# Lex Assist Supabase Schema Design

## Authentication System

### Configuration

* Email/password authentication
* Mobile OTP verification via Twilio integration
* JWT token with custom claims for role-based access
* Session management with configurable expiry
* Password policies (minimum length, complexity requirements)

### User Registration Flow

1. User submits email, mobile, and password
2. Verification email sent to confirm email address
3. Mobile OTP verification for two-factor authentication
4. User account created with default "user" role and "free" tier
5. JWT issued with role and subscription claims

## Database Schema

### Users Table

CREATE TABLE users (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

email TEXT UNIQUE NOT NULL,

phone TEXT UNIQUE,

full\_name TEXT NOT NULL,

role TEXT NOT NULL CHECK (role IN ('super\_admin', 'admin', 'user')),

is\_email\_verified BOOLEAN DEFAULT FALSE,

is\_phone\_verified BOOLEAN DEFAULT FALSE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

last\_login TIMESTAMP WITH TIME ZONE,

account\_status TEXT DEFAULT 'active' CHECK (account\_status IN ('active', 'suspended', 'deactivated'))

);

-- Row Level Security

ALTER TABLE users ENABLE ROW LEVEL SECURITY;

-- Policies

CREATE POLICY "Users can view their own data"

ON users FOR SELECT

USING (auth.uid() = id);

CREATE POLICY "Super admins can view all user data"

ON users FOR SELECT

USING (

EXISTS (

SELECT 1 FROM users

WHERE id = auth.uid() AND role = 'super\_admin'

)

);

CREATE POLICY "Admins can view non-admin user data"

ON users FOR SELECT

USING (

EXISTS (

SELECT 1 FROM users

WHERE id = auth.uid() AND role = 'admin'

) AND role = 'user'

);

CREATE POLICY "Super admins can update any user"

ON users FOR UPDATE

USING (

EXISTS (

SELECT 1 FROM users

WHERE id = auth.uid() AND role = 'super\_admin'

)

);

CREATE POLICY "Users can update their own data"

ON users FOR UPDATE

USING (auth.uid() = id);

### Profiles Table (Extended User Information)

CREATE TABLE profiles (

id UUID PRIMARY KEY REFERENCES users(id),

avatar\_url TEXT,

bio TEXT,

organization TEXT,

designation TEXT,

practice\_areas TEXT[],

years\_of\_experience INTEGER,

bar\_council\_id TEXT,

preferences JSONB DEFAULT '{}'::JSONB,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- Row Level Security

ALTER TABLE profiles ENABLE ROW LEVEL SECURITY;

-- Policies (similar to users table)

CREATE POLICY "Users can view their own profile"

ON profiles FOR SELECT

USING (auth.uid() = id);

CREATE POLICY "Super admins can view all profiles"

ON profiles FOR SELECT

USING (

EXISTS (

SELECT 1 FROM users

WHERE id = auth.uid() AND role = 'super\_admin'

)

);

CREATE POLICY "Users can update their own profile"

ON profiles FOR UPDATE

USING (auth.uid() = id);

### Subscription Tiers Table

CREATE TABLE subscription\_tiers (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

name TEXT UNIQUE NOT NULL CHECK (name IN ('free', 'pro', 'enterprise')),

display\_name TEXT NOT NULL,

price DECIMAL(10, 2) NOT NULL,

currency TEXT NOT NULL DEFAULT 'INR',

billing\_period TEXT NOT NULL CHECK (billing\_period IN ('monthly', 'annual')),

features JSONB NOT NULL,

is\_active BOOLEAN DEFAULT TRUE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

created\_by UUID REFERENCES users(id)

);

-- Row Level Security

ALTER TABLE subscription\_tiers ENABLE ROW LEVEL SECURITY;

-- Policies

CREATE POLICY "Anyone can view active subscription tiers"

ON subscription\_tiers FOR SELECT

USING (is\_active = TRUE);

CREATE POLICY "Super admins can manage subscription tiers"

ON subscription\_tiers

USING (

EXISTS (

SELECT 1 FROM users

WHERE id = auth.uid() AND role = 'super\_admin'

)

);

### Subscriptions Table

CREATE TABLE subscriptions (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

user\_id UUID NOT NULL REFERENCES users(id),

tier\_id UUID NOT NULL REFERENCES subscription\_tiers(id),

status TEXT NOT NULL CHECK (status IN ('active', 'canceled', 'expired', 'pending')),

start\_date TIMESTAMP WITH TIME ZONE NOT NULL,

end\_date TIMESTAMP WITH TIME ZONE NOT NULL,

auto\_renew BOOLEAN DEFAULT FALSE,

payment\_method JSONB,

last\_payment\_date TIMESTAMP WITH TIME ZONE,

next\_payment\_date TIMESTAMP WITH TIME ZONE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- Row Level Security

ALTER TABLE subscriptions ENABLE ROW LEVEL SECURITY;

-- Policies

CREATE POLICY "Users can view their own subscriptions"

ON subscriptions FOR SELECT

USING (auth.uid() = user\_id);

CREATE POLICY "Super admins can view all subscriptions"

ON subscriptions FOR SELECT

USING (

EXISTS (

SELECT 1 FROM users

WHERE id = auth.uid() AND role = 'super\_admin'

)

);

CREATE POLICY "Admins can view subscriptions"

ON subscriptions FOR SELECT

USING (

EXISTS (

SELECT 1 FROM users

WHERE id = auth.uid() AND role = 'admin'

)

);

CREATE POLICY "Super admins can update subscriptions"

ON subscriptions FOR UPDATE

USING (

EXISTS (

SELECT 1 FROM users

WHERE id = auth.uid() AND role = 'super\_admin'

)

);

### Payments Table

CREATE TABLE payments (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

subscription\_id UUID NOT NULL REFERENCES subscriptions(id),

amount DECIMAL(10, 2) NOT NULL,

currency TEXT NOT NULL DEFAULT 'INR',

payment\_date TIMESTAMP WITH TIME ZONE NOT NULL,

payment\_method TEXT NOT NULL,

transaction\_id TEXT UNIQUE,

status TEXT NOT NULL CHECK (status IN ('successful', 'failed', 'pending', 'refunded')),

invoice\_url TEXT,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- Row Level Security

ALTER TABLE payments ENABLE ROW LEVEL SECURITY;

-- Policies

CREATE POLICY "Users can view their own payments"

ON payments FOR SELECT

USING (

EXISTS (

SELECT 1 FROM subscriptions

WHERE id = payments.subscription\_id AND user\_id = auth.uid()

)

);

CREATE POLICY "Super admins can view all payments"

ON payments FOR SELECT

USING (

EXISTS (

SELECT 1 FROM users

WHERE id = auth.uid() AND role = 'super\_admin'

)

);

### Case Briefs Table

CREATE TABLE case\_briefs (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

user\_id UUID NOT NULL REFERENCES users(id),

title TEXT NOT NULL,

content TEXT NOT NULL,

tags TEXT[],

is\_public BOOLEAN DEFAULT FALSE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- Row Level Security

ALTER TABLE case\_briefs ENABLE ROW LEVEL SECURITY;

-- Policies

CREATE POLICY "Users can manage their own briefs"

ON case\_briefs

USING (auth.uid() = user\_id);

CREATE POLICY "Public briefs are viewable by all"

ON case\_briefs FOR SELECT

USING (is\_public = TRUE);

CREATE POLICY "Super admins can view all briefs"

ON case\_briefs FOR SELECT

USING (

EXISTS (

SELECT 1 FROM users

WHERE id = auth.uid() AND role = 'super\_admin'

)

);

### Analysis Results Table

CREATE TABLE analysis\_results (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

brief\_id UUID NOT NULL REFERENCES case\_briefs(id),

law\_sections JSONB,

case\_histories JSONB,

analysis\_content TEXT,

model\_version TEXT,

processing\_time DECIMAL,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- Row Level Security

ALTER TABLE analysis\_results ENABLE ROW LEVEL SECURITY;

-- Policies

CREATE POLICY "Users can view results for their briefs"

ON analysis\_results FOR SELECT

USING (

EXISTS (

SELECT 1 FROM case\_briefs

WHERE id = analysis\_results.brief\_id AND user\_id = auth.uid()

)

);

CREATE POLICY "Results for public briefs are viewable"

ON analysis\_results FOR SELECT

USING (

EXISTS (

SELECT 1 FROM case\_briefs

WHERE id = analysis\_results.brief\_id AND is\_public = TRUE

)

);

CREATE POLICY "Super admins can view all results"

ON analysis\_results FOR SELECT

USING (

EXISTS (

SELECT 1 FROM users

WHERE id = auth.uid() AND role = 'super\_admin'

)

);

### Case Files Table

CREATE TABLE case\_files (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

brief\_id UUID NOT NULL REFERENCES case\_briefs(id),

title TEXT NOT NULL,

content TEXT NOT NULL,

format TEXT NOT NULL DEFAULT 'markdown',

version INTEGER NOT NULL DEFAULT 1,

status TEXT NOT NULL DEFAULT 'draft' CHECK (status IN ('draft', 'final', 'archived')),

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- Row Level Security

ALTER TABLE case\_files ENABLE ROW LEVEL SECURITY;

-- Policies (similar to analysis\_results)

CREATE POLICY "Users can manage files for their briefs"

ON case\_files

USING (

EXISTS (

SELECT 1 FROM case\_briefs

WHERE id = case\_files.brief\_id AND user\_id = auth.uid()

)

);

### Feature Access Table

CREATE TABLE feature\_access (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

tier\_id UUID NOT NULL REFERENCES subscription\_tiers(id),

feature\_key TEXT NOT NULL,

feature\_name TEXT NOT NULL,

is\_enabled BOOLEAN DEFAULT TRUE,

limits JSONB,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

updated\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),

UNIQUE(tier\_id, feature\_key)

);

-- Row Level Security

ALTER TABLE feature\_access ENABLE ROW LEVEL SECURITY;

-- Policies

CREATE POLICY "Anyone can view feature access"

ON feature\_access FOR SELECT

USING (TRUE);

CREATE POLICY "Super admins can manage feature access"

ON feature\_access

USING (

EXISTS (

SELECT 1 FROM users

WHERE id = auth.uid() AND role = 'super\_admin'

)

);

### Usage Tracking Table

CREATE TABLE usage\_tracking (

id UUID PRIMARY KEY DEFAULT uuid\_generate\_v4(),

user\_id UUID NOT NULL REFERENCES users(id),

feature\_key TEXT NOT NULL,

usage\_count INTEGER NOT NULL DEFAULT 1,

usage\_date DATE NOT NULL DEFAULT CURRENT\_DATE,

created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

-- Row Level Security

ALTER TABLE usage\_tracking ENABLE ROW LEVEL SECURITY;

-- Policies

CREATE POLICY "Users can view their own usage"

ON usage\_tracking FOR SELECT

USING (auth.uid() = user\_id);

CREATE POLICY "Super admins can view all usage"

ON usage\_tracking FOR SELECT

USING (

EXISTS (

SELECT 1 FROM users

WHERE id = auth.uid() AND role = 'super\_admin'

)

);

CREATE POLICY "System can insert usage records"

ON usage\_tracking FOR INSERT

WITH CHECK (auth.uid() = user\_id OR

EXISTS (

SELECT 1 FROM users

WHERE id = auth.uid() AND role IN ('super\_admin', 'admin')

)

);

## Database Functions and Triggers

### User Registration Function

CREATE OR REPLACE FUNCTION handle\_new\_user()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO profiles (id)

VALUES (NEW.id);

-- Create default free subscription

INSERT INTO subscriptions (

user\_id,

tier\_id,

status,

start\_date,

end\_date

)

VALUES (

NEW.id,

(SELECT id FROM subscription\_tiers WHERE name = 'free' LIMIT 1),

'active',

NOW(),

(NOW() + INTERVAL '100 years')

);

RETURN NEW;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE TRIGGER on\_auth\_user\_created

AFTER INSERT ON users

FOR EACH ROW EXECUTE PROCEDURE handle\_new\_user();

### Subscription Status Update Function

CREATE OR REPLACE FUNCTION check\_subscription\_status()

RETURNS TRIGGER AS $$

BEGIN

-- Update subscription status based on dates

IF NEW.end\_date < NOW() THEN

NEW.status := 'expired';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql SECURITY DEFINER;

CREATE TRIGGER before\_subscription\_update

BEFORE UPDATE ON subscriptions

FOR EACH ROW EXECUTE PROCEDURE check\_subscription\_status();

## Initial Data Setup

### Default Subscription Tiers

INSERT INTO subscription\_tiers (name, display\_name, price, currency, billing\_period, features)

VALUES

('free', 'Free Tier', 0, 'INR', 'monthly', '{

"case\_briefs\_limit": 5,

"law\_sections": true,

"case\_histories": true,

"download\_formats": ["pdf"],

"ai\_analysis": false,

"case\_drafting": false

}'),

('pro', 'Professional', 499, 'INR', 'monthly', '{

"case\_briefs\_limit": 50,

"law\_sections": true,

"case\_histories": true,

"download\_formats": ["pdf", "docx", "rtf"],

"ai\_analysis": true,

"case\_drafting": false

}'),

('enterprise', 'Enterprise', 4999, 'INR', 'monthly', '{

"case\_briefs\_limit": -1,

"law\_sections": true,

"case\_histories": true,

"download\_formats": ["pdf", "docx", "rtf"],

"ai\_analysis": true,

"case\_drafting": true,

"priority\_support": true,

"custom\_integrations": true

}');

### Feature Access Setup

-- Free Tier Features

INSERT INTO feature\_access (tier\_id, feature\_key, feature\_name, is\_enabled, limits)

SELECT

id,

'case\_briefs',

'Case Brief Analysis',

TRUE,

'{"monthly\_limit": 5}'

FROM subscription\_tiers

WHERE name = 'free';

INSERT INTO feature\_access (tier\_id, feature\_key, feature\_name, is\_enabled, limits)

SELECT

id,

'law\_sections',

'Law Section Extraction',

TRUE,

NULL

FROM subscription\_tiers

WHERE name = 'free';

-- Pro Tier Features

INSERT INTO feature\_access (tier\_id, feature\_key, feature\_name, is\_enabled, limits)

SELECT

id,

'case\_briefs',

'Case Brief Analysis',

TRUE,

'{"monthly\_limit": 50}'

FROM subscription\_tiers

WHERE name = 'pro';

INSERT INTO feature\_access (tier\_id, feature\_key, feature\_name, is\_enabled, limits)

SELECT

id,

'ai\_analysis',

'AI-Powered Legal Analysis',

TRUE,

NULL

FROM subscription\_tiers

WHERE name = 'pro';

-- Enterprise Tier Features

INSERT INTO feature\_access (tier\_id, feature\_key, feature\_name, is\_enabled, limits)

SELECT

id,

'case\_briefs',

'Case Brief Analysis',

TRUE,

'{"monthly\_limit": null}'

FROM subscription\_tiers

WHERE name = 'enterprise';

INSERT INTO feature\_access (tier\_id, feature\_key, feature\_name, is\_enabled, limits)

SELECT

id,

'case\_drafting',

'Automated Case File Drafting',

TRUE,

NULL

FROM subscription\_tiers

WHERE name = 'enterprise';

## Supabase Authentication Hooks

### Custom Claims in JWT

// This would be implemented in a Supabase Edge Function

import { serve } from 'https://deno.land/std@0.131.0/http/server.ts'

import { createClient } from 'https://esm.sh/@supabase/supabase-js@2.0.0'

serve(async (req) => {

const supabaseClient = createClient(

Deno.env.get('SUPABASE\_URL') ?? '',

Deno.env.get('SUPABASE\_SERVICE\_ROLE\_KEY') ?? ''

)

const { event, user } = await req.json()

if (event === 'SIGNED\_IN') {

try {

// Get user role

const { data: userData, error: userError } = await supabaseClient

.from('users')

.select('role')

.eq('id', user.id)

.single()

if (userError) throw userError

// Get user subscription

const { data: subscriptionData, error: subscriptionError } = await supabaseClient

.from('subscriptions')

.select('tier\_id, status')

.eq('user\_id', user.id)

.eq('status', 'active')

.order('created\_at', { ascending: false })

.limit(1)

.single()

if (subscriptionError && subscriptionError.code !== 'PGRST116') throw subscriptionError

// Get subscription tier

let tierData = null

if (subscriptionData) {

const { data: tierInfo, error: tierError } = await supabaseClient

.from('subscription\_tiers')

.select('name')

.eq('id', subscriptionData.tier\_id)

.single()

if (tierError) throw tierError

tierData = tierInfo

}

// Set custom claims

const { error: claimsError } = await supabaseClient.auth.admin.updateUserById(

user.id,

{

app\_metadata: {

role: userData.role,

subscription\_tier: tierData ? tierData.name : 'free'

}

}

)

if (claimsError) throw claimsError

return new Response(JSON.stringify({ message: 'Custom claims added successfully' }), {

headers: { 'Content-Type': 'application/json' },

status: 200

})

} catch (error) {

return new Response(JSON.stringify({ error: error.message }), {

headers: { 'Content-Type': 'application/json' },

status: 400

})

}

}

return new Response(JSON.stringify({ message: 'No action taken' }), {

headers: { 'Content-Type': 'application/json' },

status: 200

})

})

## Security Considerations

1. **Data Encryption**
   * Enable Supabase's column-level encryption for sensitive data
   * Use TLS for all API communications
2. **Authentication Security**
   * Implement password complexity requirements
   * Use two-factor authentication for all admin accounts
   * Set appropriate token expiry times
3. **Row-Level Security**
   * All tables have RLS policies to restrict access based on user role
   * Super Admin access is carefully controlled
   * Public data is explicitly marked as such
4. **API Security**
   * Rate limiting on authentication endpoints
   * Input validation on all API endpoints
   * CORS configuration to restrict API access
5. **Audit Logging**
   * Track all significant data modifications
   * Log authentication events
   * Monitor for suspicious activity

## Next Steps

1. Create Supabase project and configure authentication settings
2. Implement database schema with all tables and RLS policies
3. Set up initial data for subscription tiers and features
4. Create authentication hooks for custom JWT claims
5. Develop and test API endpoints for user management and authentication
6. Implement subscription management functionality