haute école
neuchâtel berne jura
arc ingénierie
www.he-arc.ch
Hes·so
Certifié ISO 9001

Java Enterprise Edition (JEE)

Projet : Architecture et structure de projet

Sébastien Chèvre

- Sémantique des diapositives

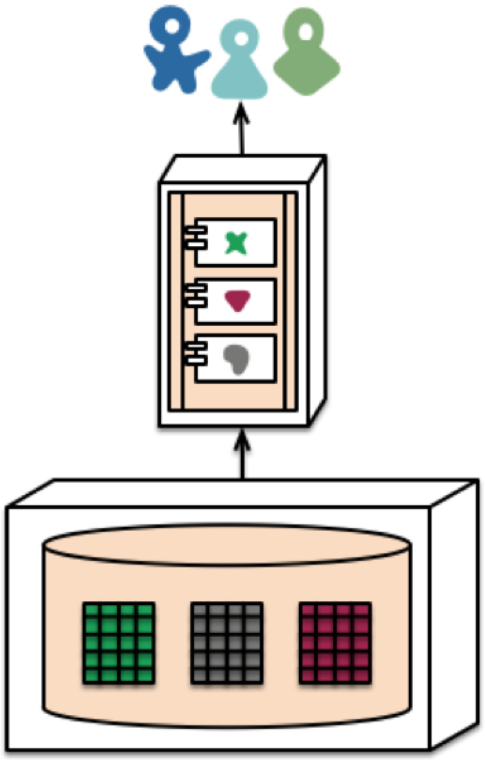
À savoir théoriquement (TE)

Sensibilisation et aspect pratiques

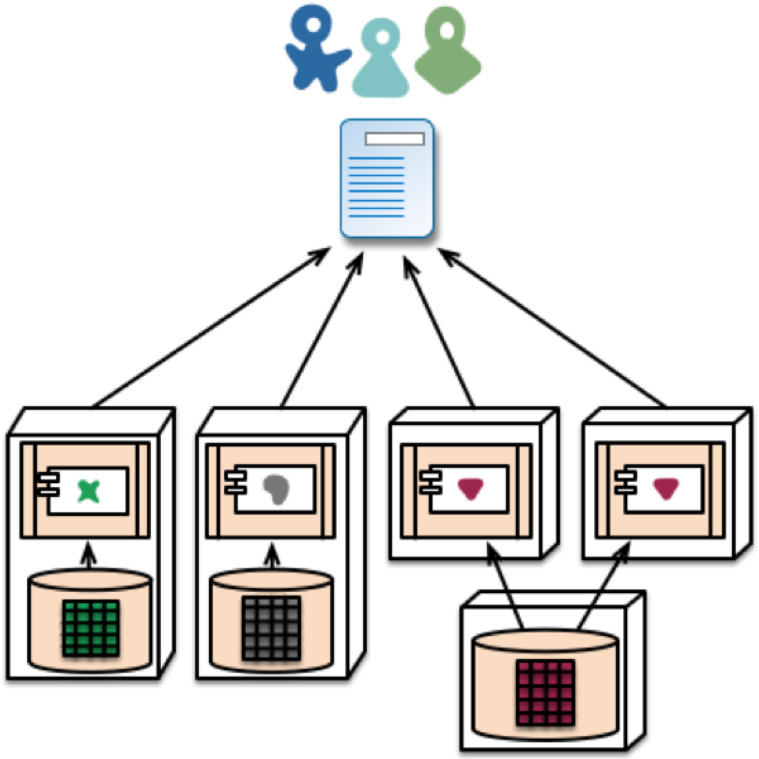
Architecture - Structure

- Pourquoi structurer une application?
  - **Pour documenter** (reflète les concepts d'architecture)
  - **Pour permettre une modularisation**
    - Fonctionnelle
    - Technique
  - **Pour organiser une application**

Rappel : typologie d'architecture



Monolithe



Distribuée

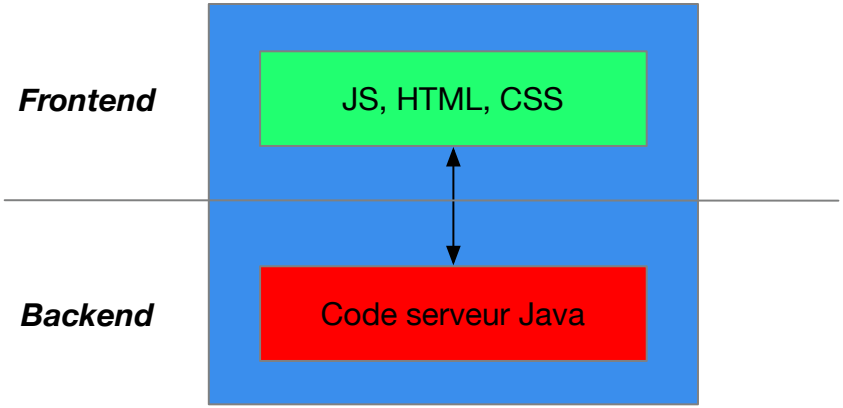
Structure d'une application (service)

- Un système distribuée peut être vu comme un ensemble de service « monolithe »
- Le focus sera donc mis sur la structure d'une application monolithique
  - Qui pourrait également être un service autonome, composant un système applicatif orienté service

Terminologie : backend - frontEnd

La **partie frontend** est la partie visible à l'utilisateur finale (la vue). Elle doit contenir le moins de code métiers possible. Dans l'idéal, la partie frontend se charge uniquement de la mise en forme visuelle des données et des aspects UX/UI*

La **partie backend** est le cœur du système. C'est cette partie qui va gérer les règles métiers, l'accès aux données, la validation



UX: user expérience
UI: user interface

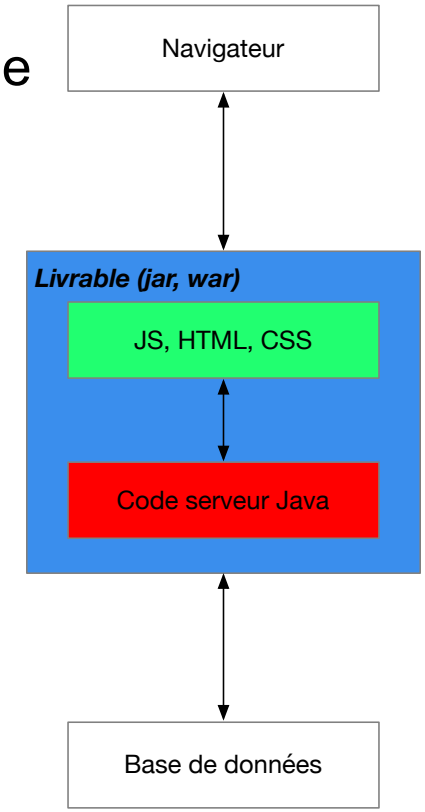
Démarrage d'un projet

- Questions à se poser:
  - Monolithe complet ?
  - Ségrégation frontend – backend ?
  - Orienté service ?
  - Micro-services ?

Structure d'une application

Monolithe complet

- **Monolithe complet**
  - le *frontend* est embarqué dans le monolithe
  - JSP, JSF
  - Moteur de template
  - Framework JS SPA* embarqué
  - Un monolithe à l'exécution peut toutefois être construit de manière modulaires
    - Utilisation de module maven

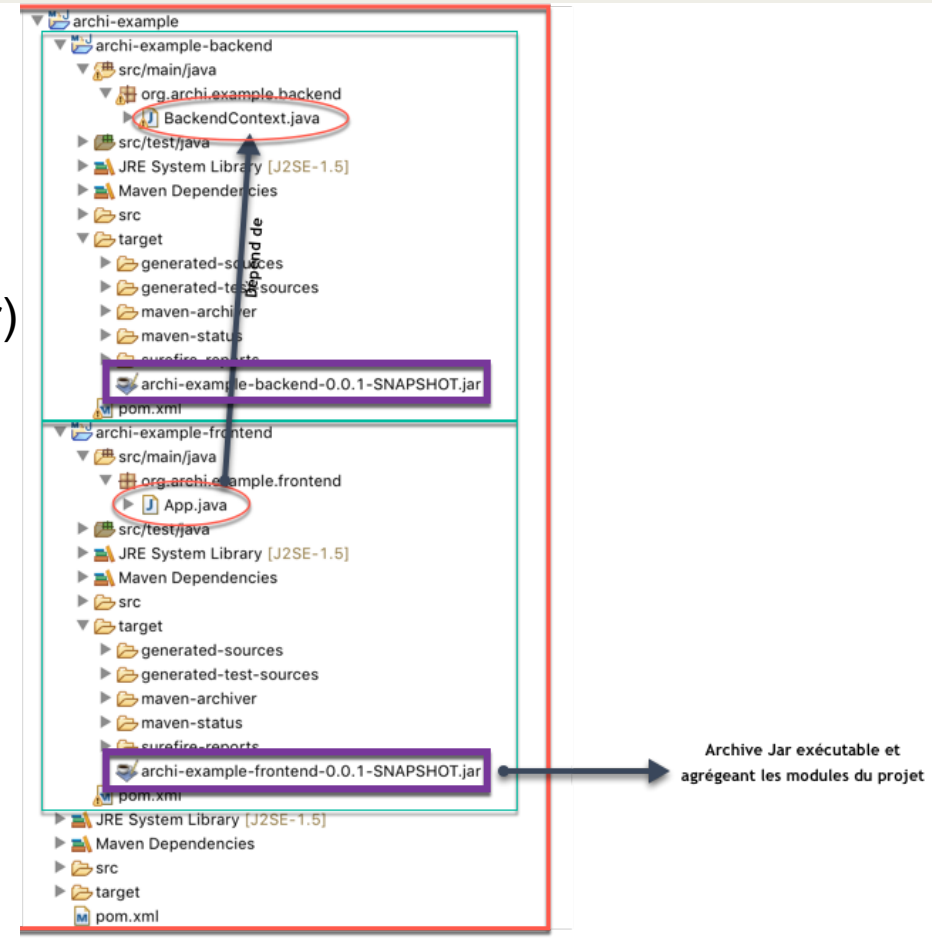


SPA: Single Page Application

Structure d'une application

Monolithe complet

- **Construction modulaire**
  - Maven permet la structuration d'un projet Java (jar, war) en module
  - Un module parent (appelé le POM parent) gère la construction des différents modules enfant



Structure d'une application

Monolithe complet

- **Avantages**
  - 1 seul livrable
  - Tout est géré par un serveur
  - Simple
  - Un seul container d'exécution (JVM, serveur applicatif)
  - Gestion transactionnelle simple

Structure d'une application

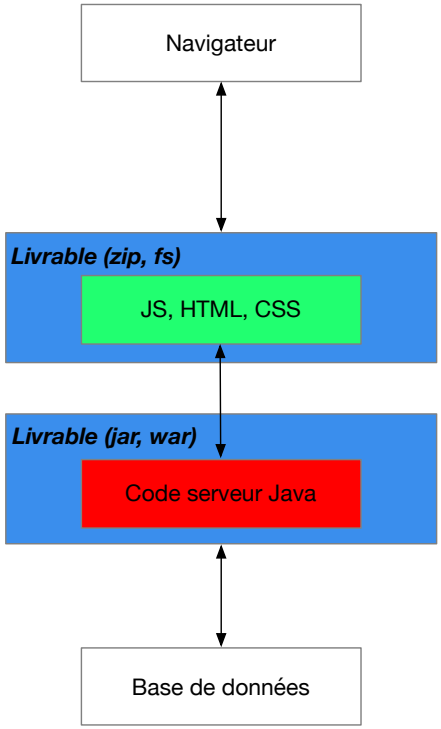
Monolithe complet

- **Inconvénients**
  - Reconstruction complète du produit à chaque modification
  - Peu flexible
  - L'application n'est pas utilisable en mode distribuée (plus compliqué)
  - Avec l'utilisation de JSP, ou moteur de template le code de la vue est fortement dépendant du code serveur
  - Pas très adapté au travail en équipe (les compétences *frontend* sont différentes des compétences *backend*)
  - Dans le cas d'une SPA, on fait travailler ensemble deux technologies différentes
  - Très compliqué à faire évoluer à partir d'un certain volume de code

Structure d'une application

Ségrégation *frontend-backend*

- **Ségrégation frontend-backend**
  - Framework JS SPA
    - Angular, React, Vue, etc…
  - Autres clients
    - Tout client supportant le protocole http
      - CLI (cUrl)
      - Client REST



Solution ségrégation front-back

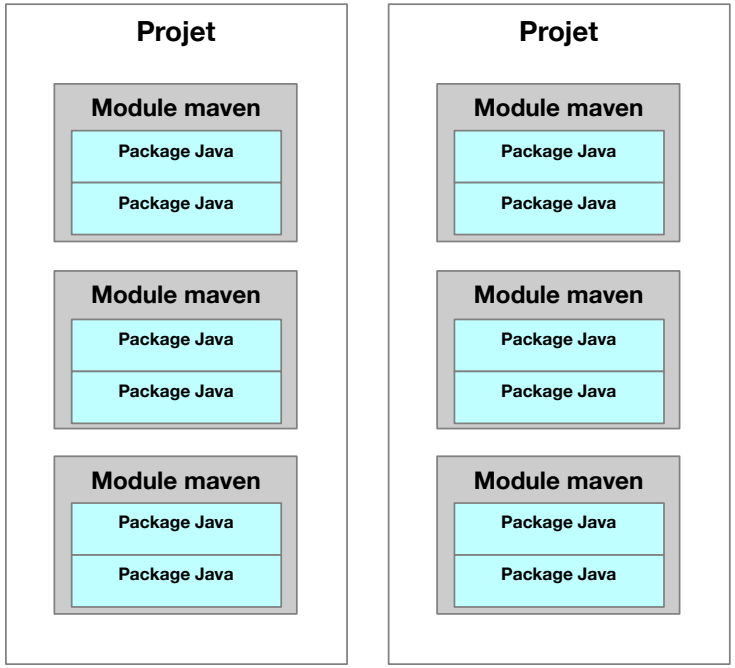
- **Avantages**
  - Flexibilité de déploiement
    - le *frontend* peut être redéployée indépendamment du *backend* et inversement
  - Le *backend* ne fourni que des données via des api
    - REST, avec json, xml, JMS, etc…
  - Le *backend* peut être testé et piloté indépendamment du *frontend*
  - Pas de dépendance du *frontend* envers le *backend*
    - dépendances fonctionnelles uniquement
  - Le *backend* peut supporter n'importe quel client utilisant les même protocoles
    - cUrl
    - Client REST graphique
    - Etc…
  - Le *backend* est réutilisable dans un contexte distribuée

Solution ségrégation front-back

- **Inconvénients**
  - Contrat entre *client – fournisseur* à surveiller
    - Nous sommes déjà dans un contexte distribuée!
  - Plus complexe globalement
  - Gestion des défaillances (si *backend* ne répond pas)
  - Complexité accrue du déploiement
  - Gestion de transactions de type BASE
    - Pattern d'architecture particulier (CQRS, Event Sourcing, Saga, etc…)
    - Gestion de la consistance éventuelle (*Eventually consistant*)

Niveau de structure d'une application Java

- Niveau 1: projets indépendants
- Niveau 2: modules maven
- Niveau 3: package Java



Rappel: Maven – Structure de projet

Maven recommande une structure de projet standardisée.
Ce n'est pas obligatoire, mais fortement recommandé!

- **src**
  - **main** → *Ressources principales*
    - java → *Code source java*
    - resources → *Ressources du projet non java*
    - webapp → *Ressources web (optionnel)*
  - **test**
    - java
    - resources

Structure de projet – Packages Java

- Packages par défaut:
  - ***java.lang***
    Importé automatiquement par le compilateur
  - ***Classes sans déclarations de package***
    Bien que autorisé, pratique à bannir!

Structure de projet - Package Java

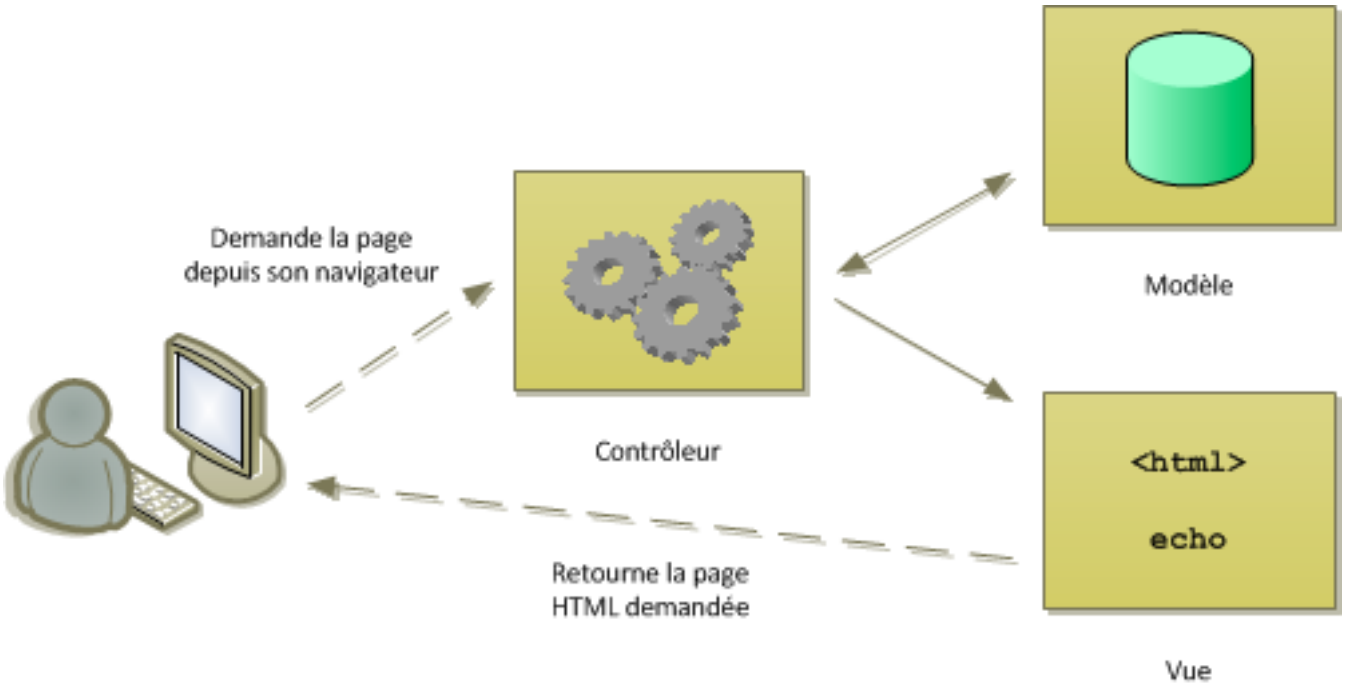
- Sert à **structurer** une application
- Permet de **gérer des droits d'accès** (public, default)
- Niveau de structure **atomique** d'un programme Java (en excluant les classes)

Nommage des package Java

- ceux-ci doivent être écrits entièrement en minuscules
- les caractères autorisés sont alphanumériques (de a à z, de 0 à 9) et peuvent contenir des points (.)
- Idéalement préférez un package root, et une structure hierarchique ensuite
- aucun mot clé Java ne doit être présent dans le nom, sauf si vous le faites suivre d'un underscore (« _ »), comme ceci: ***ch.hearc.package_***

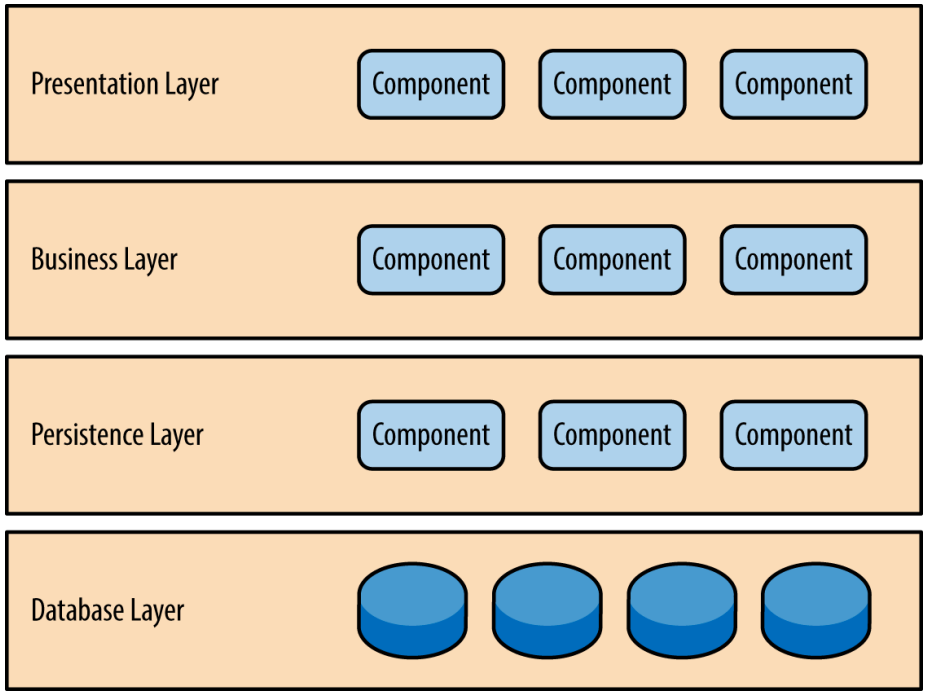
MVC - Concepts

- MVC

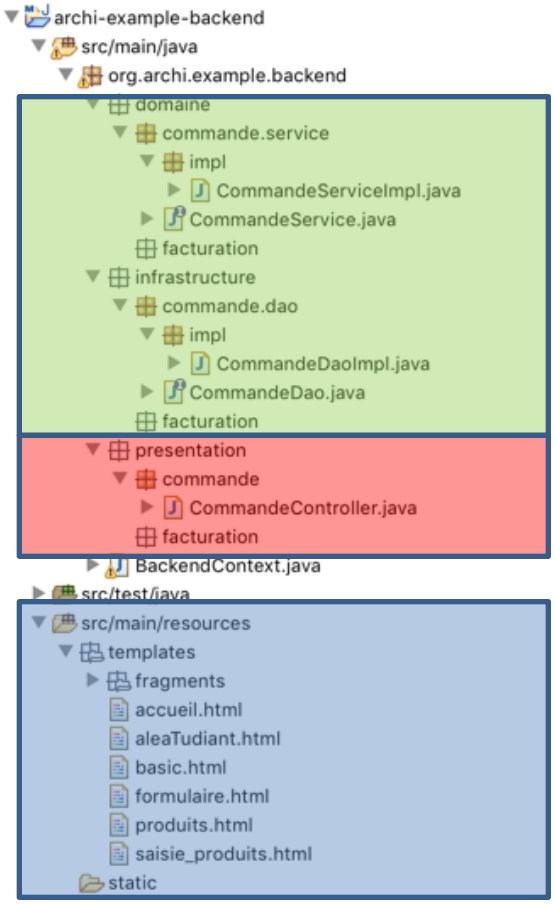


MVC – Eléments d'implémentations

- Eléments implémentation MVC



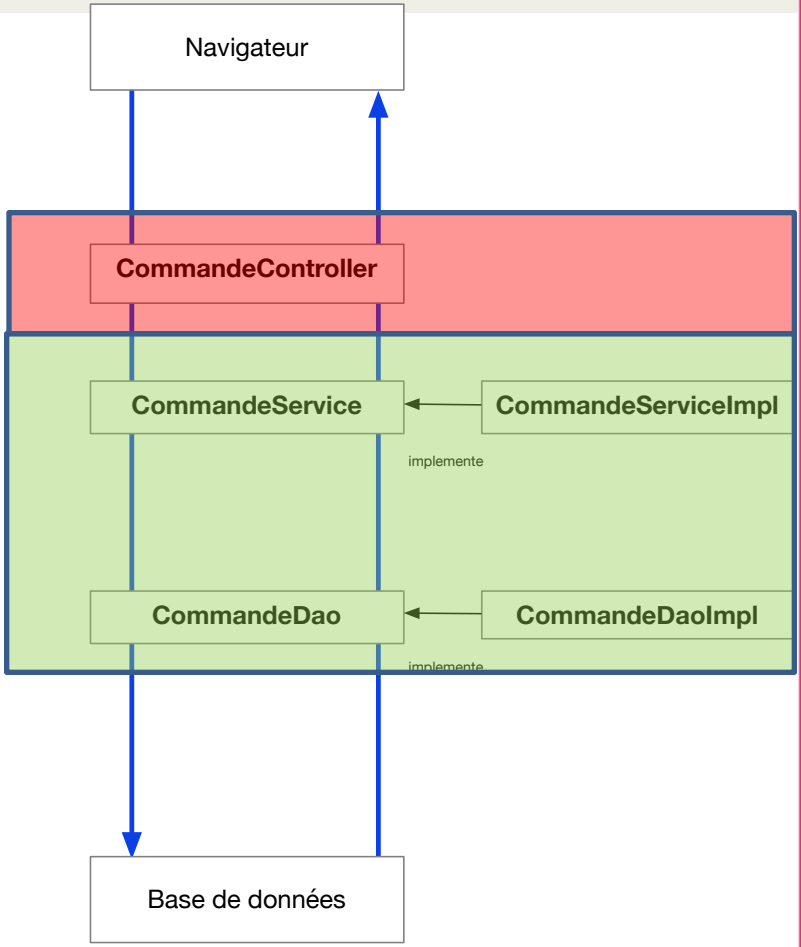
MVC – Exempe traditionel « en couche »



Modèle

Controlleur

Vue



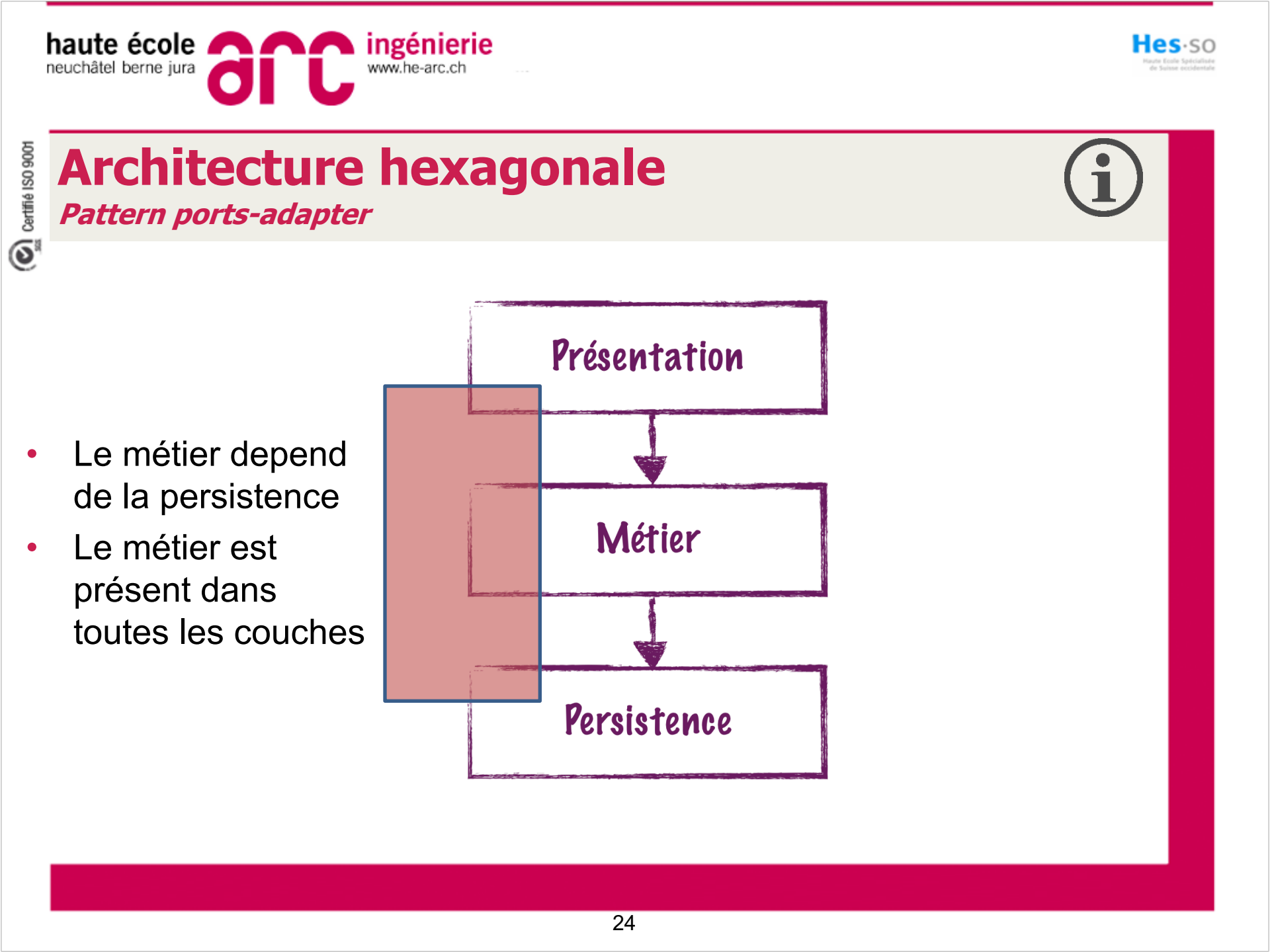
Architecture hexagonale

Pattern ports-adapter

- Variante de l'architecture en couche
- Permet une isolation **stricte** du métier
- Permet une application des principes ***Domain Driven Design***
- Permet une evolution séparée entre aspects techniques et métiers

Architecture hexagonale

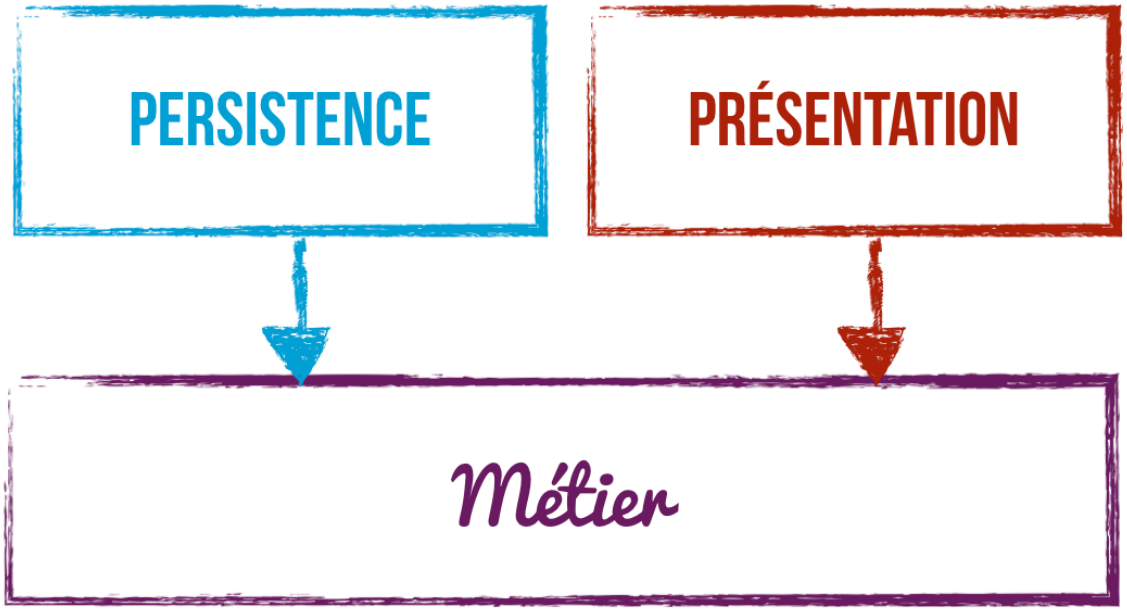
Pattern ports-adapter



- Le métier depend de la persistence
- Le métier est présent dans toutes les couches

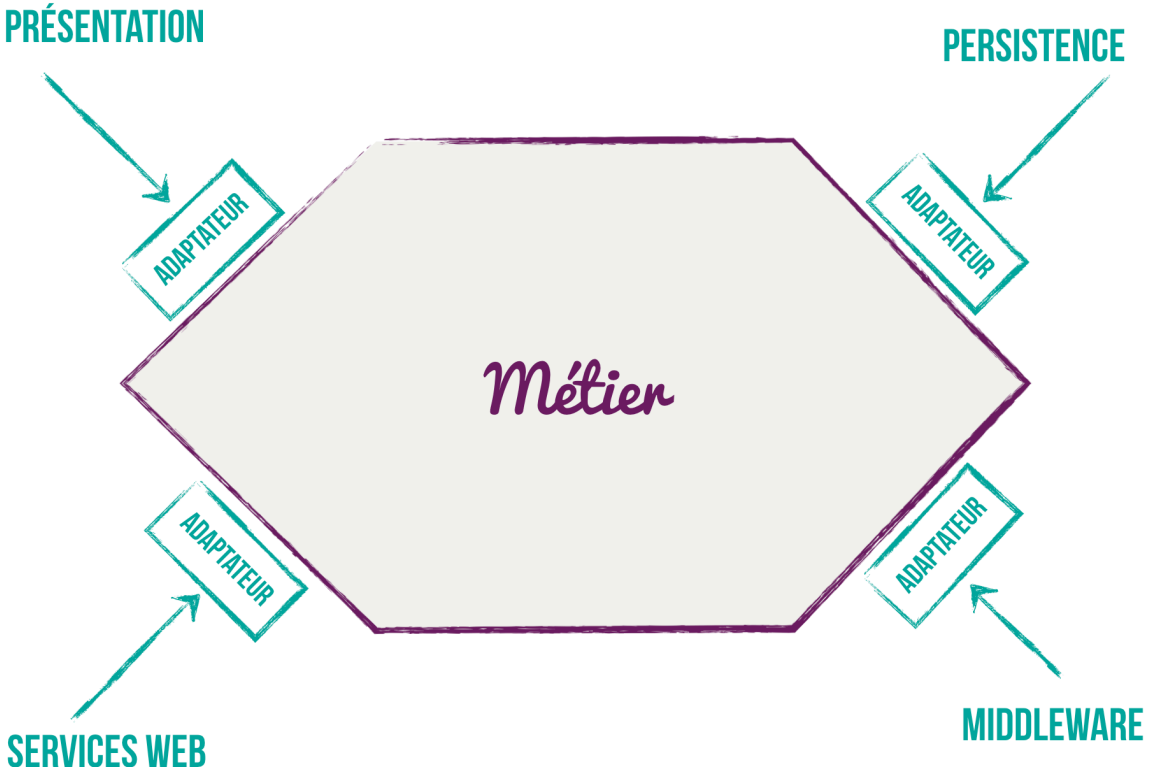
Architecture hexagonale

Dépendances entre couches



Architecture hexagonale

Dépendances entre couches



Architecture hexagonale

API - SPI

Application
Programming
Interface

Service
Programming
Interface

Architecture hexagonale

API - SPI

Application Programming Interface

- Les API sont la description des interfaces/classes/méthodes **appelé et utilisée** pour exécuter une action
- → **Ce que l'application propose**
  - Controler REST
  - Consommateur JMS

Service Programming Interface

- Les SPI sont la description des interfaces/classes/méthodes **implémentée** pour exécuter une action
- → **Ce dont l'application à besoins**
  - Repository DB
  - Accès FS
  - Producteur JMS

Architecture hexagonale

Vue globale