EZY RNPL Code Review Exercise

Time Allocation: 60 minutes (written assessment)
Date: 24-10-2025

Context

EZY RNPL is a financial technology platform for rent-now-pay-later loans in the UAE. The application was initially generated using Lovable (an AI-powered development platform) and now requires production hardening for a regulated financial environment.

Technology Stack:

- Frontend: React 18, TypeScript, Vite, Tailwind CSS
- Backend: Supabase (PostgreSQL, Authentication, Edge Functions)
- Current State: Fully functional prototype with mock data handlers

Your Mission: Assess the provided codebase for security vulnerabilities, architectural issues, and production readiness concerns. This code will handle sensitive financial data, multi-role authentication (Tenants, Landlords, Agents, Admins, Property Managers), and integration with UAE banking systems.

PART 1: Security Vulnerability Assessment (40 minutes)

Files to Review:

1. `src/stores/auth.ts` - Zustand authentication store
2. `src/components/auth/RouteGuard.tsx` - Route protection component
3. `src/types/auth.ts` - User types and mock data
4. `database-schema.md` - Supabase database schema

Your Task:

Conduct a security audit of the authentication and authorization system. Identify vulnerabilities, assess their severity, and provide specific remediation recommendations.

Deliverable:

Create a vulnerability report with the following structure:

```
CRITICAL (Production Blockers)
- [Vulnerability description]
  - Location: [file:line or table/policy]
  - Risk: [What attack is possible?]
  - Remediation: [Specific fix required]
```

```
HIGH (Must Fix Before Launch)
- [Vulnerability description]
  - Location: [file:line or table/policy]
  - Risk: [What attack is possible?]
  - Remediation: [Specific fix required]

MEDIUM (Should Fix)
- [Vulnerability description]
  - Location: [file:line or table/policy]
  - Risk: [What attack is possible?]
  - Remediation: [Specific fix required]
```

Focus Areas:

- Row-Level Security (RLS) policies
- Role-based access control implementation
- Session management
- Development vs. production code paths
- Authentication bypass mechanisms
- Privilege escalation risks
- Audit logging security

PART 2: Mock-to-Real API Transition Strategy (25 minutes)

Files to Review:

1. `src/mocks/handlers/cheques.ts` - MSW mock handlers
2. `packages/domain/src/cheques.ts` - Zod schemas and types
3. `services/api/src/providers/cheque-collection/http-cheque-collection.provider.ts` - Stub implementation
4. `packages/domain/src/providers/chequeCollection.ts` - Provider interface

Context:

The cheque collection feature currently uses Mock Service Worker (MSW) for frontend development. The real system needs to:

- Store cheque requests in Supabase (multi-role access: Landlords create, Agents/PMs manage, Admins audit)
- Upload cheque images to Supabase Storage with OCR processing
- Submit verified requests to an external banking API (UAE-based cheque collection service)

Your Task:
Design a migration strategy from mocks to production. Provide:

1. **Architecture Diagram:** Draw or describe the data flow from UI → Supabase → External Bank API
2. **Supabase Integration Plan:**
    1. Database tables needed (columns, relationships)
    2. RLS policies for multi-role access (who can read/write/update?)
    3. Storage bucket configuration for cheque images
3. **External API Integration:**
    1. Security considerations (API keys, request signing, webhooks)
    2. Error handling strategy
    3. Idempotency and retry logic
4. **Migration Steps:** Numbered list of implementation phases

Deliverable Format:

```
ARCHITECTURE
[Diagram or detailed description]

SUPABASE SCHEMA
Table: cheque_requests
- [column definitions, indexes, constraints]

RLS Policies:
- [policy name]: [rule description]

EXTERNAL API INTEGRATION
- Authentication: [approach]
- Error Handling: [strategy]
- Webhook Security: [approach]

MIGRATION PHASES
1. [Phase description]
2. [Phase description]
...
```

PART 3: TypeScript Type Safety & Error Handling (15 minutes)

Files to Review:

1. `supabase/functions/loan-schedule/index.ts` - Serverless edge function
2. `src/lib/loan-provider.ts` - Frontend service layer

Your Task:
Identify type safety issues, error handling gaps, and production concerns in these files.
Deliverable:

```
TYPE SAFETY ISSUES
1. [Issue description]
   - Location: [file:line]
   - Risk: [What can go wrong?]
   - Fix: [Specific TypeScript improvement]

ERROR HANDLING GAPS
1. [Issue description]
   - Location: [file:line]
   - Risk: [What happens on failure?]
   - Fix: [Improved error handling]

PRODUCTION READINESS CONCERNS
1. [Issue description]
   - Location: [file:line]
   - Risk: [Why is this a problem in production?]
   - Fix: [Recommended improvement]
```

Specific Focus:
- Input validation on edge functions
- Type assertions and unsafe casts
- Missing authentication checks
- PII data logging
- Development vs. production code paths
- Missing error details for debugging

Submission Guidelines

Format: PDF or Google Doc
Length: No strict limit, but focus on quality over quantity
Deadline: Tuesday 28/10/2025 End of Day
Send To: charlie.berbari@ezy.rent and khaled.hawari@ezy.rent
What We're Looking For:
- ✅ Security-first mindset

- ✅ Practical, actionable recommendations
- ✅ Understanding of Supabase/RLS architecture
- ✅ Realistic assessment of AI-generated code quality
- ✅ Clear communication

What Happens Next:
After reviewing your submission, we'll schedule a 30-minute live discussion where we'll:

1. Walk through your findings together
2. Present a live debugging scenario
3. Discuss implementation priorities
4. Answer your questions about the project

Valid Roles

The application supports 5 user roles:

- `TENANT` - Loan applicants
- `LANDLORD` - Property owners receiving disbursements
- `AGENT` - Real estate agents facilitating applications
- `PROPERTY_MANAGER` - Managers overseeing multiple properties
- `ADMIN` - System administrators with full access

Database Schema Documentation

In case the database-schema file does not work:

Create a separate `database-schema.md` file with this content:

markdown

```
# EZY RNPL Database Schema

## Table: profiles

Stores user profile information linked to Supabase Auth.

**Columns:**
- `id` (uuid, primary key)
- `user_id` (uuid, references auth.users, unique, not null)
- `email` (text, not null)
```

```
- `full_name` (text, not null)
- `phone` (text, nullable)
- `emirates_id` (text, not null)
- `uae_pass_tier` (text, not null, default '3')
- `roles` (text[], not null, default ['TENANT'])
- `created_at` (timestamp, not null, default now())
- `updated_at` (timestamp, not null, default now())

**RLS Policies:**
1. "Users can view their own profile" (SELECT)
   - `auth.uid() = user_id`

2. "Admins can view all profiles" (SELECT)
   - `(auth.uid() = user_id) OR ('ADMIN' = ANY (get_current_user_role()))`

3. "Users can insert their own profile" (INSERT)
   - `auth.uid() = user_id`

4. "Users can update their own profile" (UPDATE)
   - `auth.uid() = user_id`

5. "Only admins can update user roles via function" (UPDATE)
   - `('ADMIN' = ANY (get_current_user_role())) AND (user_id <> auth.uid())`

---

## Table: audit_events

Logs all significant system actions for compliance and debugging.

**Columns:**
- `id` (uuid, primary key)
- `user_id` (uuid, nullable)
- `actor_id` (uuid, nullable)
- `action` (text, not null)
- `resource_type` (text, not null)
- `resource_id` (text, nullable)
- `metadata` (jsonb, nullable)
- `ip_address` (inet, nullable)
- `user_agent` (text, nullable)
- `created_at` (timestamp, not null, default now())

**RLS Policies:**
1. "Anyone can insert audit events" (INSERT)
   - `true`

2. "Users can view their own audit events" (SELECT)
   - `(auth.uid() = user_id) OR (auth.uid() = actor_id)`

3. "Admins can view all audit events" (SELECT)
```

```
   - `EXISTS (SELECT 1 FROM profiles p WHERE p.user_id = auth.uid() AND
'ADMIN' = ANY (p.roles))`

---

## Database Functions

### `get_current_user_role()`

Returns the roles array for the currently authenticated user.

```sql
CREATE OR REPLACE FUNCTION public.get_current_user_role()
RETURNS text[]
LANGUAGE sql
STABLE SECURITY DEFINER
SET search_path TO 'public'
AS $function$
  SELECT roles FROM public.profiles WHERE user_id = auth.uid();
$function$
```

Security: SECURITY DEFINER (runs with elevated privileges)

update_user_role(target_user_id uuid, new_role text)
Allows admins to change a user's role.
sql

```
-- Check if current user is admin
IF NOT ('ADMIN' = ANY (get_current_user_role())) THEN
  RAISE EXCEPTION 'Only admins can update user roles';
END IF;

-- Update the user's role
UPDATE public.profiles
SET roles = ARRAY[new_role]::text[]
WHERE user_id = target_user_id;
```

Security: Only callable by admins, logs changes to audit_events
```