# SOLID

* **S –** Single-responsibility principle(đơn chức năng)

*A class should have one and only one reason to change, meaning that a class should have only one job*

* Fewer test cases, lower coupling (fewer denpendencies), well-organized
* **O –** Open-closed principle

*Objects or entities should be open for extension, but closed for modification.*

* Inheritence (extends or implement in java)
* **L –** Liskov substitution principle(thay thế)

*Let q(x) be a property provable about objects of x of type T. Then q(y) should be provable for objects y of type S where S is a subtype of T.*

* Every sub or derived class should be substitutable for their base or parent class (không được phá vỡ các quy tắc của lớp cha)
* **I –** Interface Segregation(phân tách interface)

*A client should never be forced to implement an interface that it doesn't use or clients shouldn't be forced to depend on methods they do not use*.

* Không được implement khi có các phương thức không dùng đến
* **D –** Dependency Inversion principle

*Entities must depend on abstractions not on concretions. It states that the high level module must not depend on the low level module, but they should depend on abstractions*.

* Sử dụng interface để đạt abtraction

# Inversion of Control, Dependency Injection

Diagram

Description automatically generated

Diagram

Description automatically generated

**Inversion-of-Control (IoC)** principle, is about providing *any kind* of callback (which controls reaction), instead of acting ourself directly (in other words, inversion and/or redirecting control to external handler/controller) (thay vì class A chịu trách nhiệm khởi tạo B và C 🡪 inject and tranfer the control of object to a container or framework)

**Dependency-Injection (DI)** pattern (implement IoC) is a more specific version of IoC pattern, and is all about removing dependencies from your code.

* **Note:** [**Spring @Autowire on Properties vs Constructor**](https://stackoverflow.com/questions/40620000/spring-autowire-on-properties-vs-constructor)

**Spring Ioc Container:** a common characteristic of frameworks that implement **IoC**.

# Workflow

@SpringBootApplication:

* This annotation indicates that this is a Spring context file
* it enables something called auto configuration and component scan
* Auto configuration is fundamental part in Spring Boot

SpringApplication.run(..,..): run a Spring context

Note: tạo root pakage để chạy !!!

* Eg: com.ncc
* Controller: com.ncc.controller

🡪phục vụ cho auto component scan

# Spring Boot Actuator

* là một sub-project của Spring Boot cho phép theo dõi, giám sát ứng dụng, thu thập số liệu, lưu lượng truy nhập hay trạng thái cơ sở dữ liệu,… mà không cần thêm bất kỳ dòng code nào
* Khi được cấu hình Spring Actuator thì mặc định sẽ có sẵn 16 endpoint để quản lý và theo dõi ứng dụng

# Spring AOP

Stand for Aspect Oriented Programming

* Là một kỹ thuật lập trình nhằm phân tách chương trình thành các module riêng rẽ, phân biệt, không phụ nhau

🡪khi hoạt động chương trình sẽ kết hợp các module nhưng khi sửa đổi thì chỉ cần sửa 1 module

* AOP không phải để thay thế OOP mà để bổ sung cho OOP
* Aspects enable the modularization of concerns such as tracsaction management, logging or security that cut across multiple types and objects (often termed **crosscutting concerns** )

Diagram

Description automatically generated

### Terminology and concepts

Diagram

Description automatically generated

* **Join Point**: is a point of execution of the program, such as the execution of a method or the handling of an execution (in spring a joinpoint always represents a method execution)
* **Pointcut**: is a predicate or expression that matches join points
* **Advice** is associated with a pointcut expression and runs at any join point matched by the pointcut

<https://techmaster.vn/posts/36087/spring-core-phan-5-spring-aop-la-gi-code-vi-du-voi-spring-aop#:~:text=Aspect%20Oriented%20Programming%20(AOP)%20l%C3%A0,tr%E1%BA%A3%20v%E1%BB%81%20m%E1%BB%99t%20k%E1%BA%BFt%20qu%E1%BA%A3>

# Spring Sercurity

## 5 core concepts in spring sercurity

* Authentication
* Authorization
* Principal: currently logged in user
* Granted Authority: quyền được thực hiện của authenticated user
* Roles: một nhóm quyền của authenticated user

\*Note:

* Encoding: quá trình convert data từ dạng này sang dạng khác, KHÔNG SỬ DỤNG MÃ HÓA
* Encypt: quá trình tranform data 🡪 cipher text (symetric and asymetric)
* Hashing: quá trình convert data thành 1 chuỗi hash sử dụng hash function

## Graphical user interface, application, Word Description automatically generatedTổng quan

* Spring security filters chain: mặc định khi cấu hình spring security sẽ đặt toàn bộ 1 filter chain gồm 15 filter khác nhau
* **BasicAuthenticationFilter**: cố gắng tìm Basic Auth HTTP Header theo request và xác thực nếu tìm thấy
* **UsernamePasswordAuthenticationFilter**: tìm tham số request username/password hay POST body 🡪 authenticate user nếu tìm thấy
* **DefaultLoginPageGeneratingFilter**: tạo trang login
* **DefaultLogoutPageGeneratingFilter**: tạo trang logout
* **FilterSecurityInterceptor**: thực hiện authorization

### Cấu hình Spring Security: WebSecurityConfigurerAdapter

1. @EnableWebSecurity: tell spring this is a web security configuration,
2. Extends WebSecurityConfigurer
   * Bằng cách override **configure(HttpSecurity http)** 🡪 cấu hình FilterChain

Text

Description automatically generated

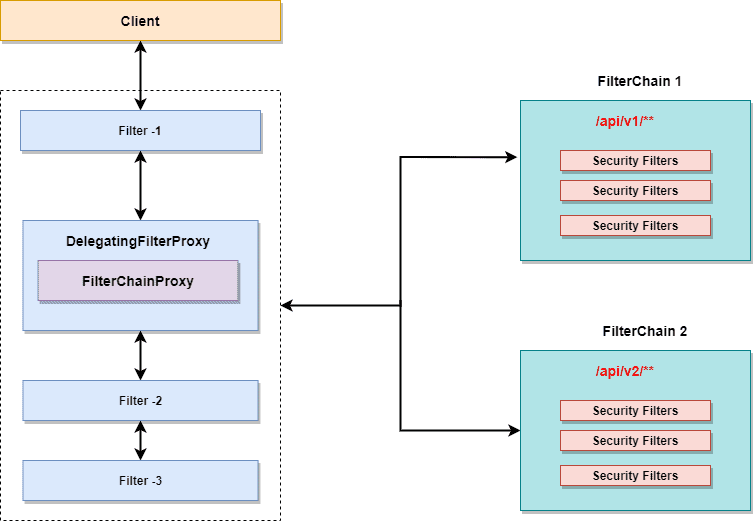
* + Tất cả các request đến “/” và “index” sẽ không cần authenticate nữa (permitAll)
  + .httpBasic() nghĩa là gửi một HTTP Basic Auth Header để authenticate

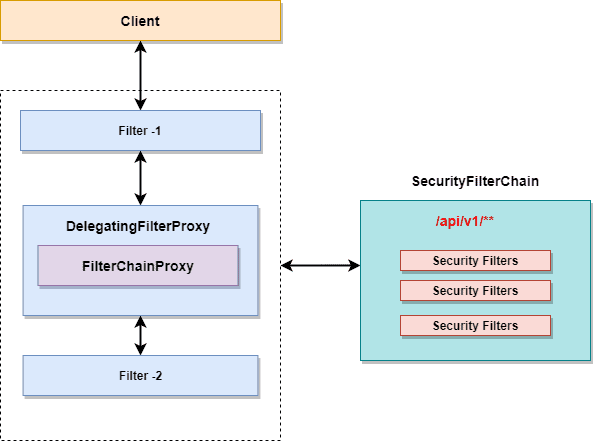
🡪mỗi lần gửi request sẽ đều gửi theo user và password cấu trúc header sẽ có thêm

* Authorization: Basic <Base 64 encode {username:password}>
* Cơ chế này dễ dàng decode (có rất nhiều trang web thực hiện đc việc này)

### How Security Filters Works

<https://www.javadevjournal.com/spring-security/spring-security-filters/>





1. DelegatingFilterProxy

It is a filter which work as a bridge between Servlet container’s life circle and Spring’s Application Context. Servlet container does not have any information about the Spring’s application context, but Spring security needs security filters to execute the task 🡪 **delegation** to spring bean to start the security flow

1. FilterChainProxy

It contains all the details about the different security filtes available through the security filter chain

* It delegates the task to the chain based on the URI’s mapping or using RequestMatcher interface
* It’s **not executed directly** but **started by the DelegatingFilterProxy**

1. SecurityFilterChain
2. SecurityFilters

A picture containing graphical user interface

Description automatically generated

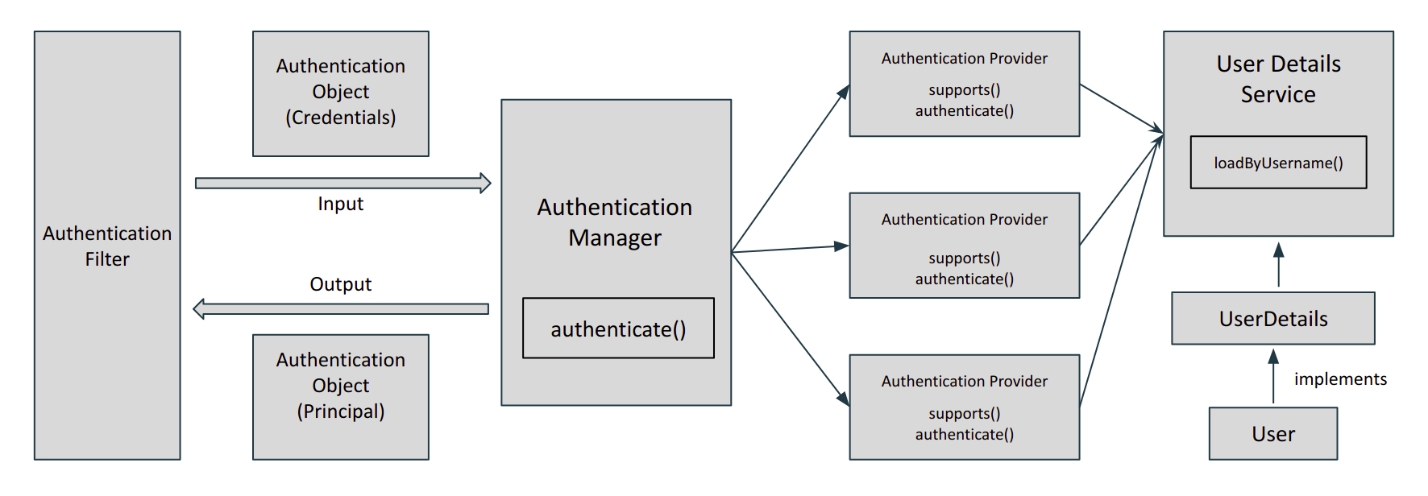
### How spring security authentication works?

Diagram

Description automatically generated

Diagram

Description automatically generated



Khi thêm spring-boot-stater-security dependency vào thì mặc định mọi URL “/\*\*” sẽ đi qua tầng filter-Authentication Filter, which initiates the process of authentication

* Once the request passes through the authentication filter, the **credentials** of the user are stored in the **Authentication object (**in **SecurityContext** that ismanaged by **SecurityContextHolder).** Now, what actually is **reponsible for authentication** is **AuthenticationProvider (**via authenticate()method**)**
* Spring can have multiple authentication providers (may using Oauth, LDAP)🡪 managed by **AuthenticationManager**. It finds the **appropriate authentication provider** by calling supports() method **for each authentication provider** 
  + If true is return 🡪 call authenticate()
* After the credentials are passed to the authentication provider 🡪 look for the existing user in the system by **UserDetailsService** 🡪 return a **UserDetails** instance
  + If success 🡪 the **authentication object** is returned with **Principal and Authorities**
  + **AuthenticationException** is thrown

**Các thành phần:**

* AuthenticationManager: khi thực hiện authenticate() sẽ có thể làm được 1 trong 3 việc sau
  + Trả về 1 đối tượng Authentication (trường hợp authenticated=true – input hợp lệ)
  + Trả về AuthenticationException nếu không hợp lệ
  + Trả về null nếu không xử lý được

**Trong Spring Security**

**Text

Description automatically generated**

**Diagram

Description automatically generated**

**SecurityBuilder** used to create an **AuthenticationManager**. Allows for easily building **in memory authentication, LDAP authentication, JDBC based authentication**, adding **UserDetailsService**, and adding **AuthenticationProvider's**.

### Example workflow

Diagram

Description automatically generated

Diagram

Description automatically generated

1. User thực hiện điền thông tin credentials và gửi request đến server
2. Đi qua tầng filter (15 filter) thì **UsernamePasswordAuthenticationFilter** sẽ thực hiện tạo **UsernamePasswordAuthenticationToken** thông qua credentials (username/password) stored in **Authentication** Object (in **SecurityContext** that ismanaged by **SecurityContextHolder**)
3. Pass this token to the **AuthenticationManager**
4. **AuthenticationManager** delegate the Authentication to **AuthenticationProvider** bằng cách sử dụng phương thức supports()
5. Sử dụng **UserDetailsService** để tìm user với username trong database

* Nếu có trong database
  + SessionAuthenticationStrategy được thông báo để tạo new login. Xử lý HTTP Session và đảm bảo 1 phiên hợp lệ tồn tại 🡪 chống session-fixation attacks
  + Thực hiện update **UserDetails** in **SecurityContextHolder**
  + If RememberMeServices is active 🡪 call loginSuccess method
  + Publish an **InteractiveAuthenticationSuccessEven**
  + **AuthenticationSuccessHandler** is invoked
* Nếu không có trong database
  + Clear **SecurityContextHolder**
  + Call **loginFail** method of the RememberMeServices to remove cookies and other related information
  + **AuthenticationFailureHandler** triggers to perform any additional action

### Spring Method Security

Cung cấp authorization semantics at the method level 🡪 secure service layer

* @Secured: used to specify a list of roles on a method (has at least one of the specified roles)
  + Does not SpEL (Spring Expression Language)
* @RoleAllowed: equivalent of the @Secured
* @PreAuthorize and @PostAuthorize: provide expression based access control 🡪 written using SpEL
  + @PreAuthorize: **checks the given expression before entering method**
  + @PostAuthorize: **verifies the given expression** **after the execution of the method and could alter the result**

## CFRS Protection

## JWT Authorization

Stands for **Json Web Tokens:** là một tiêu chuẩn mở (RFC 7519) định nghĩa cách thức truyền tin an toàn giữa các ứng dụng bằng đối tượng JSON

* Dữ liệu truyền đi sẽ được mã hóa và chứng thực
* Phần chữ ký sẽ được mã hóa lại bằng **HMAC**(symmetric) or **RSA (**asymmetric**)**

Các thành phần:

* Header
* Payload
  + Reserved claim: thông tin quy định sẵn
  + Public claim: định nghĩa công khai bởi những bên sử dụng JWT
  + Private claim: tự định nghĩa và chỉ các thành phần liên quan đến hệ thống hiểu được
* Signature

Eg: header.payload.signature

Vấn đề khi sử dụng sessionID

Workflow

Diagram

Description automatically generated

Gennerate JWT

Text

Description automatically generated

* Jwts.builder(): create a JwtBuilder instance

# Auth0, OAuth2

Auth0: Token-based Single Sign On your Apps and APIs with social, databases and enterprise identities

* Is an organization, who manages Universal Identity Platform for web, mobile and IOT can handle any of them

OAuth 2: an open standard for access delegation

* It is an authorization framework that enables a third-party application to obtain limited access to resources the end-user owns.

# Scheduler/Cronjob

|  |  |
| --- | --- |
| Fixdelay | Fixrate |
| Make sure that there is a **delay of n milisecond** between **the finish time of an execution** of a task and **the start time of the next execution** of the task | Runs the scheduled task at **every n milisecond** |
|  | 🡪does not check for any previous executions of the task (independent) |
|  | 🡪if the coming tasks do not finish quickly, it’s impossible they end up with “Out of Memory exception” |
|  |  |

InitialDelay: trễ n milisecond rồi thực hiện lần chạy đầu tiên

Cron expression: <minute> <hour> <day-of-month> <month> <day-of-week> <command>

# Email Service, Thymeleaf, Multilanguge

## Email service

MIME: Multipurpose Internet Mail Extensions (via SMTP) support image, voice, text in message

2 cách config mail:

* Use application.properties file
* Create EmailSenderService bean in java file

## Thymeleaf

## Multilanguge

## @Async

Note:

* It must be applied to *public* methods only🡪 can be proxied
* Self-invocation – calling the async method from within the same class won’t work🡪 because it bypasses the proxy and calls the underlying method directly

### Aditional:

**Future:** is a interface that was introduce in java 5 to represent a value that will be available in the future

# [Repository, Spring JPA, Native query, HQL](https://github.com/nccasia/ncc-java-spring-basic/blob/main/repository-springjpa-nativequery-hql)

## JPA repository

**Query derivation** merchanism built into theSpring Data repository infrastructure which is useful to **build constraining queries over entities** of the repository. The prefixes **findBy, readBy,** and **getBy** from method and start parsing the rest of it (concatenate them with **And** and **Or**)

Graphical user interface, text, application, chat or text message

Description automatically generated

## Projections

Problem: khi load dữ liệu từ database nhưng chỉ sử dụng một trong số các thuộc tính

🡪Spring data allows modeling dedicated return types to retrieve partial views of the managed aggregates

Type of projections:

* Interface-based Projections
* Class-based Projections (DTOs)
* Dynamic Projections

## Repository

Native query: là SQL, ngôn ngữ truy vấn trên bảng và cột

## HQL

Stands for Hibernate Query Language

Làm việc với Persistent Object (instance of POJO class)

### Custom result

1. Custom the Result with class contructor
   * The class specified needs to **have a constructor that matches the projected attributes exactly**, but it’s **not required** to be annotated with **@Entiry**

### Projections

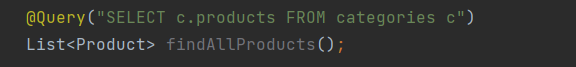
* Interface-based projection
  + Close-projection
  + Open-projection
* Class-based projection

### JOINS

* Implicit inner join

Text

Description automatically generated



* Explicit inner join

### CACHE

**@EnableCaching:** will start the search for a CacheManager bean to configure the cache provider

By default: a Simple in-memory provider would be used

The data in the cache is stored using a key-value pattern. **Spring Cache uses the parameters of the method as key anf the return value as a value in the cache**

* First level cache

Text

Description automatically generated

* Second level cache

# Mapstruct, Mapper

Mapstruct: is an open source Java based code generator which creates code for mapping implementations. It uses annotation-processing to generate mapper class implementations during compilation

**@Mapper(componentModel = "spring"):** tells Mapstruct that when generating the mapper implementation class with @component

ModelMapper**:** is an intelligent object mapping library that automatically maps object to each other

### How it works?

<http://modelmapper.org/user-manual/how-it-works/>

<http://modelmapper.org/user-manual/configuration/#matching-strategies>

Model Mapper consists of two separate process:

* Matching Process
* Mapping process

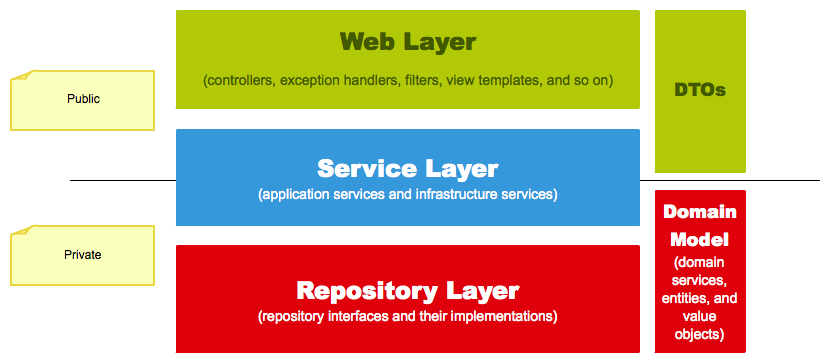
# DTO

## Optional

<https://www.petrikainulainen.net/software-development/design/understanding-spring-web-application-architecture-the-classic-way/>

Two pillars of a Good Architecture

* The separation of concerns (SoC) Principle: is a design principle for separating a computer program into distinct sections, such that each section addresses a separate concern (Eg. 3-tier)
* The Keep It Simple Stupid (KISS) principle



# TEST

## Unit Test

**Why write unit test:**

* **DON’T** find the bug that may exist in code
* To TEST the functionnality of our code weather code is **working as expected or not**
* Unit: for Testing depends upon individual programmers (1..n Function, 1..n lines of code, 1..n Objects..)
* One Unit test runs independently of any other Unit Test, Code etc

Tests written for testing a unit of code

Should Completes within miliseconds

Writing Unit Tests: **Arrange – Act – Assert (AAA)**

### Test doubles

A **test double** is an object that can stand in for a real object in a **test,** similar to how a stunt double stands in for an actor in a movie

* Test Double are patterns that allow us to control dependencies between the tested unit

Sometimes it is impossible to unit test the code because of the unavailability of the collaborator objects, or the cost of interaction and instantiation of collaborators

Used in lieu of External Dependencies

* DB, Web, API, Library, Network etc

Need to use objects or procudures that look and behave like their release-intended couterparts

**Type of doubles:**

* Dummy object:
  + mục đích để tạo các object và pass vào parameters list (điền cho đủ chứ it is never actually used)
  + A dummy object is not directly used in the test or code under test, but it required for the creation of the another object
* Stubs: these are objects which we hard code to provide canned responses whenever specific methods are called on those objects
  + In case when we are interested in method output and we want to be sure that whenever it will be called the result will be exactly as we want to
  + Cung cấp kết quả của những lời gọi hàm/method
* Mock objects is a combination of a spy and a stub (stand-ins for real objects the class being tested uses)🡪 using Mock instead of fake objects
* Fake objects - simulators
* Test spy: a spy is a variation of a stub, but instead of only setting the expectation, a spy records the method calls made to collaborator
  + Là một biến thể của stub mà có ghi lại thông tin nào đó dựa trên việc chúng được gọi như thế nào (Eg. Một dịch vụ email ghi lại bao nhiêu message đã gửi đi)

**What is Junit 5?**

**@Test**

***@AfterEach* (@After - Junit4)**

***@BeforeEach* (@Before - Junit4)**

***@BeforeAll, @AfterAll* (@BeforeClass, @AfterClass – Junit4)**

## Intergration Test

## Mokito Framework

|  |  |  |
| --- | --- | --- |
| ***Mockito.mock()*** | ***@Mock*** | ***@MockBean*** |
| Create mock object of a class or an interface | Shorthand for the Mockito.mock() | Add mock objects to the Spring application context  +replace existing bean of the same type  +a new one will be added, if no bean of the same type is defined |
|  | @Mock makes it easier to find the problem mock in case of a failure, as the name of the field appears in the failure message |  |
|  | Add @RunWith(MockitoJUnitRunner.class) if using Junit4  Or | **Useful in integration tests** |

# Redis

<https://stackoverflow.com/questions/9625246/what-are-the-underlying-data-structures-used-for-redis#:~:text=See%20this%20question%20for%20an,hash%20table%20to%20the%20second>.

# Spring Batch

# Data Migration

## Definition

**Data migration** is the process of **moving** data from one location to another location, one format to another or one application to another.

### Data migration Strategies

1. ‘Big bang’ migration

A big bang migration is the process of migrating applications and data in one operation, and switching from an old system to a new one

1. ‘Trickle’ migration

Complete the migration process in phases. During implementation, the old system and the new run in **parallel 🡪** eliminates downtime or operational interruptions. Processes running in real-time can keep data continuously migrating.

### Flyway

In flyway, all changes to the database are called **migrations**.

* **Versioned migrations**: have a *version, a description* and a *checksum*
* **Repeatable migrations**: have desciption and checksum (re-applied every time their checksum changes)

1. **Versioned migrations**

**Regular** used for :

* Creating/altering/dropping table/indexes/foreign keys/…
* Reference data updates
* User data corrections

**Undo Migrations:** responsible for undoing the effects of the versioned migrations with the same version

1. **Repeatable migrations**

Typically used for

* Re-creating views/procedures/functions/pakages/…
* Bulk reference data

1. SQL-based migrations

**Naming**

**Timeline

Description automatically generated with medium confidence**

**Graphical user interface, text, application, email

Description automatically generated**

**Java-based migrations**

Must implement the **JavaMigration** interface. Most users however should inherit from the convenience class **BaseJavaMigration**

**Graphical user interface, application

Description automatically generated**