

Les 18 — Supabase Auth + RLS

Multi-user apps met per-user data isolation — productie ready

Vak:	AI-Assisted Development
Klas:	Klas A (3 uur fysiek)
Vervolg op:	Les 17 — Externe APIs in diepte
Volgende les:	Geen — eindopdracht thuis

Leerdoelen

- Verschil Auth (wie ben jij) vs Authz (wat mag jij)
- Drie Supabase Auth methoden kennen: password, magic link, social
- Magic link flow opzetten in Next.js
- Social login (GitHub OAuth) configureren
- Supabase clients voor server + client + middleware
- RLS policies schrijven voor multi-user data
- Protected routes met Server Components
- Eindopdracht-checklist toepassen

Inhoud

1. Auth vs Authz
2. Supabase Auth methoden
3. Magic link flow
4. Social login (OAuth)
5. Session in Next.js App Router
6. Middleware — sessie refresh
7. RLS — wat en waarom
8. RLS policies in praktijk
9. Protected routes
10. Eindopdracht checklist
11. Bronnen

1 Auth vs Authz

- **Auth** — 'wie ben jij?' login flow
- **Authz** — 'wat mag jij?' permissions

In Supabase: Auth via magic link/social/password. Authz via RLS in Postgres.

2 Supabase Auth — drie methoden

Email + password

```
await supabase.auth.signUp({ email, password });
await supabase.auth.signInWithPassword({ email, password });
await supabase.auth.signOut();
```

Magic link (aanrader)

```
await supabase.auth.signInWithOtp({
  email,
  options: { emailRedirectTo: `${origin}/auth/callback` },
});
```

Social (OAuth)

```
await supabase.auth.signInWithOAuth({
  provider: "github",
  options: { redirectTo: `${origin}/auth/callback` },
});
```

3

Magic link flow

1. User opent /login en vult email in
2. Server: `supabase.auth.signInWithOtp({ email })`
3. Supabase verstuurt email met magic link
4. User klikt link
5. Supabase verifies, redirect naar `/auth/callback?code=...`
6. Callback route: exchange code voor sessie
7. Cookie geset, user is ingelogd

Callback route

```
// app/auth/callback/route.ts
import { createClient } from "@utils/supabase/server";
import { NextResponse } from "next/server";

export async function GET(request: Request) {
  const { searchParams, origin } = new URL(request.url);
  const code = searchParams.get("code");
  if (code) {
    const supabase = await createClient();
    await supabase.auth.exchangeCodeForSession(code);
  }
  return NextResponse.redirect(`${origin}/`);
}
```

4 Social login (OAuth)

Setup GitHub OAuth App

1. GitHub Settings -> Developer settings -> OAuth Apps -> New
2. Callback URL: `https://[project].supabase.co/auth/v1/callback`
3. Save -> kopieer Client ID + Client Secret

Configure in Supabase

- Auth -> Providers -> GitHub -> Enable
- Plak Client ID + Client Secret
- Save

Frontend

```
async function signInWithGitHub() {
  const supabase = createClient();
  await supabase.auth.signInWithOAuth({
    provider: "github",
    options: { redirectTo: `${origin}/auth/callback` },
  });
}
```

Supabase supports 20+ providers: Google, GitHub, Discord, Apple, Microsoft, etc.

5 Session in Next.js App Router

```
pnpm add @supabase/supabase-js @supabase/ssr
```

Vier files

1. `utils/supabase/client.ts` — voor Client Components
2. `utils/supabase/server.ts` — voor Server Components + actions
3. `utils/supabase/middleware.ts` — refresh sessie elke request
4. `middleware.ts` (root) — gebruikt #3

Server client

```
import { createServerClient } from "@supabase/ssr";
import { cookies } from "next/headers";

export async function createClient() {
  const cookieStore = await cookies();
  return createServerClient(URL, KEY, {
    cookies: {
      getAll() { return cookieStore.getAll(); },
      setAll(items) { /* set in cookieStore */ },
    },
  });
}
```

6 Middleware — sessie refresh

Sessies verlopen na een uur (default). Middleware refreshed automatisch.

```
// middleware.ts (root)
import { type NextRequest } from "next/server";
import { updateSession } from "@utils/supabase/middleware";

export async function middleware(request: NextRequest) {
  return await updateSession(request);
}

export const config = {
  matcher: ["/(?!_next/static|_next/image|favicon.ico).*"],
};
```

Elke request → middleware refreshed sessie → user blijft ingelogd.

7 RLS — wat en waarom

Zonder RLS

```
const { data } = await supabase.from('tasks').select('*');
```

→ Returns ALL tasks from ALL users — disaster.

Met RLS

```
alter table tasks enable row level security;
create policy "select own" on tasks for select using (auth.uid() = user_id);

// Now:
const { data } = await supabase.from("tasks").select("*");
// → Returns alleen tasks waar user_id = auth.uid()
```

Postgres past filter ALTIJD toe, ongeacht waar query vandaan komt.

auth.uid()

- UUID van ingelogde user (uit JWT)
- NULL als anonymous
- Te gebruiken in WHERE, defaults, custom functions

8 RLS policies in praktijk

Basis: per-user data

```
create table tasks (  
  id bigserial primary key,  
  user_id uuid not null references auth.users(id) default auth.uid(),  
  text text not null,  
  done boolean default false  
);  
  
alter table tasks enable row level security;  
  
create policy "select own" on tasks  
  for select using (auth.uid() = user_id);  
create policy "insert own" on tasks  
  for insert with check (auth.uid() = user_id);  
create policy "update own" on tasks  
  for update using (auth.uid() = user_id);  
create policy "delete own" on tasks  
  for delete using (auth.uid() = user_id);
```

using = check tijdens read. with check = check tijdens write.

Variant: team feature

```
create policy "see team tasks" on tasks  
  for select using (  
    team_id in (  
      select team_id from team_members where user_id = auth.uid()  
    )  
  );
```

Best practices

- Begin met enable row level security
- Default-deny: zonder policies kan niemand iets
- Per-operation policies: select/insert/update/delete apart
- Test in incognito — check zonder admin-context
- Hou queries simpel — complexe policies = trage queries

9

Protected routes + server checks

Protected page (Server Component)

```
import { createClient } from "@utils/supabase/server";
import { redirect } from "next/navigation";

export default async function Dashboard() {
  const supabase = await createClient();
  const { data: { user } } = await supabase.auth.getUser();
  if (!user) redirect("/login");
  return <div>Welkom {user.email}</div>;
}
```

Protected API route

```
export async function POST(req: Request) {
  const supabase = await createClient();
  const { data: { user } } = await supabase.auth.getUser();
  if (!user) return new Response("Unauthorized", { status: 401 });
  // handle request
}
```

Logout

```
"use client";
async function logout() {
  const supabase = createClient();
  await supabase.auth.signOut();
  window.location.href = "/login";
}
```

10

Eindopdracht checklist

Technische stack

- Next.js 16 + TypeScript + Tailwind
- Supabase Postgres
- Vercel AI SDK met Tool Calling
- Minstens 1 externe API
- Multi-user — Supabase Auth + RLS
- Deployed op Vercel

- Code op GitHub met PR's + CI

Feature must-haves

- Login flow (magic link of social)
- Per-user data (RLS policies actief)
- AI feature met tools (chat, agent, RAG, etc.)
- Externe API integratie (geen alleen-OpenAI)
- Productie URL die werkt

Tijd: 40-60 uur thuiswerk. Verspreid over 4-6 weken. Niet laatste weekend.

11

Bronnen

- Supabase Auth — supabase.com/docs/guides/auth
- Next.js setup — supabase.com/docs/guides/auth/server-side/nextjs
- Magic links — supabase.com/docs/guides/auth/auth-magic-link
- Social login — supabase.com/docs/guides/auth/social-login
- RLS guide — supabase.com/docs/guides/database/postgres/row-level-security
- Custom claims — supabase.com/docs/guides/database/postgres/custom-claims-and-role-based-access-control-rbac