Supabase

Postgres, auth, storage and realtime from one client.

01 Install & local stack

```
npm install @supabase/supabase-js
npm install supabase --save-dev
npx supabase init    # adds supabase/
npx supabase start   # needs Docker
npx supabase stop
```

With Homebrew (brew install supabase/tap/supabase), drop the npx. Local API on port 54321, Studio on 54323.

02 Create a client

```
import type { Database } from './db.types'
const supabase = createClient<Database>(
  process.env.SUPABASE_URL!,
  process.env.SUPABASE_PUBLISHABLE_KEY!,
  { auth: { flowType: 'pkce' } },
)
```

The publishable key (sb_publishable_) is safe in browsers with RLS on. It replaces the legacy anon key. Options also take db.schema and global.headers.

03 Server client (@supabase/ssr)

```
const jar = await cookies()  // next/headers
const supabase = createServerClient(
  url, publishableKey, { cookies: {
    getAll: () => jar.getAll(),
    setAll: (list) => list.forEach((c) =>
      jar.set(c.name, c.value, c.options)),
  } })
```

One client per request. Check users with getClaims() or getUser(), never getSession(). In Server Components set() throws: wrap setAll in try/catch.

04 Read rows

```
const { data, error } = await supabase
  .from('posts')
  .select('id, title, created_at')
  .eq('status', 'published')
  .order('created_at', { ascending: false })
  .limit(10)
```

Calls return { data, error } instead of throwing. select('*') reads every column; .single() returns one object, not an array.

05 Insert, update & delete

```
const { data } = await supabase
  .from('posts')
  .insert({ title: 'Hello' }) // or an array
  .select()                   // return rows
await supabase.from('posts')
  .update({ status: 'draft' }).eq('id', 1)
await supabase.from('posts')
  .delete().in('id', [1, 2, 3])
```

Writes return no rows unless you chain .select(). upsert() inserts or updates by primary key.

06 Filter operators

.eq() .neq()	Equal, not equal
.gt .gte .lt .lte	Greater or less than
.like() .ilike()	Pattern; ilike ignores case
.in(col, list)	Value is in the list
.is(col, null)	IS NULL; .not(col, 'is', null)
.or(filters)	'status.eq.draft,views.gt.9'

Filters chain with AND. .contains() matches arrays and JSON.

07 Joins, counts & paging

```
const { data, count } = await supabase
  .from('posts')
  .select(`
    id, title, author:users(name, email),
    comments(id, content)
  `, { count: 'exact' })
  .range(0, 9)  // first 10 rows
```

Tables embed through foreign keys; author: renames the field. range() is zero-based and inclusive.

08 Sign up & sign in

```
await supabase.auth.signUp({
  email, password,
  options: { data: { first_name: 'Jo' } },
})
const { data, error } = await supabase.auth
  .signInWithPassword({ email, password })
await supabase.auth
  .signInWithOAuth({ provider: 'github' })
```

signInWithOtp({ email }) sends a magic link.

09 Sessions & users

getUser()	Asks Auth: safe on a server
getClaims()	Verified JWT claims
getSession()	Stored session: not verified
updateUser()	Email, password or metadata
signOut()	Every session by default

signOut({ scope: 'local' }) ends one session. onAuthStateChange() fires on each auth event.

Supabase

Row Level Security, files, realtime and deploys.

10 Row Level Security

```
ALTER TABLE posts ENABLE ROW LEVEL SECURITY;
CREATE POLICY "Read published posts"
ON posts FOR SELECT
TO anon, authenticated
USING (status = 'published');
```

With RLS on, the API returns no rows until a policy allows them. The secret key (legacy service_role) bypasses RLS: keep it on the server.

11 Owner-only writes

```
CREATE POLICY "Authors edit own posts"
ON posts FOR UPDATE TO authenticated
USING ((SELECT auth.uid()) = author_id)
WITH CHECK (
  (SELECT auth.uid()) = author_id
);
```

USING picks the rows; WITH CHECK validates the new row. SELECT around auth.uid() caches it per statement.

12 Postgres functions & RPC

```
CREATE FUNCTION increment_views(post_id INT)
RETURNS void AS $$
  UPDATE posts SET views = views + 1
  WHERE id = post_id;
$$ LANGUAGE sql;
```

```
await supabase.rpc('increment_views',
  { post_id: 123 })
```

Filters can follow rpc() when the function returns a table. Triggers run a RETURNS trigger function per row.

13 Vector search (pgvector)

```
CREATE TABLE documents (
  id bigserial PRIMARY KEY, content text,
  embedding extensions.vector(1536)
);
CREATE INDEX ON documents
  USING hnsw (embedding vector_cosine_ops);
```

Needs CREATE EXTENSION vector WITH SCHEMA extensions. Rank rows by embedding <=> query in a SQL function.

14 Storage: upload & download

```
const { error } = await supabase.storage
  .from('avatars')
  .upload(`${user.id}/avatar.png`, file, {
    cacheControl: '3600',
    upsert: true,  // replace if it exists
  })
const { data } = await supabase.storage
  .from('avatars').download(path)
```

download() returns a Blob. Upload progress needs resumable TUS uploads.

15 Storage: URLs & files

`getPublicUrl()`	URL for a public bucket
`createSignedUrl()`	Expiring URL, e.g. 3600 s
`list(folder)`	limit, offset, sortBy
`move(from, to)`	Rename or move a file
`remove([paths])`	Delete one or more files

getPublicUrl(path, { transform: { width: 200 } }) resizes on the fly and serves a modern format such as WebP.

16 Realtime

```
const channel = supabase.channel('feed')
  .on('postgres_changes', {
    event: 'INSERT', schema: 'public',
    table: 'posts',
  }, (payload) => console.log(payload.new))
  .subscribe()
await supabase.removeChannel(channel)
```

Add the table to the supabase_realtime publication. filter: 'author_id=eq.7' narrows rows.

17 Edge Functions

```
// supabase/functions/hello/index.ts
Deno.serve(async (req) => {
  const { name } = await req.json()
  return Response.json({ hello: name })
})
const { data } = await supabase.functions
  .invoke('hello', { body: { name: 'Jo' } })
```

Deno runtime: import npm:@supabase/supabase-js@2; secrets come from Deno.env.get().

18 Migrations & deploy (CLI v3)

```
supabase migration new create_posts
supabase db reset          # rebuild local
supabase link --project-ref <ref>
supabase db push --remote  # to the project
supabase gen types typescript > db.types.ts
supabase functions deploy hello
supabase secrets set API_KEY=value
```

v3 targets the local stack unless you pass --remote. Hosted changes ask first; CI passes --yes.