3.Coding convention

3.1. Mục tiêu chương

Chương này quy định chuẩn coding và nguyên tắc thiết kế nhằm:

- Đảm bảo code thống nhất giữa các module và các đơn vị phát triển
- Giúp code dễ đọc, dễ hiểu và dễ bảo trì
- Giảm lỗi phát sinh do cách đặt tên hoặc coding không đồng nhất
- Tăng hiệu quả review và kiểm soát chất lượng code
- Đảm bảo code tuân thủ nguyên lý thiết kế SOLID
- Đảm bảo kiến trúc backend Spring Boot nhất quán toàn hệ thống

---

3.2. Khái niệm / phạm vi áp dụng

Quy định này áp dụng cho:

- Toàn bộ source code của hệ thống
- Core Team và Partner Team
- Tất cả các module trong `modules/**`
- Các thành phần trong `common/**` và `config/**` do Core Team quản lý
- Tất cả service, controller, repository, DTO, entity

3.3. Quy định chính

3.3.1. Quy tắc đặt tên

Các thành phần phải đặt tên theo quy ước thống nhất:

Thành phần	Quy tắc đặt tên	Ví dụ
Controller	`<Name>Controller`	`UserController`
Service	`<Name>Service`	`UserService`
Repository	`<Name>Repository`	`UserRepository`

Thành phần	Quy tắc đặt tên	Ví dụ
Entity	`<Name>Entity`	`UserEntity`
DTO Request	`<Name>CreateRequest`, `<Name>UpdateRequest`	`UserCreateRequest`
DTO Response	`<Name>Response`	`UserResponse`
Mapper	`<Name>Mapper`	`UserMapper`
Enum	`<Name>Enum`	`UserStatusEnum`

3.3.2. Quy tắc chung Java

Dùng camelCase cho:

- Biến
- Method

Dùng PascalCase cho:

- Class
- Interface
- Enum

Ví dụ:

```
private String fullName;      // đúng
private String FullName;      // sai

public class UserService { }  // đúng
public class userService { }  // sai
```

3.3.3. Quy định về logging

Không được dùng:

`System.out.println("Error");`

Phải dùng logging chuẩn:

```
private static final Logger log =
        LoggerFactory.getLogger(UserService.class);

log.info("User created");
log.error("Create user failed", ex);
```

3.3.4. Validation dữ liệu đầu vào

Tất cả dữ liệu đầu vào phải được validate bằng annotation.

Annotation chuẩn:

- `@NotNull`
- `@NotBlank`
- `@Size`
- `@Email`
- `@Pattern`

Ví dụ:

```
public class UserCreateRequest {

    @NotBlank
    @Size(max = 100)
    private String username;

    @NotBlank
    private String password;

}
```

3.4. Nguyên lý thiết kế SOLID áp dụng trong dự án

3.4.1. S — Single Responsibility

Một class chỉ có một trách nhiệm.

Quy định dự án:

- Controller → chỉ xử lý HTTP
- Service → xử lý nghiệp vụ
- Repository → truy cập DB
- Mapper → convert DTO ↔ Entity

Sai:

```
public class UserController {

    @Autowired UserRepository repo;   // sai tầng
}
```

Đúng:

```
public class UserController {

    @Autowired UserService userService;
}
```

3.4.2. O — Open/Closed

Code phải mở rộng được nhưng không sửa code cũ.

Áp dụng:

- Dùng interface Service
- Không sửa class gốc khi thêm logic

Ví dụ:

```
public interface NotificationService {
    void send(Message msg);
}
```

Triển khai:

```
public class EmailNotificationService implements NotificationService { }
public class SmsNotificationService implements NotificationService { }
```

3.4.3. L — Liskov Substitution

Class con có thể thay thế class cha.

Quy định:

- ServiceImpl phải tuân interface
- Không throw exception mới trái contract

Sai:

```
public class UserServiceImpl implements UserService {
    public User get(String id) {
```

```
        throw new UnsupportedOperationException();
    }
}
```

3.4.4. I — Interface Segregation

Không tạo interface quá lớn.

Sai:

```
public interface UserService {
    create();
    update();
    delete();
    exportExcel();
    sendEmail();
}
```

Đúng:

```
public interface UserService { }
public interface UserExportService { }
public interface UserNotificationService { }
```

3.4.5. D — Dependency Inversion

Phụ thuộc abstraction, không phụ thuộc implementation.

Quy định:

- Inject interface
- Không new service

Sai:

```
UserService service = new UserServiceImpl();
```

Đúng:

```
@Autowired
UserService userService;
```

3.5. Quy tắc kiến trúc Spring Boot trong dự án

3.5.1. Luồng chuẩn tầng

`Controller → Service → Repository → Database`

Không được:

`Controller → Repository`
`Controller → EntityManager`

3.5.2. Controller không chứa nghiệp vụ

Sai:

```
@PostMapping
public User create(...) {
    user.setCreatedDate(LocalDateTime.now()); // sai
}
```

Đúng:

```
@PostMapping
public UserResponse create(...) {
    return userService.create(request);
}
```

3.5.3. Service không chứa logic HTTP

Sai:

`throw new ResponseStatusException(HttpStatus.BAD_REQUEST);`

Đúng:

`throw new BusinessException("USER_EXISTS");`

3.5.4. Repository chỉ truy vấn dữ liệu

Không chứa:

- validation
- business logic
- mapping

3.6. Design pattern sử dụng trong hệ thống

3.6.1. Service pattern

Service là tầng nghiệp vụ trung tâm.

`Controller → Service → Repository`

3.6.2. DTO pattern

Tách DTO khỏi Entity.

Không trả Entity ra API.

Sai:

`public UserEntity create(...) { }`

Đúng:

`public UserResponse create(...) { }`

3.6.3. Mapper pattern

Convert DTO ↔ Entity.

`UserEntity entity = mapper.toEntity(request);`

3.6.4. Exception pattern

Dùng BusinessException thống nhất.

```
throw new BusinessException(ErrorCode.USER_NOT_FOUND);
```

3.7. Cách thực hiện / quy trình

Quy trình tạo DTO đúng chuẩn

Bước 1: Tạo class DTO theo quy tắc đặt tên

Ví dụ:

- UserCreateRequest
- UserUpdateRequest
- UserResponse

Bước 2: Thêm validation annotation

Bước 3: Dùng DTO trong controller

```
@PostMapping
public ApiResponse<UserResponse> create(
        @Valid @RequestBody UserCreateRequest request) {

    return ApiResponse.ok(userService.create(request));
}
```

Quy trình logging chuẩn

Bước 1: Khai báo logger trong class

```
private static final Logger log =
        LoggerFactory.getLogger(UserService.class);
```

Bước 2: Dùng logger thay cho System.out.println

3.8. Ví dụ minh họa

Trường hợp sai

```
public class usercontroller {
```

```
    public void CreateUser() {
        System.out.println("Create user");
    }

}
```

Sai vì:

- Tên class không đúng chuẩn
- Tên method không camelCase
- Dùng System.out.println
- Không theo kiến trúc tầng

Trường hợp đúng

```
@RestController
public class UserController {

    private static final Logger log =
            LoggerFactory.getLogger(UserController.class);

    private final UserService userService;

    public UserController(UserService userService) {
        this.userService = userService;
    }

    @PostMapping
    public ApiResponse<UserResponse> create(
            @Valid @RequestBody UserCreateRequest request) {

        log.info("Create user {}", request.getUsername());

        return ApiResponse.ok(userService.create(request));
    }
}
```

Ví dụ DTO sai

```
public class userDTO {

    public String name;

}
```

Sai vì:

- Tên class không chuẩn
- Không có validation

- Field public
- Không theo DTO pattern

Ví dụ DTO đúng

```
public class UserCreateRequest {

    @NotBlank
    @Size(max = 255)
    private String name;

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

}
```

3.9. Checklist áp dụng

Trước khi commit hoặc tạo PR:

Naming

- [ ]
- [ ] Controller → `*Controller`
- [ ]
- [ ] Service → `*Service`
- [ ]
- [ ] Repository → `*Repository`
- [ ]
- [ ] Entity → `*Entity`
- [ ]
- [ ] DTO → `*Request / *Response`

Coding

- [ ]
- [ ] Biến và method dùng camelCase

- [ ] Class dùng PascalCase
- [ ] Không dùng System.out.println
- [ ] Dùng logging chuẩn slf4j

Validation

- [ ] DTO có validation annotation
- [ ] Không có field public trong DTO

SOLID

- [ ] Controller không chứa nghiệp vụ
- [ ] Service không chứa HTTP logic
- [ ] Repository chỉ truy vấn DB
- [ ] Inject interface, không new
- [ ] Class có 1 trách nhiệm

Architecture

- [ ] Không Controller → Repository
- [ ] Không trả Entity ra API
- [ ]

- [ ] Có DTO mapping
- [ ]
- [ ] Exception dùng chuẩn hệ thống

3.10. Cấu trúc chuẩn của một hàm xử lý nghiệp vụ

3.10.1. Mục tiêu

Quy định cấu trúc thống nhất của một hàm xử lý nghiệp vụ nhằm:

- Dễ đọc và dễ review
- Tách rõ validate – logic – log – exception
- Tránh thiếu validate hoặc thiếu log
- Chuẩn hóa xử lý lỗi và response
- Giúp dev mới đọc code hiểu ngay luồng xử lý

3.10.2. Cấu trúc chuẩn bắt buộc

Một hàm service/controller phải theo thứ tự:

```
1. Validate input
2. Log input
3. Business logic
4. Catch exception (nếu cần)
5. Trả response / throw BusinessException
```

3.10.3. Quy định chi tiết từng phần

1. Validate

- Validate annotation ở DTO
- Validate nghiệp vụ ở service

Ví dụ:

```
if (repo.existsByUsername(request.getUsername())) {
    throw new BusinessException(ErrorCode.USER_EXISTS);
}
```

---

2. Log

Phải log:

- input chính
- hành động nghiệp vụ
- lỗi

```
log.info("Create user {}", request.getUsername());
```

---

3. Business logic

Chỉ chứa xử lý nghiệp vụ:

- mapping
- tính toán
- gọi repository
- gọi service khác

```
UserEntity entity = mapper.toEntity(request);
repo.save(entity);
```

---

4. Exception handling

Không catch Exception chung chung nếu không xử lý.

Sai:

```
try {
   repo.save(entity);
} catch (Exception e) {
}
```

Đúng:

```
try {
   repo.save(entity);
} catch (DataIntegrityViolationException ex) {
   log.error("DB error", ex);
   throw new BusinessException(ErrorCode.DB_ERROR);
}
```

---

5. Response

Service:

- trả DTO
- hoặc throw BusinessException

Controller:

- trả ApiResponse

`return UserResponse.from(entity);`

Controller:

`return ApiResponse.ok(userService.create(request));`

3.10.4. Ví dụ hàm sai

```
public User create(UserCreateRequest req) {
    UserEntity e = new UserEntity();
    e.setUsername(req.getUsername());
    repo.save(e);
    return e;
}
```

Sai vì:

- Không validate
- Không log
- Trả entity
- Không xử lý lỗi
- Không theo DTO pattern

3.10.5. Ví dụ hàm đúng chuẩn dự án

```
public UserResponse create(UserCreateRequest request) {
    // 1. Validate
    if (repo.existsByUsername(request.getUsername())) {
        throw new BusinessException(ErrorCode.USER_EXISTS);
    }
    // 2. Log
```

```
    log.info("Create user {}", request.getUsername());

    // 3. Business logic
    UserEntity entity = mapper.toEntity(request);
    repo.save(entity);

    // 4. Response
    return mapper.toResponse(entity);
}
```

3.10.6. Checklist

Khi review hàm service/controller:

- [ ] Có validate nghiệp vụ
- [ ] Có log input chính
- [ ] Không xử lý logic trong controller
- [ ] Không trả Entity
- [ ] Throw BusinessException đúng chuẩn
- [ ] Không catch Exception vô nghĩa
- [ ] Trả DTO / ApiResponse đúng chuẩn

Phiên bản #1
Được tạo 2026-02-23 10:22:20 UTC bởi admin_lifetex
Được cập nhật 2026-02-23 10:22:20 UTC bởi admin_lifetex