lookup
Manage Tweets
Timelines
Search Tweets

### Path parameters

Name	Type	Description
id	string	Unique identifier of the Twitter account (user ID) for whom to return results. User ID can be referenced using the <a href="#">user/lookup</a> endpoint. More information on Twitter IDs is <a href="#">here</a> .

Timelines introduction
Timelines quick start guide
Timelines integration guide

Notice how the parameter `id` is required.

Developer Platform

Products
Docs
Use Cases
Community

Support
Developer Portal
Sign in

Twitter API v2
Fundamentals
Tweets
Tweets lookup
Manage Tweets
Timelines
Search Tweets
Tweet counts
Filtered stream
Volume streams
Retweets
Quote Tweets
Likes
Bookmarks
Hide replies

```
GET /2/users/{id}/tweets.json
{
  "author_id":
}
```

The following data objects can be expanded using this parameter:

- The Tweet author's user object
- The user object of the Tweet's author that the original Tweet is responding to
- Any mentioned users' object
- Any referenced Tweets' author's user object
- Attached media's object
- Attached poll's object
- Attached place's object
- Any referenced Tweets' object (this includes Tweet objects for previous versions of an edited Tweet).

max_results	integer	Specifies the number of Tweets to try and retrieve, up to a maximum of 100 per distinct request. By default, 10 results are returned if this parameter is not supplied. The minimum permitted value is 5. It is possible to receive less than the <code>max_results</code> per request throughout the pagination process.
-------------	---------	---

Timelines introduction
Timelines quick start guide
Timelines integration guide
Timelines version comparison
API reference

GET /2/users/{id}/tweets
GET /2/users/{id}/mentions
GET /2/users/{id}/timelines/reverse...

Passing in `max_results` parameter allows us to control how many results we fetch on each request. This is also optional and has a value set by default.