

Génie Logiciel Avancé

Arnaud Labourel (arnaud.labourel@univ-amu.fr)

8 janvier 2025



Section 1

Bienvenue dans le cours de Génie Logiciel Avancé

Volume horaire

- 6 séances de 1h30 de cours (9h)
- 6 séances de 3h de TD/TP en salle machine (18h)

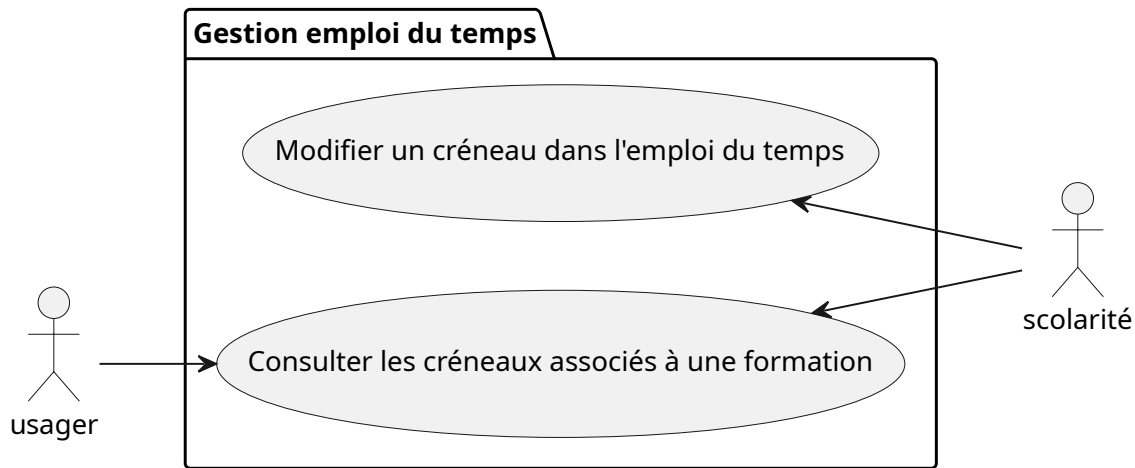
Évaluation

- un examen sur papier (60% de la note)
- un projet (40% de la note)

Max avec la note d'examen terminal

Objectif du projet

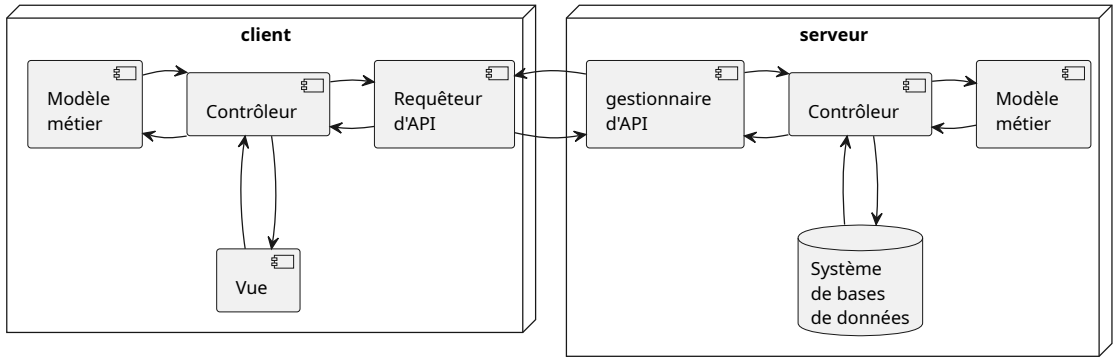
Gestion d'emploi du temps avec ressources (ADE simplifié)



Application multi-couches

- un client :
 - ▶ Un requêteur HTTP pour récupérer et modifier les créneaux du serveur
 - ▶ Un modèle local (version locale de l'emploi du temps avec les créneaux chargés depuis le serveur)
 - ▶ Une interface utilisateur utilisant JavaFX proposant une vue par semaine
 - ▶ Un contrôleur gérant les interactions entre ces trois composants
- un serveur :
 - ▶ Un gestionnaire d'API gérant les requêtes venant des clients
 - ▶ Une logique métier (emploi du temps) qui vérifie en outre les contraintes de disponibilité (pas deux créneaux s'intersectant utilisant la même ressource)
 - ▶ Système de gestion de données (relationnel SQL) pour stocker de manière durable les données des créneaux
 - ▶ Un contrôleur gérant les interactions entre ces trois composants

Diagramme de composants



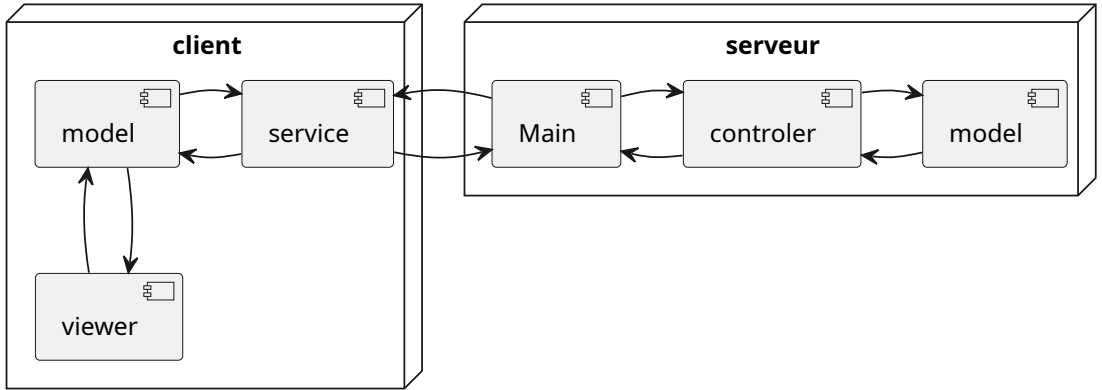
Architecture du code de base du projet

Vous devrez *forker* deux dépôts qui serviront de base au projet

Application multi-couches

- un client :
 - ▶ Un requêteur HTTP (package *service*) pour récupérer les créneaux du serveur
 - ▶ Un modèle local (package *model*) se contentant d'appeler le requêteur HTTP
 - ▶ Une interface utilisateur (package *viewer*) utilisant JavaFX proposant une vue par semaine
- un serveur :
 - ▶ Un gestionnaire d'API (classe `Main`) gérant les requêtes GET venant des clients
 - ▶ Un modèle (package `model`) qui stocke les créneaux
 - ▶ Un contrôleur (package `controller`) gérant les interactions entre ces trois composants

Diagramme de composants du code de base



Travail en équipes de 4 étudiants

Méthodologie de travail

- eXtreme Programming (XP)
- Kanban simple via gitlab (Epics et Issues)
- Intégration continue (gitlab CI/CD) qui lance automatiquement les tests à chaque push
- Développement Dirigé par les tests (TDD)
- Binôme (*pair programming*)

eXtreme Programming (XP)

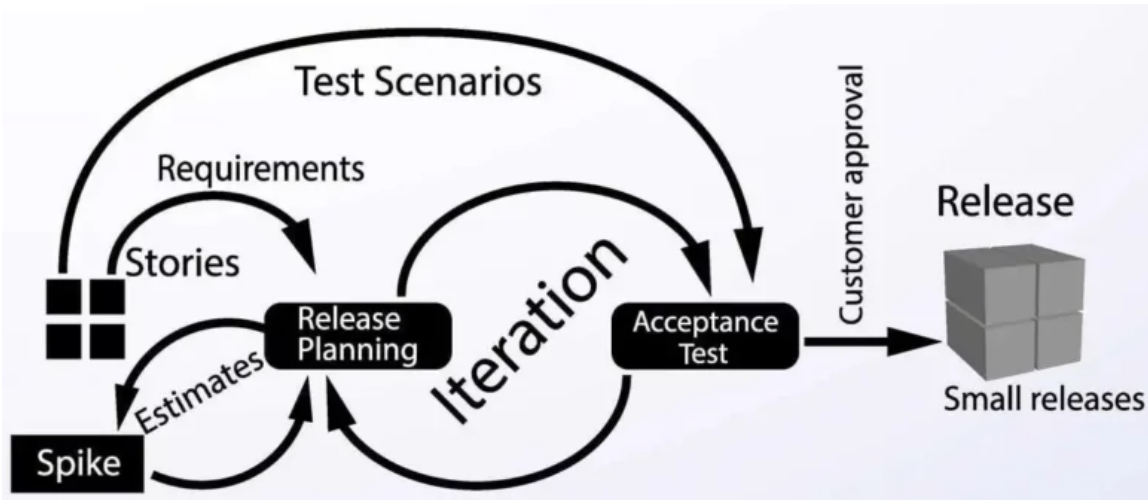
Méthodologie agile reposant sur des cycles rapides de développement (itération d'une semaine dans notre cas) avec comme étapes :

- une phase d'exploration pour déterminer les scénarios « client » ;
- créations des tâches à réaliser (incluant les tests) ;
- attribution des tâches en binôme (pair programming) ;
- livraison (pour notre cas via *gitlab*) lorsque les tests passent.

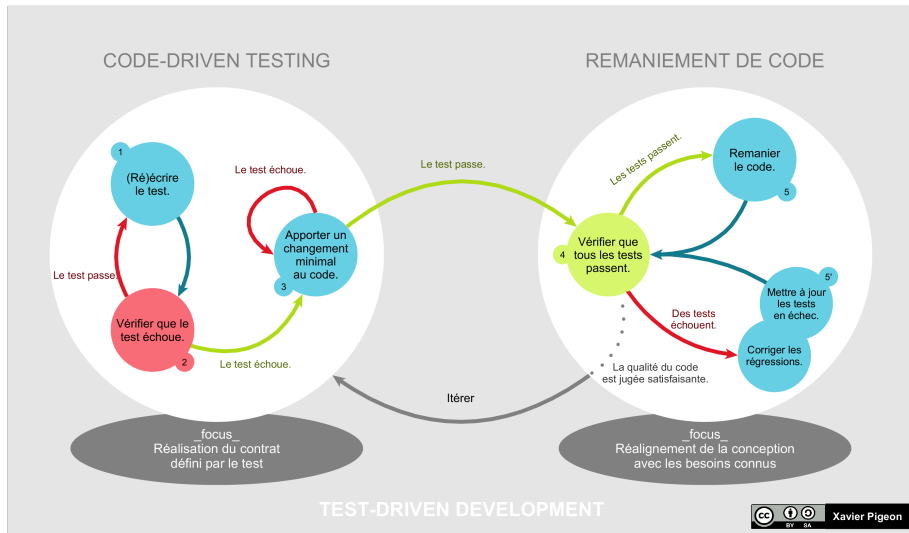
Aspect essentiel

- Importance de la communication dans l'équipe
- Importance des retours (tests et clients)

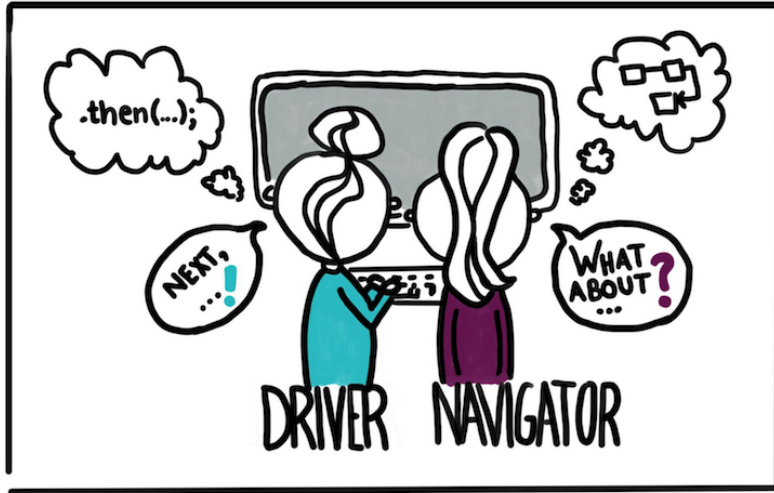
eXtreme Programming (XP) en un schéma



Développer en TDD



Programmation en Binôme (Pair Programming)



Programmation en Binôme (Pair Programming)

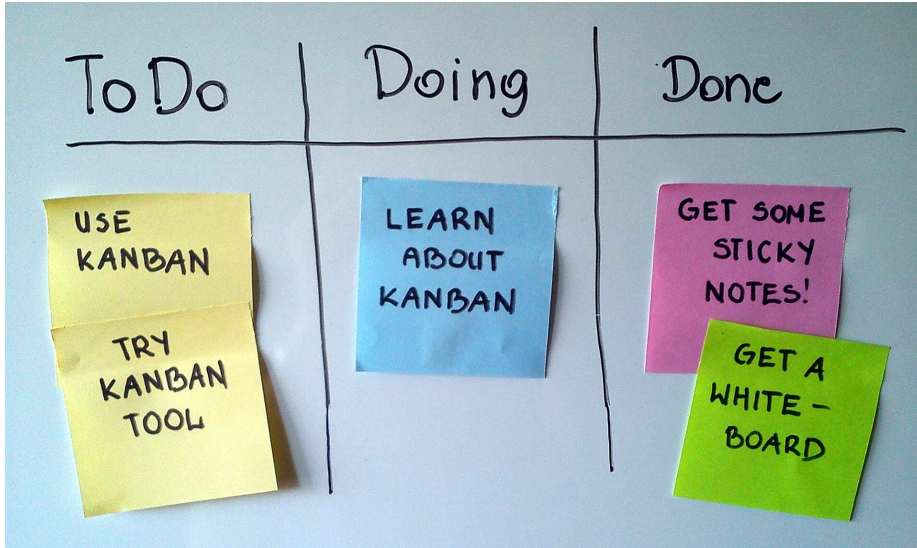
- Concept :
 - ▶ Deux développeurs travaillent sur la même tâche.
 - ▶ Un développeur écrit le code, l'autre le supervise et le vérifie en temps réel.
- Avantages :
 - ▶ Améliore la qualité du code.
 - ▶ Favorise l'échange de compétences et d'idées.
 - ▶ Réduit les erreurs.
- Défis :
 - ▶ Peut être perçu comme une perte de productivité au début.
 - ▶ Nécessite une bonne collaboration et de la communication.

Section 2

Kanban

- Kanban :
 - ▶ Méthode Agile de gestion visuelle du flux de travail.
 - ▶ Origine : Système de production de Toyota (1950).
 - ▶ Objectif : Améliorer l'efficacité en visualisant le processus, limitant le travail en cours (WIP), et optimisant le flux.
 - ▶ Terme *Kanban* ayant le sens de carte de signalisation
- Caractéristiques principales :
 - ▶ Visualisation des tâches via un tableau de tâches.
 - ▶ Limitation du nombre de tâches en cours.
 - ▶ Flux de travail continu.

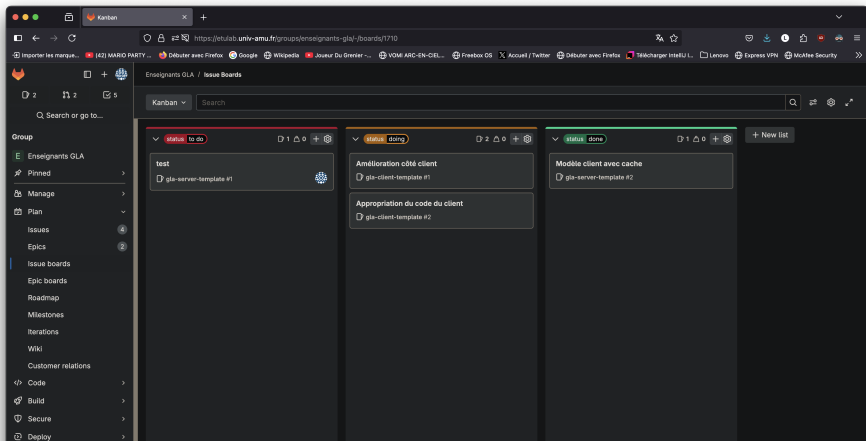
Tableau Kanban simple



- Colonnes (état d'avancement des tâches), généralement il y a au moins trois colonnes :
 - ▶ À faire : Liste des tâches (*user stories* ou *epics*) à prendre en charge.
 - ▶ En cours : Tâches en cours de traitement avec une limite Work In Progress (WIP).
 - ▶ Terminé : Tâches complétées et prêtes à être livrées.
- Cartes :
 - ▶ Chaque tâche est représentée par une carte.
 - ▶ La carte contient des informations essentielles comme la description de la tâche, la priorité, le responsable, ...
- *User Stories* : récits utilisateurs
- *Epics* (Épopées) :
 - ▶ ensemble de tâches cohérentes visant à un même but
 - ▶ peu nombreuses : quelques *epics* par mois vs dizaines d'*user stories* par mois

Kanban logiciel

Il existe des logiciels spécifiques de suivi du travail : Agile SAP, Kanban Tool, Jira Agile, Kanboard, Taiga, gitlab ...



Les Pratiques Clés de Kanban

- Visualiser le flux de travail :
 - ▶ Utilisation d'un tableau Kanban pour suivre les tâches à travers différentes étapes.
 - ▶ Permet à toute l'équipe de comprendre l'état du travail.
- Limiter le travail en cours (Work in Progress WIP) :
 - ▶ Définir des limites pour chaque colonne (par ex., nombre maximum de tâches).
 - ▶ Empêche la surcharge et améliore l'efficacité.
- Gérer le flux :
 - ▶ Surveiller et optimiser le mouvement des tâches dans le système.
 - ▶ Objectif : un flux de travail fluide et sans blocage.
- Rendre les processus explicites :
 - ▶ Clarifier et définir les règles et les processus.
 - ▶ Aider l'équipe à comprendre et suivre des étapes spécifiques pour chaque tâche.
- Boucles de rétroaction :
 - ▶ Organiser des réunions régulières pour améliorer le processus.
 - ▶ Discussions sur les goulots d'étranglement et les améliorations possibles.

Avantages et Défis de Kanban

Avantages

- Flexibilité : ajouts ou retraits des tâches à tout moment.
- Amélioration continue : révision et une optimisation continues du processus.
- Visualisation claire : l'état de chaque tâche est visible, ce qui permet une gestion efficace.
- Réduction des goulots d'étranglement : Le suivi du flux et des WIP limite les blocages.
- Meilleure collaboration : Encourage une communication continue.

Défis

- Risque de surcharge si les WIP ne sont pas respectés.
- Peut manquer de structure formelle pour les équipes moins disciplinées.
- Nécessite une culture d'amélioration continue.

Il est possible de mettre en place un tableau Kanban dans *gitlab* grâce aux notions suivantes (incluse dans le gitlab AMU : etulab) :

- Group : équipe de développement
- Issue : tâches associées à un dépôt
- Label : étiquettes pouvant être attribuées à des issues permettant de créer des colonnes d'un tableau Kanban
- Epic : regroupe des Issues au niveau d'un group (permet par exemple de regrouper des tâches sur plusieurs dépôts)

Tableau Kanban complexe

Pool of Ideas	Feature Preparation		Feature Selected	User Story Identified	User Story Preparation		User Story Development		Feature Acceptance		Deployment	Delivered
Epic 431	3 - 10 In Progress Ready		2 - 5	30	15 In Progress Ready		15 In Progress Ready (Done)		8 In Progress Ready		5	Epic 294
Epic 478	Epic 444	Epic 662	Epic 602			Story 602-03	Story 602-06	Story 602-05	Epic 401	Epic 609	Epic 694	Epic 386
Epic 562	Epic 589		Epic 302	Story 302-03 Story 302-01	Story 302-07	Story 602-03	Story 602-04	Story 602-01	Epic 468	Epic 577	Epic 276	Epic 419
Epic 439	Epic 651			Story 302-05 Story 302-06	Story 302-08				Epic 362		Epic 339	Epic 388
Epic 329			Epic 335	Story 335-08 Story 335-10 Story 335-04	Story 335-09	Story 335-08					Epic 521	Epic 287
Epic 287				Story 335-08 Story 335-01 Story 335-05	Story 335-02	Story 335-07					Epic 582	Epic 274
Epic 606	Discarded		Epic 512	Story 512-04 Story 512-07 Story 512-02	Story 512-01							
	Epic 511	Epic 213		Story 512-05 Story 512-06 Story 512-03								
	Epic 221											

Policy

Business case showing value, cost of delay, size estimate and design outline.

Policy

Selection at Replenishment meeting chaired by Product Director.

Policy

Small, well-understood, testable, agreed with PD & Team

Policy

As per "Definition of Done" (see...)

Policy

Risk assessed per Continuous Deployment policy (see...)

Section 3

API HTTP

Qu'est-ce qu'est un API HTTP

API utilisant le protocole HTTP.

Principaux types de requêtes

- GET : récupérer une donnée sur le serveur (idempotent)
- POST : créer une nouvelle entité sur le serveur (pas idempotent)
- PUT : mettre à jour une entité ou créer une entité en spécifiant son identifiant (idempotent)
- DELETE : supprimer une entité en spécifiant son identifiant (idempotent)

Idempotent : produit le même résultat (sur les données) si répété plusieurs fois.

Javalin côté serveur

- framework logiciel léger pour la création de serveur web en Java ou Kotlin
- utilise *Jetty* qui est une implémentation de serveur HTTP en Java
- permet de définir un comportement en fonction des routes et types de requêtes

Java HttpClient API (depuis Java 11)

- inclus dans Java
- permet de construire facilement

On utilisera des classes/record (DTO pour Data Transfer Object) pour représenter les objets permettant de modéliser les paramètre et réponse des requêtes.

Exemple de requêtes du projet

Requête déjà implémentée

- GET `serveur/timeslots/2025-01-06&2025-01-10` : récupère les créneaux entre le 6 et le 10 janvier 2025 au format JSON.

Requêtes à implémenter

- GET `serveur/timeslots/25` : récupère le créneau d'identifiant 25
- DELETE `serveur/timeslots/25` : supprime le créneau d'identifiant 25
- PUT `serveur/timeslots/25` : met à jour un créneau
- POST `serveur/timeslots/` : crée un créneau
- ...

Toutes les requêtes sauf GET contiennent les données d'un créneau (au format JSON) dans leur corps.

Créer la requête

Requête GET

```
HttpRequest request = HttpRequest.newBuilder()  
    .uri(new URI(route))  
    .GET()  
    .build();
```

Requête POST json

```
HttpRequest request = HttpRequest.newBuilder()  
    .uri(new URI(route))  
    .header("Content-Type", "application/json")  
    .POST(BodyPublishers.ofString(requestBody))  
    .build();
```

Envoyer la requête

```
HttpClient client = HttpClient.newBuilder().build()
```

Bloquant (synchrone)

```
HttpResponse<ClassDTO> response = client  
    .send(request, new JsonBodyHandler<>(ClassDTO.class));
```

Non-bloquant (asynchrone)

```
CompletableFuture<HttpResponse<ClassDTO>> futureResponse =  
    client.sendAsync(request, new JsonBodyHandler<>(ClassDTO.class));  
/* ... */  
HttpResponse<ClassDTO> response = futureResponse.get();
```

Code côté serveur avec javalin

```
Javalin.create(/* config */)
    .get("/calendar/{params}",
        ctx -> {
            String parameters = ctx.pathParam("params");
            DateInterval dateInterval = DateInterval.parse(parameters);
            var response = calendarController.get(dateInterval);
            ctx.json(response);
        })
    .delete("/timeslot/{id}",
        ctx -> {
            String id = ctx.pathParam("id");
            var response = calendarController.delete(Integer.parseInt(id));
            ctx.json(response);
        })
```

Section 4

Séparer l'accès aux données de la partie métier

Quatre types d'opérations pour le stockage (CRUD) :

- Create : création de données
- Read : lecture/récupération de données
- Update : mise à jour de données
- Delete : suppression de données

Dans une application bien conçue, la partie métier doit être le plus indépendante possible de la solution de stockage choisie.

Exemple emploi du temps

- Partie métier : gère les créneaux et vérifie les contraintes (disponibilité des ressources comme enseignants, salles, enseignants)
- Stockage : manière dont sont stockés les créneaux et les ressources (SQL, fichiers XML, stockage en mémoire, ...)

Patron de conception DAO

Définir une interface pour découpler la partie métier de la partie de l'accès aux données (via SQL ou une API HTTP) :

```
public interface DAO<T> {  
    Optional<T> get(long id);  
    List<T> getAll();  
    void create(T t);  
    void update(T t, String[] params);  
    void delete(T t);  
}
```

Pourquoi séparer métier et accès aux données

Pour découpler les deux : la partie métier ne doit pas être impacté par un changement de technologie de stockage.

DTO : Data Tranfert Object

Le type T du DAO est ce qu'on l'on appelle objet de transfert (DTO)

Les DTO sont des objets de transferts qui sont utilisé pour récupérer les données. Dans notre cas, on utilisera des record.

Afin de pouvoir transférer des données (heure et date des créneaux dans notre cas) via HTTP, on a besoin de définir un format de donnée.

⇒ le plus simple est d'utiliser JSON et d'utiliser la bibliothèque *Jackson* pour sérialiser et désérialiser les objets.

Attention

Les DTO ne sont pas les objets à utiliser dans le modèle, mais de simples objets de transfert de données.