**mplementationsplan för Alternativ 2: Databasprocedurer för stock-hantering**

Här är en detaljerad guide för att implementera stock-hantering med databasprocedurer (Alternativ 2), som du kan spara för framtida implementation:

**Steg 1: Skapa databasprocedur för stock-uppdatering**

Skapa en ny migrationsfil i supabase/migrations/ mappen, t.ex. 20240412000000\_stock\_management\_procedures.sql:

sql

Apply to route.ts

*-- Skapa en funktion för att uppdatera produktstock*

CREATE OR REPLACE FUNCTION update\_product\_stock(

  p\_product\_id UUID,

  p\_quantity INTEGER DEFAULT 1,

  p\_check\_stock BOOLEAN DEFAULT TRUE

)

RETURNS BOOLEAN AS $$

DECLARE

  v\_current\_stock INTEGER;

  v\_success BOOLEAN := FALSE;

  v\_product\_exists BOOLEAN;

BEGIN

*-- Kontrollera att produkten existerar*

  SELECT EXISTS(SELECT 1 FROM products WHERE id = p\_product\_id) INTO v\_product\_exists;

  IF NOT v\_product\_exists THEN

    RAISE WARNING 'Product with ID % does not exist', p\_product\_id;

    RETURN FALSE;

  END IF;

*-- Hämta nuvarande lagerstatus*

  SELECT stock\_quantity INTO v\_current\_stock

  FROM products

  WHERE id = p\_product\_id;

*-- Om stock-kontroll är aktiverad, säkerställ att tillräckligt med stock finns*

  IF p\_check\_stock AND (v\_current\_stock IS NULL OR v\_current\_stock < p\_quantity) THEN

    RAISE WARNING 'Insufficient stock for product %. Required: %, Available: %',

                 p\_product\_id, p\_quantity, v\_current\_stock;

    RETURN FALSE;

  END IF;

*-- Uppdatera produkten*

  UPDATE products

  SET

    stock\_quantity = CASE

      WHEN stock\_quantity IS NULL THEN NULL

      ELSE stock\_quantity - p\_quantity

    END,

    in\_stock = CASE

      WHEN stock\_quantity IS NULL THEN TRUE *-- Om NULL, anta obegränsad stock*

      WHEN stock\_quantity - p\_quantity > 0 THEN TRUE

      ELSE FALSE

    END,

    updated\_at = NOW()

  WHERE id = p\_product\_id;

  GET DIAGNOSTICS v\_success = ROW\_COUNT;

*-- Logga för spårbarhet*

  INSERT INTO system\_logs (level, message, metadata)

  VALUES ('INFO', 'Product stock updated', jsonb\_build\_object(

    'product\_id', p\_product\_id,

    'quantity', p\_quantity,

    'previous\_stock', v\_current\_stock,

    'new\_stock', CASE WHEN v\_current\_stock IS NULL THEN NULL ELSE v\_current\_stock - p\_quantity END,

    'success', v\_success > 0,

    'timestamp', NOW()

  ));

  RETURN v\_success > 0;

END;

$$ LANGUAGE plpgsql;

*-- Skapa en trigger-funktion för att uppdatera stock automatiskt när en order skapas*

CREATE OR REPLACE FUNCTION trigger\_update\_stock\_on\_order()

RETURNS TRIGGER AS $$

BEGIN

*-- Om detta är en ny art order, uppdatera stock*

  IF (TG\_OP = 'INSERT' AND NEW.product\_id IS NOT NULL) THEN

    PERFORM update\_product\_stock(NEW.product\_id, 1);

  END IF;

  RETURN NEW;

END;

$$ LANGUAGE plpgsql;

*-- Skapa trigger på art\_orders-tabellen*

DROP TRIGGER IF EXISTS art\_orders\_update\_stock ON art\_orders;

CREATE TRIGGER art\_orders\_update\_stock

AFTER INSERT ON art\_orders

FOR EACH ROW

EXECUTE FUNCTION trigger\_update\_stock\_on\_order();

*-- Uppdatera befintliga callback-procedurer för att använda denna funktion istället för direkt stock-uppdatering*

*-- Detta kräver modifiering av din existerande swish\_callback\_procedure.sql*

**Steg 2: Uppdatera swish\_callback\_procedure.sql**

Öppna din befintliga migration för swish\_callback (supabase/migrations/20240320000003\_swish\_callback\_procedure.sql eller liknande), och ersätt stock-uppdateringsdelen:

sql

Apply to route.ts

*-- Ersätt denna del*

UPDATE products

SET stock = stock - 1

WHERE id = v\_product\_id AND stock > 0;

*-- Med denna anrop:*

PERFORM update\_product\_stock(v\_product\_id, 1);

**Steg 3: Skapa eller uppdatera system\_logs-tabell**

Om du inte redan har en logging-tabell i databasen:

sql

Apply to route.ts

*-- Skapa system\_logs-tabell*

CREATE TABLE IF NOT EXISTS system\_logs (

  id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

  level TEXT NOT NULL,

  message TEXT NOT NULL,

  metadata JSONB,

  created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

*-- Skapa index för snabbare sökningar*

CREATE INDEX IF NOT EXISTS system\_logs\_level\_idx ON system\_logs(level);

CREATE INDEX IF NOT EXISTS system\_logs\_created\_at\_idx ON system\_logs(created\_at);

**Steg 4: Uppdatera koden för invoice-betalningar**

I src/app/api/payments/invoice/create/route.ts, ersätt den manuella stock-uppdateringen med ett anrop till databasproceduren:

javascript

Apply to route.ts

*// Efter att ha skapat art\_orders-record för fakturaköp:*

if (productType === PRODUCT\_TYPES.ART\_PRODUCT) {

  try {

*// Anropa databasproceduren för stock-uppdatering*

    const { data: stockUpdateResult, error: stockUpdateError } = await supabase

      .rpc('update\_product\_stock', {

        p\_product\_id: productId,

        p\_quantity: 1, *// Eller något annat värde baserat på din affärslogik*

        p\_check\_stock: true

      });

    if (stockUpdateError || !stockUpdateResult) {

      logWarning(`Failed to update product stock via DB procedure`, {

        requestId,

        error: stockUpdateError,

        productId

      });

    } else {

      logInfo(`Product stock updated successfully via DB procedure`, {

        requestId,

        productId,

        success: stockUpdateResult

      });

    }

  } catch (stockError) {

    logError(`Error calling stock update procedure`, {

      requestId,

      error: stockError instanceof Error ? stockError.message : 'Unknown error'

    });

  }

}

**Steg 5: Uppdatera koden för swish-betalningar**

Om du manuellt hanterar stock i swish-koden också, uppdatera det på samma sätt.

**Steg 6: Skapa admin-funktioner för stock-hantering**

För fullständig hantering, lägg till funktioner för att manuellt justera stock i admin-gränssnittet:

sql

Apply to route.ts

*-- Funktion för att manuellt justera stock (för admin)*

CREATE OR REPLACE FUNCTION admin\_adjust\_product\_stock(

  p\_product\_id UUID,

  p\_new\_quantity INTEGER,

  p\_admin\_user TEXT,

  p\_reason TEXT DEFAULT 'Manual adjustment'

)

RETURNS BOOLEAN AS $$

DECLARE

  v\_previous\_stock INTEGER;

  v\_success BOOLEAN := FALSE;

BEGIN

*-- Hämta nuvarande stock*

  SELECT stock\_quantity INTO v\_previous\_stock

  FROM products

  WHERE id = p\_product\_id;

*-- Uppdatera stock till nytt värde*

  UPDATE products

  SET

    stock\_quantity = p\_new\_quantity,

    in\_stock = CASE WHEN p\_new\_quantity > 0 THEN TRUE ELSE FALSE END,

    updated\_at = NOW()

  WHERE id = p\_product\_id;

  GET DIAGNOSTICS v\_success = ROW\_COUNT;

*-- Logga ändringen*

  INSERT INTO system\_logs (level, message, metadata)

  VALUES ('INFO', 'Admin stock adjustment', jsonb\_build\_object(

    'product\_id', p\_product\_id,

    'previous\_stock', v\_previous\_stock,

    'new\_stock', p\_new\_quantity,

    'admin\_user', p\_admin\_user,

    'reason', p\_reason,

    'timestamp', NOW()

  ));

  RETURN v\_success > 0;

END;

$$ LANGUAGE plpgsql;

**Steg 7: Skapa stock-historikfunktioner (valfritt)**

För att spåra alla stock-ändringar över tid:

sql

Apply to route.ts

*-- Skapa stock-historiktabell*

CREATE TABLE IF NOT EXISTS product\_stock\_history (

  id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

  product\_id UUID NOT NULL REFERENCES products(id),

  previous\_quantity INTEGER,

  new\_quantity INTEGER,

  change\_reason TEXT,

  order\_id UUID,

  user\_id UUID,

  created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

*-- Skapa index*

CREATE INDEX IF NOT EXISTS product\_stock\_history\_product\_id\_idx ON product\_stock\_history(product\_id);

CREATE INDEX IF NOT EXISTS product\_stock\_history\_created\_at\_idx ON product\_stock\_history(created\_at);

*-- Modifiera update\_product\_stock för att logga historia*

*-- (Uppdatera funktionen ovan för att inkludera denna funktionalitet)*

**Steg 8: Automatisk stock-synkronisering (valfritt)**

Om du vill hålla stock synkroniserad mellan olika system:

sql

Apply to route.ts

*-- Skapa schema för stock-synk*

CREATE SCHEMA IF NOT EXISTS stock\_sync;

*-- Skapa tabell för att spåra synkroniseringen*

CREATE TABLE IF NOT EXISTS stock\_sync.sync\_log (

  id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(),

  product\_id UUID NOT NULL REFERENCES products(id),

  external\_system TEXT NOT NULL,

  sync\_status TEXT NOT NULL,

  sync\_message TEXT,

  created\_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()

);

**Steg 9: Skapa underhållsskript**

För regelbundet underhåll av stock-data:

sql

Apply to route.ts

*-- Funktion för att kontrollera stock-konsistens*

CREATE OR REPLACE FUNCTION maintenance\_check\_stock\_consistency()

RETURNS TABLE(

  product\_id UUID,

  product\_title TEXT,

  calculated\_stock INTEGER,

  actual\_stock INTEGER,

  discrepancy INTEGER

) AS $$

BEGIN

  RETURN QUERY

  WITH order\_counts AS (

    SELECT

      p.id AS product\_id,

      p.title AS product\_title,

      p.initial\_stock AS initial\_stock,

      COUNT(ao.id) AS sold\_count,

      p.stock\_quantity AS current\_stock

    FROM products p

    LEFT JOIN art\_orders ao ON p.id = ao.product\_id

    WHERE ao.status != 'cancelled'

    GROUP BY p.id, p.title, p.initial\_stock, p.stock\_quantity

  )

  SELECT

    oc.product\_id,

    oc.product\_title,

    oc.initial\_stock - oc.sold\_count AS calculated\_stock,

    oc.current\_stock,

    oc.current\_stock - (oc.initial\_stock - oc.sold\_count) AS discrepancy

  FROM order\_counts oc

  WHERE oc.current\_stock != (oc.initial\_stock - oc.sold\_count);

END;

$$ LANGUAGE plpgsql;

Detta är en omfattande implementation som täcker alla aspekter av stock-hantering med databasprocedurer. Du kan göra det stegvis, börja med steg 1-4 och sedan lägga till fler funktioner när du behöver dem.