S.O.L.I.D

SRP single responsibility principle

OCP open closed principle

LSP Derived class should be substitutable for the base class.

ISP Interface Segregation Principle

DIP Dependeny Inversion Principle

Sandi Metz's Rules

100 Classes can be no longer than 100 lines of code

5 Methods can be no longer than 5 lines of code

4 Method should have no more than 4 parameters

only 1 Controller should instantiate only 1 object

You should break the rules only if you have a good reason or your pair lets you.

Basic Rules

meaningful name for class, method, variable and constant

avoid using suffix for service name

remove the unused method and variable

remove the unnecessary variable and constant

prefer to use DateTimeImmutable for handling date

return value directly and earlier, if possible

short method, and just do one thing

set the lowest visibility to class, constant, variable, method(start with final and private as default)

docblock must bring additional information

declare always the return value for method

DRY: do NOT repeat yoursefl

self-documenting code

Symfony Specific

Use php-cs-fixer to check the coding standard

avoid to use @template annotation, prefer to use $this->render

prefer to use $this->render('@AppBundle/index.html,twig');

define the Permission via @IsGranted(), @Security("is_granted()")

Create a bundle for reused code, only if it is resued as a stand-alone piece of software

Use Attributes or Annotations to Configure Routing, Caching and Security

Use ParamConverters If They Are Convenient

Use Snake Case for Template Names and Variables: user_profile.html.twig

Prefix Template Fragments with an Underscore: _user_avatar.html.twig

Use a Single Action to Render and Process the Form, i.e. GET, POST for the same form

Use Voters to Implement Fine-grained Security Restrictions

Hardcode URLs in a Functional Test

DTO Object

validate the property with constraint annotation

always final class definition

Contains no business logic

Each client has always individual DTO

Avoid **NULL** values

Use builder|command pattern to construct a complex DTO

Function

Minimalize the number of arguments
has only one level of abstraction
avoid side effects
don't use flag(e.g. $isSomethingSpecial) as function argument
avoid complex conditions
function names should say what they do
avoid using negative conditionals
just do one thing
use explicit function instead of generic setter

Doctrine

Use only Interface to access database in application & domain layers
Implement CQS in complex domain
Read operation via Repository
Write operation via EntityManager
Remove default value from doctrine annotation in Entity (Example: default values like type="string", length=255, nullable=false)
define public setter/getter if really necessary

Migration

Remove the auto-generated comments
Do not handle invalid platform, if MySQL is assumed due to full control of environment

Tests

Entity, ValueObject, DTO, DAO will not be mocked
Use Alice & faker for complex object
validation should not be mocked, prefer to use ValidationBuilder