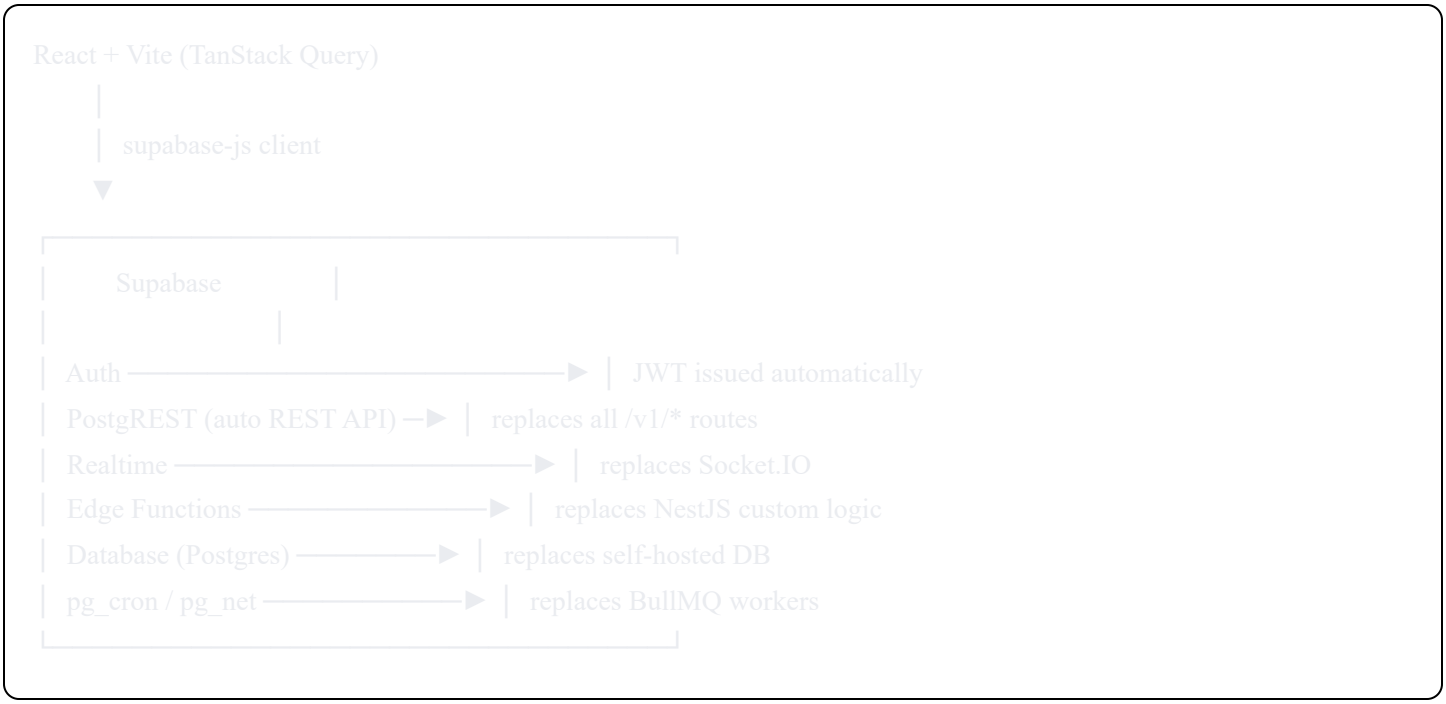
Dento → Supabase Migration Guide

Scope: Replace NestJS API + BullMQ + Socket.IO + custom JWT + self-hosted Postgres/Redis with a fully Supabase-native stack.

New Architecture



What replaces what:

Old	New
NestJS REST API	Supabase PostgREST (auto-generated) + Edge Functions (custom logic)
Custom JWT auth	Supabase Auth
BullMQ + Redis workers	Supabase Edge Functions + `pg_cron` + `pg_net`
Socket.IO realtime	Supabase Realtime (Postgres Changes)
Self-hosted Postgres	Supabase Postgres
Self-hosted Redis	Eliminated
`apps/api/`	Deleted
`apps/workers/`	Deleted
`docker-compose.yml`	Deleted

Phase 0 — Project Setup

1. Create a Supabase project

Go to supabase.com, create a new project. Note your:

- **Project URL** (`https://<ref>.supabase.co`)
- **Anon public key**
- **Service role key** (keep secret, Edge Functions only)
- **Database password**

2. Install the Supabase CLI

```bash
npm install -g supabase
supabase login
supabase init          # run from repo root — creates supabase/ directory
supabase link --project-ref <your-ref>
```

3. New monorepo structure

```
apps/
  web/                # unchanged React + Vite
supabase/
  migrations/         # replaces Prisma migrations
  functions/          # replaces apps/api + apps/workers
    confirm-appointment/
    send-reminder/
    ...
packages/
  types/              # keep — shared Zod schemas still useful
```

Delete `apps/api/`, `apps/workers/`, `docker-compose.yml`.

Phase 1 — Database (Postgres via Supabase)

Migrate your Prisma schema to SQL migrations

Supabase uses plain SQL migrations. Convert your Prisma models:

`supabase/migrations/20240001000000_init.sql`

```sql
```

```sql
-- Enable UUID generation
create extension if not exists "pgcrypto";

-- Patients
create table public.patients (
  id          uuid primary key default gen_random_uuid(),
  name        text not null,
  email       text unique not null,
  phone       text,
  created_at  timestamptz default now(),
  updated_at  timestamptz default now()
);

-- Appointments
create table public.appointments (
  id          uuid primary key default gen_random_uuid(),
  patient_id  uuid not null references public.patients(id) on delete cascade,
  scheduled_at timestamptz not null,
  status      text not null default 'pending'
              check (status in ('pending','confirmed','cancelled','completed')),
  notes       text,
  created_at  timestamptz default now(),
  updated_at  timestamptz default now()
);

-- Jobs (audit log of async work)
create table public.jobs (
  id             uuid primary key default gen_random_uuid(),
  appointment_id uuid references public.appointments(id) on delete set null,
  type           text not null,   -- 'confirmation' | 'reminder'
  status         text not null default 'pending'
                 check (status in ('pending','processing','done','failed')),
  error          text,
  created_at     timestamptz default now(),
  updated_at     timestamptz default now()
);

-- Auto-update updated_at
create or replace function public.set_updated_at()
returns trigger language plpgsql as $$
begin
  new.updated_at = now();
  return new;
```

```
end;
$$;

create trigger patients_updated_at before update on public.patients
  for each row execute function public.set_updated_at();
create trigger appointments_updated_at before update on public.appointments
  for each row execute function public.set_updated_at();
create trigger jobs_updated_at before update on public.jobs
  for each row execute function public.set_updated_at();
```

Push to Supabase:

```
bash

supabase db push
```

> Going forward, create new migrations with `supabase migration new <name>` and push with `supabase db push`.

Phase 2 — Authentication (Supabase Auth)

Supabase Auth fully replaces your `POST /v1/auth/register` and `POST /v1/auth/login` endpoints. It issues JWTs that are automatically validated by PostgREST and Edge Functions.

Row Level Security (RLS)

Enable RLS on all tables. This is the Supabase-native way to enforce authorization — no middleware needed.

```
sql
```

```
-- supabase/migrations/20240001000001_rls.sql

alter table public.patients enable row level security;
alter table public.appointments enable row level security;
alter table public.jobs enable row level security;

-- Example: authenticated users can read all patients (adjust per your access model)
create policy "Authenticated users can read patients"
  on public.patients for select
  to authenticated using (true);

create policy "Authenticated users can insert patients"
  on public.patients for insert
  to authenticated with check (true);

create policy "Authenticated users can update patients"
  on public.patients for update
  to authenticated using (true);

create policy "Authenticated users can delete patients"
  on public.patients for delete
  to authenticated using (true);

-- Repeat similarly for appointments and jobs
```

Frontend: replace custom auth

Remove: `apps/api/src/auth/**`

`apps/web/src/lib/supabase.ts`

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

export const supabase = createClient(
  import.meta.env.VITE_SUPABASE_URL,
  import.meta.env.VITE_SUPABASE_ANON_KEY
)
```

`apps/web/.env`

```
VITE_SUPABASE_URL=https://<ref>.supabase.co
VITE_SUPABASE_ANON_KEY=<anon-key>
```

Register:

ts

```ts
const { data, error } = await supabase.auth.signUp({
  email: 'user@example.com',
  password: 'password',
  options: { data: { name: 'Dr. Smith' } }
})
```

Login:

ts

```ts
const { data, error } = await supabase.auth.signInWithPassword({
  email: 'user@example.com',
  password: 'password'
})
// data.session.access_token is the JWT — supabase-js handles it automatically
```

Logout:

ts

```ts
await supabase.auth.signOut()
```

Auth state in React (replaces JWT interceptors):

ts

```ts
// apps/web/src/hooks/useAuth.ts
import { useEffect, useState } from 'react'
import { supabase } from '../lib/supabase'
import type { Session } from '@supabase/supabase-js'

export function useAuth() {
  const [session, setSession] = useState<Session | null>(null)

  useEffect(() => {
    supabase.auth.getSession().then(({ data }) => setSession(data.session))
    const { data: { subscription } } = supabase.auth.onAuthStateChange((_event, session) => {
      setSession(session)
    })
    return () => subscription.unsubscribe()
  }, [])

  return session
}
```

Phase 3 — REST API (PostgREST replaces NestJS routes)

Supabase auto-generates a REST API from your Postgres schema via PostgREST. Most of your NestJS CRUD routes become zero-code.

Route mapping

Old NestJS route	New Supabase equivalent
`GET /v1/patients`	`supabase.from('patients').select('*')`
`GET /v1/patients/:id`	`supabase.from('patients').select('*, appointments(*)').eq('id', id).single()`
`POST /v1/patients`	`supabase.from('patients').insert(body)`
`PATCH /v1/patients/:id`	`supabase.from('patients').update(body).eq('id', id)`
`DELETE /v1/patients/:id`	`supabase.from('patients').delete().eq('id', id)`
`GET /v1/appointments`	`supabase.from('appointments').select('*')`
`GET /v1/appointments/:id`	`supabase.from('appointments').select('*, patients(*), jobs(*)').eq('id', id).single()`

Old NestJS route	New Supabase equivalent
`GET /v1/jobs`	`supabase.from('jobs').select('*')`
`GET /v1/jobs/:id`	`supabase.from('jobs').select('*').eq('id', id).single()`

Example TanStack Query hook (patients):

ts

```ts
// apps/web/src/hooks/usePatients.ts
import { useQuery, useMutation, useQueryClient } from '@tanstack/react-query'
import { supabase } from '../lib/supabase'

export function usePatients() {
  return useQuery({
    queryKey: ['patients'],
    queryFn: async () => {
      const { data, error } = await supabase.from('patients').select('*')
      if (error) throw error
      return data
    }
  })
}

export function useCreatePatient() {
  const qc = useQueryClient()
  return useMutation({
    mutationFn: async (payload: { name: string; email: string; phone?: string }) => {
      const { data, error } = await supabase.from('patients').insert(payload).select().single()
      if (error) throw error
      return data
    },
    onSuccess: () => qc.invalidateQueries({ queryKey: ['patients'] })
  })
}
```

POST /v1/appointments enqueued a confirmation job. This now goes through an Edge Function (see Phase 4).

Phase 4 — Edge Functions (replace NestJS custom logic + BullMQ workers)

Edge Functions (Deno) handle all business logic that PostgREST can't — job enqueueing, appointment creation side-effects, sending reminders.

Create the functions

```bash
supabase functions new confirm-appointment
supabase functions new send-reminder
```

supabase/functions/confirm-appointment/index.ts

This replaces `POST /v1/appointments` (creates appointment + enqueues confirmation job).

```ts
```

```
import { createClient } from 'https://esm.sh/@supabase/supabase-js@2'

Deno.serve(async (req) => {
  if (req.method !== 'POST') {
    return new Response('Method not allowed', { status: 405 })
  }

  // Auth: validate the JWT from Authorization header
  const authHeader = req.headers.get('Authorization')
  if (!authHeader) return new Response('Unauthorized', { status: 401 })

  const supabase = createClient(
    Deno.env.get('SUPABASE_URL')!,
    Deno.env.get('SUPABASE_SERVICE_ROLE_KEY')!
  )

  // Verify JWT
  const token = authHeader.replace('Bearer ', '')
  const { data: { user }, error: authError } = await supabase.auth.getUser(token)
  if (authError || !user) return new Response('Unauthorized', { status: 401 })

  const body = await req.json()

  // 1. Create appointment
  const { data: appointment, error: apptError } = await supabase
    .from('appointments')
    .insert({
      patient_id: body.patient_id,
      scheduled_at: body.scheduled_at,
      notes: body.notes,
    })
    .select()
    .single()

  if (apptError) {
    return new Response(JSON.stringify({ error: apptError.message }), {
      status: 400,
      headers: { 'Content-Type': 'application/json' }
    })
  }

  // 2. Create a job record (triggers Realtime event automatically)
  const { data: job, error: jobError } = await supabase
```

```
    .from('jobs')
    .insert({
      appointment_id: appointment.id,
      type: 'confirmation',
      status: 'processing',
    })
    .select()
    .single()

  if (jobError) {
    return new Response(JSON.stringify({ error: jobError.message }), { status: 500 })
  }

  // 3. Do the actual "confirmation" work (e.g. send email via Resend/SendGrid)
  try {
    // await sendConfirmationEmail(appointment)   <-- plug in your email provider

    await supabase
      .from('jobs')
      .update({ status: 'done' })
      .eq('id', job.id)

    await supabase
      .from('appointments')
      .update({ status: 'confirmed' })
      .eq('id', appointment.id)
  } catch (err) {
    await supabase.from('jobs').update({ status: 'failed', error: String(err) }).eq('id', job.id)
  }

  return new Response(JSON.stringify({ appointment, job }), {
    status: 201,
    headers: { 'Content-Type': 'application/json' }
  })
})
```

supabase/functions/send-reminder/index.ts

Replaces `POST /v1/appointments/:id/reminder`.

ts

```
import { createClient } from 'https://esm.sh/@supabase/supabase-js@2'

Deno.serve(async (req) => {
  if (req.method !== 'POST') return new Response('Method not allowed', { status: 405 })

  const authHeader = req.headers.get('Authorization')
  if (!authHeader) return new Response('Unauthorized', { status: 401 })

  const supabase = createClient(
    Deno.env.get('SUPABASE_URL')!,
    Deno.env.get('SUPABASE_SERVICE_ROLE_KEY')!
  )

  const token = authHeader.replace('Bearer ', '')
  const { data: { user }, error: authError } = await supabase.auth.getUser(token)
  if (authError || !user) return new Response('Unauthorized', { status: 401 })

  const url = new URL(req.url)
  const appointmentId = url.pathname.split('/').at(-2) // …/:id/reminder

  const { data: appointment, error } = await supabase
    .from('appointments')
    .select('*, patients(*)')
    .eq('id', appointmentId)
    .single()

  if (error || !appointment) return new Response('Not found', { status: 404 })

  const { data: job } = await supabase
    .from('jobs')
    .insert({ appointment_id: appointment.id, type: 'reminder', status: 'processing' })
    .select()
    .single()

  try {
    // await sendReminderEmail(appointment)   <-- plug in your email provider

    await supabase.from('jobs').update({ status: 'done' }).eq('id', job!.id)
  } catch (err) {
    await supabase.from('jobs').update({ status: 'failed', error: String(err) }).eq('id', job!.id)
  }

  return new Response(JSON.stringify({ job }), {
```

```
    status: 200,
    headers: { 'Content-Type': 'application/json' }
  })
})
```

Deploy functions

```bash
supabase functions deploy confirm-appointment
supabase functions deploy send-reminder
```

Set secrets

```bash
supabase secrets set SUPABASE_SERVICE_ROLE_KEY=<service-role-key>
# Add any email provider keys:
supabase secrets set RESEND_API_KEY=<key>
```

Call Edge Functions from the frontend

```ts
// Create appointment (replaces POST /v1/appointments)
const { data, error } = await supabase.functions.invoke('confirm-appointment', {
  body: { patient_id, scheduled_at, notes }
})

// Send reminder (replaces POST /v1/appointments/:id/reminder)
const { data, error } = await supabase.functions.invoke('send-reminder', {
  body: {},
  headers: { 'x-appointment-id': appointmentId }  // or embed in URL
})
```

Scheduled jobs with pg_cron (replaces BullMQ scheduled tasks)

For recurring work (e.g. send reminders 24h before appointments):

```sql
```

```sql
-- supabase/migrations/20240001000002_cron.sql

-- Enable pg_cron (already available on Supabase)
-- Note: pg_cron runs in the pg_cron schema, must use net.http_post to call Edge Functions

select cron.schedule(
  'send-daily-reminders',
  '0 8 * * *',   -- every day at 8am UTC
  $$
    select net.http_post(
      url := 'https://<ref>.supabase.co/functions/v1/send-reminder',
      headers := '{"Authorization": "Bearer <service-role-key>", "Content-Type": "application/json"}'::jsonb,
      body := '{}'::jsonb
    )
  $$
);
```

Phase 5 — Realtime (replace Socket.IO)

Supabase Realtime listens to Postgres changes and streams them to connected clients — no separate WebSocket server needed.

Enable Realtime on your tables

```sql
-- supabase/migrations/20240001000003_realtime.sql
alter publication supabase_realtime add table public.appointments;
alter publication supabase_realtime add table public.jobs;
```

Frontend: replace Socket.IO listeners

Remove: All `io()` socket setup and `socket.on(...)` handlers.

`apps/web/src/hooks/useRealtimeSync.ts`

```ts
```

```
import { useEffect } from 'react'
import { useQueryClient } from '@tanstack/react-query'
import { supabase } from '../lib/supabase'

export function useRealtimeSync() {
  const qc = useQueryClient()

  useEffect(() => {
    const channel = supabase
      .channel('db-changes')
      .on(
        'postgres_changes',
        { event: '*', schema: 'public', table: 'appointments' },
        (payload) => {
          console.log('appointment change:', payload)
          qc.invalidateQueries({ queryKey: ['appointments'] })
          qc.invalidateQueries({ queryKey: ['appointment', payload.new?.id] })
        }
      )
      .on(
        'postgres_changes',
        { event: '*', schema: 'public', table: 'jobs' },
        (payload) => {
          console.log('job change:', payload)
          qc.invalidateQueries({ queryKey: ['jobs'] })
        }
      )
      .subscribe()

    return () => {
      supabase.removeChannel(channel)
    }
  }, [qc])
}
```

Mount this once at app root (e.g. in `App.tsx`):

tsx

```
function App() {
  useRealtimeSync()
  return <RouterProvider router={router} />
}
```

This is a direct replacement for your old `job:update` and `appointment:update` Socket.IO events — any DB write (whether from a user action or an Edge Function) triggers a Realtime event, which invalidates the relevant queries.

Phase 6 — Frontend Wiring Summary

Update `apps/web/.env`:

```
VITE_SUPABASE_URL=https://<ref>.supabase.co
VITE_SUPABASE_ANON_KEY=<anon-key>
```

Remove from `apps/web`:

- All `axios` / `fetch` calls to `localhost:3000`
- All `socket.io-client` imports and socket setup
- Any custom JWT storage / interceptors

Update `apps/web/package.json`:

```bash
pnpm add @supabase/supabase-js
pnpm remove socket.io-client axios   # or whatever HTTP client you used
```

Phase 7 — Remove the Old Stack

```bash
# Delete old apps
rm -rf apps/api
rm -rf apps/workers
rm -f docker-compose.yml

# Remove old root scripts from package.json
# (pnpm dev should now only start the web app)
```

Update root `package.json` scripts:

```json
```

```
{
  "scripts": {
    "dev": "pnpm --filter web dev",
    "build": "pnpm --filter web build",
    "supabase:start": "supabase start",
    "supabase:deploy": "supabase db push && supabase functions deploy"
  }
}
```

Local Development

Supabase has a full local emulator that replaces Docker Compose:

```bash
supabase start
# Starts local Postgres, Auth, PostgREST, Realtime, Edge Functions, Studio
```

Service	Local URL
Studio (DB admin)	http://localhost:54323
API / PostgREST	http://localhost:54321
Auth	http://localhost:54321/auth/v1
Edge Functions	http://localhost:54321/functions/v1
Inbucket (email testing)	http://localhost:54324

Update `apps/web/.env.local` for local dev:

```
VITE_SUPABASE_URL=http://localhost:54321
VITE_SUPABASE_ANON_KEY=<local-anon-key>  # printed by `supabase start`
```

Stop local stack:

```bash
supabase stop
```

Health Check

Your old `GET /health` endpoint can be replaced by checking Supabase Auth reachability:

```ts
const { data } = await supabase.auth.getSession()
// If this resolves without network error, the stack is up
```

Or just remove the health check — Supabase handles uptime monitoring on their side.

Migration Checklist

- [ ] Supabase project created, CLI installed and linked
- [ ] SQL migrations written and pushed (`supabase db push`)
- [ ] RLS policies enabled on all tables
- [ ] Supabase Auth replacing custom JWT auth
- [ ] `supabase-js` client added to `apps/web`
- [ ] TanStack Query hooks updated to use `supabase.from()` instead of REST calls
- [ ] Edge Functions created for `confirm-appointment` and `send-reminder`
- [ ] Edge Functions deployed with secrets set
- [ ] Realtime enabled on `appointments` and `jobs` tables
- [ ] `useRealtimeSync` hook replacing Socket.IO listeners
- [ ] `apps/api/`, `apps/workers/`, `docker-compose.yml` deleted
- [ ] Root `package.json` scripts updated
- [ ] Local dev verified with `supabase start`
- [ ] Production env vars set in Supabase dashboard