TIPPR

Supabase Auth Setup Guide

Version 1.0 | 2025-12-14

Översikt

Denna guide visar dig exakt hur du sätter upp Supabase Auth för Tippr-projektet. Vi använder en hybrid approach där Supabase hanterar autentisering (JWT tokens, email verification, social logins) medan din .NET backend hanterar all business logic.

Fördelar med Supabase Auth:

- **Gratis:** Email/password auth, JWT tokens, email verification
- **Social Logins:** Google, Facebook, GitHub (gratis!)
- **JWT Tokens:** Automatisk generering och refresh
- **Email Templates:** Anpassningsbara verifieringsmails
- **Security:** Rate limiting, password hashing, session management
- **Row Level Security (RLS):** Om du vill använda det

Steg 1: Skapa Supabase Projekt

1.1 Registrera dig på Supabase

- Gå till https://supabase.com
- Klicka på 'Start your project'
- Logga in med GitHub (rekommenderas)

1.2 Skapa nytt projekt

- Klicka på 'New Project'
- **Project Name:** tippr-production (eller tippr-dev för utveckling)
- **Database Password:** Generera ett starkt lösenord (spara det säkert!)
- **Region:** Välj North Europe (Stockholm) för bästa latency från Sverige
- Klicka 'Create new project' (tar ~2 minuter)

1.3 Spara viktiga credentials

När projektet är klart, gå till Settings → API:

- **Project URL:** https://xxxxx.supabase.co
- **anon/public key:** Används i frontend
- **service_role key:** Används i backend (HEMLIG!)

Gå till Settings → Database för connection string:

- **Connection String:** Används för EF Core

Steg 2: Konfigurera Authentication

2.1 Email Auth Settings

Gå till Authentication → Settings:

Setting	Rekommenderat Värde	Beskrivning
Site URL	http://localhost:5173 (dev) https://tippr.app (prod)	Din frontend URL
Redirect URLs	http://localhost:5173/** https://tippr.app/**	Tillåtna redirect URLs
Email Confirm	Enabled	Kräv email-verifiering
Secure Password	Enabled	Starkt lösenordskrav
Min Password Length	8	Minst 8 tecken

2.2 Email Templates (valfritt)

Gå till Authentication → Email Templates för att anpassa:

- **Confirm signup:** Email-verifiering
- **Magic Link:** Lösenordsfri inloggning (om du vill ha det)
- **Change Email Address:** När användare byter email
- **Reset Password:** Återställ lösenord

2.3 Social Logins (Google, Facebook)

Gå till Authentication → Providers:

Google OAuth:

1. Gå till Google Cloud Console (console.cloud.google.com)
2. Skapa nytt projekt → API & Services → Credentials
3. Skapa OAuth 2.0 Client ID
4. **Authorized redirect URIs:** https://xxxxx.supabase.co/auth/v1/callback
5. Kopiera Client ID och Client Secret till Supabase

Facebook OAuth:

1. Gå till Facebook Developers (developers.facebook.com)
2. Skapa ny app → 'Consumer' typ
3. Lägg till 'Facebook Login' product
4. **Valid OAuth Redirect URIs:** https://xxxxx.supabase.co/auth/v1/callback
5. Kopiera App ID och App Secret till Supabase

Steg 3: Backend Integration (.NET)

3.1 Installera NuGet Packages

```
dotnet add package Supabase
```

```
dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer
```

```
dotnet add package Npgsql.EntityFrameworkCore.PostgreSQL
```

3.2 Konfigurera appsettings.json

```
{ "Supabase": { "Url": "https://xxxxx.supabase.co", "Key": "your-anon-key", "ServiceRoleKey":
"your-service-role-key" }, "ConnectionStrings": { "DefaultConnection":
"Host=db.xxxxx.supabase.co;Database=postgres;Username=postgres;Password=xxx;SSL Mode=Require"
}, "JwtSettings": { "Secret": "your-jwt-secret-from-supabase", "Issuer":
"https://xxxxx.supabase.co/auth/v1", "Audience": "authenticated" } }
```

3.3 Konfigurera JWT Authentication i Program.cs

```
using Microsoft.AspNetCore.Authentication.JwtBearer; using Microsoft.IdentityModel.Tokens; using
System.Text; var builder = WebApplication.CreateBuilder(args); // JWT Authentication var
jwtSecret = builder.Configuration["JwtSettings:Secret"]; var jwtIssuer =
builder.Configuration["JwtSettings:Issuer"];
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
.AddJwtBearer(options => { options.TokenValidationParameters = new TokenValidationParameters {
ValidateIssuer = true, ValidateAudience = true, ValidateLifetime = true,
ValidateIssuerSigningKey = true, ValidIssuer = jwtIssuer, ValidAudience = "authenticated",
IssuerSigningKey = new SymmetricSecurityKey( Encoding.UTF8.GetBytes(jwtSecret)) }; });
builder.Services.AddAuthorization();
```

3.4 Skapa User Entity med Supabase Auth Link

```
public class User { public Guid Id { get; set; } // Same as auth.users.id from Supabase public string Email { get; set; } = string.Empty; public string Username { get; set; } = string.Empty; public string? DisplayName { get; set; } public string? AvatarUrl { get; set; } public DateTime CreatedAt { get; set; } public DateTime UpdatedAt { get; set; } public DateTime? LastLoginAt { get; set; } // Navigation properties public ICollection LeagueMembers { get; set; } = []; public ICollection Predictions { get; set; } = []; }
```

3.5 Skapa Auth Service för att hantera sync med Supabase

```
public interface IAuthService { Task GetOrCreateUserAsync(Guid authUserId, string email); Task UpdateLastLoginAsync(Guid userId); } public class AuthService : IAuthService { private readonly ApplicationDbContext _context; public AuthService(ApplicationDbContext context) { _context = context; } public async Task GetOrCreateUserAsync(Guid authUserId, string email) { var user = await _context.Users .FirstOrDefaultAsync(u => u.Id == authUserId); if (user == null) { // Create new user in our database user = new User { Id = authUserId, Email = email, Username = email.Split("@")[0], // Default username CreatedAt = DateTime.UtcNow, UpdatedAt = DateTime.UtcNow }; _context.Users.Add(user); await _context.SaveChangesAsync(); } return user; } public async Task UpdateLastLoginAsync(Guid userId) { var user = await _context.Users.FindAsync(userId); if (user != null) { user.LastLoginAt = DateTime.UtcNow; await _context.SaveChangesAsync(); } } }
```

3.6 Skapa Auth Controller/Endpoints

```
[ApiController] [Route("api/[controller]")] public class AuthController : ControllerBase { private readonly IAuthService _authService; public AuthController(IAuthService authService) { _authService = authService; } [HttpPost("sync")] [Authorize] // Requires valid JWT token from Supabase public async Task SyncUser() { // Get user ID from JWT token var userIdClaim = User.FindFirst("sub")?.Value; var emailClaim = User.FindFirst("email")?.Value; if (string.IsNullOrEmpty(userIdClaim) || string.IsNullOrEmpty(emailClaim)) return Unauthorized(); var userId = Guid.Parse(userIdClaim); var user = await _authService.GetOrCreateUserAsync(userId, emailClaim); await _authService.UpdateLastLoginAsync(userId); return Ok(new { user }); } [HttpGet("me")] [Authorize] public async Task GetCurrentUser() { var userId = Guid.Parse(User.FindFirst("sub")!.Value); var user = await _context.Users.FindAsync(userId); return Ok(user); } }
```

Steg 4: Frontend Integration (React)

4.1 Installera Supabase Client

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

4.2 Skapa Supabase Client

```
// src/lib/supabase.ts import { createClient } from '@supabase/supabase-js' const supabaseUrl =
import.meta.env.VITE_SUPABASE_URL const supabaseAnonKey = import.meta.env.VITE_SUPABASE_ANON_KEY
export const supabase = createClient(supabaseUrl, supabaseAnonKey)
```

```
// .env
```

```
VITE_SUPABASE_URL=https://xxxxx.supabase.co VITE_SUPABASE_ANON_KEY=your-anon-key
VITE_API_URL=http://localhost:5000/api
```

4.3 Skapa Auth Context

```
// src/contexts/AuthContext.tsx import { createContext, useContext, useEffect, useState } from
'react' import { supabase } from '@/lib/supabase' import type { User, Session } from
'@supabase/supabase-js' interface AuthContextType { user: User | null session: Session | null
signUp: (email: string, password: string) => Promise signIn: (email: string, password: string)
=> Promise signOut: () => Promise signInWithGoogle: () => Promise } const AuthContext =
createContext({} as AuthContextType) export function AuthProvider({ children }: { children:
React.ReactNode }) { const [user, setUser] = useState(null) const [session, setSession] =
useState(null) useEffect(() => { // Get initial session supabase.auth.getSession().then(({ data:
{ session } }) => { setSession(session) setUser(session?.user ?? null) }) // Listen for auth
changes const { data: { subscription } } = supabase.auth.onAuthStateChange( (_event, session) =>
{ setSession(session) setUser(session?.user ?? null) } ) return () => subscription.unsubscribe()
}, []) const signUp = async (email: string, password: string) => { const { error } = await
supabase.auth.signUp({ email, password }) if (error) throw error } const signIn = async (email:
string, password: string) => { const { error } = await supabase.auth.signInWithPassword({ email,
password }) if (error) throw error } const signOut = async () => { const { error } = await
supabase.auth.signOut() if (error) throw error } const signInWithGoogle = async () => { const {
error } = await supabase.auth.signInWithOAuth({ provider: 'google', options: { redirectTo:
window.location.origin } }) if (error) throw error } return ( {children} ) } export const
useAuth = () => useContext(AuthContext)
```

4.4 Skapa Login/Signup Components

```
// src/components/LoginForm.tsx import { useState } from 'react' import { useAuth } from
'@/contexts/AuthContext' export function LoginForm() { const [email, setEmail] = useState('')
const [password, setPassword] = useState('') const { signIn, signInWithGoogle } = useAuth()
const handleSubmit = async (e: React.FormEvent) => { e.preventDefault() try { await
signIn(email, password) // Redirect or show success } catch (error) { console.error('Login
error:', error) } } return ( setEmail(e.target.value)} placeholder="Email" required />
setPassword(e.target.value)} placeholder="Password" required /> Sign In Sign in with Google ) }
```

4.5 Skicka JWT till Backend

```
// src/lib/api.ts import { supabase } from './supabase' export async function
apiClient(endpoint: string, options: RequestInit = {}) { const { data: { session } } = await
supabase.auth.getSession() const headers = { 'Content-Type': 'application/json',
...options.headers, } if (session?.access_token) { headers['Authorization'] = `Bearer
${session.access_token}` } const response = await fetch(
`${import.meta.env.VITE_API_URL}${endpoint}`, { ...options, headers } ) if (!response.ok) throw
new Error('API request failed') return response.json() } // Usage: const user = await
apiClient('/auth/me') const leagues = await apiClient('/leagues')
```

Steg 5: Complete User Flow

Steg	Vad händer	Var
1	Användare registrerar sig med email/password	Frontend → Supabase Auth
2	Supabase skapar user i auth.users tabell	Supabase
3	Supabase skickar verifieringsmail	Email
4	Användare klickar på länk och verifierar	Email → Supabase
5	Användare loggar in	Frontend → Supabase Auth
6	Supabase returnerar JWT access_token + refresh_token	Supabase → Frontend
7	Frontend skickar JWT till backend /auth/sync	Frontend → .NET Backend
8	Backend validerar JWT och skapar User i public.users	.NET Backend
9	Backend returnerar user data	.NET Backend → Frontend
10	Användare är inloggad och kan använda appen!	Frontend

Steg 6: Testa Authentication

6.1 Registrera en testanvändare

1. Starta din frontend (npm run dev)
2. Gå till signup-sidan
3. Registrera med email + password
4. Kolla din email för verifieringslänk
5. Klicka på länken för att verifiera

6.2 Verifiera i Supabase Dashboard

Gå till Authentication → Users i Supabase:

• Du ska se din nya användare i listan

• **Email Confirmed At:** ska vara satt

• Kopiera UUID (detta är användarens ID)

6.3 Verifiera i din databas

Gå till Table Editor → users:

• Efter första inloggning ska användaren finnas här

• ID ska matcha UUID från auth.users

6.4 Testa Protected Endpoint

```
curl -H "Authorization: Bearer YOUR_JWT_TOKEN" \
http://localhost:5000/api/auth/me
```

Du ska få tillbaka användardata!

Best Practices & Tips

Security

- **ALDRIG** committa service_role_key till Git
- Använd environment variables för alla secrets
- Använd HTTPS i production
- Sätt upp CORS korrekt i backend
- Validera JWT tokens på varje request i backend

Token Management

- Supabase hanterar refresh tokens automatiskt
- Access tokens är giltiga i 1 timme (default)
- Refresh tokens är giltiga i 30 dagar
- supabase.auth.getSession() hanterar refresh automatiskt

User Sync

- Skapa User i public.users vid första inloggning
- Använd Supabase auth.users.id som primary key
- Synka email och metadata från Supabase
- Låt användare uppdatera DisplayName, Bio, Avatar i din app

Email Verification

- **Development:** Kolla Supabase dashboard för magic links
- **Production:** Konfigurera SMTP i Supabase för riktiga emails
- Överväg att inaktivera email confirmation under development

Troubleshooting

Problem	Lösning
JWT validation fails	Kontrollera att JWT Secret i appsettings.json matchar Supabase
CORS errors	Lägg till frontend URL i CORS policy i Program.cs
User not created in public.users	Kolla att /auth/sync anropas efter login
Email not sending	Kolla Supabase → Settings → Auth → SMTP settings
Social login redirect error	Verifiera Redirect URLs i både Supabase och OAuth provider
Token expired	Använd supabase.auth.getSession() som automatiskt refreshar

Sammanfattning

Nu har du en komplett Supabase Auth-setup med hybrid approach där Supabase hanterar autentisering och din .NET backend hanterar business logic. Detta ger dig det bästa av båda världarna: gratis, säker auth från Supabase och full kontroll över din applikationslogik.

Nästa steg:

1. Implementera protected routes i React
2. Lägg till [Authorize] attribut på backend endpoints
3. Implementera profile-sida där användare kan uppdatera sina uppgifter
4. Testa social logins (Google, Facebook)
5. Konfigurera production email-templates i Supabase