Python Client Library

supabase-py

[View on GitHub]

This reference documents every object and method available in Supabase's Python library, supabase-py. You can use supabase-py to interact with your Postgres database, listen to database changes, invoke Deno Edge Functions, build login and user management functionality, and manage large files.

Installing

Install with PyPi#

You can install supabase-py via the terminal. (for Python > 3.8)

PIP

Conda

Terminal

```
pip install supabase
```

Initializing

You can initialize a new Supabase client using the `create_client()` method.

The Supabase client is your entrypoint to the rest of the Supabase functionality and is the easiest way to interact with everything we offer within the Supabase ecosystem.

Parameters

---

supabase_urlRequiredstring
The unique Supabase URL which is supplied when you create a new project in your project dashboard.

supabase_keyRequiredstring

The unique Supabase Key which is supplied when you create a new project in your project dashboard.
optionsOptionalClientOptions
Options to change the Auth behaviors.
Details

create_client()With timeout option

```
import os
from supabase import create_client, Client

url: str = os.environ.get("SUPABASE_URL")
key: str = os.environ.get("SUPABASE_KEY")
supabase: Client = create_client(url, key)
```

Fetch data

By default, Supabase projects return a maximum of 1,000 rows. This setting can be changed in your project's API settings. It's recommended that you keep it low to limit the payload size of accidental or malicious requests. You can use `range()` queries to paginate through your data.

`select()` can be combined with Filters

`select()` can be combined with Modifiers

`apikey` is a reserved keyword if you're using the Supabase Platform and should be avoided as a column name.

Parameters

columnsOptionalstring
The columns to retrieve, defaults to *.

countOptionalCountMethod

The property to use to get the count of rows returned.

Getting your dataSelecting specific columnsQuery referenced tablesQuery referenced tables through a join tableQuery the same referenced table multiple timesFiltering through referenced tablesQuerying referenced table with countQuerying with count optionQuerying JSON dataQuerying referenced table with inner joinSwitching schemas per query

```
response = (
   supabase.table("planets")
   .select("*")
   .execute()
)
```

Data source
Response

Insert data

Parameters

jsonRequireddict, list
The values to insert. Pass an dict to insert a single row or an list to insert multiple rows.
countOptionalCountMethod
The property to use to get the count of rows returned.
returningOptionalReturnMethod
Either 'minimal' or 'representation'. Defaults to 'representation'.

default_to_nullOptionalbool
Make missing fields default to `null`. Otherwise, use the default value for the column. Only applies for bulk inserts.

Create a recordBulk create

```
response = (
  supabase.table("planets")
  .insert({"id": 1, "name": "Pluto"})
  .execute()
)
```

Data source
Response

Update data

`update()` should always be combined with [Filters](#) to target the item(s) you wish to update.

Parameters

jsonRequireddict, list The values to insert. Pass an dict to insert a single row or an list to insert multiple rows.
countOptionalCountMethod
The property to use to get the count of rows returned.

Updating your dataUpdating JSON data

```
response = (
    supabase.table("instruments")
    .update({"name": "piano"})
    .eq("id", 1)
    .execute()
)
```

Data source
Response

Upsert data

Primary keys must be included in the `values` dict to use upsert.

Parameters

jsonRequireddict, list
The values to insert. Pass an dict to insert a single row or an list to insert multiple rows.
countOptionalCountMethod
The property to use to get the count of rows returned.
returningOptionalReturnMethod
Either 'minimal' or 'representation'. Defaults to 'representation'.
ignore_duplicatesOptionalbool
Whether duplicate rows should be ignored.
on_conflictOptionalstring
Specified columns to be made to work with UNIQUE constraint.

default_to_nullOptionalbool

Make missing fields default to `null`. Otherwise, use the default value for the column. Only applies for bulk inserts.

Upsert your dataBulk Upsert your dataUpserting into tables with constraints

```
response = (
    supabase.table("instruments")
    .upsert({"id": 1, "name": "piano"})
    .execute()
)
```

Data source
Response

Delete data

`delete()` should always be combined with filters to target the item(s) you wish to delete.
If you use `delete()` with filters and you have [RLS](RLS) enabled, only rows visible through `SELECT` policies are deleted. Note that by default no rows are visible, so you need at least one `SELECT`/`ALL` policy that makes the rows visible.
When using `delete().in_()`, specify an array of values to target multiple rows with a single query. This is particularly useful for batch deleting entries that share common criteria, such as deleting users by their IDs. Ensure that the array you provide accurately represents all records you intend to delete to avoid unintended data removal.

Parameters

countOptionalCountMethod
The property to use to get the count of rows returned.

returningOptionalReturnMethod

Either 'minimal' or 'representation'. Defaults to 'representation'.

Delete recordsDelete multiple records

```
response = (
   supabase.table("countries")
   .delete()
   .eq("id", 1)
   .execute()
)
```

Data source
Response

Call a Postgres function

You can call Postgres functions as *Remote Procedure Calls*, logic in your database that you can execute from anywhere. Functions are useful when the logic rarely changes—like for password resets and updates.

```
create or replace function hello_world() returns text as $$
```

```
 select 'Hello world';

$$ language sql;
```

Parameters

fnRequiredcallable
The stored procedure call to be executed.
paramsOptionaldict of any
Parameters passed into the stored procedure call.
getOptionaldict of any
When set to `true`, `data` will not be returned. Useful if you only need the count.
headOptionaldict of any
When set to `true`, the function will be called with read-only access mode.

countOptionalCountMethod

Count algorithm to use to count rows returned by the function. Only applicable for set-returning functions. `"exact"`: Exact but slow count algorithm. Performs a `COUNT(*)` under the hood. `"planned"`: Approximated but fast count algorithm. Uses the Postgres statistics under the hood. `"estimated"`: Uses exact count for low numbers and planned count for high numbers.

Call a Postgres function without argumentsCall a Postgres function with argumentsBulk processingCall a Postgres function with filtersCall a read-only Postgres function

```
response = (
   supabase.rpc("hello_world")
   .execute()
)
```

Data source
Response

Using filters

Filters allow you to only return rows that match certain conditions.

Filters can be used on `select()`, `update()`, `upsert()`, and `delete()` queries.

If a Postgres function returns a table response, you can also apply filters.
Applying FiltersChainingConditional chainingFilter by values within JSON columnFilter Foreign Tables

```
# Correct

response = (

   supabase.table("instruments")

   .select("name, section_id")

   .eq("name", "flute")

   .execute()

)
```

```
# Incorrect
response = (
   supabase.table("instruments")
   .eq("name", "flute")
   .select("name, section_id")
   .execute()
)
```

Data source
Notes

Column is equal to a value

Match only rows where `column` is equal to `value`.

Parameters

- columnRequiredstring
  The column to filter on
- valueRequiredany
  The value to filter by

With `select()`

```
response = (
   supabase.table("planets")
   .select("*")
   .eq("name", "Earth")
   .execute()
)
```

Data source
Response

Column is not equal to a value

Match only rows where `column` is not equal to `value`.

Parameters

columnRequiredstring The column to filter on	
valueRequiredany	
The value to filter by	

With `select()`

```
response = (
   supabase.table("planets")
```

```
    .select("*")
    .neq("name", "Earth")
    .execute()
)
```

Data source
Response

Column is greater than a value

Match only rows where `column` is greather than `value`.

Parameters

columnRequiredstring	
The column to filter on	
valueRequiredany	
The value to filter by	

With `select()`

```
response = (
    supabase.table("planets")
    .select("*")
    .gt("id", 2)
```

```
   .execute()
)
```

Data source
Response
Notes

Column is greater than or equal to a value

Match only rows where `column` is greater than or equal to `value`.

Parameters

columnRequiredstring	The column to filter on
valueRequiredany	The value to filter by

With `select()`

```
response = (
   supabase.table("planets")
   .select("*")
   .gte("id", 2)
   .execute()
)
```

Data source
Response

Column is less than a value

Match only rows where `column` is less than `value`.

Parameters

columnRequiredstring	The column to filter on
valueRequiredany	The value to filter by

With `select()`

```
response = (
   supabase.table("planets")
   .select("*")
   .lt("id", 2)
   .execute()
)
```

Data source

Response

Column is less than or equal to a value

Match only rows where `column` is less than or equal to `value`.

Parameters

columnRequiredstring The column to filter on	
valueRequiredany	
The value to filter by	

With `select()`

```
response = (
   supabase.table("planets")
   .select("*")
   .lte("id", 2)
   .execute()
)
```

Data source
Response

Column matches a pattern

Match only rows where `column` matches `pattern` case-sensitively.

Parameters

columnRequiredstring The name of the column to apply a filter on	
patternRequiredstring	
The pattern to match by	

With `select()`

```
response = (
  supabase.table("planets")
  .select("*")
  .like("name", "%Ea%")
  .execute()
)
```

Data source
Response

Column matches a case-insensitive pattern

Match only rows where `column` matches `pattern` case-insensitively.

Parameters

columnRequiredstring The name of the column to apply a filter on	
patternRequiredstring	
The pattern to match by	

With `select()`

```
response = (
   supabase.table("planets")
   .select("*")
   .ilike("name", "%ea%")
   .execute()
)
```

Data source
Response

Column is a value

Match only rows where `column` IS `value`.

Parameters

- columnRequiredstring
  The name of the column to apply a filter on
- valueRequirednull | boolean
  The value to match by

Checking for nullness, True or False

```
response = (
   supabase.table("planets")
   .select("*")
   .is_("name", "null")
   .execute()
)
```

Data source
Response
Notes

Column is in an array

Match only rows where `column` is included in the `values` array.

Parameters

- column Required string
  The column to filter on
- values Required array
  The values to filter by

With `select()`

```
response = (
   supabase.table("planets")
   .select("*")
   .in_("name", ["Earth", "Mars"])
   .execute()
)
```

Data source
Response

Column contains every element in a value

Only relevant for jsonb, array, and range columns. Match only rows where `column` contains every element appearing in `value`.

Parameters

columnRequiredstring The column to filter on	
valuesRequiredobject	
The jsonb, array, or range value to filter with	

On array columnsOn range columnsOn `jsonb` columns

```
response = (
   supabase.table("issues")
   .select("*")
   .contains("tags", ["is:open", "priority:low"])
   .execute()
)
```

Data source
Response

Contained by value

Only relevant for jsonb, array, and range columns. Match only rows where every element appearing in `column` is contained by `value`.

Parameters

- columnRequiredstring
  The jsonb, array, or range column to filter on
- valueRequiredobject
  The jsonb, array, or range value to filter with

On array columnsOn range columnsOn `jsonb` columns

```
response = (
   supabase.table("classes")
   .select("name")
   .contained_by("days", ["monday", "tuesday", "wednesday", "friday"])
```

```
    .execute()
)
```

Data source
Response

Greater than a range

Only relevant for range columns. Match only rows where every element in `column` is greater than any element in `range`.

Parameters

- column Required string
  The range column to filter on
- range Required array
  The range to filter with

With `select()`

```
response = (
    supabase.table("reservations")
    .select("*")
    .range_gt("during", ["2000-01-02 08:00", "2000-01-02 09:00"])
    .execute()
)
```

Data source
Response
Notes

Greater than or equal to a range

Only relevant for range columns. Match only rows where every element in `column` is either contained in `range` or greater than any element in `range`.

Parameters

columnRequiredstring The range column to filter on
rangeRequiredstring
The range to filter with

With `select()`

```
response = (
  supabase.table("reservations")
  .select("*")
  .range_gte("during", ["2000-01-02 08:30", "2000-01-02 09:30"])
  .execute()
)
```

Data source
Response
Notes

Less than a range

Only relevant for range columns. Match only rows where every element in `column` is less than any element in `range`.

Parameters

columnRequiredstring The range column to filter on	
rangeRequiredarray	
The range to filter with	

With `select()`

```
response = (
  supabase.table("reservations")
  .select("*")
  .range_lt("during", ["2000-01-01 15:00", "2000-01-01 16:00"])
  .execute()
)
```

Data source
Response
Notes

Less than or equal to a range

Only relevant for range columns. Match only rows where every element in `column` is less than any element in `range`.

Parameters

columnRequiredstring The range column to filter on
rangeRequiredarray
The range to filter with

With `select()`

```
response = (
  supabase.table("reservations")
  .select("*")
  .range_lte("during", ["2000-01-01 14:00", "2000-01-01 16:00"])
  .execute()
)
```

Data source
Response
Notes

Mutually exclusive to a range

Only relevant for range columns. Match only rows where `column` is mutually exclusive to `range` and there can be no element between the two ranges.

Parameters

columnRequiredstring The range column to filter on	
rangeRequiredarray	
The range to filter with	

With `select()`

```
response = (
   supabase.table("reservations")
   .select("*")
   .range_adjacent("during", ["2000-01-01 12:00", "2000-01-01 13:00"])
   .execute()
)
```

Data source
Response
Notes

With a common element

Only relevant for array and range columns. Match only rows where `column` and `value` have an element in common.

Parameters

columnRequiredstring
The array or range column to filter on

valueRequiredIterable[Any]
The array or range value to filter with

On array columnsOn range columns

```
response = (
    supabase.table("issues")
    .select("title")
    .overlaps("tags", ["is:closed", "severity:high"])
    .execute()
)
```

Data source
Response

Match a string

Only relevant for text and tsvector columns. Match only rows where `column` matches the query string in `query`.

For more information, see Postgres full text search.

Parameters

columnRequiredstring
The text or tsvector column to filter on

queryRequiredstring
The query text to match with

optionsOptionalobject

Named parameters

Details

Text searchBasic normalizationFull normalizationWebsearch

```
response = (
   supabase.table("texts")
   .select("content")
   .text_search(
       "content",
       "'eggs' & 'ham'",
       options={"config": "english"},
   )
   .execute()
)
```

Data source
Response

Match an associated value

Match only rows where each column in `query` keys is equal to its associated value. Shorthand for multiple `.eq()`s.

Parameters

queryRequireddict	
The object to filter with, with column names as keys mapped to their filter values	

With `select()`

```
response = (
   supabase.table("planets")
   .select("*")
   .match({"id": 2, "name": "Earth"})
   .execute()
)
```

Data source
Response

Don't match the filter

Match only rows which doesn't satisfy the filter. `not_` expects you to use the raw PostgREST syntax for the filter values.

```
.not_.in_('id', '(5,6,7)')  # Use `()` for `in` filter
.not_.contains('arraycol', '{"a","b"}')  # Use `{}` for array values
```

With `select()`

```
response = (
   supabase.table("planets")
   .select("*")
   .not_.is_("name", "null")
   .execute()
)
```

Data source
Response

Match at least one filter

or_() expects you to use the raw PostgREST syntax for the filter names and values.

```
.or_('id.in.(5,6,7), arraycol.cs.{"a","b"}')  # Use `()` for `in` filter, `{}` for array values and `cs` for `contains()`.
.or_('id.in.(5,6,7), arraycol.cd.{"a","b"}')  # Use `cd` for `containedBy()`
```

Parameters

- filtersRequiredstring
  The filters to use, following PostgREST syntax
- reference_tableOptionalstring
  Set this to filter on referenced tables instead of the parent table

With `select()`Use `or` with `and`Use `or` on referenced tables

```
response = (
   supabase.table("planets")
   .select("name")
   .or_("id.eq.2,name.eq.Mars")
   .execute()
)
```

Data source
Response

Match the filter

filter() expects you to use the raw PostgREST syntax for the filter values.

```
.filter('id', 'in', '(5,6,7)')  # Use `()` for `in` filter
.filter('arraycol', 'cs', '{"a","b"}')  # Use `cs` for `contains()`, `{}` for array values
```

Parameters

- columnRequiredstring
  The column to filter on
- operatorOptionalstring
  The operator to filter with, following PostgREST syntax
- valueOptionalany
  The value to filter with, following PostgREST syntax

With `select()`On a foreign table

```
response = (
    supabase.table("planets")
    .select("*")
```

```
  .filter("name", "in", '("Mars","Tatooine")')
  .execute()
)
```

Data source
Response

Using modifiers

Filters work on the row level—they allow you to return rows that only match certain conditions without changing the shape of the rows. Modifiers are everything that don't fit that definition—allowing you to change the format of the response (e.g., returning a CSV string).

Modifiers must be specified after filters. Some modifiers only apply for queries that return rows (e.g., `select()` or `rpc()` on a function that returns a table response).

Order the results

Order the query result by `column`.

Parameters

columnRequiredstring The column to order by descOptionalbool Whether the rows should be ordered in descending order or not. foreign_tableOptionalstring Foreign table name whose results are to be ordered.	
nullsfirstOptionalbool	
Order by showing nulls first	

With `select()`On a foreign tableOrder parent table by a referenced table

```
response = (
  supabase.table("planets")
```

```
   .select("*")
   .order("name", desc=True)
   .execute()
)
```

Data source
Response

Limit the number of rows returned

Parameters

sizeRequirednumber	
The maximum number of rows to return	
foreign_tableOptionalstring	
Set this to limit rows of foreign tables instead of the parent table.	

With `select()`On a foreign table

```
response = (
   supabase.table("planets")
   .select("name")
```

```
    .limit(1)
    .execute()
)
```

Data source
Response

Limit the query to a range

Limit the query result by starting at an offset (`start`) and ending at the offset (`end`). Only records within this range are returned. This respects the query order and if there is no order clause the range could behave unexpectedly.

The `start` and `end` values are 0-based and inclusive: `range(1, 3)` will include the second, third and fourth rows of the query.

Parameters

startRequirednumber
The starting index from which to limit the result.
endRequirednumber
The last index to which to limit the result.
foreign_tableOptionalstring
Set this to limit rows of foreign tables instead of the parent table.

With `select()`On a foreign table

```
response = (
    supabase.table("planets")
    .select("name")
    .range(0, 1)
    .execute()
)
```

Data source
Response

Retrieve one row of data

Return `data` as a single object instead of an array of objects.
With `select()`

```
response = (
    supabase.table("planets")
    .select("name")
    .limit(1)
    .single()
    .execute()
)
```

Data source
Response

Retrieve zero or one row of data

Return `data` as a single object instead of an array of objects.
With `select()`

```
response = (
   supabase.table("planets")
   .select("*")
   .eq("name", "Earth")
   .maybe_single()
   .execute()
)
```

Data source
Response

Retrieve as a CSV

Return `data` as a string in CSV format.
Return data as CSV

```
response = (
   supabase.table("planets")
   .select("*")
   .csv()
   .execute()
)
```

Data source
Response
Notes

Using explain

For debugging slow queries, you can get the [Postgres `EXPLAIN` execution plan] of a query using the `explain()` method. This works on any query, even for `rpc()` or writes.

Explain is not enabled by default as it can reveal sensitive information about your database. It's best to only enable this for testing environments but if you wish to enable it for production you can provide additional protection by using a `pre-request` function.

Follow the Performance Debugging Guide to enable the functionality on your project.

Parameters

walOptionalboolean

If `true`, include information on WAL record generation.

verboseOptionalboolean

If `true`, the query identifier will be returned and `data` will include the output columns of the query.

settingsOptionalboolean

If `true`, include information on configuration parameters that affect query planning.

formatOptionalboolean

The format of the output, can be `"text"` (default) or `"json"`.

formatOptional"text" | "json"

The format of the output, can be `"text"` (default) or `"json"`.

buffersOptionalboolean

If `true`, include information on buffer usage.

analyzeOptionalboolean

If `true`, the query will be executed and the actual run time will be returned.

Get the execution planGet the execution plan with analyze and verbose

```
response = (
   supabase.table("planets")
   .select("*")
   .explain()
   .execute()
)
```

Data source

Response
Notes

Overview

The auth methods can be accessed via the `supabase.auth` namespace.
By default, the supabase client sets `persist_session` to true and attempts to store the session in memory.
Any email links and one-time passwords (OTPs) sent have a default expiry of 24 hours. We have the following rate limits in place to guard against brute force attacks.
The expiry of an access token can be set in the "JWT expiry limit" field in your project's auth settings. A refresh token never expires and can only be used once.

Create a new user

By default, the user needs to verify their email address before logging in. To turn this off, disable Confirm email in your project.
Confirm email determines if users need to confirm their email address after signing up.
- If Confirm email is enabled, a `user` is returned but `session` is null.
- If Confirm email is disabled, both a `user` and a `session` are returned.

By default, when the user confirms their email address, they are redirected to the `SITE_URL`. You can modify your `SITE_URL` or add additional redirect URLs in your project.
If sign_up() is called for an existing confirmed user:
- When both Confirm email and Confirm phone (even when phone provider is disabled) are enabled in your project, an obfuscated/fake user object is returned.
- When either Confirm email or Confirm phone (even when phone provider is disabled) is disabled, the error message, `User already registered` is returned.

To fetch the currently logged-in user, refer to `get_user()`.

Parameters

credentialsRequiredSignUpWithPasswordCredentials

Details

Sign up with an email and passwordSign up with a phone number and password (SMS)Sign up with a phone number and password (whatsapp)Sign up with additional user metadataSign up with a redirect URL

```
response = supabase.auth.sign_up(
   {
       "email": "email@example.com",
       "password": "password",
   }
)
```

Response

Create an anonymous user

Returns an anonymous user
It is recommended to set up captcha for anonymous sign-ins to prevent abuse. You can pass in the captcha token in the `options` param.

Parameters

credentialsRequiredSignInAnonymouslyCredentials

Details

Create an anonymous userCreate an anonymous user with custom user metadata

```
response = supabase.auth.sign_in_anonymously(
   {"options": {"captcha_token": ""}}
)
```

Response

Sign in a user

Log in an existing user with an email and password or phone and password.

Requires either an email and password or a phone number and password.

Parameters

credentialsRequiredSignInWithPasswordCredentials

Details

Sign in with email and passwordSign in with phone and password

```
response = supabase.auth.sign_in_with_password(
    {
        "email": "email@example.com",
        "password": "example-password",
    }
)
```

Response

Sign in with ID Token

Allows signing in with an OIDC ID token. The authentication provider used should be enabled and configured.

Parameters

credentials Required SignInWithIdTokenCredentials
Details

Sign In using ID Token

```
response = supabase.auth.sign_in_with_id_token(
    {
        "provider": "google",
        "token": "your-id-token",
    }
)
```

Response

Sign in a user through OTP

Requires either an email or phone number.
This method is used for passwordless sign-ins where a OTP is sent to the user's email or phone number.
If the user doesn't exist, `sign_in_with_otp()` will signup the user instead. To restrict this behavior, you can set `should_create_user` in `SignInWithPasswordlessCredentials.options` to `false`.

If you're using an email, you can configure whether you want the user to receive a magiclink or a OTP.
If you're using phone, you can configure whether you want the user to receive a OTP.
The magic link's destination URL is determined by the `SITE_URL`.
See redirect URLs and wildcards to add additional redirect URLs to your project.
Magic links and OTPs share the same implementation. To send users a one-time code instead of a magic link, modify the magic link email template to include `{{ .Token }}` instead of `{{ .ConfirmationURL }}`.

Parameters

credentialsRequiredSignInWithPasswordCredentials

Details

Sign in with emailSign in with SMS OTPSign in with WhatsApp OTP

```
response = supabase.auth.sign_in_with_otp(
    {
        "email": "email@example.com",
        "options": {
            "email_redirect_to": "https://example.com/welcome",
        },
    }
)
```

Response
Notes

Sign in a user through OAuth

This method is used for signing in using a third-party provider.
Supabase supports many different third-party providers.

Parameters

credentialsRequiredSignInWithOAuthCredentials
Details

Sign in using a third-party providerSign in using a third-party provider with redirectSign in with scopes

```
response = supabase.auth.sign_in_with_oauth(
   {"provider": "github"}
)
```

Sign in a user through SSO

Before you can call this method you need to establish a connection to an identity provider. Use the CLI commands to do this.
If you've associated an email domain to the identity provider, you can use the `domain` property to start a sign-in flow.
In case you need to use a different way to start the authentication flow with an identity provider, you can use the `provider_id` property. For example:

- Mapping specific user email addresses with an identity provider.
- Using different hints to identity the identity provider to be used by the user, like a company-specific page, IP address or other tracking information.

Parameters

paramsRequiredSignInWithSSOCredentials

Details

Sign in with email domainSign in with provider UUID

```
response = supabase.auth.sign_in_with_sso(
   {"domain": "company.com"}
)
```

Response

Notes

Sign out a user

In order to use the `sign_out()` method, the user needs to be signed in first.
By default, `sign_out()` uses the global scope, which signs out all other sessions that the user is logged into as well.
Since Supabase Auth uses JWTs for authentication, the access token JWT will be valid until it's expired. When the user signs out, Supabase revokes the refresh token and deletes the JWT from the client-side. This does not revoke the JWT and it will still be valid until it expires.

Parameters

optionsOptionalSignOutOptions

Details

Sign out

```
response = supabase.auth.sign_out()
```

Send a password reset request

The password reset flow consist of 2 broad steps: (i) Allow the user to login via the password reset link; (ii) Update the user's password.
The `reset_password_for_email()` only sends a password reset link to the user's email. To update the user's password, see [`update_user()`](#).
When the user clicks the reset link in the email they are redirected back to your application. You can configure the URL that the user is redirected to with the `redirectTo` parameter. See [redirect URLs and wildcards](#) to add additional redirect URLs to your project.
After the user has been redirected successfully, prompt them for a new password and call `update_user()`:

```
response = supabase.auth.update_user(
    {"password": new_password}
)
```

Parameters

- emailRequiredstring
  The email address of the user.
- optionsOptionalobject
  Details

Reset password

```
supabase.auth.reset_password_for_email(
    email,
    {
        "redirect_to": "https://example.com/update-password",
    }
)
```

Verify and log in through OTP

The `verify_otp` method takes in different verification types. If a phone number is used, the type can either be `sms` or `phone_change`. If an email address is used, the type can be one of the following: `email`, `recovery`, `invite` or `email_change` (`signup` and `magiclink` types are deprecated).
The verification type used should be determined based on the corresponding auth method called before `verify_otp` to sign up / sign-in a user.
The `TokenHash` is contained in the email templates and can be used to sign in. You may wish to use the hash with Magic Links for the PKCE flow for Server Side Auth. See this guide for more details.

Parameters

paramsRequiredVerifyOtpParams

Details

Verify Signup One-Time Password (OTP)Verify SMS One-Time Password (OTP)Verify Email Auth (Token Hash)

```
response = supabase.auth.verify_otp(
    {
        "email": "email@example.com",
      "token": "123456",
      "type": "email",
   }
)
```

Response

Retrieve a session

> This method retrieves the current local session (i.e in memory).
> The session contains a signed JWT and unencoded session data.
> Since the unencoded session data is retrieved from the local storage medium, do not rely on it as a source of trusted data on the server. It could be tampered with by the sender. If you need verified, trustworthy user data, call `get_user` instead.
> If the session has an expired access token, this method will use the refresh token to get a new session.

Get the session data

```
response = supabase.auth.get_session()
```

Response

Retrieve a new session

Returns a new session, regardless of expiry status. Takes in an optional refresh token. If not passed in, then refresh_session() will attempt to retrieve it from get_session(). If the current session's refresh token is invalid, an error will be thrown.

> This method will refresh the session whether the current one is expired or not.

Parameters

refresh_tokenOptionalstring

Refresh session using the current session

```
response = supabase.auth.refresh_session()
```

Response

Retrieve a user

This method fetches the user object from the database instead of local session.
This method is useful for checking if the user is authorized because it validates the user's access token JWT on the server.

Parameters

jwtOptionalstring

Takes in an optional access token JWT. If no JWT is provided, the JWT from the current session is used.

Get the logged in user with the current existing sessionGet the logged in user with a custom access token jwt

```
response = supabase.auth.get_user()
```

Response

Update a user

In order to use the `update_user()` method, the user needs to be signed in first.
By default, email updates sends a confirmation link to both the user's current and new email. To only send a confirmation link to the user's new email, disable Secure email change in your project's email auth provider settings.

Update the email for an authenticated userUpdate the phone number for an authenticated userUpdate the password for an authenticated userUpdate the user's metadataUpdate the user's password with a nonce

```
response = supabase.auth.update_user(
    {"email": "new@email.com"}
)
```

Response
Notes

Retrieve identities linked to a user

Gets all the identities linked to a user.

> The user needs to be signed in to call `get_user_identities()`.

Returns a list of identities linked to the user

```
response = supabase.auth.get_user_identities()
```

Response

Link an identity to a user

> The Enable Manual Linking option must be enabled from your project's authentication settings.
> The user needs to be signed in to call `link_identity()`.
> If the candidate identity is already linked to the existing user or another user, `link_identity()` will fail.
> If `link_identity` is run on the server, you should handle the redirect.

Parameters

- credentialsRequiredSignInWithOAuthCredentials
- Details

Link an identity to a user

```
response = supabase.auth.link_identity(
  {provider: "github"}
)
```

Response

Unlink an identity from a user

- The Enable Manual Linking option must be enabled from your [project's authentication settings](#).
- The user needs to be signed in to call `unlink_identity()`.
- The user must have at least 2 identities in order to unlink an identity.
- The identity to be unlinked must belong to the user.

Parameters

identity Required UserIdentity

Details

Unlink an identity

```
# retrieve all identites linked to a user
response = supabase.auth.get_user_identities()

# find the google identity
google_identity = list(
   filter(lambda identity: identity.provider == "google",
res.identities)
).pop()

# unlink the google identity
response = supabase.auth.unlink_identity(google_identity)
```

Send a password reauthentication nonce

This method is used together with `updateUser()` when a user's password needs to be updated.
If you require your user to reauthenticate before updating their password, you need to enable the Secure password change option in your project's email provider settings.
A user is only require to reauthenticate before updating their password if Secure password change is enabled and the user hasn't recently signed in. A user is deemed recently signed in if the session was created in the last 24 hours.
This method will send a nonce to the user's email. If the user doesn't have a confirmed email address, the method will send the nonce to the user's confirmed phone number instead.

Send reauthentication nonce

```
response = supabase.auth.reauthenticate()
```

Notes

Resend an OTP

Resends a signup confirmation, email change or phone change email to the user.
Passwordless sign-ins can be resent by calling the `sign_in_with_otp()` method again.
Password recovery emails can be resent by calling the `reset_password_for_email()` method again.
This method will only resend an email or phone OTP to the user if there was an initial signup, email change or phone change request being made.
You can specify a redirect url when you resend an email link using the `email_redirect_to` option.

Parameters

credentialsRequiredResendCredentials
Details

Resend an email signup confirmationResend a phone signup confirmationResend email change emailResend phone change OTP

```
response = supabase.auth.resend(
  {
    "type": "signup",
    "email": "email@example.com",
    "options": {
      "email_redirect_to": "https://example.com/welcome",
        },
    }
)
```

Notes

Set the session data

Sets the session data from the current session. If the current session is expired, setSession will take care of refreshing it to obtain a new session. If the refresh token or access token in the current session is invalid, an error will be thrown.

> This method sets the session using an `access_token` and `refresh_token`.
> If successful, a `SIGNED_IN` event is emitted.

Parameters

- access_tokenRequiredstring
- refresh_tokenRequiredstring

Refresh the session

```
response = supabase.auth.set_session(access_token, refresh_token)
```

Response
Notes

Exchange an auth code for a session

Log in an existing user by exchanging an Auth Code issued during the PKCE flow.

> Used when `flow_type` is set to `pkce` in client options.

Parameters

- auth_codeRequiredstring

Exchange Auth Code

```
response = supabase.auth.exchange_code_for_session(
   {"auth_code": "34e770dd-9ff9-416c-87fa-43b31d7ef225"}
)
```

Response

Auth MFA

This section contains methods commonly used for Multi-Factor Authentication (MFA) and are invoked behind the `supabase.auth.mfa` namespace.

Currently, we only support time-based one-time password (TOTP) as the 2nd factor. We don't support recovery codes but we allow users to enroll more than 1 TOTP factor, with an upper limit of 10.

Having a 2nd TOTP factor for recovery frees the user of the burden of having to store their recovery codes somewhere. It also reduces the attack surface since multiple recovery codes are usually generated compared to just having 1 backup TOTP factor.

Enroll a factor

> Currently, `totp` is the only supported `factor_type`. The returned `id` should be used to create a challenge.
> To create a challenge, see [`mfa.challenge()`].
> To verify a challenge, see [`mfa.verify()`].
> To create and verify a challenge in a single step, see [`mfa.challenge_and_verify()`].

Enroll a time-based, one-time password (TOTP) factor

```
response = supabase.auth.mfa.enroll(
    {
        "factor_type": "totp",
        "friendly_name": "your_friendly_name",
    }
)
```

Create a challenge

An enrolled factor is required before creating a challenge.
To verify a challenge, see `mfa.verify()`.

Create a challenge for a factor

```
response = supabase.auth.mfa.challenge(
    {"factor_id": "34e770dd-9ff9-416c-87fa-43b31d7ef225"}
)
```

Verify a challenge

To verify a challenge, please create a challenge first.

Verify a challenge for a factor

```
response = supabase.auth.mfa.verify(
    {
        "factor_id": "34e770dd-9ff9-416c-87fa-43b31d7ef225",
        "challenge_id": "4034ae6f-a8ce-4fb5-8ee5-69a5863a7c15",
        "code": "123456",
    }
)
```

Create and verify a challenge

An enrolled factor is required before invoking `challengeAndVerify()`.
Executes `mfa.challenge()` and `mfa.verify()` in a single step.

Create and verify a challenge for a factor

```
response = supabase.auth.mfa.challenge_and_verify(
    {
        "factor_id": "34e770dd-9ff9-416c-87fa-43b31d7ef225",
        "code": "123456",
    }
)
```

Unenroll a factor

Unenroll a factor

```
response = supabase.auth.mfa.unenroll(
    {"factor_id": "34e770dd-9ff9-416c-87fa-43b31d7ef225"}
)
```

Get Authenticator Assurance Level

Authenticator Assurance Level (AAL) is the measure of the strength of an authentication mechanism.
In Supabase, having an AAL of `aal1` refers to having the 1st factor of authentication such as an email and password or OAuth sign-in while `aal2` refers to the 2nd factor of authentication such as a time-based, one-time-password (TOTP).
If the user has a verified factor, the `next_level` field will return `aal2`, else, it will return `aal1`.

Get the AAL details of a session

```
response = supabase.auth.mfa.get_authenticator_assurance_level()
```

Auth Admin

Any method under the `supabase.auth.admin` namespace requires a `service_role` key.
These methods are considered admin methods and should be called on a trusted server. Never expose your `service_role` key in the browser.

Create server-side auth client

```
from supabase import create_client
from supabase.lib.client_options import ClientOptions

supabase = create_client(
   supabase_url,
   service_role_key,
   options=ClientOptions(
       auto_refresh_token=False,
       persist_session=False,
   )
)

# Access auth admin api
admin_auth_client = supabase.auth.admin
```

Retrieve a user

Fetches the user object from the database based on the user's id.
The `get_user_by_id()` method requires the user's id which maps to the `auth.users.id` column.

Parameters

uid Required string

The user's unique identifier

This function should only be called on a server. Never expose your `service_role` key in the browser.

Fetch the user object using the access_token jwt

```
response = supabase.auth.admin.get_user_by_id(1)
```

Response

List all users

Defaults to return 50 users per page.

Parameters

params Optional PageParams

An object which supports page and per_page as numbers, to alter the paginated results.

Details

Get a page of usersPaginated list of users

```
response = supabase.auth.admin.list_users()
```

Create a user

To confirm the user's email address or phone number, set `email_confirm` or `phone_confirm` to true. Both arguments default to false.

`create_user()` will not send a confirmation email to the user. You can use [`invite_user_by_email()`](#) if you want to send them an email invite instead.

If you are sure that the created user's email or phone number is legitimate and verified, you can set the `email_confirm` or `phone_confirm` param to `true`.

Parameters

attributes Required AdminUserAttributes

Details

With custom user metadataAuto-confirm the user's emailAuto-confirm the user's phone number

```
response = supabase.auth.admin.create_user(
    {
        "email": "user@email.com",
        "password": "password",
        "user_metadata": {"name": "Yoda"},
    }
)
```

Response

Delete a user

Delete a user. Requires a `service_role` key.

> The `delete_user()` method requires the user's ID, which maps to the `auth.users.id` column.

Parameters

- idRequiredstring

  The user id you want to remove.
- should_soft_deleteOptionalboolean

  If true, then the user will be soft-deleted (setting `deleted_at` to the current timestamp and disabling their account while preserving their data) from the auth schema.

Defaults to false for backward compatibility.
This function should only be called on a server. Never expose your `service_role` key in the browser.

Removes a user

```
supabase.auth.admin.delete_user(
   "715ed5db-f090-4b8c-a067-640ecee36aa0"
)
```

Send an email invite link

Sends an invite link to an email address.

Sends an invite link to the user's email address.
The `invite_user_by_email()` method is typically used by administrators to invite users to join the application.
Note that PKCE is not supported when using `invite_user_by_email`. This is because the browser initiating the invite is often different from the browser accepting the invite which makes it difficult to provide the security guarantees required of the PKCE flow.

Parameters

emailRequiredstring
The email address of the user.

optionsOptionalInviteUserByEmailOptions

Details

Invite a user

```
response =
supabase.auth.admin.invite_user_by_email("email@example.com")
```

Response

Generate an email link

The following types can be passed into `generate_link()`: `signup`, `magiclink`, `invite`, `recovery`, `email_change_current`, `email_change_new`, `phone_change`.
`generate_link()` only generates the email link for `email_change_email` if the Secure email change is enabled in your project's email auth provider settings.
`generate_link()` handles the creation of the user for `signup`, `invite` and `magiclink`.

Parameters

paramsRequiredGenerateLinkParams

Details

Generate a signup linkGenerate an invite linkGenerate a magic linkGenerate a recovery linkGenerate links to change current email address

```
response = supabase.auth.admin.generate_link(
    {
        "type": "signup",
        "email": "email@example.com",
        "password": "secret",
    }
)
```

Response

Update a user

Parameters

uidRequiredstring
attributesRequiredAdminUserAttributes
The data you want to update. This function should only be called on a server. Never expose your `service_role` key in the browser.
Details

Updates a user's emailUpdates a user's passwordUpdates a user's metadataUpdates a user's app_metadataConfirms a user's email addressConfirms a user's phone number

```
response = supabase.auth.admin.update_user_by_id(
   "11111111-1111-1111-1111-111111111111",
   {
       "email": "new@email.com",
   }
)
```

Response

Delete a factor for a user

Deletes a factor on a user. This will log the user out of all active sessions if the deleted factor was verified.

Parameters

paramsRequiredAuthMFAAdminDeleteFactorParams
Details

Delete a factor for a user

```
response = supabase.auth.admin.mfa.delete_factor(
    {
        "id": "34e770dd-9ff9-416c-87fa-43b31d7ef225",
        "user_id": "a89baba7-b1b7-440f-b4bb-91026967f66b"
    }
)
```

Response

Invokes a Supabase Edge Function.

Invoke a Supabase Function.

Requires an Authorization header.
When you pass in a body to your function, we automatically attach the Content-Type header for `Blob`, `ArrayBuffer`, `File`, `FormData` and `String`. If it doesn't match any of these types we assume the payload is `json`, serialise it and attach the `Content-Type` header as `application/json`. You can override this behaviour by passing in a `Content-Type` header of your own.

Basic invocationError handlingPassing custom headers

```
response = supabase.functions.invoke(
   "hello-world",
   invoke_options={
       "body": {"name": "Functions"},
   },
)
```

Subscribe to channel

- By default, Broadcast and Presence are enabled for all projects.
- By default, listening to database changes is disabled for new projects due to database performance and security concerns. You can turn it on by managing Realtime's replication.
- You can receive the "previous" data for updates and deletes by setting the table's `REPLICA IDENTITY` to `FULL` (e.g., `ALTER TABLE your_table REPLICA IDENTITY FULL;`).
- Row level security is not applied to delete statements. When RLS is enabled and replica identity is set to full, only the primary key is sent to clients.

Listen to broadcast messagesListen to presence syncListen to presence joinListen to presence leaveListen to all database changesListen to a specific tableListen to insertsListen to updatesListen to deletesListen to multiple eventsListen to row level changes

```
channel = supabase.channel("room1")
```

```
def on_subscribe(status, err):
   if status == RealtimeSubscribeStates.SUBSCRIBED:
       channel.send_broadcast(
           "cursor-pos",
           {"x": random.random(), "y": random.random()}
       )

def handle_broadcast(payload):
   print("Cursor position received!", payload)
```

```
channel.on_broadcast(event="cursor-pos",
callback=handle_broadcast).subscribe(on_subscribe)
```

Unsubscribe from a channel

> Removing a channel is a great way to maintain the performance of your project's Realtime service as well as your database if you're listening to Postgres changes. Supabase will automatically handle cleanup 30 seconds after a client is disconnected, but unused channels may cause degradation as more clients are simultaneously subscribed.

Removes a channel

```
supabase.remove_channel(myChannel)
```

Unsubscribe from all channels

> Removing channels is a great way to maintain the performance of your project's Realtime service as well as your database if you're listening to Postgres changes. Supabase will automatically handle cleanup 30 seconds after a client is disconnected, but unused channels may cause degradation as more clients are simultaneously subscribed.

Remove all channels

```
supabase.remove_all_channels()
```

Retrieve all channels

Get all channels

```
channels = supabase.get_channels()
```

Broadcast a message

Broadcast a message to all connected clients to a channel.
Send a message via websocket

```
channel = supabase.channel("room1")

def on_subscribe(status, err):
   if status == RealtimeSubscribeStates.SUBSCRIBED:
       channel.send_broadcast('cursor-pos', {"x": random.random(), "y": random.random()})

channel.subscribe(on_subscribe)
```

Response

Create a bucket

Creates a new Storage bucket

RLS policy permissions required:

- `buckets` table permissions: `insert`
- `objects` table permissions: none

Refer to the Storage guide on how access control works

Parameters

idRequiredstring

A unique identifier for the bucket you are creating.

optionsRequiredCreateOrUpdateBucketOptions

Details

Create bucket

```
response = (
  supabase.storage
  .create_bucket(
    "avatars",
    options={
      "public": False,
      "allowed_mime_types": ["image/png"],
      "file_size_limit": 1024,
    }
  )
)
```

Response

Retrieve a bucket

Retrieves the details of an existing Storage bucket.

- RLS policy permissions required:
  - `buckets` table permissions: `select`
  - `objects` table permissions: none
- Refer to the [Storage guide](#) on how access control works

Parameters

- idRequiredstring
  - The unique identifier of the bucket you would like to retrieve.

Get bucket

```
response = supabase.storage.get_bucket("avatars")
```

Response

List all buckets

Retrieves the details of all Storage buckets within an existing project.

- RLS policy permissions required:
  - `buckets` table permissions: `select`
  - `objects` table permissions: none
- Refer to the [Storage guide](#) on how access control works

List buckets

```
response = supabase.storage.list_buckets()
```

Response

Update a bucket

Updates a Storage bucket

RLS policy permissions required:

- `buckets` table permissions: `select` and `update`
- `objects` table permissions: none

Refer to the Storage guide on how access control works

Parameters

idRequiredstring A unique identifier for the bucket you are creating.
optionsRequiredCreateOrUpdateBucketOptions
Details

Update bucket

```
response = (
  supabase.storage
  .update_bucket(
      "avatars",
        options={
            "public": False,
            "allowed_mime_types": ["image/png"],
            "file_size_limit": 1024,
        }
    )
)
```

Response

Delete a bucket

Deletes an existing bucket. A bucket can't be deleted with existing objects inside it. You must first `empty()` the bucket.

RLS policy permissions required:

- `buckets` table permissions: `select` and `delete`
- `objects` table permissions: none

Refer to the Storage guide on how access control works

Parameters

idRequiredstring
The unique identifier of the bucket you would like to delete.

Delete bucket

```
response = supabase.storage.delete_bucket("avatars")
```

Response

Empty a bucket

Removes all objects inside a single bucket.

RLS policy permissions required:

- `buckets` table permissions: `select`
- `objects` table permissions: `select` and `delete`

Refer to the Storage guide on how access control works

Parameters

idRequiredstring

The unique identifier of the bucket you would like to empty.

Empty bucket

```
response = supabase.storage.empty_bucket("avatars")
```

Response

Upload a file

Uploads a file to an existing bucket.

RLS policy permissions required:

- `buckets` table permissions: none
- `objects` table permissions: only `insert` when you are uploading new files and `select`, `insert` and `update` when you are upserting files

Refer to the Storage guide on how access control works

Please specify the appropriate content MIME type if you are uploading images or audio. If no `file_options` are specified, the MIME type defaults to `text/html`.

Parameters

pathRequiredstring

The file path, including the file name. Should be of the format `folder/subfolder/filename.png`. The bucket must already exist before attempting to upload.

fileRequiredBufferedReader | bytes | FileIO | string | Path
The body of the file to be stored in the bucket.

file_optionsRequiredFileOptions

Details

Upload file using filepath

```
with open("./public/avatar1.png", "rb") as f:
   response = (
       supabase.storage
       .from_("avatars")
       .upload(
           file=f,
           path="public/avatar1.png",
           file_options={"cache-control": "3600", "upsert": "false"}
       )
```

```
 )
```

Response

Download a file

Downloads a file from a private bucket. For public buckets, make a request to the URL returned from `get_public_url` instead.

RLS policy permissions required:
- `buckets` table permissions: none
- `objects` table permissions: `select`

Refer to the Storage guide on how access control works

Parameters

pathRequiredstring
The full path and file name of the file to be downloaded. For example `folder/image.png`.

optionsRequiredDownloadOptions

Details

Download fileDownload file with transformations

```
with open("./myfolder/avatar1.png", "wb+") as f:
   response = (
       supabase.storage
       .from_("avatars")
       .download("folder/avatar1.png")
```

```
  )
  f.write(response)
```

List all files in a bucket

Lists all the files within a bucket.

RLS policy permissions required:
- `buckets` table permissions: none
- `objects` table permissions: `select`

Refer to the [Storage guide](#) on how access control works

Parameters

pathOptionalstring
The folder path.

optionsOptionalSearchOptions

Details

List files in a bucketSearch files in a bucket

```
response = (
  supabase.storage
  .from_("avatars")
  .list(
       "folder",
       {
           "limit": 100,
           "offset": 0,
           "sortBy": {"column": "name", "order": "desc"},
       }
   )
)
```

Response

Replace an existing file

Replaces an existing file at the specified path with a new one.

RLS policy permissions required:
- `buckets` table permissions: none
- `objects` table permissions: `update` and `select`

Refer to the [Storage guide](#) on how access control works

Parameters

pathRequiredstring
The file path, including the file name. Should be of the format `folder/subfolder/filename.png`. The bucket must already exist before attempting to upload.
fileRequiredBufferedReader | bytes | FileIO | string | Path
The body of the file to be stored in the bucket.

file_optionsRequiredFileOptions

Details

Update file

```
with open("./public/avatar1.png", "rb") as f:
    response = (
        supabase.storage
        .from_("avatars")
        .update(
            file=f,
            path="public/avatar1.png",
            file_options={"cache-control": "3600", "upsert": "true"}
```

```
      )
  )
```

Response

Move an existing file

Moves an existing file to a new path in the same bucket.

- RLS policy permissions required:
  - `buckets` table permissions: none
  - `objects` table permissions: `update` and `select`
- Refer to the Storage guide on how access control works

Parameters

from_pathRequiredstring
The original file path, including the current file name. For example `folder/image.png`.

to_pathRequiredstring

The new file path, including the new file name. For example `folder/image-new.png`.

Move file

```
response = (
  supabase.storage
  .from_("avatars")
  .move(
      "public/avatar1.png",
      "private/avatar2.png"
   )
)
```

Response

Copy an existing file

Copies an existing file to a new path in the same bucket.

- RLS policy permissions required:
  - `buckets` table permissions: none
  - `objects` table permissions: `update` and `select`
- Refer to the [Storage guide](https://supabase.com/docs/guides/storage/security/access-control) on how access control works

Parameters

from_pathRequiredstring The original file path, including the current file name. For example `folder/image.png`.
to_pathRequiredstring
The new file path, including the new file name. For example `folder/image-new.png`.

Copy file

```
response = (
   supabase.storage
   .from_("avatars")
   .copy(
       "public/avatar1.png",
       "private/avatar2.png"
   )
)
```

Response

Delete files in a bucket

Deletes files within the same bucket

RLS policy permissions required:
- `buckets` table permissions: none
- `objects` table permissions: `delete` and `select`

Refer to the [Storage guide](#) on how access control works

Parameters

pathsRequiredlist[string]
An array of files to delete, including the path and file name. For example `["folder/image.png"]`.

Delete file

```
response = (
  supabase.storage
  .from_("avatars")
  .remove(["folder/avatar1.png"])
)
```

Response

Create a signed URL

Creates a signed URL for a file. Use a signed URL to share a file for a fixed amount of time.

- RLS policy permissions required:
  - `buckets` table permissions: none
  - `objects` table permissions: `select`
- Refer to the [Storage guide](#) on how access control works

Parameters

pathRequiredstring

The file path, including the file name. For example `"folder/image.png"`.

expires_inRequirednumber

The number of seconds until the signed URL expires. For example, `60` for URLs which are valid for one minute.

optionsOptionalURLOptions

Details

Create Signed URLCreate a signed URL for an asset with transformationsCreate a signed URL which triggers the download of the asset

```
response = (
    supabase.storage
    .from_("avatars")
    .create_signed_url(
        "folder/avatar1.png",
        60
    )
)
```

Response

Create signed URLs

Creates multiple signed URLs. Use a signed URL to share a file for a fixed amount of time.

- RLS policy permissions required:
    - `buckets` table permissions: none
    - `objects` table permissions: `select`
- Refer to the [Storage guide](#) on how access control works

Parameters

- pathsRequiredlist[string]
  The file paths to be downloaded, including the current file names. For example `["folder/image.png", "folder2/image2.png"]`.
- expires_inRequirednumber

The number of seconds until the signed URLs expire. For example, `60` for URLs which are valid for one minute.

options Optional CreateSignedURLsOptions

Details

Create Signed URLs

```
response = (
  supabase.storage
  .from_("avatars")
  .create_signed_urls(
    ["folder/avatar1.png", "folder/avatar2.png"],
    60
  )
)
```

Response

Create signed upload URL

Creates a signed upload URL. Signed upload URLs can be used to upload files to the bucket without further authentication. They are valid for 2 hours.

- RLS policy permissions required:
  - `buckets` table permissions: none
  - `objects` table permissions: `insert`
- Refer to the [Storage guide](#) on how access control works

Parameters

pathRequiredstring	
The file path, including the current file name. For example `"folder/image.png"`.	

Create Signed URL

```
response = (
  supabase.storage
  .from_("avatars")
  .create_signed_upload_url("folder/avatar1.png")
)
```

Response

Upload to a signed URL

Upload a file with a token generated from `create_signed_upload_url`.

- RLS policy permissions required:
  - `buckets` table permissions: none
  - `objects` table permissions: none

Refer to the Storage guide on how access control works

Parameters

pathRequiredstring
The file path, including the file name. Should be of the format `folder/subfolder/filename.png`. The bucket must already exist before attempting to upload.
tokenRequiredstring
The token generated from `create_signed_upload_url`
fileRequiredBufferedReader | bytes | FileIO | string | Path
The body of the file to be stored in the bucket.

optionsRequiredFileOptions

Details

Create Signed URL

```
with open("./public/avatar1.png", "rb") as f:
   response = (
       supabase.storage
       .from_("avatars")
       .upload_to_signed_url(
           path="folder/cat.jpg",
           token="token-from-create_signed_upload_url",
        file=f,
    )
)
```

Response

Retrieve public URL

A simple convenience function to get the URL for an asset in a public bucket. If you do not want to use this function, you can construct the public URL by concatenating the bucket URL with the path to the asset. This function does not verify if the bucket is public. If a public URL is created for a bucket which is not public, you will not be able to download the asset.

The bucket needs to be set to public, either via update_bucket() or by going to Storage on supabase.com/dashboard, clicking the overflow menu on a bucket and choosing "Make public"
RLS policy permissions required:
- `buckets` table permissions: none
- `objects` table permissions: none

Refer to the Storage guide on how access control works

Parameters

pathRequiredstring
The path and name of the file to generate the public URL for. For example `folder/image.png`.