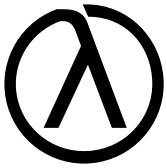


L’architecture contrôleurs / services / dépôts

L’architecture contrôleurs / services / dépôts est une architecture utilisée pour la conception de services web RESTful.

Elle présente de fortes similarités avec l’architecture 3 tiers, du fait qu’elle appelle elle aussi à une décomposition du logiciel en 3 couches avec des domaines de responsabilités similaires. De ce fait on peut également la classer parmi les **architectures impures** pour ce qui est de la gestion des dépendances.

1. Objectif

L’objectif de l’architecture contrôleurs / services / dépôts est de proposer une organisation du code pour la création de **services web RESTful**.

Tout comme l’architecture 3 tiers, dont elle s’inspire fortement, elle demeure **simple** à comprendre et à mettre en œuvre.

Mais contrairement à l’architecture 3 tiers, ce n’est pas la totalité de l’ancienne architecture client / serveur à deux couches qu’elle se propose de redécouper en trois couches, mais seulement sa couche serveur, en lui faisant prendre la forme de services web. Si on devait lui ajouter la notion de client, représentée par la couche présentation dans l’architecture 3 tiers, alors on se retrouverait en fait avec une architecture en non pas trois, mais quatre couches.

Mais, de fait, l’architecture contrôleurs / services / dépôts laisse de côté la question du logiciel client pour ne s’intéresser qu’à celle du côté serveur. L’une des raisons à cela est que, ce côté serveur prenant la forme de services web, on envisage qu’il peut être utilisé par différents clients qui seront éventuellement développés par d’autres organisations que celle qui aura à développer ces services web.

Soulignons en outre que cette architecture est une option proposée par des générateurs de code comme OpenApi Generator. Ce générateur de code permet de générer (avec plus ou moins de réussite) les classes de base et la documentation Swagger pour des services web RESTful, dans différents langages (au premier rang desquels le langage Java, qui est le langage dans lequel est lui-même écrit ce générateur).

Sans en augmenter (ni en diminuer) les mérites, cela tend à installer l’architecture contrôleurs / services / dépôts comme un **classique** de la discipline.

2. Principe

L’architecture contrôleurs / services / dépôts, comme son nom et sa parenté avec l’architecture 3 tiers le laissent supposer, propose d’organiser le code d’un ensemble de services web autour de trois couches :

La couche **contrôleurs**

Elle correspond à la couche présentation de l'architecture 3 tiers.

Mais contrairement à cette dernière, elle ne consiste pas en un client devant présenter les données directement à un·e utilisateur·ice humain·e, mais en l'interface frontale des services web, avec laquelle interagiront des logiciels clients.

Cette interface va typiquement proposer des **routes** pouvant être appelées en **HTTP(S)** avec divers verbes (GET, P0ST, PUT, DELETE…) proposés par ce protocole pour réaliser des opérations de collecte, de création, de mise à jour ou de suppression de données. Ces données seront généralement représentées en JSON, format qui a l'avantage d'être directement exploitable par le code Javascript des clients qui prendraient la forme d'applications web.

On appelle généralement **DTO** (pour *Data Transfer Objects*) les structures de données qui sont utilisées pour cette communication avec les logiciels clients. Ils sont distincts des objets entités utilisés par la couche dépôt.

La couche contrôleur a notamment la charge de vérifier que les pramètres ou les DTO montants qu'elle aura éventuellement reçus avec la requête HTTP sont au bon format et font sens par rapport à la demande représentée par la route appelée.

Elle a également la charge de vérifier si la requête est autorisée. Toutes les API web ne sont en effet pas publiques et leur emploi peut nécessiter autorisation et authentification.

La couche **services**

Elle correspond à la couche logique métier de l'architecture 3 tiers.

Comme son analogue, cette couche a pour responsabilité d'appliquer les règles de gestion métier correspondant au demandes que lui transmet la couche contrôleurs.

Pour cela, elle aura généralement à collecter des données auprès de la couche dépôts. Ces données lui seront retournées sous la forme d'objets entités, auxquels elle devra appliquer divers traitements correspondant aux règles de gestion, avant de les convertir en DTO qu'elle retournera à la couche contrôleurs.

La couche **dépôts**

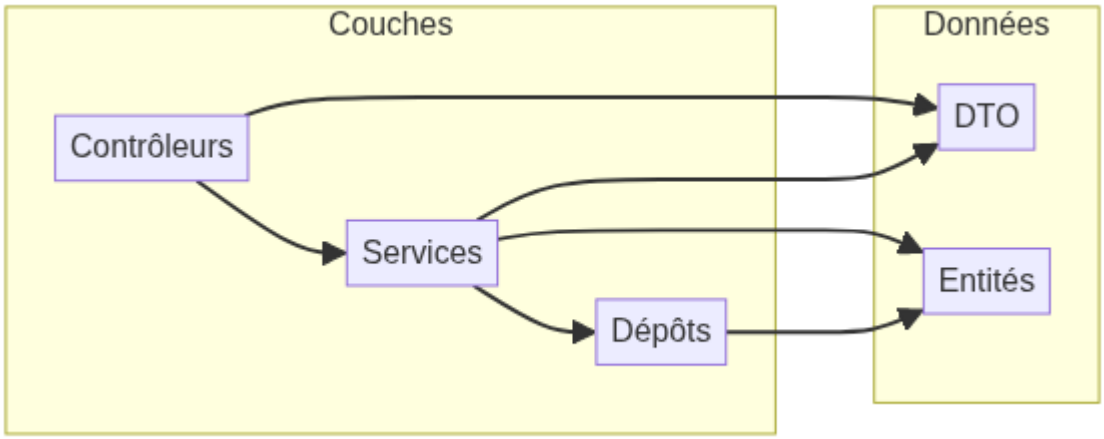
Elle correspond à la couche persistance de l'architecture 3 tiers.

Elle se charge donc de collecter des données depuis des supports de stockage persistants, ou d'y inscrire des données.

Le plus souvent, dans les applications de gestion, ces supports de stockage persistants seront des base de données exploitées au travers d'ORM (*Object-Relational Mapping*) effectuant la correspondance entre des objets du langage d'implémentation de la couche dépôts et les structures attendues par les systèmes de gestion de bases de données. Ces objets, dans le langage d'implémetation de la couche dépots sont des **objets entités**.

Les dépendances entre ces couches sont très exactement calquées sur celles entre les couches correspondantes dans l'architecture 3 tiers. On peut toutefois y ajouter les

dépendances vers les DTO d'une part et les objets entités d'autre par, pour voir quelle couche utilise quoi.



Le flux typique du traitement d'une requête est donc le suivant :

1. La couche contrôleurs accepte des requêtes HTTP, lui transmettant éventuellement des DTO correspondant à des données montantes. Ces DTO sont alors désérialisés du JSON vers le langage d'implémentation du serveur.

   Si quelque chose se passe mal à ce stade, cela peut donner lieu à une réponse HTTP précoce qui coupe court à la suite du traitement. Il peut, par exemple, s'agir d'une réponse de code HTTP 400 (*Bad Request*), si les données accompagnant la requête HTTP ne correspondent pas à ce qui est attendu.

2. Si toutefois la requête est valide et autorisée, la couche contrôleurs fait alors appelle à la couche services en lui transmettant les éventuelles données montantes sous forme de DTO désérialisés.

3. Si elle a besoin de collecter des données pour élaborer sa réponse (ce qui sera généralement le cas), la couche services va s'adresser à la couche dépôts. Ce sera également le cas si l'opération demandée consiste en la création ou la modification de données dans un support de stockage persistant.

   Comme la couche dépôts ne parle pas le langage des DTO mais celui des objets entités, la couche services devra effectuer des conversions des DTO qu'elle aura reçus de la couche contrôleurs en objets entités qu'elle transmettra à la couche dépôts.

4. La couche dépôts s'occupe alors d'effectuer les opérations de persistance correspondant à la demande, qu'il s'agisse d'opérations de lecture ou d'écriture vers des supports persistants.

   Il s'agit souvent de bases de données qui sont utilisées avec des ORM s'appuyant sur les objets entités.

   S'il y a des données descendantes à transmettre en réponse à la couche services, typiquement dans le cas d'une demande de collecte de données, alors celles-ci lui seront transmises sous forme d'objets entités.

5. La couche services réceptionne donc ces éventuelles entités.

Elle peut alors leur appliquer divers traitements pouvant correspondre à l'application de règles de gestion.

Puis elle devra les convertir en DTO pour les retourner à la couche contrôleurs.

6. Enfin, la couche dépôts devra sérialiser ces DTO en JSON pour les faire figurer dans sa réponse HTTP(S) à la requête qu'elle a initialement reçue.

Notons que la séparation entre les trois couches a plus vocation à être logique que physique. C'est à dire que ces couches prendront généralement la forme d'une organisation du code (en paquets ou modules séparés) au sein d'un même projet logiciel plutôt que la forme de composants logiciels distincts.

Il existe cependant des logiciels qui font le choix de séparer ces couches physiquement, mais cette approche est marginale car elle complexifie considérablement la mise en œuvre de l'architecture (mais cela permettrait, si besoin était, d'écrire ces couches avec des langages de programmation différents). Chaque couche prend alors la forme d'un ensemble de services web communiquant avec les autres en HTTP(S). Dans ce cas, seule la couche contrôleurs sera constitutive de services web publiés vers l'extérieur de l'organisation.

3. Critiques

Des conversions parfois fastidieuses

À l'architecture 3 tiers, on adressait une critique concernant des ambigüités sur les formats de communication entre couches.

Avec l'architecture contrôleurs / services / dépôts, ces ambigüités sont levées : la couche contrôleurs parle le langage des DTO, la couche dépôts celui des objets entités et la couche services assure la traduction entre les deux.

Il peut arriver qu'il y ait de petites différences marginales entre les structures des DTO et celles des objets entités. Mais dans l'ensemble elles sont généralement tout de même extrêmement similaires.

Aussi, lorsqu'on écrit du code dans le cadre d'un projet ayant adopté l'architecture contrôleurs / services / dépôts, on a souvent l'impression de faire de la quasi duplication de code et de passer beaucoup de temps à transvaser des données des DTO vers des entités, puis des entités vers des DTO à nouveau ; même si certains langages de programmation proposent des choses pour automatiser cela au maximum.

L'impureté des dépendances services → dépôts

Les dépendances entre la couche services et la couche dépôts sont impures exactement comme le sont, concernant l'architecture 3 tiers, les dépendances entre la couche logique métier et la couche persistance.

La couche dépôts est intrinsèquement impure, du fait qu'elle va devoir gérer des opérations d'entrées et de sorties vers des supports de stockage persistants. De ce fait, toute dépendance vers cette couche va automatiquement contaminer les composants qui, sans cela, aurait pu avoir l'opportunité de rester purs.