CHƯƠNG 2: KIẾN TRÚC KỸ THUẬT

Chương này cung cấp cái nhìn sâu sắc về mặt kỹ thuật, giúp Developer hiểu cách hệ thống vận hành bên trong để có thể bảo trì và mở rộng.

Trang 1: Công nghệ cốt lõi (Core Stack)

Dự án được xây dựng dựa trên các tiêu chuẩn hiện đại của Java Ecosystem:

1. **Framework chính**: Spring Boot 3.3.0.
2. **Ngôn ngữ**: Java 17 (Cung cấp các tính năng như Record, Text Blocks, Switch Expression).
3. **HTTP Client**: `Spring WebFlux (WebClient)`. Được chọn thay thế cho RestTemplate cũ vì tính linh hoạt và hiệu suất cao hơn.
4. **Resiliency**: `Spring Retry`. Tự động quản lý việc thử lại các yêu cầu thất bại.
5. **Tiện ích**:
   - `Lombok`: Giảm thiểu mã lặp (Getter, Setter, Constructor).
   - `Logback`: Ghi log chi tiết cho quá trình Proxy.
   - `Maven`: Quản lý phụ thuộc và đóng gói JAR.

Trang 2: Cấu hình WebClient & Kết nối

Lớp `WebClientConfig` đóng vai trò quan trọng trong việc thiết lập giao tiếp mạng:

2.1 Xử lý dung lượng file lớn

Mặc định, WebClient giới hạn bộ nhớ đệm cho dữ liệu nhận về. Vì hệ thống xử lý ký các file PDF dung lượng lớn, cấu hình đã được mở rộng lên **16 MB**:

```
ExchangeStrategies strategies = ExchangeStrategies.builder()
    .codecs(configurer -> configurer.defaultCodecs().maxInMemorySize(16 * 1024 * 1024))
    .build();
```

2.2 Quản lý Timeout

Hệ thống sử dụng các tham số cấu hình linh hoạt trong `application.yml` cho từng loại backend:

- `connect-timeout-ms`: Thời gian tối đa để thiết lập kết nối (Mặc định 60s cho Signing).
- `read-timeout-ms`: Thời gian chờ Server phản hồi dữ liệu (Mặc định 60s cho Signing).

Trang 3: Cơ chế Kháng lỗi (Resiliency)

Hệ thống sử dụng cơ chế **Retry-with-Recovery** tại lớp `ExternalApiClient`:

3.1 Cơ chế Retry

Sử dụng Annotation `@Retryable` cho các phương thức gọi API:

- `maxAttempts = 3`: Thử lại tối đa 3 lần nếu xảy ra Exception.
- `backoff = @Backoff(delay = 2000)`: Chờ 2 giây giữa mỗi lần thử lại.

3.2 Cơ chế Recovery (Hồi phục)

Khi cả 3 lần thử lại đều thất bại, phương thức `@Recover` sẽ được kích hoạt để trả về một phản hồi "an toàn" thay vì làm sập ứng dụng:

```
@Recover
public ResponseEntity<?> recoverSigning(Exception e) {
    log.warn("Recovering after signing call failed", e);
    return ResponseFactory.error("Signing service error: " + e.getMessage(), ...);
}
```

Trang 4: Phân tích cấu trúc mã nguồn

Cấu trúc Package được tổ chức theo chuẩn Clean Architecture đơn giản:

Package	Trách nhiệm
`com.example.serviceproxy.controller`	Tiếp nhận HTTP Request từ Client, thực hiện Mapping Header.
`com.example.serviceproxy.service`	Chứa Interface và Implementation điều phối các yêu cầu Proxy.
`com.example.serviceproxy.client`	**Trái tim của hệ thống**, thực hiện các cuộc gọi WebClient và logic Retry.
`com.example.serviceproxy.config`	Thiết lập Security, WebClient và các Bean hệ thống.
`com.example.serviceproxy.dto`	Chứa các đối tượng trao đổi dữ liệu (Request/Response) cho cả Signing và BPMN.
`com.example.serviceproxy.response`	Lớp tiện ích để tạo các phản hồi chuẩn (Success/Error).

Sơ đồ Logic xử lý:

1. `SigningProxyController` nhận request.
2. `SigningProxyServiceImpl` nhận lệnh.
3. `ExternalApiClient` thực thi WebClient (có Retry).
4. Phản hồi được trả ngược lại qua các lớp.