Contents

SECTION 40: APPLICATION-LEVEL DATA ISOLATION (v4.6.0)

CRITICAL: This section implements complete isolation between client apps (Think Tank, Launch Board, AlwaysMe, Mechanical Maker) AND Think Tank. Same user email in different apps = completely separate identities and data.

40.1 ARCHITECTURE OVERVIEW

The Problem

Before v4.6.0, RADIANT had **tenant-level isolation** but not **application-level isolation**:

```
BEFORE (v4.5.0 and earlier):

  Tenant: Acme Corp (tenant_id: abc-123)


   User: alice@acme.com (single user record)
     Can access Think Tank data
     Can access Launch Board data     ← PROBLEM!
     Can access Think Tank data           ← PROBLEM!
     All apps share same chat history, preferences, etc.

   RLS Policy: WHERE tenant_id = current_setting('app.current_tenant_id')
     Only filters by tenant, not by app
```

The Solution

v4.6.0 implements **application-level isolation**:

```
AFTER (v4.6.0):

  Tenant: Acme Corp (tenant_id: abc-123)



     App: Think Tank (app_id: thinktank)

      AppUser: alice@acme.com (app_user_id: usr-tt-001)
        Chats, preferences, history ONLY for Think Tank
```

```
  App: Launch Board (app_id: launchboard)

    AppUser: alice@acme.com (app_user_id: usr-lb-002)
      Chats, preferences, history ONLY for Launch Board



  App: Think Tank (app_id: thinktank)

    AppUser: alice@acme.com (app_user_id: usr-sv-003)
      Chats, preferences, history ONLY for Think Tank
      COMPLETELY ISOLATED from Think Tank and Launch Board



RLS Policy: WHERE tenant_id = :tenant AND app_id = :app
  Filters by BOTH tenant AND app
```

Isolation Layers

```
                        RADIANT v4.6.0 ISOLATION LAYERS


LAYER 1: COGNITO (Identity)

  Think Tank Pool       Launch Board Pool     Think Tank Pool
  (thinktank-prod)      (launchboard-prod)    (thinktank-prod)

  custom:app_id =       custom:app_id =       custom:app_id =
  "thinktank"           "launchboard"         "thinktank"



LAYER 2: API GATEWAY (Routing)

  thinktank.domain.com    → thinktank API
  launchboard.domain.com  → launchboard API
  thinktank.domain.com  → thinktank API

  JWT Validation: Verify app_id in token matches route
```

```
LAYER 3: LAMBDA (Context)

  AuthContext = {
    tenantId: 'abc-123',
    appId: 'thinktank',      ← NEW: App ID extracted from JWT
    userId: 'usr-tt-001',    ← App-scoped user ID
    email: 'alice@acme.com'
  }

  Database Connection:
    SET app.current_tenant_id = 'abc-123';
    SET app.current_app_id = 'thinktank';   ← NEW



LAYER 4: DATABASE (RLS)

  CREATE POLICY app_isolation ON user_data
    USING (
      tenant_id = current_setting('app.current_tenant_id')::UUID
      AND
      app_id = current_setting('app.current_app_id')
    );

  Every SELECT/INSERT/UPDATE/DELETE filtered by BOTH
```

40.2 DATABASE SCHEMA CHANGES

Migration: 040_app_level_isolation.sql

```sql
-- ============================================================================
-- RADIANT v4.6.0 - Application-Level Data Isolation
-- ============================================================================
-- This migration adds app_id isolation to all user-facing tables
-- ============================================================================

-- Create function to get current app_id from session
CREATE OR REPLACE FUNCTION current_app_id() RETURNS VARCHAR(50) AS $$
BEGIN
    RETURN NULLIF(current_setting('app.current_app_id', true), '');
EXCEPTION WHEN OTHERS THEN RETURN NULL;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER STABLE;
```

```sql
-- ============================================================================
-- APP_USERS: App-Scoped User Instances
-- ============================================================================
-- A single email can have MULTIPLE app_user records (one per app)
-- This is the PRIMARY user identity within each application

CREATE TABLE IF NOT EXISTS app_users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),

    -- Multi-tenant + Multi-app isolation
    tenant_id UUID NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,
    app_id VARCHAR(50) NOT NULL,

    -- Link to base user (optional - for cross-app identity correlation by admins only)
    base_user_id UUID REFERENCES users(id) ON DELETE SET NULL,

    -- Identity (same email can exist in multiple apps)
    cognito_sub VARCHAR(100) NOT NULL,
    email VARCHAR(255) NOT NULL,
    email_verified BOOLEAN DEFAULT false,

    -- Profile (app-specific)
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    display_name VARCHAR(200),
    avatar_url TEXT,

    -- App-specific preferences
    preferences JSONB DEFAULT '{}',
    timezone VARCHAR(50) DEFAULT 'UTC',
    locale VARCHAR(10) DEFAULT 'en-US',

    -- Status
    status VARCHAR(20) NOT NULL DEFAULT 'active' CHECK (status IN ('active', 'inactive', 'suspe
    last_login_at TIMESTAMPTZ,
    login_count INTEGER DEFAULT 0,

    -- Timestamps
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    deleted_at TIMESTAMPTZ,

    -- Unique constraint: one user per email per app per tenant
    UNIQUE(tenant_id, app_id, email),
    -- Cognito sub is unique per app
    UNIQUE(app_id, cognito_sub)
);
```

```sql
-- Indexes for app_users
CREATE INDEX idx_app_users_tenant_app ON app_users(tenant_id, app_id);
CREATE INDEX idx_app_users_email ON app_users(email);
CREATE INDEX idx_app_users_cognito ON app_users(cognito_sub);
CREATE INDEX idx_app_users_base_user ON app_users(base_user_id);
CREATE INDEX idx_app_users_status ON app_users(status) WHERE deleted_at IS NULL;

-- RLS for app_users
ALTER TABLE app_users ENABLE ROW LEVEL SECURITY;

CREATE POLICY app_users_isolation ON app_users
    FOR ALL
    USING (
        tenant_id = current_tenant_id()
        AND app_id = current_app_id()
    );

-- Admin policy: Platform admins can view all users within their tenant
CREATE POLICY app_users_admin_view ON app_users
    FOR SELECT
    USING (
        tenant_id = current_tenant_id()
        AND current_setting('app.is_admin', true) = 'true'
    );

-- ============================================================================
-- ADD app_id TO EXISTING TABLES
-- ============================================================================

-- Add app_id column to user-facing tables that need isolation
-- Using IF NOT EXISTS pattern for idempotent migrations

-- Chats / Conversations (Think Tank and Client Apps)
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                   WHERE table_name = 'thinktank_chats' AND column_name = 'app_id') THEN
        ALTER TABLE thinktank_chats ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinktank'
    END IF;
END $$;

DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                   WHERE table_name = 'thinktank_messages' AND column_name = 'app_id') THEN
        ALTER TABLE thinktank_messages ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinkta
    END IF;
```

```
END $$;

DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                   WHERE table_name = 'thinktank_conversations' AND column_name = 'app_id') THE
        ALTER TABLE thinktank_conversations ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thi
    END IF;
END $$;

-- Concurrent Sessions
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                   WHERE table_name = 'concurrent_sessions' AND column_name = 'app_id') THEN
        ALTER TABLE concurrent_sessions ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinkta
    END IF;
END $$;

-- Voice Sessions
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                   WHERE table_name = 'voice_sessions' AND column_name = 'app_id') THEN
        ALTER TABLE voice_sessions ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinktank';
    END IF;
END $$;

-- Memory
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                   WHERE table_name = 'memory_stores' AND column_name = 'app_id') THEN
        ALTER TABLE memory_stores ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinktank';
    END IF;

    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                   WHERE table_name = 'memories' AND column_name = 'app_id') THEN
        ALTER TABLE memories ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinktank';
    END IF;
END $$;

-- Canvases & Artifacts
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                   WHERE table_name = 'canvases' AND column_name = 'app_id') THEN
        ALTER TABLE canvases ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinktank';
```

```
    END IF;

    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                   WHERE table_name = 'artifacts' AND column_name = 'app_id') THEN
        ALTER TABLE artifacts ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinktank';
    END IF;
END $$;

-- User Personas
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                   WHERE table_name = 'user_personas' AND column_name = 'app_id') THEN
        ALTER TABLE user_personas ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinktank';
    END IF;
END $$;

-- Scheduled Prompts
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                   WHERE table_name = 'scheduled_prompts' AND column_name = 'app_id') THEN
        ALTER TABLE scheduled_prompts ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinktanl
    END IF;
END $$;

-- User Preferences (Neural Engine)
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                   WHERE table_name = 'user_preferences' AND column_name = 'app_id') THEN
        ALTER TABLE user_preferences ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinktank
    END IF;
END $$;

-- User Memory (Neural Engine)
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                   WHERE table_name = 'user_memory' AND column_name = 'app_id') THEN
        ALTER TABLE user_memory ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinktank';
    END IF;
END $$;

-- User Behavior Patterns
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
```

```
                    WHERE table_name = 'user_behavior_patterns' AND column_name = 'app_id') THEN
        ALTER TABLE user_behavior_patterns ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thi
    END IF;
END $$;

-- Feedback tables
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                    WHERE table_name = 'feedback_explicit' AND column_name = 'app_id') THEN
        ALTER TABLE feedback_explicit ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinktan
    END IF;

    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                    WHERE table_name = 'feedback_implicit' AND column_name = 'app_id') THEN
        ALTER TABLE feedback_implicit ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinktan
    END IF;

    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                    WHERE table_name = 'feedback_voice' AND column_name = 'app_id') THEN
        ALTER TABLE feedback_voice ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinktank';
    END IF;
END $$;

-- Execution Manifests
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                    WHERE table_name = 'execution_manifests' AND column_name = 'app_id') THEN
        ALTER TABLE execution_manifests ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinkta
    END IF;
END $$;

-- Think Tank-specific tables
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                    WHERE table_name = 'thinktank_sessions' AND column_name = 'app_id') THEN
        ALTER TABLE thinktank_sessions ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinktan
    END IF;

    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                    WHERE table_name = 'thinktank_user_model_preferences' AND column_name = 'app
        ALTER TABLE thinktank_user_model_preferences ADD COLUMN app_id VARCHAR(50) NOT NULL DEF
    END IF;
END $$;

-- Collaboration
```

```
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                   WHERE table_name = 'collaboration_rooms' AND column_name = 'app_id') THEN
        ALTER TABLE collaboration_rooms ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinkta
    END IF;
END $$;

-- Time Machine snapshots
DO $$
BEGIN
    IF NOT EXISTS (SELECT 1 FROM information_schema.columns
                   WHERE table_name = 'chat_snapshots' AND column_name = 'app_id') THEN
        ALTER TABLE chat_snapshots ADD COLUMN app_id VARCHAR(50) NOT NULL DEFAULT 'thinktank';
    END IF;
END $$;

-- ============================================================================
-- CREATE INDEXES FOR app_id COLUMNS
-- ============================================================================

-- Indexes for efficient app-scoped queries
CREATE INDEX CONCURRENTLY IF NOT EXISTS idx_thinktank_chats_app ON thinktank_chats(tenant_id, a
CREATE INDEX CONCURRENTLY IF NOT EXISTS idx_thinktank_messages_app ON thinktank_messages(tenant
CREATE INDEX CONCURRENTLY IF NOT EXISTS idx_concurrent_sessions_app ON concurrent_sessions(tena
CREATE INDEX CONCURRENTLY IF NOT EXISTS idx_voice_sessions_app ON voice_sessions(tenant_id, app
CREATE INDEX CONCURRENTLY IF NOT EXISTS idx_memory_stores_app ON memory_stores(tenant_id, app_i
CREATE INDEX CONCURRENTLY IF NOT EXISTS idx_canvases_app ON canvases(tenant_id, app_id);
CREATE INDEX CONCURRENTLY IF NOT EXISTS idx_user_personas_app ON user_personas(tenant_id, app_i
CREATE INDEX CONCURRENTLY IF NOT EXISTS idx_user_preferences_app ON user_preferences(tenant_id
CREATE INDEX CONCURRENTLY IF NOT EXISTS idx_feedback_explicit_app ON feedback_explicit(tenant_i
CREATE INDEX CONCURRENTLY IF NOT EXISTS idx_execution_manifests_app ON execution_manifests(tena

-- ============================================================================
-- UPDATE RLS POLICIES TO INCLUDE app_id
-- ============================================================================

-- Drop existing policies and recreate with app_id filtering
-- Using a function to standardize policy creation

CREATE OR REPLACE FUNCTION create_app_isolation_policy(
    table_name TEXT,
    policy_name TEXT DEFAULT NULL
) RETURNS VOID AS $$
DECLARE
    p_name TEXT;
BEGIN
    p_name := COALESCE(policy_name, table_name || '_app_isolation');
```

```
    -- Drop existing policy if exists
    EXECUTE format('DROP POLICY IF EXISTS %I ON %I', p_name, table_name);

    -- Create new policy with app_id filtering
    EXECUTE format('
        CREATE POLICY %I ON %I
        FOR ALL
        USING (
            tenant_id = current_tenant_id()
            AND app_id = current_app_id()
        )
    ', p_name, table_name);
END;
$$ LANGUAGE plpgsql;

-- Apply app isolation policies to all relevant tables
SELECT create_app_isolation_policy('thinktank_chats');
SELECT create_app_isolation_policy('thinktank_messages');
SELECT create_app_isolation_policy('thinktank_conversations');
SELECT create_app_isolation_policy('concurrent_sessions');
SELECT create_app_isolation_policy('voice_sessions');
SELECT create_app_isolation_policy('memory_stores');
SELECT create_app_isolation_policy('memories');
SELECT create_app_isolation_policy('canvases');
SELECT create_app_isolation_policy('artifacts');
SELECT create_app_isolation_policy('user_personas');
SELECT create_app_isolation_policy('scheduled_prompts');
SELECT create_app_isolation_policy('user_preferences');
SELECT create_app_isolation_policy('user_memory');
SELECT create_app_isolation_policy('user_behavior_patterns');
SELECT create_app_isolation_policy('feedback_explicit');
SELECT create_app_isolation_policy('feedback_implicit');
SELECT create_app_isolation_policy('feedback_voice');
SELECT create_app_isolation_policy('execution_manifests');
SELECT create_app_isolation_policy('thinktank_sessions');
SELECT create_app_isolation_policy('thinktank_user_model_preferences');
SELECT create_app_isolation_policy('collaboration_rooms');

-- ============================================================================
-- ADMIN CROSS-APP VIEW POLICIES
-- ============================================================================
-- Platform administrators need to view data across apps for support

CREATE OR REPLACE FUNCTION create_admin_view_policy(
    table_name TEXT
) RETURNS VOID AS $$
BEGIN
```

```
    EXECUTE format('DROP POLICY IF EXISTS %I ON %I', table_name || '_admin_view', table_name);

    EXECUTE format('
        CREATE POLICY %I ON %I
        FOR SELECT
        USING (
            tenant_id = current_tenant_id()
            AND current_setting(''app.is_admin'', true) = ''true''
        )
    ', table_name || '_admin_view', table_name);
END;
$$ LANGUAGE plpgsql;

-- Apply admin view policies
SELECT create_admin_view_policy('thinktank_chats');
SELECT create_admin_view_policy('thinktank_messages');
SELECT create_admin_view_policy('concurrent_sessions');
SELECT create_admin_view_policy('memory_stores');
SELECT create_admin_view_policy('canvases');
SELECT create_admin_view_policy('user_personas');
SELECT create_admin_view_policy('feedback_explicit');
SELECT create_admin_view_policy('execution_manifests');

-- ============================================================================
-- APP REGISTRY TABLE
-- ============================================================================
-- Track all registered applications

CREATE TABLE IF NOT EXISTS app_registry (
    id VARCHAR(50) PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    display_name VARCHAR(200) NOT NULL,
    description TEXT,

    -- App type
    app_type VARCHAR(50) NOT NULL CHECK (app_type IN ('client_app', 'consumer', 'admin', 'inte

    -- Cognito configuration
    cognito_user_pool_id VARCHAR(100),
    cognito_client_id VARCHAR(100),

    -- Routing
    subdomain VARCHAR(100),
    custom_domain VARCHAR(255),

    -- Features enabled for this app
    features JSONB DEFAULT '{}',
```

```
    -- Status
    status VARCHAR(20) NOT NULL DEFAULT 'active' CHECK (status IN ('active', 'inactive', 'depr

    -- Timestamps
    created_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Seed default applications
INSERT INTO app_registry (id, name, display_name, description, app_type, subdomain) VALUES
    ('thinktank', 'thinktank', 'Think Tank', 'Consumer-facing AI chat application', 'consumer'
    ('thinktank', 'thinktank', 'Think Tank', 'Collaborative AI workspace', 'client_app', 'thin
    ('launchboard', 'launchboard', 'Launch Board', 'Project management with AI', 'client_app',
    ('alwaysme', 'alwaysme', 'Always Me', 'Personal AI assistant', 'client_app', 'alwaysme'),
    ('mechanicalmaker', 'mechanicalmaker', 'Mechanical Maker', 'Engineering AI tools', 'client
    ('admin', 'admin', 'Admin Dashboard', 'Platform administration', 'admin', 'admin')
ON CONFLICT (id) DO NOTHING;

-- ============================================================================
-- CROSS-APP CORRELATION TABLE (Admin Only)
-- ============================================================================
-- For platform admins to correlate user identities across apps
-- End users CANNOT access this table

CREATE TABLE IF NOT EXISTS cross_app_user_correlation (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    tenant_id UUID NOT NULL REFERENCES tenants(id) ON DELETE CASCADE,

    -- Email is the correlation key
    email VARCHAR(255) NOT NULL,

    -- App user IDs for each app
    app_user_ids JSONB NOT NULL DEFAULT '{}',
    -- Example: {"thinktank": "uuid-1", "launchboard": "uuid-2", "launchboard": "uuid-3"}

    -- Metadata
    first_seen_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),
    last_activity_at TIMESTAMPTZ NOT NULL DEFAULT NOW(),

    -- Unique per tenant
    UNIQUE(tenant_id, email)
);

-- Only admins can access this table
ALTER TABLE cross_app_user_correlation ENABLE ROW LEVEL SECURITY;

CREATE POLICY cross_app_admin_only ON cross_app_user_correlation
    FOR ALL
```

```
    USING (
        tenant_id = current_tenant_id()
        AND current_setting('app.is_admin', true) = 'true'
    );

-- ============================================================================
-- MIGRATION TRACKING
-- ============================================================================

INSERT INTO schema_migrations (version, name, applied_by, checksum)
VALUES ('040', 'app_level_isolation', 'system', md5('v4.6.0_app_isolation'))
ON CONFLICT (version) DO NOTHING;
```

40.3 COGNITO CONFIGURATION

Per-App User Pool Stack

```
// packages/infrastructure/lib/stacks/app-cognito.stack.ts

import * as cdk from 'aws-cdk-lib';
import * as cognito from 'aws-cdk-lib/aws-cognito';
import { Construct } from 'constructs';

export interface AppCognitoStackProps extends cdk.StackProps {
  appId: string;
  appName: string;
  environment: string;
  domain: string;
  tier: number;
}

/**
 * Creates a dedicated Cognito User Pool for each application.
 * This ensures complete identity isolation between apps.
 */
export class AppCognitoStack extends cdk.Stack {
  public readonly userPool: cognito.UserPool;
  public readonly userPoolClient: cognito.UserPoolClient;
  public readonly userPoolDomain: cognito.UserPoolDomain;

  constructor(scope: Construct, id: string, props: AppCognitoStackProps) {
    super(scope, id, props);

    const resourcePrefix = `${props.appId}-${props.environment}`;

    // ============================================================================
    // USER POOL (Per-App)
```

```
// ============================================================================

this.userPool = new cognito.UserPool(this, 'UserPool', {
  userPoolName: `${resourcePrefix}-users`,

  // Self sign-up enabled for Think Tank, disabled for client apps
  selfSignUpEnabled: props.appId === 'thinktank',

  // Sign-in options
  signInAliases: {
    email: true,
    username: false,
  },

  // Password policy
  passwordPolicy: {
    minLength: 12,
    requireLowercase: true,
    requireUppercase: true,
    requireDigits: true,
    requireSymbols: true,
  },

  // MFA configuration
  mfa: props.tier >= 3
    ? cognito.Mfa.OPTIONAL
    : cognito.Mfa.OFF,
  mfaSecondFactor: {
    sms: true,
    otp: true,
  },

  // Account recovery
  accountRecovery: cognito.AccountRecovery.EMAIL_ONLY,

  // Standard attributes
  standardAttributes: {
    email: {
      required: true,
      mutable: true,
    },
    fullname: {
      required: false,
      mutable: true,
    },
  },

  // Custom attributes - CRITICAL: app_id is immutable
```

```
    customAttributes: {
      tenantId: new cognito.StringAttribute({ mutable: false }),
      appId: new cognito.StringAttribute({ mutable: false }), // NEW: App identifier
      role: new cognito.StringAttribute({ mutable: true }),
      appUserId: new cognito.StringAttribute({ mutable: false }), // NEW: App-scoped user ID
    },

    // Lambda triggers for app isolation
    lambdaTriggers: {
      preSignUp: this.createPreSignUpLambda(resourcePrefix, props.appId),
      postConfirmation: this.createPostConfirmationLambda(resourcePrefix, props.appId),
      preTokenGeneration: this.createPreTokenGenerationLambda(resourcePrefix, props.appId),
    },

    // Removal policy
    removalPolicy: props.environment === 'prod'
      ? cdk.RemovalPolicy.RETAIN
      : cdk.RemovalPolicy.DESTROY,
  });

  // ============================================================================
  // USER POOL CLIENT (Per-App)
  // ============================================================================

  this.userPoolClient = this.userPool.addClient('UserPoolClient', {
    userPoolClientName: `${resourcePrefix}-web-client`,

    // OAuth configuration
    oAuth: {
      flows: {
        authorizationCodeGrant: true,
        implicitCodeGrant: false,
      },
      scopes: [
        cognito.OAuthScope.EMAIL,
        cognito.OAuthScope.OPENID,
        cognito.OAuthScope.PROFILE,
      ],
      callbackUrls: [
        `https://${props.appId}.${props.domain}/auth/callback`,
        'http://localhost:3000/auth/callback',
      ],
      logoutUrls: [
        `https://${props.appId}.${props.domain}/auth/logout`,
        'http://localhost:3000/auth/logout',
      ],
    },
```

```
    // Token configuration
    accessTokenValidity: cdk.Duration.hours(1),
    idTokenValidity: cdk.Duration.hours(1),
    refreshTokenValidity: cdk.Duration.days(30),

    // Auth flows
    authFlows: {
      userPassword: true,
      userSrp: true,
    },

    // Prevent user existence errors
    preventUserExistenceErrors: true,

    // Read/write attributes
    readAttributes: new cognito.ClientAttributes()
      .withStandardAttributes({
        email: true,
        fullname: true,
      })
      .withCustomAttributes('tenantId', 'appId', 'role', 'appUserId'),

    writeAttributes: new cognito.ClientAttributes()
      .withStandardAttributes({
        fullname: true,
      }),
  });

  // ========================================================================
  // USER POOL DOMAIN (Per-App)
  // ========================================================================

  this.userPoolDomain = this.userPool.addDomain('Domain', {
    cognitoDomain: {
      domainPrefix: `${props.appId}-${props.environment}-auth`,
    },
  });

  // ========================================================================
  // OUTPUTS
  // ========================================================================

  new cdk.CfnOutput(this, 'UserPoolId', {
    value: this.userPool.userPoolId,
    exportName: `${resourcePrefix}-user-pool-id`,
  });

  new cdk.CfnOutput(this, 'UserPoolClientId', {
```

```
      value: this.userPoolClient.userPoolClientId,
      exportName: `${resourcePrefix}-user-pool-client-id`,
    });

    new cdk.CfnOutput(this, 'UserPoolDomain', {
      value: this.userPoolDomain.domainName,
      exportName: `${resourcePrefix}-user-pool-domain`,
    });
  }

  /**
   * Pre-SignUp Lambda: Validates sign-up requests and enforces app isolation
   */
  private createPreSignUpLambda(resourcePrefix: string, appId: string): lambda.Function {
    return new lambdaNodejs.NodejsFunction(this, 'PreSignUpFunction', {
      functionName: `${resourcePrefix}-pre-signup`,
      runtime: lambda.Runtime.NODEJS_20_X,
      handler: 'handler',
      entry: 'lambda/cognito/pre-signup.ts',
      environment: {
        APP_ID: appId,
      },
    });
  }

  /**
   * Post-Confirmation Lambda: Creates app_users record after confirmation
   */
  private createPostConfirmationLambda(resourcePrefix: string, appId: string): lambda.Function
    return new lambdaNodejs.NodejsFunction(this, 'PostConfirmationFunction', {
      functionName: `${resourcePrefix}-post-confirmation`,
      runtime: lambda.Runtime.NODEJS_20_X,
      handler: 'handler',
      entry: 'lambda/cognito/post-confirmation.ts',
      environment: {
        APP_ID: appId,
      },
    });
  }

  /**
   * Pre-Token-Generation Lambda: Adds app_id and app_user_id to JWT claims
   */
  private createPreTokenGenerationLambda(resourcePrefix: string, appId: string): lambda.Functi
    return new lambdaNodejs.NodejsFunction(this, 'PreTokenGenerationFunction', {
      functionName: `${resourcePrefix}-pre-token-gen`,
      runtime: lambda.Runtime.NODEJS_20_X,
      handler: 'handler',
```

```
      entry: 'lambda/cognito/pre-token-generation.ts',
      environment: {
        APP_ID: appId,
      },
    });
  }
}
```

Cognito Lambda Triggers

```
// lambda/cognito/pre-signup.ts

import { PreSignUpTriggerEvent, PreSignUpTriggerHandler } from 'aws-lambda';

/**
 * Pre-SignUp Lambda
 * Validates that the user is signing up for the correct app
 */
export const handler: PreSignUpTriggerHandler = async (event) => {
  const appId = process.env.APP_ID;

  // For Think Tank, auto-confirm email in non-prod environments
  if (appId === 'thinktank' && process.env.ENVIRONMENT !== 'prod') {
    event.response.autoConfirmUser = true;
    event.response.autoVerifyEmail = true;
  }

  // Log sign-up attempt for audit
  console.log('Pre-SignUp', {
    appId,
    email: event.request.userAttributes.email,
    userPoolId: event.userPoolId,
  });

  return event;
};
```

```
// lambda/cognito/post-confirmation.ts

import { PostConfirmationTriggerEvent, PostConfirmationTriggerHandler } from 'aws-lambda';
import { getDbClient } from '../shared/db';

/**
 * Post-Confirmation Lambda
 * Creates the app_users record after user confirms their email
 */
export const handler: PostConfirmationTriggerHandler = async (event) => {
  const appId = process.env.APP_ID!;
```

```
const tenantId = event.request.userAttributes['custom:tenantId'];
const cognitoSub = event.request.userAttributes.sub;
const email = event.request.userAttributes.email;
const fullName = event.request.userAttributes.name || '';

const db = await getDbClient();

try {
  // Create app-scoped user record
  const result = await db.query(`
    INSERT INTO app_users (
      tenant_id, app_id, cognito_sub, email,
      first_name, last_name, display_name, status
    )
    VALUES ($1, $2, $3, $4, $5, $6, $7, 'active')
    ON CONFLICT (tenant_id, app_id, email) DO UPDATE SET
      cognito_sub = EXCLUDED.cognito_sub,
      last_login_at = NOW(),
      login_count = app_users.login_count + 1
    RETURNING id
  `, [
    tenantId,
    appId,
    cognitoSub,
    email,
    fullName.split(' ')[0] || '',
    fullName.split(' ').slice(1).join(' ') || '',
    fullName || email.split('@')[0],
  ]);

  const appUserId = result.rows[0].id;

  // Update cross-app correlation (for admin use only)
  await db.query(`
    INSERT INTO cross_app_user_correlation (tenant_id, email, app_user_ids)
    VALUES ($1, $2, $3::jsonb)
    ON CONFLICT (tenant_id, email) DO UPDATE SET
      app_user_ids = cross_app_user_correlation.app_user_ids || $3::jsonb,
      last_activity_at = NOW()
  `, [tenantId, email, JSON.stringify({ [appId]: appUserId })]);

  console.log('Created app_user', { appId, appUserId, email });

} catch (error) {
  console.error('Error creating app_user', error);
  throw error;
}
```

```
  return event;
};
```

```
// lambda/cognito/pre-token-generation.ts

import { PreTokenGenerationTriggerEvent, PreTokenGenerationTriggerHandler } from 'aws-lambda';
import { getDbClient } from '../shared/db';

/**
 * Pre-Token-Generation Lambda
 * Adds app_id and app_user_id to JWT claims
 */
export const handler: PreTokenGenerationTriggerHandler = async (event) => {
  const appId = process.env.APP_ID!;
  const cognitoSub = event.request.userAttributes.sub;

  const db = await getDbClient();

  // Get app_user_id for this user
  const result = await db.query(`
    SELECT id FROM app_users
    WHERE cognito_sub = $1 AND app_id = $2
  `, [cognitoSub, appId]);

  const appUserId = result.rows[0]?.id;

  if (!appUserId) {
    console.error('No app_user found for', { cognitoSub, appId });
    throw new Error('User not found in this application');
  }

  // Add custom claims to the ID token
  event.response.claimsOverrideDetails = {
    claimsToAddOrOverride: {
      'custom:app_id': appId,
      'custom:app_user_id': appUserId,
    },
  };

  return event;
};
```

40.4 LAMBDA AUTH CONTEXT UPDATE

> **NOTE**: The `AuthContext` interface and `extractAuthContext` function are defined in **Section 4** (`lambda/shared/auth.ts`). The canonical definition already includes all app isolation fields: `appId`, `appUserId`, `isSuperAdmin`, `tokenExpiry`.

DO NOT redefine these types. Import from `@radiant/shared` or `../shared/auth`.

App Isolation Validation Logic

The following validation logic is already integrated into Section 4's `extractAuthContext`:

```
// Already in Section 4: lambda/shared/auth.ts
// Key validation points for app isolation:

// 1. Extract app-specific claims
const appId = claims['custom:appId'] || claims['custom:app_id'];
const appUserId = claims['custom:appUserId'] || claims['custom:app_user_id'];

// 2. Validate required claims for app isolation
if (!appId) throw new UnauthorizedError('Missing app ID');
if (!appUserId) throw new UnauthorizedError('Missing app user ID');

// 3. Validate app_id matches route (defense in depth)
const routeAppId = extractAppIdFromRoute(event);
if (routeAppId && routeAppId !== appId) {
  throw new ForbiddenError(`Token app_id (${appId}) does not match route (${routeAppId})`);
}
```

Extract App ID from Route Helper

```
// lambda/shared/auth.ts - extractAppIdFromRoute function
/**
 * Extract app_id from route for validation
 */
function extractAppIdFromRoute(event: APIGatewayProxyEvent): string | null {
  // Extract from subdomain: thinktank.domain.com -> thinktank
  const host = event.headers.Host || event.headers.host;
  if (host) {
    const subdomain = host.split('.')[0];
    if (['thinktank', 'launchboard', 'alwaysme', 'mechanicalmaker'].includes(subdomain)) {
      return subdomain;
    }
  }

  // Extract from path: /api/thinktank/... -> thinktank
  const pathMatch = event.path.match(/^\/api\/(thinktank|launchboard|alwaysme|mechanicalmaker)/
  if (pathMatch) {
    return pathMatch[1];
  }

  return null;
}

/**
```

```
 * Set database context for RLS policies
 * CRITICAL: This must be called before any database queries
 */
export async function setDatabaseContext(
  db: PoolClient,
  auth: AuthContext
): Promise<void> {
  await db.query(`
    SET LOCAL app.current_tenant_id = $1;
    SET LOCAL app.current_app_id = $2;
    SET LOCAL app.is_admin = $3;
  `, [auth.tenantId, auth.appId, auth.isAdmin ? 'true' : 'false']);
}

/**
 * Create authenticated database client with context
 */
export async function getAuthenticatedDb(
  auth: AuthContext
): Promise<{ client: PoolClient; release: () => void }> {
  const pool = await getPool();
  const client = await pool.connect();

  try {
    await setDatabaseContext(client, auth);

    return {
      client,
      release: () => client.release(),
    };
  } catch (error) {
    client.release();
    throw error;
  }
}
```

40.5 API ROUTING BY APP

API Gateway Per-App Routes

```
// packages/infrastructure/lib/stacks/api.stack.ts - Addition

/**
 * Create per-app API routes with app_id validation
 */
private createAppRoutes(props: APIStackProps): void {
  const apps = ['thinktank', 'launchboard', 'alwaysme', 'mechanicalmaker'];
```

```
for (const appId of apps) {
  // Create resource for this app
  const appResource = this.api.root.addResource(appId);

  // Apply app-specific authorizer
  const authorizer = new apigateway.CognitoUserPoolsAuthorizer(
    this,
    `${appId}Authorizer`,
    {
      cognitoUserPools: [props.appUserPools[appId]],
      identitySource: 'method.request.header.Authorization',
    }
  );

  // Chat endpoint
  const chatsResource = appResource.addResource('chats');
  chatsResource.addMethod('GET',
    new apigateway.LambdaIntegration(this.chatFunction),
    { authorizer }
  );
  chatsResource.addMethod('POST',
    new apigateway.LambdaIntegration(this.chatFunction),
    { authorizer }
  );

  // Chat by ID
  const chatResource = chatsResource.addResource('{chatId}');
  chatResource.addMethod('GET',
    new apigateway.LambdaIntegration(this.chatFunction),
    { authorizer }
  );
  chatResource.addMethod('DELETE',
    new apigateway.LambdaIntegration(this.chatFunction),
    { authorizer }
  );

  // Messages
  const messagesResource = chatResource.addResource('messages');
  messagesResource.addMethod('GET',
    new apigateway.LambdaIntegration(this.chatFunction),
    { authorizer }
  );
  messagesResource.addMethod('POST',
    new apigateway.LambdaIntegration(this.chatFunction),
    { authorizer }
  );
```

```
    // User preferences (app-specific)
    const preferencesResource = appResource.addResource('preferences');
    preferencesResource.addMethod('GET',
      new apigateway.LambdaIntegration(this.preferencesFunction),
      { authorizer }
    );
    preferencesResource.addMethod('PUT',
      new apigateway.LambdaIntegration(this.preferencesFunction),
      { authorizer }
    );

    // Memory (app-specific)
    const memoryResource = appResource.addResource('memory');
    memoryResource.addMethod('GET',
      new apigateway.LambdaIntegration(this.memoryFunction),
      { authorizer }
    );
  }
}
```

40.6 ADMIN DASHBOARD UPDATES

Cross-App User View (Admin Only)

```
// apps/admin-dashboard/src/app/(dashboard)/users/cross-app/page.tsx

'use client';

import { useState } from 'react';
import { useQuery } from '@tanstack/react-query';
import {
  Box,
  Card,
  CardContent,
  Typography,
  Table,
  TableHead,
  TableBody,
  TableRow,
  TableCell,
  Chip,
  TextField,
  InputAdornment,
  IconButton,
  Tooltip,
  Dialog,
  DialogTitle,
```

```
  DialogContent,
} from '@mui/material';
import { Search, Visibility, Apps, Person } from '@mui/icons-material';

interface CrossAppUser {
  email: string;
  tenantId: string;
  appUsers: {
    appId: string;
    appUserId: string;
    displayName: string;
    lastLoginAt: string;
    status: string;
    chatCount: number;
  }[];
  firstSeenAt: string;
  lastActivityAt: string;
}

export default function CrossAppUsersPage() {
  const [search, setSearch] = useState('');
  const [selectedUser, setSelectedUser] = useState<CrossAppUser | null>(null);

  const { data: users, isLoading } = useQuery({
    queryKey: ['cross-app-users', search],
    queryFn: async () => {
      const response = await fetch(
        `/api/admin/users/cross-app?search=${encodeURIComponent(search)}`
      );
      return response.json() as Promise<CrossAppUser[]>;
    },
  });

  const appColors: Record<string, string> = {
    thinktank: 'primary',
    thinktank: 'secondary',
    launchboard: 'success',
    alwaysme: 'warning',
    mechanicalmaker: 'info',
  };

  return (
    <Box>
      <Typography variant="h4" gutterBottom>
        Cross-App User Correlation
      </Typography>
      <Typography variant="body2" color="textSecondary" sx={{ mb: 3 }}>
        View users across all applications. This is for admin support purposes only.
```

```
  End users cannot see data from other applications.
</Typography>

<Card sx={{ mb: 3 }}>
  <CardContent>
    <TextField
      fullWidth
      placeholder="Search by email..."
      value={search}
      onChange={(e) => setSearch(e.target.value)}
      InputProps={{
        startAdornment: (
          <InputAdornment position="start">
            <Search />
          </InputAdornment>
        ),
      }}
    />
  </CardContent>
</Card>

<Card>
  <Table>
    <TableHead>
      <TableRow>
        <TableCell>Email</TableCell>
        <TableCell>Applications</TableCell>
        <TableCell>First Seen</TableCell>
        <TableCell>Last Activity</TableCell>
        <TableCell>Actions</TableCell>
      </TableRow>
    </TableHead>
    <TableBody>
      {users?.map((user) => (
        <TableRow key={user.email}>
          <TableCell>
            <Box sx={{ display: 'flex', alignItems: 'center', gap: 1 }}>
              <Person fontSize="small" />
              {user.email}
            </Box>
          </TableCell>
          <TableCell>
            <Box sx={{ display: 'flex', gap: 0.5, flexWrap: 'wrap' }}>
              {user.appUsers.map((appUser) => (
                <Chip
                  key={appUser.appId}
                  label={appUser.appId}
                  size="small"
```

```
                    color={appColors[appUser.appId] as any || 'default'}
                    variant={appUser.status === 'active' ? 'filled' : 'outlined'}
                  />
                ))}
              </Box>
            </TableCell>
            <TableCell>
              {new Date(user.firstSeenAt).toLocaleDateString()}
            </TableCell>
            <TableCell>
              {new Date(user.lastActivityAt).toLocaleString()}
            </TableCell>
            <TableCell>
              <Tooltip title="View Details">
                <IconButton onClick={() => setSelectedUser(user)}>
                  <Visibility />
                </IconButton>
              </Tooltip>
            </TableCell>
          </TableRow>
        ))}
      </TableBody>
    </Table>
  </Card>

  {/* User Detail Dialog */}
  <Dialog
    open={!!selectedUser}
    onClose={() => setSelectedUser(null)}
    maxWidth="md"
    fullWidth
  >
    <DialogTitle>
      <Box sx={{ display: 'flex', alignItems: 'center', gap: 1 }}>
        <Apps />
        User Details: {selectedUser?.email}
      </Box>
    </DialogTitle>
    <DialogContent>
      {selectedUser && (
        <Table>
          <TableHead>
            <TableRow>
              <TableCell>Application</TableCell>
              <TableCell>Display Name</TableCell>
              <TableCell>Status</TableCell>
              <TableCell>Chats</TableCell>
              <TableCell>Last Login</TableCell>
```

```
                </TableRow>
              </TableHead>
              <TableBody>
                {selectedUser.appUsers.map((appUser) => (
                  <TableRow key={appUser.appId}>
                    <TableCell>
                      <Chip
                        label={appUser.appId}
                        color={appColors[appUser.appId] as any || 'default'}
                      />
                    </TableCell>
                    <TableCell>{appUser.displayName}</TableCell>
                    <TableCell>
                      <Chip
                        label={appUser.status}
                        size="small"
                        color={appUser.status === 'active' ? 'success' : 'default'}
                      />
                    </TableCell>
                    <TableCell>{appUser.chatCount}</TableCell>
                    <TableCell>
                      {appUser.lastLoginAt
                        ? new Date(appUser.lastLoginAt).toLocaleString()
                        : 'Never'}
                    </TableCell>
                  </TableRow>
                ))}
              </TableBody>
            </Table>
          )}
        </DialogContent>
      </Dialog>
    </Box>
  );
}
```

40.7 MIGRATION GUIDE

Step-by-Step Migration for Existing Deployments

```
# Migration Guide: v4.5.0 → v4.6.0 (Application-Level Isolation)

## Overview

This migration adds app_id isolation to all user-facing tables. Existing data
will be migrated with a default app_id based on the primary application.
```

```
## Pre-Migration Checklist

- [ ] Backup database
- [ ] Schedule maintenance window (migration takes ~10-30 minutes depending on data size)
- [ ] Notify users of maintenance
- [ ] Verify AWS Lambda permissions for Cognito triggers

## Migration Steps

### 1. Apply Database Migration

```bash
# Apply the new migration
cd packages/infrastructure
pnpm run migrate:up

# Verify migration applied
psql $DATABASE_URL -c "SELECT * FROM schema_migrations WHERE version = '040';"
```

2. Migrate Existing Users to app_users

```
-- Migration script to copy existing users to app_users
-- Run this ONCE after applying 040_app_level_isolation.sql

INSERT INTO app_users (
    tenant_id,
    app_id,
    base_user_id,
    cognito_sub,
    email,
    email_verified,
    first_name,
    last_name,
    display_name,
    avatar_url,
    preferences,
    timezone,
    locale,
    status,
    last_login_at,
    created_at,
    updated_at
)
SELECT
    u.tenant_id,
    t.app_id,  -- Use tenant's app_id
    u.id,      -- Link to base user
    u.cognito_sub,
```
```

```
    u.email,
    u.email_verified,
    u.first_name,
    u.last_name,
    u.display_name,
    u.avatar_url,
    u.preferences,
    u.timezone,
    u.locale,
    u.status,
    u.last_login_at,
    u.created_at,
    u.updated_at
FROM users u
JOIN tenants t ON u.tenant_id = t.id
WHERE NOT EXISTS (
    SELECT 1 FROM app_users au
    WHERE au.tenant_id = u.tenant_id
    AND au.app_id = t.app_id
    AND au.email = u.email
);
```

3. Update Existing Data with app_id

```
-- Set app_id on existing data based on tenant's primary app
UPDATE thinktank_chats SET app_id = 'thinktank' WHERE app_id IS NULL OR app_id = '';
UPDATE thinktank_messages SET app_id = 'thinktank' WHERE app_id IS NULL OR app_id = '';
UPDATE concurrent_sessions SET app_id = 'thinktank' WHERE app_id IS NULL OR app_id = '';
-- ... repeat for all tables
```

4. Deploy New Cognito Pools

```
# Deploy per-app Cognito pools
cd packages/infrastructure
cdk deploy AppCognitoThinkTankStack
cdk deploy AppCognitoThinkTankStack
cdk deploy AppCognitoLaunchBoardStack
cdk deploy AppCognitoAlwaysMeStack
cdk deploy AppCognitoMechanicalMakerStack
```

5. Update API Gateway

```
# Deploy updated API with per-app routes
cdk deploy APIStack
```

6. Verify Migration

```
-- Verify app_users populated
SELECT app_id, COUNT(*) FROM app_users GROUP BY app_id;
```

```sql
-- Verify RLS policies working
SET app.current_tenant_id = 'your-tenant-id';
SET app.current_app_id = 'thinktank';
SELECT COUNT(*) FROM thinktank_chats; -- Should only show Think Tank chats

SET app.current_app_id = 'thinktank';
SELECT COUNT(*) FROM thinktank_chats; -- Should show 0 (no Think Tank chats in thinktank_chats
```

Rollback Procedure

If issues occur, you can temporarily disable app isolation:

```sql
-- Emergency rollback: Remove app_id from RLS policies
CREATE OR REPLACE FUNCTION current_app_id() RETURNS VARCHAR(50) AS $$
BEGIN
    RETURN NULL;  -- Returns NULL, effectively disabling app filtering
END;
$$ LANGUAGE plpgsql SECURITY DEFINER STABLE;
```

Post-Migration

- ☐ Monitor error rates in CloudWatch
- ☐ Verify cross-app isolation working (test with same email in different apps)
- ☐ Update client applications with new Cognito pool IDs
- ☐ Train support staff on cross-app user correlation dashboard

---

40.8 TESTING

Isolation Test Suite

```typescript
// tests/isolation/app-isolation.test.ts

import { describe, it, expect, beforeAll, afterAll } from '@jest/globals';
import { getDbClient, closePool } from '../utils/db';

describe('Application-Level Data Isolation', () => {
  let db: any;

  beforeAll(async () => {
    db = await getDbClient();
  });

  afterAll(async () => {
    await closePool();


```

```
});

describe('RLS Policies', () => {
  it('should isolate data between apps in same tenant', async () => {
    const tenantId = 'test-tenant-001';

    // Create test chats in different apps
    await db.query(`SET app.current_tenant_id = $1`, [tenantId]);

    // Insert chat in Think Tank
    await db.query(`SET app.current_app_id = 'thinktank'`);
    await db.query(`
      INSERT INTO thinktank_chats (id, tenant_id, app_id, user_id, title)
      VALUES ('chat-thinktank-1', $1, 'thinktank', 'user-1', 'Think Tank Chat')
    `, [tenantId]);

    // Insert chat in Think Tank
    await db.query(`SET app.current_app_id = 'thinktank'`);
    await db.query(`
      INSERT INTO thinktank_chats (id, tenant_id, app_id, user_id, title)
      VALUES ('chat-tt-1', $1, 'thinktank', 'user-1', 'Think Tank Chat')
    `, [tenantId]);

    // Query from Think Tank context
    await db.query(`SET app.current_app_id = 'thinktank'`);
    const thinktankChats = await db.query(`SELECT * FROM thinktank_chats`);

    expect(thinktankChats.rows).toHaveLength(1);
    expect(thinktankChats.rows[0].title).toBe('Think Tank Chat');

    // Query from Think Tank context
    await db.query(`SET app.current_app_id = 'thinktank'`);
    const ttChats = await db.query(`SELECT * FROM thinktank_chats`);

    expect(ttChats.rows).toHaveLength(1);
    expect(ttChats.rows[0].title).toBe('Think Tank Chat');
  });

  it('should allow admin cross-app view', async () => {
    const tenantId = 'test-tenant-001';

    await db.query(`SET app.current_tenant_id = $1`, [tenantId]);
    await db.query(`SET app.is_admin = 'true'`);

    // Admin should see all chats
    const allChats = await db.query(`
      SELECT * FROM thinktank_chats WHERE tenant_id = $1
    `, [tenantId]);
```

```
      expect(allChats.rows.length).toBeGreaterThanOrEqual(2);
    });

    it('should prevent cross-tenant access', async () => {
      await db.query(`SET app.current_tenant_id = 'other-tenant'`);
      await db.query(`SET app.current_app_id = 'thinktank'`);

      const otherTenantChats = await db.query(`SELECT * FROM thinktank_chats`);

      expect(otherTenantChats.rows).toHaveLength(0);
    });
  });

  describe('App Users', () => {
    it('should create separate app_users for same email', async () => {
      const tenantId = 'test-tenant-002';
      const email = 'alice@test.com';

      // Create app_user in Think Tank
      await db.query(`
        INSERT INTO app_users (tenant_id, app_id, cognito_sub, email, status)
        VALUES ($1, 'thinktank', 'sub-thinktank', $2, 'active')
      `, [tenantId, email]);

      // Create app_user in Think Tank
      await db.query(`
        INSERT INTO app_users (tenant_id, app_id, cognito_sub, email, status)
        VALUES ($1, 'thinktank', 'sub-thinktank', $2, 'active')
      `, [tenantId, email]);

      // Verify both exist
      const thinktankUser = await db.query(`
        SELECT * FROM app_users
        WHERE tenant_id = $1 AND app_id = 'thinktank' AND email = $2
      `, [tenantId, email]);

      const ttUser = await db.query(`
        SELECT * FROM app_users
        WHERE tenant_id = $1 AND app_id = 'thinktank' AND email = $2
      `, [tenantId, email]);

      expect(thinktankUser.rows).toHaveLength(1);
      expect(ttUser.rows).toHaveLength(1);
      expect(thinktankUser.rows[0].id).not.toBe(ttUser.rows[0].id);
    });
  });
});
```

40.9 SUMMARY

What v4.6.0 Achieves

Before (v4.5.0)	After (v4.6.0)
Users isolated by tenant_id only	Users isolated by tenant_id + app_id
Same email = same user across all apps	Same email = separate identity per app
Think Tank data visible to client apps	Think Tank completely isolated
Single Cognito pool per tenant	Separate Cognito pool per app
RLS filters by tenant only	RLS filters by tenant AND app
No cross-app admin view	Admin dashboard for cross-app correlation

Key Files Changed/Added

```
packages/
  infrastructure/
      lib/stacks/
          app-cognito.stack.ts           # NEW: Per-app Cognito
      migrations/
          040_app_level_isolation.sql     # NEW: Database changes
  lambda/
      cognito/
          pre-signup.ts                   # NEW: Sign-up validation
          post-confirmation.ts            # NEW: App user creation
          pre-token-generation.ts         # NEW: Add app claims
      shared/
          auth/
              context.ts                  # UPDATED: App context
apps/
  admin-dashboard/
      src/app/(dashboard)/users/
          cross-app/page.tsx              # NEW: Admin view
tests/
  isolation/
      app-isolation.test.ts               # NEW: Isolation tests
```

ðŸš€ HOW TO USE THIS PROMPT

This is the **definitive, fully deduplicated** implementation prompt for RADIANT v2.2.0. All duplicate type definitions have been removed and replaced with a single source of truth.

For Windsurf IDE:

1. Open Windsurf IDE
2. Create a new folder: `radiant`

3. Paste this entire prompt into Cascade (Claude Opus 4.5)
4. The AI will implement systematically, section by section
5. Review and commit incrementally

Implementation Order (Dependency Graph):

```
â"Œâ"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
â",  SECTION 0: Shared Types (@radiant/shared)                    â",
â",  â""â"€â"€ Must be implemented FIRST - all other sections import it  â",
â""â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
                              â",
                              â–¼
â"Œâ"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
â",  SECTION 1: Foundation & Swift App                             â",
â",  â""â"€â"€ Depends on: Section 0                                       â",
â",  â""â"€â"€ Creates: Monorepo structure, Swift macOS app               â",
â""â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
                              â",
                              â–¼
â"Œâ"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
â",  SECTION 2: CDK Infrastructure                                 â",
â",  â""â"€â"€ Depends on: Sections 0, 1                                   â",
â",  â""â"€â"€ Creates: VPC, Aurora, DynamoDB, S3, KMS, WAF               â",
â""â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
                              â",
                              â–¼
â"Œâ"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
â",  SECTION 3: CDK AI & API Stacks                                â",
â",  â""â"€â"€ Depends on: Sections 0-2                                    â",
â",  â""â"€â"€ Creates: Cognito, LiteLLM, API Gateway, AppSync            â",
â""â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
                              â",
                              â–¼
â"Œâ"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
â",  SECTION 4: Lambda Functions - Core                             â",
â",  â""â"€â"€ Depends on: Sections 0-3                                    â",
â",  â""â"€â"€ Creates: Router, Chat, Models, Providers, PHI handlers     â",
â""â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
                              â",
                              â–¼
â"Œâ"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
â",  SECTION 5: Lambda Functions - Admin & Billing                  â",
â",  â""â"€â"€ Depends on: Sections 0-4                                    â",
â",  â""â"€â"€ Creates: Invitations, Approvals, Metering, Billing         â",
â""â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
                              â",
                              â–¼
â"Œâ"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
```

```
â",  SECTION 6: Self-Hosted Models & Services                      â",
â",  â""â"€â"€ Depends on: Sections 0-5                                   â",
â",  â""â"€â"€ Creates: 30+ SageMaker models, Thermal state management    â",
â""â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
                              â",
                              â–¼
â"Œâ"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
â",  SECTION 7: Database Schema (CANONICAL)                        â",
â",  â""â"€â"€ Depends on: Sections 0-6                                   â",
â",  â""â"€â"€ Creates: Complete PostgreSQL schema, all migrations        â",
â""â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
                              â",
                              â–¼
â"Œâ"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
â",  SECTION 8: Admin Web Dashboard                                 â",
â",  â""â"€â"€ Depends on: Sections 0-7                                   â",
â",  â""â"€â"€ Creates: Next.js 14 dashboard, all management UIs         â",
â""â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
                              â",
                              â–¼
â"Œâ"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
â",  SECTION 9: Assembly & Deployment                               â",
â",  â""â"€â"€ Depends on: ALL sections                                    â",
â",  â""â"€â"€ Provides: Verification, testing, troubleshooting          â",
â""â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€â"€
```

ðŸ“‹ CONFIGURATION - REPLACE THESE PLACEHOLDERS

Before deployment, find and replace these placeholders:

Placeholder	Description	Your Value
‘YOUR_DOMAIN.com‘	Your base domain	e.g., ‘zynapses.com‘
‘YOUR_APP_ID‘	Application identifier	e.g., ‘thinktank‘
‘YOUR_AWS_ACCOUNT_ID‘	12-digit AWS account	e.g., ‘123456789012‘
‘YOUR_AWS_REGION‘	Primary AWS region	e.g., ‘us-east-1‘
‘YOUR_ORG_IDENTIFIER‘	Bundle ID prefix	e.g., ‘com.yourcompany‘

ðŸ“Š CANONICAL DATABASE TABLES

Use these table names consistently (not the alternatives):

Canonical Name	Do NOT Use
`tenants`	-

Canonical Name	Do NOT Use
users	-
administrators	admin_users
invitations	admin_invitations
approval_requests	promotions, admin_approvals
providers	-
models	ai_models (legacy)
ai_models	- (v4.1.0 orchestration registry)
model_licenses	- (v4.1.0 license tracking)
model_dependencies	- (v4.1.0 dependency tracking)
workflow_definitions	- (v4.1.0 workflow DAGs)
workflow_tasks	- (v4.1.0 workflow steps)
workflow_parameters	- (v4.1.0 configurable params)
workflow_executions	- (v4.1.0 execution tracking)
task_executions	- (v4.1.0 task-level logs)
service_definitions	- (v4.1.0 mid-level services)
orchestration_audit_log	- (v4.1.0 change audit)
unified_model_registry	- (v4.2.0 combined view - ALL models)
registry_sync_log	- (v4.2.0 sync history)
provider_health	- (v4.2.0 health monitoring)
self_hosted_models	- (v4.2.0 SageMaker models catalog)
execution_manifests	- (v4.3.0 full execution provenance)
feedback_explicit	- (v4.3.0 thumbs up/down + categories)
feedback_implicit	- (v4.3.0 regenerate, copy, abandon signals)
feedback_voice	- (v4.3.0 voice feedback with transcription)
neural_model_scores	- (v4.3.0 learned model effectiveness)
neural_routing_recommendations	- (v4.3.0 Brain advice from Neural Engine)
user_trust_scores	- (v4.3.0 anti-gaming trust levels)
ab_experiments	- (v4.3.0 A/B testing experiments)
ab_experiment_assignments	- (v4.3.0 user experiment assignments)
localization_languages	- (v4.7.0 supported languages)
localization_registry	- (v4.7.0 all translatable strings)
localization_translations	- (v4.7.0 per-language translations)
localization_audit_log	- (v4.7.0 translation change history)
configuration_categories	- (v4.8.0 config categories)
system_configuration	- (v4.8.0 global config parameters)
tenant_configuration_overrides	- (v4.8.0 per-tenant config overrides)
configuration_audit_log	- (v4.8.0 config change history)
configuration_cache_invalidation	- (v4.8.0 cache invalidation queue)
usage_events	usage_records
invoices	-
audit_logs	-