Low Level Design Document

Introduction

This Low Level Design (LLD) document outlines the core implementation details for **PropSecure - Real Estate Property Security Platform**. PropSecure is a secure backend system for managing property listings, user roles, and access logs, featuring Spring Security with JWT, AOP-based logging, async background checks, MySQL persistence, Docker deployment, and an Angular frontend.

---

1. System Components

Component	Technology	Key Responsibilities
API Gateway	Spring Boot	Route requests, JWT validation
Auth Service	Spring Security	User authentication, JWT issuance/validation
Property Service	Spring Boot	CRUD for property listings
User Service	Spring Boot	User/role management
Logging Aspect	Spring AOP	Log access/actions
Background Check	Spring Async	Run background checks asynchronously
Database	MySQL	Persist users, properties, logs
Frontend	Angular	UI for agents/admins
Deployment	Docker	Containerized deployment

---

2. Class/Interface Overview

Class/Interface	Responsibility	Key Methods/Attributes
`UserController`	User endpoints	`register()`, `login()`, `getUser()`
`PropertyController`	Property endpoints	`create()`, `update()`, `list()`
`AuthService`	Auth logic, JWT handling	`authenticate()`, `generateToken()`
`JwtFilter`	JWT validation	`doFilterInternal()`
`LoggingAspect`	AOP logging	`@Around` advice for controllers
`BackgroundCheckService`	Async background checks	`@Async checkUserBackground()`
`UserRepository`	User DB access	`findByEmail()`, `save()`
`PropertyRepository`	Property DB access	`findById()`, `save()`, `findAll()`

Relationships:

- Controllers use Services; Services use Repositories; LoggingAspect intercepts Controllers.

3. Data Structure Overview

Entity	Fields (Type)
User	id (Long), name (String), email (String), password (String, hashed), role (Enum)
Property	id (Long), address (String), ownerId (Long), status (Enum), createdAt (Timestamp)
AccessLog	id (Long), userId (Long), action (String), timestamp (Timestamp)

4. Algorithms/Logic

Authentication Flow (Pseudocode):

```
POST /login:
  user = userRepository.findByEmail(email)
  if (user == null || !passwordMatches(user, inputPassword)) throw AuthExceptior
  token = jwtUtil.generateToken(user)
  return token
```

Async Background Check (Pseudocode):

```
@Async
void checkUserBackground(Long userId) {
  // Call external service, update user status
}
```

AOP Logging (Pseudocode):

```
@Around("controllerMethods()")
logAccess(userId, action, timestamp)
proceed()
```

5. Error Handling

Scenario	Handling Approach
Invalid JWT / Unauthorized	Return 401 Unauthorized, log attempt
Access Denied (Role)	Return 403 Forbidden, log action
Entity Not Found	Return 404 Not Found
Validation Failure	Return 400 Bad Request with error details
DB/Service Unavailable	Return 503 Service Unavailable, retry if needed
Async Task Failure	Log error, notify admin if critical

End of Document