Tổng quan kiến trúc hệ thống

Nội dung chi tiết

1. Tổng quan kiến trúc hệ thống

1.1. Mục tiêu kiến trúc

Kiến trúc backend được thiết kế nhằm:

- Đảm bảo khả năng mở rộng và bảo trì lâu dài
- Tách biệt rõ trách nhiệm giữa các layer
- Chuẩn hóa cách tổ chức code giữa các module
- Hỗ trợ phát triển song song nhiều developer
- Dễ dàng tích hợp các hệ thống ngoài (SSO, Cache, MQ…)

Tất cả các service backend phải tuân thủ kiến trúc chuẩn được định nghĩa trong tài liệu này.

1.2. Mô hình kiến trúc

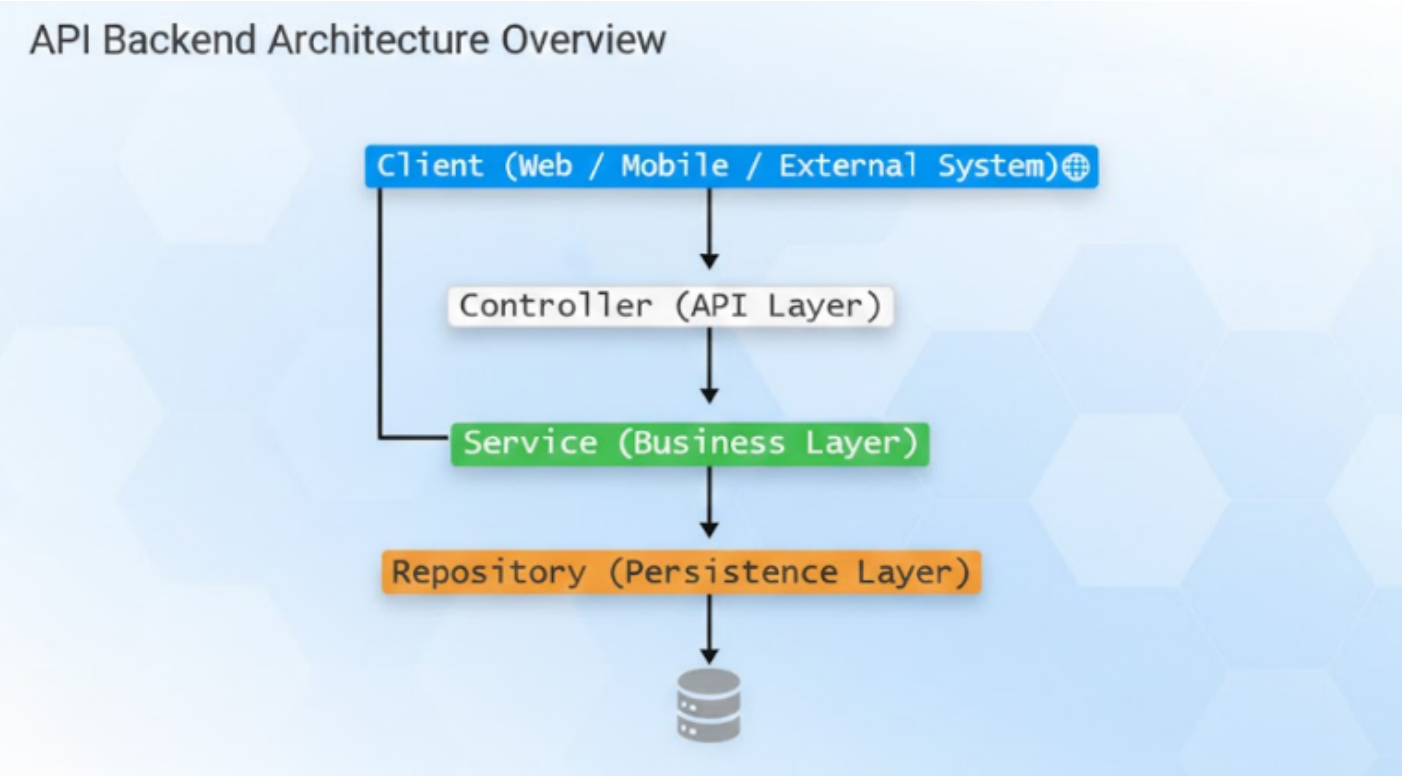
Hệ thống backend sử dụng mô hình:

Modular Monolith – Layered Architecture

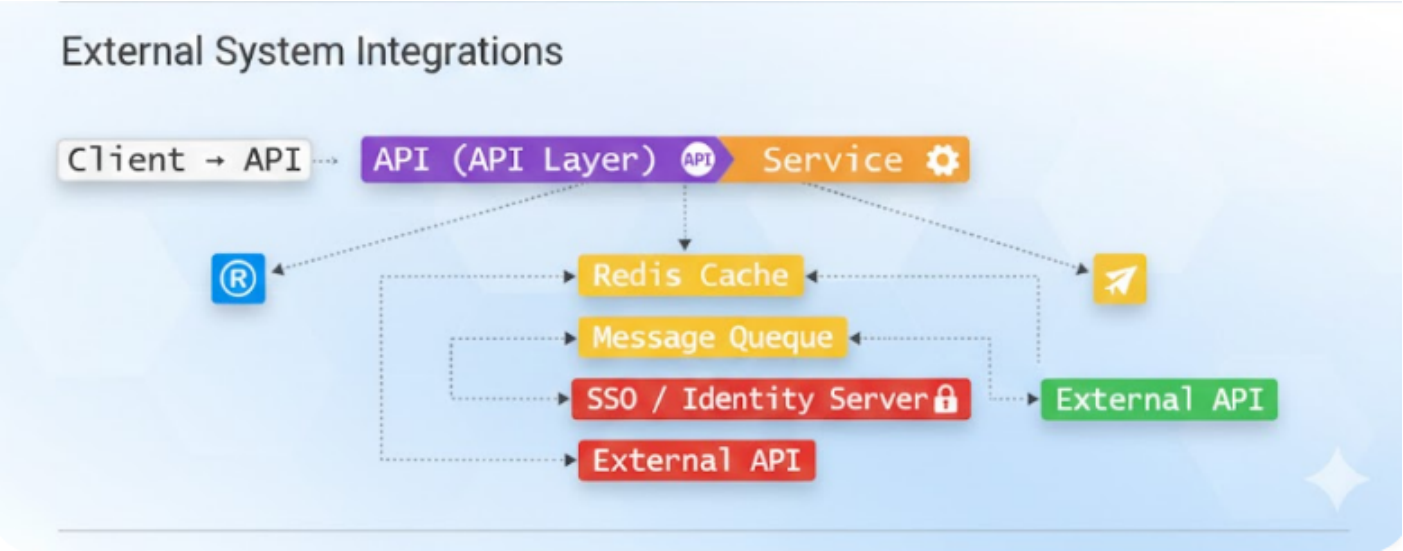
Đặc điểm:

- Một service Spring Boot deploy độc lập
- Bên trong chia module theo domain
- Mỗi module có đầy đủ layer Controller → Service → Repository
- Có thể tách thành microservice khi cần

Sơ đồ tổng thể:



Tích hợp ngoài:



1.3. Quy tắc phân lớp (Layered Architecture)

Mỗi module phải tuân thủ cấu trúc phân lớp sau:

`controller`
`service`

`repository`
`entity`
`dto`
`mapper`
`exception`

1.3.1. Controller Layer (API Layer)

Chức năng:

- Nhận request HTTP
- Validate input DTO
- Gọi service xử lý
- Trả response chuẩn

Quy định:

- Không chứa business logic
- Không truy cập DB trực tiếp
- Không xử lý transaction
- Không map entity

Ví dụ:

```
@RestController
@RequestMapping("/api/users")
public class UserController {

    private final UserService userService;

    @PostMapping
    public ApiResponse<UserResponse> create(@Valid @RequestBody UserRequest req) {
        return ApiResponse.success(userService.create(req));
    }
}
```

1.3.2. Service Layer (Business Layer)

Chức năng:

- Chứa toàn bộ business logic
- Điều phối repository
- Xử lý transaction
- Mapping entity ↔ DTO

Quy định:

- Không nhận HttpServletRequest
- Không trả Entity ra ngoài
- Phải qua DTO/Response
- Transaction đặt tại Service

Ví dụ:

```
@Service
public class UserService {

    @Transactional
    public UserResponse create(UserRequest req) {
        User entity = mapper.toEntity(req);
        repository.save(entity);
        return mapper.toResponse(entity);
    }
}
```

1.3.3. Repository Layer (Persistence Layer)

Chức năng:

- Truy cập DB
- Query dữ liệu
- Mapping ORM

Quy định:

- Không chứa business logic
- Không gọi service
- Chỉ thao tác entity

Ví dụ:

```
public interface UserRepository extends JpaRepository<User, Long> {
}
```

1.3.4. Entity Layer

Chức năng:

- Mapping bảng DB
- Quan hệ ORM

Quy định:

- Không chứa logic nghiệp vụ
- Không trả trực tiếp ra API
- Không dùng cho request/response

1.3.5. DTO Layer

Gồm:

- Request DTO
- Response DTO

Chức năng:

- Trao đổi dữ liệu với client
- Tách biệt entity và API

Quy định:

- Controller chỉ nhận/trả DTO
- Không expose entity

1.3.6. Mapper Layer

Chức năng:

- Chuyển đổi DTO ↔ Entity

Có thể dùng:

- MapStruct
- Manual mapping

Quy định:

- Không đặt trong controller
- Không đặt trong repository

1.4. Quy tắc phụ thuộc (Dependency Direction)

Dependency chỉ được phép đi theo một chiều:

`Controller → Service → Repository → DB`

Không được phép:

- Repository gọi Service
- Service gọi Controller
- Controller gọi Repository trực tiếp
- DTO gọi Repository

Nguyên tắc:

Layer trên được gọi layer dưới
Layer dưới không được gọi layer trên

1.5. Tổ chức module theo domain

Project được tổ chức theo domain thay vì technical layer toàn cục.

Cấu trúc chuẩn:

```
com.company.project
├── common
│   ├── config
│   ├── exception
│   ├── util
│
├── user
│   ├── controller
│   ├── service
│   ├── repository
│   ├── entity
│   ├── dto
│   ├── mapper
│
├── document
│   ├── controller
│   ├── service
│   ├── repository
│   ├── entity
│   ├── dto
│   ├── mapper
```

1.6. Tích hợp hệ thống ngoài

Backend có thể tích hợp các thành phần sau:

- SSO / Identity Server
- Cache (Redis)
- Message Queue
- File Storage
- External API

Quy tắc tích hợp:

- Gọi qua Service
- Không gọi trực tiếp từ Controller
- Tách client class riêng

Ví dụ:

```
integration
├── sso
│   ├── SsoClient
│
├── redis
│   ├── RedisService
│
├── external
│   ├── ExternalApiClient
```

1.7. Nguyên tắc kiến trúc bắt buộc

Tất cả backend service phải tuân thủ:

- Không expose entity ra API
- Không viết business logic trong controller
- Không truy cập DB ngoài repository
- Không phụ thuộc ngược layer
- Transaction đặt tại service
- Mapping qua DTO

Vi phạm kiến trúc được xem là lỗi nghiêm trọng trong review code.

1.8. Khả năng mở rộng Microservice

Kiến trúc hiện tại cho phép tách module thành microservice khi cần:

Ví dụ:

```
user module → user-service
document module → document-service
auth module → auth-service
```

1.9. Xử lý ngoại lệ toàn cục (Exception Handling Strategy)

1.9.1. Mục tiêu

Chuẩn hóa cơ chế xử lý lỗi toàn hệ thống nhằm:

- Đảm bảo response lỗi thống nhất giữa các API
- Ẩn thông tin nội bộ hệ thống
- Dễ dàng log và truy vết lỗi
- Hỗ trợ frontend xử lý lỗi chính xác
- Tránh trả lỗi không kiểm soát (stacktrace, SQL…)

Tất cả API backend phải sử dụng cơ chế xử lý ngoại lệ toàn cục.

1.9.2. Global Exception Handler

Hệ thống sử dụng Global Exception Handler thông qua `@ControllerAdvice`.

Chức năng:

- Bắt tất cả exception phát sinh từ Controller / Service
- Mapping exception → HTTP status
- Trả response lỗi chuẩn
- Ghi log lỗi hệ thống

Quy định:

- Không xử lý lỗi trực tiếp trong Controller
- Không trả stacktrace ra client
- Không trả message DB/SQL
- Không throw Exception chung chung

Ví dụ:

```
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<ApiError> handleNotFound(ResourceNotFoundException ex) {
        return build(HttpStatus.NOT_FOUND, ex.getCode(), ex.getMessage());
    }

    @ExceptionHandler(AccessDeniedException.class)
    public ResponseEntity<ApiError> handleForbidden(AccessDeniedException ex) {
        return build(HttpStatus.FORBIDDEN, "ACCESS_DENIED", "Access denied");
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<ApiError> handleValidation(MethodArgumentNotValidException ex) {
        return build(HttpStatus.BAD_REQUEST, "VALIDATION_ERROR", "Invalid request");
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<ApiError> handleSystem(Exception ex) {
        return build(HttpStatus.INTERNAL_SERVER_ERROR, "SYSTEM_ERROR", "Internal server error");
    }

    private ResponseEntity<ApiError> build(HttpStatus status, String code, String message) {
        ApiError error = ApiError.of(code, message);
        return ResponseEntity.status(status).body(error);
    }
}
```

1.9.3. Mapping Exception → HTTP Status

Chuẩn mapping bắt buộc:

Exception	HTTP Status
ValidationException	400 Bad Request
MethodArgumentNotValid	400 Bad Request
IllegalArgumentException	400 Bad Request
AccessDeniedException	403 Forbidden
AuthenticationException	401 Unauthorized
ResourceNotFoundException	404 Not Found
BusinessException	422 Unprocessable
SystemException	500 Internal Server Error
Exception (default)	500 Internal Server Error

Không được trả:

- 200 với lỗi business
- 500 với lỗi validation
- 500 với not found

1.9.4. Chuẩn format response lỗi

Tất cả API khi lỗi phải trả format thống nhất:

```
{
 "timestamp": "2026-02-23T10:15:30Z",
 "code": "USER_NOT_FOUND",
 "message": "User not found",
 "traceId": "abc123"
}
```

Ý nghĩa:

Field	Mô tả
timestamp	Thời điểm lỗi
code	Mã lỗi hệ thống
message	Thông báo lỗi
traceId	ID truy vết log

Quy định:

- `code` phải ổn định, dùng cho frontend
- `message` có thể hiển thị user
- Không trả stacktrace
- Không trả SQL/exception message nội bộ

1.9.5. Quy tắc định nghĩa Exception

Mỗi loại lỗi phải có class riêng:

```
exception
 ├── ResourceNotFoundException
 ├── ValidationException
 ├── BusinessException
 ├── SystemException
```

Ví dụ:

```
public class ResourceNotFoundException extends RuntimeException {

    private final String code;

    public ResourceNotFoundException(String code, String message) {
        super(message);
        this.code = code;
    }

    public String getCode() {
        return code;
    }
}
```

1.9.6. Quy tắc sử dụng Exception

Service layer phải throw exception domain:

```
public User getUser(Long id) {
    return repository.findById(id)
        .orElseThrow(() -> new ResourceNotFoundException(
            "USER_NOT_FOUND",
            "User not found"
        ));
}
```

Không được:

- throw new Exception()
- throw RuntimeException chung chung
- trả null thay vì lỗi
- trả Optional ra controller

1.9.7. TraceId và logging

Mỗi request phải có traceId để truy vết lỗi:

- Sinh tại filter/interceptor
- Ghi vào MDC log
- Trả về response lỗi

Ví dụ:

```
public class ApiError {

    private String timestamp;
```

```
    private String code;
    private String message;
    private String traceId;

    public static ApiError of(String code, String message) {
        ApiError e = new ApiError();
        e.timestamp = Instant.now().toString();
        e.code = code;
        e.message = message;
        e.traceId = MDC.get("traceId");
        return e;
    }
}
```

1.9.8. Nguyên tắc bắt buộc

Tất cả backend service phải tuân thủ:

- Mọi lỗi phải qua GlobalExceptionHandler
- Không trả exception raw ra client
- Không xử lý lỗi trong controller
- Exception phải có code
- Response lỗi phải đúng format chuẩn
- Mapping đúng HTTP status

Vi phạm quy tắc xử lý lỗi được xem là lỗi nghiêm trọng trong code review