

Nội dung chi tiết

1. Nguyên tắc Unit Test API

Mỗi API cần test các nhóm case cơ bản:

1. Validate input
 2. Auth / permission
 3. Business logic
 4. Repository interaction
 5. Exception handling
 6. Logging
 7. HTTP status & response
-

2. Ví dụ API mẫu

Controller:

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

    private final UserService userService;

    @PostMapping
    public ResponseEntity<UserResponse> create(
        @Valid @RequestBody CreateUserRequest req) {
        return ResponseEntity.ok(userService.create(req));
    }
}
```

DTO:

```
@Data
public class CreateUserRequest {
```

```
@NotBlank
private String username;

>Email
private String email;
}
```

Service:

```
@Service
@RequiredArgsConstructor
public class UserService {

    private final UserRepository repo;

    public UserResponse create(CreateUserRequest req) {
        User u = new User(req.getUsername(), req.getEmail());
        repo.save(u);
        return new UserResponse(u.getId(), u.getUsername());
    }
}
```

3. Cấu trúc test chuẩn

Dependency:

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
```

Controller test dùng:

- @WebMvcTest
- MockMvc
- @MockBean

4. Test Validate Input

Case cần có:

- thiếu field bắt buộc
- email sai format
- request null

```
@WebMvcTest(UserController.class)
class UserControllerValidationTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private UserService userService;

    @Test
    void create_shouldFail_whenUsernameBlank() throws Exception {
        String json = """
            {"username":"","email":"a@test.com"}
            """;

        mockMvc.perform(post("/users")
            .contentType(MediaType.APPLICATION_JSON)
            .content(json)
            .andExpect(status().isBadRequest());
        }

    @Test
    void create_shouldFail_whenEmailInvalid() throws Exception {
        String json = """
            {"username":"john","email":"invalid"}
            """;

        mockMvc.perform(post("/users")
            .contentType(MediaType.APPLICATION_JSON)
            .content(json)
            .andExpect(status().isBadRequest());
        }
    }
}
```

5. Test Auth / Permission

Giả sử API cần login:

```
@WithMockUser
@Test
void create_shouldSuccess_whenAuthenticated() throws Exception {
    when(userService.create(any()))
        .thenReturn(new UserResponse(1L,"john"));
}
```

```

String json = ""
{"username":"john","email":"a@test.com"}
"";

mockMvc.perform(post("/users")
.contentType(MediaType.APPLICATION_JSON)
.content(json))
.andExpect(status().isOk());
}

```

Test chưa login:

```

@Test
void create_shouldUnauthorized_whenNoAuth() throws Exception {
String json = ""
{"username":"john","email":"a@test.com"}
"";

mockMvc.perform(post("/users")
.contentType(MediaType.APPLICATION_JSON)
.content(json))
.andExpect(status().isUnauthorized());
}

```

6. Test Business Logic (Service)

```

@ExtendWith(MockitoExtension.class)
class UserServiceTest {

    @Mock
    private UserRepository repo;

    @InjectMocks
    private UserService service;

    @Test
    void create_shouldSaveUser() {
        CreateUserRequest req = new CreateUserRequest();
        req.setUsername("john");
        req.setEmail("a@test.com");

        when(repo.save(any())).thenReturn(i -> i.getArgument(0));
    }
}

```

```
UserResponse res = service.create(req);  
  
assertEquals("john", res.getUsername());  
verify(repo).save(any());  
}  
}
```

7. Test Exception Handling

Giả sử email trùng:

```
when(repo.save(any()))  
    .thenThrow(new DataIntegrityViolationException("dup"));
```

Test:

```
@Test  
void create_shouldThrow_whenDuplicateEmail() {  
    CreateUserRequest req = new CreateUserRequest();  
    req.setUsername("john");  
    req.setEmail("a@test.com");  
  
    when(repo.save(any()))  
        .thenThrow(new RuntimeException("duplicate"));  
  
    assertThrows(RuntimeException.class,  
        () -> service.create(req));  
}
```

8. Test Logging

Giả sử service có log:

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

Test log:

```
@ExtendWith(MockitoExtension.class)  
class UserServiceLogTest {  
  
    @Mock  
    private UserRepository repo;
```

```
@InjectMocks
private UserService service;

@Test
void create_shouldLog() {
    CreateUserRequest req = new CreateUserRequest();
    req.setUsername("john");
    req.setEmail("a@test.com");

    when(repo.save(any())).thenAnswer(i -> i.getArgument(0));

    service.create(req);

    verify(repo).save(any());
}
}
```

(Thực tế có thể dùng LogCaptor)

9. Test HTTP Response

```
@Test
@WithMockUser
void create_shouldReturnBody() throws Exception {

    when(userService.create(any()))
        .thenReturn(new UserResponse(1L,"john"));

    String json = ""
        {"username":"john","email":"a@test.com"}
        "";

    mockMvc.perform(post("/users")
        .contentType(MediaType.APPLICATION_JSON)
        .content(json)
        .andExpect(status().isOk())
        .andExpect(jsonPath("$.username").value("john"));
}
```

10. Danh sách Case chuẩn cho mỗi API

Mỗi API cần tối thiểu:

Validate

-
- thiếu field
-
- format sai
-
- null body

Auth

-
- chưa login
-
- sai role
-
- đúng role

Business

-
- flow thành công
-
- dữ liệu biên
-
- điều kiện đặc biệt

Repository

-
- save OK
-
- not found
-
- duplicate

Exception

-
- DB lỗi
-
- logic lỗi
-
- external lỗi

Response

-
- status code
-
- body
-
- schema

Log

- - log khi success
 -
 - log khi error
-

11. Checklist Unit Test API

-
- Test validate
-
- Test auth
-
- Test service
-
- Mock repo
-
- Test exception
-
- Test response
-
- Verify interaction
-
- No DB thật
-
- No network

Phiên bản #1

Được tạo 2026-02-23 10:22:22 UTC bởi admin_lifetex

Được cập nhật 2026-02-23 10:22:22 UTC bởi admin_lifetex