

# FRAMEWORKS : POUR QUOI ? POUR QUI ? PAR QUI ?

---

Utilisateur

Utilisateur avancé

Et/ou "créateur"



alamy

Image ID: 859000  
[www.Alamy.com](http://www.Alamy.com)

# VOTRE EXPERIENCE

Qui a déjà utilisé des frameworks ?  
de façon avancée ?  
écrit un Framework ?

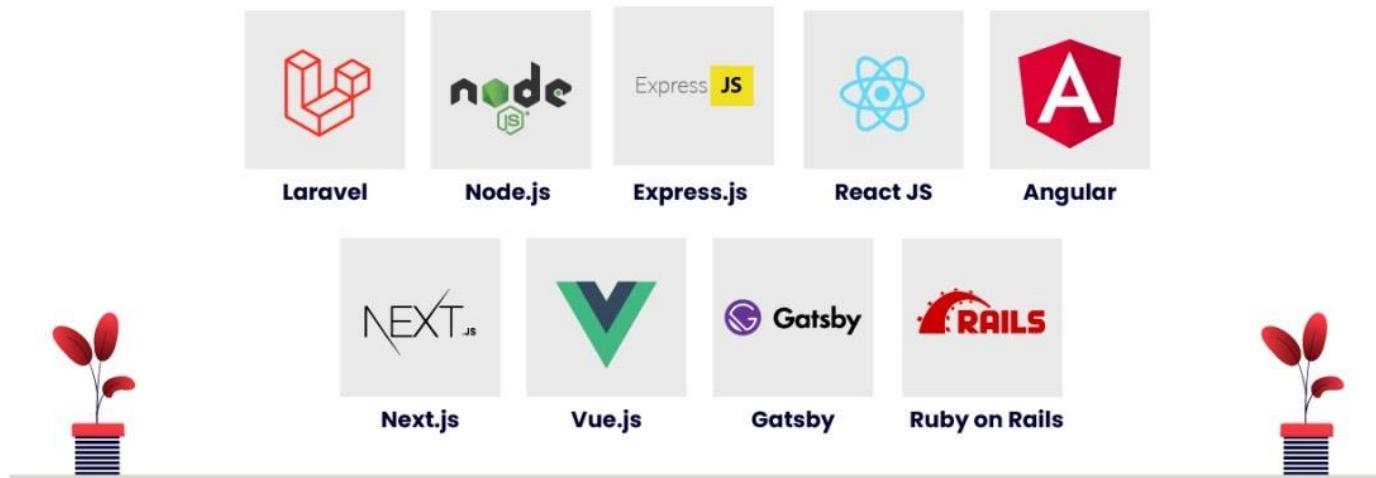
Quels Frameworks connaissez-vous ?  
Leurs avantages ?  
Leurs inconvénients ?

Comment fonctionnent-ils ?



# Pléthore de Frameworks

## Top Web Development Frameworks for Success in 2025



<https://d3logics.com/top-web-development-frameworks-for-2025/>

<https://invedus.com/blog/popular-java-frameworks-for-web-development/>

Quelles sont les spécificités de SPRING ?  
A quels besoins répond ce Framework ?





# LE BUT DE SPRING

---

Aider à coder proprement

Mettre l'accent sur l'essentiel

Eviter la réécriture de code similaire et systématique

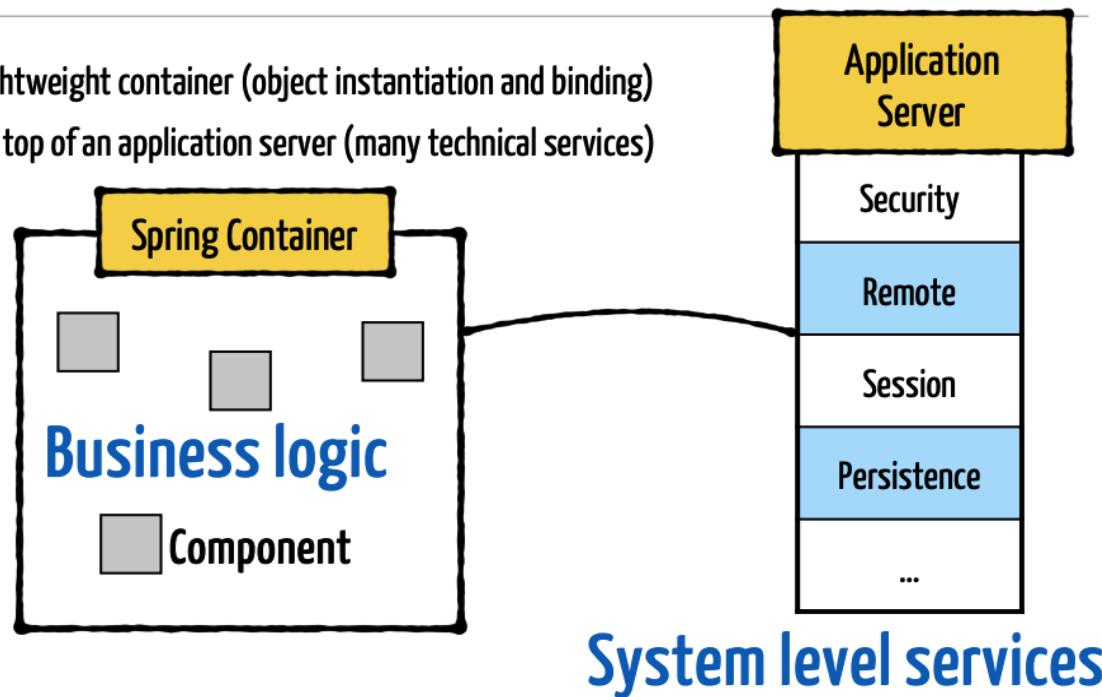
Ne pas polluer la **logique métier**  
(annotations, injection de dépendances)  
Faciliter **l'hétérogénéité** hors système



# SPRING : séparer préoccupations techniques et logique métier

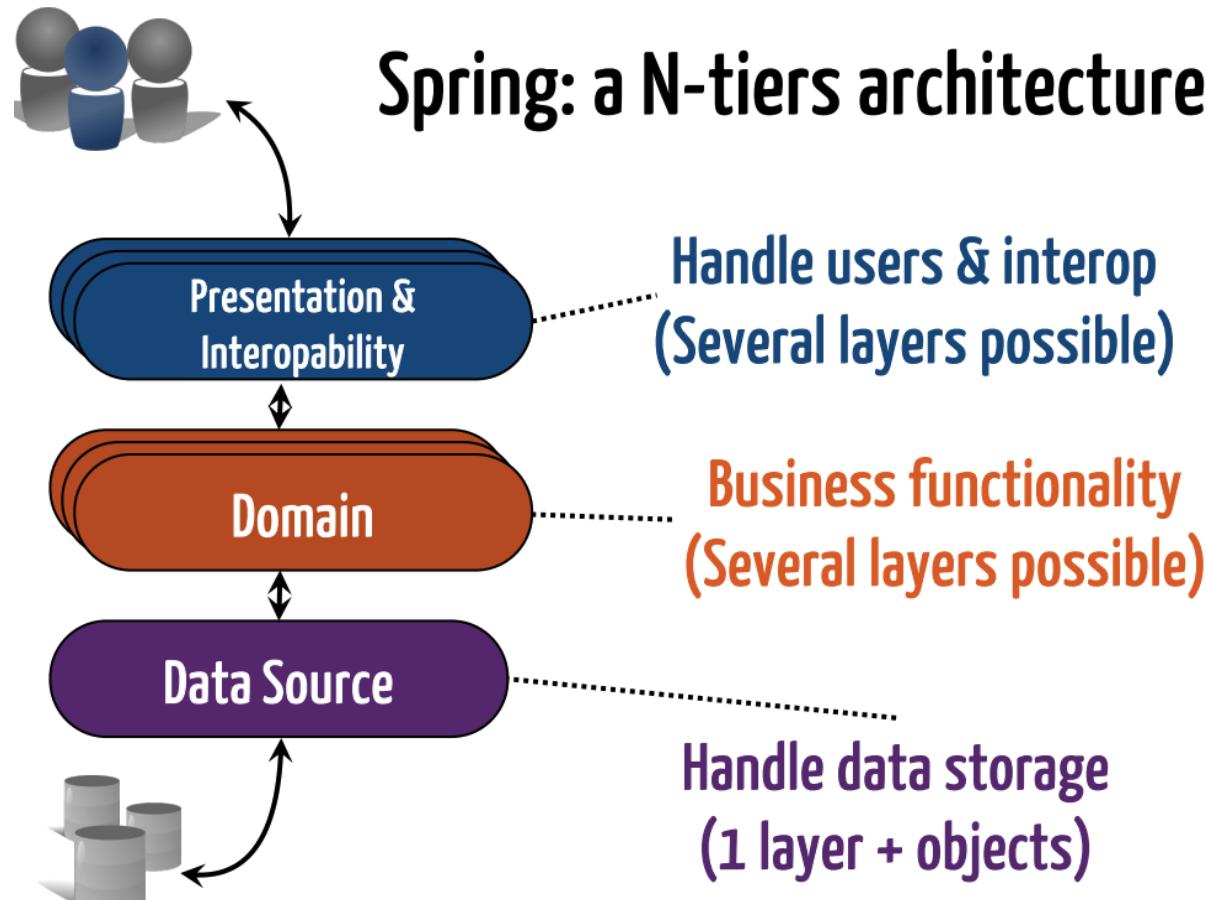
## Spring Principles (and Benefits)

- Lightweight container (object instantiation and binding)
- On top of an application server (many technical services)



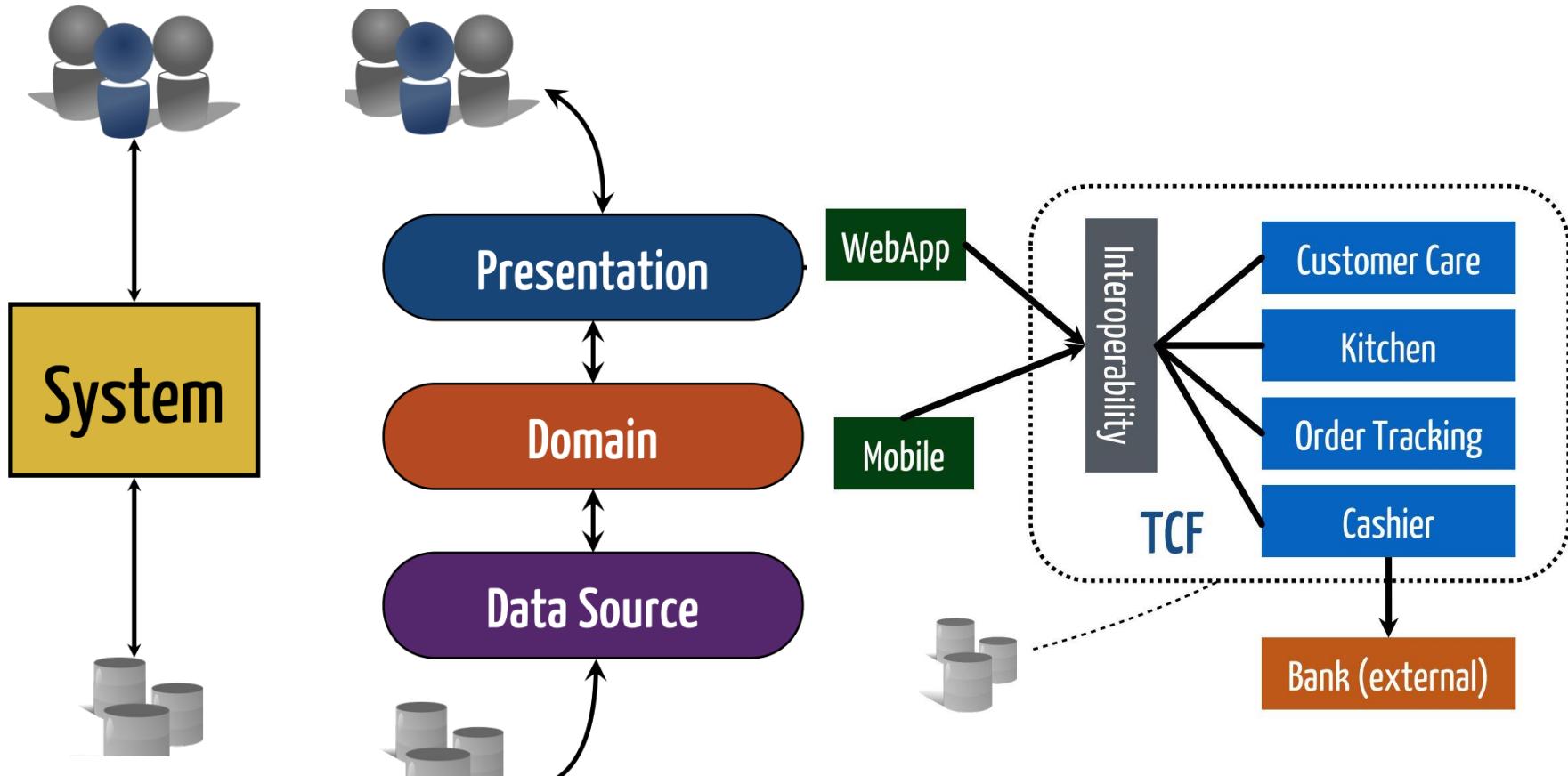
# SPRING : être ouvert à l'extérieur

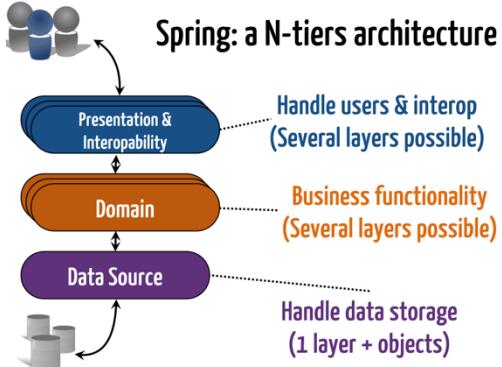
## Architecture 3-tiers



Faciliter les  
interactions avec  
le **système**  
Et  
la gestion des  
données

# Logique métier et Hétérogénéité



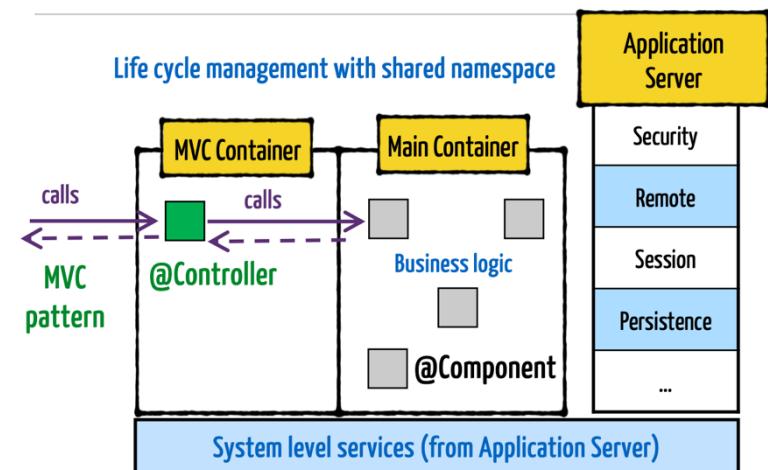


# EXPOSER LE METIER A L'EXTERIEUR

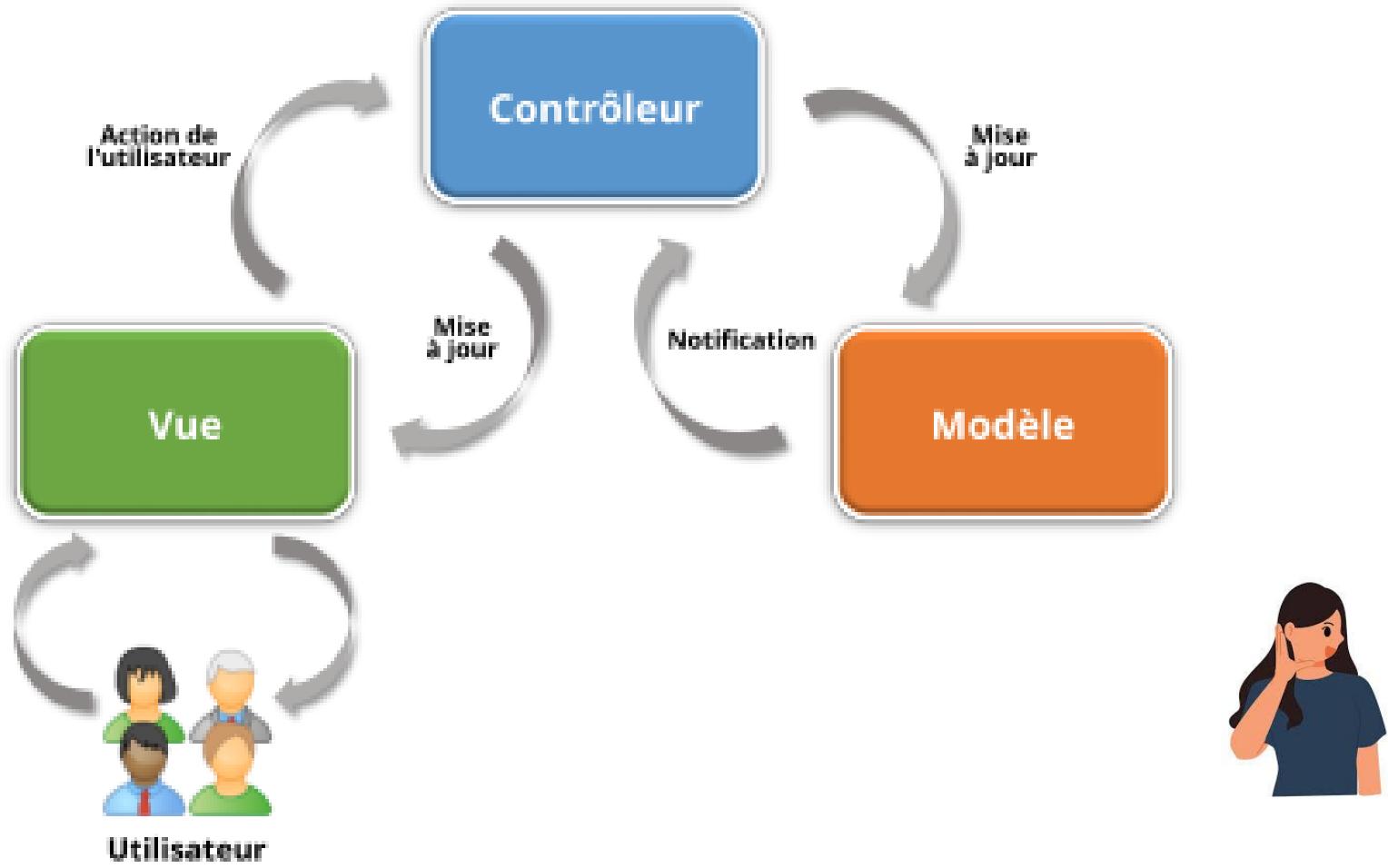
## Everything started with SpringMVC

- Aim: building flexible and loosely coupled web applications
- Model-view-controller design pattern helps in separating
  - the business logic,
  - presentation logic,
  - navigation logic.
- Models are responsible for encapsulating the application data (POJOs)
- The Views render response to the user with the help of the model object (HTML)
- Controllers are responsible for receiving the request from the user, calling the back-end services, and passing the resulting model to the right view

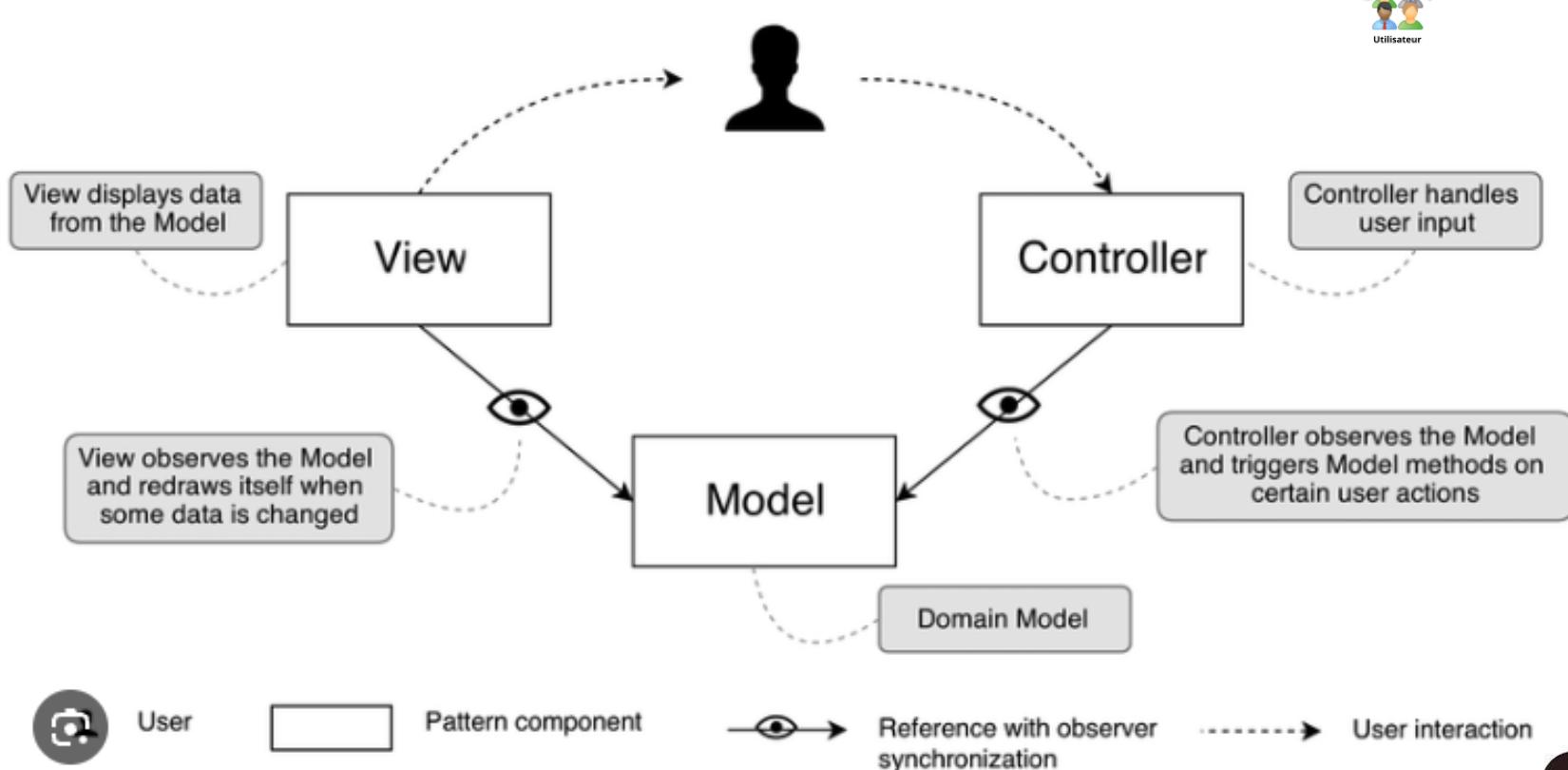
## Spring MVC infrastructure



# Un MVC peut en cacher un autre



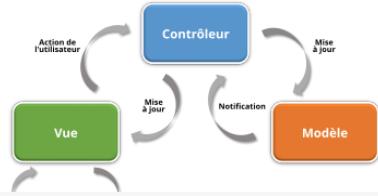
# MVC définition Smalltalk



[https://www.researchgate.net/figure/Smalltalk80-MVC-pattern-View-and-Controller-work-as-a-pair-allowing-the-user-to-interact\\_fig2\\_269303611/](https://www.researchgate.net/figure/Smalltalk80-MVC-pattern-View-and-Controller-work-as-a-pair-allowing-the-user-to-interact_fig2_269303611/)



# MVC Django (Python)



## Les contrôleurs (views) sous Django : du MVC élégant !

[Framework Django](#) / [Les contrôleurs \(views\)](#)

Cette section est dédiée à l'écriture de contrôleurs avec Django : vous y découvrirez ce que recouvre le concept de contrôleur sous Django, et comment les mettre en œuvre.

### Django et les contrôleurs : le programme

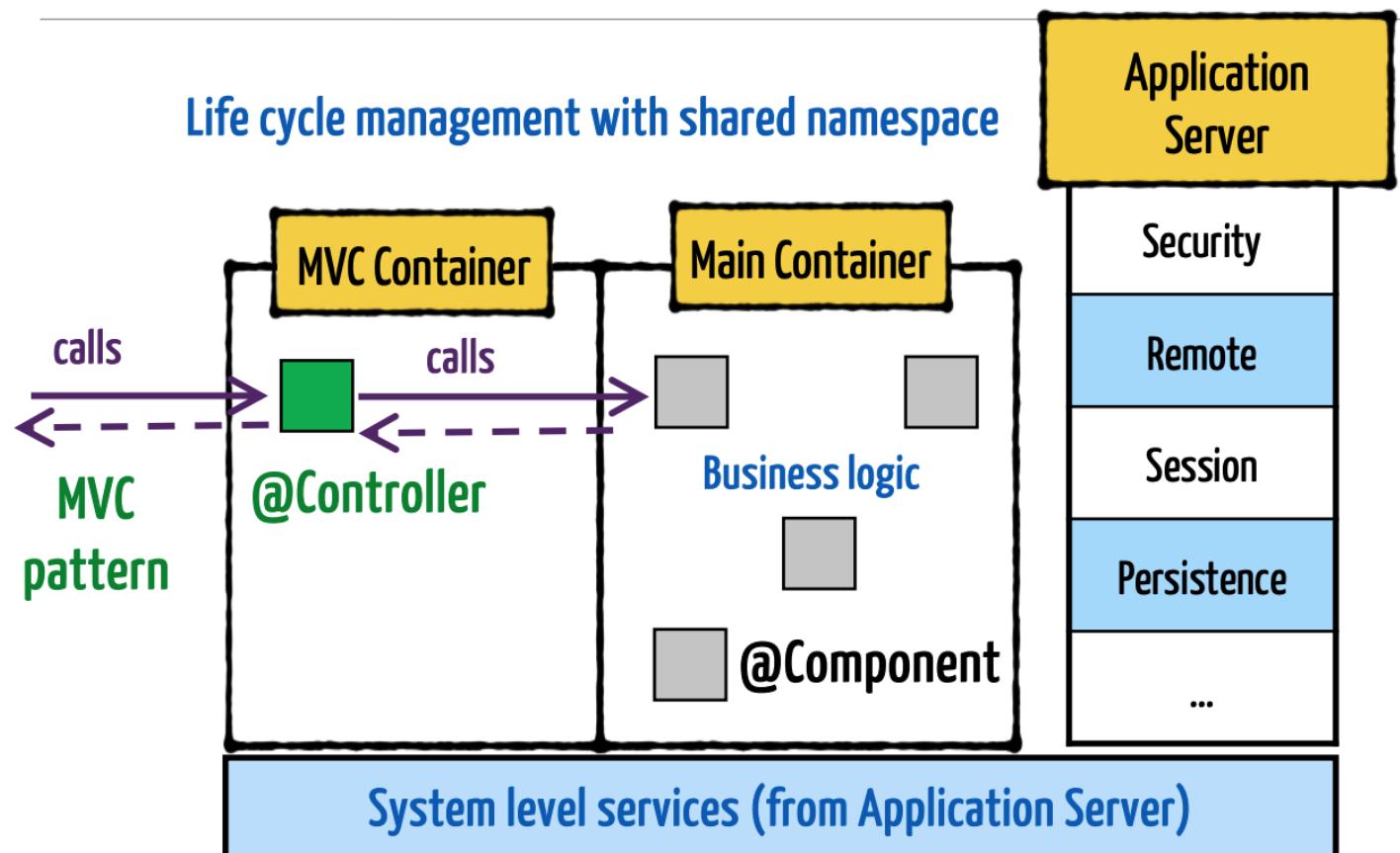
1. [Quelques petits rappels sur les contrôleurs en MVC](#) – Pour comprendre l'implémentation des contrôleurs (views) sous Django, il faut bien comprendre ce que l'on entend par contrôleur côté MVC, et connaître les spécificités des contrôleurs sous Django. Cette page est faite pour ça !
2. [Django function based views : 1 contrôleur = 1 fonction](#) – Nous allons voir sur cette page comment écrire des contrôleurs (views) puissants très simplement en Django, sous la forme de fonctions : un contrôleur = une fonction. Tout simplement !

### Contrôler vs view ?

<https://www.formation-django.fr/framework-django/controleurs-views/>

# MVC en Spring

## Spring MVC infrastructure

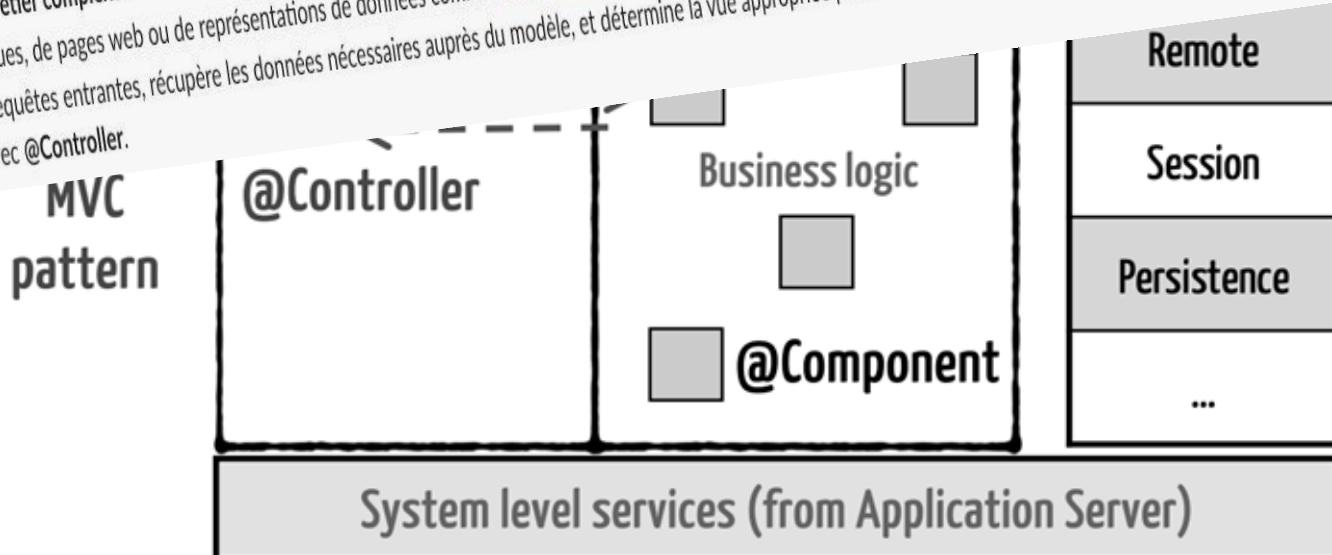


# MVC en Spring

## Spring MVC infrastructure

### Life cycle management with shared beans

- Modèle : Dans le contexte de Spring, le modèle peut être implémenté à l'aide de classes POJO (Plain Old Java Objects) ou de composants plus avancés comme des services, des repositories ou des entités. Ces composants encapsulent la logique métier complexe.
- Vue : interfaces graphiques, de pages web ou de représentations de données comme JSON ou XML.
- Controller : Il gère les requêtes entrantes, récupère les données nécessaires auprès du modèle, et détermine la vue appropriée pour le rendu final. Dans Spring MVC, les contrôleurs sont généralement implémentés sous forme de classes annotées avec @Controller.



# Au centre de la logique métier : les composants et les containers

Un composant : une **brique** permettant la programmation par assemblage en se centrant sur **la logique métier**

Un container : gère l'assemblage, le **cycle de vie** des composants et l'intégration des **services orthogonaux** (persistance, sécurité....)

# ASSEMBLER DES BRIQUES IMPORTANCE DES CONTRATS : INTERFACES



## All-in-One (in Spring)

### Component

```
@Component
public class CartHandler implements CartModifier, CartProcessor {

    CustomerRepository customerRepository;

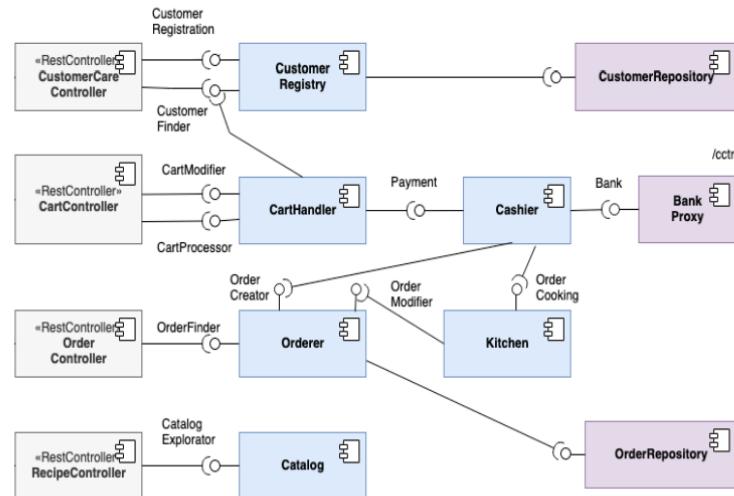
    Payment payment;

    @Autowired
    public CartHandler(CustomerRepository customerRepository, Payment payment) {
        this.customerRepository = customerRepository;
        this.payment = payment;
    }

    // Implementation methods for CartModifier and CartProcessor interfaces
}
```

### Provided Interfaces

### Required Interfaces

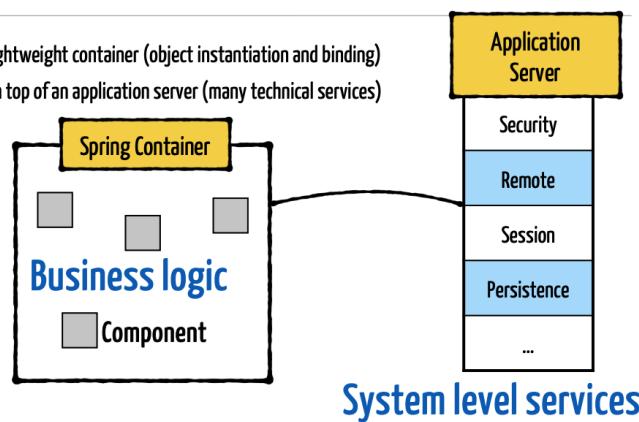


## Component diagrams

# AVEC DES CONTAINERS et ECO SYSTÈME COMPOSANTS METIERS

## Spring Principles (and Benefits)

- Lightweight container (object instantiation and binding)
- On top of an application server (many technical services)



5

Domain Beans interfaces as Verbs

DataSource Beans as Nouns

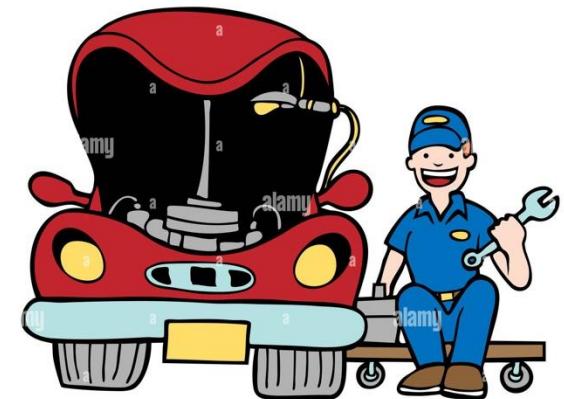
## Inversion of Control

aka the Hollywood pattern

Your code reuses a library

A framework reuses your code

# COMMENT => OUVRIR LE CAPOT



alamy

Image ID: BY6MK  
www.alamy.com

Utilisateur : Pour conduire que faut-il connaître ?

Utilisateur avancé : Pour réparer ou "tuner" que faut-il connaitre ?

"Créateur" : Pour construire un nouveau véhicule, que faut-il connaître ?





# Questions à se poser pour créer un framework à la Spring ?

Sans framework :

que réimplemente-t-on à chaque fois ?

quelles propriétés orthogonales "polluent" le code ?

comment gère-t-on la communication avec l'exterieur ?

Avec un framework :

Comment éviter les codes similaires ?

Peut-on les générer ?

Comment intégrer les propriétés orthogonales dans le code ?

Peut-on les "tisser" ?

Comment gérer les IHM, les bases de données, les systèmes externes ?

Peut-on être ouvert et faciliter l'hétérogénéité ?

# ZOOM SUR LA LOGIQUE METIER ET SON ÉCOSYSTÈME

---



alamy

Image ID: B8M8K  
www.alamy.com

# De nouveaux concepts vraiment ?

Composants - isolation des logiques métiers et construction par assemblage

Conteneurs – gestion automatique du cycle de vie

et des propriétés orthogonales au métier

- Annotations – enrichir le code avec des propriétés orthogonales
- Cycle de vie : gestion de la création, suppression, état des composants
- Interopérabilité - communication indépendante des technos



**Interfaces vous  
avez dit  
INTERFACE ?**

**Interactions dans  
un graphe de  
composants**

# Communications entre objets

## Retour sur POO et COO

### Principes de POO

un objet = ? composant avec des interfaces fournies (javadoc) avec une interface requise cachée (méthodes définies dans les autres classes nécessaires au fonctionnement)

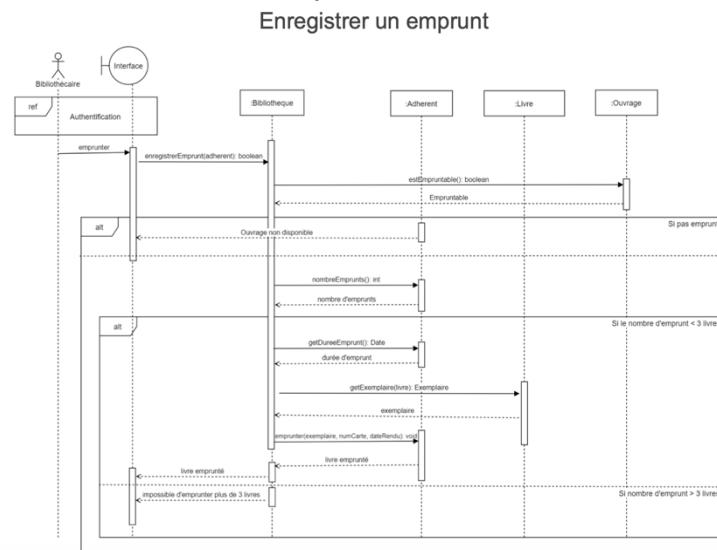
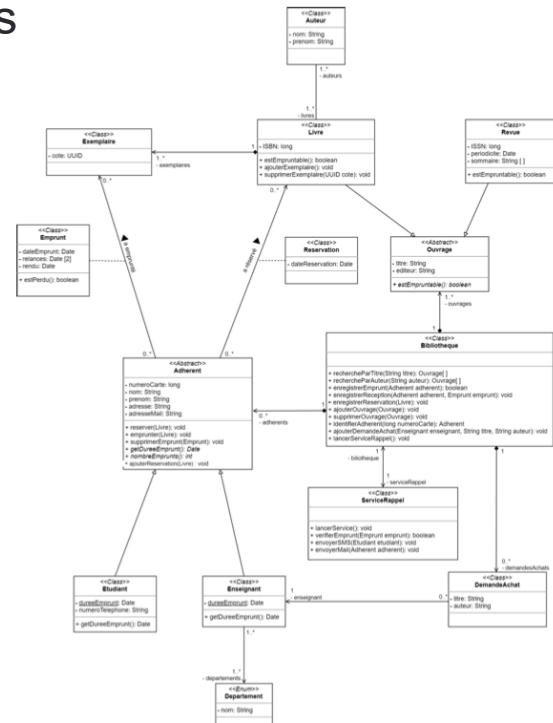


Diagramme de classes



Cohérence entre les méthodes du **diagramme de classes (interface fournie)** et les interactions du **diagramme de séquences (interface requise)** pour implémenter la logique métier

# Interfaces : cachées dans les outils

Utiliser et générer des documentation (Javadoc)

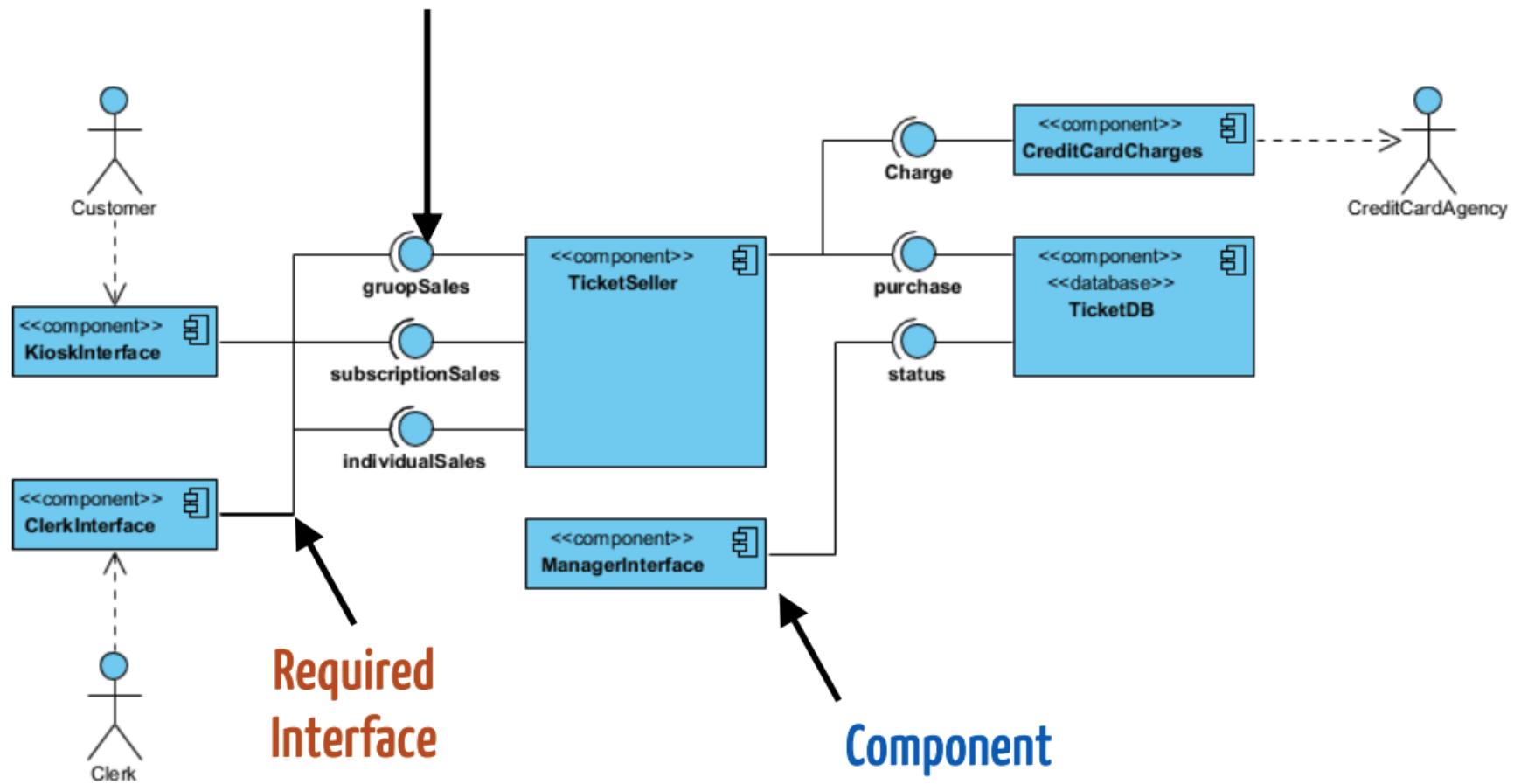


Utiliser des IDEs (Complétion, importation de packages à la main)...

Comprendre les messages d'erreurs à la compilation ou à l'exécution (ClassNotFoundException....)

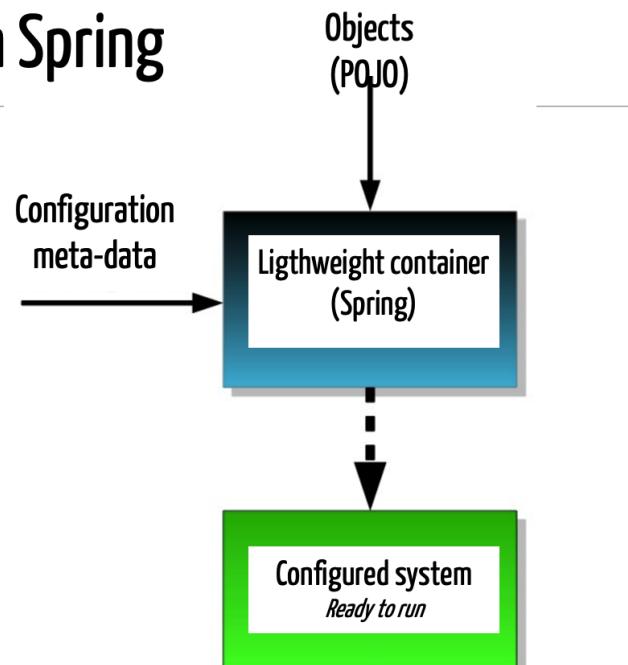
Comprendre la génération de code Copilot et GPT

## Provided Interface



# Interfaces à l'exécution

## IoC with Spring



Expliciter les interactions entre objets via les interfaces des composants

Injection des dépendances

Utilisation d'une factory et Des configurations



# Dans le code

## @Autowired

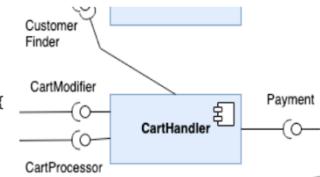
- Automatic dependency injection
- 2 annotations perform the same function
  - @Autowired : Spring annotation
  - @Inject : JSR-330 annotation
- Spring supports both
  - @Autowired has a property (required) that the @Inject annotation does not. This is useful to indicate the optional nature of an injection.
- Resolution by type, based on the object type to find the injected dependency

```
@Service
public class CartHandler implements CartModifier, CartProcessor {
    private static final Logger LOG = LoggerFactory.getLogger(CartHandler.class);

    private final Payment payment;

    private final CustomerFinder customerFinder;

    @Autowired
    public CartHandler(Payment payment, CustomerFinder customerFinder) {
        this.payment = payment;
        this.customerFinder = customerFinder;
    }
}
```



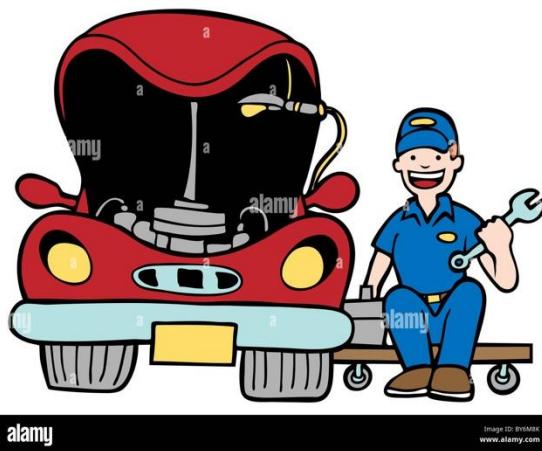
## All-in-One (in Spring)

Component	Provided Interfaces
<pre>@Component</pre>	<pre>public class CartHandler implements CartModifier, CartProcessor {</pre>
	<pre>CustomerRepository customerRepository;</pre>
Required Interfaces	
<pre>@Autowired</pre>	<pre>Payment payment;</pre>
<pre>@Autowired</pre>	<pre>public CartHandler(CustomerRepository customerRepository, Payment payment) {</pre>
	<pre>    this.customerRepository = customerRepository;</pre>
	<pre>    this.payment = payment;</pre>
	<pre>}</pre>



# ZOOM SUR L'INTEROPERABILITE (CLI, SERVICES EXTERNES)

---



...



# Concepts déjà étudiés ?

Endpoints, Services Web, RFC services réseau

Couches réseau : marshalling, sérialisation

Génération de code : interprétation compilation  
(chargement dynamique de classes)





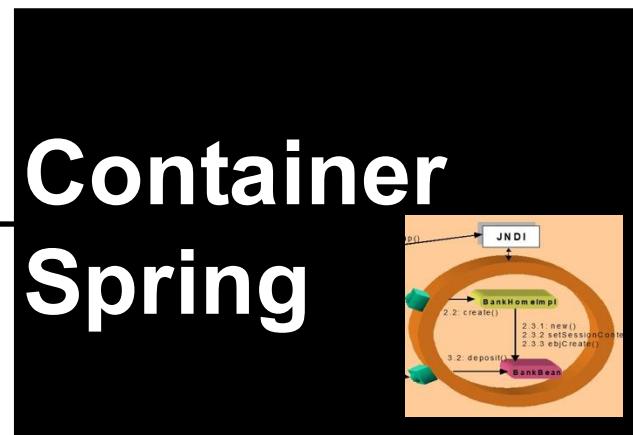
# Interfaces la clé de l'interopérabilité

COMMUNICATION  
AVEC  
L'EXTERIEUR

# Communication - Interopérabilité



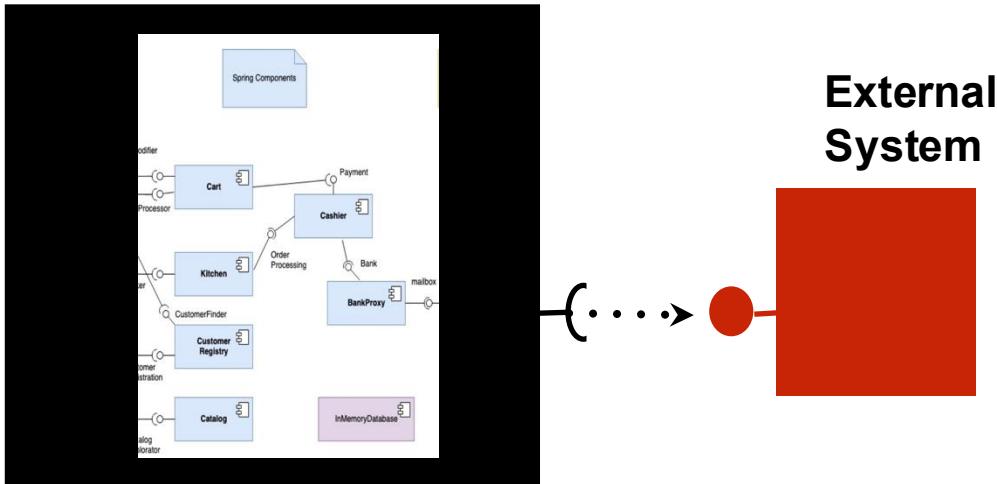
Client



External System

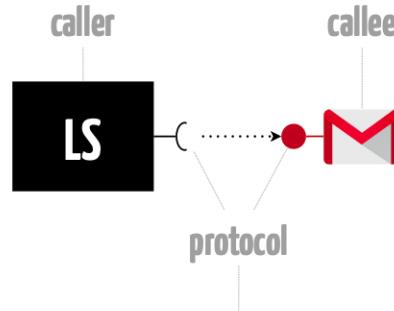
JPA  
CRUD et  
requêtes basiques

# Interoperabilité : le système externe expose son contrat



External System

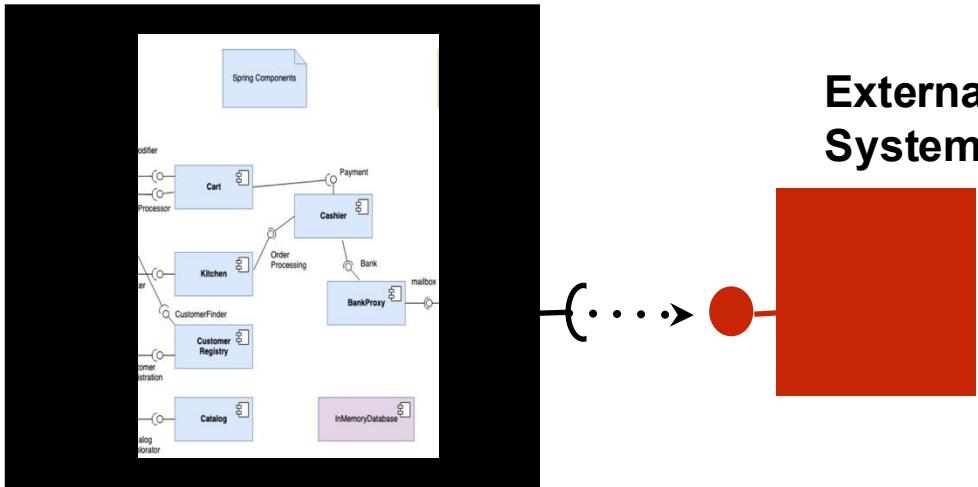
Abstracting from Implementation



- Endpoint: Where, How ?
- Operations: Why ?
- Business Object: What ?

# Interoperabilité : système externe

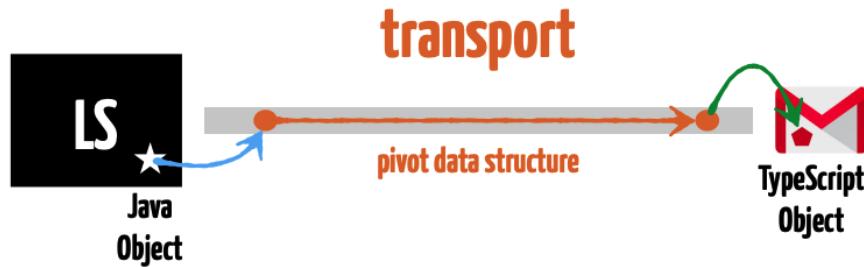
## Transport et sérialisation



External  
System

marshalling:  
Object → Pivot

unmarshalling:  
Pivot → Object



# COMMUNICATION ET INTEROPERABILITE -

OBJETS sur des machines différentes

---

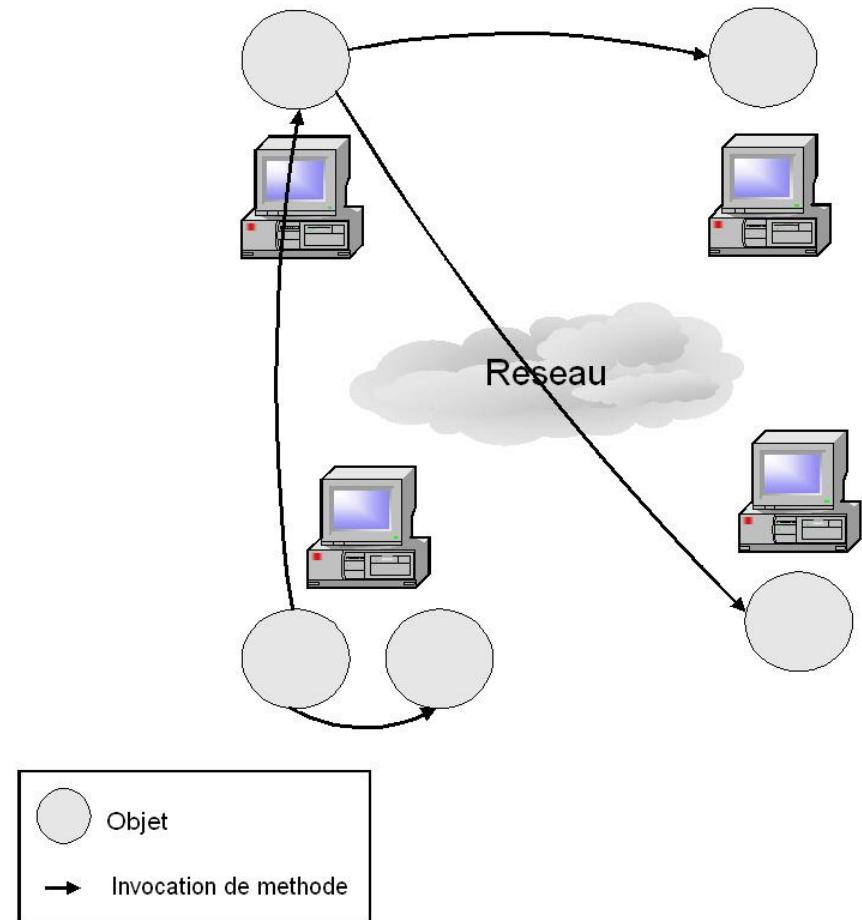


# Exemple RMI : abstraire l'Invocation de méthodes distantes

---

Mécanisme qui permet à des objets localisés sur des machines distantes de s'échanger des messages (invoyer des méthodes)

Au lieu de tout implémenter



# LA BASE : COMMUNICATION CLIENT SERVEUR



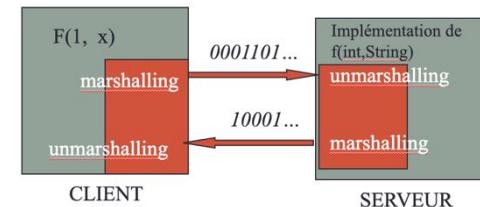
## 1/ Protocole d'application

Exemple : le Protocole SMTP,  
RFC1822/3

HELO  
MAIL From: [pinna@essi.fr](mailto:pinna@essi.fr)  
RCPT To: [pinna@essi.fr](mailto:pinna@essi.fr)  
DATA  
From: [pinna@essi.fr](mailto:pinna@essi.fr)  
Subject: Qui est là ?\n";  
"Vous suivez toujours ?  
QUIT

Marshalling : préparer le format et le contenu de la trame réseau

Unmarshalling : reconstruire l'information échangée à partir de la trame réseau

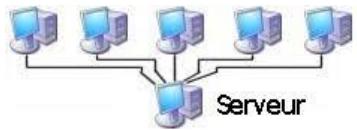


Comment cela fonctionne au niveau du réseau ?



## 3/ Communication réseau

- Identification de la machine qui abrite le serveur par le client
- Identification du serveur sur la machine
- Canal de communication entre le serveur et le client
- Construction de la trame réseau
- Echange du protocole d'application au dessus d'un protocole de transport



# Les données transmises

---

Dans le cas d'un appel de méthodes à distance  
On sera souvent amenés à passer des paramètres aux méthodes distantes...

Et les méthodes distantes peuvent retourner une valeur ou lever une exception...

On a deux types de paramètres

- Les objets locaux
- Les objets distants

# Passage de paramètres



Certains objets sont copiés (les objets locaux), d'autres non (les objets distants)

Les objets distants, non copiés, résident sur le serveur

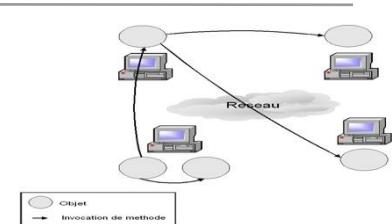
Les objets locaux passés en paramètre doivent être sérialisables afin d'être copiés, et ils doivent être indépendants de la plate-forme



Que passer comme paramètres dans nos interfaces vers l'extérieur du système ????  
Des identifiants, des objets, des DTO ?

# Que doit connaître un client ?

Le nom et le service de nommage



Lorsqu'un objet est passé au client soit comme paramètre soit comme valeur de retour, le client doit avoir le stub associé

les classes des paramètres, des valeurs de retour et des exceptions doivent aussi être connues...

Le programme client doit connaître la **classe** des stubs associés



**Et vous ? Le rôle des contrôleurs ?**

# Conception Logicielle

## Code "récurrent / polluant" ?

---

### Mise en place de l'IHM

Accès aux services à distance

Ecriture des End Points : API

### Exigences

Gestion ADHOC des requêtes

==> Parsing des URLs

Gestion des \*\*erreurs et réponses\*\*

Structuration nécessaire pour séparer la logique métier du workflow

# Solutions pour éviter les duplications de code

---

Démarche RMI SOAP etc : Génération des échanges à partir de la description des services en **OpenAPI**

**Swagger** : <https://swagger.io/tools/swagger-codegen/>

<https://openapi-generator.tech/docs/generators/>

Pour Spring : <https://openapi-generator.tech/docs/generators/spring>

ou

Démarche **mini Framework** : Tisser du code avec des annotations à la façon de Spring

# Solution : génération de code

---

Démarche RMI SOAP etc : Génération des échanges à partir de la description des services en **OpenAPI**

## Swagger Codegen

Swagger Codegen can simplify your build process by generating server stubs and client SDKs for any API, defined with the OpenAPI (formerly known as Swagger) specification, so your team can focus better on your API's implementation and adoption.

# ZOOM SUR LES MOYENS MIS EN ŒUVRE

---

Comment insérer du code non fonctionnel à la logique métier ?

Injection de dépendances

Annotations

Cycle de vie



**Why** should we **write** piece of  
**codes** instead of **being lazy**  
and write **pieces of code** that  
**will write pieces of code** on  
our behalf

- Jean-Marc Jézéquel

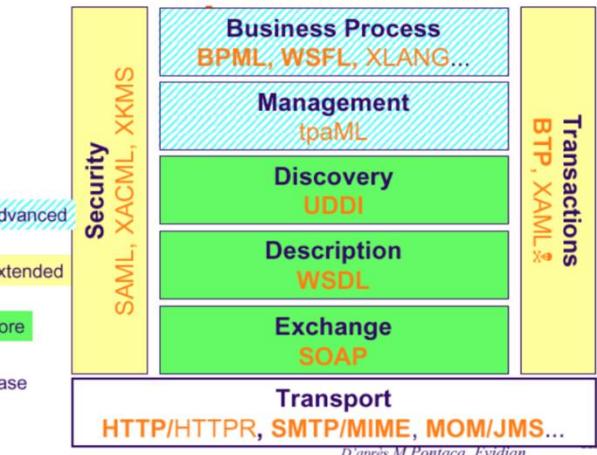
# MAGIE DES ANNOTATIONS PREOCCUPATIONS ORTHOGONALES AOP

# De quoi a-t-on besoin hors de la logique métier ?

- De retrouver les éléments (service de nommage)
- De la sécurité (service de sécurité)
- De transactions...
- **De la persistance**

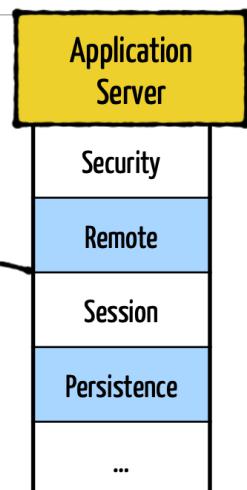
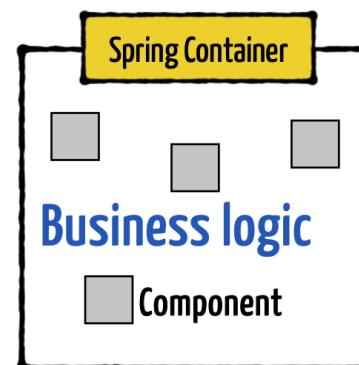


Comment les intégrer à la logique métier ?



## Spring infrastructure

### Life cycle management



System level services

# Intercepting messages

Un concept qui date : **Aspect Oriented Programming** (Xerox Parc, Gregor Kickzales)

AspectJ extension de Java. IBM en 2001 propose HyperJ.

<https://eclipse.dev/aspectj/>

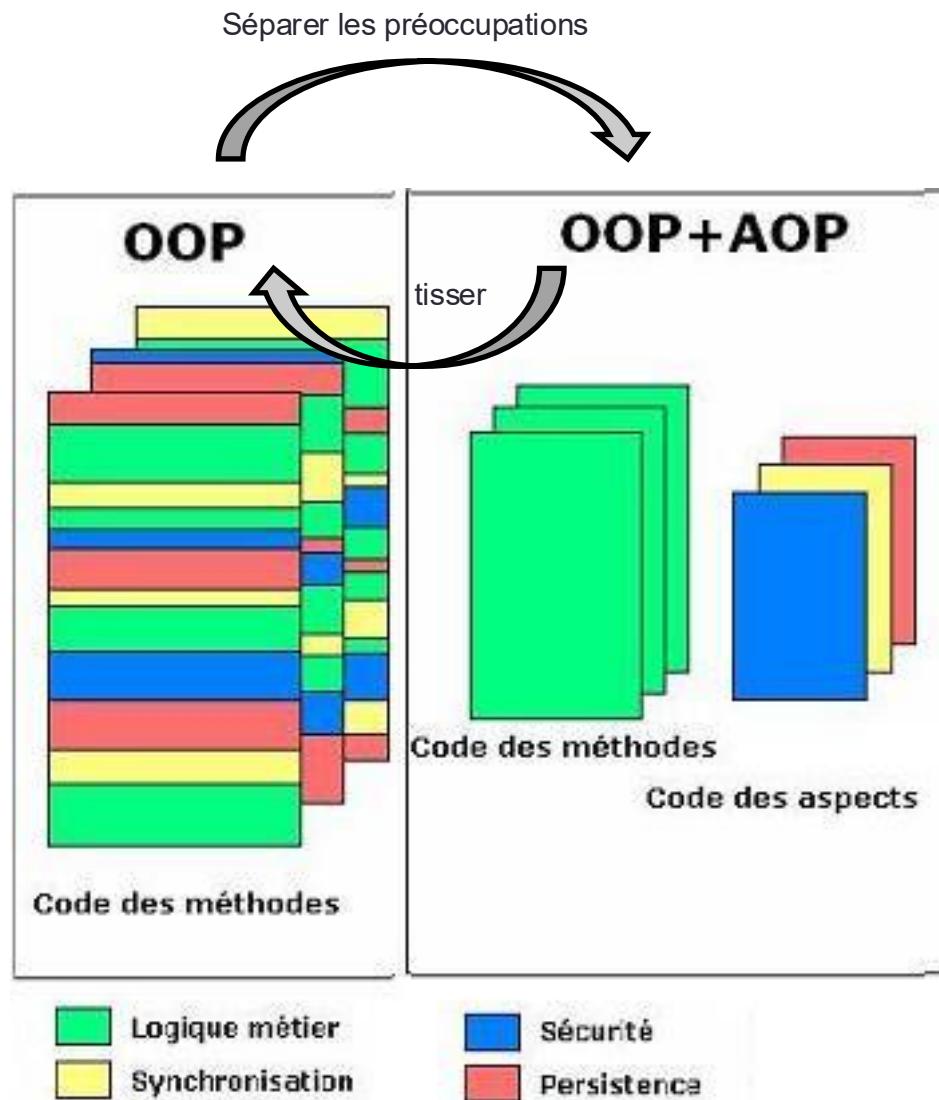
Augmenter la modularité

Séparer des préoccupations orthogonales au code

Ne pas polluer la logique métier par des comportements relatifs à des préoccupations fonctionnelles différentes :

log, sécurité, persistance....

# Aspect Oriented Programming : Principes



# A.O.P.

## Concepts

### Définition d'Annotations

Exemple un **advice** prenant en charge une préoccupation : « *log* » peut être « insérée » au code existant à chaque appel d'accesseur en écriture (set\*)

**Cross-cutting concerns** : « fonctionnalités secondaires / préoccupations orthogonales » qui peuvent être partagées par plusieurs classes

**Advice** : code additionnel gérant un *cross-cutting concern* à appliquer au code métier existant.

**Pointcut** : le point d'exécution où le cross-cutting concern doit être appliqué, l'advice correspondant ajouté.

**Aspect** : advices et pointcuts.

**Weaver** : tissage des advices dans le code pour obtenir le code final

# Exemples

**Cross-cutting** concerns : log / trace

**Advice** : le code du log que l'on veut appliquer (afficher la valeur des paramètres ou des variables)

**Pointcut** : avant l'exécution et après l'exécution des accesseurs en écriture

# Spring : Annotations et Intercepteurs

Intercepteurs : support pour développer des préoccupations orthogonales (cross-cutting features) et diminuer la duplication de code au niveau du code métier.

Permettent d'intercepter  
des opérations,  
des services...

Pour insérer des pré ou post actions au code existant



# Exemples d'interceptions pour TCF

1. Suivre des exécutions : mise en place de Logs pré et/ou post action de quelles opérations ?
2. Intégrer des statistiques au code métier pré et/ou post action de quelles opérations ?
3. Vérifier les données reçues pré et/ou post action de quelles opérations ?

## Smarter Logging with AOP

simpleTCFS / backend / src / main / java / fr / univcotedazur / simpletcfs / aspects / ControllerLogger.java

### Step 2: Create an Aspect component (and that's all)

```
@Aspect
@Component
public class ControllerLogger {

    private static final Logger LOG = LoggerFactory.getLogger(ControllerLogger.class);

    @Pointcut("execution(public * fr.univcotedazur.simpletcfs.controllers..*(..))")
    private void allControllerMethods() {
        // This enables to attach the pointcut to a method name we can reuse below
    }

    @Before("allControllerMethods()")
    public void logMethodNameAndParametersAtEntry(JoinPoint joinPoint) {
        LOG.info("TCFS:Rest=Controller: {}:{} Called {} {}", joinPoint.getThis(), joinPoint.getSignature().getName(), joinPoint.getArgs());
    }
}
```

Aspect

Pointcut (what we intercept)

Advice (what we do on the pointcuts)

# INTROSPECTION DE CODE

---

## JAVA REFLECT ?

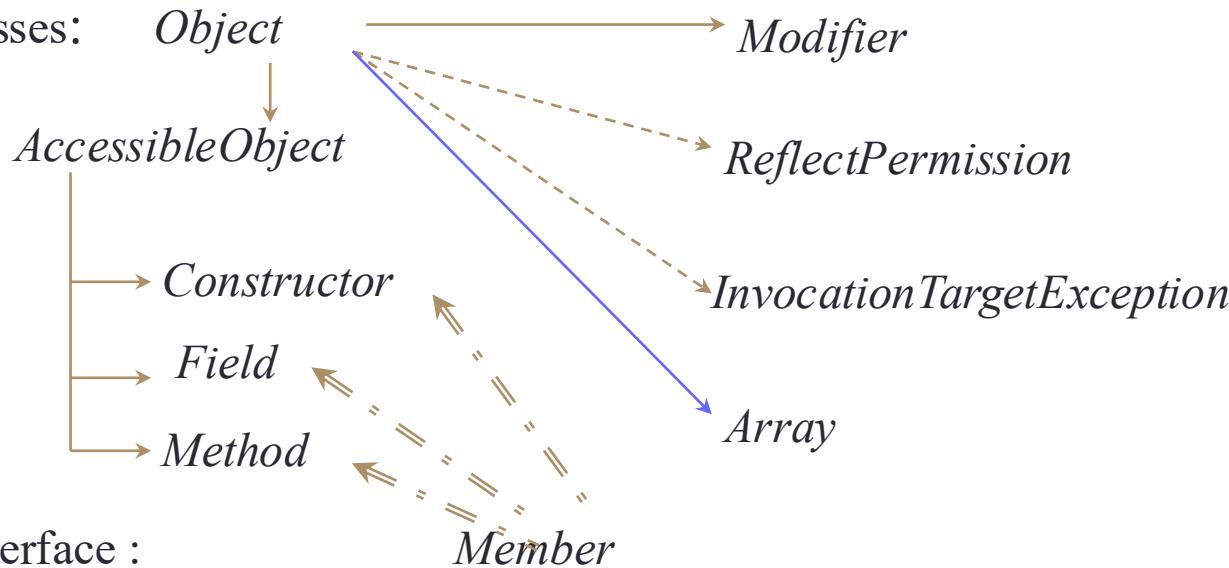


# Réflexivité en Java

La réflexivité en Java permet  
à un programme Java de s'examiner en cours d'exécution  
de manipuler ses propriétés internes.

`java.lang.reflect.*`

des classes:



une interface :

# Que peut-on faire ?

Obtenir des informations sur les classes

Simuler l'opérateur *instanceof*

Découvrir la liste et le descriptif des méthodes

Obtenir des informations sur les constructeurs

Avoir des informations sur les variables

Invoquer des méthodes, des constructeurs, affecter des variables

Créer des objets et des tableaux

**dynamiquement** lors de l'**exécution** du programme

sans connaître à la compilation

le nom et les arguments d'une méthode / constructeur

le nom de la variable

le type des objets, des tableaux....

Utilisation connue de la réflexivité : l'édition sous Eclipse IntelliJ

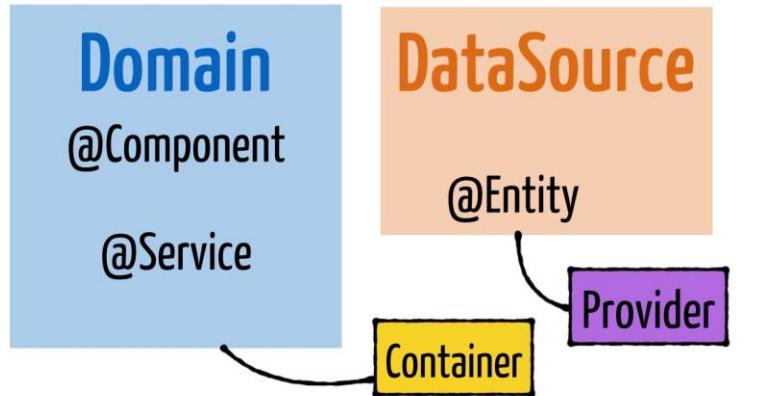
L'outil utilise la réflexivité pour obtenir liste des méthodes publiques qui peuvent être envoyées à une instance d'une classe

Sérialisation à partir de la signature des interfaces

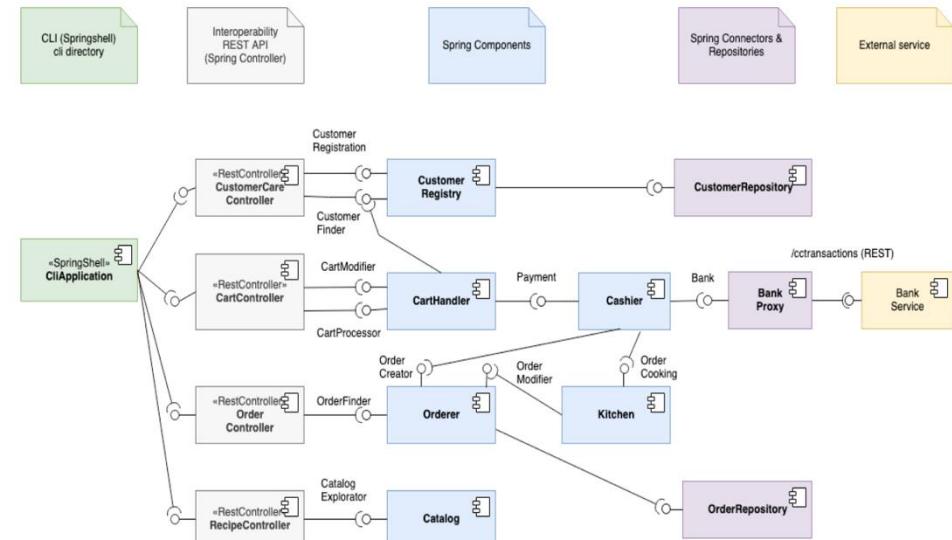
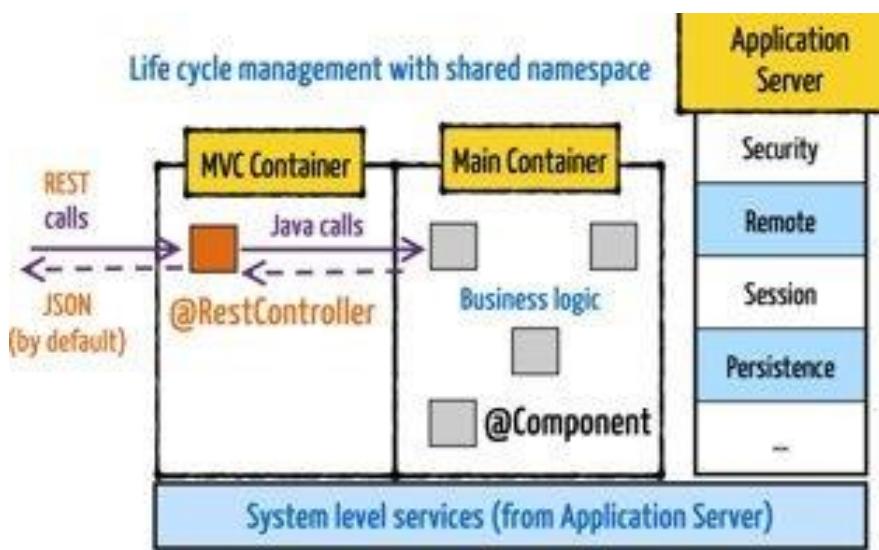
TOUT CELA POUR QUOI ?



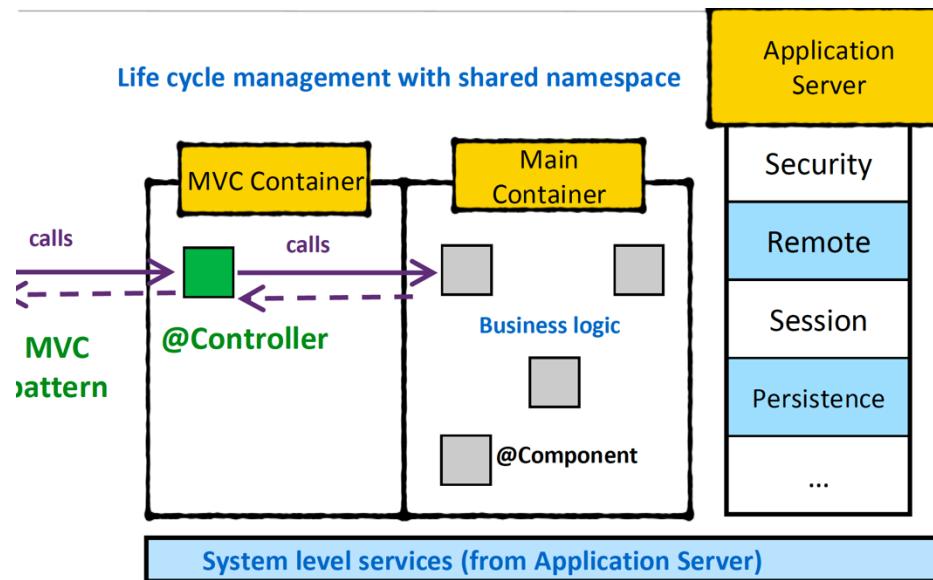
# Concevoir à différents niveaux de composants



20

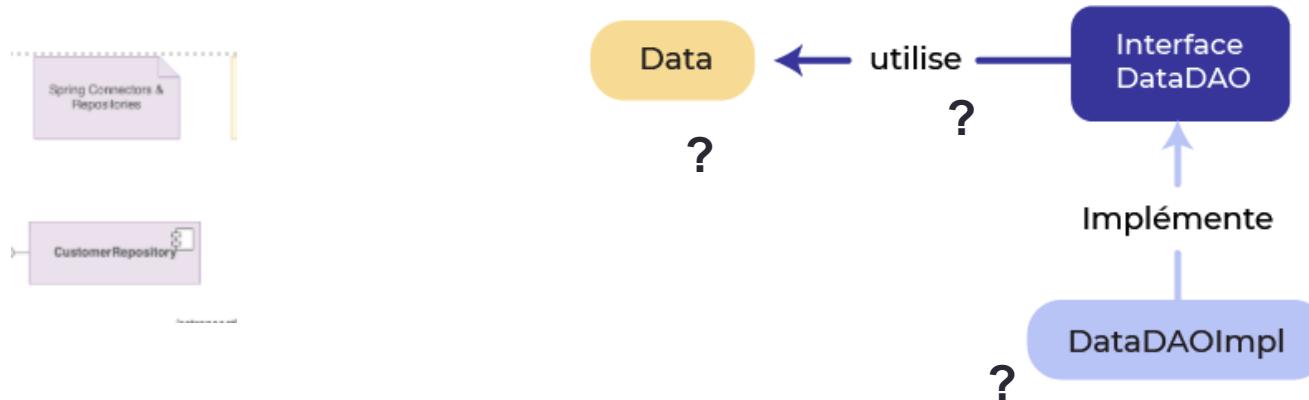


# Voir la persistance comme une propriété orthogonale



pour stocker et accéder aux données

# Utiliser les repositories pour accéder à la base de données



<https://openclassrooms.com/fr/courses/6982461-utilisez-spring-data-pour-interagir-avec-vos-bases-de-donnees/7201184-implementez-vos-entites-et-les-interfaces-repository>

BDD créée à partir des entités ou le contraire

# Dans le code

## Provided Interfaces

The component is very basic, still we apply interface segregation with two interfaces:

- CartModifier : operations to modify a given customer's cart, like adding or removing cookies, and to retrieve the contents of the cart;

```
public interface CartModifier {  
  
    Item update(Long customerId, Item it) throws NegativeQuantityException, CustomerIdNotFoundException;  
  
    Set<Item> cartContent(Long customerId) throws CustomerIdNotFoundException;  
}
```

- CartProcessor : operations for computing the cart's price and validating it to process the associated order;

```
public interface CartProcessor {  
  
    double cartPrice(Long customerId) throws CustomerIdNotFoundException;  
  
    Order validate(Long customerId) throws PaymentException, EmptyCartException, CustomerIdNotFoundException;  
}
```

```
@Component  
public class BankProxy implements Bank {  
  
    @Value("${bank.host.baseurl}")  
    private String bankHostandPort;  
  
    private RestTemplate restTemplate = new RestTemplate();  
  
    @Override  
    public boolean pay(Customer customer, double value) {  
        try {  
            ResponseEntity<PaymentDTO> result = restTemplate.postForEntity(  
                bankHostandPort + "/cctransactions",  
                new PaymentDTO(customer.getCreditCard(), value),  
                PaymentDTO.class  
            );  
            return (result.getStatusCode().equals(HttpStatus.CREATED));  
        } catch (HttpClientErrorException errorException) {  
            if (errorException.getStatusCode().equals(HttpStatus.BAD_REQUEST)) {  
                return false;  
            }  
            throw errorException;  
        }  
    }  
}
```

**@Value injection + property (can be changed for tests...)**

**RestTemplate**

**Plenty of methods for all http verbs (here POST)**

**Payment DTO created**

**Payment DTO returned (and ResponseEntity variant used to check the status Code)**

# Dans le code

```
@Component
public class BankProxy implements Bank
    @Value("${bank.host.baseurl}")
    private String bankHostandPort;

    private RestTemplate restTemplate = new RestTemplate();

    @Override
    public boolean pay(Customer customer, double value) {
        try {
            ResponseEntity<PaymentDTO> result = restTemplate.postForEntity(
                UN: bankHostandPort + "/cctransactions",
                new PaymentDTO(customer.getCreditCard(), value),
                PaymentDTO.class
            );
            return (result.getStatusCode().equals(HttpStatus.CREATED));
        }
        catch (HttpClientErrorException errorException) {
            if (errorException.getStatusCode().equals(HttpStatus.BAD_REQUEST)) {
                return false;
            }
            throw errorException;
        }
    }
}
```

**@Value injection + property  
(can be changed for tests...)**

**Calling a REST service**

**RestTemplate**

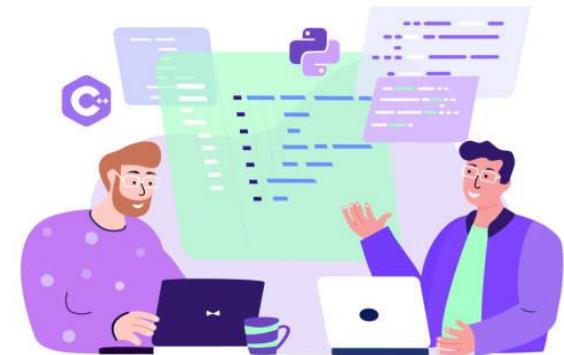
**Plenty of methods for all http  
verbs (here POST)**

**Payment DTO  
created**

**Payment DTO returned (and ResponseEntity  
variant used to check the status Code)**

# SPRING POUR UN UTILISATEUR : EN ENTREPRISE

---



Onboarding en entreprise

La configuration de Spring est faite et le développeur développe uniquement la partie serveur

# SPRING POUR UN UTILISATEUR ISA

---



shutterstock.com - 2389356043

Configuration TCF donnée

La configuration doit être connue et comprise par l'étudiant

<https://github.com/CookieFactoryInSpring/simpleTCFS/blob/main/docker-compose.yml>

<https://github.com/CookieFactoryInSpring/simpleTCFS/blob/main/cli/pom.xml>

# Spécificités de la configuration

```
# the postgres DB to be connected to the backend (watch out: no volume specified, everything can be lost)
postgres:
  image: postgres:17.2
  container_name: db
  environment:
    - POSTGRES_PASSWORD=postgrespass
    - POSTGRES_USER=postgresuser
    - POSTGRES_DB=tcf-db
  restart: always
  healthcheck:
    test: [ "CMD-SHELL", "pg_isready -d tcf-db -U $$POSTGRES_USER" ]
    interval: 5s
    timeout: 3s
    retries: 3
    start_period: 5s
```

Docker compose démarrant la base de données

```
# postgres DB # IN DOCKER COMPOSE SHOULD BE OVERRIDEN BY ENV VARIABLES

# POSTGRES_USER
spring.datasource.username=postgresuser
# POSTGRES_PASSWORD
spring.datasource.password=postgrespass
spring.datasource.url=jdbc:postgresql:// ${POSTGRES_HOST}/tcf-db
spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.show-sql=true
spring.jpa.generate-ddl=true

spring.jpa.open-in-view=false
```

Configuration de spring grâce au fichier persistence.properties

# SPRING POUR UN UTILISATEUR AVANCÉ



Savoir configurer

Savoir "tuner" les services orthogonaux

Persistiance, hétérogénéité...

Savoir réagir au passage à l'échelle

# EXEMPLE : PERSISTANCE

---

Pourquoi la configuration ISA ?

# Méthodes transactionnelles : pourquoi et où ?

spring.jpa.open-in-view=false

The screenshot shows the Spring Initializr web application. On the left, there are sections for Project (Gradle - Groovy, Gradle - Kotlin, Maven), Language (Java, Kotlin, Groovy), and Spring Boot (3.5.0 (SNAPSHOT), 3.5.0 (M1), 3.4.3 (SNAPSHOT), 3.4.2, 3.3.9 (SNAPSHOT), 3.3.8). The 'Spring Web' dependency is selected under Dependencies. Project Metadata includes Group (fr.univ-cotedazur.polytech), Artifact (demo-osiv), Name (demo-osiv), Description (Dummy project to demonstrate the default OSIV behavior), Package name (fr.univ-cotedazur.polytech.demo-osiv), Packaging (Jar), Java version (23), and Java modules (21, 17). A large button labeled 'GENERATE CTRL + ⌘' is at the bottom. Below the interface, the generated code is shown in a terminal-like window:

```
2025-02-12T22:52:11.838+01:00  WARN 27592 --- [demo-osiv] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2025-02-12T22:52:11.996+01:00  INFO 27592 --- [demo-osiv] [main] f.u.p.demo_osiv.DemoOsivApplication : Started DemoOsivApplication in 1.43 seconds (process running for 1.605)
2025-02-12T22:52:12.000+01:00  INFO 27592 --- [demo-osiv] [main]
```

A red arrow points from the text "Par défaut les opérations à partir du controller sont transactionnelles : bonne idée ?" to the warning message in the terminal output.

Par défaut les opérations à partir du controller sont transactionnelles : bonne idée ?

-> Pourquoi faire remonter cet aspect aux composants métiers mais pas aux controllers ?

spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning

<https://www.baeldung.com/spring-open-session-in-view>

# EXEMPLE INTEROPERABILITE

---

**Comportement ISA : API REST**

Quelles sont les autres possibilités ?

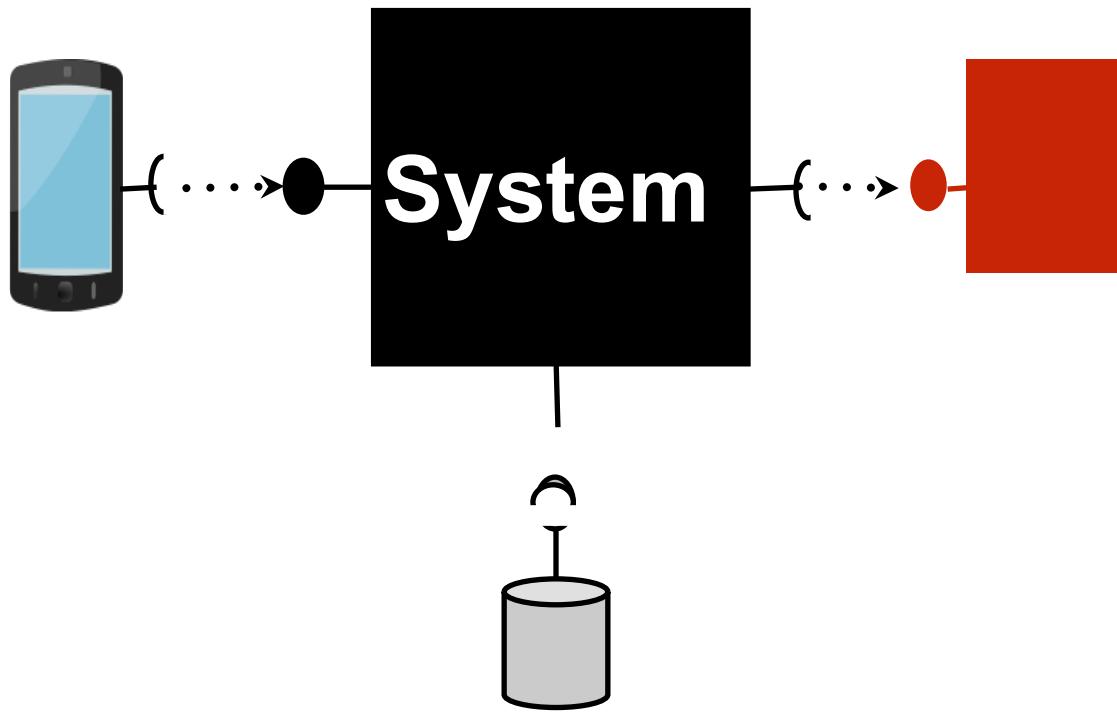
# Illustration par l'interopérabilité

---

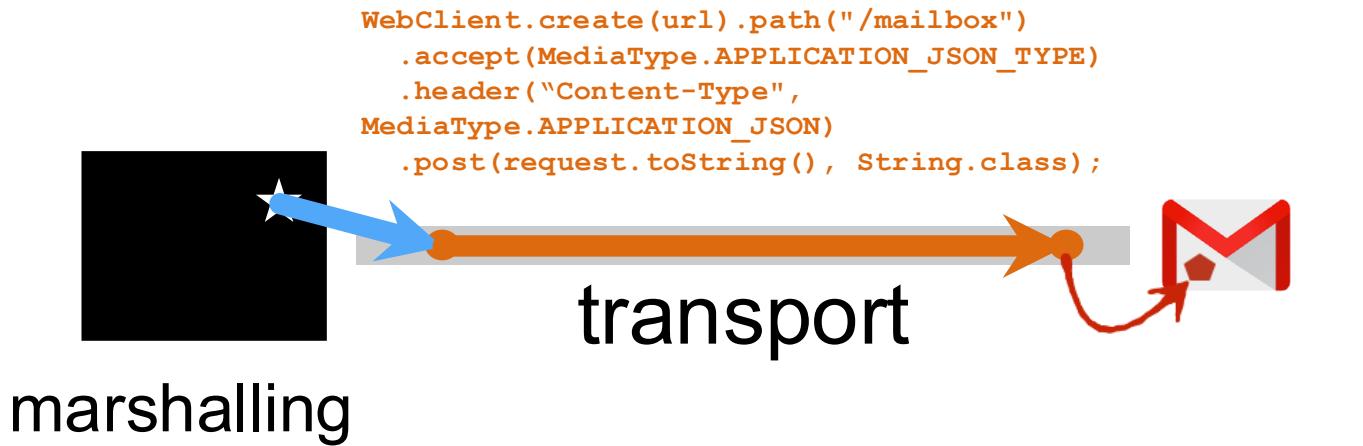


REST API

?



# The All Together

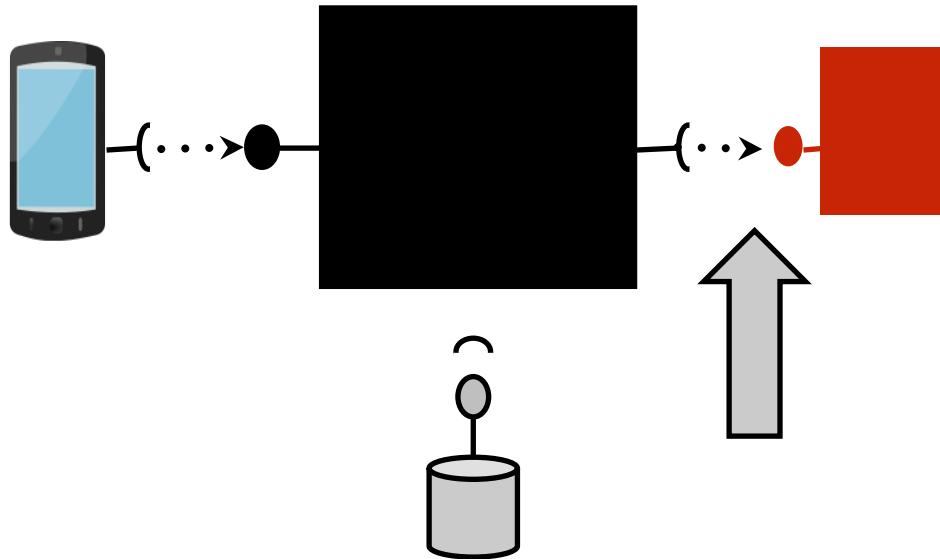


```
JSONObject request =
new JSONObject()
.put("CreditCard", customer.getCreditCard())
.put("Amount", value);
```

```
[WebInvoke( Method = "POST", UriTemplate = "mailbox",
RequestFormat = WebMessageFormat.Json,
ResponseFormat = WebMessageFormat.Json) ]
```

# Interoperabilité : le système externe expose son contrat

---



Remoting technologies:

- *Remote Method Invocation (RMI)*.
- *Spring's HTTP invoker*.
- *JAX-RPC*. Spring provides remoting support for web services via JAX-RPC (J2EE 1.4's web service API).
- *JAX-WS*. Spring provides remoting support for web services via JAX-WS (the successor of JAX-RPC, as introduced in Java EE 5 and Java 6).
- *JMS*. Remoting using JMS

# Contrat fort

## Nothing new : Java2WSDL (Cours SoC)

```
@WebService
public interface CartWebService {
    @WebMethod
    void addItemToCustomerCart(
        @WebParam(name = "customer_name") String customerName,
        @WebParam(name = "item") Item item
    ) throws UnknownCustomerException;

    @WebMethod
    void removeItemToCustomerCart(
        @WebParam(name = "customer_name") String customerName,
        @WebParam(name = "item") Item item
    ) throws UnknownCustomerException;

    @WebMethod
    @WebResult(name = "cart_contents")
    Set<Item> getCustomerCartContents(
        @WebParam(name = "customer_name")
    ) throws UnknownCustomerException;

    @WebMethod
    @WebResult(name = "order_id")
    String validate(@WebParam(name = "order_id") String orderId)
        throws PaymentException, EmptyCartException;
}
```

# Web Service Description Language



```
<wsdl:portType name="CartWebService">
    <wsdl:operation name="addItemToCustomerCart">
        <wsdl:input message="ns1:addItemToCustomerCart"
            name="addItemToCustomerCart" />
        <wsdl:output message="ns1:addItemToCustomerCartResponse"
            name="addItemToCustomerCartResponse" />
        <wsdl:fault message="ns1:UnknownCustomerException"
            name="UnknownCustomerException" />
    </wsdl:operation>
    ...
</wsdl:portType>
```

# C'est bien gentil mais demain ?



# Que disent ceux qui utilisent l'IA pour coder des projets Spring Boot ?



# Que disent ceux qui utilisent l'IA pour coder des projets Spring Boot ?



Then-Boat8912 · il y a 6 m.

J'ai essayé un peu en planche à voile. C'est pas mal pour certaines choses. Ça a souvent besoin d'être corrigé, mais ça peut aider à accélérer des trucs que tu aurais fait de toute façon.

Pour tester le code qu'il écrit, c'est juste correct. Pareil pour les fichiers de configuration yaml. Java dans les LLM, ça traîne pas mal de trucs du passé quand même.

↑ 3 ↓

issskk · il y a 6 m.

IntelliJ and Claude Code plugin in intelliJ. Usually work on a per feature basis and prompt everytyime to get the "bulk" of the work done and then I clean it up by hand after.



cielNoirr · il y a 6 m.

Google Jules, c'est plutôt sympa. C'est comme avoir un pote programmeur qui t'aide sur ton projet. Il se plaint parfois, mais c'est un bon point de départ.

⊖ ↑ 2 ↓



UnionSea2688 · il y a 6 m.

De quelle manière ? Genre, te filer le code ou t'aider à comprendre et vraiment écrire du code de qualité que tu "comprends" ?

⊖ ↑ 1 ↓



cielNoirr · il y a 6 m.

Si tu débutes, c'est probablement mieux pour toi d'apprendre d'abord les concepts de base de Spring Boot.

↑ 2 ↓

# Quel intérêt d'intégrer l'IA à Spring Boot ? SpringIA ?



# l'IA et SpringBoot ?

<https://www.skiils.com/blog/spring-boot-ia-generative-duo-innovant>

## Les briques technologiques de Spring AI

Au cœur de ce framework se trouvent plusieurs composants technologiques qui rendent son adoption rapide et performante. Parmi ces briques :

- **Compatibilité avec des bibliothèques de machine learning** : Spring AI facilite l'intégration avec des frameworks populaires comme TensorFlow et PyTorch, permettant aux développeurs de réutiliser des modèles déjà entraînés.
- **API génératives** : Il offre des connecteurs simplifiés pour interagir avec des services d'IA, comme OpenAI GPT, ce qui réduit la complexité d'implémentation.
- **Gestion des données** : Grâce à l'intégration avec des outils de gestion de flux de données tels que Kafka ou RabbitMQ, Spring AI garantit un traitement efficace des entrées et sorties des modèles d'IA.
- **Support natif des environnements cloud** : Avec des outils comme Spring Cloud, il devient aisément de déployer des solutions d'IA à l'échelle, sur AWS, Azure ou Google Cloud Platform.

## Exemples d'utilisation de Spring AI dans des applications Spring

Les cas d'utilisation de Spring AI sont nombreux et couvrent plusieurs secteurs :

- **E-commerce** : Génération de descriptions de produits personnalisées en fonction des préférences des clients.
- **Service client** : Mise en œuvre de chatbots avancés capables de comprendre les requêtes complexes et d'y répondre efficacement.
- **Santé** : Création de résumés de dossiers médicaux pour faciliter la prise de décision clinique.
- **Éducation** : Applications qui génèrent des exercices adaptés au niveau d'apprentissage des étudiants. Ces exemples montrent comment l'IA générative peut offrir des solutions innovantes dans des contextes variés.

## Limitations et défis de l'IA générative dans les applications Spring

Malgré ses avantages, l'utilisation de l'IA générative présente des défis. Les coûts liés à l'entraînement et au déploiement des modèles peuvent être élevés, en particulier pour des entreprises aux ressources limitées. Les préoccupations éthiques, telles que les biais dans les modèles ou l'utilisation abusive des données, nécessitent une attention particulière. Enfin, la complexité technique de l'intégration des modèles génératifs peut poser des problèmes aux équipes moins expérimentées, bien que Spring AI réduise ces barrières grâce à ses outils intuitifs.

# Peut-on créer un site web uniquement avec l'IA ?



# Peut-on créer un site web uniquement avec l'IA ?

---

<https://medium.com/@shubhamvartak01/can-ai-really-build-a-complete-website-a-java-spring-boot-developers-honest-perspective-a234e2ff10b7>

Partout où l'on regarde, le message est clair et net :

« L'IA peut créer des sites web. »

« L'IA remplacera les développeurs. »

« Il suffit de donner une instruction et votre application complète est prête. »

L'IA est extrêmement utile.

L'IA est puissante.

Mais l'IA ne peut pas, à elle seule, construire entièrement un site web réel.

<https://medium.com/@reyanshicodes/ai-generated-spring-boot-code-what-works-and-what-fails-miserably-acd5209787d0>

<https://www.impulse-web.fr/blog/avis-creation-site-internet-ia/>

<https://www.web-leader.fr/articles-de-blog/utiliser-lia-pour-faire-son-site-web-bonne-ou-mauvaise-idee>

MERCIA TOUS



ALORS Utilisateur ? Utilisateur avancé ? Et ou créateur ?