# Backend Architecture

## Supabase RLS (Row-Level Security)

**Core security layer**. All data access is controlled at the database level via RLS policies.

### RLS Helper Functions

-- Get current user's municipality  
CREATE OR REPLACE FUNCTION auth.user\_municipality\_id()  
RETURNS uuid  
LANGUAGE sql  
STABLE  
AS $$  
 SELECT municipality\_id FROM public.users WHERE id = auth.uid();  
$$;  
  
-- Get current user's department  
CREATE OR REPLACE FUNCTION auth.user\_department\_id()  
RETURNS uuid  
LANGUAGE sql  
STABLE  
AS $$  
 SELECT department\_id FROM public.users WHERE id = auth.uid();  
$$;  
  
-- Get current user's role  
CREATE OR REPLACE FUNCTION auth.user\_role()  
RETURNS text  
LANGUAGE sql  
STABLE  
AS $$  
 SELECT role FROM public.users WHERE id = auth.uid();  
$$;  
  
-- Check if user is admin or city manager  
CREATE OR REPLACE FUNCTION auth.is\_admin\_or\_manager()  
RETURNS boolean  
LANGUAGE sql  
STABLE  
AS $$  
 SELECT role IN ('admin', 'city\_manager') FROM public.users WHERE id = auth.uid();  
$$;

### RLS Policy Examples

**Strategic Plans - Department Scoped**:

-- Department users can view their own plans  
CREATE POLICY "Users can view own department plans"  
ON strategic\_plans  
FOR SELECT  
USING (  
 department\_id = auth.user\_department\_id()  
 OR auth.is\_admin\_or\_manager()  
);  
  
-- Department directors can create plans  
CREATE POLICY "Directors can create plans"  
ON strategic\_plans  
FOR INSERT  
WITH CHECK (  
 department\_id = auth.user\_department\_id()  
 AND auth.user\_role() IN ('admin', 'department\_director')  
);  
  
-- Department directors can update their drafts  
CREATE POLICY "Directors can update own drafts"  
ON strategic\_plans  
FOR UPDATE  
USING (  
 department\_id = auth.user\_department\_id()  
 AND status IN ('draft', 'under\_review')  
 AND auth.user\_role() IN ('admin', 'department\_director')  
);  
  
-- City Manager can approve plans  
CREATE POLICY "City Manager can approve plans"  
ON strategic\_plans  
FOR UPDATE  
USING (auth.user\_role() = 'city\_manager')  
WITH CHECK (  
 status IN ('approved', 'active')  
 AND auth.user\_role() = 'city\_manager'  
);

**Initiatives - Cross-Department Collaboration**:

-- Users can view initiatives from their department or collaborating departments  
CREATE POLICY "Users can view accessible initiatives"  
ON initiatives  
FOR SELECT  
USING (  
 lead\_department\_id = auth.user\_department\_id()  
 OR EXISTS (  
 SELECT 1 FROM initiative\_collaborators  
 WHERE initiative\_id = initiatives.id  
 AND department\_id = auth.user\_department\_id()  
 )  
 OR auth.is\_admin\_or\_manager()  
);

**Comments - Universal Access**:

-- Any authenticated user can view comments on entities they can access  
CREATE POLICY "Users can view comments"  
ON comments  
FOR SELECT  
USING (  
 CASE entity\_type  
 WHEN 'strategic\_plan' THEN EXISTS (  
 SELECT 1 FROM strategic\_plans  
 WHERE id = comments.entity\_id  
 AND (department\_id = auth.user\_department\_id() OR auth.is\_admin\_or\_manager())  
 )  
 WHEN 'initiative' THEN EXISTS (  
 SELECT 1 FROM initiatives  
 WHERE id = comments.entity\_id  
 AND (lead\_department\_id = auth.user\_department\_id() OR auth.is\_admin\_or\_manager())  
 )  
 ELSE false  
 END  
);  
  
-- Users can create comments  
CREATE POLICY "Authenticated users can create comments"  
ON comments  
FOR INSERT  
WITH CHECK (auth.uid() IS NOT NULL);

## Database Triggers

**Audit Log Trigger**:

CREATE OR REPLACE FUNCTION audit\_trigger\_function()  
RETURNS TRIGGER  
LANGUAGE plpgsql  
AS $$  
BEGIN  
 IF (TG\_OP = 'DELETE') THEN  
 INSERT INTO audit\_logs (  
 table\_name,  
 record\_id,  
 action,  
 old\_values,  
 changed\_by,  
 changed\_at  
 ) VALUES (  
 TG\_TABLE\_NAME,  
 OLD.id,  
 'delete',  
 to\_jsonb(OLD),  
 auth.uid(),  
 now()  
 );  
 RETURN OLD;  
 ELSIF (TG\_OP = 'UPDATE') THEN  
 INSERT INTO audit\_logs (  
 table\_name,  
 record\_id,  
 action,  
 old\_values,  
 new\_values,  
 changed\_by,  
 changed\_at  
 ) VALUES (  
 TG\_TABLE\_NAME,  
 NEW.id,  
 'update',  
 to\_jsonb(OLD),  
 to\_jsonb(NEW),  
 auth.uid(),  
 now()  
 );  
 RETURN NEW;  
 ELSIF (TG\_OP = 'INSERT') THEN  
 INSERT INTO audit\_logs (  
 table\_name,  
 record\_id,  
 action,  
 new\_values,  
 changed\_by,  
 changed\_at  
 ) VALUES (  
 TG\_TABLE\_NAME,  
 NEW.id,  
 'insert',  
 to\_jsonb(NEW),  
 auth.uid(),  
 now()  
 );  
 RETURN NEW;  
 END IF;  
END;  
$$;  
  
-- Apply to strategic\_plans  
CREATE TRIGGER strategic\_plans\_audit  
AFTER INSERT OR UPDATE OR DELETE ON strategic\_plans  
FOR EACH ROW EXECUTE FUNCTION audit\_trigger\_function();  
  
-- Apply to initiatives  
CREATE TRIGGER initiatives\_audit  
AFTER INSERT OR UPDATE OR DELETE ON initiatives  
FOR EACH ROW EXECUTE FUNCTION audit\_trigger\_function();

## Database Indexes

**Performance-critical indexes**:

-- Strategic Plans  
CREATE INDEX idx\_strategic\_plans\_department\_status  
ON strategic\_plans(department\_id, status);  
  
CREATE INDEX idx\_strategic\_plans\_fiscal\_year  
ON strategic\_plans(start\_fiscal\_year\_id);  
  
-- Initiatives  
CREATE INDEX idx\_initiatives\_goal  
ON initiatives(strategic\_goal\_id);  
  
CREATE INDEX idx\_initiatives\_priority\_rank  
ON initiatives(priority\_level, rank\_within\_priority);  
  
CREATE INDEX idx\_initiatives\_status  
ON initiatives(status);  
  
-- Initiative Budgets  
CREATE INDEX idx\_initiative\_budgets\_fiscal\_year  
ON initiative\_budgets(fiscal\_year\_id, funding\_source);  
  
-- Comments  
CREATE INDEX idx\_comments\_entity  
ON comments(entity\_type, entity\_id);  
  
-- Audit Logs  
CREATE INDEX idx\_audit\_logs\_table\_record  
ON audit\_logs(table\_name, record\_id);  
  
CREATE INDEX idx\_audit\_logs\_changed\_at  
ON audit\_logs(changed\_at DESC);