Software Design Doc

Here is the Software Design Document (SDD) for the 28HSE Assignment Platform. This document translates the requirements from the PRD into technical specifications for the engineering team.

📐 Software Design Document (SDD)

Project Name: 28HSE Assignment Platform

Version: 1.0

Date: November 24, 2025

Based on PRD Version: 2.1

1. Introduction

1.1 Purpose

The 28HSE platform is a secure web application designed to facilitate the sale of pre-sale condo assignments. The system must enforce strict visibility controls to comply with REDMA/MLS regulations [1][1][1]while managing a credit-based monetization system[2].

1.2 Scope

This design covers the MVP (Minimum Viable Product), including:

- **Web Client: React.js frontend.**
- **Backend API: Node.js/TypeScript REST API.**
- **Database: PostgreSQL (Relational data).**
- **Storage: AWS S3 (Secure document storage).**
- **Integrations: Stripe (Payments), Twilio (MFA), SendGrid (Email).**

2. System Architecture

2.1 High-Level Architecture

The system follows a Monolithic Architecture (modularized) for the MVP to reduce deployment complexity, transitioning to Microservices in Phase 3. It utilizes a standard Client-Server pattern.

- **Client Layer: Single Page Application (SPA) built with React.js.**
- **API Layer: Node.js (NestJS framework) serving REST endpoints.**
- **Data Layer: PostgreSQL for structured data; Redis for caching and session management.**
- **Infrastructure: Hosted on AWS (Canada Central Region) using Docker containers (ECS/Fargate).**

2.2 Component Diagram

```
Code snippet

graph TD
    Client[Web Client (React)] -->|HTTPS/JSON| LB[Load Balancer]
    LB --> API[Node.js API Server]
    API -->|Read/Write| DB[(PostgreSQL)]
    API -->|Cache/Session| Redis[(Redis)]
    API -->|File Storage| S3[AWS S3 (Encrypted)]
    API -->|Payment| Stripe[Stripe API]
    API -->|SMS| Twilio[Twilio API]
```

3. Data Design

3.1 Database Schema (ERD)

The database must support complex relationships between Users, Listings, and Transactions.

Users Table

Stores authentication and role data.

```
SQL

CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    email VARCHAR(255) UNIQUE NOT NULL,
    password_hash VARCHAR NOT NULL,
    role ENUM('HOMEOWNER', 'REALTOR', 'ADMIN') NOT NULL,
    subscription_tier ENUM('FREE', 'BASIC_AGENT', 'PRO_AGENT') DEFAULT 'FREE',
    credits_balance INT DEFAULT 0,
    is_verified_license BOOLEAN DEFAULT FALSE, -- For Realtors [cite: 59]
    mfa_enabled BOOLEAN DEFAULT TRUE
);
```

Listings Table

Stores property data and visibility states.

```
SQL
```

```
CREATE TABLE listings (
    id UUID PRIMARY KEY,
    owner_id UUID REFERENCES users(id),

    -- Property Data
    project_name VARCHAR(255), -- Hidden from public [cite: 27]
    unit_number VARCHAR(50),  -- Hidden from public [cite: 27]
    original_price DECIMAL(12,2),
    asking_price DECIMAL(12,2),

    -- Verification
    verification_status ENUM('UNVERIFIED', 'PENDING', 'VERIFIED_OWNER') DEFAULT 'UNVERIFIED',
    proof_doc_s3_key VARCHAR, -- Path to redacted contract

    -- Lead Pool Logic [cite: 87, 93]
    lead_pool_status ENUM('NOT_IN_POOL', 'ACTIVE_TIER_1', 'ACTIVE_TIER_2', 'CLAIMED') DEFAULT 'NOT_IN_POOL',
    pool_entry_at TIMESTAMP,
    assigned_agent_id UUID REFERENCES users(id), -- The Realtor who claimed it

    created_at TIMESTAMP DEFAULT NOW()
);
```

Transactions Table

Immutable ledger for credit usage.

SQL

```
CREATE TABLE transactions (
    id UUID PRIMARY KEY,
    user_id UUID REFERENCES users(id),
    amount INT NOT NULL, -- Negative for spend, Positive for purchase
```

```
  action_type ENUM('POST_LISTING', 'UNLOCK_CONTACT', 'CLAIM_LEAD', 'PURCHASE_PACK'),
  reference_id UUID, -- Links to listing_id or stripe_charge_id
  created_at TIMESTAMP DEFAULT NOW()
);
```

3.2 Data Storage Strategy (S3)

- **Bucket Policy:** Private, Block Public Access.
- **Encryption:** Server-Side Encryption (SSE-S3).
- **Lifecycle:** Temporary upload folder (auto-delete after 24h if not submitted).
- **Structure:** /contracts/{user_id}/{listing_id}/redacted_agreement.pdf

4. Component Design & Logic

4.1 Authentication & Authorization Module

- **JWT Strategy:** Access Tokens (15 min expiry) + Refresh Tokens (7 day expiry).
- **Role Guards:** Middleware to verify user.role before accessing routes (e.g., only REALTOR can access /lead-pool).
- **MFA:** Intercept login flow; generate 6-digit TOTP, store in Redis (5 min TTL), send via Twilio.

4.2 Listing Visibility Engine

The API must return different data shapes based on the requester.

- **Public (Guest):** Returns ListingTeaserDTO.
  - *Logic:* Masks unit_number, price (returns range), project_name (returns neighborhood).
- **Member (Logged In):** Returns ListingDetailDTO.
  - *Logic:* Returns full fields. Checks if listing.verification_status == 'VERIFIED_OWNER' to display badge.

4.3 Lead Pool State Machine

A scheduled service (Cron) running on the backend.

- **Frequency:** Hourly.
- **Job Logic:**
  1. Query listings where created_at < NOW() - 14 DAYS AND inquiries < 3.
  2. Update lead_pool_status = 'ACTIVE_TIER_1'.
  3. Trigger Push Notification to Users where subscription_tier = 'PRO_AGENT'.
- **Expiration Logic:**
  1. Query listings where lead_pool_status = 'ACTIVE_TIER_1' AND pool_entry_at < NOW() - 24 HOURS.
  2. Update lead_pool_status = 'ACTIVE_TIER_2'.

4.4 Financial/Tax Calculator Module

A utility class utilized during Listing Creation.

- **BC Home Flipping Tax (BCHFT):** [3]
- TypeScript

```
function calculateBCHFT(contractDate: Date, assignmentDate: Date, profit: number): number {
  const daysHeld = differenceInDays(assignmentDate, contractDate);
  if (daysHeld < 365) return profit * 0.20;
  if (daysHeld >= 365 && daysHeld < 730) return calculateSlidingScale(daysHeld, profit);
  return 0;
}
```

5. Interface Design (API Specification)

5.1 Key Endpoints

Method	Endpoint	Access	Description
POST	**/auth/register**	**Public**	**Register new user.**
POST	**/auth/verify-mfa**	**Public**	**Verify SMS code.**
POST	**/listings**	**Member**	**Create listing (deducts credits).**
POST	**/listings/:id/verify-owner**	**Member**	**Upload contract document (S3 signed URL).**
GET	**/listings**	**Public**	**Returns Teaser list (search/filter).**
GET	**/lead-pool**	**Realtor**	**Returns available leads based on Tier.**
POST	**/lead-pool/:id/claim**	**Realtor**	**Atomic Transaction.** Deducts 50 credits, assigns lead.

6. Security & Compliance

6.1 Data Residency

All production data must be stored in the AWS Canada (Central) region to align with Canadian data privacy expectations for financial transactions.

6.2 "Private Sale" Enforcement

To comply with REDMA[4], the API must strictly ensure that project_name and unit_number are never included in the JSON response for public/unauthenticated endpoints. Frontend hiding is insufficient; data must be sanitized at the API layer.

6.3 Transaction Safety (Optimistic Locking)

To prevent two Realtors from claiming the same lead simultaneously:

- **Database Level: Use SELECT ... FOR UPDATE or a version column version_id check during the update query.**
- **Logic: If the update returns 0 rows affected (meaning status changed since read), throw 409 Conflict.**

7. Infrastructure & Deployment

7.1 Tech Stack

- **Language: TypeScript (Strict Mode).**
- **Framework: NestJS (Backend), Next.js (Frontend).**
- **ORM: Prisma or TypeORM.**

7.2 CI/CD Pipeline

- **GitHub Actions:**
  1. **Lint & Test (Unit/Integration).**
  2. **Build Docker Images.**
  3. **Push to AWS ECR.**
  4. **Deploy to AWS ECS (Fargate).**

7.3 Monitoring

- **Logs: AWS CloudWatch.**
- **Metrics: Track "Lead Claim Success Rate" and "Verification Wait Time".**

8. Development Roadmap

- **Sprint 1: Database setup, User Auth, MFA integration.**
- **Sprint 2: Listing CRUD, Visibility Logic (Teaser vs. Full), S3 Integration.**
- **Sprint 3: Admin Dashboard (Verification UI), Tax Calculator Logic.**
- **Sprint 4: Lead Pool State Machine, Credit/Stripe Integration.**
- **Sprint 5: QA, Security Audit, Beta Launch.**