

# Python Client Library

supabase-py 

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.

## Install with PyPi

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

## Installing

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_url` REQUIRED string

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

---

`supabase_key` REQUIRED string

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

---

`options` OPTIONAL ClientOptions

## Initializing

```
+   import os
2   from supabase import create_client, Client
3
4   url: str = os.environ.get("SUPABASE_URL")
5   key: str = os.environ.get("SUPABASE_KEY")
6   supabase: Client = create_client(url, key)
```

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

---

`columns` OPTIONAL string

The columns to retrieve, defaults to `*`.

## Fetch data

[Filtering through referenced tables](#)    [Querying referenced table with count](#)    [Querying with count option](#)

[Querying JSON data](#)    [Querying referenced table with inner join](#)    [Switching schemas per query](#)

```
1  response = (
2      supabase.table("planets")
3      .select("*")
4      .execute()
5  )
```

> Data source

> Response

---

## Parameters

---

json REQUIRED dict, list

The values to insert. Pass an dict to insert a single row or an list to insert multiple rows.

---

count OPTIONAL CountMethod

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

---

returning OPTIONAL ReturnMethod

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

---

↳ [View source](#)

Python Reference v2.0

---

```
1     response = (
2         supabase.table("planets")
3             .insert({"id": 1, "name": "Pluto"})
4             .execute()
5     )
```

> Data source

> Response

---

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

## Parameters

---

`json` REQUIRED dict, list

## Update data

```
1  response = (
2      supabase.table("instruments")
3          .update({ "name": "piano" })
4          .eq("id", 1)
5          .execute()
6  )
```

> Data source

> Response

## Upsert data

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

## Parameters

---

`json` REQUIRED dict, list

The values to insert. Pass an dict to insert a single row or an list to insert multiple rows.

---

`count` OPTIONAL CountMethod

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

---

`returning` OPTIONAL ReturnMethod

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

---

`ignore_duplicates` OPTIONAL bool

Whether duplicate rows should be ignored.

---

`on_conflict` OPTIONAL string

Specified columns to be made to work with UNIQUE constraint.

---

`default_to_null` OPTIONAL bool

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

---

[Upsert your data](#)   [Bulk Upsert your data](#)   [Upserting into tables with constraints](#)

```
1   response = (
2       supabase.table("instruments")
3       .upsert({"id": 1, "name": "piano"})
4       .execute()
5   )
```

> 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` 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

`count` (OPTIONAL) `CountMethod`

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

`returning` (OPTIONAL) `ReturnMethod`

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

`Delete records`    `Delete multiple records`

```
1   response = (
2     supabase.table("countries")
3     .delete()
4     .eq("id", 1)
5     .execute()
6   )
```

> 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.

```
1  create or replace function hello_world() returns text as $$  
2      select 'Hello world';  
3  $$ language sql;
```

## Parameters

`fn` REQUIRED callable

The stored procedure call to be executed.

`params` OPTIONAL dict of any

Parameters passed into the stored procedure call.

`get` OPTIONAL dict of any

When set to `true`, `data` will not be returned. Useful if you only need the count.

`head` OPTIONAL dict of any

When set to `true`, the function will be called with read-only access mode.

`count` OPTIONAL CountMethod

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 arguments

Call a Postgres function with arguments

Bulk processing

Call a Postgres function with filters

Call a read-only Postgres function

```
1   response = (
2     supabase.rpc("hello_world")
3     .execute()
4   )
```

> 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 Filters

Chaining

Conditional chaining

Filter by values within JSON column

Filter Foreign Tables

```
1  # Correct
2  response = (
3    supabase.table("instruments")
4    .select("name, section_id")
5    .eq("name", "flute")
6    .execute()
7  )
```

```
8
9  # Incorrect
10 response = (
11     supabase.table("instruments")
12     .eq("name", "flute")
13     .select("name, section_id")
14     .execute()
15 )
```

> Data source

> Notes

## Column is equal to a value

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

### Parameters

`column` REQUIRED string

The column to filter on

`value` REQUIRED any

The value to filter by

With `select()`

```
1  response = (
2      supabase.table("planets")
3      .select("*")
4      .eq("name", "Earth")
5      .execute()
```

```
6      )
```

> Data source

> Response

## Column is not equal to a value

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

### Parameters

`column` REQUIRED string

The column to filter on

`value` REQUIRED any

The value to filter by

With `select()`

```
1  response = (
2      supabase.table("planets")
3      .select("*")
4      .neq("name", "Earth")
5      .execute()
6  )
```

> Data source

> Response

# Column is greater than a value

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

## Parameters

`column` REQUIRED string

The column to filter on

`value` REQUIRED any

The value to filter by

With `select()`

```
1  response = (
2      supabase.table("planets")
3          .select("*")
4          .gt("id", 2)
5          .execute()
6      )
```

> 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

---

`column` REQUIRED string

The column to filter on

---

`value` REQUIRED any

The value to filter by

---

With `select()`

```
1 response = (
2     supabase.table("planets")
3     .select("*")
4     .gte("id", 2)
5     .execute()
6 )
```

> Data source

> Response

---

## Column is less than a value

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

## Parameters

---

column REQUIRED string

The column to filter on

---

value REQUIRED any

The value to filter by

---

With `select()`

```
1   response = (
2       supabase.table("planets")
3           .select("*")
4           .lt("id", 2)
5           .execute()
6   )
```

> 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

---

column REQUIRED string

The column to filter on

---

value REQUIRED any

The value to filter by

---

With `select()`

```
1   response = (
2     supabase.table("planets")
3       .select("*")
4       .lte("id", 2)
5       .execute()
6   )
```

> Data source

> Response

---

## Column matches a pattern

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

---

### Parameters

`column` REQUIRED string

The name of the column to apply a filter on

---

`pattern` REQUIRED string

The pattern to match by

---

With `select()`

```
1   response = (
```

```
2     supabase.table("planets")
3       .select("*")
4       .like("name", "%Ea%")
5       .execute()
6   )
```

> Data source

> Response

## Column matches a case-insensitive pattern

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

### Parameters

`column` REQUIRED string

The name of the column to apply a filter on

`pattern` REQUIRED string

The pattern to match by

With `select()`

```
1   response = (
2     supabase.table("planets")
3       .select("*")
4       .ilike("name", "%ea%")
5       .execute()
6   )
```

> Data source

> Response

## Column is a value

Match only rows where `column IS value`.

### Parameters

`column` REQUIRED string

The name of the column to apply a filter on

`value` REQUIRED null | boolean

The value to match by

Checking for nullness, True or False

```
1 response = (
2     supabase.table("planets")
3     .select("*")
4     .is_("name", "null")
5     .execute()
6 )
```

> 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()`

```
1  response = (
2      supabase.table("planets")
3      .select("*")
4      .in_("name", ["Earth", "Mars"])
5      .execute()
6  )
```

> 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

---

`column` REQUIRED string

The column to filter on

---

`values` REQUIRED object

The jsonb, array, or range value to filter with

---

On array columns

On range columns

On `jsonb` columns

```
1  response = (
2      supabase.table("issues")
3      .select("*")
4      .contains("tags", ["is:open", "priority:low"])
5      .execute()
6  )
```

> 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

---

column REQUIRED string

The jsonb, array, or range column to filter on

---

value REQUIRED object

The jsonb, array, or range value to filter with

---

On array columns

On range columns

On `jsonb` columns

```
1  response = (
2      supabase.table("classes")
3          .select("name")
4          .contained_by("days", ["monday", "tuesday", "wednesday", "friday"])
5          .execute()
6      )
```

> 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()`

```
1  response = (
2      supabase.table("reservations")
3      .select("*")
4      .range_gt("during", ["2000-01-02 08:00", "2000-01-02 09:00"])
5      .execute()
6  )
```

> 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

column REQUIRED string

The range column to filter on

---

range REQUIRED string

The range to filter with

---

With `select()`

```
1  response = (
2      supabase.table("reservations")
3      .select("*")
4      .range_gte("during", ["2000-01-02 08:30", "2000-01-02 09:30"])
5      .execute()
6  )
```

> 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

`column` REQUIRED string

The range column to filter on

`range` REQUIRED array

The range to filter with

With `select()`

```
1  response = (
2      supabase.table("reservations")
3      .select("*")
4      .range_lt("during", ["2000-01-01 15:00", "2000-01-01 16:00"])
5      .execute()
6  )
```

> 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

`column` REQUIRED string

The range column to filter on

`range` REQUIRED array

The range to filter with

With `select()`

```
1  response = (
2      supabase.table("reservations")
3      .select("*")
```

```
4     .range_lte("during", ["2000-01-01 14:00", "2000-01-01 16:00"])
5     .execute()
6 )
```

> 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

`column` REQUIRED string

The range column to filter on

`range` REQUIRED array

The range to filter with

With `select()`

```
1 response = (
2     supabase.table("reservations")
3     .select("*")
4     .range_adjacent("during", ["2000-01-01 12:00", "2000-01-01 13:00"])
5     .execute()
6 )
```

> 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

`column` REQUIRED string

The array or range column to filter on

`value` REQUIRED Iterable[Any]

The array or range value to filter with

On array columns

On range columns

```
1  response = (
2      supabase.table("issues")
3      .select("title")
4      .overlaps("tags", ["is:closed", "severity:high"])
5      .execute()
6  )
```

> 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

`column` REQUIRED string

The text or tsvector column to filter on

`query` REQUIRED string

The query text to match with

`options` OPTIONAL object

Named parameters

+ Details

Text search

Basic normalization

Full normalization

Websearch

```
1  response = (
2      supabase.table("texts")
3          .select("content")
4          .text_search(
5              "content",
6              "'eggs' & 'ham'",
7              options={"config": "english"},
8          )
9          .execute()
10     )
```

> 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

`query` REQUIRED dict

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

With `select()`

```
1 response = (
2     supabase.table("planets")
3     .select("*")
4     .match({"id": 2, "name": "Earth"})
5     .execute()
6 )
```

> 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.

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

With `select()`

```
1 response = (
2     supabase.table("planets")
3     .select("*")
4     .not_.is_("name", "null")
5     .execute()
6 )
```

> Data source

> Response

## Match at least one filter

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

```
1 .or_('id.in.(5,6,7), arraycol.cs>{"a","b"}') # Use `()` for `in`  
2 .or_('id.in.(5,6,7), arraycol.cd>{"a","b"}') # Use `cd` for `contains`
```

## Parameters

`filters` REQUIRED string

The filters to use, following PostgREST syntax

`reference_table` OPTIONAL string

Set this to filter on referenced tables instead of the parent table

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

```
1 response = (  
2     supabase.table("planets")  
3     .select("name")  
4     .or_("id.eq.2,name.eq.Mars")  
5     .execute()  
6 )
```

> Data source

> Response

## Match the filter

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

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

## Parameters

column REQUIRED string

The column to filter on

operator OPTIONAL string

The operator to filter with, following PostgREST syntax

value OPTIONAL any

The value to filter with, following PostgREST syntax

With `select()`      On a foreign table

```
1   response = (
2       supabase.table("planets")
3       .select("*")
4       .filter("name", "in", '("Mars", "Tatooine")')
5       .execute()
6   )
```

> 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

---

`column` REQUIRED string

The column to order by

---

`desc` OPTIONAL bool

Whether the rows should be ordered in descending order or not.

---

`foreign_table` OPTIONAL string

Foreign table name whose results are to be ordered.

---

`nullsfirst` OPTIONAL bool

Order by showing nulls first

---

```
1 response = (
2     supabase.table("planets")
3     .select("*")
4     .order("name", desc=True)
5     .execute()
6 )
```

> Data source

> Response

## Limit the number of rows returned

### Parameters

**size** REQUIRED number

The maximum number of rows to return

**foreign\_table** OPTIONAL string

Set this to limit rows of foreign tables instead of the parent table.

With `select()`

On a foreign table

```
1 response = (
2     supabase.table("planets")
3     .select("name")
4     .limit(1)
5     .execute()
6 )
```

> 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

`start` REQUIRED number

The starting index from which to limit the result.

`end` REQUIRED number

The last index to which to limit the result.

`foreign_table` OPTIONAL string

Set this to limit rows of foreign tables instead of the parent table.

With `select()`

On a foreign table

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

```
6      )
```

> Data source

> Response

## Retrieve one row of data

Return `data` as a single object instead of an array of objects.

With `select()`

```
1  response = (
2      supabase.table("planets")
3      .select("name")
4      .limit(1)
5      .single()
6      .execute()
7  )
```

> Data source

> Response

## Retrieve zero or one row of data

Return `data` as a single object instead of an array of objects.

With `select()`

```
1  response = (
2      supabase.table("planets")
3      .select("*")
4      .eq("name", "Earth")
5      .maybe_single()
6      .execute()
7  )
```

> Data source

> Response

## Retrieve as a CSV

Return `data` as a string in CSV format.

Return data as CSV

```
1  response = (
2      supabase.table("planets")
3      .select("*")
4      .csv()
5      .execute()
6  )
```

> 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

`wal` OPTIONAL boolean

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

`verbose` OPTIONAL boolean

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

`settings` OPTIONAL boolean

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

`format` OPTIONAL boolean

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

`format` OPTIONAL `"text" | "json"`

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

**buffers** OPTIONAL boolean

If true, include information on buffer usage.

**analyze** OPTIONAL boolean

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

Get the execution plan

Get the execution plan with analyze and verbose

```
1   response = (
2     supabase.table("planets")
3       .select("*")
4       .explain()
5       .execute()
6   )
```

> 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

---

`credentials` REQUIRED `SignUpWithPasswordCredentials`

+ Details

[Sign up with an email and password](#)

[Sign up with a phone number and password \(SMS\)](#)

[Sign up with a phone number and password \(whatsapp\)](#)

[Sign up with additional user metadata](#)

[Sign up with a redirect URL](#)

```
1   response = supabase.auth.sign_up(  
2       {  
3           "email": "email@example.com",  
4           "password": "password",  
5       }  
6   )
```

> 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

`credentials` REQUIRED `SignInAnonymouslyCredentials`

Details

[Create an anonymous user](#)

[Create an anonymous user with custom user metadata](#)

```
1   response = supabase.auth.sign_in_anonymously(  
2       {"options": {"captcha_token": ""}}  
3   )
```

> 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

credentials SignInWithPasswordCredentials

 Details

[Sign in with email and password](#)

[Sign in with phone and password](#)

```
1   response = supabase.auth.sign_in_with_password(  
2       {  
3           "email": "email@example.com",  
4           "password": "example-password",  
5       }  
6   )
```

> Response

# Sign in with ID token (native sign-in)

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

## Parameters

---

credentials SignInWithIdTokenCredentials

+ Details

Sign In using ID Token

```
1 response = supabase.auth.sign_in_with_id_token(  
2     {  
3         "provider": "google",  
4         "token": "your-id-token",  
5     }  
6 )
```

> 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

`credentials` `SignInWithPasswordCredentials`

 Details

[Sign in with email](#) [Sign in with SMS OTP](#) [Sign in with WhatsApp OTP](#)

```
1   response = supabase.auth.sign_in_with_otp(
2     {
3       "email": "email@example.com",
4       "options": {
5         "email_redirect_to": "https://example.com/welcome",
6       },
7     }
8   )
```

> 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

`credentials` REQUIRED `SignInWithOAuthCredentials`

 Details

Sign in using a third-party provider

Sign in using a third-party provider with redirect

Sign in with scopes

```
1   response = supabase.auth.sign_in_with_oauth(  
2       {"provider": "github"}  
3   )
```

## 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 identify the identity provider to be used by the user, like a company-specific page, IP address or other tracking information.

### Parameters

params SignInWithSSOCredentials

 Details

Sign in with email domain

Sign in with provider UUID

```
1     response = supabase.auth.sign_in_with_sso(  
2         {"domain": "company.com"}  
3     )
```

> Response

> Notes

## Get user claims from verified JWT

Extracts the JWT claims present in the access token by first verifying the JWT against the server's JSON Web Key Set endpoint `/well-known/jwks.json` which is often cached, resulting in significantly faster responses. Prefer this method over `get_user` which always sends a request to the Auth server for each JWT.

### Parameters

`jwt` OPTIONAL string

An optional specific JWT you wish to verify, not the one you can obtain from `get_session`.

`jwks` OPTIONAL JWKSet

If set, this JSON Web Key Set is going to have precedence over the cached value available on the server.

 Details

Get user object

```
1     response = supabase.auth.get_claims()
```

## 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

`options` OPTIONAL `SignInOptions`

+ Details

Sign out

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

## Send a password reset request

The password reset flow consists 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()`:

```
1  response = supabase.auth.update_user(  
2      {"password": new_password}  
3  )
```

## Parameters

`email` REQUIRED string

The email address of the user.

`options` OPTIONAL object

Details

Reset password

```
1  supabase.auth.reset_password_for_email(  
2      email,  
3      {  
4          "redirect_to": "https://example.com/update-password",  
5      }  
6  )
```

# 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

params VerifyOtpParams

 Details

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

```
1   response = supabase.auth.verify_otp(  
2       {  
3           "email": "email@example.com",  
4           "token": "123456",  
5           "type": "email",  
6       }  
7   )
```

> 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

```
1     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_token` OPTIONAL string

Refresh session using the current session

```
1   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

**jwt** OPTIONAL string

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 session

Get the logged in user with a custom access token jwt

```
1   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](#).

## Parameters

**attributes** REQUIRED UserAttributes

[+ Details](#)

**options** OPTIONAL object

[+ Details](#)

[Update the email for an authenticated user](#)

[Update the phone number for an authenticated user](#)

[Update the password for an authenticated user](#)

[Update the user's metadata](#)

[Update the user's password with a nonce](#)

```
1   response = supabase.auth.update_user(  
2       {"email": "new@email.com"}  
3   )
```

> 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

```
1   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

`credentials` REQUIRED `SignInWithOAuthCredentials`

+ Details

Link an identity to a user

```
1   response = supabase.auth.link_identity(
```

```
2     {provider: "github"}  
3 )
```

> 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

```
1 # retrieve all identities linked to a user  
2 response = supabase.auth.get_user_identities()  
3  
4 # find the google identity  
5 google_identity = list(  
6     filter(lambda identity: identity.provider == "google", res.identities)  
7 ).pop()  
8  
9 # unlink the google identity  
10 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 required 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

```
1   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

credentials REQUIRED ResendCredentials

Details

Resend an email signup confirmation

Resend a phone signup confirmation

Resend email change email

Resend phone change OTP

```
1   response = supabase.auth.resend(  
2       {  
3           "type": "signup",  
4           "email": "email@example.com",  
5           "options": {  
6               "email_redirect_to": "https://example.com/welcome",  
7           },  
8       }  
9   )
```

> 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_token` REQUIRED string

`refresh_token` REQUIRED string

Refresh the session

```
1   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_code` REQUIRED string

Exchange Auth Code

```
1     response = supabase.auth.exchange_code_for_session(  
2         {"auth_code": "34e770dd-9ff9-416c-87fa-43b31d7ef225"}  
3     )
```

> 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

```
1   response = supabase.auth.mfa.enroll(  
2       {  
3           "factor_type": "totp",  
4           "friendly_name": "your_friendly_name",  
5       }  
6   )
```

## 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

```
1   response = supabase.auth.mfa.challenge(  
2       {"factor_id": "34e770dd-9ff9-416c-87fa-43b31d7ef225"}  
3   )
```

## Verify a challenge

To verify a challenge, please [create a challenge](#) first.

Verify a challenge for a factor

```
1 response = supabase.auth.mfa.verify(
2     {
3         "factor_id": "34e770dd-9ff9-416c-87fa-43b31d7ef225",
4         "challenge_id": "4034ae6f-a8ce-4fb5-8ee5-69a5863a7c15",
5         "code": "123456",
6     }
7 )
```

## 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

```
1 response = supabase.auth.mfa.challenge_and_verify(
2     {
3         "factor_id": "34e770dd-9ff9-416c-87fa-43b31d7ef225",
4         "code": "123456",
5     }
6 )
```

## Unenroll a factor

Unenroll a factor

```
1 response = supabase.auth.mfa.unenroll(  
2     {"factor_id": "34e770dd-9ff9-416c-87fa-43b31d7ef225"}  
3 )
```

## 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

```
1 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

```
1  from supabase import create_client
2  from supabase.lib.client_options import ClientOptions
3
4  supabase = create_client(
5      supabase_url,
6      service_role_key,
7      options=ClientOptions(
8          auto_refresh_token=False,
9          persist_session=False,
10     )
11    )
12
13 # Access auth admin api
14 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

```
1   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 users

Paginated list of users

```
1   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 metadata

Auto-confirm the user's email

Auto-confirm the user's phone number

```
1   response = supabase.auth.admin.create_user(
2       {
3           "email": "user@email.com",
4           "password": "password",
5           "user_metadata": {"name": "Yoda"},
6       }
7   )
```

> 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

`id` REQUIRED string

The user id you want to remove.

`should_soft_delete` OPTIONAL boolean

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

```
1 supabase.auth.admin.delete_user(  
2     "715ed5db-f090-4b8c-a067-640ecee36aa0"  
3 )
```

# 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

---

`email` REQUIRED string

The email address of the user.

---

`options` OPTIONAL InviteUserByEmailOptions

Details

---

Invite a user

```
1     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

params GenerateLinkParams

 Details

[Generate a signup link](#)   [Generate an invite link](#)   [Generate a magic link](#)   [Generate a recovery link](#)

Generate links to change current email address

```
1   response = supabase.auth.admin.generate_link(  
2       {  
3           "type": "signup",  
4           "email": "email@example.com",  
5           "password": "secret",  
6       }  
7   )
```

> Response

## Update a user

## Parameters

uid REQUIRED string

attributes REQUIRED AdminUserAttributes

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 email

Updates a user's password

Updates a user's metadata

Updates a user's app\_metadata

Confirms a user's email address

Confirms a user's phone number

```
1   response = supabase.auth.admin.update_user_by_id(  
2       "11111111-1111-1111-1111-111111111111",  
3       {  
4           "email": "new@email.com",  
5       }  
6   )
```

> 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

params REQUIRED AuthMFAAdminDeleteFactorParams

Details

Delete a factor for a user

```
1   response = supabase.auth.admin.mfa.delete_factor(  
2     {  
3       "id": "34e770dd-9ff9-416c-87fa-43b31d7ef225",  
4       "user_id": "a89baba7-b1b7-440f-b4bb-91026967f66b"  
5     }  
6   )
```

> Response

## OAuth Admin

OAuth 2.1 client administration endpoints are available via the `supabase.auth.admin.oauth` namespace.

These methods require a `service_role` key and should only be called server-side.

Only relevant when the OAuth 2.1 server is enabled in Supabase Auth.

This is an alpha feature and may change in the future.

## List OAuth clients

Lists OAuth clients with optional pagination.

Requires `service_role` key.

Only available when OAuth 2.1 server is enabled.

This is an alpha feature and may change in the future.

## Parameters

---

params OPTIONAL PageParams

Optional pagination parameters.

Details

---

[List OAuth clients](#)

[List OAuth clients with pagination](#)

```
1     response = supabase.auth.admin.oauth.list_clients()
```

## Get OAuth client

Retrieves details of an OAuth client by ID.

Requires `service_role` key.

Only available when OAuth 2.1 server is enabled.

This is an alpha feature and may change in the future.

## Parameters

---

`client_id` REQUIRED string

The ID of the OAuth client to retrieve.

---

[Get OAuth client](#)

```
1     response = supabase.auth.admin.oauth.get_client("client-id")
```

## Create OAuth client

Creates a new OAuth client.

Requires `service_role` key.

Only available when OAuth 2.1 server is enabled.

This is an alpha feature and may change in the future.

### Parameters

`params` REQUIRED `CreateOAuthClientParams`

Parameters for creating a new OAuth client.

+ Details

Create OAuth client

```
1     response = supabase.auth.admin.oauth.create_client(
2         {
3             "name": "My OAuth Client",
4             "redirect_uris": ["https://example.com/callback"]
5         }
6     )
```

# Update OAuth client

Updates an existing OAuth client.

Requires `service_role` key.

Only available when OAuth 2.1 server is enabled.

This is an alpha feature and may change in the future.

## Parameters

---

`client_id` REQUIRED string

The ID of the OAuth client to update.

---

`params` REQUIRED UpdateOAuthClientParams

Parameters for updating the OAuth client.

+ Details

Update OAuth client

```
1  response = supabase.auth.admin.oauth.update_client(
2      "client-id",
3      {
4          "name": "Updated OAuth Client",
5          "redirect_uris": ["https://example.com/callback", "https://example.com/cal
6      }
7  )
```

# Delete OAuth client

Deletes an OAuth client.

Requires `service_role` key.

Only available when OAuth 2.1 server is enabled.

This is an alpha feature and may change in the future.

## Parameters

---

`client_id` REQUIRED string

The ID of the OAuth client to delete.

---

Delete OAuth client

```
1 supabase.auth.admin.oauth.delete_client("client-id")
```

## Regenerate client secret

Regenerates the client secret for an OAuth client.

Requires `service_role` key.

Only available when OAuth 2.1 server is enabled.

This is an alpha feature and may change in the future.

## Parameters

---

`client_id` REQUIRED string

The ID of the OAuth client.

---

[Regenerate OAuth client secret](#)

```
1   response = supabase.auth.admin.oauth.regenerate_client_secret("client-id")
```

## 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 invocation](#)    [Error handling](#)    [Passing custom headers](#)

```
1   response = supabase.functions.invoke(
2     "hello-world",
3     invoke_options={
4       "body": {"name": "Functions"},
5     },
6   )
```

# Overview

Realtime in Python only works with the asynchronous client. You can initialize a new Supabase client using the `acreate_client()` method.

Some Realtime methods are asynchronous and must be awaited. Ensure these methods are called within an `async` function.

In the following Realtime examples, certain methods are awaited. These should be enclosed within an `async` function.

When an asynchronous method needs to be used within a synchronous context, such as the callback for `.subscribe()`, utilize `asyncio.create_task()` to schedule the coroutine. This is why the `acreate_client` example includes an import of `asyncio`.

`aclient()`

```
1  import os
2  import asyncio
3  from supabase import acreate_client, AsyncClient
4
5  url: str = os.environ.get("SUPABASE_URL")
6  key: str = os.environ.get("SUPABASE_KEY")
7
8  async def create_supabase():
9      supabase: AsyncClient = await acreate_client(url, key)
10     return supabase
```

## Subscribe to channel

By default, Broadcast and Presence are enabled for all projects.

Presence is automatically enabled when you attach presence callbacks (`on_presence_sync()`, `on_presence_join()`, or `on_presence_leave()`). If you add presence callbacks to an already joined channel, the channel will automatically resubscribe with presence enabled.

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 messages    Listen to presence sync    Listen to presence join    Listen to presence leave  
Listen to all database changes    Listen to a specific table    Listen to inserts    Listen to updates  
Listen to deletes    Listen to multiple events    Listen to row level changes

```
1  channel = supabase.channel("room1")
2
3  def on_subscribe(status, err):
4      if status == RealtimeSubscribeStates.SUBSCRIBED:
5          asyncio.create_task(channel.send_broadcast(
6              "cursor-pos",
7              {"x": random.random(), "y": random.random()})
8      )
9
10 def handle_broadcast(payload):
11     print("Cursor position received!", payload)
12
13 await channel.on_broadcast(event="cursor-pos", callback=handle_broadcast).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

```
1 await 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

```
1 await supabase.remove_all_channels()
```

## Retrieve all channels

Get all channels

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

## Broadcast a message

Broadcast a message to all connected clients to a channel.

Send a message via websocket

```
1   channel = supabase.channel("room1")
2
3   def on_subscribe(status, err):
4       if status == RealtimeSubscribeStates.SUBSCRIBED:
5           asyncio.create_task(channel.send_broadcast('cursor-pos', {"x": random.rand
6
7   await channel.subscribe(on_subscribe)
```

> Response

## File Buckets

This section contains methods for working with File Buckets.

---

## 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

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

> 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

---

`id` REQUIRED string

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

---

Get bucket

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

> 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

---

**id** REQUIRED string

A unique identifier for the bucket you are creating.

**options** REQUIRED CreateOrUpdateBucketOptions

+ Details

Create bucket

```
1  response = (
2      supabase.storage
3          .create_bucket(
4              "avatars",
5              options={
6                  "public": False,
7                  "allowed_mime_types": ["image/png"],
8                  "file_size_limit": 1024,
9              }
10         )
11     )
```

> 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

---

`id` REQUIRED string

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

---

Empty bucket

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

> 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

---

`id` REQUIRED string

A unique identifier for the bucket you are creating.

---

`options` REQUIRED CreateOrUpdateBucketOptions

Details

## Update bucket

```
1  response = (
2      supabase.storage
3          .update_bucket(
4              "avatars",
5                  options={
6                  "public": False,
7                  "allowed_mime_types": ["image/png"],
8                  "file_size_limit": 1024,
9              }
10         )
11     )
```

> 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

`id` REQUIRED string

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

## Delete bucket

```
1     response = supabase.storage.delete_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

path REQUIRED string

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.

file REQUIRED BufferedReader | bytes | FileIO | string | Path

The body of the file to be stored in the bucket.

file\_options REQUIRED FileOptions

Details

## Upload file using filepath

```
1  with open("./public/avatar1.png", "rb") as f:
2      response = (
3          supabase.storage
4          .from_("avatars")
5          .upload(
6              file=f,
7              path="public/avatar1.png",
8              file_options={"cache-control": "3600", "upsert": "false"}
9          )
10     )
```

> 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

path REQUIRED string

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.

file REQUIRED BufferedReader | bytes | FileIO | string | Path

The body of the file to be stored in the bucket.

file\_options REQUIRED FileOptions

Details

Update file

```
1  with open("./public/avatar1.png", "rb") as f:
2      response = (
3          supabase.storage
4              .from_("avatars")
5              .update(
6                  file=f,
7                  path="public/avatar1.png",
8                  file_options={"cache-control": "3600", "upsert": "true"})
9              )
10         )
```

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_path` REQUIRED string

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

`to_path` REQUIRED string

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

Move file

```
1  response = (
2      supabase.storage
3      .from_("avatars")
4      .move(
5          "public/avatar1.png",
6          "private/avatar2.png"
7      )
8  )
```

> 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](#) on how access control works

### Parameters

`from_path` REQUIRED string

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

`to_path` REQUIRED string

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

`Copy file`

```
1  response = (
2      supabase.storage
3          .from_("avatars")
4          .copy(
5              "public/avatar1.png",
6              "private/avatar2.png"
7          )
8      )
```

> 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

`path` REQUIRED string

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

`expires_in` REQUIRED number

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

`options` OPTIONAL URLOptions

+ Details

[Create Signed URL](#)    [Create a signed URL for an asset with transformations](#)

Create a signed URL which triggers the download of the asset

```
1   response = (
2     supabase.storage
3       .from_("avatars")
4       .create_signed_url(
5         "folder/avatar1.png",
6         60
7       )
8   )
```

> 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

---

paths REQUIRED list[string]

The file paths to be downloaded, including the current file names. For example

[ "folder/image.png", "folder2/image2.png" ] .

---

expires\_in REQUIRED number

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

```
1  response = (
2      supabase.storage
3          .from_("avatars")
4          .create_signed_urls(
5              ["folder/avatar1.png", "folder/avatar2.png"],
6              60
7          )
8      )
```

> 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

path REQUIRED string

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

options OPTIONAL CreateSignedUploadUrlOptions

Additional options for the upload URL creation.

Details

[Create Signed URL](#)

[Create Signed URL with upsert option](#)

```
1 response = (
2     supabase.storage
3     .from_("avatars")
4     .create_signed_upload_url("folder/avatar1.png")
5 )
```

> 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

path REQUIRED string

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.

token REQUIRED string

The token generated from `create_signed_upload_url`

file REQUIRED BufferedReader | bytes | FileIO | string | Path

The body of the file to be stored in the bucket.

options REQUIRED UploadSignedUrlFileOptions

Details

Create Signed URL

```
1  with open("./public/avatar1.png", "rb") as f:
2      response = (
3          supabase.storage
4              .from_("avatars")
5              .upload_to_signed_url(
6                  path="folder/cat.jpg",
7                  token="token-from-create_signed_upload_url",
8                  file=f,
9              )
10         )
```

> 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

`path` REQUIRED string

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

`options` OPTIONAL `URLOptions`

Details

Returns the URL for an asset in a public bucket

Returns the URL for an asset in a public bucket with transformations

Returns the URL which triggers the download of an asset in a public bucket

```
1  response = (
2      supabase.storage
3      .from_("avatars")
4      .get_public_url("folder/avatar1.jpg")
5  )
```

> 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

`path` REQUIRED string

The full path and file name of the file to be downloaded. For example

`folder/image.png`.

`options` REQUIRED DownloadOptions

Details

[Download file](#)

[Download file with transformations](#)

```
1  with open("./myfolder/avatar1.png", "wb+") as f:  
2      response = (  
3          supabase.storage  
4              .from_("avatars")  
5              .download("folder/avatar1.png")  
6      )  
7      f.write(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

---

`paths` REQUIRED list[string]

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

---

Delete file

```
1  response = (
2      supabase.storage
3      .from_("avatars")
4      .remove(["folder/avatar1.png"])
5  )
```

> 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

path OPTIONAL string

The folder path.

options OPTIONAL SearchOptions

Details

List files in a bucket

Search files in a bucket

```
1 response = (
2     supabase.storage
3         .from_("avatars")
4         .list(
5             "folder",
6             {
7                 "limit": 100,
8                 "offset": 0,
9                 "sortBy": {"column": "name", "order": "desc"},
10            }
11        )
12    )
```

> Response

## Analytics Buckets

This section contains methods for working with Analytics Buckets.

Analytics buckets are accessed via the `supabase.storage.analytics()` namespace.

Analytics buckets are used for storing and querying analytics data.

---

## Create a new analytics bucket

Creates a new Analytics bucket.

This is an alpha feature and may change in the future.

Analytics buckets are used for storing and querying analytics data.

### Parameters

---

`bucket_name` REQUIRED string

The name of the analytics bucket to create.

---

Create analytics bucket

```
1   response = supabase.storage.analytics().create("analytics-bucket")
```

> Response

# List analytics buckets

Lists all Analytics buckets with optional filtering and sorting.

This is an alpha feature and may change in the future.

## Parameters

---

`limit` (OPTIONAL) number

Maximum number of buckets to return.

---

`offset` (OPTIONAL) number

Number of buckets to skip.

---

`sort_column` (OPTIONAL) string

Column to sort by. Options: `id`, `name`, `created_at`, `updated_at`.

---

`sort_order` (OPTIONAL) string

Sort order. Options: `asc`, `desc`.

---

`search` (OPTIONAL) string

Search term to filter buckets by name.

---

List analytics buckets

```
1   response = supabase.storage.analytics().list()
```

> Response

# Delete an analytics bucket

Deletes an Analytics bucket.

This is an alpha feature and may change in the future.

## Parameters

---

bucket\_name REQUIRED string

The name of the analytics bucket to delete.

---

Delete analytics bucket

```
1   response = supabase.storage.analytics().delete("analytics-bucket")
```

> Response

---

# Vector Buckets

This section contains methods for working with Vector Buckets.

---

# Create a vector bucket

Creates a new Vector bucket.

This is an alpha feature and may change in the future.

Vector buckets are used for storing and querying vector embeddings.

## Parameters

---

bucket\_name REQUIRED string

The name of the vector bucket to create.

---

Create vector bucket

```
1 supabase.storage.vectors().create_bucket("vectors-bucket")
```

# Delete a vector bucket

Deletes a Vector bucket.

This is an alpha feature and may change in the future.

## Parameters

---

bucket\_name REQUIRED string

The name of the vector bucket to delete.

---

Delete vector bucket

```
1 supabase.storage.vectors().delete_bucket("vectors-bucket")
```

## Retrieve a vector bucket

Retrieves details of a Vector bucket.

This is an alpha feature and may change in the future.

### Parameters

---

bucket\_name REQUIRED string

The name of the vector bucket to retrieve.

---

Get vector bucket

```
1 response = supabase.storage.vectors().get_bucket("vectors-bucket")
```

> Response

# List all vector buckets

Lists all Vector buckets with optional pagination.

This is an alpha feature and may change in the future.

## Parameters

---

`prefix` OPTIONAL string

Filter buckets by name prefix.

---

`max_results` OPTIONAL number

Maximum number of buckets to return.

---

`next_token` OPTIONAL string

Token for pagination.

---

List vector buckets

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

```
> Response
```

---

# Create a vector index

Creates a new vector index in a vector bucket.

This is an alpha feature and may change in the future.

## Parameters

---

`bucket_name` REQUIRED string

The name of the vector bucket.

---

`index_name` REQUIRED string

The name of the index to create.

---

`dimension` REQUIRED number

The dimension of the vectors in this index.

---

`distance_metric` REQUIRED string

The distance metric to use. Options: `cosine`, `euclidean`.

---

`data_type` REQUIRED string

The data type of the vectors.

---

`metadata` OPTIONAL MetadataConfiguration

Optional metadata configuration for the index.

---

Create vector index

```
1 supabase.storage.vectors().from_("vectors-bucket").create_index(  
2     index_name="my-index",  
3     dimension=128,  
4     distance_metric="cosine",  
5     data_type="float32"  
6 )
```

# Delete a vector index

Deletes a vector index.

This is an alpha feature and may change in the future.

## Parameters

---

`bucket_name` REQUIRED string

The name of the vector bucket.

---

`index_name` REQUIRED string

The name of the index to delete.

---

Delete vector index

```
1 supabase.storage.vectors().from_("vectors-bucket").delete_index("my-index")
```

---

# Retrieve a vector index

Retrieves details of a vector index.

This is an alpha feature and may change in the future.

## Parameters

---

`bucket_name` REQUIRED string

The name of the vector bucket.

---

`index_name` REQUIRED string

The name of the index to retrieve.

---

Get vector index

```
1     response = supabase.storage.vectors().from_("vectors-bucket").get_index("my-index")
```

## List all vector indexes

Lists all indexes in a vector bucket.

This is an alpha feature and may change in the future.

### Parameters

---

`bucket_name` REQUIRED string

The name of the vector bucket.

---

`next_token` OPTIONAL string

Token for pagination.

---

`max_results` OPTIONAL number

Maximum number of indexes to return.

---

`prefix` OPTIONAL string

Filter indexes by name prefix.

---

List vector indexes

```
1     response = supabase.storage.vectors().from_("vectors-bucket").list_indexes()
```

## Delete vectors from index

Deletes vectors by their keys.

This is an alpha feature and may change in the future.

Keys batch size must be between 1 and 500.

### Parameters

---

`bucket_name` REQUIRED string

The name of the vector bucket.

---

`index_name` REQUIRED string

The name of the index.

---

`keys` REQUIRED list[string]

List of vector keys to delete. Must contain between 1 and 500 keys.

---

Delete vectors

```
1 supabase.storage.vectors().from_("vectors-bucket").index("my-index").delete(["vect
```

## Retrieve vectors from index

Retrieves vectors by their keys.

This is an alpha feature and may change in the future.

### Parameters

`bucket_name` REQUIRED string

The name of the vector bucket.

`index_name` REQUIRED string

The name of the index.

`keys` REQUIRED list[string]

List of vector keys to retrieve.

`return_data` OPTIONAL bool

Whether to return vector data. Defaults to `true`.

`return_metadata` OPTIONAL bool

Whether to return metadata. Defaults to `true`.

[Get vectors](#)

```
1     response = supabase.storage.vectors().from_("vectors-bucket").index("my-index").ge
```

## List vectors in index

Lists vectors in an index with optional pagination.

This is an alpha feature and may change in the future.

### Parameters

`bucket_name` REQUIRED string

The name of the vector bucket.

`index_name` REQUIRED string

The name of the index.

`max_results` OPTIONAL number

Maximum number of vectors to return.

`next_token` OPTIONAL string

Token for pagination.

`return_data` OPTIONAL bool

Whether to return vector data. Defaults to `true`.

`return_metadata` OPTIONAL bool

Whether to return metadata. Defaults to `true`.

---

`segment_count` OPTIONAL number

Number of segments for distributed queries.

---

`segment_index` OPTIONAL number

Segment index for distributed queries.

---

List vectors

```
1   response = supabase.storage.vectors().from_("vectors-bucket").index("my-index").li
```

## Add vectors to index

Inserts or updates vectors in an index.

This is an alpha feature and may change in the future.

### Parameters

---

`bucket_name` REQUIRED string

The name of the vector bucket.

---

`index_name` REQUIRED string

The name of the index.

---

`vectors` REQUIRED list[VectorObject]

List of vector objects to insert or update.

---

Put vectors

```
1 supabase.storage.vectors().from_("vectors-bucket").index("my-index").put([
2   {
3     "key": "vector-1",
4     "data": {"float32": [0.1, 0.2, 0.3]},
5     "metadata": {"category": "example"}
6   }
7 ])
```

## Search vectors in index

Queries vectors using a query vector to find similar vectors.

This is an alpha feature and may change in the future.

### Parameters

---

`bucket_name` REQUIRED string

The name of the vector bucket.

---

`index_name` REQUIRED string

The name of the index.

---

`query_vector` REQUIRED VectorData

The query vector to search with.

---

**topK** OPTIONAL number

Number of results to return.

---

**filter** OPTIONAL VectorFilter

Optional filter to apply to the query.

---

**return\_distance** OPTIONAL bool

Whether to return distance scores. Defaults to `true`.

---

**return\_metadata** OPTIONAL bool

Whether to return metadata. Defaults to `true`.

---

Query vectors

```
1 response = supabase.storage.vectors().from_("vectors-bucket").index("my-index").qu
2     query_vector={"float32": [0.1, 0.2, 0.3]},
3     topK=10
4 )
```

⌚ Need some help? [Contact support](#)

🧪 Latest product updates? [See Changelog](#)

ⓘ Something's not right? [Check system status](#)

---

—

Contributing

Author Styleguide



Open Source

SupaSquad

Privacy Settings