Notes: disable emails if you don`t want users to confirm as it can be a problem

Here's a clear summary of the steps and commands we used to successfully install Supabase:

1. **Cleaned up the environment** (to resolve any previous installation issues):

bash

CopyInsert

npm cache clean --force

rm -r -Force node\_modules, package-lock.json

1. **Installed the core Supabase client** with legacy peer deps flag to bypass version conflicts:

bash

CopyInsert in Terminal

npm install @supabase/supabase-js --legacy-peer-deps

1. **Installed the SSR package** (recommended for Next.js 13+):

bash

CopyInsert in Terminal

npm install @supabase/ssr @supabase/auth-helpers-nextjs --legacy-peer-deps

1. **Removed the deprecated auth helpers** (as recommended by the warning):

bash

CopyInsert in Terminal

npm uninstall @supabase/auth-helpers-nextjs --legacy-peer-deps

1. **Verified the installation**:

bash

CopyInsert in Terminal

npm list @supabase/supabase-js @supabase/ssr

The key was using the

--legacy-peer-deps

 flag to bypass the version conflicts with

date-fns

 and other dependencies. This flag tells npm to ignore peer dependency conflicts and proceed with the installation.

Your project now has:

* @supabase/supabase-js

 (v2.50.0)

* @supabase/ssr

 (v0.6.1)

Video: <https://www.youtube.com/watch?v=iXz82niZ3OA&list=PL8HkCX2C5h0X9ZFgSSUhzTI5Nh8-Olijv>

# Supabase Project URL

NEXT\_PUBLIC\_SUPABASE\_URL=https://ugbowxhprswswyaqxjkw.supabase.co

# Supabase Anon Key (public)

NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InVnYm93eGhwcnN3c3d5YXF4amt3Iiwicm9sZSI6ImFub24iLCJpYXQiOjE3NDkzMDI5OTgsImV4cCI6MjA2NDg3ODk5OH0.mNZjQhyBZTZKHADjU4sRXYSSw87SVH93Ec7h74vfntg

# Supabase Service Role Key (keep this secure!)

SUPABASE\_SERVICE\_ROLE\_KEY=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InVnYm93eGhwcnN3c3d5YXF4amt3Iiwicm9sZSI6InNlcnZpY2Vfcm9sZSIsImlhdCI6MTc0OTMwMjk5OCwiZXhwIjoyMDY0ODc4OTk4fQ.vi-YiHjgfEt2Rrere1dfFqlEDsJTecxDWUFFNQzLwtg

npm install @hookform/resolvers@^3.3.4 react-hook-form@^7.53.0 zod@^3.23.8 sonner@^1.4.2 @supabase/supabase-js@^2.39.7 @supabase/ssr@^0.1.0 --legacy-peer-deps

Let's restart the development server to apply all the changes we've made. First, I'll stop any running instances:

Ran terminal command

…\tpp-system > taskkill /F /IM node.exe

**# Supabase Authentication Setup Guide**

This document outlines the steps to set up Supabase authentication with a Next.js application, including database schema, environment variables, and configuration.

**## Prerequisites**

1. Node.js (v16 or later)

2. npm or yarn

3. Supabase account (https://supabase.com/)

4. Next.js project

**## Setup Steps**

**### 1. Install Dependencies**

```bash

# Core dependencies with specific versions and legacy peer deps

npm install @hookform/resolvers@^3.3.4 react-hook-form@^7.53.0 zod@^3.23.8 sonner@^1.4.2 @supabase/supabase-js@^2.39.7 @supabase/ssr@^0.1.0 --legacy-peer-deps

# Additional UI components (if using shadcn/ui)

npm install @radix-ui/react-dropdown-menu @radix-ui/react-slot class-variance-authority clsx tailwind-merge lucide-react --legacy-peer-deps

```

**\*\*Important\*\***: The `--legacy-peer-deps` flag is necessary to avoid dependency conflicts with React 18+.

**### 2. Environment Variables**

Create a `.env.local` file in your project root with the following variables:

```env

# Supabase

NEXT\_PUBLIC\_SUPABASE\_URL=your\_supabase\_project\_url

NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=your\_supabase\_anon\_key

SUPABASE\_SERVICE\_ROLE\_KEY=your\_supabase\_service\_role\_key

```

**### 3. Database Setup**

Run the following SQL in your Supabase SQL Editor to set up the database schema:

```sql

-- Create profiles table

create table public.profiles (

  id uuid references auth.users on delete cascade not null primary key,

  email text not null,

  first\_name text not null,

  last\_name text not null,

  role text not null default 'student' check (role in ('student', 'coordinator', 'admin')),

  created\_at timestamp with time zone default timezone('utc'::text, now()) not null,

  updated\_at timestamp with time zone default timezone('utc'::text, now()) not null

);

-- Create indexes

create index profiles\_email\_idx on public.profiles (email);

create index profiles\_role\_idx on public.profiles (role);

-- Update timestamp function

create or replace function public.update\_updated\_at\_column()

returns trigger

language plpgsql

as $$

begin

  new.updated\_at = now();

  return new;

end;

$$;

-- Create update trigger

create trigger update\_profiles\_updated\_at

  before update on public.profiles

  for each row execute procedure public.update\_updated\_at\_column();

-- Handle new user function

create or replace function public.handle\_new\_user()

returns trigger

language plpgsql

security definer

set search\_path = public

as $$

begin

  insert into public.profiles (id, email, first\_name, last\_name, role)

  values (

    new.id,

    new.email,

    new.raw\_user\_meta\_data->>'firstName',

    new.raw\_user\_meta\_data->>'lastName',

    new.raw\_user\_meta\_data->>'userType'

  );

  return new;

end;

$$;

-- Create trigger for new users

create trigger on\_auth\_user\_created

  after insert on auth.users

  for each row execute procedure public.handle\_new\_user();

-- Enable RLS

alter table public.profiles enable row level security;

-- RLS Policies

create policy "Allow public read access"

  on public.profiles

  for select

  using (true);

create policy "Allow insert for service role"

  on public.profiles

  for insert

  with check (true);

create policy "Allow update for own profile"

  on public.profiles

  for update

  using (auth.uid() = id);

```

**### 4. Supabase Client Setup**

Create `lib/supabase.ts`:

```typescript

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

import { Database } from '@/types/supabase';

// Client-side client

export const supabase = createClient<Database>(

  process.env.NEXT\_PUBLIC\_SUPABASE\_URL!,

  process.env.NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY!

);

// Server-side client

export const supabaseAdmin = createClient<Database>(

  process.env.NEXT\_PUBLIC\_SUPABASE\_URL!,

  process.env.SUPABASE\_SERVICE\_ROLE\_KEY!,

  {

    auth: {

      autoRefreshToken: false,

      persistSession: false

    }

  }

);

// Helper functions

export const getSession = async () => {

  const { data: { session } } = await supabase.auth.getSession();

  return session;

};

export const getUser = async () => {

  const { data: { user } } = await supabase.auth.getUser();

  return user;

};

```

**### 5. Type Definitions**

Create or update `types/supabase.ts`:

```typescript

type Json =

  | string

  | number

  | boolean

  | null

  | { [key: string]: Json | undefined }

  | Json[]

export interface Database {

  public: {

    Tables: {

      profiles: {

        Row: {

          id: string;

          email: string;

          first\_name: string;

          last\_name: string;

          role: 'student' | 'coordinator' | 'admin';

          created\_at: string;

          updated\_at: string;

        };

        Insert: {

          id: string;

          email: string;

          first\_name: string;

          last\_name: string;

          role: 'student' | 'coordinator' | 'admin';

          created\_at?: string;

          updated\_at?: string;

        };

        Update: {

          id?: string;

          email?: string;

          first\_name?: string;

          last\_name?: string;

          role?: 'student' | 'coordinator' | 'admin';

          updated\_at?: string;

        };

      };

    };

    Enums: {

      user\_role: 'student' | 'coordinator' | 'admin';

    };

  };

}

```

**## Troubleshooting**

**### Common Issues**

1. **\*\*RLS Policy Errors\*\***

   - Ensure all necessary RLS policies are in place

   - Check that the service role key is being used for server-side operations

2. **\*\*Trigger Not Firing\*\***

   - Verify the trigger is correctly set up on the `auth.users` table

   - Check the function has `security definer`

3. **\*\*Type Errors\*\***

   - Ensure your TypeScript types match your database schema

   - Restart your TypeScript server after making changes to type definitions

**## Useful SQL Commands**

```sql

-- View all profiles

select \* from public.profiles;

-- View all users

select \* from auth.users;

-- View RLS policies

select \* from pg\_policies where tablename = 'profiles';

-- View triggers

select trigger\_name, event\_manipulation, event\_object\_table, action\_statement

from information\_schema.triggers;

```

**## Security Notes**

1. Never expose the `SUPABASE\_SERVICE\_ROLE\_KEY` in client-side code

2. Always use environment variables for sensitive information

3. Regularly audit your RLS policies

4. Keep your Supabase dependencies up to date

**## Next Steps**

1. Implement email confirmation

2. Add password reset functionality

3. Set up protected routes

4. Add role-based access control (RBAC)

5. Implement logging and monitoring

**# Talent Pipeline Programme (TPP) System**

A Next.js web application for managing the Talent Pipeline Programme, featuring role-based access control for students and coordinators.

**## Features**

- **\*\*Authentication\*\***: Secure login/signup with Supabase Auth

- **\*\*Role-based Access Control\*\***:

  - Students: View and manage their profile and academic performance

  - Coordinators: Manage student data and track academic progress

  - Strict role-based route protection

- **\*\*Modern UI\*\***: Built with Tailwind CSS and shadcn/ui components

- **\*\*Form Handling\*\***: Robust form validation with React Hook Form and Zod

- **\*\*Loading States\*\***: Beautiful loading animations with the TPP logo

- **\*\*Responsive Design\*\***: Works on all device sizes

- **\*\*Error Handling\*\***: Graceful error states and access denied pages

**## Tech Stack**

- **\*\*Frontend\*\***: Next.js 14 (App Router), TypeScript, React 18

- **\*\*Styling\*\***: Tailwind CSS with shadcn/ui components

- **\*\*State Management\*\***: React Context + Hooks

- **\*\*Database & Auth\*\***: Supabase (PostgreSQL)

- **\*\*Form Handling\*\***: React Hook Form with Zod validation

- **\*\*Icons\*\***: Lucide React

- **\*\*Build Tool\*\***: Vite

**## Prerequisites**

- Node.js 18 or later

- npm or pnpm

- Supabase account (https://supabase.com/)

- Git for version control

**## Getting Started**

1. **\*\*Clone the repository\*\***

   ```bash

   git clone [repository-url]

   cd tpp-system

   ```

2. **\*\*Install dependencies\*\***

   ```bash

   npm install

   # or

   pnpm install

   ```

3. **\*\*Set up environment variables\*\***

   Create a `.env.local` file in the root directory with your Supabase credentials:

   ```env

   NEXT\_PUBLIC\_SUPABASE\_URL=your\_supabase\_project\_url

   NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=your\_supabase\_anon\_key

   SUPABASE\_SERVICE\_ROLE\_KEY=your\_supabase\_service\_role\_key

   ```

4. **\*\*Run database migrations\*\***

   Make sure to set up the database schema as described in DATABASE.md

5. **\*\*Start the development server\*\***

   ```bash

   npm run dev

   # or

   pnpm dev

   ```

**## Testing Authentication**

1. **\*\*Create a test user\*\***

   - Visit `/access-portal`

   - Sign up with an email and password

   - The system will automatically create a profile with the 'student' role

2. **\*\*Access different dashboards\*\***

   - Students will be redirected to `/student`

   - Coordinators will be redirected to `/coordinator`

   - Accessing unauthorized routes will show an access denied page

**## Development Workflow**

- **\*\*Branching\*\***: Create feature branches from `main`

- **\*\*Commits\*\***: Use conventional commit messages

- **\*\*Pull Requests\*\***: Required for all changes

- **\*\*Testing\*\***: Test all authentication flows after changes

   ```

4. **\*\*Set up the database\*\***

   - Create a new project in Supabase

   - Run the SQL from `AUTH\_SETUP.md` in the Supabase SQL Editor

5. **\*\*Run the development server\*\***

   ```bash

   npm run dev

   # or

   pnpm dev

   ```

6. **\*\*Open [http://localhost:3000](**http://localhost:3000**) in your browser\*\***

**## Project Structure**

```

├── app/                    # App router pages and layouts

├── components/             # Reusable UI components

├── hooks/                  # Custom React hooks

├── lib/                    # Utility functions and configurations

├── public/                 # Static assets

├── styles/                 # Global styles

└── types/                  # TypeScript type definitions

```

**## Available Scripts**

- `npm run dev` - Start development server

- `npm run build` - Build for production

- `npm start` - Start production server

- `npm run lint` - Run ESLint

**## Authentication Implementation**

The authentication system is built with the following key components:

1. **\*\*Sign Up Flow\*\***

   - Collects user details (name, email, password, role)

   - Creates user in Supabase Auth

   - Creates corresponding profile in the database

   - Automatically logs in the user

2. **\*\*Sign In Flow\*\***

   - Validates credentials against Supabase Auth

   - Retrieves user profile

   - Stores session in localStorage

   - Redirects based on user role

3. **\*\*Session Management\*\***

   - JWT-based authentication

   - Persistent sessions

   - Automatic token refresh

**## Security Notes**

⚠️ **\*\*Important\*\***: This implementation is for development purposes. For production, consider adding:

- Email confirmation

- Password strength requirements

- Rate limiting

- Additional security headers

- Proper RLS policies

**## Deployment**

1. **\*\*Build the application\*\***

   ```bash

   npm run build

   ```

2. **\*\*Start the production server\*\***

   ```bash

   npm start

   ```

For production deployment, consider using:

- Vercel (recommended for Next.js)

- Netlify

- AWS Amplify

- Self-hosted on a VPS with Nginx/PM2

**## Contributing**

1. Fork the repository

2. Create your feature branch (`git checkout -b feature/AmazingFeature`)

3. Commit your changes (`git commit -m 'Add some AmazingFeature'`)

4. Push to the branch (`git push origin feature/AmazingFeature`)

5. Open a Pull Request

**## License**

This project is licensed under the MIT License - see the [LICENSE](LICENSE) file for details.

**## Support**

For support, please open an issue in the repository or contact the development team.

**# Authentication Setup Guide**

This document provides comprehensive instructions for setting up and configuring authentication in the TPP System using Supabase. It covers database setup, environment configuration, and implementation details.

**## Table of Contents**

- [Prerequisites](#prerequisites)

- [Setup Steps](#setup-steps)

- [Environment Variables](#environment-variables)

- [Database Setup](#database-setup)

- [Authentication Flows](#authentication-flows)

- [Role-Based Access Control](#role-based-access-control)

- [Testing Authentication](#testing-authentication)

- [Troubleshooting](#troubleshooting)

**## Prerequisites**

1. Node.js (v18 or later)

2. npm, pnpm, or yarn

3. Supabase account ([sign up here](https://supabase.com/))

4. Git for version control

5. Basic understanding of Next.js and React

**## Setup Steps**

**### 1. Project Initialization**

Clone the repository and install dependencies:

```bash

# Clone the repository

git clone [repository-url]

cd tpp-system

# Install dependencies

npm install

# or

pnpm install

```

**## Setup Steps**

**### 1. Install Dependencies**

```bash

# Core dependencies with specific versions and legacy peer deps

npm install @hookform/resolvers@^3.3.4 react-hook-form@^7.53.0 zod@^3.23.8 sonner@^1.4.2 @supabase/supabase-js@^2.39.7 @supabase/ssr@^0.1.0 --legacy-peer-deps

# Additional UI components (if using shadcn/ui)

npm install @radix-ui/react-dropdown-menu @radix-ui/react-slot class-variance-authority clsx tailwind-merge lucide-react --legacy-peer-deps

```

**\*\*Important\*\***: The `--legacy-peer-deps` flag is necessary to avoid dependency conflicts with React 18+.

**### 2. Environment Setup**

1. Create a `.env.local` file in your project root:

```bash

# Copy example environment file

cp .env.example .env.local

```

2. Update the following variables in `.env.local` with your Supabase credentials:

```env

# Supabase

NEXT\_PUBLIC\_SUPABASE\_URL=your\_supabase\_project\_url

NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY=your\_supabase\_anon\_key

SUPABASE\_SERVICE\_ROLE\_KEY=your\_supabase\_service\_role\_key

# Optional: Set to 'true' in development to enable debug logging

NEXT\_PUBLIC\_DEBUG\_AUTH=false

```

> **\*\*Note\*\***: Never commit `.env.local` to version control. It's already included in `.gitignore`.

**## Database Setup**

**### 1. Create a New Supabase Project**

1. Go to [Supabase Dashboard](https://app.supabase.com/)

2. Click "New Project"

3. Choose your organization and project details

4. Set a secure database password

5. Choose a region closest to your users

6. Click "Create project"

**### 2. Set Up Database Schema**

Run the following SQL in your Supabase SQL Editor:

1. Go to SQL Editor in the Supabase dashboard

2. Click "New Query"

3. Paste the following SQL code

4. Click "Run" or press Cmd+Enter (Mac) / Ctrl+Enter (Windows/Linux)

```sql

-- Enable required extensions

create extension if not exists "uuid-ossp" with schema extensions;

create extension if not exists "pgcrypto" with schema extensions;

-- Create profiles table

create table public.profiles (

  id uuid references auth.users on delete cascade not null primary key,

  email text not null,

  first\_name text,

  last\_name text,

  role text not null default 'student' check (role in ('student', 'coordinator')),

  student\_number text,

  school text,

  grade text,

  created\_at timestamp with time zone default timezone('utc'::text, now()) not null,

  updated\_at timestamp with time zone default timezone('utc'::text, now()) not null,

  constraint unique\_email unique (email)

);

-- Create indexes for better performance

create index if not exists profiles\_email\_idx on public.profiles (email);

create index if not exists profiles\_role\_idx on public.profiles (role);

create index if not exists profiles\_student\_number\_idx on public.profiles (student\_number) where student\_number is not null;

-- Update timestamp function

create or replace function public.update\_updated\_at\_column()

returns trigger

language plpgsql

as $$

begin

  new.updated\_at = now();

  return new;

end;

$$;

-- Create update trigger

create trigger update\_profiles\_updated\_at

  before update on public.profiles

  for each row execute procedure public.update\_updated\_at\_column();

-- Handle new user function

create or replace function public.handle\_new\_user()

returns trigger

language plpgsql

security definer

set search\_path = public

as $$

begin

  insert into public.profiles (

    id,

    email,

    first\_name,

    last\_name,

    role,

    student\_number,

    school,

    grade

  )

  values (

    new.id,

    new.email,

    new.raw\_user\_meta\_data->>'firstName',

    new.raw\_user\_meta\_data->>'lastName',

    coalesce((new.raw\_user\_meta\_data->>'role')::text, 'student'),

    new.raw\_user\_meta\_data->>'studentNumber',

    new.raw\_user\_meta\_data->>'school',

    new.raw\_user\_meta\_data->>'grade'

  )

  on conflict (id) do update set

    email = excluded.email,

    first\_name = excluded.first\_name,

    last\_name = excluded.last\_name,

    updated\_at = now();

  return new;

end;

$$;

-- Create trigger for new users

create or replace trigger on\_auth\_user\_created

  after insert on auth.users

  for each row execute procedure public.handle\_new\_user();

-- Enable RLS

alter table public.profiles enable row level security;

-- Allow users to read their own profile

create policy "Users can view their own profile"

on profiles for select

  using (auth.uid() = id);

-- Allow users to update their own profile

create policy "Users can update their own profile"

on profiles for update

  using (auth.uid() = id);

-- Allow coordinators to view all profiles

create policy "Coordinators can view all profiles"

on profiles for select

  using (auth.role() = 'authenticated' and

         exists (select 1 from profiles where id = auth.uid() and role = 'coordinator'));

-- Allow authenticated users to see basic profile info

create policy "Public profiles are viewable by authenticated users"

on profiles for select

  to authenticated

  using (true);

-- RLS Policies

create policy "Allow public read access"

  on public.profiles

  for select

  using (true);

create policy "Allow insert for service role"

  on public.profiles

  for insert

  with check (true);

create policy "Allow update for own profile"

  on public.profiles

  for update

  using (auth.uid() = id);

```

**### 4. Supabase Client Setup**

Create `lib/supabase.ts`:

```typescript

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

import { Database } from '@/types/supabase';

// Client-side client

export const supabase = createClient<Database>(

  process.env.NEXT\_PUBLIC\_SUPABASE\_URL!,

  process.env.NEXT\_PUBLIC\_SUPABASE\_ANON\_KEY!

);

// Server-side client

export const supabaseAdmin = createClient<Database>(

  process.env.NEXT\_PUBLIC\_SUPABASE\_URL!,

  process.env.SUPABASE\_SERVICE\_ROLE\_KEY!,

  {

    auth: {

      autoRefreshToken: false,

      persistSession: false

    }

  }

);

// Helper functions

export const getSession = async () => {

  const { data: { session } } = await supabase.auth.getSession();

  return session;

};

export const getUser = async () => {

  const { data: { user } } = await supabase.auth.getUser();

  return user;

};

```

**### 5. Type Definitions**

Create or update `types/supabase.ts`:

```typescript

type Json =

  | string

  | number

  | boolean

  | null

  | { [key: string]: Json | undefined }

  | Json[]

export interface Database {

  public: {

    Tables: {

      profiles: {

        Row: {

          id: string;

          email: string;

          first\_name: string;

          last\_name: string;

          role: 'student' | 'coordinator' | 'admin';

          created\_at: string;

          updated\_at: string;

        };

        Insert: {

          id: string;

          email: string;

          first\_name: string;

          last\_name: string;

          role: 'student' | 'coordinator' | 'admin';

          created\_at?: string;

          updated\_at?: string;

        };

        Update: {

          id?: string;

          email?: string;

          first\_name?: string;

          last\_name?: string;

          role?: 'student' | 'coordinator' | 'admin';

          updated\_at?: string;

        };

      };

    };

    Enums: {

      user\_role: 'student' | 'coordinator' | 'admin';

    };

  };

}

```

**## Troubleshooting**

**### Common Issues**

1. **\*\*RLS Policy Errors\*\***

   - Ensure all necessary RLS policies are in place

   - Check that the service role key is being used for server-side operations

2. **\*\*Trigger Not Firing\*\***

   - Verify the trigger is correctly set up on the `auth.users` table

   - Check the function has `security definer`

3. **\*\*Type Errors\*\***

   - Ensure your TypeScript types match your database schema

   - Restart your TypeScript server after making changes to type definitions

**## Useful SQL Commands**

```sql

-- View all profiles

select \* from public.profiles;

-- View all users

select \* from auth.users;

-- View RLS policies

select \* from pg\_policies where tablename = 'profiles';

-- View triggers

select trigger\_name, event\_manipulation, event\_object\_table, action\_statement

from information\_schema.triggers;

```

**## Security Notes**

1. Never expose the `SUPABASE\_SERVICE\_ROLE\_KEY` in client-side code

2. Always use environment variables for sensitive information

3. Regularly audit your RLS policies

4. Keep your Supabase dependencies up to date

**## Next Steps**

1. Implement email confirmation

2. Add password reset functionality

**## Authentication Flows**

**### 1. Sign Up**

1. User visits `/access-portal`

2. Clicks "Create an account"

3. Fills in the registration form

4. System creates a new user in Supabase Auth

5. The `on\_auth\_user\_created` trigger creates a profile

6. User receives a confirmation email (if email confirmation is enabled)

7. After email confirmation, user is redirected to the appropriate dashboard

**### 2. Sign In**

1. User visits `/access-portal`

2. Enters email and password

3. System authenticates with Supabase

4. On success, fetches the user's profile

5. Redirects to the appropriate dashboard based on role

**### 3. Sign Out**

1. User clicks "Sign Out" in the dashboard

2. System calls `supabase.auth.signOut()`

3. Redirects to the access portal

**## Role-Based Access Control**

The system uses two main roles:

**### 1. Student**

- Can view and update their own profile

- Can view their academic performance

- Can submit assignments

- Cannot access coordinator features

**### 2. Coordinator**

- Can view all student profiles

- Can manage student data

- Can generate reports

- Has access to admin features

**## Testing Authentication**

**### 1. Create a Test User**

```sql

-- Create a test student

insert into auth.users (id, email, encrypted\_password, email\_confirmed\_at, raw\_user\_meta\_data)

values (

  '00000000-0000-0000-0000-000000000001',

  'student@example.com',

  crypt('password123', gen\_salt('bf')),

  now(),

  '{"firstName":"Test","lastName":"Student","role":"student"}'

);

-- Create a test coordinator

insert into auth.users (id, email, encrypted\_password, email\_confirmed\_at, raw\_user\_meta\_data)

values (

  '00000000-0000-0000-0000-000000000002',

  'coordinator@example.com',

  crypt('password123', gen\_salt('bf')),

  now(),

  '{"firstName":"Test","lastName":"Coordinator","role":"coordinator"}'

);

```

**### 2. Test Login Credentials**

**\*\*Student Account\*\***

- Email: student@example.com

- Password: password123

**\*\*Coordinator Account\*\***

- Email: coordinator@example.com

- Password: password123

**## Troubleshooting**

**### Common Issues**

1. **\*\*Authentication Fails\*\***

   - Check if the user exists in the `auth.users` table

   - Verify the password hash is correct

   - Ensure email confirmation is not required or the email is confirmed

2. **\*\*Profile Not Created\*\***

   - Check the `auth.users` table for the user

   - Look for errors in the Supabase logs

   - Verify the trigger is properly set up

3. **\*\*Role-Based Access Issues\*\***

   - Check the `profiles` table for the user's role

   - Verify RLS policies are correctly set up

   - Check the JWT claims with `select auth.jwt();`

4. **\*\*CORS Issues\*\***

   - Ensure your frontend URL is in the Supabase CORS settings

   - Check the browser console for CORS errors

**### Debugging**

1. **\*\*Check Supabase Logs\*\***

   - Go to the Supabase dashboard

   - Navigate to Logs > Edge Functions

   - Look for any error messages

2. **\*\*Enable Debug Logging\*\***

   Set `NEXT\_PUBLIC\_DEBUG\_AUTH=true` in your `.env.local` to enable debug logging.

3. **\*\*Inspect JWT\*\***

   ```typescript

   const { data: { session } } = await supabase.auth.getSession();

   console.log('JWT:', session?.access\_token);

   // Decode at https://jwt.io/

   ```

**## Security Best Practices**

1. **\*\*Environment Variables\*\***

   - Never commit `.env.local` to version control

   - Use different credentials for development and production

   - Rotate API keys regularly

2. **\*\*Password Policies\*\***

   - Enforce strong passwords

   - Implement rate limiting

   - Consider adding MFA for coordinators

3. **\*\*Session Management\*\***

   - Use secure, HTTP-only cookies

   - Set appropriate session expiry

   - Implement proper sign-out functionality

4. **\*\*Regular Audits\*\***

   - Review access logs

   - Monitor for suspicious activity

   - Keep dependencies updated

**## Deployment**

1. **\*\*Production Environment Variables\*\***

   - Set up environment variables in your hosting provider

   - Never commit production credentials to version control

2. **\*\*CORS Configuration\*\***

   - Add your production domain to Supabase CORS settings

   - Remove development URLs in production

3. **\*\*Backup Strategy\*\***

   - Set up regular database backups

   - Test restoration procedures

**## Support**

For additional help, please contact the development team or refer to:

- [Supabase Documentation](https://supabase.com/docs)

- [Next.js Authentication](https://nextjs.org/docs/authentication)

- [Project Wiki](link-to-your-wiki)