DOCS

Back to Main Menu

JavaScript

v2.0

Search docs...

K

Supabase.comDashboard

-

Initializing

Create a new client for use in the browser.

You can initialize a new Supabase client using the createClient() 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

- supabaseUrl

REQUIRED

string

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

- supabaseKey

REQUIRED

string

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

- options

Optional

SupabaseClientOptions

Open accepted values

createClient()With additional parametersAPI schemasCustom `fetch` implementationReact Native options

```
import { createClient } from '@supabase/supabase-js'


// Create a single supabase client for interacting with your database

const supabase = createClient('https://xyzcompany.supabase.co', 'public-anon-key')
```

Fetch data

Perform a SELECT query on the table or view.

- 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

Query

The columns to retrieve, separated by commas. Columns can be renamed when returned with `customName:columnName`

- options

Optional

object

Named parameters

Open accepted values

Getting your dataSelecting specific columnsQuery foreign tablesQuery foreign tables through a join tableQuery the same foreign table multiple timesFiltering through foreign tablesQuerying foreign table with countQuerying with count optionQuerying JSON data

```
const { data, error } = await supabase
  .from('countries')
  .select()
```

Data source

Response

Insert data

Perform an INSERT into the table or view.

Parameters

- values

REQUIRED

object

The values to insert. Pass an object to insert a single row or an array to insert multiple rows.

- options

Optional

object

Named parameters

Open accepted values

Create a recordCreate a record and return itBulk create

```
const { error } = await supabase
  .from('countries')
  .insert({ id: 1, name: 'Denmark' })
```

Data source

Response

Update data

Perform an UPDATE on the table or view.

- update() should always be combined with Filters to target the item(s) you wish to update.

Parameters

- values

REQUIRED

Row

The values to update with

- options

Optional

object

Named parameters

Open accepted values

Updating your dataUpdate a record and return itUpdating JSON data

```
const { error } = await supabase
  .from('countries')
  .update({ name: 'Australia' })
  .eq('id', 1)
```

Data source

Response

Upsert data

Perform an UPSERT on the table or view. Depending on the column(s) passed to onConflict, .upsert() allows you to perform the equivalent of .insert() if a row with the corresponding onConflict columns doesn't exist, or if it does exist, perform an alternative action depending on ignoreDuplicates.

- Primary keys must be included in values to use upsert.

Parameters

- values

REQUIRED

object

The values to upsert with. Pass an object to upsert a single row or an array to upsert multiple rows.

- options

Optional

object

Named parameters

Open accepted values

Upsert your dataBulk Upsert your dataUpserting into tables with constraints

```
const { data, error } = await supabase
  .from('countries')
  .upsert({ id: 1, name: 'Albania' })
  .select()
```

Data source

Response

Delete data

Perform a DELETE on the table or view.

- 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.

Parameters

- options

Optional

object

Named parameters

Open accepted values

Delete records

```
const { error } = await supabase
  .from('countries')
  .delete()
  .eq('id', 1)
```

Data source

Response

Call a Postgres function

Perform a function call.

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

- fn

REQUIRED

FunctionName

The function name to call

- args

Optional

object

The arguments to pass to the function call

- options

Optional

object

Named parameters

Open accepted values

Call a Postgres function without argumentsCall a Postgres function with argumentsBulk processingCall a Postgres function with filters

```
const { data, error } = await supabase.rpc('hello_world')
```

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 a JSON columnFilter Foreign Tables

```
const { data, error } = await supabase
  .from('cities')
  .select('name, country_id')
  .eq('name', 'The Shire')    // Correct


const { data, error } = await supabase
  .from('cities')
  .eq('name', 'The Shire')    // Incorrect
  .select('name, country_id')
```

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 with

With `select()`

```
const { data, error } = await supabase
  .from('countries')
  .select()
  .eq('name', 'Albania')
```

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 with

With `select()`

```
const { data, error } = await supabase
  .from('countries')
  .select()
  .neq('name', 'Albania')
```

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 with

With `select()`

```
const { data, error } = await supabase
  .from('countries')
  .select()
  .gt('id', 2)
```

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 with

With `select()`

```
const { data, error } = await supabase
  .from('countries')
  .select()
  .gte('id', 2)
```

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 with

With `select()`

```
const { data, error } = await supabase
  .from('countries')
  .select()
  .lt('id', 2)
```

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 with

With `select()`

```
const { data, error } = await supabase
  .from('countries')
  .select()
  .lte('id', 2)
```

Data source

Response

Column matches a pattern

Match only rows where column matches pattern case-sensitively.

Parameters

- column

REQUIRED

string

The column to filter on

- pattern

REQUIRED

string

The pattern to match with

With `select()`

```
const { data, error } = await supabase
  .from('countries')
  .select()
  .like('name', '%Alba%')
```

Data source

Column matches a case-insensitive pattern

Match only rows where column matches pattern case-insensitively.

Parameters

- column

REQUIRED

string

The column to filter on

- pattern

REQUIRED

string

The pattern to match with

With `select()`

```
const { data, error } = await supabase
  .from('countries')
  .select()
  .ilike('name', '%alba%')
```

Data source

Column is a value

Match only rows where column IS value.

Parameters

- column

REQUIRED

string

The column to filter on

- value

REQUIRED

null | boolean

The value to filter with

Checking for nullness, true or false

```
const { data, error } = await supabase
  .from('countries')
  .select()
  .is('name', null)
```

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

any[]

The values array to filter with

With `select()`

```
const { data, error } = await supabase
  .from('countries')
  .select()
  .in('name', ['Albania', 'Algeria'])
```

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 jsonb, array, or range column to filter on

- value

REQUIRED

object

The jsonb, array, or range value to filter with

On array columnsOn range columnsOn `jsonb` columns

```
const { data, error } = await supabase
  .from('users')
  .select()
  .contains('name', ['is:online', 'faction:red'])
```

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 columnsOn range columnsOn `jsonb` columns

```
const { data, error } = await supabase
  .from('classes')
  .select('name')
  .containedBy('days', ['monday', 'tuesday', 'wednesday', 'friday'])
```

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

string

The range to filter with

With `select()`

```
const { data, error } = await supabase
  .from('reservations')
  .select()
  .rangeGt('during', '[2000-01-02 08:00, 2000-01-02 09:00)')
```

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()`

```
const { data, error } = await supabase
  .from('reservations')
  .select()
  .rangeGte('during', '[2000-01-02 08:30, 2000-01-02 09:30)')
```

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

string

The range to filter with

With `select()`

```
const { data, error } = await supabase
  .from('reservations')
  .select()
  .rangeLt('during', '[2000-01-01 15:00, 2000-01-01 16:00)')
```

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 either contained in range or less 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()`

```
const { data, error } = await supabase
  .from('reservations')
  .select()
  .rangeLte('during', '[2000-01-01 14:00, 2000-01-01 16:00)')
```

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

string

The range to filter with

With `select()`

```
const { data, error } = await supabase
  .from('reservations')
  .select()
  .rangeAdjacent('during', '[2000-01-01 12:00, 2000-01-01 13:00)')
```

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

object

The array or range value to filter with

On array columnsOn range columns

```
const { data, error } = await supabase
  .from('issues')
  .select('title')
  .overlaps('tags', ['is:closed', 'severity:high'])
```

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

Open accepted values

Text searchBasic normalizationFull normalizationWebsearch

```
const { data, error } = await supabase
  .from('quotes')
  .select('catchphrase')
  .textSearch('catchphrase', `'fat' & 'cat'`, {
    config: 'english'
  })
```

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

Record

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

With `select()`

```
const { data, error } = await supabase
  .from('countries')
  .select('name')
  .match({ id: 2, name: 'Albania' })
```

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('id', 'in', '(5,6,7)')  // Use `()` for `in` filter
.not('arraycol', 'cs', '{"a","b"}')  // Use `cs` for `contains()`, `{}` for array values
```

Parameters

- column

REQUIRED

string

The column to filter on

- operator

REQUIRED

string

The operator to be negated to filter with, following PostgREST syntax

- value

REQUIRED

any

The value to filter with, following PostgREST syntax

With `select()`

```
const { data, error } = await supabase
  .from('countries')
  .select()
  .not('name', 'is', null)
```

Data source

Response

Match at least one filter

Match only rows which satisfy at least one of the filters.

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

- filters

REQUIRED

string

The filters to use, following PostgREST syntax

- foreignTable

Optional

object

Set this to filter on foreign tables instead of the current table

Open accepted values

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

```
const { data, error } = await supabase
```

```
  .from('countries')
  .select('name')
  .or('id.eq.2,name.eq.Algeria')
```

Data source

Response

Match the filter

Match only rows which satisfy the filter. This is an escape hatch - you should use the specific filter methods wherever possible.

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

- column

REQUIRED

string

The column to filter on

- operator

REQUIRED

string

The operator to filter with, following PostgREST syntax

- value

REQUIRED

any

The value to filter with, following PostgREST syntax

With `select()`On a foreign table

```
const { data, error } = await supabase
  .from('countries')
  .select()
  .filter('name', 'in', '("Algeria","Japan")')
```

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).

Return data after inserting

Perform a SELECT on the query result.

Parameters

- columns

Optional

Query

The columns to retrieve, separated by commas

With `upsert()`

```
const { data, error } = await supabase
  .from('countries')
  .upsert({ id: 1, name: 'Algeria' })
  .select()
```

Data source

Response

Order the results

Order the query result by column.

Parameters

- column

REQUIRED

string

The column to order by

- options

Optional

object

Named parameters

Open accepted values

With `select()`On a foreign table

```
const { data, error } = await supabase
  .from('countries')
  .select('id', 'name')
  .order('id', { ascending: false })
```

Data source

Response

Limit the number of rows returned

Limit the query result by count.

Parameters

- count

REQUIRED

number

The maximum number of rows to return

- options

Optional

object

Named parameters

Open accepted values

With `select()`On a foreign table

```
const { data, error } = await supabase
```

```
  .from('countries')
  .select('name')
  .limit(1)
```

Data source

Response

Limit the query to a range

Limit the query result by from and to inclusively.

Parameters

- from

REQUIRED

number

The starting index from which to limit the result

- to

REQUIRED

number

The last index to which to limit the result

- options

Optional

object

Named parameters

Open accepted values

With `select()`

```
const { data, error } = await supabase
  .from('countries')
  .select('name')
  .range(0, 1)
```

Data source

Response

Set an abort signal

Set the AbortSignal for the fetch request.

Parameters

- signal

REQUIRED

AbortSignal

The AbortSignal to use for the fetch request

Aborting requests in-flight

```
const ac = new AbortController()

ac.abort()

const { data, error } = await supabase

  .from('very_big_table')

  .select()

  .abortSignal(ac.signal)
```

Data source

Response

Notes

Retrieve the query as one row

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

With `select()`

```
const { data, error } = await supabase

  .from('countries')

  .select('name')

  .limit(1)

  .single()
```

Data source

Response

Retrieve the query as 0-1 rows

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

With `select()`

```
const { data, error } = await supabase
  .from('countries')
  .select()
  .eq('name', 'Singapore')
  .maybeSingle()
```

Data source

Response

Retrieve the query as a CSV string

Return data as a string in CSV format.

Return data as CSV

```
const { data, error } = await supabase
  .from('countries')
  .select()
  .csv()
```

Data source

Response

Notes

Override type of successful response

Override the type of the returned data.

Override type of successful response

```
const { data } = await supabase
  .from('countries')
  .select()
  .returns<MyType>()
```

Response

Overview

- The auth methods can be accessed via the supabase.auth namespace.

Create auth clientCreate auth client (server-side)

```
import { createClient } from '@supabase/supabase-js'


const supabase = createClient(supabase_url, anon_key)
```

Create a new user

Creates 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.
- When the user confirms their email address, they are redirected to the SITE_URL by default. You can modify your SITE_URL or add additional redirect URLs in your project.
- If signUp() is called for an existing confirmed user:
    - If **Confirm email** is enabled in your project, an obfuscated/fake user object is returned.
    - If **Confirm email** is disabled, the error message, User already registered is returned.
- To fetch the currently logged-in user, refer to getUser().

Parameters

- credentials

REQUIRED

SignUpWithPasswordCredentials

Sign upSign up with additional user metadataSign up with a redirect URL

```
const { data, error } = await supabase.auth.signUp({
  email: 'example@email.com',
  password: 'example-password',
})
```

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

REQUIRED

SignInWithPasswordCredentials

Sign in with email and passwordSign in with phone and password

```
const { data, error } = await supabase.auth.signInWithPassword({
  email: 'example@email.com',
  password: 'example-password',
})
```

Sign in a user through OTP

Log in a user using magiclink or a one-time password (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, signInWithOtp() will signup the user instead. To restrict this behaviour, you can set shouldCreateUser 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 }}.
- See our Twilio Phone Auth Guide for details about configuring WhatsApp sign in.

Parameters

- credentials

REQUIRED

SignInWithPasswordlessCredentials

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

```
const { data, error } = await supabase.auth.signInWithOtp({
  email: 'example@email.com',
  options: {
    emailRedirectTo: 'https://example.com/welcome'
  }
})
```

Notes

Sign in a user through OAuth

Log in an existing user via a third-party provider.

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

Parameters

- credentials

REQUIRED

SignInWithOAuthCredentials

Open accepted values

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

```
const { data, error } = await supabase.auth.signInWithOAuth({
  provider: 'github'
})
```

Sign in a user through SSO

Attempts a single-sign on using an enterprise Identity Provider. A successful SSO attempt will redirect the current page to the identity provider authorization page. The redirect URL is implementation and SSO protocol specific.

- 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 providerId 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

- params

REQUIRED

SignInWithSSO

Sign in with email domainSign in with provider UUID

```
  // You can extract the user's email domain and use it to trigger the
  // authentication flow with the correct identity provider.

  const { data, error } = await supabase.auth.signInWithSSO({
    domain: 'company.com'
  })

  if (data?.url) {
    // redirect the user to the identity provider's authentication flow
    window.location.href = data.url
  }
```

Sign out a user

Inside a browser context, signOut() will remove the logged in user from the browser session and log them out - removing all items from localstorage and then trigger a "SIGNED_OUT" event.

- In order to use the signOut() method, the user needs to be signed in first.

Sign out

```
const { error } = await supabase.auth.signOut()
```

Verify and log in through OTP

Log in a user given a User supplied OTP received via mobile.

- The verifyOtp 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 verifyOtp to sign up / sign-in a user.

Parameters

- params

REQUIRED

VerifyOtpParams

Verify Sms One-Time Password (OTP)Verify Signup One-Time Password (OTP)

```
const { data, error } = await supabase.auth.verifyOtp({ phone, token, type: 'sms'})
```

Retrieve a session

Returns the session, refreshing it if necessary. The session returned can be null if the session is not detected which can happen in the event a user is not signed-in or has logged out.

- This method retrieves the current local session (i.e local storage).
- If the session has an expired access token, this method will use the refresh token to get a new session.

Get the session data

```
const { data, error } = await supabase.auth.getSession()
```

Retrieve a new session

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

- This method will refresh and return a new session whether the current one is expired or not.

Parameters

- currentSession

Optional

object

The current session. If passed in, it must contain a refresh token.

Open accepted values

Refresh session using the current sessionRefresh session using a refresh token

```
const { data, error } = await supabase.auth.refreshSession()

const { session, user } = data
```

Retrieve a user

Gets the current user details if there is an existing session.

- 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.
- Should be used only when you require the most current user data. For faster results, getSession().session.user is recommended.

Parameters

- jwt

Optional

string

Takes in an optional access token jwt. If no jwt is provided, getUser() will attempt to get the jwt from the current session.

Get the logged in user with the current existing sessionGet the logged in user with a custom access token jwt const { data: { user } } = await supabase.auth.getUser()

Update a user

Updates user data for a logged in user.

- In order to use the updateUser() 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

Open accepted values

- options

Optional

object

Open accepted values

Update the email for an authenticated userUpdate the password for an authenticated userUpdate the user's metadata

const { data, error } = await supabase.auth.updateUser({email: 'new@email.com'})

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.

- setSession() takes in a refresh token and uses it to get a new session.
- The refresh token can only be used once to obtain a new session.
- Refresh token rotation is enabled by default on all projects to guard against replay attacks.
- You can configure the REFRESH_TOKEN_REUSE_INTERVAL which provides a short window in which the same refresh token can be used multiple times in the event of concurrency or offline issues.

Parameters

- currentSession

REQUIRED

object

The current session that minimally contains an access token and refresh token.

Open accepted values

Refresh the session

```
  const { data, error } = supabase.auth.setSession({
    access_token,
    refresh_token
  })
```

Notes

Listen to auth events

Receive a notification every time an auth event happens.

- Types of auth events: SIGNED_IN, SIGNED_OUT, TOKEN_REFRESHED, USER_UPDATED, PASSWORD_RECOVERY
- Currently, onAuthStateChange() does not work across tabs. For instance, in the case of a password reset flow, the original tab which requested for the password reset link will not receive the SIGNED_IN and PASSWORD_RECOVERY event when the user clicks on the link.

Parameters

- callback

REQUIRED

function

A callback function to be invoked when an auth event happens.

Listen to auth changesListen to password recovery eventsListen to sign inListen to sign outListen to token refreshListen to user updates

```
supabase.auth.onAuthStateChange((event, session) => {
  console.log(event, session)
})
```

Exchange an auth code for a session

Log in an existing user via a third-party provider.

- Used when flowType is set to pkce in client options.

Parameters

- authCode

REQUIRED

string

Exchange Auth Code

```
supabase.auth.exchangeCodeForSession('34e770dd-9ff9-416c-87fa-43b31d7ef225')
```

Enroll a factor

Starts the enrollment process for a new Multi-Factor Authentication (MFA) factor. This method creates a new unverified factor. To verify a factor, present the QR code or secret to the user and ask them to add it to their authenticator app. The user has to enter the code from their authenticator app to verify it.

- Currently, totp is the only supported factorType. 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.challengeAndVerify().
- To generate a QR code for the totp secret in nextjs, you can do the following:

```
<Image src={data.totp.qr_code} alt={data.totp.uri} layout="fill"></Image>
```

Parameters

- params

REQUIRED

MFAEnrollParams

Open accepted values

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

```
const { data, error } = await supabase.auth.mfa.enroll({
  factorType: 'totp'
})


// Use the id to create a challenge.
// The challenge can be verified by entering the code generated from the authenticator app.
// The code will be generated upon scanning the qr_code or entering the secret into the authenticator app.
const { id, type, totp: { qr_code, secret, uri } } = data
```

Create a challenge

Prepares a challenge used to verify that a user has access to a MFA factor.

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

Parameters

- params

REQUIRED

MFAChallengeParams

Open accepted values

Create a challenge for a factor

```
const { data, error } = await supabase.auth.mfa.challenge({
  factorId: '34e770dd-9ff9-416c-87fa-43b31d7ef225'
})
```

Verify a challenge

Verifies a code against a challenge. The verification code is provided by the user by entering a code seen in their authenticator app.

- To verify a challenge, please create a challenge first.

Parameters

- params

REQUIRED

MFAVerifyParams

Open accepted values

Verify a challenge for a factor

```
const { data, error } = await supabase.auth.mfa.verify({
  factorId: '34e770dd-9ff9-416c-87fa-43b31d7ef225',
  challengeId: '4034ae6f-a8ce-4fb5-8ee5-69a5863a7c15',
  code: '123456'
})
```

Create and verify a challenge

Helper method which creates a challenge and immediately uses the given code to verify against it thereafter. The verification code is provided by the user by entering a code seen in their authenticator app.

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

Parameters

- params

REQUIRED

MFAChallengeAndVerifyParams

Open accepted values

Create and verify a challenge for a factor

```
const { data, error } = await supabase.auth.mfa.challengeAndVerify({
  factorId: '34e770dd-9ff9-416c-87fa-43b31d7ef225',
  code: '123456'
})
```

Unenroll a factor

Unenroll removes a MFA factor. A user has to have an aal2 authenticator level in order to unenroll a verified factor.

Parameters

- params

REQUIRED

MFAUnenrollParams

Open accepted values

Unenroll a factor

```
const { data, error } = await supabase.auth.mfa.unenroll({
  factorId: '34e770dd-9ff9-416c-87fa-43b31d7ef225',
})
```

Get Authenticator Assurance Level

Returns the Authenticator Assurance Level (AAL) for the active session.

- 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 nextLevel field will return aal2, else, it will return aal1.

Get the AAL details of a session

```
const { data, error } = await supabase.auth.mfa.getAuthenticatorAssuranceLevel()
const { currentLevel, nextLevel, currentAuthenticationMethods } = data
```

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

```
import { createClient } from '@supabase/supabase-js'


const supabase = createClient(supabase_url, service_role_key, {
  auth: {
    autoRefreshToken: false,
    persistSession: false
  }
})

// Access auth admin api
const adminAuthClient = supabase.auth.admin
```

Retrieve a user

Get user by id.

- Fetches the user object from the database based on the user's id.
- The getUserById() 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

```
const { data, error } = await supabase.auth.admin.getUserById(1)
```

List all users

Get a list of users.

- Defaults to return 50 users per page.

Parameters

- params

Optional

PageParams

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

Open accepted values

Get a page of usersPaginated list of users

```
const { data: { users }, error } = await supabase.auth.admin.listUsers()
```

Create a user

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

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

Parameters

- attributes

REQUIRED

AdminUserAttributes

Open accepted values

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

```
const { data, error } = await supabase.auth.admin.createUser({
  email: 'user@email.com',
  password: 'password',
  user_metadata: { name: 'Yoda' }
})
```

Delete a user

Delete a user. Requires a service_role key.

- The deleteUser() 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.

- shouldSoftDelete

Optional

boolean

If true, then the user will be soft-deleted 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

```
const { data, error } = await supabase.auth.admin.deleteUser(
  '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.

Parameters

- email

REQUIRED

string

The email address of the user.

- options

Optional

object

Open accepted values

Invite a user

```
const { data, error } = await supabase.auth.admin.inviteUserByEmail('email@example.com')
```

Send a password reset request

Sends a password reset request to an email address.

- 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 resetPasswordForEmail() only sends a password reset link to the user's email. To update the user's password, see updateUser().
- A SIGNED_IN and PASSWORD_RECOVERY event will be emitted when the password recovery link is clicked. You can use onAuthStateChange() to listen and invoke a callback function on these events.
- 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 updateUser():

```
const { data, error } = await supabase.auth.updateUser({
  password: new_password
})
```

Parameters

- email

REQUIRED

string

The email address of the user.

- options

Optional

object

Open accepted values

Reset passwordReset password (React)

```
const { data, error } = await supabase.auth.resetPasswordForEmail(email, {
  redirectTo: 'https://example.com/update-password',
})
```

Generate an email link

Generates email links and OTPs to be sent via a custom email provider.

Parameters

- params

REQUIRED

GenerateLinkParams

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

```
const { data, error } = await supabase.auth.admin.generateLink({
  type: 'signup',
  email: 'email@example.com',
  password: 'secret'
})
```

Update a user

Updates the user data.

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.

Open accepted values

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

```
const { data: user, error } = await supabase.auth.admin.updateUserById(
  '6aa5d0d4-2a9f-4483-b6c8-0cf4c6c98ac4',
  { email: 'new@email.com' }
)
```

List all factors for a user

Lists all factors associated to a user.

Parameters

- params

REQUIRED

AuthMFAAdminListFactorsParams

Open accepted values

List all factors for a user

```
const { data, error } = await supabase.auth.admin.mfa.listFactors()
```

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

Open accepted values

Delete a factor for a user

```
const { data, error } = await supabase.auth.admin.mfa.deleteFactor({
  id: '34e770dd-9ff9-416c-87fa-43b31d7ef225',
  userId: 'a89baba7-b1b7-440f-b4bb-91026967f66b',
})
```

Invoke a function

Invokes a function

Invoke a Supabase Function.

- Requires an Authorization header.
- Invoke params generally match the Fetch API spec.

- 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.
- Responses are automatically parsed as json, blob and form-data depending on the Content-Type header sent by your function. Responses are parsed as text by default.

Parameters

- functionName

REQUIRED

string

The name of the Function to invoke.

- options

Optional

FunctionInvokeOptions

Options for invoking the Function.

Open accepted values

Basic invocationError handlingPassing custom headers

```
const { data, error } = await supabase.functions.invoke('hello', {
  body: { foo: 'bar' }
})
```

Subscribe to channel

Creates an event handler that listens to changes.

- 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.

Parameters

- type

REQUIRED

"broadcast"

One of "broadcast", "presence", or "postgres_changes".

- filter

REQUIRED

object

Custom object specific to the Realtime feature detailing which payloads to receive.

Open accepted values

- callback

REQUIRED

function

Function to be invoked when event handler is triggered.

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

```
supabase
  .channel('any')
  .on('broadcast', { event: 'cursor-pos' }, payload => {
    console.log('Cursor position received!', payload)
  })
  .subscribe((status) => {
    if (status === 'SUBSCRIBED') {
      channel.send({
        type: 'broadcast',
        event: 'cursor-pos',
        payload: { x: Math.random(), y: Math.random() },
      })
  }
 })
```

Unsubscribe from a channel

Unsubscribes and removes Realtime channel from Realtime client.

- 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.

Parameters

- channel

REQUIRED

default

The name of the Realtime channel.

Removes a channel

```
supabase.removeChannel(myChannel)
```

Unsubscribe from all channels

Unsubscribes and removes all Realtime channels from Realtime client.

- 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.removeAllChannels()
```

Retrieve all channels

Returns all Realtime channels.

Get all channels

```
const channels = supabase.getChannels()
```

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

Optional

object

Open accepted values

Create bucket

```
const { data, error } = await supabase
  .storage
  .createBucket('avatars', {
    public: false,
    allowedMimeTypes: ['image/png'],
    fileSizeLimit: 1024
  })
```

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

```
const { data, error } = await supabase
  .storage
  .getBucket('avatars')
```

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

```
const { data, error } = await supabase
  .storage
  .listBuckets()
```

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 updating.

- options

REQUIRED

object

Open accepted values

Update bucket

```
const { data, error } = await supabase
  .storage
  .updateBucket('avatars', {
    public: false,
    allowedMimeTypes: ['image/png'],
    fileSizeLimit: 1024
  })
```

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

```
const { data, error } = await supabase
  .storage
  .deleteBucket('avatars')
```

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

```
const { data, error } = await supabase
  .storage
  .emptyBucket('avatars')
```

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
- For React Native, using either Blob, File or FormData does not work as intended. Upload file using ArrayBuffer from base64 file data instead, see example below.

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.

- fileBody

REQUIRED

FileBody

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

- fileOptions

Optional

FileOptions

Open accepted values

Upload fileUpload file using `ArrayBuffer` from base64 file data

```
const avatarFile = event.target.files[0]

const { data, error } = await supabase

  .storage

  .from('avatars')

  .upload('public/avatar1.png', avatarFile, {

    cacheControl: '3600',

    upsert: false

  })
```

Download a file

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

Optional

object

Open accepted values

Download fileDownload file with transformations

```
const { data, error } = await supabase
  .storage
  .from('avatars')
  .download('folder/avatar1.png')
```

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

Open accepted values

- parameters

Optional

FetchParameters

Open accepted values

List files in a bucketSearch files in a bucket

```
const { data, error } = await supabase
  .storage
  .from('avatars')
  .list('folder', {
    limit: 100,
    offset: 0,
    sortBy: { column: 'name', order: 'asc' },
  })
```

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
- For React Native, using either Blob, File or FormData does not work as intended. Update file using ArrayBuffer from base64 file data instead, see example below.

Parameters

- path

REQUIRED

string

The relative file path. Should be of the format `folder/subfolder/filename.png`. The bucket must already exist before attempting to update.

- fileBody

REQUIRED

object

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

- fileOptions

Optional

FileOptions

Open accepted values

Update fileUpdate file using `ArrayBuffer` from base64 file data

```
const avatarFile = event.target.files[0]

const { data, error } = await supabase

  .storage

  .from('avatars')

  .update('public/avatar1.png', avatarFile, {

    cacheControl: '3600',

    upsert: true

  })
```

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

- fromPath

REQUIRED

string

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

- toPath

REQUIRED

string

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

Move file

```
const { data, error } = await supabase
  .storage
  .from('avatars')
  .move('public/avatar1.png', 'private/avatar2.png')
```

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: insert and select
- Refer to the Storage guide on how access control works

Parameters

- fromPath

REQUIRED

string

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

- toPath

REQUIRED

string

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

Copy file

```
const { data, error } = await supabase
  .storage
  .from('avatars')
  .copy('public/avatar1.png', 'private/avatar2.png')
```

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

string[]

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

Delete file

```
const { data, error } = await supabase
  .storage
  .from('avatars')
  .remove(['folder/avatar1.png'])
```

Create a signed URL

Creates a signed URL. 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 current file name. For example `folder/image.png`.

- expiresIn

REQUIRED

number

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

- options

Optional

object

Open accepted values

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

```
const { data, error } = await supabase
  .storage
  .from('avatars')
  .createSignedUrl('folder/avatar1.png', 60)
```

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

string[]

The file paths to be downloaded, including the current file names. For example `['folder/image.png', 'folder2/image2.png']`.

- expiresIn

REQUIRED

number

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

- options

Optional

object

Open accepted values

Create Signed URLs

```
const { data, error } = await supabase
  .storage
  .from('avatars')
  .createSignedUrls(['folder/avatar1.png', 'folder/avatar2.png'], 60)
```

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`.

Create Signed Upload URL

```
const { data, error } = await supabase
  .storage
  .from('avatars')
  .createSignedUploadUrl('folder/cat.jpg')
```

Upload to a signed URL

Upload a file with a token generated from createSignedUploadUrl.

- 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 `createSignedUploadUrl`

- fileBody

REQUIRED

FileBody

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

- fileOptions

Optional

FileOptions

Open accepted values

Upload to a signed URL

```
const { data, error } = await supabase
  .storage
  .from('avatars')
  .uploadToSignedUrl('folder/cat.jpg', 'token-from-createSignedUploadUrl', file)
```

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 updateBucket() or by going to Storage on app.supabase.com, 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

object

Open accepted values

Returns the URL for an asset in a public bucketReturns the URL for an asset in a public bucket with transformationsReturns the URL which triggers the download of an asset in a public bucket

```
const { data } = supabase
```

```
 .storage
 .from('public-bucket')
 .getPublicUrl('folder/avatar1.png')
```

- Need some help?

Contact support

- Lastest product updates?

See Changelog

- Something's not right?

Check system status
