

# Les 10 — Supabase Auth & RLS

Lesstof — Authenticatie, Sessies, Middleware en Row Level Security

**Vak:** AI-Assisted Development

**Opleiding:** NOVI Hogeschool Utrecht

**Docent:** Tim

**Tech stack:** Next.js 16, TypeScript, Tailwind CSS, Supabase

## Leerdoelen

Na het lezen van dit document kun je:

- Het verschil uitleggen tussen authenticatie en autorisatie
- Supabase Auth opzetten met email/wachtwoord en magic links
- Uitleggen hoe JWT tokens en cookies samenwerken voor sessies
- De browser client en server client aanmaken met `@supabase/ssr`
- Middleware schrijven die sessies vernieuwt en routes beschermt
- Row Level Security (RLS) inschakelen en policies schrijven

## Inhoud

1. Authenticatie vs Autorisatie
2. Supabase Auth — drie login-methodes
3. Sessies en JWT Tokens
4. `@supabase/ssr` in Next.js
5. Middleware — de bewaker van je app
6. Auth Callback Route
7. Beschermd Routes & Conditional UI
8. Row Level Security (RLS)

## 1

## Authenticatie vs Autorisatie

Authenticatie en autorisatie zijn twee termen die vaak door elkaar worden gehaald, maar ze betekenen iets heel anders. Beide zijn essentieel voor een veilige applicatie.

	Authenticatie	Autorisatie
<b>Vraag</b>	WIE ben je?	WAT mag je doen?
<b>Doel</b>	Bewijs dat je bent wie je zegt	Bepaal welke acties je mag uitvoeren
<b>Voorbeeld</b>	Inloggen met email + wachtwoord	Alleen je eigen polls mogen verwijderen
<b>Wanneer?</b>	Bij het inloggen	Na het inloggen, bij elke actie
<b>Techniek vandaag</b>	Supabase Auth	Row Level Security (RLS)

**Vergelijking uit het dagelijks leven:** als je naar een festival gaat, dan is je ID-bewijs de **authenticatie** — je bewijst wie je bent. Je ticket is de **autorisatie** — het bepaalt of je naar binnen mag en welke gebieden je in mag.

In deze les bouwen we allebei: authenticatie met **Supabase Auth** (wie ben je?) en autorisatie met **Row Level Security** (wat mag je?).

## 2

## Supabase Auth

Supabase heeft een ingebouwd authenticatiesysteem. Je hoeft geen eigen login-systeem te bouwen — Supabase regelt alles: wachtwoorden hashen, sessies beheren, tokens genereren. Er zijn drie hoofdmethodes:

Methode	Hoe werkt het?	Wanneer gebruiken?
<b>Email + Wachtwoord</b>	Gebruiker maakt account met email en wachtwoord. Supabase hashet het wachtwoord en slaat het veilig op.	De standaard keuze. Werkt voor 90% van de apps.
<b>Magic Link</b>	Gebruiker vult alleen email in en krijgt een link per mail. Klik op de link = ingelogd. Geen wachtwoord nodig.	Wanneer je geen wachtwoorden wilt beheren. Heel veilig.

Methode	Hoe werkt het?	Wanneer gebruiken?
<b>Google OAuth</b>	Gebruiker klikt op 'Login met Google', wordt doorgestuurd naar Google, en komt terug in jouw app.	Wanneer je social login wilt. Iets complexer om op te zetten.

## Supabase Auth functies

Functie	Wat doet het?
<code>supabase.auth.signUp()</code>	Nieuw account aanmaken met email + wachtwoord
<code>supabase.auth.signInWithPassword()</code>	Inloggen met email + wachtwoord
<code>supabase.auth.signInWithOtp()</code>	Magic link versturen naar emailadres
<code>supabase.auth.signOut()</code>	Uitloggen en sessie verwijderen
<code>supabase.auth.getUser()</code>	Huidige ingelogde gebruiker ophalen

**Tip:** voor development kun je email-bevestiging uitschakelen in het Supabase dashboard via Authentication > Providers > Email > Confirm email (toggle uit). In productie laat je dit aan staan!

## 3

## Sessies en JWT Tokens

Als je inlogt bij Supabase, krijg je een **JWT** terug — een JSON Web Token. Dat is een lange string, een soort pasje, dat bewijst dat jij ingelogd bent. Die token wordt opgeslagen als **cookie** in je browser, zodat je niet bij elke pagina opnieuw hoeft in te loggen.

### Hoe werkt de sessie-flow?

- 1 Login** — Gebruiker logt in met email + wachtwoord
- 2 Check** — Supabase controleert de gegevens
- 3 Token** — Supabase stuurt een JWT token terug
- 4 Cookie** — De token wordt opgeslagen als cookie in de browser
- 5 Verzoek** — Bij elk volgend verzoek stuurt de browser de cookie mee
- 6 Sessie** — De server leest de cookie, checkt de token, en weet wie je bent

## Wat zit er in een JWT?

Onderdeel	Wat bevat het?	Voorbeeld
Header	Type token en algoritme	Algoritme: HS256
Payload	Gebruikersdata: user ID, email, rol, verlooptijd	sub: "abc-123", email: "tim@novi.nl"
Signature	Handtekening om te bewijzen dat de token niet is aangepast	Versleutelde hash van header + payload

## Waarom cookies?

Cookies worden **automatisch meegestuurd** bij elk verzoek naar de server. Dat betekent dat de server altijd weet wie je bent, zonder dat je de token handmatig hoeft mee te sturen. Bovendien kun je cookies lezen in Next.js middleware en Server Components — dat is essentieel voor server-side rendering.

## 4

## @supabase/ssr in Next.js

Next.js draait zowel op de **server** als in de **browser**. Op de server moet je cookies op een andere manier lezen en schrijven dan in de browser. Het package `@supabase/ssr` regelt dat voor je. Installeer het met:

```
# Terminal
npm install @supabase/ssr @supabase/supabase-js
```

## Browser Client

De browser client wordt gebruikt in **Client Components** — componenten die in de browser draaien. Het is het simpelste bestand:

```
# src/lib/supabase/client.ts
import { createBrowserClient } from '@supabase/ssr'

export function createClient() {
  return createBrowserClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!
  )
}
```

## Server Client

De server client wordt gebruikt in **Server Components** en **Server Actions**. Het gebruikt `cookies()` van `next/headers` om cookies te lezen en schrijven op de server:

```
# src/lib/supabase/server.ts
import { createServerClient } from '@supabase/ssr'
import { cookies } from 'next/headers'

export async function createClient() {
  const cookieStore = await cookies()

  return createServerClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!,
    {
      cookies: {
        getAll() {
          return cookieStore.getAll()
        },
        setAll(cookiesToSet) {
          cookiesToSet.forEach(({ name, value, options }) =>
            cookieStore.set(name, value, options)
          )
        },
      },
    }
  )
}
```

**Let op:** de server client functie is `async` omdat `cookies()` asynchroon is in Next.js 16. Vergeet niet `await createClient()` te gebruiken!

## 5 Middleware

Middleware is code die draait **voor elk verzoek** — elke keer als een gebruiker een pagina opent. Denk aan de middleware als een **beveiliging bij de deur**: elke bezoeker wordt gecheckt voordat ze naar binnen mogen.

### Wat doet de middleware?

- **Sessie vernieuwen:** als de JWT token bijna verlopen is, vernieuwt de middleware deze automatisch
- **Routes beschermen:** als er geen geldige sessie is, redirect de middleware naar de login pagina
- **Cookies bijwerken:** de vernieuwde token wordt opgeslagen als nieuwe cookie

## De middleware code

```
# src/middleware.ts
import { createServerClient } from '@supabase/ssr'
import { NextResponse, type NextRequest } from 'next/server'

export async function middleware(request: NextRequest) {
  let supabaseResponse = NextResponse.next({ request })

  const supabase = createServerClient(
    process.env.NEXT_PUBLIC_SUPABASE_URL!,
    process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!,
    {
      cookies: {
        getAll() {
          return request.cookies.getAll()
        },
        setAll(cookiesToSet) {
          cookiesToSet.forEach(({ name, value }) =>
            request.cookies.set(name, value)
          )
          supabaseResponse = NextResponse.next({ request })
          cookiesToSet.forEach(({ name, value, options }) =>
            supabaseResponse.cookies.set(name, value, options)
          )
        },
      },
    }
  )

  // Haal de user op om de sessie te vernieuwen
  const { data: { user } } = await supabase.auth.getUser()

  // Geen user? Redirect naar login
  if (
    !user &&
    !request.nextUrl.pathname.startsWith('/login') &&
    !request.nextUrl.pathname.startsWith('/auth')
  ) {
    const url = request.nextUrl.clone()
    url.pathname = '/login'
    return NextResponse.redirect(url)
  }

  return supabaseResponse
}

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

**Belangrijk:** het middleware bestand staat in `src/middleware.ts`, NIET in `src/app/`. Het is een speciaal Next.js bestand. De `matcher` config bepaalt voor welke URLs de middleware draait — we sluiten statische bestanden en de login/auth pagina's uit.

## 6

## Auth Callback Route

De auth callback route is nodig voor **magic links** en **OAuth redirects**. Als een gebruiker op een magic link klikt, stuurt Supabase ze naar `/auth/callback?code=abc123`. Deze route pakt die code, wisselt het om voor een sessie, en redirect de gebruiker naar de homepage.

```
# src/app/auth/callback/route.ts
import { createClient } from '@lib/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)
}
```

### Waarom is dit nodig?

Magic links en OAuth providers werken met een **code exchange** patroon. De gebruiker komt terug met een tijdelijke code in de URL. Die code moet worden omgezet naar een echte sessie via `exchangeCodeForSession()`. Zonder deze route werken magic links en OAuth niet.

**Dit is een API route** — het toont geen pagina maar handelt een verzoek af. Het bestand moet in `src/app/auth/callback/route.ts` staan.

## 7

## Beschermde Routes & Conditional UI

Er zijn twee manieren om je app te beveiligen op de frontend: **routes beschermen** (gebruikers die niet zijn ingelogd kunnen bepaalde pagina's niet bereiken) en **conditional UI** (de interface aanpassen op basis van de inlogstatus).

### Routes beschermen met middleware

De middleware die we in sectie 5 hebben geschreven doet dit al: als er geen geldige sessie is, wordt de gebruiker geredirect naar `/login`. De `matcher` config bepaalt welke routes beschermd zijn.

## Conditional UI — de Navbar

Je kunt de UI aanpassen op basis van de inlogstatus. Bijvoorbeeld: het emailadres tonen en een uitlog-knop aanbieden als de gebruiker is ingelogd:

```
# src/components/Navbar.tsx (vereenvoudigd)
'use client'

import { createClient } from '@lib/supabase/client'
import { useRouter } from 'next/navigation'
import { useEffect, useState } from 'react'
import type { User } from '@supabase/supabase-js'

export default function Navbar() {
  const [user, setUser] = useState<User | null>(null)
  const router = useRouter()
  const supabase = createClient()

  useEffect(() => {
    const getUser = async () => {
      const { data: { user } } = await supabase.auth.getUser()
      setUser(user)
    }
    getUser()
  }, [])

  const handleSignOut = async () => {
    await supabase.auth.signOut()
    router.push('/login')
    router.refresh()
  }

  return (
    <nav>
      {user && (
        <div>
          <span>{user.email}</span>
          <button onClick={handleSignOut}>Uitloggen</button>
        </div>
      )}
    </nav>
  )
}
```

**Patroon:** gebruik `useEffect` om de user op te halen als de component laadt. Toon UI-elementen conditioneel op basis van of `user` null is of niet. Gebruik `router.refresh()` na sign-out om de server-side state te resetten.



## 8

## Row Level Security (RLS)

RLS is een feature van **PostgreSQL** — de database zelf. Het betekent dat je regels kunt instellen op de database die bepalen wie welke rijen mag lezen, aanmaken, updaten of verwijderen. Dit is de **autorisatie**-kant van het verhaal.

### Zonder vs. met RLS

	Zonder RLS	Met RLS
<b>Beveiliging</b>	Iedereen met de URL en anon key kan alles doen	Elke query wordt gecheckt tegen policies
<b>Risico</b>	Iemand kan al je data lezen, wijzigen of verwijderen	Alleen toegestane acties zijn mogelijk
<b>Standaard</b>	Alles is toegestaan	Alles is geblokkeerd (tenzij een policy het toestaat)

### RLS inschakelen

```
# SQL Editor in Supabase Dashboard
-- Stap 1: RLS inschakelen
ALTER TABLE polls ENABLE ROW LEVEL SECURITY;
ALTER TABLE options ENABLE ROW LEVEL SECURITY;

-- Na deze stap is ALLES geblokkeerd!
-- Je moet policies toevoegen om acties toe te staan.
```

**Belangrijk:** als je RLS inschakelt zonder policies, is alles geblokkeerd. Je ziet dan geen data meer in je app! Voeg altijd direct policies toe na het inschakelen.

### Policies schrijven

Een policy is een regel die bepaalt wie wat mag. Je schrijft ze in SQL. Hier zijn de vier soorten policies:

#### SELECT — lezen

```
-- Iedereen kan polls lezen
CREATE POLICY "Polls are viewable by everyone"
ON polls FOR SELECT
USING (true);
```

## INSERT — aanmaken

```
-- Alleen ingelogde gebruikers kunnen polls aanmaken
CREATE POLICY "Authenticated users can create polls"
ON polls FOR INSERT
TO authenticated
WITH CHECK (true);
```

## UPDATE — wijzigen

```
-- Ingelogde gebruikers kunnen stemmen
CREATE POLICY "Authenticated users can vote"
ON options FOR UPDATE
TO authenticated
USING (true);
```

## DELETE — verwijderen (met auth.uid())

```
-- Gebruikers mogen alleen hun eigen posts verwijderen
CREATE POLICY "Users can delete own posts"
ON posts FOR DELETE
TO authenticated
USING (auth.uid() = user_id);
```

## De auth.uid() functie

`auth.uid()` is een speciale PostgreSQL-functie die Supabase toevoegt. Het geeft het **UUID** terug van de ingelogde gebruiker. Als niemand is ingelogd, geeft het `NULL` terug. Hiermee kun je regels schrijven als: "een gebruiker mag alleen zijn eigen data bewerken".

## Voorbeeld: eigen data beschermen

```
-- Gebruikers mogen alleen hun eigen rijen updaten
CREATE POLICY "Users can only edit own posts"
ON posts FOR UPDATE
TO authenticated
USING (auth.uid() = user_id)
WITH CHECK (auth.uid() = user_id);
```

SQL clause	Betekenis	Bij welke operaties?
<code>USING (...)</code>	Welke bestaande rijen mag je zien/wijzigen?	SELECT, UPDATE, DELETE
<code>WITH CHECK (...)</code>	Welke nieuwe rijen mag je toevoegen/wijzigen?	INSERT, UPDATE

SQL clause	Betekenis	Bij welke operaties?
TO authenticated	Alleen voor ingelogde gebruikers	Alle operaties
auth.uid()	UUID van de ingelogde gebruiker	In USING en WITH CHECK

## Samenvatting: 5 Key Takeaways

### Auth vs Autorisatie

Authenticatie = wie ben je (Supabase Auth). Autorisatie = wat mag je (RLS). Je hebt beide nodig voor een veilige app.

### JWT & Cookies

Na het inloggen krijg je een JWT token die als cookie wordt opgeslagen. De middleware vernieuwt deze automatisch bij elk verzoek.

### Twee Clients

Browser client (createBrowserClient) voor Client Components. Server client (createServerClient) voor Server Components. Middleware heeft een eigen client.

### Middleware

De middleware is de bewaker van je app. Het draait voor elk verzoek, vernieuwt de sessie, en beschermt routes. Bestand staat in src/middleware.ts.

### Row Level Security

RLS beschermt je data op database-niveau. Zonder policies is alles geblokkeerd. Gebruik auth.uid() om eigen data te beschermen.

## Quick Reference

Bestand / Commando	Doel
<code>npm install @supabase/ssr @supabase/supabase-js</code>	Installeer auth packages
<code>src/lib/supabase/client.ts</code>	Browser client (Client Components)
<code>src/lib/supabase/server.ts</code>	Server client (Server Components)
<code>src/middleware.ts</code>	Middleware: sessie vernieuwen + routes beschermen
<code>src/app/auth/callback/route.ts</code>	Auth callback voor magic links / OAuth
<code>supabase.auth.signUp()</code>	Account aanmaken
<code>supabase.auth.signInWithPassword()</code>	Inloggen met wachtwoord
<code>supabase.auth.signInWithOtp()</code>	Magic link versturen
<code>supabase.auth.signOut()</code>	Uitloggen
<code>supabase.auth.getUser()</code>	Huidige user ophalen
<code>ALTER TABLE ... ENABLE ROW LEVEL SECURITY</code>	RLS inschakelen op een tabel
<code>CREATE POLICY ... ON ... FOR SELECT</code>	Lees-policy aanmaken
<code>TO authenticated</code>	Alleen voor ingelogde gebruikers
<code>auth.uid()</code>	UUID van de ingelogde user (in SQL policies)

**Dit document bevat de kernstof van Les 10. Gebruik het als naslagwerk bij het maken van het huiswerk en bij je eindopdracht. Vragen? Stel ze via het les-kanaal.**