Supabase Setup Guide

Table of Contents

Project Setup

1. Create a Supabase Project

1. Go to Supabase Dashboard
2. Click "New Project"
3. Fill in project details:
   - Name: `education-analytics`
   - Database Password: (generate a strong password)
   - Region: (choose closest to your users)
   - Pricing Plan: Free tier

2. Get API Credentials

1. In project dashboard, go to Settings → API
2. Copy these credentials:

   ```
   VITE_SUPABASE_URL=your_project_url
   VITE_SUPABASE_ANON_KEY=your_anon_key
   ```

3. Add them to your `.env` file

Database Schema

1. Create Tables

Navigate to SQL Editor and run the following migrations:

```sql
-- Students table
CREATE TABLE IF NOT EXISTS students (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id uuid REFERENCES auth.users(id),
  first_name text NOT NULL,
  last_name text NOT NULL,
  email text UNIQUE NOT NULL,
```

```sql
  enrollment_date timestamptz DEFAULT now(),
  grade_level integer NOT NULL,
  is_active boolean DEFAULT true,
  created_at timestamptz DEFAULT now(),
  updated_at timestamptz DEFAULT now()
);

-- Courses table
CREATE TABLE IF NOT EXISTS courses (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  title text NOT NULL,
  description text,
  category text NOT NULL,
  difficulty_level text NOT NULL,
  credits integer DEFAULT 1,
  is_active boolean DEFAULT true,
  created_at timestamptz DEFAULT now(),
  updated_at timestamptz DEFAULT now()
);

-- Enrollments table
CREATE TABLE IF NOT EXISTS enrollments (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  student_id uuid REFERENCES students(id) ON DELETE CASCADE,
  course_id uuid REFERENCES courses(id) ON DELETE CASCADE,
  enrollment_date timestamptz DEFAULT now(),
  completion_date timestamptz,
  status text DEFAULT 'enrolled',
  progress integer DEFAULT 0,
  created_at timestamptz DEFAULT now(),
  updated_at timestamptz DEFAULT now()
);

-- Performance Records table
CREATE TABLE IF NOT EXISTS performance_records (
  id uuid PRIMARY KEY DEFAULT gen_random_uuid(),
  student_id uuid REFERENCES students(id) ON DELETE CASCADE,
  course_id uuid REFERENCES courses(id) ON DELETE CASCADE,
  assessment_type text NOT NULL,
  score numeric NOT NULL,
  max_score numeric NOT NULL,
  completed_at timestamptz DEFAULT now(),
  created_at timestamptz DEFAULT now()
);
```

2. Enable Row Level Security

```sql
-- Enable RLS
ALTER TABLE students ENABLE ROW LEVEL SECURITY;
ALTER TABLE courses ENABLE ROW LEVEL SECURITY;
ALTER TABLE enrollments ENABLE ROW LEVEL SECURITY;
ALTER TABLE performance_records ENABLE ROW LEVEL SECURITY;
```

Authentication

1. Configure Auth Settings

1. Go to Authentication → Settings
2. Configure Email Auth:
    - Disable email confirmations
    - Set password strength requirements
    - Configure SMTP (optional for production)

2. Create Auth Policies

```sql
-- Students table policies
CREATE POLICY "Users can view their own student profile"
  ON students
  FOR SELECT
  TO authenticated
  USING (auth.uid() = user_id);

-- Courses table policies
CREATE POLICY "Users can view all courses"
  ON courses
  FOR SELECT
  TO authenticated
  USING (true);

-- Enrollments table policies
CREATE POLICY "Users can view their own enrollments"
  ON enrollments
  FOR SELECT
  TO authenticated
  USING (
    student_id IN (
      SELECT id FROM students WHERE user_id = auth.uid()
    )
  );

-- Performance records policies
CREATE POLICY "Users can view their own performance records"
  ON performance_records
  FOR SELECT
  TO authenticated
  USING (
    student_id IN (
      SELECT id FROM students WHERE user_id = auth.uid()
    )
  );
```

API Integration

1. Install Supabase Client

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

2. Initialize Client

Create `src/lib/supabase.ts`:

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

const supabaseUrl = import.meta.env.VITE_SUPABASE_URL;
const supabaseKey = import.meta.env.VITE_SUPABASE_ANON_KEY;

export const supabase = createClient(supabaseUrl, supabaseKey);
```

3. Generate TypeScript Types

1. Install Supabase CLI:

   ```
   npm install supabase --save-dev
   ```

2. Generate types:

   ```
   npx supabase gen types typescript --project-id your-project-id >
   src/lib/database.types.ts
   ```

Data Seeding

1. Create Sample Data

```
-- Insert sample courses
INSERT INTO courses (title, description, category, difficulty_level, credits)
VALUES
  ('Advanced Mathematics', 'Complex mathematical concepts', 'Mathematics',
'Advanced', 3),
  ('Physics Fundamentals', 'Basic physics principles', 'Science', 'Intermediate',
4),
  ('World Literature', 'Classic literature studies', 'English', 'Intermediate',
3),
  ('Computer Science 101', 'Programming basics', 'Technology', 'Beginner', 4);
```

```sql
-- Insert sample students (after user registration)
INSERT INTO students (user_id, first_name, last_name, email, grade_level)
VALUES
  ('auth-user-id', 'John', 'Doe', 'john.doe@example.com', 11);
```

Testing

1. Test RLS Policies

```sql
-- Test as authenticated user
SELECT * FROM students WHERE user_id = 'auth-user-id';

-- Test as anonymous
SELECT * FROM courses;
```

2. Test Data Access

```typescript
// Example: Fetch student data
const fetchStudentData = async (userId: string) => {
  const { data, error } = await supabase
    .from('students')
    .select('*')
    .eq('user_id', userId)
    .single();

  if (error) console.error('Error:', error.message);
  return data;
};
```

Monitoring

1. Database Health

1. Go to Database → Health
2. Monitor:
   - Connection count
   - Cache hit ratio
   - Disk usage

2. API Usage

1. Go to API → Statistics
2. Monitor:
   - Request volume
   - Error rates

    - Response times

Backup and Recovery

1. Enable Point-in-Time Recovery

1. Go to Database → Backups
2. Enable PITR (Pro plan required)
3. Configure backup retention period

2. Manual Backups

1. Go to Database → Backups
2. Click "Create Backup"
3. Download backup file

Production Considerations

1. **Security**
    - Regularly rotate API keys
    - Use environment-specific API keys
    - Enable SSL enforcement
2. **Performance**
    - Add indexes for frequently queried columns
    - Use connection pooling
    - Implement caching strategies
3. **Monitoring**
    - Set up error alerts
    - Monitor API usage
    - Track performance metrics
4. **Maintenance**
    - Schedule regular backups
    - Plan database upgrades
    - Monitor disk space