THE STACK

A Step-by-Step Developer Guide

Terminal	**GitHub**	**Vite / React / TS**	**Supabase**	**Cloudflare Workers**	**Vercel**

Built for self-taught developers who learn by doing.

thestack.dev • 2026

Contents

01

Terminal Basics

Commands every developer needs to know

The terminal is where you control your computer with text commands. It looks intimidating at first but 15 commands cover 90% of real development work. The other 10% gets looked up as needed — even experienced developers Google terminal commands constantly.

HOW TO OPEN THE TERMINAL

Mac	Cmd + Space → type Terminal → Enter (or find it in Applications → Utilities)
Windows	Win + R → type powershell → Enter (PowerShell is better than cmd for dev work)
VS Code	Ctrl + ` (backtick key, top-left of keyboard) — opens terminal inside VS Code

NAVIGATION & FILE COMMANDS

Command	Name	What it does
`pwd`	Print Working Directory	Shows exactly where you are on your computer
`ls`	List	Shows all files and folders in your current location
`ls -la`	List (detailed)	Shows hidden files too — including .env and .gitignore
`cd folder-name`	Change Directory	Move into a folder. Example: cd my-project
`cd ..`	Go up one level	Move out of the current folder to its parent
`cd ~`	Go home	Jump to your home directory from anywhere
`cd ~/Desktop`	Go to Desktop	Jump directly to your Desktop folder
`mkdir folder-name`	Make Directory	Creates a new empty folder
`touch filename`	Create empty file	Creates a blank file. Example: touch index.ts
`cp file copy`	Copy	Copies a file. Example: cp app.ts app-backup.ts
`mv file dest`	Move / Rename	Moves or renames a file
`rm filename`	Remove file	Permanently deletes a file — no trash, no undo
`rm -rf folder`	Remove folder	Deletes an entire folder recursively — use with caution
`cat filename`	Show file contents	Prints the contents of a file in the terminal
`clear`	Clear screen	Cleans up the terminal display
`history`	Command history	Shows all commands you've previously run

NPM COMMANDS (Node Package Manager)

`npm install`	Install all project dependencies from package.json

`npm install package-name`	Add a new package to your project
`npm install -D package-name`	Add a dev-only dependency (not shipped to production)
`npm uninstall package-name`	Remove a package from your project
`npm run dev`	Start the local development server (usually localhost:5173)
`npm run build`	Build the production-ready version of your app into /dist
`npm run preview`	Preview the production build locally before deploying
`npm list`	Show all installed packages and their versions
`node -v`	Check your Node.js version
`npm -v`	Check your npm version

✓ PRO TIP
Use Tab to auto-complete folder and file names. Up arrow recalls your last command. These two shortcuts save enormous time.

■ HEADS UP
rm -rf has no confirmation prompt and no undo. Always double-check the path before running it.

02

Setting Up a New Project

Vite + React + TypeScript from scratch

Every app starts here. Vite is the build tool that makes development fast. React is the UI framework. TypeScript is JavaScript with type-checking that catches bugs before they happen. This trio is the modern standard for web apps and what the industry uses.

1

Install Node.js first (one-time setup)

Go to nodejs.org and download the LTS version.

Run the installer. When done, verify in terminal:

```
node -v        # should show v18 or higher
npm -v         # should show v9 or higher
```

2

Open your terminal in the right place

In VS Code: File → Open Folder → pick where you keep projects

Then press Ctrl+` to open the terminal — you'll already be in that folder.

3

Create the project with Vite

RUN THIS COMMAND

```
npm create vite@latest my-app-name
```

Replace my-app-name with your project name (no spaces, use dashes).

You'll be prompted: pick React, then TypeScript (not TypeScript + SWC for simplicity).

4

Install dependencies and start

RUN THESE COMMANDS ONE AT A TIME

```
cd my-app-name
npm install
npm run dev
```

Open browser: http://localhost:5173 — you should see the Vite welcome screen.

5

Install common packages you'll almost always need

RUN THESE TO ADD COMMONLY USED PACKAGES

```
npm install @supabase/supabase-js
npm install react-router-dom
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init -p
```

YOUR PROJECT STRUCTURE — WHAT EVERYTHING IS

`my-app-name/`	Root folder — this is your entire project
`src/`	All your source code lives here — this is where you work
`src/App.tsx`	Main app component — start editing here first
`src/main.tsx`	Entry point — mounts your app into index.html, rarely edit this
`src/components/`	Create this folder for reusable UI components
`src/lib/`	Create this for utilities, supabase client, helpers
`src/types/`	Create this for TypeScript type definitions
`public/`	Static files served directly — images, fonts, icons
`index.html`	The HTML shell — usually leave as-is
`vite.config.ts`	Vite configuration — sometimes edited for deployment
`tailwind.config.js`	Tailwind CSS configuration
`tsconfig.json`	TypeScript configuration
`package.json`	Lists all dependencies and available npm scripts
`package-lock.json`	Locked versions — never edit manually
`.env`	Your secret keys — NEVER commit this to GitHub
`.gitignore`	Tells Git which files to ignore when pushing
`node_modules/`	All installed packages — never edit, never upload to GitHub
`dist/`	Built production files — created when you run npm run build

03

GitHub Setup & Pushing Code

Version control and everyday workflow

GitHub stores your code in the cloud. Every time you push, it saves a snapshot you can go back to. It also makes deploying to Vercel effortless — every push can trigger an automatic redeploy. You need a GitHub account at github.com before starting.

FIRST-TIME SETUP — DO THIS ONCE

Step	Action	Command / Where
1	Install Git	`Download from git-scm.com — pre-installed on Mac`
2	Set your name	`git config --global user.name "Your Name"`
3	Set your email	`git config --global user.email "you@email.com"`
4	Check it worked	`git config --list  (look for user.name and user.email)`
5	Auth with GitHub	`Use GitHub Desktop app OR set up SSH keys (see GitHub docs)`

1

Create a new repository on GitHub

github.com → click the + icon (top right) → New repository

Give it a name → choose Public or Private → click Create repository

Do NOT check 'Initialize with README' if you already have local code.

2

Initialize Git in your project (terminal)

RUN INSIDE YOUR PROJECT FOLDER

```
git init
git add .
git commit -m "initial commit"
```

3

Connect to GitHub and push

GITHUB WILL SHOW YOU THESE EXACT COMMANDS AFTER CREATING THE REPO

```
git remote add origin https://github.com/yourname/repo.git
git branch -M main
git push -u origin main
```

After this first push, you just need: git push

EVERYDAY GIT WORKFLOW

Every time you finish a chunk of work and want to save + upload:

`git status`	See what files changed since your last commit
`git diff filename`	See exactly what changed inside a specific file
`git add .`	Stage ALL changed files for the next commit
`git add filename`	Stage only a specific file
`git commit -m "describe what you did"`	Save a snapshot — be descriptive
`git push`	Upload your commits to GitHub
`git pull`	Download latest changes (useful if working from multiple machines)
`git log --oneline`	See a condensed history of all commits
`git stash`	Temporarily save uncommitted changes without committing
`git stash pop`	Restore stashed changes

YOUR .gitignore FILE — CREATE THIS IN EVERY PROJECT

Create a file named exactly .gitignore in your project root and paste this content:

```
node_modules/ dist/ .env .env.local .env.production .DS_Store *.log
```

HEADS UP
If you accidentally push a .env file with real API keys, change all those keys immediately. GitHub scans public repos for exposed secrets.

04 Supabase Setup

Database, auth, storage + full SQL reference

Supabase is your complete backend in one place — PostgreSQL database, user authentication, and file storage. It has a visual dashboard so you can manage everything without writing server code. The SQL Editor is powerful and you'll use it regularly to fix issues and set up tables.

ACCOUNT & PROJECT SETUP

1 Create a Supabase account and project

Go to supabase.com → Sign up → New Project

Set a strong database password and save it somewhere safe.

Pick a region close to your users.

2 Get your API credentials

Dashboard → Settings → API

COPY THESE TWO VALUES INTO YOUR .ENV FILE

```
Project URL:   https://xxxxxxxxxx.supabase.co
Anon/Public Key: eyJhbGci... (long string)
```

The anon key is safe to use in frontend code — it has limited permissions by design.

3 Install the Supabase SDK

TERMINAL COMMAND

```
npm install @supabase/supabase-js
```

4

Create src/lib/supabase.ts

CREATE THIS FILE AND PASTE EXACTLY

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

const url = import.meta.env.VITE_SUPABASE_URL
const key = import.meta.env.VITE_SUPABASE_ANON_KEY

export const supabase = createClient(url, key)
```

Import this in any file that needs database/auth/storage access.

SQL REFERENCE — TABLE CREATION

Go to your Supabase Dashboard → SQL Editor → New Query, paste the SQL, and click Run. Use these templates as starting points for your tables.

USERS / PROFILES TABLE — run after auth is set up

```
CREATE TABLE public.profiles (
  id uuid REFERENCES auth.users(id) PRIMARY KEY,
  username text UNIQUE,
  full_name text,
  avatar_url text,
  created_at timestamptz DEFAULT now()
);
```

TRACKS TABLE — for music/audio apps

```
CREATE TABLE public.tracks (
  id uuid DEFAULT gen_random_uuid() PRIMARY KEY,
  user_id uuid REFERENCES auth.users(id) ON DELETE CASCADE,
  title text NOT NULL,
  artist text,
  album text,
  duration_seconds integer,
  file_url text,
  created_at timestamptz DEFAULT now()
);
```

PLAYLISTS TABLE

```
CREATE TABLE public.playlists (
  id uuid DEFAULT gen_random_uuid() PRIMARY KEY,
  user_id uuid REFERENCES auth.users(id) ON DELETE CASCADE,
  name text NOT NULL,
  description text,
  track_ids uuid[],
  created_at timestamptz DEFAULT now()
);
```

PLAY LOGS TABLE — for tracking plays / PRO reporting

```
CREATE TABLE public.play_logs (
  id uuid DEFAULT gen_random_uuid() PRIMARY KEY,
  user_id uuid REFERENCES auth.users(id),
  track_id uuid REFERENCES public.tracks(id),
  played_at timestamptz DEFAULT now(),
  duration_played integer
);
```

ADD COLUMNS TO EXISTING TABLE — used to extend profiles

```
ALTER TABLE public.profiles
  ADD COLUMN IF NOT EXISTS primary_pro text,
  ADD COLUMN IF NOT EXISTS ascap_id text,
  ADD COLUMN IF NOT EXISTS bmi_id text,
  ADD COLUMN IF NOT EXISTS sesac_id text,
  ADD COLUMN IF NOT EXISTS ipi_number text,
  ADD COLUMN IF NOT EXISTS publisher_name text,
  ADD COLUMN IF NOT EXISTS distributor_name text,
  ADD COLUMN IF NOT EXISTS label_name text;
```

SQL REFERENCE — AUTHENTICATION FIXES

CONFIRM A USER'S EMAIL MANUALLY — fixes login blocked by email confirmation

```
UPDATE auth.users
SET
  email_confirmed_at = now(),
  confirmed_at = now()
WHERE email = 'your-email@example.com';
```

CHECK IF A USER EXISTS IN auth.users

```
SELECT id, email, email_confirmed_at, created_at
FROM auth.users
WHERE email = 'your-email@example.com';
```

CREATE PROFILE AUTOMATICALLY WHEN USER SIGNS UP — trigger

```
CREATE OR REPLACE FUNCTION public.handle_new_user()
RETURNS TRIGGER AS $$
BEGIN
  INSERT INTO public.profiles (id, full_name, avatar_url)
  VALUES (NEW.id, NEW.raw_user_meta_data->>'full_name',
          NEW.raw_user_meta_data->>'avatar_url');
  RETURN NEW;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE TRIGGER on_auth_user_created
  AFTER INSERT ON auth.users
  FOR EACH ROW EXECUTE FUNCTION public.handle_new_user();
```

SQL REFERENCE — ROW LEVEL SECURITY (RLS)

RLS ensures users can only access their own data. Always enable it on tables that store user data. Without RLS, any authenticated user can read everyone's data.

ENABLE RLS AND SET USER-ONLY ACCESS POLICIES

```
-- Enable RLS on a table
ALTER TABLE public.tracks ENABLE ROW LEVEL SECURITY;

-- Users can only SELECT their own rows
CREATE POLICY "Users can view own tracks"
  ON public.tracks FOR SELECT
  USING (auth.uid() = user_id);

-- Users can only INSERT their own rows
CREATE POLICY "Users can insert own tracks"
  ON public.tracks FOR INSERT
  WITH CHECK (auth.uid() = user_id);

-- Users can only DELETE their own rows
CREATE POLICY "Users can delete own tracks"
  ON public.tracks FOR DELETE
  USING (auth.uid() = user_id);
```

ALLOW PUBLIC READ ACCESS (for shared/public content)

```
CREATE POLICY "Public read access"
  ON public.tracks FOR SELECT
  USING (true);
```

SQL REFERENCE — USEFUL QUERIES

CHECK ALL TABLES IN YOUR PROJECT

```
SELECT table_name FROM information_schema.tables
WHERE table_schema = 'public';
```

CHECK COLUMNS IN A TABLE

```
SELECT column_name, data_type, is_nullable
FROM information_schema.columns
WHERE table_name = 'tracks';
```

DELETE ALL ROWS FROM A TABLE (keep structure)

```
TRUNCATE TABLE public.play_logs;
-- or for a single row:
DELETE FROM public.tracks WHERE id = 'your-uuid-here';
```

CHECK ALL ACTIVE RLS POLICIES ON A TABLE

```
SELECT policyname, cmd, qual
FROM pg_policies
WHERE tablename = 'tracks';
```

AUTHENTICATION — SDK CALLS

Register new user	`supabase.auth.signUp({ email, password })`
Sign in	`supabase.auth.signInWithPassword({ email, password })`
Sign out	`supabase.auth.signOut()`
Get current user	`const { data: { user } } = await supabase.auth.getUser()`
Listen for auth changes	`supabase.auth.onAuthStateChange((event, session) => { })`
Get session	`const { data: { session } } = await supabase.auth.getSession()`
Update user email/password	`supabase.auth.updateUser({ email: 'new@email.com' })`
Password reset email	`supabase.auth.resetPasswordForEmail(email)`

DATABASE QUERIES — SDK CALLS

Select all rows	`supabase.from('tracks').select('*')`
Select with filter	`supabase.from('tracks').select('*').eq('user_id', userId)`
Select specific columns	`supabase.from('tracks').select('id, title, artist')`
Insert a row	`supabase.from('tracks').insert([{ title, artist, user_id }])`
Update a row	`supabase.from('tracks').update({ title }).eq('id', trackId)`

Delete a row	`supabase.from('tracks').delete().eq('id', trackId)`
Order results	`supabase.from('tracks').select('*').order('created_at', { ascending: false })`
Limit results	`supabase.from('tracks').select('*').limit(10)`

FILE STORAGE — SDK CALLS

Upload file	`supabase.storage.from('bucket-name').upload(filePath, file)`
Upload with upsert (overwrite)	`supabase.storage.from('bucket').upload(path, file, { upsert: true })`
Get public URL	`supabase.storage.from('bucket-name').getPublicUrl(filePath)`
Download file	`supabase.storage.from('bucket-name').download(filePath)`
Delete file	`supabase.storage.from('bucket-name').remove([filePath])`
List files in folder	`supabase.storage.from('bucket-name').list('folder/')`
Move/rename file	`supabase.storage.from('bucket').move(oldPath, newPath)`

✓ PRO TIP
After creating any table, go to Authentication → Policies and enable RLS. Then add your policies. Tables without RLS are open to all authenticated users.

05

Environment Variables & Key Safety

Never expose your secrets

API keys and database credentials must never be hardcoded in your source code. If you push a file with a real API key to a public GitHub repo, it can be stolen within minutes by automated scanners. Environment variables let your code reference secrets by name without ever containing the actual value.

1

Create your .env file

In VS Code: right-click your project root folder → New File

Name it exactly: .env   (the dot is part of the name)

It sits at the same level as package.json

This file will NOT show up in Finder/Explorer by default — it's hidden.

2

Add your variables — format is KEY=value

EXAMPLE .ENV FOR A TYPICAL PROJECT

```
VITE_SUPABASE_URL=https://xxxxxxxxxx.supabase.co
VITE_SUPABASE_ANON_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
VITE_CLOUDFLARE_WORKER_URL=https://my-worker.workers.dev
```

No quotes. No spaces around =. VITE_ prefix required for Vite/React apps.

3

Reference them in your code

IN ANY .TS OR .TSX FILE

```
const url = import.meta.env.VITE_SUPABASE_URL
const key = import.meta.env.VITE_SUPABASE_ANON_KEY
```

Vite exposes only VITE_ prefixed variables to the browser — others stay private.

4

Add .env to .gitignore

Open .gitignore (or create it) and make sure these lines are in it:

```
.env
.env.local
.env.production
```

Without this, your secrets will be uploaded to GitHub.

5

Add variables to your deployment platforms

Your .env file only works on your local machine. For live apps:

VERCEL

Project Settings → Environment Variables → add each one

CLOUDFLARE WORKERS

Worker dashboard → Settings → Variables → add each secret → Encrypt

GITHUB ACTIONS (FOR GITHUB PAGES DEPLOYMENT)

Repo → Settings → Secrets and Variables → Actions → New secret

BAD vs GOOD — Hardcoded vs Environment Variable

BAD — never do this:

```
const supabaseUrl = 'https://abc123xyzdef.supabase.co' // visible to everyone in your code
const apiKey = 'sk-ant-api03-...' // this is your API key in plain text!
```

GOOD — do this:

```
const supabaseUrl = import.meta.env.VITE_SUPABASE_URL // value comes from .env
const apiKey = import.meta.env.VITE_API_KEY // never in your code
```

■ **HEADS UP**

If you accidentally commit a .env file or hardcoded key: immediately go to that service and regenerate/rotate the key. Old key becomes invalid within seconds.

06

Cloudflare Workers

API protection — manual dashboard, no Wrangler

A Cloudflare Worker is a tiny server that runs between your app and any external API. Your app sends a request to the Worker — the Worker holds your secret API key and forwards the request to the real API. The key never touches your frontend code. This is how you protect Anthropic, OpenAI, and any other paid API keys in your projects.

Your React App → **Cloudflare Worker (holds secret keys)** → **Anthropic / Supabase / Any API**

1 Create a Cloudflare account

Go to cloudflare.com → Sign up for free.

Free tier includes 100,000 Worker requests per day — more than enough.

2 Create a new Worker

Dashboard sidebar: Workers & Pages → Create Application → Create Worker

Give it a meaningful name (e.g. my-app-api-proxy) → Deploy

You'll see a default Hello World worker. We'll replace the code next.

3 Open the code editor

On your Worker page: click Edit Code (top right of the page)

Select all the existing code and delete it — paste your Worker code in.

4 Add your secret keys to the Worker

In the Worker dashboard: Settings tab → Variables → Environment Variables

Click Add Variable → Name: ANTHROPIC_API_KEY, Value: your actual key

Click the Encrypt checkbox → Save and Deploy

Encrypted variables are never visible again after saving — they're stored securely.

5

Note your Worker URL and add it to .env

Your Worker URL appears at the top of the dashboard page.

Format: https://worker-name.your-account.workers.dev

```
VITE_WORKER_URL=https://my-worker.your-account.workers.dev
```

ANTHROPIC API PROXY — WORKER CODE TEMPLATE

Paste this into your Worker editor. It proxies requests to the Anthropic API:

```
export default { async fetch(request, env) { // Handle CORS preflight if (request.method === 'OPTIONS')
{ return new Response(null, { headers: { 'Access-Control-Allow-Origin': '*',
'Access-Control-Allow-Methods': 'POST, OPTIONS', 'Access-Control-Allow-Headers': 'Content-Type', } }) }
const body = await request.json() const response = await fetch('https://api.anthropic.com/v1/messages',
{ method: 'POST', headers: { 'x-api-key': env.ANTHROPIC_API_KEY, 'anthropic-version': '2023-06-01',
'content-type': 'application/json', }, body: JSON.stringify(body), }) const data = await
response.json() return new Response(JSON.stringify(data), { headers: { 'Content-Type':
'application/json', 'Access-Control-Allow-Origin': '*', } }) } }
```

CALLING YOUR WORKER FROM REACT

```
const response = await fetch(import.meta.env.VITE_WORKER_URL, { method: 'POST', headers: {
'Content-Type': 'application/json' }, body: JSON.stringify({ model: 'claude-sonnet-4-6', max_tokens:
1024, messages: [{ role: 'user', content: userMessage }] }) }) const data = await response.json() const
reply = data.content[0].text
```

✓ **PRO TIP**
To update Worker code: paste new code in the editor and click Save and Deploy. No CLI, no Wrangler, no config files needed. The manual dashboard approach works for any project.

07

Vercel Deployment

Shipping your app to the world

Vercel connects to your GitHub repo and automatically redeploys your app every time you push code. It handles building, hosting, SSL certificates, and global CDN distribution — all for free on the Hobby plan. This is the fastest path from code to live URL.

1

Create a Vercel account

Go to vercel.com → Sign up with GitHub (use your GitHub account)

Signing up with GitHub lets Vercel access your repos directly.

2

Import your project

Dashboard → Add New → Project → Import Git Repository

Find your GitHub repo in the list and click Import

Your GitHub account must be connected — authorize Vercel if prompted.

3

Configure build settings

These are usually auto-detected but verify they are correct:

SETTINGS TO CONFIRM

Framework Preset:	Vite
Build Command:	npm run build
Output Directory:	dist
Install Command:	npm install

4

Add your environment variables BEFORE deploying

Expand the Environment Variables section before clicking Deploy

Add each variable from your .env file one at a time:

```
VITE_SUPABASE_URL = https://your-project.supabase.co
VITE_SUPABASE_ANON_KEY = eyJhb...
VITE_WORKER_URL = https://your-worker.workers.dev
```

Missing env vars are the #1 cause of failed Vercel builds.

5

Click Deploy

Vercel builds your app — takes about 60-120 seconds.

When complete you get a live URL: your-project.vercel.app

Share that URL — it's your real production app.

6

Your deploy workflow going forward

EVERY FUTURE UPDATE IS JUST THESE 3 COMMANDS

```
git add .
git commit -m "describe your change"
git push
```

Vercel detects the push and auto-redeploys. Takes about 60 seconds.

Monitor builds in your Vercel dashboard under the Deployments tab.

UPDATING ENVIRONMENT VARIABLES AFTER DEPLOYMENT

If you add a new env variable to .env locally, you must also add it to Vercel. Go to your project in the Vercel dashboard → Settings → Environment Variables → Add. Then trigger a new deployment (push a small change or use Redeploy in the dashboard) to apply the new variable.

✓ **PRO TIP**
Custom domain: Project → Settings → Domains → add your domain. Vercel handles the SSL certificate automatically.

■ **HEADS UP**
Build failed? Click the failed deployment to see logs. 9 times out of 10 it is a missing environment variable or a TypeScript type error that only surfaces during the build.

08

Troubleshooting & Common Fixes

Solutions to issues you will definitely hit

Every developer hits these same walls. This section is a reference for the most common problems across each part of the stack — what causes them and exactly how to fix them.

SUPABASE ISSUES

Problem: Login fails / user can't sign in

Cause: Email confirmation is blocking the login.

Fix — run in Supabase SQL Editor

```
UPDATE auth.users
SET email_confirmed_at = now(), confirmed_at = now()
WHERE email = 'your@email.com';
```

Problem: Row Level Security blocking all queries

Cause: RLS is enabled but no policies exist. Add SELECT/INSERT/UPDATE/DELETE policies for the table or temporarily disable for testing.

Fix — run in Supabase SQL Editor

```
-- Check existing policies:
SELECT * FROM pg_policies WHERE tablename = 'your_table';
-- Temporarily disable RLS for testing only:
ALTER TABLE public.tracks DISABLE ROW LEVEL SECURITY;
```

Problem: Column doesn't exist error after adding fields

Cause: You added the column to your TypeScript types but forgot to run the ALTER TABLE in Supabase SQL Editor.

Fix — run in Supabase SQL Editor

```
ALTER TABLE public.profiles
ADD COLUMN IF NOT EXISTS new_field text;
```

Problem: Supabase storage upload returns 403 Forbidden

Cause: Bucket policy is blocking uploads. Set bucket to public or add an INSERT policy.

Fix — run in Supabase SQL Editor

```
-- Allow authenticated users to upload to bucket:
CREATE POLICY "Allow uploads"
ON storage.objects FOR INSERT
WITH CHECK (bucket_id = 'your-bucket' AND auth.role() = 'authenticated');
```

Problem: Profile not created when user signs up

Cause: The trigger function doesn't exist or wasn't created. Run the trigger SQL from Section 04.

Fix — run in Supabase SQL Editor

```
-- Check if trigger exists:
SELECT * FROM information_schema.triggers
WHERE trigger_name = 'on_auth_user_created';
```

CLOUDFLARE WORKER ISSUES

CORS error in browser console	Your Worker is missing the Access-Control-Allow-Origin header. Add OPTIONS handling and head
Worker returns 500 / internal error	Check Worker logs: Dashboard → Worker → Logs tab. Usually a missing env variable or malforme
Variable env.MY_KEY is undefined	You added the variable but didn't click Save and Deploy after. Go to Settings → Variables → verify
Worker URL returns 404	The Worker hasn't been deployed or the URL is wrong. Check Workers & Pages dashboard for yo
Request reaches Worker but API returns 401	Your API key is wrong or has been rotated. Double check Settings → Variables, delete the old one

VERCEL BUILD ISSUES

Build fails — 'cannot find module'	A package is imported but not installed. Run npm install locally then push again.
Build fails — TypeScript errors	TS errors that your editor ignores will fail Vercel's strict build. Fix the TypeScript error show
App deploys but is blank / white screen	Missing VITE_ env variables. Go to Vercel → Settings → Environment Variables and add t
App works locally but crashes on Vercel	Almost always an env variable issue. Compare your local .env to what's in Vercel Settings
Old version is still showing after deploy	Hard refresh: Ctrl+Shift+R (Windows) or Cmd+Shift+R (Mac). Or check Vercel Deployment
Custom domain not working	DNS hasn't propagated yet — can take up to 24 hours. Check Vercel → Settings → Domai

GIT / GITHUB ISSUES

git push rejected / non-fast-forward	Run git pull first to sync remote changes, then push again.
Pushed node_modules by accident	Add node_modules/ to .gitignore, then: git rm -r --cached node_modules && git commit -
Pushed .env by accident	Add .env to .gitignore, run git rm --cached .env, commit and push. Then rotate ALL keys
Merge conflict	Open the conflicting file, look for <<<< HEAD and ==== markers, manually edit to keep t
fatal: not a git repository	You're not inside your project folder. Use cd to navigate into the project root, then run git

09 Quick Reference Cheatsheet

Everything on one page

TERMINAL

`pwd`	Where am I?
`ls / ls -la`	List files / show hidden
`cd folder`	Enter folder
`cd ..`	Go up one level
`mkdir name`	Create folder
`touch file`	Create file
`rm file / rm -rf folder`	Delete file / folder
`clear`	Clear screen
`Tab`	Auto-complete names
`↑ arrow`	Recall last command

NPM

`npm install`	Install dependencies
`npm run dev`	Start dev server
`npm run build`	Build for production
`npm install pkg`	Add a package
`npm uninstall pkg`	Remove a package
`node -v`	Check Node version

VITE / NEW PROJECT

`npm create vite@latest name`	Create project
`cd name && npm install`	Enter + install
`npm run dev`	localhost:5173
`npm run build`	Creates /dist folder

GIT / GITHUB

`git init`	Start tracking
`git add .`	Stage all changes
`git commit -m "msg"`	Save snapshot
`git push`	Upload to GitHub
`git pull`	Download changes
`git status`	See what changed
`git log --oneline`	View history
`git stash / pop`	Temp save / restore

SUPABASE — SDK

`supabase.auth.signUp()`	Register user
`supabase.auth.signInWithPassword()`	Login

`supabase.auth.signOut()`	Logout
`supabase.auth.getUser()`	Get current user
`.from('t').select('*')`	Read all rows
`.from('t').insert([{...}])`	Add row
`.from('t').update().eq()`	Update row
`.from('t').delete().eq()`	Delete row
`.storage.from('b').upload()`	Upload file
`.storage.from('b').getPublicUrl()`	Get file URL

SUPABASE — SQL FIXES

`UPDATE auth.users SET email_confirmed_at...`	Fix login block
`ALTER TABLE t ADD COLUMN IF NOT EXISTS`	Add column
`ALTER TABLE t ENABLE ROW LEVEL SECURITY`	Enable RLS
`CREATE POLICY ... USING (auth.uid()=user_id)`	Add RLS policy
`SELECT * FROM pg_policies WHERE tablename=`	Check policies
`TRUNCATE TABLE public.table_name`	Clear all rows

CLOUDFLARE WORKERS

`Workers & Pages → Create`	Create Worker
`Edit Code → paste → Save & Deploy`	Update code
`Settings → Variables → Encrypt`	Add secret key
`env.KEY_NAME in Worker code`	Access secrets
`Access-Control-Allow-Origin: *`	Enable CORS
`Logs tab`	Debug errors

VERCEL

`New Project → Import from GitHub`	Deploy app
`Settings → Environment Variables`	Add .env vars
`git push → auto redeploy`	Update live site
`Deployments tab → build logs`	Debug build
`Settings → Domains`	Add custom domain
`Redeploy button`	Force redeploy

THE STACK • Built for developers who learn by doing.
Learn by doing. Ship before perfect. Stack it up.