

27 FÉVRIER 2024

# RÉALISATION DE PROGRAMME

ÉTUDE ET  
EXPÉRIMENTATION  
D'UN FRAMEWORK  
POUR LA  
RÉALISATION  
COLLABORATIVE  
D'UNE APPLICATION  
WEB

# Table des matières

<b>Présentation générale du projet</b>	<b>4</b>
Étude et expérimentation d'un framework pour la réalisation collaborative d'une application web . . . . .	4
Spécification Jakarta EE . . . . .	4
Utilisation du logiciel de compréhension et de gestion de développement de projets logiciels Apache Maven . .	4
Conception . . . . .	5
Architecture multi-tiers . . . . .	5
Les patrons de conception DAO et Singleton . . . . .	5
Diagramme de cas d'utilisations et diagrammes UML . . . . .	5
Diagramme de cas d'utilisation . . . . .	6
Diagramme entités associations . . . . .	6
Diagramme de classes . . . . .	6
Application du framework à la collaboration . . . . .	6
Collaboration avec les créateurs du framework . . . . .	6
Rôle de la conception dans la collaboration . . . . .	6
Rôle de la spécification dans la collaboration . . . . .	7
. . . . .	7
Application de la gestion des versions à la collaboration . . . . .	7
Utilisation du logiciel de gestion des versions Git . . . . .	7
Utilisation d'un service de gestion de version et d'hébergement de code source GitHub . . . . .	7
Rôle des bonnes pratiques dans la collaboration . . . . .	7
Séparation des préoccupations . . . . .	8
Couplage faible . . . . .	8
Gestion efficace des versions . . . . .	8
Autres Pratiques Essentielles . . . . .	8
Utilisation du système de gestion de bases de données relationnelles MySQL . . . . .	8
Utilisation des environnements de développement Eclipse et IntelliJ . . . . .	8
Déploiement pour le travail en développement . . . . .	8
Utilisation d'une plateforme de déploiement fournie par le serveur d'application Tomcat . . . . .	8
Déploiement pour le travail en production . . . . .	8
Utilisation du service d'hébergement de machine virtuelle DigitalOcean . . . . .	8
Utilisation du registraire de domaine Namecheap . . . . .	8
Utilisation du service DNS [TO DO] . . . . .	8
Utilisation d [TO DO] . . . . .	8
[TO DO] . . . . .	8
<b>Outils utilisés</b>	<b>9</b>
<b>Construction de l'environnement de développement</b>	<b>12</b>
<b>Conception</b>	<b>18</b>
Diagramme de cas d'utilisation (Use Case Diagram) . . . . .	18
Diagramme entités associations . . . . .	19
Diagramme de classes . . . . .	19
Base de données . . . . .	20
<b>Création de la base de donnée</b>	<b>21</b>
<b>Initialisation du projet Maven et référencement du premier commit Git</b>	<b>22</b>
<b>Téléversement du projet Git dans un dépôt de GitHub</b>	<b>27</b>
<b>Développement de la version minimale 1.0-SNAPSHOT</b>	<b>28</b>
<b>VRAC</b>	<b>30</b>
	<b>31</b>
<b>ANNEXES</b>	<b>1</b>
<b>A Liens étudiés</b>	<b>1</b>

<b>B Étude du cours « Développez des sites web avec Java EE » de Mathieu Nebra sur openclassrooms.com pour développer une application web proposant les services de lecture et d'écriture dans une table d'une base de données MySQL tout en respectant MVC et DAO</b>	<b>2</b>
B.0.1 Création d'une application web	2
B.0.2 Gestion des dépendances tierces	2
B.0.3 Ajout d'un servlet	2
B.0.4 Création d'une vue	2
B.0.5 Implémentation de la méthode doGet du servlet pour associer la vue à au servlet pour la méthode GET	3
B.0.6 Installation de la JSTL (Java server page Standard Tag Library)	3
B.0.7 Création d'une base de données "jakartaee" et une table "noms" dans cette base via le serveur MySQL	4
B.0.8 Installation de JDBC (Java Data Base Connectivity)	4
B.0.9 Implémentation respectant le patron de conception DAO (Data Access Objet)	4
B.0.10 Implémentation de la vue guidb.jsp	7
B.0.11 Gestion des erreurs	7
B.0.12 Gestion des transactions	8
<b>C Étude du cours « Organiser et packager une application Java avec Apache Maven » de Loïc Guibert sur openclassrooms.com</b>	<b>9</b>
C.1 Étapes	9
C.1.1 Création d'un répertoire env/ pour l'environnement de développement	9
C.1.2 Création d'un projet maven respectant la version 1.1 de l'archétype maven-archetype-quickstart	9
C.1.3 Compilation et packaging de l'application	10
C.1.4 Exécution de l'application	11
C.1.5 Importation du projet dans Eclipse	11
C.1.6 Modification de env/maven/maven-workspace/mon-appli/pom.xml pour définir les informations générales et les propriétés du projet	11
C.1.7 Modification de env/maven/maven-workspace/mon-appli/pom.xml relatives au processus du construction du projet par Maven, le build	12
C.1.7.1 Premier exemple : Personnaliser le répertoire de sortie	12
Deuxième exemple : Définir la classe Main dans le JAR	12
C.1.8 Lancement de la construction Maven	13
C.1.9 Exécution du JAR mon-appli/target/mon-appli-1.0-SNAPSHOT.jar	13
C.1.10 Autre exemple d'utilisation des propriétés avec Maven : Affichage de la version du projet, en récupérant la valeur d'une propriété Maven définie dans un fichier de ressources	13
C.1.11 Utilisation de profils pour créer des options dans le build Maven	14
C.1.12 Respecter une architecture multi-tiers; exemple avec un projet de gestion de tickets d'incident	15
C.1.12.1 Description	15
C.1.12.2 Création d'un projet multi-modules respectant l'architecture multi-tiers	16
C.1.12.3 Gestion des dépendances tierces	20
C.1.12.4 Gestion des scopes des dépendances	24
C.1.12.5 Gestion des dépendances d'un projet multi-module de manière globale	26
C.1.13 Description générale des cycle de vie d'une construction Maven (build lifecycle)	27
C.1.14 Utilisation des plugins pour personnaliser la construction Maven	27
C.1.15 Packager les livrables; exemple avec un projet de gestion de tickets d'incident	31
C.1.16 Générer un site pour un projet	36
Généralités	36
Ajout d'une page de documentation à partir du format markdown	39
Ajout d'une FAQ à partir du format fml	40
Générer des rapports	41
Générer la documentation de Java	44
<b>D Arborescence d'un projet Maven</b>	<b>46</b>
<b>E Un exemple minimal de serveur Back-end derrière Apache 2 avec la bibliothèque Python websocket-server</b>	<b>47</b>
<b>F Exemple de conversion manuelle d'un projet Jakarta EE existant en un projet Maven</b>	<b>48</b>
<b>G Exemple de déploiement manuel d'une l'application web sur un serveur Tomcat</b>	<b>49</b>
<b>H Sources</b>	<b>50</b>
.	50

I	Fichiers de sortie	51
	output/.out . . . . .	51

## Présentation générale du projet

### Étude et expérimentation d'un framework pour la réalisation collaborative d'une application web

L'objectif central de ce projet est de fournir une expérience d'apprentissage pratique et complète sur les outils et méthodes essentiels dans le développement collaboratif d'applications web. Il couvre l'intégralité du cycle de développement, depuis la conception initiale et la structuration d'un environnement de développement, jusqu'à la maintenance en production, en soulignant l'importance d'une mise en œuvre collaborative.

En développant une application web inspirée de plateformes telles que chess.com, mais dédiée au jeu de Puissance 4, ce projet met en pratique un ensemble d'outils et de méthodologies éprouvés dans le secteur professionnel. Ces outils et méthodes, ainsi que leur application concrète à chaque étape du développement, sont présentés dans les sous-sections suivantes.

Cette démarche vise non seulement à fournir aux participants des connaissances théoriques, mais aussi à les engager dans une application pratique de ces concepts, les dotant ainsi de compétences directement transférables et valorisées dans le milieu professionnel, et renforçant par conséquent leur profil auprès des recruteurs.

### Spécification Jakarta EE

Pour ce projet, nous avons choisi de nous appuyer sur la spécification Jakarta EE, l'évolution de Java EE, qui représente le pilier des applications d'entreprise modernes en Java. Une spécification, dans le contexte du développement logiciel, est un ensemble de directives et de normes qui définissent comment une technologie doit être utilisée et mise en œuvre. Elle garantit la standardisation et la compatibilité entre différents développements et plateformes.

Jakarta EE est une spécification riche et étendue, conçue pour fournir un cadre robuste pour le développement d'applications d'entreprise. Elle englobe une variété de technologies et de composants qui sont essentiels pour construire des applications web et d'entreprise évolutives et performantes. Parmi ces composants, les servlets et les JavaServer Pages (JSP) sont des éléments centraux. Les servlets permettent de créer des pages web dynamiques en répondant aux requêtes des utilisateurs, tandis que les JSP facilitent l'écriture de pages web en intégrant du code Java directement dans le HTML.

Dans notre projet, l'adoption de Jakarta EE nous permettra de tirer parti de ses nombreuses fonctionnalités pour développer une application web Puissance 4. Nous mettrons un accent particulier sur la modularité, la sécurité et la performance, qui sont des aspects fondamentaux de cette spécification. L'utilisation de Jakarta EE assurera que notre application soit non seulement conforme aux standards de l'industrie, mais aussi qu'elle bénéficie de la robustesse, de la scalabilité et de la flexibilité nécessaires pour répondre aux exigences d'une application d'entreprise moderne.

Dans l'annexe B : « Étude du cours « Développez des sites web avec Java EE » de Mathieu Nebra sur [openclassrooms.com](https://openclassrooms.com) pour développer une application web proposant les services de lecture et d'écriture dans une table d'une base de données MySQL tout en respectant MVC et DAO », nous apprendrons à utiliser la spécification Jakarta EE tout en étudiant le cours « Développez des sites web avec Java EE » sur [openclassrooms.com](https://openclassrooms.com).

### Utilisation du logiciel de compréhension et de gestion de développement de projets logiciels Apache Maven

Apache Maven, un outil essentiel en gestion de projets logiciels, sera au cœur de notre projet. Sa portée va bien au-delà de la simple automatisation de la construction de logiciels. En effet, Maven est réputé dans de nombreux milieux professionnels pour sa capacité à standardiser et gérer des structures de projets complexes. Sa compétence dans la gestion des dépendances et la standardisation des processus de construction est hautement valorisée par les recruteurs, fournissant ainsi aux participants une compétence directement transférable dans le monde professionnel.

Maven se distingue par sa capacité à structurer un projet de manière claire et ordonnée, inculquant les bonnes pratiques de développement. Cette structuration naturelle selon des conventions éprouvées est cruciale pour assurer la cohérence et la maintenabilité du code, particulièrement dans les grands projets ou les environnements collaboratifs.

En outre, Maven excelle dans la facilitation de l'intégration continue et de la livraison continue, essentielles à un développement logiciel moderne et efficace. Ces processus automatisent les tests et le déploiement, garantissant une qualité constante et minimisant les erreurs humaines lors des cycles de release.

Une fonctionnalité remarquable de Maven est sa capacité à générer un site de documentation automatique pour le projet, ainsi que des rapports détaillés sur la qualité du code. Ces rapports peuvent inclure des analyses de couverture de tests, des contrôles de style de codage et d'autres indicateurs de qualité, améliorant la compréhension du projet et encourageant le maintien d'un code de haute qualité.

L'adoption d'Apache Maven dans ce projet équipe les participants d'une compétence prisée, augmentant leur attractivité auprès des employeurs potentiels et leur fournissant les outils nécessaires pour gérer efficacement des projets logiciels.

Dans l'annexe C : « Étude du cours « Organiser et packager une application Java avec Apache Maven » de Loïc Guibert sur [openclassrooms.com](https://openclassrooms.com) », nous apprendrons à utiliser Apache Maven tout en étudiant le cours « Organisez et packagez une application Java avec Apache Maven » sur [openclassrooms.com](https://openclassrooms.com).

## Conception

### Architecture multi-tiers

L'architecture de notre application sera structurée selon un modèle multi-tiers, divisant l'application en trois niveaux d'abstraction distincts : la présentation, la logique métier, et la persistance des données. Chacune de ces couches est conçue pour opérer de manière autonome tout en interagissant efficacement avec les autres, ce qui garantit une séparation nette des responsabilités et facilite la maintenance du code.

Pour concrétiser cette architecture, nous utiliserons Maven pour créer cinq modules distincts : batch, webapp, business, consumer, et model. Ces modules représentent trois niveaux d'abstraction, partageant tous le module model, ce dernier module gérant les objets les plus concrets de l'application, dénués d'analyse et de logique métier. Ces objets, basiques et essentiels, peuvent être manipulés à travers les trois niveaux d'abstraction suivant :

- Couche de persistance (consumer) :

Contenant le module consumer, cette couche est responsable de l'interaction avec les bases de données, gérant le stockage et la récupération des données essentielles à l'application. Cette gestion de données permet d'économiser la mémoire et de sauvegarder les informations importantes en cas d'arrêt inattendu du serveur.

- Couche métier (business) :

Utilisant les services de la couche de persistance, le module business au sein de cette couche est chargé de toute la logique métier de l'application. Par exemple, il détermine la légalité d'un coup dans une partie de Puissance 4.

- Couche de présentation (batch et webapp) :

Cette couche contient les modules batch et webapp, qui exploitent les services de la couche métier. Le module webapp est conçu pour offrir une interface utilisateur interactive et attrayante, facilitant l'engagement des utilisateurs humains avec l'application. En parallèle, le module batch joue un rôle différent : il exécute un programme automatisé qui simule un utilisateur virtuel de l'application web. Ce programme, une fois qu'il termine une partie dans l'arène, en lance immédiatement une nouvelle, restant ainsi constamment actif pour proposer des parties aux utilisateurs réels. Pour prendre ses décisions de jeu, ce simulateur d'utilisateur fait appel au sous-package business.ai, exploitant ainsi l'algorithme d'intelligence artificielle min-max développé dans le cadre d'un autre projet stocké dans le répertoire **LegacyMVC\_Codebase** dont on va ré-utiliser le code source comme expliqué dans la [section « Réutilisation de code source »](#). Cette approche offre une expérience interactive continue aux utilisateurs, leur permettant d'affronter une IA sophistiquée à tout moment.

En adoptant cette structure modulaire avec Maven, nous assurons une organisation claire du projet, facilitant à la fois le développement et la maintenance. Chaque module a un rôle défini et interagit avec les autres de manière cohérente, reflétant la puissance et la flexibilité de l'architecture multi-tiers.

Dans l'annexe C : « Étude du cours « Organiser et packager une application Java avec Apache Maven » de Loïc Guibert sur [openclassrooms.com](#) », nous apprendrons à utiliser Apache Maven pour construire un projet multi-module respectant l'architecture multi-tiers tout en étudiant le cours « Organisez et packagez une application Java avec Apache Maven » sur [openclassrooms.com](#).

### Les patrons de conception DAO et Singleton

Les patrons de conception DAO (Data Access Object) seront mis en œuvre pour encapsuler la logique d'accès aux données, offrant une abstraction et une isolation par rapport à la source de données. Cela permet une meilleure organisation du code et facilite les modifications et la maintenance des opérations de base de données. En ce qui concerne le patron de conception Singleton, son utilisation sera spécifiquement adaptée à deux cas principaux dans notre application :

- Dans le package consumer, nous créerons une classe DaoFactory qui suivra le pattern Singleton. Cette approche garantira qu'une seule instance de DaoFactory sera créée et partagée dans toute l'application. L'objectif est de centraliser la création des objets DAO, permettant une gestion cohérente et efficace des accès aux données, ainsi qu'une réduction de l'overhead lié à la création répétée d'instances DAO.
- Dans le package model, une classe UserActivities suivra également le pattern Singleton. Cette classe unique servira de référentiel central pour toutes les informations relatives aux utilisateurs connectés, y compris l'utilisateur virtuel IA géré dans le module batch. Elle conservera également un suivi des parties proposées et en cours. L'adoption de Singleton pour UserActivities assure que chaque instance de servlet ait accès aux informations nécessaires concernant l'état global de l'application, favorisant ainsi une synchronisation efficace et une vue cohérente des activités des utilisateurs.

Dans l'annexe B : « Étude du cours « Développez des sites web avec Java EE » de Mathieu Nebra sur [openclassrooms.com](#) pour développer une application web proposant les services de lecture et d'écriture dans une table d'une base de données MySQL tout en respectant MVC et DAO », nous apprendrons en particulier à utiliser ces deux patrons de conception tout en étudiant le cours « Développez des sites web avec Java EE » sur [openclassrooms.com](#).

### Diagramme de cas d'utilisations et diagrammes UML

Le diagramme de cas d'utilisation, le diagramme entités-associations et le diagramme de classe joueront un rôle essentiel dans la conception et la documentation de notre projet.

D'une part, nous apprendrons à utiliser le diagramme de cas d'utilisation en étudiant le livre « [Use Case Modeling](#) » de Kurt Bittner et Ian Spence téléchargeable depuis [dokumen.pub/qdownload](#) et, d'autre part, nous apprendrons à utiliser les deux diagrammes susmentionnés en étudiant le cours « [UML 2 - De l'apprentissage à la pratique](#) » sur [laurent-audibert.developpez.com](#).

**Diagramme de cas d'utilisation** Ce diagramme est essentiel pour comprendre comment les utilisateurs interagiront avec notre système. Il joue un rôle crucial dans la conception de la couche de présentation, illustrant les différentes interactions que les utilisateurs peuvent avoir avec l'application. Cela inclut la visualisation des fonctionnalités requises et des flux de travail, offrant ainsi une vue d'ensemble claire des opérations utilisateur et de l'interface utilisateur. Bien que principalement axé sur la couche de présentation, ce diagramme aide également à informer les besoins et les exigences de la couche métier, car il met en lumière les opérations que cette dernière doit soutenir.

**Diagramme entités associations** Ce diagramme est crucial pour la conception du module model et la structure de la base de données de l'application. Il nous permet de structurer la base de données en respectant la théorie de la normalisation, en évitant les redondances et en assurant une maintenance aisée. Ce diagramme facilite une compréhension claire de la manière dont les données sont organisées et interconnectées.

**Diagramme de classes** Ce diagramme est utilisé pour détailler la structure des classes dans le système, montrant les relations entre elles, leurs attributs et méthodes. Il joue un rôle clé pour assurer une conception orientée objet cohérente et pour faciliter l'implémentation du code.

## Application du framework à la collaboration

### Collaboration avec les créateurs du framework

Dans le monde complexe et interconnecté du développement moderne, la création en solo est devenue une notion presque obsolète. Même lorsqu'un projet semble être l'œuvre d'un individu, il s'inscrit inévitablement dans un réseau de collaborations implicites et explicites. Prenez l'exemple simple de l'utilisation d'un outil de développement : chaque fois que nous utilisons un framework, une bibliothèque ou un logiciel, nous entrons en collaboration indirecte avec ses créateurs. Ces outils sont le fruit de connaissances accumulées, d'expertises partagées et d'efforts collectifs. En les utilisant, nous nous appuyons sur le génie et l'ingéniosité de nombreux autres esprits, créant ainsi une synergie qui transcende les frontières individuelles. Cette réalité met en lumière l'importance fondamentale de la collaboration dans le domaine du développement de logiciels, soulignant que chaque création est, en quelque sorte, un effort collectif.

### Rôle de la conception dans la collaboration

La phase de conception dans le développement de logiciels joue un rôle crucial en matière de collaboration, agissant comme la pierre angulaire pour établir et partager une vision commune parmi toutes les parties prenantes. Cette étape est essentielle non seulement pour faciliter la collaboration entre les développeurs, mais également pour impliquer les parties prenantes non techniques dans le processus de création. Elle assure une compréhension claire et partagée des objectifs et fonctionnalités du logiciel, ce qui est vital pour le succès du projet.

Par exemple, les diagrammes présentés dans la [section « Diagramme de cas d'utilisations et diagrammes UML »](#) ne sont pas uniquement des outils de conception ; ils constituent également des moyens de collaboration efficaces. Ils permettent aux développeurs de partager une vision commune de la structure et de la logique du système, favorisant ainsi une communication claire et une compréhension partagée. Cette approche collaborative est indispensable pour garantir que tous les développeurs soient alignés sur les objectifs du projet, réduisant ainsi les risques de malentendus et d'erreurs pendant le développement.

Ces diagrammes, et en particulier le diagramme de cas d'utilisation, jouent un rôle crucial non seulement pour la conception technique, mais aussi pour la collaboration étendue avec toutes les parties prenantes du projet. Ils garantissent que les développeurs, les représentants des utilisateurs, les investisseurs, l'équipe marketing, l'équipe de vente, les spécialistes de la publicité, et même les juristes impliqués dans les questions de propriété intellectuelle et de conformité légale partagent une vision commune du projet.

Comme l'explique en détail le [livre « Use Case Modeling »](#) de Kurt Bittner et Ian Spence téléchargeable depuis [dokumen.pub/qdownload](#), le diagramme de cas d'utilisation est particulièrement utile pour cette collaboration étendue. Il présente une vue globale et accessible de l'application, en termes compréhensibles pour tous. Cela aide les parties prenantes non techniques, telles que les équipes de vente et de marketing, à comprendre le produit et à élaborer des stratégies efficaces pour sa promotion et sa vente. De plus, il fournit aux investisseurs et aux juristes une vision claire de la portée et des fonctionnalités du projet, facilitant les discussions sur le financement, les droits de propriété intellectuelle, et la conformité réglementaire.

Cette approche collaborative globale est essentielle pour le succès du projet. Elle assure que tous les aspects, des attentes des utilisateurs aux contraintes légales, sont pris en compte dès le début du processus de développement. Cela minimise les risques de malentendus, d'ajustements tardifs coûteux et de non-conformité.



## Rôle de la spécification dans la collaboration

Dans notre projet, l'adoption de la spécification Jakarta EE joue un rôle crucial dans la facilitation de la collaboration. Un exemple concret est l'utilisation des Jakarta Servlets, une partie intégrante de Jakarta EE, pour le traitement des requêtes et des réponses sur le serveur. Cette spécification offre un cadre standardisé pour développer ces composants, garantissant ainsi que tous les développeurs suivent les mêmes pratiques et standards.

Cette uniformité est essentielle pour la collaboration. En travaillant avec les Jakarta Servlets, chaque membre de l'équipe comprend non seulement comment implémenter les fonctionnalités serveur de manière cohérente, mais aussi comment ces composants s'intègrent dans l'architecture globale de l'application. Cela réduit significativement les risques de conflits dans le code et assure une meilleure cohérence dans le développement.

En somme, la spécification Jakarta EE, notamment à travers les Jakarta Servlets, offre un langage commun et des directives claires, facilitant une collaboration efficace et harmonieuse au sein de notre équipe de développement.

## Réutilisation de code source

La réutilisation de code source est un aspect fondamental de la collaboration en développement logiciel, que ce soit avec d'autres développeurs ou pour soi-même dans des projets futurs. Elle souligne l'importance d'une bonne conception, respectueuse des bonnes pratiques et accompagnée de commentaires pertinents, pour faciliter la compréhension et l'adaptation du code.

Dans le cadre de notre projet, nous avons intégré une application préexistante, contenue dans le répertoire **LegacyMVC\_Codebase**. Ce code, développé en Java pour un cours de programmation objet et basé sur le modèle MVC (Modèle-Vue-Contrôleur), propose une expérience de jeu contre un autre utilisateur ou une intelligence artificielle utilisant l'algorithme min-max. La réutilisation de ce code préexistant offre une opportunité précieuse d'apprendre à travailler avec des bases de code existantes, mettant en lumière l'importance de la documentation et de la qualité de conception initiale.

En réutilisant ce code, nous pouvons non seulement gagner du temps de développement, mais aussi analyser et apprendre de la structure MVC utilisée. Cela nous permet de contraster cette architecture avec l'architecture trois-tiers que nous adoptons pour notre nouveau projet, fournissant ainsi une compréhension approfondie des différences et avantages de chaque approche.

Pour construire, exécuter puis nettoyer le projet du répertoire **LegacyMVC\_Codebase**, suivez ces étapes :

- installer Java
- se positionner dans le répertoire **LegacyMVC\_Codebase**
- pour compiler le programme, soumettre au terminal la commande :

```
javac *.java $(find ../LegacyMVC_Codebase -type f -name "*.java") -cp "../:*"
```

- pour exécuter le programme, soumettre au terminal la commande :

```
java Main
```

- pour nettoyer le projet, soumettre au terminal la commande :

```
find ../LegacyMVC_Codebase -type f -name "*.class" -delete
```

## Application de la gestion des versions à la collaboration

**Utilisation du logiciel de gestion des versions Git** Dans notre projet, Git joue un rôle clé en facilitant la collaboration des développeurs. Git permet à chaque développeur de travailler sur une fonctionnalité spécifique dans une branche séparée. Ces branches permettent aux développeurs de travailler de manière isolée, sans affecter le code principal. Lorsqu'un développeur termine son travail sur une fonctionnalité, il effectue une fusion de sa branche avec la branche principale. Cette fusion est le processus d'agrégation du travail effectué dans les différentes branches, permettant d'assembler les fonctionnalités développées séparément en un produit cohérent. Cette méthode assure que le code principal reste stable tout en permettant l'intégration progressive des nouvelles fonctionnalités.

**Utilisation d'un service de gestion de version et d'hébergement de code source GitHub** GitHub, quant à lui, agit comme une plateforme centralisée où ce code est stocké et partagé. Il permet à tous les membres de l'équipe de voir le travail en cours, d'accéder aux différentes branches et de suivre les modifications effectuées via les commits. En utilisant GitHub, les développeurs peuvent facilement collaborer, peu importe leur emplacement géographique. Ils peuvent également voir l'historique complet des changements et gérer les différentes versions du projet. Ceci est crucial pour synchroniser le travail d'équipe, intégrer les contributions de chaque membre et maintenir une vue d'ensemble claire de l'évolution du projet.

## Rôle des bonnes pratiques dans la collaboration

Pour garantir le succès d'un projet de développement logiciel, il est crucial d'adopter des bonnes pratiques de conception et de développement. Chacune de ces pratiques joue un rôle spécifique dans la facilitation de la collaboration au sein de l'équipe. Notre projet intègre ces bonnes pratiques pour créer un logiciel modulaire et facile à maintenir, ce qui est fondamental dans un environnement de développement agile.



**Séparation des préoccupations** Dans notre projet, la séparation des préoccupations est illustrée par l'architecture multi-tiers, où nous avons distinctement séparé les couches Présentation, Métier et Persistance. Par exemple, les modules webapp et webapp.servlet dans la couche Présentation sont clairement dissociés des modules business et business.ai dans la couche Métier. Cette séparation facilite les mises à jour indépendantes et améliore la maintenabilité du code.

**Couplage faible** Le principe de couplage faible est mis en œuvre dans notre projet par la réduction des dépendances entre les différents modules et packages. Par exemple, le module consumer dans la couche Persistance fonctionne indépendamment des modules dans les autres couches. Cela permet des développements et des tests isolés, réduisant les risques de conflits dans le code.

**Gestion efficace des versions** Notre utilisation de Git et GitHub pour la gestion de versions illustre l'importance de cette bonne pratique. Ces outils nous permettent de gérer efficacement les différentes branches de développement, facilitant ainsi la collaboration et l'intégration des fonctionnalités développées parallèlement.

**Autres Pratiques Essentielles** D'autres pratiques, comme la génération de documentation et de rapports sur la qualité du code via Maven, jouent un rôle crucial dans notre projet. Elles garantissent que le code reste accessible et compréhensible pour tous les membres de l'équipe, favorisant ainsi une collaboration continue et adaptative. De plus, l'adoption de patrons de conception tels que DAO et Singleton dans notre architecture contribue à une structure de code cohérente et efficace.

## Utilisation du système de gestion de bases de données relationnelles MySQL

Nous intégrerons MySQL, un système de gestion de base de données relationnelle, pour le stockage et la gestion des données de notre application.

## Utilisation des environnements de développement Eclipse et IntelliJ

Le développement de notre application s'effectuera en utilisant les IDE Eclipse et IntelliJ IDEA, qui offrent des fonctionnalités avancées pour la programmation et la gestion de projets.

## Déploiement pour le travail en développement

### Utilisation d'une plateforme de déploiement fournie par le serveur d'application Tomcat

Notre projet fera appel à Apache Tomcat, un serveur d'applications web léger et open-source, qui servira de plateforme de déploiement pour notre application web.

## Déploiement pour le travail en production

### Utilisation du service d'hébergement de machine virtuelle DigitalOcean

Pour l'hébergement de notre application web, nous avons choisi DigitalOcean en raison de sa simplicité d'utilisation, de sa tarification transparente et de ses performances fiables. DigitalOcean offre une gamme de droplets (machines virtuelles), ce qui nous permet de sélectionner une configuration qui correspond aux besoins spécifiques de notre projet en termes de mémoire, de CPU, et de stockage.

L'avantage principal de DigitalOcean réside dans sa facilité de configuration et de gestion des machines virtuelles. Cela nous permet de déployer rapidement notre application web sans la complexité souvent associée à la configuration et à la gestion d'infrastructures serveur. De plus, DigitalOcean propose une intégration facile avec divers outils de développement et de déploiement, ce qui est idéal pour automatiser et simplifier nos processus de CI/CD (Intégration Continue et Déploiement Continu).

Un autre aspect important est la tarification prévisible de DigitalOcean, ce qui aide à maîtriser les coûts du projet, surtout important pour les projets ayant des budgets limités. En outre, DigitalOcean offre un excellent support technique et une vaste communauté d'utilisateurs, ce qui est un atout précieux pour résoudre rapidement les problèmes et partager les meilleures pratiques.

### Utilisation du registraire de domaine Namecheap

#### Utilisation du service DNS [TO DO]

#### Utilisation d [TO DO]

#### [TO DO]

## Outils utilisés

### 1. Conception :

- a. Architecture : multi-tiers
- b. Patrons de conception : MVC, DAO
- c. Modélisations des données :
  - i. Diagrammes :
    - A. Diagramme de cas d'utilisation
    - B. Diagramme entités associations
    - C. Diagramme des classes
  - ii. Outil de conception de base de données visuel : MySQL Workbench
  - iii. Outils de conception de diagrammes UML : UMLet, draw.io

### 2. Spécifications : Java SE, Jakarta EE

- a. Une **spécification** est un ensemble de normes et de règles.
- b. Synonymes de spécifications :
  - i. Modèle de Conception : Un terme qui indique un plan ou un cadre pour la façon dont quelque chose doit être construit ou mis en œuvre.
  - ii. Cahier des Charges : Souvent utilisé en ingénierie et en conception, cela implique une liste détaillée des exigences et des fonctionnalités attendues.
  - iii. Protocole : Bien qu'habituellement utilisé dans le contexte des communications, il peut être vu comme un ensemble de règles et de normes qui définissent les interactions dans un système.
  - iv. Guide de Normes : Un document ou ensemble de règles qui définissent les attentes pour un domaine particulier.
- c. Bibliothèques tierces relevant de la spécification Jakarta EE :
  - i. jakarta.servlet:jakarta.servlet-api:6.0.0
  - ii. org.glassfish.web:jakarta.servlet.jsp.jstl:3.0.0
  - iii. jakarta.servlet.jsp.jstl:jakarta.servlet.jsp.jstl-api:3.0.0
  - iv. org.mindrot:jbcrypt:0.4

### TO DO

### 3. Bonnes pratiques :

- a. Tests
  - i. TDD (Test Driven Development)
  - ii. Suivre le principe FIRST (Fast, Isolate, Repeatly, Self-validating, Timely (maintenant Thourugh))
  - iii. Suivre la méthode AAA : Arrange, Act, Assert
- b. Intégration Continue (Continuous Integration - CI)  
Automatisez la construction, l'exécution des tests et l'analyse de code à chaque soumission de code dans le système de contrôle de version. Cela aide à détecter rapidement les erreurs et à maintenir la qualité du code.
- c. Revue de Code (Code Review)  
Pratiquez des revues de code régulières pour améliorer la qualité du code et partager les connaissances au sein de l'équipe. Cela permet d'identifier les erreurs, d'améliorer les approches de codage et de maintenir un niveau élevé de compréhension du code au sein de l'équipe.
- d. Refactoring  
Améliorez et nettoyez régulièrement le code existant pour améliorer sa lisibilité, sa maintenabilité et ses performances sans changer son comportement externe. Cela aide à garder le codebase sain et facile à évoluer.
- e. Programmation en Paire (Pair Programming) si possible  
Pratiquez la programmation en paire, où deux développeurs travaillent ensemble sur le même code. Cela permet non seulement de produire un code de meilleure qualité mais aussi de transférer les connaissances au sein de l'équipe.
- f. Documentation  
Maintenez une documentation à jour pour le code, les systèmes et les processus. Cela aide les nouveaux membres de l'équipe à monter en compétence plus rapidement et sert de référence utile pour l'équipe.
- g. Gestion des Dépendances  
Gérez soigneusement les bibliothèques et les dépendances externes pour éviter les conflits et garantir la compatibilité. Utilisez des outils de gestion des dépendances comme Maven, Gradle, npm, etc.

- h. Principes SOLID  
Appliquez les principes SOLID (Single Responsibility, Open-closed, Liskov Substitution, Interface Segregation, Dependency Inversion) pour concevoir des classes et des systèmes flexibles, maintenables et évolutifs.
- i. Modélisation et Design Patterns  
Utilisez des modèles de conception appropriés pour résoudre des problèmes récurrents de manière efficace et élégante. La modélisation, telle que les diagrammes UML, peut également aider à la compréhension et à la communication autour de la structure et du comportement des systèmes.
- j. Sécurité  
Intégrez les considérations de sécurité dès le début du cycle de développement et appliquez les meilleures pratiques de sécurité pour protéger votre application des vulnérabilités.
- k. Performance et Optimisation  
Surveillez les performances de votre application et optimisez le code et les ressources systèmes pour garantir une expérience utilisateur fluide et efficace.
- l. Accessibilité  
Assurez-vous que vos applications sont accessibles à tous les utilisateurs, y compris ceux ayant des besoins spécifiques, en suivant les directives d'accessibilité web (WCAG).
- m. Conventions de codage :
  - i. Structure du projet et nomination des modules :
    - A. Nom des modules : Utilisez des noms descriptifs et cohérents pour les modules Maven, reflétant leur rôle au sein de l'architecture. Par exemple, webapp, service, repository, model, etc.
    - B. Structure des répertoires : Suivez la structure standard de Maven (src/main/java, src/main/resources, src/test/java, etc.) pour une organisation claire et une meilleure intégration avec les outils.
  - ii. Conventions de Nommage :
    - A. Classes et Interfaces : Utilisez le CamelCase avec la première lettre en majuscule. Les noms doivent être descriptifs et refléter leur rôle (e.g., CustomerService, OrderRepository).
    - B. Méthodes : Utilisez le lowerCamelCase et nommez les méthodes de manière à refléter l'action effectuée et/ou le résultat attendu (e.g., findById, saveCustomer).
    - C. Variables : Utilisez également le lowerCamelCase et assurez-vous que les noms sont descriptifs (e.g., customerList, orderForm).
    - D. Constantes : Utilisez des lettres majuscules avec des underscores pour séparer les mots (e.g., MAX\_SIZE, DEFAULT\_TIMEOUT).
  - iii. Style de Codage :
    - A. Indentation : Utilisez 4 espaces pour l'indentation et non des tabulations pour assurer une cohérence visuelle dans différents éditeurs.
    - B. Accolades : Utilisez le style "K&R" où les accolades ouvrantes sont à la fin de la ligne et les fermantes sur leur propre ligne, alignées verticalement avec la ligne de début du bloc.
    - C. Longueur de ligne : Limitez la longueur des lignes à 120 caractères pour faciliter la lecture du code sans avoir besoin de faire défiler horizontalement.
  - iv. Commentaires et Documentation :
    - A. Commentaires : Utilisez les commentaires pour expliquer le "pourquoi" derrière des blocs de code complexes, pas le "quoi" que le code fait déjà.
    - B. Javadoc : Documentez toutes les classes publiques et les méthodes avec Javadoc, en décrivant le but, les paramètres, le retour et les exceptions lancées.
  - v. Gestion des Exceptions
    - A. Utilisation des exceptions : Préférez les exceptions spécifiques au contexte plutôt que les exceptions génériques (e.g., OrderNotFoundException au lieu de Exception).
    - B. Journalisation : Loggez les exceptions à un niveau approprié (ERROR, WARN) avec un message descriptif, en incluant le stack trace si nécessaire.
  - vi. Tests
    - A. Nom des classes de test : Suivez le format NomDeLaClasseATesterTests (e.g., CustomerServiceTests).
    - B. Couverture des tests : Visez une couverture de test significative sur les logiques métier, en incluant des tests unitaires pour les composants individuels et des tests d'intégration pour les interactions entre composants.
  - vii. Utilisation de Maven :
    - A. Dépendances : Gérez les versions des dépendances dans la section <dependencyManagement> du POM parent pour une gestion centralisée.

B. Plugins : Configurez les plugins nécessaires dans <pluginManagement> pour une utilisation cohérente dans tous les modules.

4. Bibliothèques tierces ne relevant pas de la spécification Jakarta EE :

- a. org.junit:jupiter:junit-jupiter:5.10.1
- b. org.mockito:mockito-core:5.10.0
- c. org.mockito:mockito-junit-jupiter:5.10.0
- d. com.mysql:mysql-connector-j:8.2.0
- e. org.mindrot:jbcrypt:0.4

TO DO

5. Hardware :

- a. Architecture : x86-64
- b. Hardware Vendor : ASUSTeK COMPUTER INC.
- c. Hardware Model : X705UAP

6. Système d'exploitation : Ubuntu

- a. Version : 22.04.3 LTS
- b. Noyau : Linux 6.2.0-37-generic

7. JDK (Java Développement Kit)

- a. Version : 21.0.1

8. serveur d'application : Tomcat

- a. Version : 10.1.16

9. IDE : Eclipse, IntelliJ

- a. Version de Eclipse : eclipse-jee-2023-09-R
- b. Version de IntelliJ : idea-IC-232.10227.8

10. Outil de compréhension et de gestion de développement de projets logiciels : Apache Maven

- a. Version : 3.9.5
- b. Plugins Apache Maven :
  - i. org.apache.maven.plugins:maven-compiler-plugin:3.12.1
  - ii. org.apache.maven.plugins:maven-clean-plugin:3.3.2
  - iii. org.apache.maven.plugins:maven-resources-plugin:3.3.1
  - iv. org.apache.maven.plugins:maven-surefire-plugin:3.2.5
  - v. org.apache.maven.plugins:maven-war-plugin:3.4.0
  - vi. org.apache.maven.plugins:maven-jar-plugin:3.3.0
  - vii. org.apache.maven.plugins:maven-install-plugin:3.1.1
  - viii. org.apache.maven.plugins:maven-deploy-plugin:3.1.1

TO DO

11. SGBDR : MySQL

- a. Version : 8.0.35
- b. Langage : SQL

12. Logiciel de gestion de versions : Git

- a. Version : 2.34.1

13. Logiciel de visualisation de l'historique des versions : Gitk

- a. Visualiseur de commit pour git
- b. Copyright \u00a9 2005-2016 Paul Mackerras
- c. Utilisation et redistribution soumises aux termes de la GNU General Public License

14. Service d'analyse statique du code SonarCloud :

SonarCloud appartient à la catégorie des outils d'analyse statique du code en tant que service (SaaS). Il est conçu pour intégrer et améliorer les workflows de DevOps cloud en permettant aux équipes de livrer un code propre de manière cohérente et efficace. SonarCloud se spécialise dans l'examen du code en ligne, s'intégrant facilement aux plateformes DevOps cloud et étendant le workflow CI/CD de votre projet.

## Construction de l'environnement de développement

- télécharger un JDK
  - Version : 21.0.1
  - url de la page de téléchargement  
<https://www.oracle.com/java/technologies/downloads/>
  - url de l'archive  
[https://download.oracle.com/java/21/latest/jdk-21\\_linux-x64\\_bin.tar.gz](https://download.oracle.com/java/21/latest/jdk-21_linux-x64_bin.tar.gz)
  - x64-Compressed Archive for Linux
  - vérification de la checksum
    - Lien de la checksum 256 : [https://download.oracle.com/java/21/latest/jdk-21\\_linux-x64\\_bin.tar.gz.sha256](https://download.oracle.com/java/21/latest/jdk-21_linux-x64_bin.tar.gz.sha256)
    - checksum sha256  
7c1f7689db0f4b48ee6978029c4a1aec1442a8a7637cdf43a5471d0c79712a8
    - vérification :  
`sha256sum ../env/java/jdk-21_linux-x64_bin.tar.gz`
  - répertoire de téléchargement : env/java/
  - décompression :  
`tar -xzf ../env/java/jdk-21_linux-x64_bin.tar.gz -C ../env/java/jdk-21.0.1/`  
`rm ../env/java/jdk-21_linux-x64_bin.tar.gz`
  - Initialisation de la version du JDK utilisée dans le répertoire de développement :  
`ln -s ~/env/java/jdk-21.0.1 ~/env/java/current`
- télécharger Apache Maven
  - version : 3.9.5
  - url de la page de téléchargement  
<https://maven.apache.org/download.cgi>
  - url de l'archive  
<https://dlcdn.apache.org/maven/maven-3/3.9.5/binaries/apache-maven-3.9.5-bin.tar.gz>
  - Binary tar.gz archive
  - vérification de la checksum
    - Lien de la checksum sha512  
<https://downloads.apache.org/maven/maven-3/3.9.5/binaries/apache-maven-3.9.5-bin.tar.gz.sha512>
    - checksum sha512  
4810523ba025104106567d8a15a8aa19db35068c8c8be19e30b219a1d7e83bcab96124bf86dc424b1cd3c5edba25d69ec0b31751c136f88975d15406cab3842b
    - lien de la signature  
<https://downloads.apache.org/maven/maven-3/3.9.5/binaries/apache-maven-3.9.5-bin.tar.gz.asc>
    - Signature :  
-----BEGIN PGP SIGNATURE-----  
  
iQIzBAABCGAdFiEEKb6ipkXy1s7X+xLgKxcuPhVkZugFAMUax+kACgkQKxcuPhVk  
Zujzqw/8DKGZ97ckuqu3D0Y/QnfQDfD9fqBp+Y+9JNUa/6ZV0Sx1+++4wD+Avn8Y  
pzcJKKFqX31TmhqD4y+WEWih2cXR3Y70asIR6yBi8uVS4Q4FswlZv+yKVSnyo7Bv  
XbxDLcv7/9VlAj74vEf0LWmYvaKqqbj5ZkJP76/51XlELkR1taZGzS49Bhia4hl  
q13TB77Sfwx1VkpD0EcrffLwe+u3gjEQdov5mhS8n0MXmCudZk7/hX07SUTwiR0e  
3dJgP0c62op11DWbam/brm9Mz3VvZrWw/XwMXQmDs//3dls44WeEuEMU6f+p5flj  
as2SotYt1hLRDDfjBY1tvLRKA+62mvc7V2SWTBIkBzequTFJan/vvHA2RvQlXhEk  
rEKcvNAFEAFaPUYVg6i/hyoACWjAS1Xw2X0Av10hPi19Zw+GmRKV0zT5GSRB8udP  
EFB850u0do6N5jif0kQbUff1Y10S8I1i0m9lidG+VbYT372jMmf65NrrffxLHF6t  
gmfrQ558x9kFbnV9gsCcaWpZ1TqN34dbPrdD2JUn0KYJkeB9/wxL0epSo502Zok+  
tSz3xUquhHwsY7Wqp4ef3rxEqiMNMqrxw5aT0Xv5PRgyDTucic8I5zLDiHfXkZbk  
+UU+xE3bmPBWalIoKfv6aYXaiUGRRZG5CWdQi+98n2EGBJaUclM=  
=pf8d  
-----END PGP SIGNATURE-----
    - vérification :  
`sha512sum ../env/maven/apache-maven-3.9.5-bin.tar.gz`

- répertoire de téléchargement : `env/maven/`
- décompression :  

```
tar -xzf ../env/maven/apache-maven-3.9.5-bin.tar.gz -C ../env/maven/
rm ../env/maven/apache-maven-3.9.5-bin.tar.gz
```
- Initialisation de la version de Apache Maven utilisée dans le répertoire de développement :  

```
ln -s ~/env/maven/apache-maven-3.9.5 ~/env/maven/current
```
- définir des variables d'environnement relatives à Java et à Apache Maven
  - Pourquoi : pour que les exécutables des logiciels Java et Apache Maven soit recherchés par le système dans les répertoires relatifs à ces deux logiciels que l'on vient d'installer.
  - Ajout des lignes suivantes à la fin de `~/ .bashrc`  

```
# developpement directory ~/env/ configuration
# Java
export JAVA_HOME="/home/workaholic/env/java/jdk-21.0.1"
export MAVEN_HOME="/home/workaholic/env/maven/apache-maven-3.9.5"
export PATH="$JAVA_HOME/bin:$MAVEN_HOME/bin:$PATH"
```
  - explication : `./bashrc` est exécuté par toute session du shell pour se configurer à son lancement relative à « l'utilisateur de `~` »
  - Dans une nouvelle session du shell  

```
java -version
mvn -v
```
- configurer Apache Maven
  - Lors de l'exécution de la première instruction qui déclenche le processus de résolution des dépendances ou d'autres interactions avec le repository local (comme par exemple « `mvn help:help` »), Apache Maven se configure via `~/env/maven/apache-maven-3.9.5/conf/settings.xml` en créant notamment le répertoire `~/ .m2/` ; cette information n'est pas dans le fichier de configuration `settings.xml`, elle est dans le schéma que ce dernier respecte et récupérable via l'url <https://maven.apache.org/xsd/settings-1.2.0.xsd>
  - preuve :  

```
cd ~/env/maven
grep -ri "\.m2/settings\.xml"
grep -ri "\.m2/repository"
cat ~/env/maven/apache-maven-3.9.5/conf/settings.xml | grep xsi:schemaLocation=
wget https://maven.apache.org/xsd/settings-1.2.0.xsd
cat settings-1.2.0.xsd | grep -n '\.m2'
rm settings-1.2.0.xsd
```

```
$ pwd
/home/workaholic/env/maven
$ grep -ri "\.m2/settings\.xml"
apache-maven-3.9.5/conf/settings.xml: | and is normally provided in ${user.home}/.m2/settings.xml.
$ grep -ri "\.m2/repository"
apache-maven-3.9.5/conf/settings.xml: | Default: ${user.home}/.m2/repository
$ cat ~/env/maven/apache-maven-3.9.5/conf/settings.xml | grep xsi:schemaLocation=
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.2.0 https://maven.apache.org/xsd/settings-1.2.0.xsd"
$ wget https://maven.apache.org/xsd/settings-1.2.0.xsd
--2023-11-15 09:31:03-- https://maven.apache.org/xsd/settings-1.2.0.xsd
Résolution de maven.apache.org (maven.apache.org)... 2a04:4e42::644, 151.101.2.132
Connexion à maven.apache.org (maven.apache.org)[2a04:4e42::644]:443... connecté.
requête HTTP transmise, en attente de la réponse... 200 OK
Taille : 30831 (30K) [application/xml]
Enregistre : 'settings-1.2.0.xsd'

settings-1.2.0.xsd 100%[=====] 30,11K --.-KB/s ds 0,01s
2023-11-15 09:31:04 (2,04 MB/s) - 'settings-1.2.0.xsd' enregistré [30831/30831]
$ cat settings-1.2.0.xsd | grep -n '\.m2'
46: The local repository.<br><b>Default value is:<br><code>${user.home}/.m2/repository</code>
$ rm settings-1.2.0.xsd
$
```

- création du fichier de configuration optionnel de Apache Maven `~/ .m2/settings.xml` en vu de personnaliser sa configuration ; on y recopie pour l'instant son fichier de configuration par défaut `~/env/maven/apache-maven-3.9.5/conf/settings.xml`  

```
cp ~/env/maven/apache-maven-3.9.5/conf/settings.xml ~/ .m2/
```
- modification du répertoire de dépôt local `~/ .m2/repository` de Apache Maven
  - création d'un répertoire de dépôt local `~/env/maven/repository` (au sein du répertoire de développement) pour Apache Maven  

```
mkdir ~/env/maven/repository
```
  - configuration de l'emplacement `~/env/maven/repository` pour le nouveau répertoire de dépôt local en modifiant les lignes suivantes dans `~/ .m2/settings.xml`

```
| Default: ${user.home}/.m2/repository -->
<localRepository>${user.home}/env/maven/apache-maven-3.9.5/repository</localRepository>
```

- Maintenant l'instruction shell « `mvn help:help` » remplit ce répertoire  
`~/env/maven/apache-maven-3.9.5/repository`

- télécharger Eclipse

- Version : Eclipse for JEE developpers de la révision de septembre 2023 (« 2023-09-R »)
- url de la page de téléchargement

[http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2023-09/R/eclipse-jee-2023-09-R-linux-gtk-x86\\_64.tar.gz](http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2023-09/R/eclipse-jee-2023-09-R-linux-gtk-x86_64.tar.gz)

- url de l'archive

[https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2023-09/R/eclipse-jee-2023-09-R-linux-gtk-x86\\_64.tar.gz&mirror\\_id=1045](https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2023-09/R/eclipse-jee-2023-09-R-linux-gtk-x86_64.tar.gz&mirror_id=1045)

- x64-Compressed Archive for Linux
- vérification de la checksum

- Lien de la checksum 512 :

[https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2023-09/R/eclipse-jee-2023-09-R-linux-gtk-x86\\_64.tar.gz&btn-ajax-checksum-sha512sum](https://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/2023-09/R/eclipse-jee-2023-09-R-linux-gtk-x86_64.tar.gz&btn-ajax-checksum-sha512sum)

- checksum sha512

46d02838283b2f84b26b5d2a39c223f14c88a9cae1e22a77666db78664dc292dc3482e3f68fd4fa77fcdc9f18b40e52c366d57aad1929ac03a26684842a565bc

- vérification :

`sha512sum ../env/eclipse/eclipse-jee-2023-09-R-linux-gtk-x86_64.tar.gz`

- répertoire de téléchargement : `env/eclipse/`

- décompression :

```
tar -xzvf ../env/eclipse/eclipse-jee-2023-09-R-linux-gtk-x86_64.tar.gz -C ../env/eclipse/
rm ../env/eclipse/eclipse-jee-2023-09-R-linux-gtk-x86_64.tar.gz
```

- Initialisation de la version de Eclipse utilisée dans le répertoire de développement :

`ln -s ~/env/eclipse/eclipse-jee-2023-09-R ~/env/eclipse/current`

- télécharger IntelliJ

- Version : `ideaIC-2023.2.5`

- url de la page de téléchargement

<https://www.jetbrains.com/idea/download/?section=linux>

- url de l'archive

<https://www.jetbrains.com/idea/download/download-thanks.html?platform=linux&code=IIC>

- x64-Compressed Archive for Linux

- vérification de la checksum

- Lien de la checksum 256 :

<https://download.jetbrains.com/idea/ideaIC-2023.2.5.tar.gz.sha256>

- checksum sha256

4fc5817f8bfd86bdb3af924d3ca32e349517710ac5f986ae20f765f79285e00b

- vérification :

`sha256sum ../env/intellij/ideaIC-2023.2.5.tar.gz`

- répertoire de téléchargement : `env/intellij`

- décompression :

```
tar -xzvf ../env/intellij/ideaIC-2023.2.5.tar.gz -C ../env/intellij/
rm ../env/intellij/ideaIC-2023.2.5.tar.gz
```

- Initialisation de la version de IntelliJ utilisée dans le répertoire de développement :

`ln -s ~/env/intellij/idea-IC-232.10227.8 ~/env/intellij/current`

- télécharger Tomcat

- version : `10.1.16`

- url de la page de téléchargement

<https://tomcat.apache.org/download-10.cgi>

- url de l'archive

<https://dlcdn.apache.org/tomcat/tomcat-10/v10.1.16/bin/apache-tomcat-10.1.16.tar.gz>

- Binary tar.gz archive

- vérification de la checksum



- Lien de la checksum sha512  
<https://downloads.apache.org/tomcat/tomcat-10/v10.1.16/bin/apache-tomcat-10.1.16.tar.gz.sha512>
- checksum sha512  
d469d0c68cf5e321bbc264c3148d28899e320942f34636e0aff3d79fc43e8472cd0420d0d3df5ef6ece4be4810a3f8fd518f605c5a9c13cac4e8f96f5f138e92
- lien de la signature  
<https://downloads.apache.org/tomcat/tomcat-10/v10.1.16/bin/apache-tomcat-10.1.16.tar.gz.asc>
- Signature :

-----BEGIN PGP SIGNATURE-----

```
iQIzBAABCAAdFiEEMmKgYcQvxMe7tcJcHPApP6U8pFgFamV0WL4ACgkQHPApP6U8
pFgRyA/8CxKqVcNErsRuDUHmPL/HMl9w7VEkQ1VL0eL8NQIRLcoh3QkRnT322SSN
RyYrUbSCYP0Q5fAvn5PPTra2fo8V7ZjETpTUAZMwFgZIU3Lj+Pvyt1BhlX/RcPFS
Viul05PHLW4odA4n1mrKtDKzF1pstH0njZ2lg44q6SoULeetZb3Ii07WP0dsMAvk
HUfhWAhZ/m1cBCs98IMV+xHCJME9LZcoYtBuLV3VBXdc0NcS8r+0LgK9x9iUMTPP
+AZYJiKM3sCCUGCEVrl25pLAGMNuybVp8x1f/j/Ked8pAuiiLwQUqaZbS4km/v0Z
mTPgMCXgtqWV1CM35E/mduDPPRma5X13m8/ZU2aKLhypWE2cGNgexKbxupD7UDk1
f4LdsiRasemFNtpyvUhd7xZlXu4fMLxZLDDNudn/3RGqAyjuWagtHszpSwTJqvGq
20UBLa2YfmMCQtXc2yICh/ZXMfRcYRj5WX6aeh20CtspA2owLce1DXFNQue+3rpK
4mbVDxnAb+ZSJFC8L2ispqPd88lUNayjqsT8201/m+ZdJT1uRYU7CtmtExa89aKK
6kPbMDSDMoLn0DwlALpEjjsnknjU0WpbEDVG0VHvmxtEJeuA+uHGekr6psL+1tJu
VUVn4PAW+xYM2X+EbMdQERAZhbZ+q4gZ1EFc8LAYPr2j0336KFw=
=wy+l
```

-----END PGP SIGNATURE-----

- vérification :

```
sha512sum ../env/tomcat/apache-tomcat-10.1.16.tar.gz
```

- répertoire de téléchargement : env/tomcat/

- décompression :

```
tar -xzf ../env/tomcat/apache-tomcat-10.1.16.tar.gz -C ../env/tomcat/
rm ../env/tomcat/apache-tomcat-10.1.16.tar.gz
```

- Initialisation de la version de Tomcat utilisée dans le répertoire de développement :

```
ln -s ~/env/tomcat/apache-tomcat-10.1.16 ~/env/tomcat/current
```

- télécharger le serveur et le client MySQL

- version : 8.2.0

- nom de l'archive : mysql-server\_8.2.0-1ubuntu22.04\_amd64.deb-bundle

- url de la page de téléchargement

<https://dev.mysql.com/downloads/mysql/>

- url de l'archive

[https://dev.mysql.com/get/Downloads/MySQL-8.2/mysql-server\\_8.2.0-1ubuntu22.04\\_amd64.deb-bundle.tar](https://dev.mysql.com/get/Downloads/MySQL-8.2/mysql-server_8.2.0-1ubuntu22.04_amd64.deb-bundle.tar)

- POSIX tar archive (GNU)

- vérification de la checksum

- Lien de la checksum md5

<https://dev.mysql.com/downloads/mysql/>

- checksum MD5

dc4c37b9fb24352c5355a87dbbb8471d

- lien de la signature

[https://dev.mysql.com/downloads/gpg/?file=mysql-server\\_8.2.0-1ubuntu22.04\\_amd64.deb-bundle.tar&p=23](https://dev.mysql.com/downloads/gpg/?file=mysql-server_8.2.0-1ubuntu22.04_amd64.deb-bundle.tar&p=23)

- Signature :

-----BEGIN PGP SIGNATURE-----

```
iQIzBAABCAAdFiEEhZvo18WG9ThDCxnCRnuULTp5vSkFamUpCBsACgkQRnuULTp5
vSm3vg//W+YAEkX86/AjhxR2oJtXLHFamg4Jqc/g0fPb4g7SN4T0hZEHB+7kdTIQ
YSyrlmz3W9VsFkizDhTSAjiYF3ZEt+W1r6seMCKqTFn0K0ISrvjzyow58bdTNKy9
rXNu6rr/fM1Y9nUMsGs7W8Dlj+4QTj8KMPGhD/+2DjLN455c4wXj2dF/UNTLsVwd
gPxfNY1bXafYX0uKxEjJoPoY8isnClVUx9qsFf50i4kW7EHpIa73KULTDN/zyPE
```

```

UTlQ3cw8fpF2neop50Kp0ermt7E59iMIlCvW8DbqDXwuxzxep8CLK0szFPuXNrGy
h0HvD3mNXjo1U+zsaQ0JA9UvWt8ae12d9xGB4lbot8uLn2PZLfdnp4Lk7uDejU1h
Pi49S27DEc7Hwy3Ws/L5b/y8yAPF2t6abj72ex5dGxU4mhR0XYm8vc/R0XQSDpAk
aBTu6asv8ntr2wB0zL1VqiPg64aAeF6vA+mE+JAoKIXjyIVzL0wZhLfefoxl+sW
6bB+Umvk/WMCPdnfe3kxauo3Dw0/XwIRdXywCBrchn8DmTdDdAIBsx4BdDqUIJN1
pjCVMqzwxYcV+xJTPI5D2n7y96XudU7VBRTL4WEF56antAXfsTo6BoWs198YibD
jB92ySL6cv1tX3lS7ZHi/bTMgkhNNH/1i6CgG7/lW21TD2RwCbC=
=IBRL

```

-----END PGP SIGNATURE-----

- vérification :

```
md5sum ../env/sbdr/mysql/mysql-server_8.2.0-1ubuntu22.04_amd64.deb-bundle.tar
```

- répertoire de téléchargement : env/sbdr/mysql/mysql\_8.2.0/

- décompression :

```
tar -xvf ../env/sbdr/mysql/mysql_8.2.0/mysql-server_8.2.0-1ubuntu22.04_amd64.deb-bundl
```

```
rm ../env/sbdr/mysql/mysql_8.2.0/mysql-server_8.2.0-1ubuntu22.04_amd64.deb-bundle.tar
```

- liste des paquets téléchargés :

- libmysqlclient22\_8.2.0-1ubuntu22.04\_amd64.deb
- libmysqlclient-dev\_8.2.0-1ubuntu22.04\_amd64.deb
- mysql-client\_8.2.0-1ubuntu22.04\_amd64.deb
- mysql-common\_8.2.0-1ubuntu22.04\_amd64.deb
- mysql-community-client\_8.2.0-1ubuntu22.04\_amd64.deb
- mysql-community-client-core\_8.2.0-1ubuntu22.04\_amd64.deb
- mysql-community-client-plugins\_8.2.0-1ubuntu22.04\_amd64.deb
- mysql-community-server\_8.2.0-1ubuntu22.04\_amd64.deb
- mysql-community-server-core\_8.2.0-1ubuntu22.04\_amd64.deb
- mysql-community-server-debug\_8.2.0-1ubuntu22.04\_amd64.deb
- mysql-community-test\_8.2.0-1ubuntu22.04\_amd64.deb
- mysql-community-test-debug\_8.2.0-1ubuntu22.04\_amd64.deb
- mysql-server\_8.2.0-1ubuntu22.04\_amd64.deb
- mysql-testsuite\_8.2.0-1ubuntu22.04\_amd64.deb

- Extraction des fichiers .deb dans le répertoire de développement :

```

dpkg-deb -x ../env/sbdr/mysql/mysql_8.2.0/mysql-common_8.2.0-1ubuntu22.04_amd64.deb /home/workaholic/env/sbdr/mysql/mysql_8.2.0/
dpkg-deb -x ../env/sbdr/mysql/mysql_8.2.0/mysql-community-client_8.2.0-1ubuntu22.04_amd64.deb /home/workaholic/env/sbdr/mysql/mysql_8.2.0/
dpkg-deb -x ../env/sbdr/mysql/mysql_8.2.0/mysql-community-server_8.2.0-1ubuntu22.04_amd64.deb /home/workaholic/env/sbdr/mysql/mysql_8.2.0/
dpkg-deb -x ../env/sbdr/mysql/mysql_8.2.0/mysql-community-server-core_8.2.0-1ubuntu22.04_amd64.deb /home/workaholic/env/sbdr/mysql/mysql_8.2.0/
dpkg-deb -x ../env/sbdr/mysql/mysql_8.2.0/mysql-community-client-core_8.2.0-1ubuntu22.04_amd64.deb /home/workaholic/env/sbdr/mysql/mysql_8.2.0/

```

- Installation, configuration et exécution du serveur mysqld et du client mysql

- Configuration :

```
cat >> ~/env/sbdr/mysql/mysql_8.2.0/etc/mysql/conf.d/mysql.cnf << EOF
```

```
[mysqld]
```

```
port = 3307
```

```
socket = /home/workaholic/env/sbdr/mysql/mysql_8.2.0/mysqld.sock
```

```
datadir = /home/workaholic/env/sbdr/mysql/mysql_8.2.0/data
```

```
log_error = /home/workaholic/env/sbdr/mysql/mysql_8.2.0/data/mysql-error.log
```

```
EOF
```

```
$HOME/env/sbdr/mysql/mysql_8.2.0/usr/sbin/mysqld \
```

```
--defaults-file=/home/workaholic/env/sbdr/mysql/mysql_8.2.0/etc/mysql/conf.d/mysql.cnf
```

```
--initialize \
```

```
--datadir=/home/workaholic/env/sbdr/mysql/mysql_8.2.0/data
```

- Exécution du serveur en arrière-plan :

```
# On écrit les instructions suivantes à la fin du fichier $HOME/.bashrc
```

```
# mysql
```

```
# Vérifie si mysqld du répertoire de développement est déjà en cours d'exécution
```

```
if ! pgrep -f "$HOME/env/sbdr/mysql/mysql_8.2.0/usr/sbin/mysqld" > /dev/null
```

```
then
```

```
# Démarre mysqld du répertoire de développement si ce n'est pas déjà fait
```

```
$HOME/env/sbdr/mysql/mysql_8.2.0/usr/sbin/mysqld \
```

```
--defaults-file=$HOME/env/sbdr/mysql/mysql_8.2.0/etc/mysql/conf.d/mysql.cnf \
```

```
--daemonize
```

```
fi
```

- Initialisation de la version de MySQL utilisée dans le répertoire de développement :

```
ln -s ~/env/sgbdr/mysql/mysql_8.2.0/usr/sbin/mysqld ~/env/sgbdr/mysql/current-server
ln -s ~/env/sgbdr/mysql/mysql_8.2.0/usr/bin/mysql ~/env/sgbdr/mysql/current-client
```

- Vérification et configuration du mot de passe root :

```
# récupération du mot de passe root provisoire
cat ~/env/sgbdr/mysql/mysql_8.2.0/data/mysql-error.log | grep password
# connection en tant que root via le client du répertoire de développement
../env/sgbdr/mysql/current-client -u root -p -h 127.0.0.1 -P 3307
ALTER USER 'root'@'localhost' IDENTIFIED BY <mot de passe choisi pour root>;
# terminer l'exécution du client mysql
quit;
```

- définir des variables d'environnement relatives à Tomcat

- Pourquoi : pour que le système sache où trouver les fichiers de configuration, les bibliothèques, et autres ressources nécessaires pour exécuter la version de Tomcat que l'on veut utiliser.

- Ajout des lignes suivantes à la fin de ~/.bashrc

```
# developpement directory ~/env/ configuration
# Tomcat
export CATALINA_HOME="/home/workaholic/env/tomcat/current"
```

- explication : ~/.bashrc est exécuté par toute session du shell pour se configurer à son lancement relative à « l'utilisateur de ~ »

- configurer Eclipse pour utiliser jdk-21.0.1 et tomcat-10.1.16

- Window → Preferences → Server → Runtime Environments → Add → Apache → Apache Tomcat v10.1 → Name : Apache Tomcat v10.1.16 → Tomcat installation directory : /home/workaholic/env/tomcat/apache-tomcat-10.1.16 → JRE : jdk-21.0.1 → Apply and Close

- ajoutez bibliothèque Java EE : clic droit sur le projet dans Eclipse → Properties → Java Build Path → Libraries → Add Library → Server Runtime → Next → Apache Tomcat v10.1.16 → Finish →

- configurer IntelliJ pour utiliser la version du JDK courante

- File → Project Structure... → SDKs → Name : current-jdk → JDK home path : /home/workaholic/env/java/-current → Apply → OK

- mettre à jour eclipse-jee-2023-09-R : Help → Check for Updates →...

- configurer Eclipse pour utiliser Apache Maven

- sur les versions d'Eclipse fournies par des packages « Eclipse IDE for Java EE Developers » ou « Eclipse IDE for Java Developers », le plugin de support de Maven est déjà embarqué.

- url de la page de téléchargement : <https://eclipse.dev/m2e/>

- ajouter une installation de Apache Maven : Window → Preferences → Maven → Installations → Add → Installation home : ~/env/maven/current → Installation name : current-maven → Finish → cocher current-maven → User Settings → User Settings : \$user.home/.m2/settings.xml → Update Settings → Apply and Close

# Conception

## Diagramme de cas d'utilisation

Pour s'initier au diagramme de cas d'utilisation, on étudie le livre « Use Case Modeling » de Kurt Bittner et Ian Spence téléchargeable depuis [dokumen.pub/qdownload](http://dokumen.pub/qdownload).

On utilise le logiciel UMLet pour construire le diagramme de cas d'utilisation suivant dans le fichier diagrams/umlet/use\_case\_diagram.uxf que l'on exporte dans le format pdf dans le fichier diagrams/pdf/use\_case\_diagram.pdf.

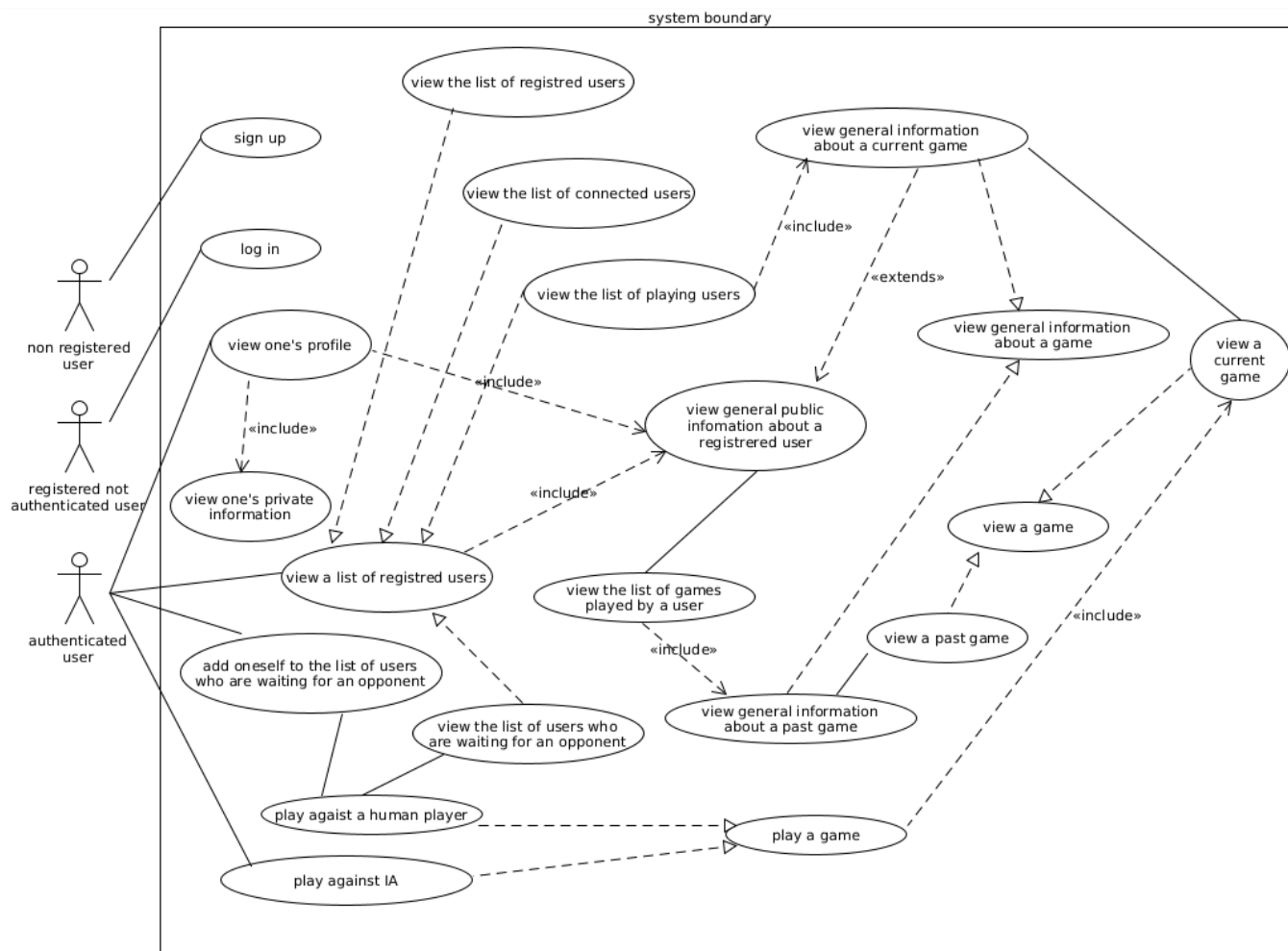


FIGURE 1 – Diagramme de cas d'utilisation construit via UMLet.

Analyse :

TO DO

## Diagramme entités associations

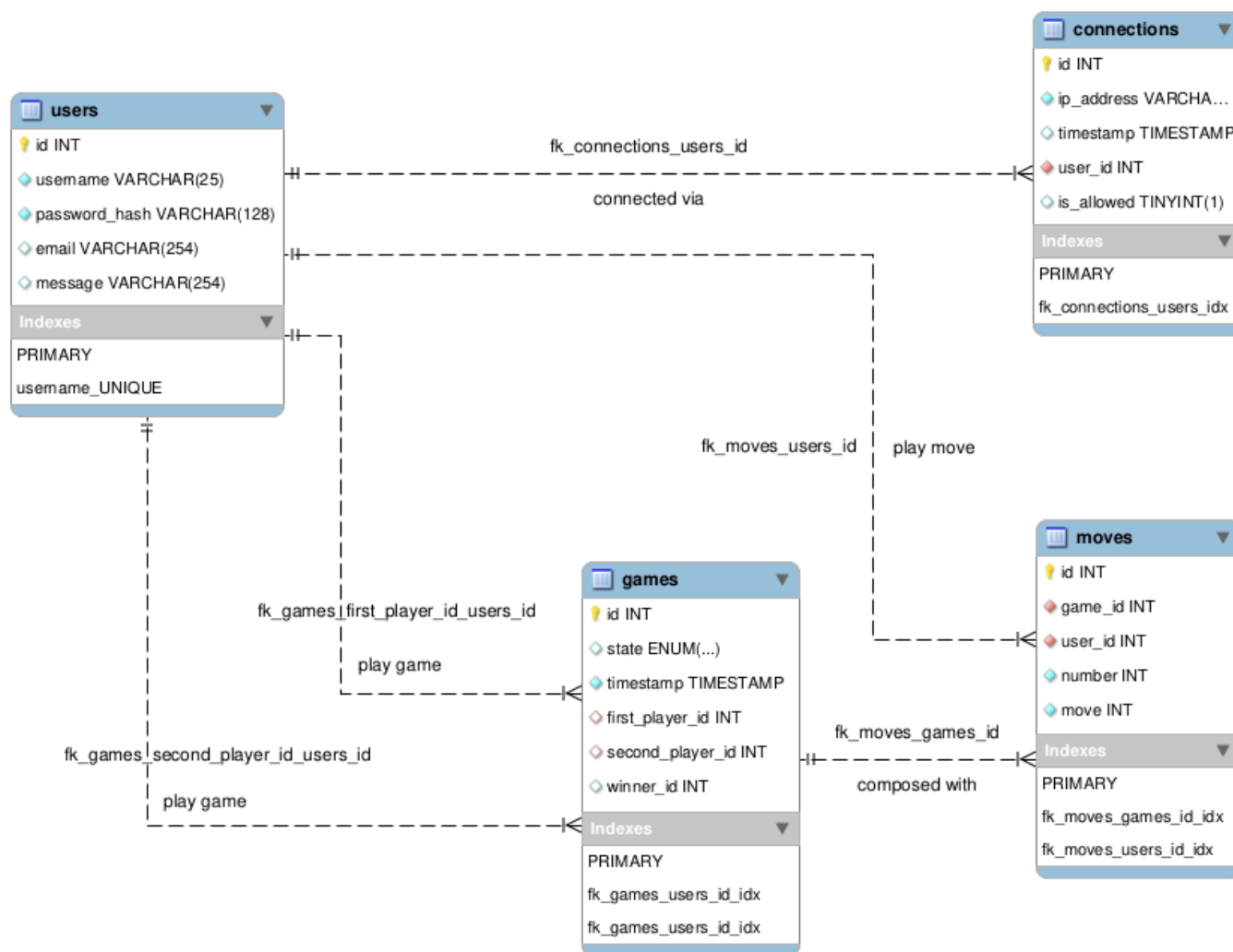


FIGURE 2 – Diagramme entités associations construit via MySQL Workbench.

Analyse :

TO DO

## Diagramme de classes

Pour s'initier au diagramme de classe, on étudie le [Cours « Modélisez vos bases de données »](#) sur [openclassrooms.com](#).

On utilise le logiciel UMLet pour construire le diagramme de classes suivant dans le fichier `diagrams/umlet/class_diagramm.uxf` que l'on exporte dans le format pdf dans le fichier `diagrams/pdf/class_diagramm.pdf`.

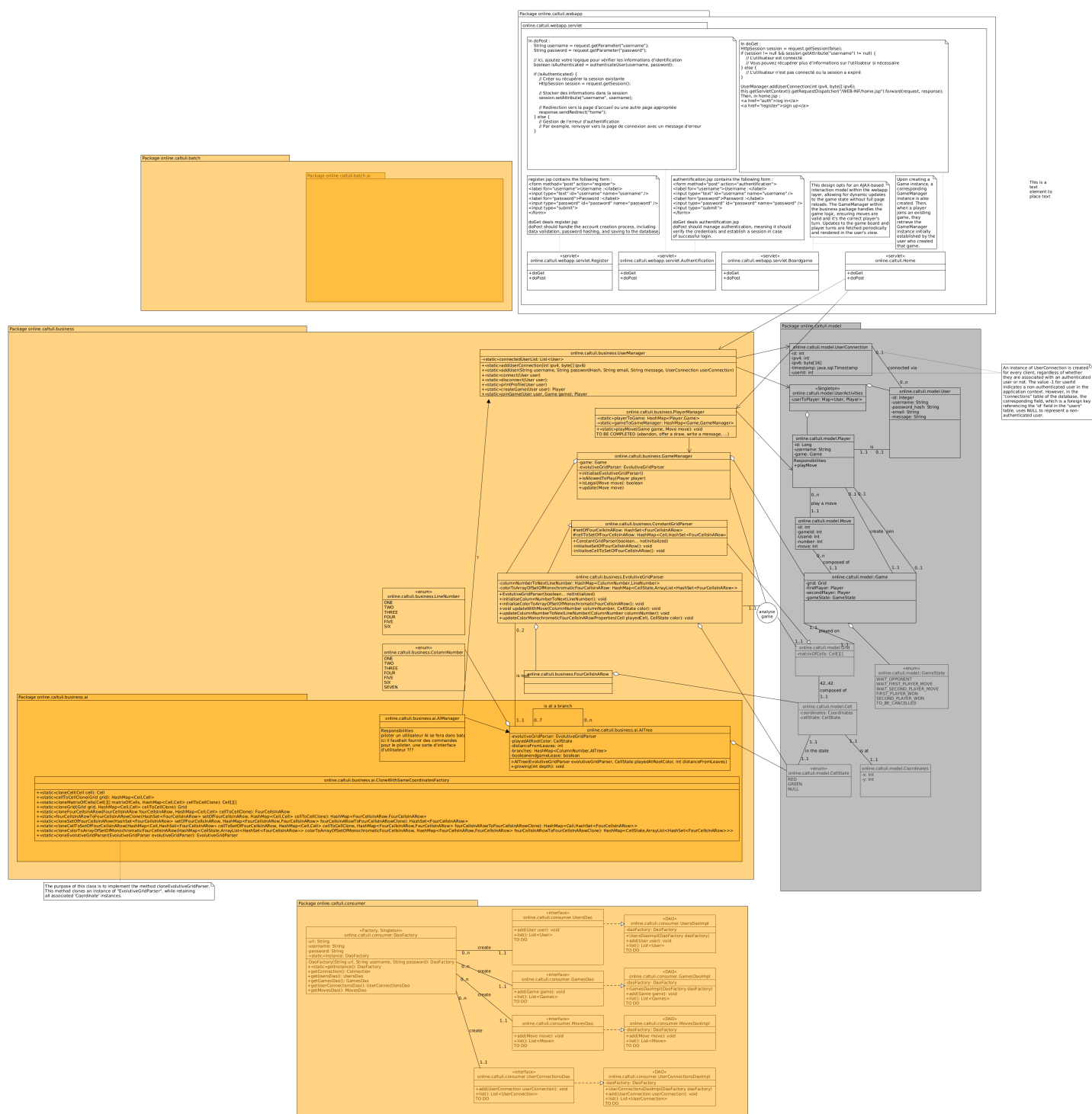


FIGURE 3 – Diagramme de classes construit via UMLet [TO DO : provisoire ; version du 13/01/24].

Analyse :

TO DO

Base de données

## Création de la base de donnée

```
../env/sqbdr/mysql/current-client -h localhost -u root -p -P 3307

CREATE DATABASE onlineplay default character set utf8 collate utf8_general_ci;

USE onlineplay;

CREATE TABLE users (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  username VARCHAR(25) NOT NULL,
  password_hash VARCHAR(128) NOT NULL,
  email VARCHAR(254),
  message VARCHAR(254)
);

CREATE TABLE connections (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  ip_address VARCHAR(45) NOT NULL,
  timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP NOT NULL,
  user_id INT,
  is_allowed TINYINT(1),
  FOREIGN KEY (user_id) REFERENCES users(id)
);

CREATE TABLE games (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  state ENUM('in_progress', 'finished', 'pending', 'cancelled') NOT NULL,
  timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  first_player_id INT,
  second_player_id INT,
  winner_id INT,
  FOREIGN KEY (first_player_id) REFERENCES users(id),
  FOREIGN KEY (second_player_id) REFERENCES users(id)
);

CREATE TABLE moves (
  id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  game_id INT NOT NULL,
  user_id INT NOT NULL,
  number INT NOT NULL,
  move INT NOT NULL,
  FOREIGN KEY (game_id) REFERENCES games(id),
  FOREIGN KEY (user_id) REFERENCES users(id)
);
```



## Initialisation du projet Maven et référencement du premier commit Git

- Configuration de Git

```
git config --global user.name "Caltuli"  
git config --global user.email "lucas.caltuli@gmail.com"
```

- Création d'un projet Maven :

- Initialisation d'un nouveau projet Maven dans \$HOME/env/maven/maven-workspace/ :

```
cd \ $HOME/env/maven/maven-workspace  
mvn archetype:generate \  
-DarchetypeGroupId=org.apache.maven.archetypes \  
-DarchetypeArtifactId=maven-archetype-quickstart \  
-DarchetypeVersion=1.1 \  
-DgroupId=online.caltuli \  
-DartifactId=onlineplay \  
-Dversion=1.0-SNAPSHOT  
rm -r onlineplay/src/
```

- Dans \$HOME/env/maven/maven-workspace/onlineplay/pom.xml, on change le <packaging> en pom

- Ajout des modules :

```
cd \ $HOME/env/maven/maven-workspace/onlineplay  
## module : batch  
mvn -B archetype:generate \  
-DarchetypeGroupId=org.apache.maven.archetypes \  
-DarchetypeArtifactId=maven-archetype-quickstart \  
-DarchetypeVersion=1.1 \  
-DgroupId=online.caltuli \  
-DartifactId=batch \  
-Dpackage=online.caltuli.batch  
  
## module : webapp  
mvn -B archetype:generate \  
-DarchetypeGroupId=org.apache.maven.archetypes \  
-DarchetypeArtifactId=maven-archetype-webapp \  
-DarchetypeVersion=1.4 \  
-DgroupId=online.caltuli \  
-DartifactId=webapp \  
-Dpackage=online.caltuli.webapp  
  
## module : business  
mvn -B archetype:generate \  
-DarchetypeGroupId=org.apache.maven.archetypes \  
-DarchetypeArtifactId=maven-archetype-quickstart \  
-DarchetypeVersion=1.1 \  
-DgroupId=online.caltuli \  
-DartifactId=business \  
-Dpackage=online.caltuli.business  
  
## module : consumer  
mvn -B archetype:generate \  
-DarchetypeGroupId=org.apache.maven.archetypes \  
-DarchetypeArtifactId=maven-archetype-quickstart \  
-DarchetypeVersion=1.1 \  
-DgroupId=online.caltuli \  
-DartifactId=consumer \  
-Dpackage=online.caltuli.consumer  
  
## module : model  
mvn -B archetype:generate \  
-DarchetypeGroupId=org.apache.maven.archetypes \  
-DarchetypeArtifactId=maven-archetype-quickstart \  
-DarchetypeVersion=1.1 \  
-DgroupId=online.caltuli \  
-DartifactId=model \  
-Dpackage=online.caltuli.model
```

- Définition des dépendances entre les modules :

- Dans le POM du projet parent onlineplay/pom.xml, on liste les modules en tant que dépendances pour le projet :

```

<!-- ===== -->
<!-- Dependency Management -->
<!-- ===== -->
<dependencyManagement>
  <dependencies>
    <!-- ===== Modules ===== -->
    <dependency>
      <groupId>online.caltuli</groupId>
      <artifactId>batch</artifactId>
    </dependency>
    <dependency>
      <groupId>online.caltuli</groupId>
      <artifactId>webapp</artifactId>
    </dependency>
    <dependency>
      <groupId>online.caltuli</groupId>
      <artifactId>business</artifactId>
    </dependency>
    <dependency>
      <groupId>online.caltuli</groupId>
      <artifactId>consumer</artifactId>
    </dependency>
    <dependency>
      <groupId>online.caltuli</groupId>
      <artifactId>model</artifactId>
    </dependency>
  </dependencies>
</dependencyManagement>

```

- On ajoute des dépendances aux modules suivant l'architecture multi-tiers :

- Dans onlineplay/batch/pom.xml :

```

<!-- ===== -->
<!-- Dependency Management -->
<!-- ===== -->
<dependencies>
  <!-- ===== Modules ===== -->
  <dependency>
    <groupId>online.caltuli</groupId>
    <artifactId>business</artifactId>
  </dependency>
  <dependency>
    <groupId>online.caltuli</groupId>
    <artifactId>model</artifactId>
  </dependency>
  <!-- ===== Third-party Libraries ===== -->
</dependencies>

```

- Dans onlineplay/webapp/pom.xml :

```

<!-- ===== -->
<!-- Dependency Management -->
<!-- ===== -->
<dependencies>
  <!-- ===== Modules ===== -->
  <dependency>
    <groupId>online.caltuli</groupId>
    <artifactId>business</artifactId>
  </dependency>
  <dependency>
    <groupId>online.caltuli</groupId>
    <artifactId>model</artifactId>
  </dependency>

```

```

    <!-- ===== Third-party Libraries ===== -->
</dependencies>

```

- Dans onlineplay/business/pom.xml :

```

<!-- ===== -->
<!-- Dependency Management -->
<!-- ===== -->
<dependencies>
    <!-- ===== Modules ===== -->
    <dependency>
        <groupId>online.caltuli</groupId>
        <artifactId>consumer</artifactId>
    </dependency>
    <dependency>
        <groupId>online.caltuli</groupId>
        <artifactId>model</artifactId>
    </dependency>
    <!-- ===== Third-party Libraries ===== -->
</dependencies>

```

- Dans onlineplay/consumer/pom.xml :

```

<!-- ===== -->
<!-- Dependency Management -->
<!-- ===== -->
<dependencies>
    <!-- ===== Modules ===== -->
    <dependency>
        <groupId>online.caltuli</groupId>
        <artifactId>model</artifactId>
    </dependency>
    <!-- ===== Third-party Libraries ===== -->
</dependencies>

```

- Création des sous-packages webapp.servlet, business.ai et batch.ai :

```

mkdir ../onlineplay/batch/src/main/java/online/caltuli/batch/ai
mkdir -p ../onlineplay/webapp/src/main/java/online/caltuli/webapp/servlet
mkdir ../onlineplay/business/src/main/java/online/caltuli/business/ai

```

- Construction du projet Maven :

```

mvn package

```

- Initialisation du répertoire Git :

```

cd $HOME/env/maven/maven-workspace/onlineplay
git init
git add .
git commit -m "Initial commit"

```

- Apprentissage de Git via quelques utilisations élémentaires :

- Ajout et commit de répertoires, navigation entre les commits :

```

$ cd $HOME/env/maven/maven-workspace/onlineplay/
$ git add docs/conception/schemas
$ git commit -m "Ajout des schémas de conception dans docs/conception/schemas"
[master b05b44f] Ajout des schémas de conception dans docs/conception/schemas
6 files changed, 2294 insertions(+)
create mode 100644 docs/conception/schemas/erd.mwb
create mode 100644 docs/conception/schemas/erd.mwb.bak
create mode 100644 docs/conception/schemas/erd.uxf
create mode 100644 docs/conception/schemas/ucd.uxf
create mode 100644 docs/conception/schemas/uml.pdf
create mode 100644 docs/conception/schemas/uml.uxf
$ git status
Sur la branche master
rien à valider, la copie de travail est propre
$ git log

```

```
commit b05b44f52e3d202286f633ff9e2c4edf3f9681f6 (HEAD -> master)
Author: Caltuli <lucas.caltuli@gmail.com>
Date: Sat Jan 6 20:27:58 2024 +0100
```

Ajout des schémas de conception dans docs/conception/schemas

```
commit 531ef7a5b4376dc93c8295aff0cdbedf07aff510
Author: Caltuli <lucas.caltuli@gmail.com>
Date: Sat Jan 6 20:06:56 2024 +0100
```

```
Initial commit
$ git checkout 531ef7a5b4376dc93c8295aff0cdbedf07aff510
Note : basculement sur '531ef7a5b4376dc93c8295aff0cdbedf07aff510'.
```

Vous êtes dans l'état « HEAD détachée ». Vous pouvez visiter, faire des modifications expérimentales et les valider. Il vous suffit de faire un autre basculement pour abandonner les commits que vous faites dans cet état sans impacter les autres branches

Si vous voulez créer une nouvelle branche pour conserver les commits que vous créez, il vous suffit d'utiliser l'option -c de la commande switch comme ceci :

```
git switch -c <nom-de-la-nouvelle-branche>
```

Ou annuler cette opération avec :

```
git switch -
```

Désactivez ce conseil en renseignant la variable de configuration `advice.detachedHead` à

```
HEAD est maintenant sur 531ef7a Initial commit
$ ls
batch business consumer model pom.xml webapp
$ git log
commit 531ef7a5b4376dc93c8295aff0cdbedf07aff510 (HEAD)
Author: Caltuli <lucas.caltuli@gmail.com>
Date: Sat Jan 6 20:06:56 2024 +0100
```

```
Initial commit
$ git checkout master
La position précédente de HEAD était sur 531ef7a Initial commit
Basculement sur la branche 'master'
$ ls
batch business consumer docs model pom.xml webapp
$
```

- Gestion de branches et contrôle de version avec Git : Suppression de la classe Player dans le diagramme de classes :

```
git branch player-class-removal 531ef7a5b4376dc93c8295aff0cdbedf07aff510
git checkout player-class-removal
<modification of docs/conception/schemas/class_diagramm.xpdf>
xpdf docs/conception/schemas/class_diagramm.xpdf
<see modifications>
git add docs/conception/schemas/class_diagramm.xpdf # include area in next commit
git commit -m "Player class removal in docs/conception/class_diagramm"
git checkout master
xpdf docs/conception/schemas/class_diagramm.xpdf
<see previous version of docs/conception/schemas/class_diagramm.xpdf>
```

- Mise à jour de la version du plugin JUnit dans la branche master puis fusion des branches master et player-class-removal

```
git checkout master
<JUnit version update in pom.xml>
git add pom.xml
git commit -m "JUnit version update"
git log --all --graph --decorate --pretty=oneline --abbrev-commit # to visualize
# output :
# * d9c1d70 (HEAD -> master) JUnit version update
```

```
# * b05b44f Ajout des schémas de conception dans docs/conception/schemas
# | * 870631b (player-class-removal) Player class removal in docs/conception/# class_diagramm
# | /
# * 531ef7a Initial commit
#
# merge the two branches master and player-class-removal
git checkout master
git merge player-class-removal
git log --all --graph --decorate --pretty=oneline --abbrev-commit # to visualize
# * 9153104 (HEAD, master) Merge branch 'player-class-removal'
# | \
# | * 870631b (player-class-removal) Player class removal in docs/conception/class_diagramm
# * | d9c1d70 JUnit version update
# * | b05b44f Ajout des schémas de conception dans docs/conception/schemas
# | /
# * 531ef7a Initial commit
```

## Téléversement du projet Git dans un dépôt de GitHub

On crée un compte GitHub et un répertoire d'url suivant :

<https://github.com/workaholic-v-0-0-0/onlineplay>

On lie le répertoire local onlineplay au répertoire distant du même nom, on vérifie que cette liaison a bien été établie et on crée un token avec le scope repo et la date d'expiration 2024-06-10 puis on pousse (téléverse) le projet dans le dépôt distant via :

```
cd $HOME/env/maven/maven-workspace/onlineplay
git remote add origin https://github.com/workaholic-v-0-0-0/onlineplay.git
git push origin master
# Username for 'https://github.com': workaholic-v-0-0-0
# Password for 'https://workaholic-v-0-0-0@github.com': ghp_ZRKorXUk4FQUukw1xEp0gMSZtS7Dov1z38Yl
```

La collaboration à distance entre développeurs est désormais possible.

## Développement de la version minimale 1.0-SNAPSHOT

- mise à jour des versions des plugins Maven et des bibliothèques tierce que Maven a automatiquement ajouté au projet en dépendance
- dans <dependencyManagement> du POM parent, on ajoute les références aux plugins Maven :
  - junit:junit:4.11
- dans <dependencyManagement> du POM parent, on ajoute les dépendances aux bibliothèques tierces :
  - jakarta.servlet:jakarta.servlet-api:6.0.0
  - org.glassfish.web:jakarta.servlet.jsp.jstl:3.0.0
  - jakarta.servlet.jsp.jstl:jakarta.servlet.jsp.jstl-api:3.0.0
  - com.mysql:mysql-connector-j:8.2.0

Remarque : org.glassfish.web:jakarta.servlet.jsp.jstl:3.0.0 est une implémentation de jakarta.servlet.jsp.jstl:jakarta.servlet.jsp.jstl-api:3.0.0.

- dans webapp/pom.xml, on ajoute une dépendance vers jakarta.servlet:jakarta.servlet-api:6.0.0
- créer webapp/src/main/java/online/caltuli/webapp/servlet/home.java
- implémenter doGet de Home :

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException {
    this.getServletContext().getRequestDispatcher("/WEB-INF/home.jsp").forward(request, response);
}
```

- créer webapp/src/main/webapp/WEB-INF/home.jsp
- implémenter webapp/src/main/webapp/WEB-INF/home.jsp :

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>home</title>
</head>

<body>
    <p>
        Minimal version of the web application onlineplay.
    </p>
</body>
</html>
```

- créer webapp/src/main/webapp/WEB-INF/web.xml
- mappage du servlet Home au motif d'url /home dans webapp/src/main/webapp/WEB-INF/web.xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
        https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
    version="6.0">

    <servlet>
        <servlet-name>Home</servlet-name>
        <servlet-class>online.caltuli.webapp.servlet.Home</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Home</servlet-name>
        <url-pattern>/home</url-pattern>
    </servlet-mapping>

</web-app>
```

- créer le fichier
- implémenter la classe User dans :



- créer le fichier `business/src/main/java/online/caltuli/business/UserManager.java`
- implémenter la classe `UserManager` dans `business/src/main/java/online/caltuli/business/UserManager.java` :
- bbb
- Comme expliqué dans la section annexe « [Exemple de déploiement manuel d'une l'application web sur un serveur Tomcat](#) », on peut vérifier le fonctionnement via :
  - L'exécution du serveur Tomcat configuré pour déployer l'application web :

```
cp \
~/env/maven/maven-workspace/onlineplay/webapp/target/webapp.war \
~/env/tomcat/apache-tomcat-10.1.16/webapps/
~/env/tomcat/current/bin/startup.sh
```
  - La soumission de l'url <http://localhost:8080/webapp/home> dans un navigateur.

## VRAC

- `AppContextListener` (Écouteur de contexte) : pour exécuter du code au démarrage de l'application pour par exemple créer une unique instance de `Model` et l'initialiser en récupérant des données dans la base de données.
- `Git` avec notamment sa fonctionnalité de fusionner des branches, le service d'hébergement de code source `GitHub` intégrant notamment `Git` pour suivre l'évolution des fichiers et le respect de la séparation des préoccupations vont permettre aux membres de l'équipe de collaborer en travaillant chacun sur une partie distincte du projet.

Le répertoire `online/docs` ne devra pas être pris en compte lors de la fusion sur le dépôt distant partagé par les collaborateurs car il est destiné à contenir le travail de documentation individuel de chaque membre de l'équipe concernant leur propre contribution au projet.



# ANNEXES

## Sommaire des Annexes

Annexe **A** : [Liens étudiés](#)

Annexe **B** : Étude du cours « Développez des sites web avec Java EE » de Mathieu Nebra sur [openclassrooms.com](#) pour développer une application web proposant les services de lecture et d'écriture dans une table d'une base de données MySQL tout en respectant MVC et DAO

Annexe **C** : Étude du cours « Organiser et packager une application Java avec Apache Maven » de Loïc Guibert sur [openclassrooms.com](#)

Annexe **D** : Étude du cours « Modélisez vos bases de données » de Nicolas Rangeon sur [openclassrooms.com](#) pour concevoir la structure de bases de données relationnelles destinées aux données (structurées) de l'application « Puissance 4 »

Annexe **E** : [Arborescence d'un projet Maven](#)

Annexe **F** : [Un exemple minimal de serveur Back-end derrière Apache 2 avec la bibliothèque Python websocket-server](#)

Annexe **G** : [Exemple de conversion manuelle d'un projet Jakarta EE existant en un projet Maven](#)

Annexe **H** : [Exemple de déploiement manuel d'une l'application web sur un serveur Tomcat](#)

Annexe **I** : Première idée d'architecture pour un autre projet de portfolio interactif rendu possible par le projet en cours

Annexe **J** : Initiation à **Git**

Annexe **K** : [Sources](#)

Annexe **L** : [Fichiers de sortie](#)

## A Liens étudiés

- [illegible]

## B Étude du cours « Développez des sites web avec Java EE » de Mathieu Nebra sur openclassrooms.com pour développer une application web proposant les services de lecture et d'écriture dans une table d'une base de données MySQL tout en respectant MVC et DAO

lien du cours étudié : [cours « Développez des sites web avec Java EE »](#) sur [openclassrooms.com](#)

### B.0.1 Création d'une application web

file → new → dynamic web project → Project name : test\_DAO

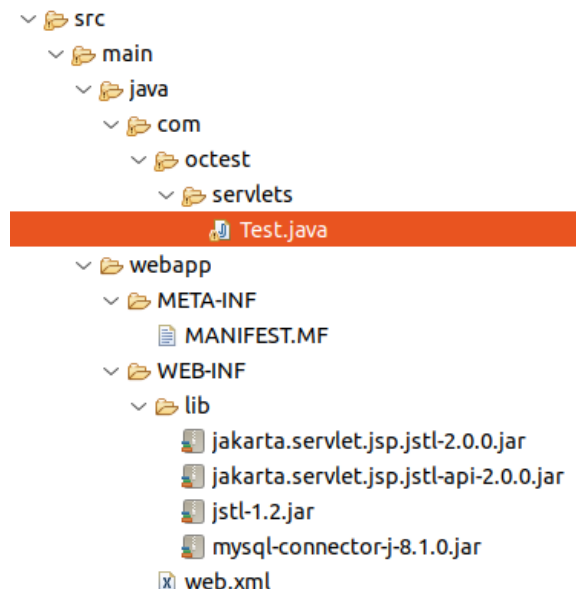
### B.0.2 Gestion des dépendances tierces

bibliothèque : les copier dans src/main/webapp/WEB-INF/lib/ puis build path → configure build path → Classpath → Add JARs (voir figure suivante)

### B.0.3 Ajout d'un servlet

ajouter une servlet

- Java Resources → new → servlet → Java package : com.oc<nom de domaine de niveau 2>.servlets → Class name : <nom du servlet>



- configuration : WEB-INF → New → Other → filter : xml → XML File → Next → File name : web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="https://jakarta.ee/xml/ns/jakartaee"
  xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
  version="6.0">

  <!-- déclaration du servlet relatif à la classe Test du package com.octest.servlets.Test ;
  ce bloc de configuration indique au serveur web ou à l'application de créer une instance de la classe
  com.octest.servlets.Test lorsqu'il démarre, et de la gérer en tant que servlet sous le nom "Test".
  Cela permet au servlet d'être accessible et de répondre aux requêtes qui lui sont adressées en fonction
  de la configuration supplémentaire spécifiée dans le web.xml, comme les motifs d'URL (URL patterns) qui
  déterminent quelles requêtes HTTP le servlet doit traiter.
  -->
  <servlet>
    <servlet-name>Test</servlet-name> <!-- Donne un nom unique à la servlet pour la référencer ailleurs dans le web.xml-->
    <servlet-class>com.octest.servlets.Test</servlet-class> <!-- spécifie le nom complet de la classe du servlet, y compris le package -->
  </servlet>

  <!-- avec le bloc précédent, on indique au serveur d'application que les requêtes vers l'URL spécifique "/guidb" doivent être gérées par la
  classe de servlet com.octest.servlets.Test. C'est la manière dont le serveur web sait quelle classe invoquer pour traiter la requête entrante
  correspondant au motif d'URL donné.
  -->
  <servlet-mapping>
    <servlet-name>Test</servlet-name>
    <url-pattern>/guidb</url-pattern>
  </servlet-mapping>
</web-app>
```

### B.0.4 Création d'une vue

créer une vue : WEB-INF → New → JSP File → File name : guidb.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Test_DAO</title>
</head>
<body>
<p>
Pense-bête pour commencer un dynamic web project avec GUI pour écrire et lire dans une table d'une base
de données MySQL tout en respectant MVC et DAO
</p>
</body>
</html>
```

### B.0.5 Implémentation de la méthode doGet du servlet pour associer la vue à au servlet pour la méthode GET

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
    this.getRequestDispatcher("/WEB-INF/gui/db.jsp").forward(request, response);
}

/*
 * L'objet ServletContext récupéré par la méthode getServletContext() héritée de la classe HttpServlet est destiné
 * à contenir des informations relatives à l'exécution de l'application web. Cependant, il est important de noter que
 * ServletContext existe en une seule instance par application web (par contexte d'application) et est partagé par toutes
 * les servlets au sein de cette application.
 *
 * ServletContext donne à la servlet la capacité de communiquer avec son "environnement extérieur" en lui fournissant
 * un moyen d'accéder à des informations globales (comme des paramètres de configuration) et des ressources (comme des
 * objets stockés dans le contexte), et non pas seulement celles dont elle a spécifiquement besoin de connaître. Cela
 * signifie que ServletContext permet à la servlet d'interagir avec un large éventail de données qui concernent l'ensemble
 * de l'application, pas seulement celles propres à une instance de servlet.
 *
 * En résumé, ServletContext est un objet qui représente l'environnement de l'application web au sein du serveur (comme
 * Tomcat) et fournit un moyen de partager des informations entre toutes les parties de l'application, garantissant ainsi
 * un accès cohérent et centralisé aux données importantes de l'application.
 *
 * getServletContext("/WEB-INF/gui/db.jsp") récupère une instance de classe RequestDispatcher associée à la vue
 * /WEB-INF/gui/db.jsp
 *
 * (NOTION DE CONTENEUR)
 * L'objet RequestDispatcher agit comme un wrapper ou un conteneur pour la ressource que l'on souhaite atteindre.
 * Lorsqu'on appelle la méthode forward(request, response) sur cet objet, il utilise les objets HttpServletRequest
 * et HttpServletResponse que l'on lui passe pour déléguer la requête et la réponse à la ressource JSP spécifiée.
 * La JSP peut alors générer du contenu dynamique qui sera renvoyé au client, comme si ce contenu avait été généré
 * par le servlet lui-même.
 *
 * (FLUX DE CONTRÔLE)
 * Il est important de noter que, pendant l'opération de transfert (forward), la requête et la réponse ne sont pas
 * réinitialisées ou modifiées par le conteneur de servlet avant d'être passées à la ressource cible. Ainsi, toutes
 * les données ou objets qui ont été ajoutés à la requête restent disponibles pour la ressource JSP.
 *
 * (RESPECT DU PATRON DE CONCEPTION MVC)
 * objet RequestDispatcher obtenu par getServletContext("/WEB-INF/gui/db.jsp") est un mécanisme qui permet à la
 * servlet de déléguer la réponse à une requête à une page JSP spécifique pour le rendu de la vue, tout en gardant la
 * logique de traitement de la requête au sein du servlet.
 */
```

### B.0.6 Installation de la JSTL (Java server page Standard Tag Library)

- Copier jstl-1.2.jar, jakarta.servlet.jsp.jstl-2.0.0.jar et jakarta.servlet.jsp.jstl-api-2.0.0.jar puis « Add Jars » (déjà fait ; voir plus haut)
- Un autre .jsp nommé taglibs.jsp destiné à être inclus dans les autres .jsp pour interpréter les balises JSTL (et pour l'encodage)

```
<%@ page pageEncoding="UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

- configuration dans web.xml pour inclure taglibs.jsp en prélude dans tous les fichiers .jsp

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="https://jakarta.ee/xml/ns/jakartaee"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee https://jakarta.ee/xml/ns/jakartaee
    version="6.0">

    <!-- VOIR PRÉCÉDEMMENT POUR LE DÉBUT DE L'IMPLÉMENTATION -->

    <!-- inclusion de taglibs.jsp en prélude de tous les fichiers .jsp
    -->
    <jsp-config>
        <jsp-property-group>
            <url-pattern>*.jsp</url-pattern>
            <include-prelude>/WEB-INF/taglibs.jsp</include-prelude>
        </jsp-property-group>
    </jsp-config>

</web-app>
```



### B.0.7 Création d'une base de données "jakartae" et une table "noms" dans cette base via le serveur MySQL

```
mysql -h localhost -u root -p
CREATE DATABASE jakartae default character set utf8 collate utf8_general_ci;
USE jakartae;
CREATE TABLE noms (
  id INT( 11 ) NOT NULL AUTO_INCREMENT ,
  nom VARCHAR( 200 ) NOT NULL ,
  prenom VARCHAR( 200 ) NOT NULL ,
  PRIMARY KEY ( id )
) ENGINE = INNODB;
INSERT INTO noms(nom, prenom) VALUES("Dupont", "Jean");
INSERT INTO noms(nom, prenom) VALUES("La Police", "Léo");
QUIT;
```

### B.0.8 Installation de JDBC (Java Data Base Connectivity)

Copier mysql-connector-j-8.1.0.jar puis « Add Jars » (déjà fait ; voir plus haut)

### B.0.9 Implémentation respectant le patron de conception DAO (Data Access Objet)

- **Implémentation de la classe DaoFactory** servant d'intermédiaire entre le modèle et la base de données ; cette classe ne peut générer qu'une seule instance (ie son implémentation respecte le pattern singleton) ; cette unique instance possède une méthode getConnection() chargée d'établir une connexion avec la base de données et de retourner une instance de classe Connection gérant les données relatives à cette connexion ; cette unique instance de DaoFactory possède aussi des méthodes retournant des instances d'objets TrucDaoImpl implémentant les interfaces correspondant chacune à une couche de services chargée d'interagir avec l'une des tables de la base de données (qui correspond à une entité) ; par exemple, getUtilisateurDao() retourne une instance de la classe UtilisateurDaoImpl implémentant l'interface UtilisateurDao, cette dernière instance connaissant l'unique instance de DaoFactory, elle peut établir des connexions avec la base de données et faire notamment des opérations sur la table "noms" et ses méthodes fournissent une couche de services pour interagir avec cette table ; elle peut aussi créer éventuellement d'autres instances fournissant de façon analogue des couches de services pour interagir avec d'autres tables de la base de données.

```
package com.octest.dao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import com.octest.dao.interfaces.UtilisateurDao;
import com.octest.dao.impl.UtilisateurDaoImpl;

public class DaoFactory {
  private String url;
  private String username;
  private String password;

  // Instance statique privée (utile pour respecter le pattern Singleton)
  private static DaoFactory instance;

  private DaoFactory(String url, String username, String password) {
    this.url = url;
    this.username = username;
    this.password = password;
  }

  public static DaoFactory getInstance() {
    if (instance == null) { // pour respecter le pattern Singleton
      try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        instance = new DaoFactory(
          "jdbc:mysql://localhost:3306/jakartae",
          "root",
          "<root's password> ");
      } catch (ClassNotFoundException e) {
      }
    }
    return instance;
  }

  public Connection getConnection() throws SQLException {
    Connection connexion = DriverManager.getConnection(url, username, password);
    connexion.setAutoCommit(false);
    return connexion;
  }

  // récupération du Dao
  public UtilisateurDao getUtilisateurDao() {
    return new UtilisateurDaoImpl(this);
  }
}
```

- **Implémentation de l'interface UtilisateurDao** spécifiant la couche de services nécessaire pour interagir avec la table "noms" stockant et organisant les données relatives aux utilisateurs.

```
package com.octest.dao.interfaces;

import java.util.List;
import com.octest.beans.Utilisateur;
import com.octest.dao.DaoException;

public interface UtilisateurDao {
    void ajouter(Utilisateur utilisateur) throws DaoException;
    List<Utilisateur> lister() throws DaoException;
}
```

- **Implémentation de la classe UtilisateurDaoImpl** dont les instances fournissent une couche de services nécessaire pour interagir avec la table "noms" stockant et organisant les données relatives aux utilisateurs.

```
package com.octest.dao.impl;

import java.sql.*;
import java.util.*;
import com.octest.dao.DaoFactory;
import com.octest.dao.interfaces.UtilisateurDao;
import com.octest.beans.BeanException;
import com.octest.dao.DaoException;
import com.octest.beans.Utilisateur;

public class UtilisateurDaoImpl implements UtilisateurDao {
    private DaoFactory daoFactory;

    public UtilisateurDaoImpl(DaoFactory daoFactory) {
        this.daoFactory = daoFactory;
    }

    @Override
    public void ajouter(Utilisateur utilisateur) throws DaoException {
        Connection connexion = null;
        PreparedStatement preparedStatement = null;
        try {
            connexion = daoFactory.getConnection();
            preparedStatement = connexion.prepareStatement("INSERT INTO noms(nom, prenom) VALUES(?, ?);");
            preparedStatement.setString(1, utilisateur.getNom());
            preparedStatement.setString(2, utilisateur.getPrenom());
            preparedStatement.executeUpdate();
            connexion.commit();
        } catch (SQLException e) {
            try {
                if (connexion != null) {
                    connexion.rollback(); // en fait ici autoCommit à false est inutile car il n'y a qu'un appel à executeUpdate()
                }
            } catch (SQLException e2) {
            }
            throw new DaoException("Impossible de communiquer avec la base de données.");
        } finally {
            try {
                if (connexion != null) {
                    connexion.close();
                }
            } catch (SQLException e) {
                throw new DaoException("Impossible de communiquer avec la base de données.");
            }
        }
    }

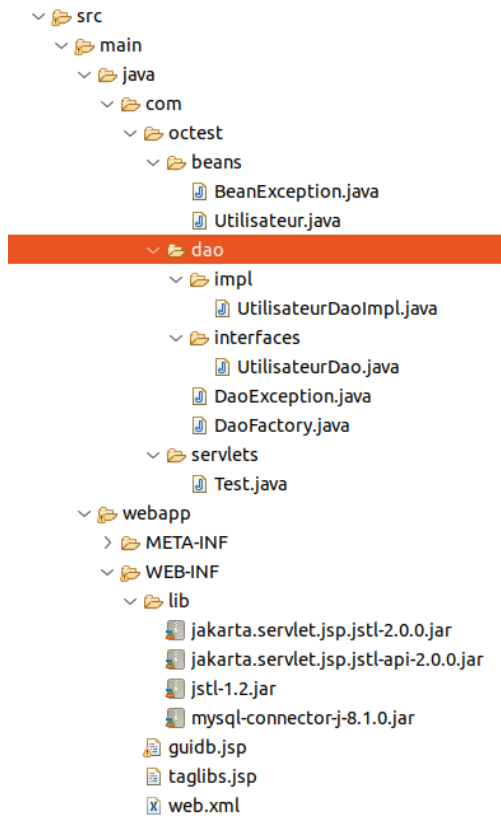
    @Override
    public List<Utilisateur> lister() throws DaoException {
        List<Utilisateur> utilisateurs = new ArrayList<Utilisateur>();
        Connection connexion = null;
        Statement statement = null;
        ResultSet resultat = null;
        try {
            connexion = daoFactory.getConnection();
            statement = connexion.createStatement();
            resultat = statement.executeQuery("SELECT nom, prenom FROM noms;");
            while (resultat.next()) {
                utilisateurs.add(new Utilisateur(resultat.getString("nom"), resultat.getString("prenom")));
            }
        } catch (SQLException e) {
            throw new DaoException("Impossible de communiquer avec la base de données.");
        } catch (BeanException e) {
            throw new DaoException("Les données de la base sont invalides.");
        } finally {
            try {
                if (connexion != null) {
                    connexion.close();
                }
            } catch (SQLException e) {
                throw new DaoException("Impossible de communiquer avec la base de données.");
            }
        }
        return utilisateurs;
    }
}
```

```

}
}

```

- packages relatifs au DAO :



- Implémentation des méthodes du servlet pour lire et écrire dans la table via une couche de services fournie par DaoFactory et UtilisateurDaoImpl; le servlet écrit dans la table les données récupérées via la méthode POST dans des variables d'identifiants "nom" et "prenom" et il transmet à la vue une liste "utilisateurs" d'instances de classe Utilisateur stockant et organisant les données lues dans la table :

```

package com.octest.servlets;

import jakarta.servlet.ServletException;

import jakarta.servlet.annotation.WebServlet;
import jakarta.servlet.http.HttpServlet;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import java.io.IOException;

import com.octest.beans.Utilisateur;
import com.octest.dao.DaoException;
import com.octest.dao.DaoFactory;
import com.octest.dao.interfaces.UtilisateurDao;

@WebServlet("/Test_DAO")
public class Test extends HttpServlet {

    private static final long serialVersionUID = 1L;
    private UtilisateurDao utilisateurDao;

    public void init() throws ServletException {
        // génération d'une couche de services pour interagir avec la table noms suivant DAO
        DaoFactory daoFactory = DaoFactory.getInstance();
        this.utilisateurDao = daoFactory.getUtilisateurDao();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // transmission des données à la vue
        try {
            request.setAttribute("utilisateurs", this.utilisateurDao.lister());
        } catch (DaoException e) {
            request.setAttribute("erreur", e.getMessage());
        }

        this.getServletContext().getRequestDispatcher("/WEB-INF/guidb.jsp").forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

```

```

// enregistrement des informations soumises via la méthode POST suivant DAO
try {
    utilisateurDao.ajouter(new Utilisateur(request.getParameter("nom"), request.getParameter("prenom")));
} catch (Exception e) {
    request.setAttribute("erreur", e.getMessage());
}

// transmission des données à la vue
try {
    request.setAttribute("utilisateurs", utilisateurDao.lister());
} catch (DaoException e) {
    request.setAttribute("erreur", e.getMessage());
}

this.getServletContext().getRequestDispatcher("/WEB-INF/guidb.jsp").forward(request, response);
}
}

```

### B.0.10 Implémentation de la vue guidb.jsp

- pour afficher les données de la table envoyées par le servlet via la liste "utilisateurs" d'instances de classe Utilisateur stockant et organisant les données lues dans la table;
- pour envoyer au servlet des données soumises par le client via un formulaire avec la méthode POST dans les paramètres "nom" et "prenom" afin que ce-dernier enregistre dans la table la nouvelle entrée correspondante via une couche de services fournie par DaoFactory et UtilisateurDaoImpl

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Test_DAO</title>
</head>
<body>
<p>
    Pense-bête pour commencer un dynamic web project avec GUI pour écrire et lire dans une table d'une base
    de données MySQL tout en respectant MVC et DAO
</p>
<c:choose>
<c:when test="${ !empty erreur }">
    <p style="color: red">
        Erreur : <c:out value="${ erreur }" />
    </p>
</c:when>
<c:otherwise>
    <p>
        Lecture de la table :
    </p>
    <ul>
        <c:forEach var="utilisateur" items="${ utilisateurs }">
            <li>
                <c:out value="${ utilisateur.getPrenom() } " />
                <c:out value="${ utilisateur.getNom() } " />
            </li>
        </c:forEach>
    </ul>
    <p>
        Ajouter une entrée dans la table :
    </p>
    <form method="post" action="guidb">
        <label for="nom">Nom :</label>
        <input type="text" id="nom" name="nom" />
        <label for="prenom">Prénom :</label>
        <input type="text" id="prenom" name="prenom" />
        <input type="submit">
    </form>
    </c:otherwise>
</c:choose>
</body>
</html>

```

### B.0.11 Gestion des erreurs

(déjà implémenté)

- com.octest.beans.BeanException.java

```

package com.octest.beans;

public class BeanException extends Exception {
    private static final long serialVersionUID = 1L;

    public BeanException(String message) {
        super(message);
    }
}

```

- com.octest.dao.DaoException

```
package com.octest.dao;

public class DaoException extends Exception {
    private static final long serialVersionUID = 1L;

    public DaoException(String message) {
        super(message);
    }
}
```

### B.0.12 Gestion des transactions

(déjà fait)

« connexion.setAutoCommit(**false**); », « connexion.commit(); », « connexion.rollback(); »

## C Étude du cours « Organiser et packager une application Java avec Apache Maven » de Loïc Guibert sur openclassrooms.com

lien du cours étudié : [cours « Organisez et packagez une application Java avec Apache Maven »](#) sur [openclassrooms.com](#)

### C.1 Étapes

#### C.1.1 Création d'un répertoire env/ pour l'environnement de développement

comme expliqué dans la section « [Construction de l'environnement de développement](#) » :

```
$ tree -a -L 2 env
env
├── current-workspace -> maven/maven-workspace
├── eclipse
│   ├── current -> eclipse-jee-2023-09-R
│   ├── eclipse-jee-2023-09-R
│   └── eclipse-workspace
├── env_config
│   ├── bashrc
│   ├── bashrc~
│   └── env_config.txt
├── intellij
│   ├── current -> /home/workaholic/env/intellij/idea-IC-232.10227.8
│   ├── idea-IC-232.10227.8
│   ├── idea-IU-232.10227.8
│   └── intellij-workspace
├── java
│   ├── current -> jdk-21.0.1
│   └── jdk-21.0.1
├── maven
│   ├── apache-maven-3.9.5
│   ├── current -> apache-maven-3.9.5
│   └── maven-workspace
└── tomcat
    ├── apache-tomcat-10.1.16
    └── current -> apache-tomcat-10.1.16
```

21 directories, 3 files

#### C.1.2 Création d'un projet maven respectant la version 1.1 de l'archétype maven-archetype-quickstart

- de groupId org.exemple.demo (l'organisation qui porte le projet)
- de artifactId mon-appli (le nom du projet)
- de version 1.0-SNAPSHOT (proposée par défaut)

Remarque : Par convention Maven, le suffixe SNAPSHOT indique que le code de la version n'est pas figé. Par opposition, si la version ne se termine pas par le suffixe -SNAPSHOT, cela signifie qu'il s'agit d'une release de l'application et que son code est figé. Les versions snapshot font l'objet de traitements particuliers, notamment dans la gestion des dépendances. Voir plus loin. Pendant le développement de l'application, on doit avoir le suffixe -SNAPSHOT pour en faire une version snapshot et une fois la version terminée, prête à être déployée, on doit enlever ce suffixe pour en faire une version release.

- de package principal org.exemple.demo (le groupId est proposé par défaut)

```
cd maven/maven-workspace/
mvn archetype:generate -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.1
```

```
Define value for property 'groupId': org.exemple.demo
Define value for property 'artifactId': mon-appli
Define value for property 'version' 1.0-SNAPSHOT: :
Define value for property 'package' org.exemple.demo: :
Confirm properties configuration:
groupId: org.exemple.demo
artifactId: mon-appli
```

version: 1.0-SNAPSHOT

package: org.exemple.demo

```
$ tree maven-workspace
```

```
maven-workspace
├── mon-appli
│   ├── pom.xml
│   └── src
│       ├── main
│       │   ├── java
│       │   │   ├── org
│       │   │   │   ├── exemple
│       │   │   │   │   ├── demo
│       │   │   │   │   │   App.java
│       │   └── test
│       │       ├── java
│       │       │   ├── org
│       │       │   │   ├── exemple
│       │       │   │   │   ├── demo
│       │       │   │   │   │   AppTest.java
```

12 directories, 3 files

Sous env/maven/maven-workspace/mon-appli :

- pom.xml : Project Object Model; fichier principal du projet maven, contient toute la configuration du projet maven
- un répertoire src : contient toutes les sources de l'application mon-appli

### C.1.3 Compilation et packaging de l'application

```
cd ../env/maven/maven-workspace/mon-appli
mvn package
```

Plus précisément :

- un fichier .jar a été généré dans le répertoire target
- les ressources ont été filtrées
- une compilation a été lancée
- des ressources de test ont été copiées
- les classes de test ont été compilées
- les tests unitaires ont été exécutés

```
$ tree ../env/maven/maven-workspace/
../env/maven/maven-workspace/
```

```
├── mon-appli
│   ├── pom.xml
│   └── src
│       ├── main
│       │   ├── java
│       │   │   ├── org
│       │   │   │   ├── exemple
│       │   │   │   │   ├── demo
│       │   │   │   │   │   App.java
│       │   └── test
│       │       ├── java
│       │       │   ├── org
│       │       │   │   ├── exemple
│       │       │   │   │   ├── demo
│       │       │   │   │   │   AppTest.java
│       └── target
│           ├── classes
│           │   ├── org
│           │   │   ├── exemple
│           │   │   │   ├── demo
│           │   │   │   │   App.class
│           ├── generated-sources
│           │   ├── annotations
│           ├── generated-test-sources
│           │   ├── test-annotations
│           └── maven-archiver
```

```

├── pom.properties
├── maven-status
│   ├── maven-compiler-plugin
│   │   ├── compile
│   │   │   ├── default-compile
│   │   │   │   ├── createdFiles.lst
│   │   │   │   └── inputFiles.lst
│   │   └── testCompile
│   │       ├── default-testCompile
│   │       │   ├── createdFiles.lst
│   │       │   └── inputFiles.lst
│   └── mon-appli-1.0-SNAPSHOT.jar
├── surefire-reports
│   ├── org.exemple.demo.AppTest.txt
│   └── TEST-org.exemple.demo.AppTest.xml
├── test-classes
│   └── org
│       └── exemple
│           └── demo
│               └── AppTest.class

```

33 directories, 13 files

#### C.1.4 Exécution de l'application

```
# cd ../env/maven/maven-workspace/mon-appli
java -cp target/mon-appli-1.0-SNAPSHOT.jar org.exemple.demo.App
```

Hello World!

#### C.1.5 Importation du projet dans Eclipse

Se fait sans difficulté via le GUI d'Eclipse.

#### C.1.6 Modification de env/maven/maven-workspace/mon-appli/pom.xml pour définir les informations générales et les propriétés du projet

- On complète la section concernant les informations du projet

```

<!-- ===== Informations générales ===== -->
<name>mon-appli</name>
<description>
  La super application qui sert à faire ceci/cela...
</description>
<url>http://maven.apache.org</url>

<!-- ===== Organisation ===== -->
<organization>
  <name>Mon entreprise</name>
  <url>http://www.exemple.org</url>
</organization>

<!-- ===== Licenses ===== -->
<licenses>
  <license>
    <name>Apache License, Version 2.0</name>
    <url>https://www.apache.org/licenses/LICENSE-2.0.txt</url>
  </license>
</licenses>

```

Remarque : Un projet Maven est identifié par ses coordonnées : groupId:artifactId:version

- On complète la section concernant les propriétés
  - Les propriétés sont des sortes de constantes ; elles seront remplacées par leur valeur lors de l'exécution de Maven.
  - Par exemple, on veut utiliser dans le projet un framework composé de plusieurs bibliothèques comme Apache Struts ; on ajoute alors toutes les dépendances vers les bibliothèques de Apache Struts que l'on veut utiliser puis on peut définir et utiliser la propriété apache.struts.version comme suit :

```

<!-- ===== -->
<!-- Propriétés -->
<!-- ===== -->

```



```

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <apache.struts.version>2.5.10.1</apache.struts.version>
</properties>

<dependencies>
  <!-- ===== Apache Struts ===== -->
  <dependency>
    <groupId>org.apache.struts</groupId>
    <artifactId>struts2-core</artifactId>
    <version>${apache.struts.version}</version>
  </dependency>
  <dependency>
    <groupId>org.apache.struts</groupId>
    <artifactId>struts2-json-plugin</artifactId>
    <version>${apache.struts.version}</version>
  </dependency>

```

**C.1.7 Modification de env/maven/maven-workspace/mon-appli/pom.xml relatives au processus de construction du projet par Maven, le build**

**C.1.7.1 Premier exemple : Personnaliser le répertoire de sortie** Par défaut, il s'agit du répertoire target. On peut alors le changer en le répertoire output comme suit :

```

<!-- ===== -->
<!-- Build -->
<!-- ===== -->
<build>
  <directory>${project.basedir}/output</directory>
</build>

```

**Deuxième exemple : Définir la classe Main dans le JAR**

- Pour qu'un .jar soit exécutable, il faut définir quelle est la classe Main à exécuter dans mon-appli-classes/META-INF/MANIFEST.MF.
- Pour ce faire, on utilise le plugin Maven Jar

```

<!-- ===== -->
<!-- Build -->
<!-- ===== -->
<build>
  <!-- Gestion des plugins (version) -->
  <pluginManagement>
    <plugins>
      <!-- Plugin responsable de la génération du fichier JAR -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>3.0.2</version>
      </plugin>
    </plugins>
  </pluginManagement>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <configuration>
        <archive>
          <!-- Création du Manifest pour la définition de la classe Main -->
          <manifest>
            <mainClass>org.exemple.demo.App</mainClass>
          </manifest>
        </archive>
      </configuration>
    </plugin>
  </plugins>

```

```

</plugin>
</plugins>
</build>

```

### C.1.8 Lancement de la construction Maven

```

cd ~/env/maven/maven-workspace/mon-appli/
mvn package

```

Remarque : On peut aussi lancer la construction Maven via le GUI d'Eclipse : clic droit sur le projet → Run As → Maven build → Goals : package → Run

### C.1.9 Exécution du JAR mon-appli/target/mon-appli-1.0-SNAPSHOT.jar

```
java -jar ./mon-appli/target/mon-appli-1.0-SNAPSHOT.jar
```

Remarque : Comme Maven a indiqué la classe Main dans le Manifest du JAR, il n'est plus nécessaire d'indiquer cette classe dite principale dans la ligne de commande.

Hello World!

### C.1.10 Autre exemple d'utilisation des propriétés avec Maven : Affichage de la version du projet, en récupérant la valeur d'une propriété Maven définie dans un fichier de ressources

- On modifie la classe principale App de façon à ce que sa méthode Main affiche la version de l'application. La version de l'application va être récupérée dans la propriété org.exemple.demo.version que l'on va définir dans le fichier info.properties.

```

package org.exemple.demo;

import java.io.InputStream;
import java.util.Properties;

public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Hello World!" );

        Properties vProp = new Properties();
        InputStream vInputStream = null;
        try {
            vInputStream = App.class.getResourceAsStream("/info-properties");
            vProp.load(vInputStream);
        } finally {
            if (vInputStream != null) {
                vInputStream.close();
            }
        }
        System.out.println("Application version : " + vProp.getProperty("org.exemple.demo.version", " ?"));
    }
}

```

- On embarque le fichier src/main/resources/info-properties dans le JAR. Par convention, dans un projet Maven, les fichiers properties comme tous les autres fichiers de ressources doivent se trouver dans le répertoire src/main/resources/. On crée donc le répertoire src/main/resources/ puis le fichier src/main/resources/info-properties. On écrit dans src/main/resources/info.properties la ligne suivante :

```
org.exemple.demo.version=1.0
```

On appelle un tel fichier un fichier de propriétés.

- On reconstruit le projet Maven puis on compile le JAR :

```

cd ~/env/maven/maven-workspace/mon-appli
mvn package
java -jar ../mon-appli/target/mon-appli-1.0-SNAPSHOT.jar

```

Hello World!

Application version : 1.0

- Amélioration : Récupération de la propriété Maven depuis le fichier pom.xml par filtrage de ressources
  - Création de la resource src/main/resources et activation du filtrage pour cette dernière dans la section « Build » de pom.xml :

```

<resources>
  <resource>
    <directory>src/main/resources</directory>
    <filtering>true</filtering>
  </resource>
</resources>

```

Ainsi, à la construction de l'application par Maven, les fichiers contenus dans `src/main/resources` seront filtrés. C'est-à-dire que Maven va chercher à l'intérieur de ses fichiers des propriétés et il va les remplacer par leur valeur comme il le ferait dans le fichier `pom.xml`.

- On modifie le fichier `src/main/resources/info.properties` comme suit :

```
org.exemple.demo.version=${project.version}
```

- On reconstruit le projet Maven puis on compile le JAR :

```
cd ~/env/maven/maven-workspace/mon-appli
mvn package
java -jar ../mon-appli/target/mon-appli-1.0-SNAPSHOT.jar
```

Hello World!

Application version : 1.0-SNAPSHOT

### C.1.11 Utilisation de profils pour créer des options dans le build Maven

Par exemple, on peut gérer deux fichiers de propriétés, un pour un environnement de test et un pour un environnement de production, et faire en sorte que Maven utilise l'un ou l'autre de ces fichiers de propriétés selon la cible pour laquelle on est en train de générer le livrable.

- Création de deux sous-répertoires dans `src/main/resources/` :
  - `src/main/resources/conf-test`
  - `src/main/resources/conf-prod`
- Création de fichiers de propriétés (on supprime aussi `src/main/resources/info.properties` de l'exemple précédent) :
  - `src/main/resources/conf-test/info.properties`
  - `src/main/resources/conf-prod/info.properties`
- Édition des fichiers de propriétés :
  - Dans `src/main/resources/conf-test/info.properties` :
 

```
org.exemple.demo.version=${project.version} TEST
```
  - Dans `src/main/resources/conf-prod/info.properties` :
 

```
org.exemple.demo.version=${project.version} PROD
```
- Création de deux profils dans `pom.xml` avant la section `build` :

```

<!-- ===== -->
<!-- Profils -->
<!-- ===== -->
<profiles>
  <profile>
    <id>test</id>
    <build>
      <resources>
        <resource>
          <directory>src/main/resources/conf-test</directory>
          <filtering>true</filtering>
        </resource>
      </resources>
    </build>
  </profile>
  <profile>
    <id>prod</id>
    <build>
      <resources>
        <resource>
          <directory>src/main/resources/conf-prod</directory>
          <filtering>true</filtering>

```

```

    </resource>
  </resources>
</build>
</profile>
</profiles>

```

```

<!-- ===== -->
<!-- Build -->
<!-- ===== -->
<build>
<!-- ... -->
<!-- On y supprime aussi la balise resources de l'exemple précédent -->
<!-- ... -->
</build>

```

- Nettoyage et reconstruction Maven puis compilation Java :

```

cd ~/env/maven/maven-workspace/mon-appli/
mvn clean package -P test
java -jar target/mon-appli-1.1-SNAPSHOT.jar

```

Hello World!

Application version : 1.1-SNAPSHOT TEST

Remarque : Par convention Maven, tous les fichiers sous `src/main/resources/` sont copiés à la racine du JAR final. Ainsi l'instruction « `App.class.getResourceAsStream("/info-properties");` » de `App.java` récupère bien un flux sur le fichier contenant les propriétés relatif au profil concerné.

## C.1.12 Respecter une architecture multi-tiers ; exemple avec un projet de gestion de tickets d'incident

### C.1.12.1 Description

- Une architecture en couches
- L'interaction de l'utilisateur avec l'application se fait au niveau de la couche Présentation
- La couche Présentation contient le modèle et la vue du modèle MVC
- Une fois l'action utilisateur identifiée, la couche Présentation va faire appel à la couche Métier
- La couche Métier est responsable de la logique métier de l'application ; on y retrouve l'implémentation des règles de gestion fonctionnelles
- Si lors du traitement de la couche Métier, il est nécessaire de faire appel à des services extérieurs (comme ceux fournis par le serveur d'une base de données), alors la couche Métier fait appel à la couche Persistance
- Dans la couche Persistance, on retrouve le patron DAO
- Toutes les couches partagent une vision commune du domaine fonctionnel via le dit Modèle
- Le Modèle contient les javabeans manipulés par l'application
- Attention : Il ne faut pas confondre le Modèle de l'architecture multi-tiers avec le Modèle du patron MVC
- Le Modèle du patron MVC s'étale sur les couches Métier et Persistance et le Modèle de l'architecture multi-tiers
- Chaque couche de l'architecture multi-tiers ne peut qu'appeler que la couche inférieure et n'a aucune connaissance de la couche supérieure

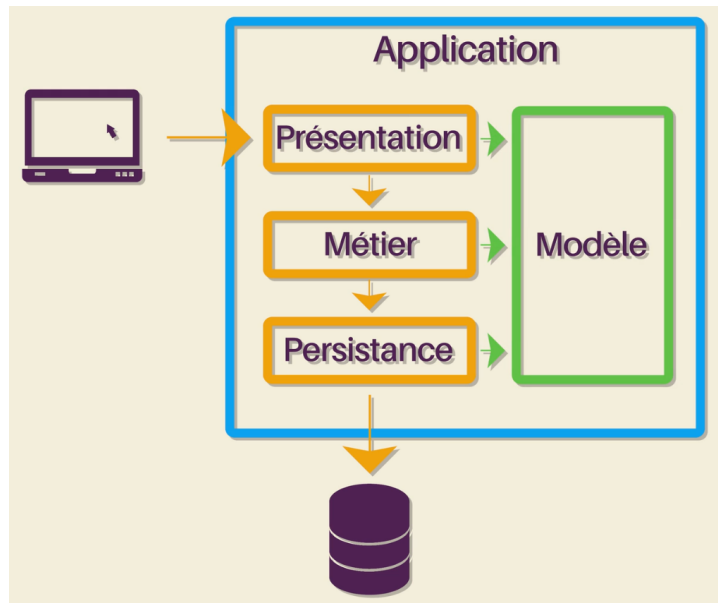


FIGURE 4 – Schéma de l'architecture multi-tiers provenant du cours « Organisez et packager une application Java avec Apache Maven » de Loïc Guibert sur [openclassrooms.com](https://openclassrooms.com).

- Plusieurs modules implémentant la couche Présentation peuvent utiliser les services d'un même module implémentant la couche Métier :

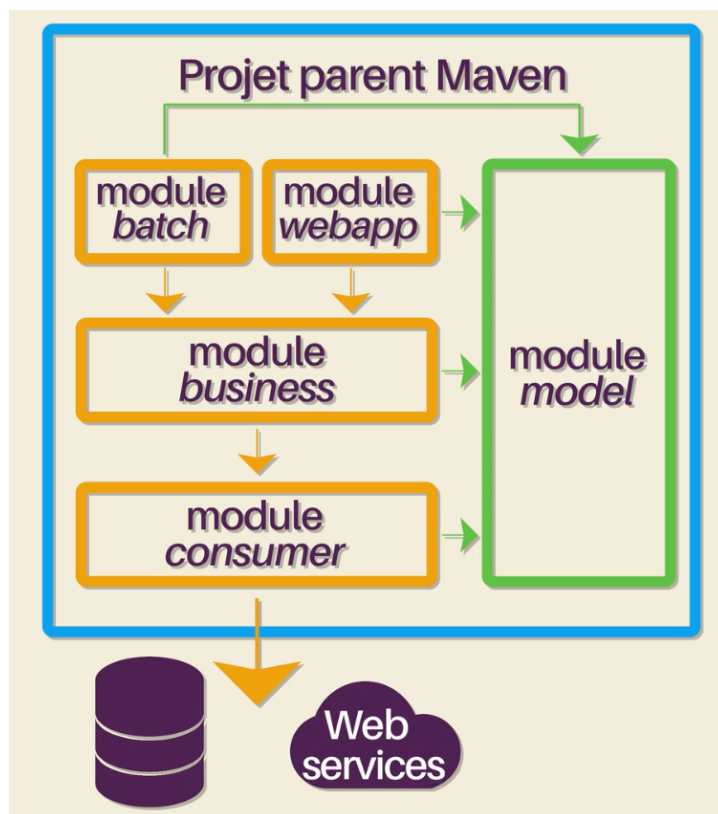


FIGURE 5 – Schéma de l'architecture multi-tiers provenant du cours « Organisez et packager une application Java avec Apache Maven » de Loïc Guibert sur [openclassrooms.com](https://openclassrooms.com) assimilant les couches à des modules et montrant qu'il peut y avoir plusieurs modules Présentation utilisant chacun les services du même module Métier.

#### C.1.12.2 Création d'un projet multi-modules respectant l'architecture multi-tiers

- Le projet contiendra une application web et et jeu de batchs. Il y aura donc deux couches Présentation.
- Création du projet parent

```
cd ~/env/maven/maven-workspace/
mvn archetype:generate \
  -DarchetypeGroupId=org.apache.maven.archetypes \
```

```

-DarchetypeArtifactId=maven-archetype-quickstart \
-DarchetypeVersion=1.1 \
-DgroupId=org.exemple.demo \
-DartifactId=ticket \
-Dversion=1.0-SNAPSHOT
rm -r ticket/src/

```

- importation du projet env/maven/maven-workspace/ticket dans le GUI de Eclipse : File → Import... → Maven → Existing Maven Projects → Next → Browse... → env/maven/maven-workspace/ticket → Finish
- Édition du fichier pom.xml

```

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0"
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.exemple.demo</groupId>
  <artifactId>ticket</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>

  <name>ticket</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

</project>

```

- Création de tous les modules de ce projet multi-modules :

```

cd ~/env/maven/maven-workspace/ticket
## module : ticket-batch
mvn -B archetype:generate \
  -DarchetypeGroupId=org.apache.maven.archetypes \
  -DarchetypeArtifactId=maven-archetype-quickstart \
  -DarchetypeVersion=1.1 \
  -DgroupId=org.exemple.demo \
  -DartifactId=ticket-batch \
  -Dpackage=org.exemple.demo.batch

## module : ticket-webapp
## Remarquez ici l'utilisation de l'archetype "webapp"
mvn -B archetype:generate \
  -DarchetypeGroupId=org.apache.maven.archetypes \
  -DarchetypeArtifactId=maven-archetype-webapp \
  -DgroupId=org.exemple.demo \
  -DartifactId=ticket-webapp \
  -Dpackage=org.exemple.demo.webapp

## module : ticket-business
mvn -B archetype:generate \
  -DarchetypeGroupId=org.apache.maven.archetypes \
  -DarchetypeArtifactId=maven-archetype-quickstart \
  -DarchetypeVersion=1.1 \
  -DgroupId=org.exemple.demo \
  -DartifactId=ticket-business \
  -Dpackage=org.exemple.demo.business

## module : ticket-consumer
mvn -B archetype:generate \
  -DarchetypeGroupId=org.apache.maven.archetypes \
  -DarchetypeArtifactId=maven-archetype-quickstart \
  -DarchetypeVersion=1.1 \
  -DgroupId=org.exemple.demo \

```

```
-DartifactId=ticket-consumer \
-Dpackage=org.exemple.demo.consumer
```

```
## module : ticket-model
```

```
mvn -B archetype:generate \
  -DarchetypeGroupId=org.apache.maven.archetypes \
  -DarchetypeArtifactId=maven-archetype-quickstart \
  -DarchetypeVersion=1.1 \
  -DgroupId=org.exemple.demo \
  -DartifactId=ticket-model \
  -Dpackage=org.exemple.demo.model
```

- On constate que tous les modules du projet ont été considérés dans ticket/pom.xml :

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema"
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.exemple.demo</groupId>
  <artifactId>ticket</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>

  <name>ticket</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <modules>
    <module>ticket-batch</module>
    <module>ticket-webapp</module>
    <module>ticket-business</module>
    <module>ticket-consumer</module>
    <module>ticket-model</module>
  </modules>
</project>
```

- Définition des dépendances entre les modules

- Importation du projet env/maven/maven-workspace/ticket/ dans IntelliJ (voir aaa)
- Dans le POM du projet parent ticket/pom.xml après les balises modules, on liste les modules en tant que dépendances pour le projet :

```
<modules>
  <module>ticket-batch</module>
  <module>ticket-webapp</module>
  <module>ticket-business</module>
  <module>ticket-consumer</module>
  <module>ticket-model</module>
</modules>

<!-- ===== -->
<!-- Gestions des dépendances -->
<!-- ===== -->
<dependencyManagement>
  <dependencies>
    <!-- ===== Modules ===== -->
    <dependency>
      <groupId>org.exemple.demo</groupId>
      <artifactId>ticket-batch</artifactId>
    </dependency>
    <dependency>
      <groupId>org.exemple.demo</groupId>
```

```

    <artifactId>ticket-business</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>
  <dependency>
    <groupId>org.exemple.demo</groupId>
    <artifactId>ticket-consumer</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>
  <dependency>
    <groupId>org.exemple.demo</groupId>
    <artifactId>ticket-model</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>
  <dependency>
    <groupId>org.exemple.demo</groupId>
    <artifactId>ticket-webapp</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>
</dependencies>
</dependencyManagement>

```

- On ajoute des dépendances à chaque module suivant le schéma suivant pour suivre l'architecture multi-tiers :

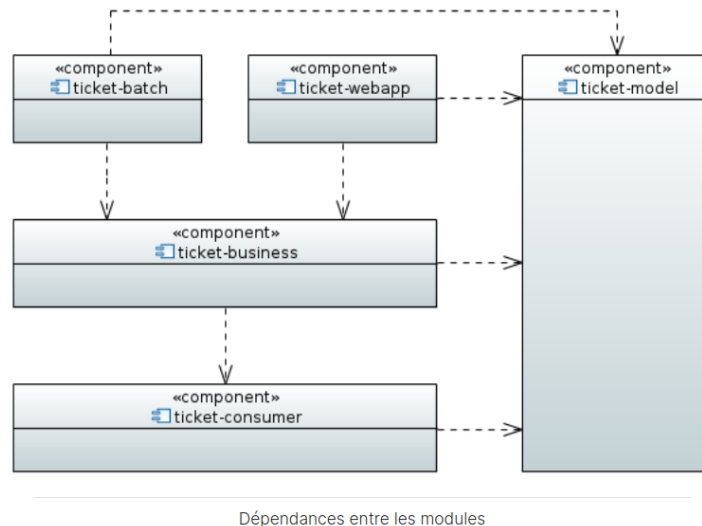


FIGURE 6 – Schéma représentant les dépendances entre les modules du projet env/maven/maven-workspace/ticket/ du cours « Organisez et packager une application Java avec Apache Maven » de Loïc Guibert sur openclassrooms.com.

- Dans ticket/ticket-batch/pom.xml :

```

<!-- ===== -->
<!-- Gestions des dépendances -->
<!-- ===== -->
<dependencies>
  <!-- ===== Modules ===== -->
  <dependency>
    <groupId>org.exemple.demo</groupId>
    <artifactId>ticket-business</artifactId>
  </dependency>
  <dependency>
    <groupId>org.exemple.demo</groupId>
    <artifactId>ticket-model</artifactId>
  </dependency>

  <!-- ===== Bibliothèques tierces ===== -->
  <!-- ... -->

```

- Dans ticket/ticket-webapp/pom.xml :

```

<!-- ===== -->
<!-- Gestions des dépendances -->

```



```

<!-- ===== -->
<dependencies>
  <!-- ===== Modules ===== -->
  <dependency>
    <groupId>org.exemple.demo</groupId>
    <artifactId>ticket-business</artifactId>
  </dependency>
  <dependency>
    <groupId>org.exemple.demo</groupId>
    <artifactId>ticket-model</artifactId>
  </dependency>

  <!-- ===== Bibliothèques tierces ===== -->
  <!-- ... -->

```

- Dans ticket/ticket-business/pom.xml :

```

<!-- ===== -->
<!-- Gestions des dépendances -->
<!-- ===== -->
<dependencies>
  <!-- ===== Modules ===== -->
  <dependency>
    <groupId>org.exemple.demo</groupId>
    <artifactId>ticket-consumer</artifactId>
  </dependency>
  <dependency>
    <groupId>org.exemple.demo</groupId>
    <artifactId>ticket-model</artifactId>
  </dependency>

  <!-- ===== Bibliothèques tierces ===== -->
  <!-- ... -->

```

- Dans ticket/ticket-consumer/pom.xml :

```

<!-- ===== -->
<!-- Gestions des dépendances -->
<!-- ===== -->
<dependencies>
  <!-- ===== Modules ===== -->
  <dependency>
    <groupId>org.exemple.demo</groupId>
    <artifactId>ticket-model</artifactId>
  </dependency>

  <!-- ===== Bibliothèques tierces ===== -->
  <!-- ... -->

```

- Création du projet en tant que projet multi-module

```

cd ~/env/maven/maven-workspace/ticket/
mvn package

```

### C.1.12.3 Gestion des dépendances tierces

- Ajouter une dépendance vers la bibliothèque commons-collections4 dans le projet env/maven/maven-workspace/ticket
  - Récupération des coordonnées Maven de la bibliothèque et ajout d'une dépendance vers cette bibliothèque :
    - On se rend sur le site d'url : <https://central.sonatype.com>  
Ce site permet de rechercher dans le repository central de Maven
    - On soumet dans la barre de recherche la chaîne " commons-collections"
    - On choisit la bibliothèque commons-collections4 fournie par l'organisation org.apache.commons
    - On choisit la dernière version 4.4
    - On récupère les coordonnées Maven de la bibliothèque dans des balises dependency :

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-collections4</artifactId>
  <version>4.4</version>
</dependency>
```

- On ajoute ce snippet dans la section « Bibliothèques tierces » de ticket/ticket-webapp/pom.xml :

```
<!-- ===== Bibliothèques tierces ===== -->
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.11</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-collections4</artifactId>
  <version>4.4</version>
</dependency>
</dependencies>
```

- On récupère
- `cd ~/env/maven/maven-workspace/ticket`  
`mvn package`

```

[INFO] -----< org.example.demo:ticket-webapp >-----
[INFO] Building ticket-webapp Maven Webapp 1.0-SNAPSHOT [6/6]
[INFO] from ticket-webapp/pom.xml
[INFO] -----[ war ]-----
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-collections4/4.4/commons-collections4-4.4.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-collections4/4.4/commons-collections4-4.4.pom (24 kB at 26
kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-collections4/4.4/commons-collections4-4.4.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-collections4/4.4/commons-collections4-4.4.jar (752 kB at 1.
1 MB/s)
[INFO]
[INFO] --- resources:3.0.2:resources (default-resources) @ ticket-webapp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/workaholic/env/maven/maven-workspace/ticket/ticket-webapp/src/main/resources
[INFO]
[INFO] --- compiler:3.8.0:compile (default-compile) @ ticket-webapp ---
[INFO] No sources to compile
[INFO]
[INFO] --- resources:3.0.2:testResources (default-testResources) @ ticket-webapp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/workaholic/env/maven/maven-workspace/ticket/ticket-webapp/src/test/resources
[INFO]
[INFO] --- compiler:3.8.0:testCompile (default-testCompile) @ ticket-webapp ---
[INFO] No sources to compile
[INFO]
[INFO] --- surefire:2.22.1:test (default-test) @ ticket-webapp ---

```

FIGURE 7 – GUI de IntelliJ. Extrait des retours du terminal lors du packaging de env/maven/maven-workspace/ticket. On constate que la bibliothèque commons-collections4-4.4 est téléchargée.

```
$ tree -a ~/env/maven/maven-workspace/ticket/ticket-webapp/
/home/workaholic/env/maven/maven-workspace/ticket/ticket-webapp/
```

```

├── pom.xml
├── src
│   ├── main
│   │   └── webapp
│   │       ├── index.jsp
│   │       ├── WEB-INF
│   │       └── web.xml
├── target
│   ├── maven-archiver
│   │   └── pom.properties
│   ├── ticket-webapp
│   │   ├── index.jsp
│   │   ├── META-INF
│   │   ├── WEB-INF
│   │   │   ├── classes
│   │   │   ├── lib
│   │   │   │   ├── commons-collections4-4.4.jar
│   │   │   │   ├── ticket-business-1.0-SNAPSHOT.jar
│   │   │   │   ├── ticket-consumer-1.0-SNAPSHOT.jar
│   │   │   │   └── ticket-model-1.0-SNAPSHOT.jar
│   │   └── web.xml
└── ticket-webapp.war

```

11 directories, 11 files

- Ajouter la bibliothèque commons-lang3 par transitivité en ajoutant la bibliothèque commons-text
  - Dans ticket/ticket-webapp/pom.xml :

```

<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-text</artifactId>
  <version>1.1</version>
</dependency>

```

- On package ticket :
 

```
cd ~/env/maven/maven-workspace/ticket/
mvn package
```

La bibliothèque commons-lang3 a bien été ajoutée :

```
$ tree -a ~/env/maven/maven-workspace/ticket/ticket-webapp/
```

```

/home/workaholic/env/maven/maven-workspace/ticket/ticket-webapp/
├── pom.xml
├── src
│   └── main
│       └── webapp
│           ├── index.jsp
│           ├── WEB-INF
│           └── web.xml
└── target
    ├── maven-archiver
    │   └── pom.properties
    ├── ticket-webapp
    │   ├── index.jsp
    │   ├── META-INF
    │   ├── WEB-INF
    │   ├── classes
    │   ├── lib
    │   │   ├── commons-collections4-4.4.jar
    │   │   ├── commons-lang3-3.5.jar
    │   │   ├── commons-text-1.1.jar
    │   │   ├── ticket-business-1.0-SNAPSHOT.jar
    │   │   ├── ticket-consumer-1.0-SNAPSHOT.jar
    │   │   └── ticket-model-1.0-SNAPSHOT.jar
    │   └── web.xml
    └── ticket-webapp.war

```

11 directories, 13 files

- Exclusion de la bibliothèque commons-lang3 lors de l'ajout de la bibliothèque commons-text (qui dépend de la bibliothèque commons-lang3)
- Dans ticket/ticket-webapp/pom.xml :

```

<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-text</artifactId>
  <version>1.1</version>
  <exclusions>
    <exclusion>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-lang3</artifactId>
    </exclusion>
  </exclusions>
</dependency>

```

- On nettoie et on package ticket :

```

cd ~/env/maven/maven-workspace/ticket/
mvn clean package

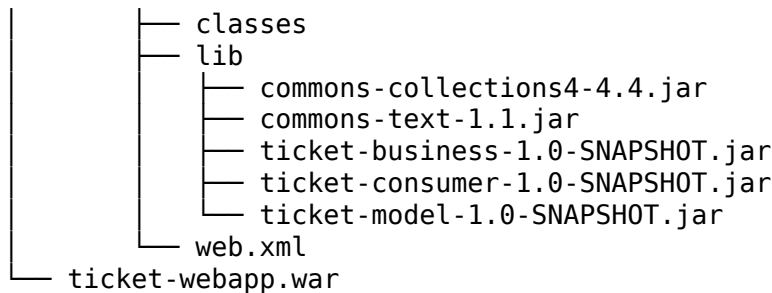
```

La bibliothèque commons-lang3 a bien été exclue :

```

$ tree -a ~/env/maven/maven-workspace/ticket/ticket-webapp/
/home/workaholic/env/maven/maven-workspace/ticket/ticket-webapp/
├── pom.xml
├── src
│   └── main
│       └── webapp
│           ├── index.jsp
│           ├── WEB-INF
│           └── web.xml
└── target
    ├── maven-archiver
    │   └── pom.properties
    ├── ticket-webapp
    │   ├── index.jsp
    │   ├── META-INF
    │   └── WEB-INF

```



11 directories, 12 files

#### C.1.12.4 Gestion des scopes des dépendances

- Par défaut, le scope d'une dépendance est compile, i.e. la dépendance est accessible à la compilation et dans tous les contextes.
- Les autres scopes :
  - test : accessible que lors de la compilation et lors des tests
  - provided : la dépendance est accessible lors de la compilation mais doit être fournie par le contexte d'exécution, par exemple le serveur d'applications sur lequel va tourner l'application
  - runtime : accessible que lors de l'exécution
- Exemple 1 avec le scope test

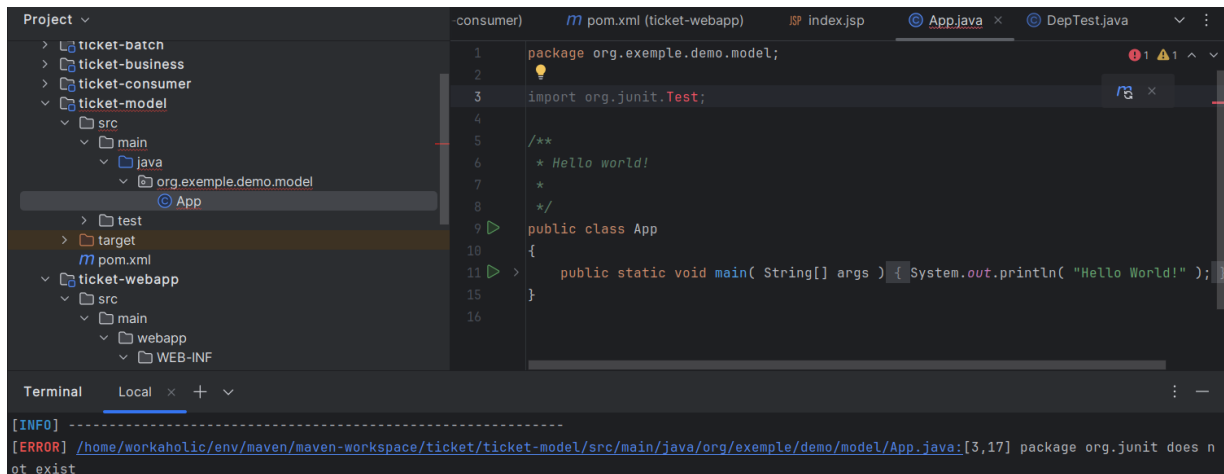


FIGURE 8 – GUI de IntelliJ montrant que l'importation de `import org.junit.Test` dans une classe qui n'est pas une classe de test provoque une erreur.

- Exemple 2 avec le scope provided  
L'application ticket-webapp va être déployée sur un serveur d'applications Jakarta EE. Ce serveur d'applications va déjà fournir la bibliothèque servlet-api dans l'environnement d'exécution. Pour éviter un conflit, il ne faut donc pas avoir cette bibliothèque dans le WAR. Cependant, cette bibliothèque est nécessaire pour la compilation. Le scope provided résout ce problème.

- Ajout de la dépendance servlet-api au module ticket-webapp avec le scope provided ; dans ticket/ticket-webapp/pom.xml :

```

<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>3.1.0</version>
  <scope>provided</scope>
</dependency>

```

Ainsi, on peut accéder aux éléments de la servlet-api en compilation mais la servlet-api ne sera pas mise dans les lib du WAR

- On crée une servlet ; dans ticket-webapp/src/main/java/org/exemple/demo/servlet/MaServlet.java :

```

package org.exemple.demo.servlet;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

```

```

public class MaServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    public MaServlet() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        response.getWriter().append("Served at: ").append(request.getContextPath());
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        // TODO Auto-generated method stub
        doGet(request, response);
    }
}

```

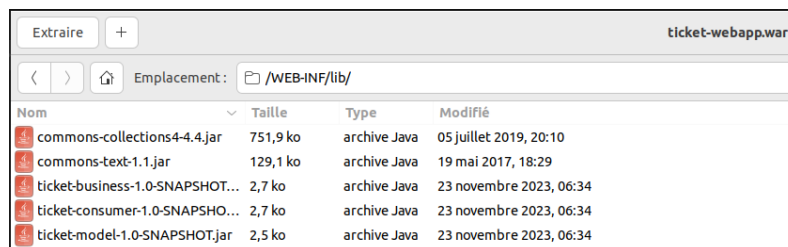
- On nettoie et on package le projet Maven ticket :

```

cd ~/env/maven/maven-workspace/ticket/
mvn clean package

```

- On constate qu'il n'y a pas de bibliothèque servlet-api dans ticket/ticket-webapp/target/ticket-webapp.war/WEB-INF/lib/ :



Nom	Taille	Type	Modifié
commons-collections4-4.4.jar	751,9 ko	archive Java	05 juillet 2019, 20:10
commons-text-1.1.jar	129,1 ko	archive Java	19 mai 2017, 18:29
ticket-business-1.0-SNAPSHOT.jar	2,7 ko	archive Java	23 novembre 2023, 06:34
ticket-consumer-1.0-SNAPSHOT.jar	2,7 ko	archive Java	23 novembre 2023, 06:34
ticket-model-1.0-SNAPSHOT.jar	2,5 ko	archive Java	23 novembre 2023, 06:34

- Exemple 3 : Validation de bean

- Ajout des dépendances javax.validation et org.apache.bval au projet ticket avec le scope aaa ; dans ticket/pom.xml :

```

<!-- ===== Bibliothèques tierces ===== -->
<dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
    <version>2.0.1.Final</version>
</dependency>
<dependency>
    <groupId>org.apache.bval</groupId>
    <artifactId>bval-jsr303</artifactId>
    <version>0.5</version>
    <scope>runtime</scope>
</dependency>

```

Via ces bibliothèques, on peut faire de la validation de bean. Il s'agit de la spécification JSR 303. javax.validation fournit notamment les annotations à poser sur les beans pour pouvoir faire de la validation de bean. Pour que cette validation fonctionne, il faut une implémentation de cette API de validation. Il en existe plusieurs. On utilise ici org.apache.bval.bundle.

Il ne faut pas utiliser bval directement. En effet, l'API de validation des beans s'appuie sur le mécanisme SPI (Service provider Interfaces). C'est-à-dire que l'implémentation de l'API est chargée et câblée dynamiquement par la JVM à l'exécution. Ce qui veut dire que l'on a pas besoin de bval pour compiler mon code. J'en ai besoin uniquement pendant l'exécution de l'application. On utilise donc le scope runtime. Grâce à ce scope, bval va être intégré au WAR généré mais on ne pourra pas accéder à bval depuis les classes lors de la compilation

- On nettoie et on package le projet Maven ticket :

```

cd ~/env/maven/maven-workspace/ticket/
mvn clean package

```

- On constate qu'on a bien les deux bibliothèques susmentionnées dans ticket/ticket-webapp/target/ticket-webapp.war/WEB-INF/lib/ et d'autres bibliothèques obtenues par transitivité des dépendances :

Extraire +		ticket-webapp.war		
		Emplacement : /WEB-INF/lib/		
Nom	Taille	Type	Modifié	
bval-core-0.5.jar	62,0 ko	archive Java	18 septembre 2012, 15:31	
bval-jsr303-0.5.jar	247,0 ko	archive Java	18 septembre 2012, 15:35	
commons-beanutils-core-1.8.3...	206,7 ko	archive Java	24 mars 2010, 18:13	
commons-collections4-4.4.jar	751,9 ko	archive Java	05 juillet 2019, 20:10	
commons-lang3-3.1.jar	315,8 ko	archive Java	15 novembre 2011, 08:27	
commons-text-1.1.jar	129,1 ko	archive Java	19 mai 2017, 18:29	
ticket-business-1.0-SNAPSHOT...	2,7 ko	archive Java	23 novembre 2023, 13:01	
ticket-consumer-1.0-SNAPSHO...	2,7 ko	archive Java	23 novembre 2023, 13:01	
ticket-model-1.0-SNAPSHOT.jar	2,5 ko	archive Java	23 novembre 2023, 13:01	
validation-api-2.0.1.Final.jar	93,1 ko	archive Java	19 décembre 2017, 17:23	

### C.1.12.5 Gestion des dépendances d'un projet multi-module de manière globale

- Dans le fichier POM ticket/pom.xml du projet parent, il liste toutes les dépendances tierces des sous-modules après la liste de ces derniers :

```

<!-- ===== Modules ===== -->
<!-- Gestions des dépendances -->
<!-- ===== Modules ===== -->
<dependencyManagement>
  <dependencies>
    <!-- ===== Modules ===== -->
    <dependency>
      <groupId>org.exemple.demo</groupId>
      <artifactId>ticket-batch</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>
    <dependency>
      <groupId>org.exemple.demo</groupId>
      <artifactId>ticket-business</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>
    <dependency>
      <groupId>org.exemple.demo</groupId>
      <artifactId>ticket-consumer</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>
    <dependency>
      <groupId>org.exemple.demo</groupId>
      <artifactId>ticket-model</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>
    <dependency>
      <groupId>org.exemple.demo</groupId>
      <artifactId>ticket-webapp</artifactId>
      <version>1.0-SNAPSHOT</version>
    </dependency>

    <!-- ===== Bibliothèques tierces ===== -->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>

    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>javax.validation</groupId>
      <artifactId>validation-api</artifactId>
      <version>2.0.1.Final</version>
    </dependency>

    <dependency>
      <groupId>org.apache.bval</groupId>
      <artifactId>bval-jsr303</artifactId>
      <version>0.5</version>
      <scope>runtime</scope>
    </dependency>

    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-collections4</artifactId>
      <version>4.4</version>
    </dependency>
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-text</artifactId>
      <version>1.1</version>
      <exclusions>
        <exclusion>
          <groupId>org.apache.commons</groupId>
          <artifactId>commons-lang3</artifactId>
        </exclusion>
      </exclusions>
    </dependency>
  </dependencies>
</dependencyManagement>

```

- Dans les sous-modules, on peut alors alléger la section des dépendances du fait du mécanisme d'héritage ; par exemple, dans le fichier POM ticket/ticket-webapp.xml du sous-module ticket/webapp, on écrit simplement :

```

<!-- ===== Modules ===== -->
<!-- Gestions des dépendances -->
<!-- ===== Modules ===== -->
<dependencies>
  <!-- ===== Modules ===== -->
  <dependency>
    <groupId>org.exemple.demo</groupId>
    <artifactId>ticket-business</artifactId>
  </dependency>
  <dependency>
    <groupId>org.exemple.demo</groupId>

```

```

    <artifactId>ticket-model</artifactId>
  </dependency>

  <!-- ===== Bibliothèques tierces ===== -->
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
  </dependency>

  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
  </dependency>

  <dependency>
    <groupId>javax.validation</groupId>
    <artifactId>validation-api</artifactId>
  </dependency>

  <dependency>
    <groupId>org.apache.bval</groupId>
    <artifactId>bval-jsr303</artifactId>
  </dependency>

  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-collections4</artifactId>
  </dependency>

  <dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-text</artifactId>
  </dependency>
</dependencies>

```

### C.1.13 Description générale des cycle de vie d'une construction Maven (build lifecycle)

- Trois cycles de vie :
  - default : Construit un déploie le projet.
  - clean : Supprime tous les éléments issus des constructions précédentes.
  - site : Crée un site web pour le projet.
- Chaque cycle de vie est découpé en phases qui s'exécutent séquentiellement.
- Par exemple, le cycle de vie default se découpe en les phases suivantes :
  - validate : vérifie que la configuration projet est correcte (POM, pas d'éléments manquants...)
  - compile : compile les sources du projet
  - test : teste le code compilé avec les classes de tests unitaires contenues dans le projet
  - package : package les éléments issus de la compilation dans un format distribuable (JAR, WAR...)
  - install : installe le package dans votre repository local
  - deploy : envoie le package dans le repository distant défini dans le POM
- Par exemple :
  - « **mvn package** » exécute toutes les phases du cycle de vie default jusqu'à la phase package incluse.
  - « **mvn clean package** » exécute toutes les phases du cycle de vie clean puis toutes les phases du cycle de vie default jusqu'à la phase package incluse.
- Chaque phase se découpe en tâche ; chaque tâche est assurée par un plugin ; dans le jargon de Maven, une tâche s'appelle est goal. Par exemple, la phase test est réalisée par défaut par le goal surefire:test, c'est-à-dire le goal test du plugin surefire.
- Pour les projets multi-modules, quand on lance la commande mvn sur le projet parent, Maven lance le build lifecycle dans chaque sous-projet (module), les uns à la suite des autres, en respectant l'ordre des dépendances inter-modules.

### C.1.14 Utilisation des plugins pour personnaliser la construction Maven

- À chaque phase sont câblés différentes goals assurés par différent plugins
- Via <https://maven.apache.org/ref/3.9.5/maven-core/lifecycles.html>, on trouve :
  - La liste des différentes phases pour chaque cycle de vie.
  - Des liens comme <https://maven.apache.org/ref/3.9.5/maven-core/default-bindings.html> lien pour récupérer la liste des goals des plugin câblés aux différentes phases du build lifecycle default.
  - Par exemple, on voit ci-dessous que le goal install de la phase install du cycle de vie default est assuré par le plugin org.apache.maven.plugins:maven-install-plugin:3.1.1

```

<phases>
  <install>
    org.apache.maven.plugins:maven-install-plugin:3.1.1:install
  </install>
  <deploy>
    org.apache.maven.plugins:maven-deploy-plugin:3.1.1:deploy
  </deploy>
</phases>

```



- Exemple de personnalisation 1 : Afficher lors de la compilation tous les messages d'avertissement de compilation et l'utilisation des méthodes dépréciées
  - On remarque que la phase compile du cycle de vie default est assuré par le goal compile du plugin org.apache.maven.plugins:maven-compiler-plugin:3.11.0
  - On récupère la documentation des plugins (étiquette « Plugins » dans le menu vertical liée à <https://maven.apache.org/plugins/index.html>)
  - On choisit le plugin compiler (<https://maven.apache.org/plugins/maven-compiler-plugin/>)
  - On récupère la page concernant ses goals (étiquette « Goals » dans le menu vertical liée à <https://maven.apache.org/plugins/maven-compiler-plugin/plugin-info.html>)
  - On récupère la documentation de son goal compile (étiquette « Compiler:compile » liée à <https://maven.apache.org/plugins/maven-compiler-plugin/compile-mojo.html>)
  - On parcourt la liste des options
  - On remarque les noms d'option « showDeprecation » et « showWarning »
    - Par défaut, ses options sont à false.
    - On veut les passer à true.
  - On écrit dans ticket/ticket-webapp/pom.xml :

```
<build>
  <finalName>ticket-webapp</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <configuration>
        <showDeprecation>true</showDeprecation>
        <showWarnings>true</showWarnings>
      </configuration>
    </plugin>
    <plugin>
      [...]
```

- On remarque aussi à l'url <https://maven.apache.org/plugins/maven-compiler-plugin/compile-mojo.html> que les options showDeprecation et showWarning sont associées respectivement aux propriétés maven.compiler.showDeprecation et maven.compiler.showWarnings. Pour utiliser le mécanisme des propriétés :

- On supprime l'environnement balisé configuration précédent
- On écrit dans ticket/ticket-webapp/pom.xml :

```
<properties>
  <maven.compiler.showDeprecation>true</maven.compiler.showDeprecation>
  <maven.compiler.showWarnings>true</maven.compiler.showWarnings>
  [...]
</properties>
[...]
<build>
  <finalName>ticket-webapp</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
    </plugin>
    <plugin>
      [...]
```

- Ces deux manières de configurer sont équivalentes
- Exemple de personnalisation 2 : Définition de la classe Main dans le MANIFEST du JAR généré par Maven
  - Page du site consacré à Apache Maven concernant les cycles de vie : <https://maven.apache.org/ref/3.9.5/maven-core/lifecycles.html>
  - Page du site consacré à Apache Maven concernant la câblage des plugins aux goals des tâches du cycle de vie default : <https://maven.apache.org/ref/3.9.5/maven-core/default-bindings.html>
  - Dans la section « Plugin bindings for jar packaging », on voit ci-dessous que le goal jar de la phase package du cycle de vie default est assuré par le plugin org.apache.maven.plugins:maven-jar-plugin:3.3.0

```

<phases>
  [...]
  <package>
    org.apache.maven.plugins:maven-jar-plugin:3.3.0:jar
  </package>
</phases>

```

- section « Plugins » via <https://maven.apache.org/plugins/index.html>
- Dans la section « Packaging types/tools », on choisit « jar ».
- En cliquant sur « jar », via <https://maven.apache.org/plugins/maven-jar-plugin/>, on récupère la page concernant les goals du plugin susmentionné
- On choisit « jar:jar » liée à <https://maven.apache.org/plugins/maven-jar-plugin/jar-mojo.html>
- Dans la section « Optional Parameters », on remarque l'option archive qui renvoie sur la documentation de Apache Maven Archiver via <https://maven.apache.org/shared/maven-archiver/index.html>
- Dans la section « manifest », on obtient les éléments de configuration du manifest.
- On en déduit donc la configuration adéquate à ajouter dans ticket/ticket-batch/pom.xml :

```

<!-- ===== -->
<!-- Build -->
<!-- ===== -->
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <configuration>
        <manifest>
          <mainClass>org.exemple.demo.batch.App</mainClass>
        </manifest>
      </configuration>
    </plugin>
  </plugins>
</build>

```

Et la configuration globale à ajouter dans ticket/pom.xml :

```

<!-- ===== -->
<!-- Build -->
<!-- ===== -->
<build>
  <!-- Gestion des plugins (version) -->
  <pluginManagement>
    <plugins>
      <!-- Plugin responsable de la génération du fichier JAR -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>3.0.2</version>
        <configuration>
          <archive>
            <manifest>
              <mainClass>
                org.exemple.demo.
              </mainClass>
            </manifest>
          </archive>
        </plugin>
      <plugin>
        [...]

```

Comme le chemin vers la classe principale diffère suivant les sous-modules, on a pas pu le configurer de manière globale via le mécanisme d'héritage.

- Exemple de personnalisation 3 : S'assurer qu'au moins un des profils est activé
  - Dans ticket/ticket-webapp/pom.xml :

```

<!-- ===== -->
<!-- Profils -->
<!-- ===== -->
<profiles>
  <profile>
    <id>target-dev</id>
    <!-- ... -->
  </profile>
  <profile>
    <id>target-test</id>
    <!-- ... -->
  </profile>
  <profile>
    <id>target-prod</id>
    <!-- ... -->
  </profile>
</profiles>

```

- On utilise le plugin maven-enforcer-plugin.
- On récupère la page de documentation le concernant via l'url : <https://maven.apache.org/enforcer/maven-enforcer-plugin/>
- On choisit le goal enforce via l'url <https://maven.apache.org/enforcer/maven-enforcer-plugin/enforce-mo.html>
- On récupère la liste des built-in rules via l'url <https://maven.apache.org/enforcer/enforcer-rules/index.html>
- On choisit la règle requireActiveProfile, ce qui nous mène à sa page de documentation à l'url <https://maven.apache.org/enforcer/enforcer-rules/requireActiveProfile.html> qui présente des exemple de configuration du plugin faisant appel à elle.
- Dans ticket/pom.xml :

```

<!-- ===== -->
<!-- Build -->
<!-- ===== -->
<build>
  <!-- Gestion des plugins (version) -->
  <pluginManagement>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-enforcer-plugin</artifactId>
        <version>3.4.1</version>
      </plugin>
      <!-- Plugin responsable de la génération du fichier JAR -->
      <plugin>
        [...]

```

- Dans ticket/pom.xml :

```

<!-- ===== -->
<!-- Build -->
<!-- ===== -->
<build>
  <finalName>ticket-webapp</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-enforcer-plugin</artifactId>
      <executions>
        <execution>
          <id>enforce-profile-target</id>
          <phase>validate</phase>
          <goals>
            <goal>enforce</goal>
          </goals>

```

```

    <configuration>
      <rules>
        <requireActiveProfile>
          <profiles>target-dev,target-test,target-prod</profiles>
          <all>false</all>
        </requireActiveProfile>
      </rules>
    </configuration>
  </execution>
</executions>
</plugin>
<plugin>
  [...]

```

- Les sections execution permettent de câbler de nouveaux goals aux phases

### C.1.15 Packager les livrables ; exemple avec un projet de gestion de tickets d'incident

Objectifs :

- Créer un fichier WAR pour l'application web
- Créer un fichier ZIP pour les batches
- Obtenir un livrable contenant tout le nécessaire, c'est-à-dire l'application, les dépendances, les fichiers de configuration, etc...
- Générer un fichier WAR pour le déploiement de l'application web
  - On étoffe le processus de construction pour atteindre les objectifs suivant :
    - Ajouter automatiquement certaines informations du projet dans les JSP
    - Assurer qu'aucune version SNAPSHOT ne soit envoyée en production
  - Ajouter automatiquement certaines informations du projet dans les JSP :
    - On utilise le mécanisme de filtrage pour filtrer des webresources. Par convention Maven, ces webresources doivent être situées sous le répertoire src/main/webapp/.
    - Pour effectuer ce filtrage de webresources, on utilise le plugin Maven org.apache.maven.plugins:maven-war-plugin:3.4.0 (<https://mavenlibs.com/maven/plugin/org.apache.maven.plugins/maven-war-plugin~:text=Latest%20Stable%3A%203,in%20this%20tag%20as%20follows>)
  - Dans ticket/pom.xml :

```

<!-- ===== -->
<!-- Build -->
<!-- ===== -->
<build>
  <pluginManagement>
    [...]
  <plugins>
    <!-- Packaging WAR -->
    <!-- https://mavenlibs.com/maven/plugin/org.apache.maven.plugins/maven-war-plugin -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-war-plugin</artifactId>
      <version>3.4.0</version>
    </plugin>
  </plugins>

```

- Dans les conventions Maven, le répertoire par défaut des web resources est src/main/webapp. Dans ticket/ticket-webapp/pom.xml, on configure donc le plugin war comme suit :

```

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-war-plugin</artifactId>
  <configuration>
    <webResources>
      <resource>
        <directory>src/main/webapp</directory>
        <filtering>true</filtering>
        <includes>
          <include>jsp/_include/header.jsp</include>

```

```

        <include>jsp/_include/footer.jsp</include>
        <include>jsp/about.jsp</include>
    </includes>
</resource>
</webResources>
</configuration>
</plugin>

```

On configure ainsi un filtrage des ressources dans les fichiers ticket/ticket-webapp/src/main/webapp/jsp/\_include/header.jsp, ticket/ticket-webapp/src/main/webapp/jsp/\_include/footer.jsp et ticket/ticket-webapp/src/main/

- Encore dans ticket/ticket-webapp/pom.xml, on définit les propriétés qui seront utiles par la suite :

```

<!-- ===== -->
<!-- Propriétés -->
<!-- ===== -->
<properties>
    <!-- Le nom "public" de l'application -->
    <application.name>TicketTac</application.name>
    <!-- Le format à utiliser pour afficher la date du build -->
    <maven.build.timestamp.format>dd/MM/yyyy</maven.build.timestamp.format>
    <!-- Propriété servant à contourner le bug du non remplacement
         de la propriété maven.build.timestamp lors du filtrage des ressources -->
    <maven.build.timestamp>${maven.build.timestamp}</maven.build.timestamp>
    <buildTimestamp>${maven.build.timestamp}</buildTimestamp>
</properties>

```

- Dans ticket/pom.xml, on ajoute des informations complémentaires :

```

<!-- ===== -->
<!-- Informations du projet -->
<!-- ===== -->
<!-- ===== Informations Maven ===== -->
<groupId>org.exemple.demo</groupId>
<artifactId>ticket</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>pom</packaging>

<!-- ===== Informations générales ===== -->
<name>ticket</name>
<url>http://maven.apache.org</url>

<organization>
    <name>Mon Entreprise</name>
    <url>http://exemple.org</url>
</organization>

```

- On complète les JSP en utilisant les propriétés et les informations complémentaires précédentes de façon à ce que leur valeur soit injectée par Maven automatiquement lors du build

- Dans ticket/ticket-webapp/src/main/webapp/jsp/\_include/header.jsp :

```

<%% page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>

<nav class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-
                <span class="sr-only">Toggle navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="#">${application.name}</a>
        </div>
        <div id="navbar" class="collapse navbar-collapse">
            <ul class="nav navbar-nav">
                <li class="active"><a href="..">Accueil</a></li>
                <li><a href="..jsp/about.jsp">A propos</a></li>
            </ul>
        </div><!--/.nav-collapse -->
    </div>
</nav>

```

- Dans ticket/ticket-webapp/src/main/webapp/jsp/\_include/footer.jsp :

```

<%% page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>

<footer class="footer">

```

```

<div class="container">
  <p>
    ${application.name} - version ${project.version}
    &copy; <a href="${organization.url}">${organization.name}</a>
  </p>
</div>
</footer>

<!-- Bootstrap -->
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
  integrity="sha384-Tc5IQib027qvyjSMfHj0MaLkfuWvXZxUPnCJA7L2mCWNIPG9mGCD8wGNIcPD7Txa"
  crossorigin="anonymous"></script>

```

- Dans ticket/ticket-webapp/src/main/webapp/jsp/about.jsp :

```

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>

<!DOCTYPE html>
<html lang="fr">

<head>
  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>${application.name} - A propos</title>
  <!-- Bootstrap -->
  <link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
    integrity="sha384-BVYiIIFeKldGmJRAKycuHAHRg320mUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
    crossorigin="anonymous" />
  <link rel="stylesheet"
    href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap-theme.min.css"
    integrity="sha384-rHyoNliRsVXV4nD0JutlnGaslCJuC7uwjduW9SVrLvRYooPp2bWYgmgJQIXwL/Sp"
    crossorigin="anonymous" />
  <link rel="stylesheet" href="../style/custom.css" />
</head>

<body>

<%@ include file="_include/header.jsp" %>

<div class="container">
  <ul>
    <li>Application : ${application.name}</li>
    <li>Version : ${project.version}</li>
    <li>Date du build : ${maven.build.timestamp}</li>
  </ul>
</div>

<%@ include file="_include/footer.jsp" %>
</body>
</html>

```

- Assurer qu'aucune version SNAPSHOT de l'application web ne soit envoyée en production
  - On utilise plugin org.apache.maven.plugins:maven-enforcer-plugin:3.4.1
  - Ce plugin est déjà listé dans la section pluginManagement du POM parent ticket/pom.xml :

```

<!-- ===== -->
<!-- Build -->
<!-- ===== -->
<build>
  <pluginManagement>
    <plugins>
      [...]
      <!-- Vérification des règles sur le contexte de construction Maven -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-enforcer-plugin</artifactId>
        <version>3.4.1</version>
      </plugin>
      [...]
    </plugins>
  </pluginManagement>
</build>

```

- On associe cette configuration au profil target-prod. On le configure donc dans la section project.profiles.profile.target-prod du POM de l'application web ticket/ticket-webapp/pom.xml :

```

<!-- ===== -->
<!-- Profils -->
<!-- ===== -->
<profiles>
  <profile>
    <id>target-dev</id>
    <!-- ... -->
  </profile>

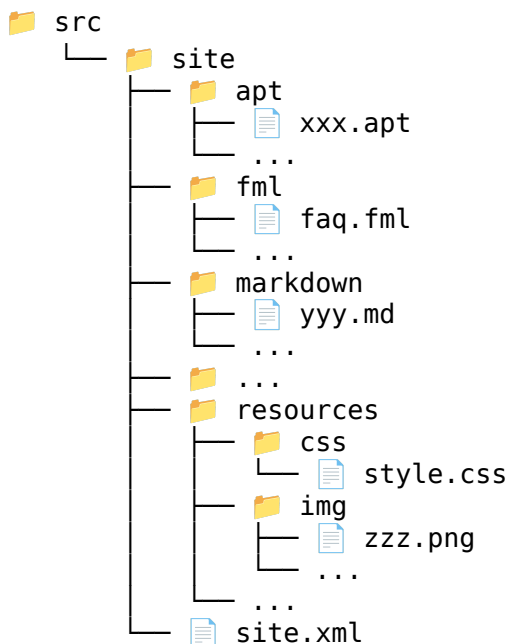
```

```

<profile>
  <id>target-test</id>
  <!-- ... -->
</profile>
<profile>
  <id>target-prod</id>
  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-enforcer-plugin</artifactId>
        <executions>
          <execution>
            <id>enforce-no-snapshot-prod</id>
            <phase>validate</phase>
            <goals>
              <goal>enforce</goal>
            </goals>
            <configuration>
              <rules>
                <!-- Le profil et son parent ne doivent pas être en SNAPSHOT -->
                <requireReleaseDeps>
                  <message>No Snapshots Allowed!</message>
                </requireReleaseDeps>
                <requireReleaseDeps>
                  <message>No Snapshots Allowed!</message>
                </requireReleaseDeps>
              </rules>
            </configuration>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</profile>
</profiles>

```

- Générer une archive pour le jeu de batches
  - Objectifs :
    - générer une archive (un fichier TAR.GZ et un fichier ZIP) pour le jeu de batchs de l'application de gestion de tickets
    - Cette archive contiendra :
      - le JAR du module ticket-batch
      - les JAR de toutes ses dépendances
      - les fichiers de configuration (modifiables facilement par l'administrateur)
      - les scripts shell de lancement de chaque batch
    - Cette archive aura l'arborescence suivante :



- On crée les répertoires ticket-batch/src/data/scripts et ticket-batch/src/data/conf destinés à contenir respectivement tous les scripts de lancement des batchs et les fichiers de configuration :

```

mkdir ~/env/maven/maven-workspace/ticket/ticket-batch/src/data
mkdir ~/env/maven/maven-workspace/ticket/ticket-batch/src/data/scripts
mkdir ~/env/maven/maven-workspace/ticket/ticket-batch/src/data/conf

```

Ce répertoire ticket-batch/src/data va permettre d'alimenter les répertoires conf et bin de l'archive.

- On configure la création du JAR des batches
  - On ajoute le classpath dans le manifest de ce JAR. Ainsi, toutes les classes seront automatiquement retrouvées par le class loader de la JVM lors de l'exécution du JAR. Pour cela, on complète la configuration du plugin jar dans ticket/ticket-batch/pom.xml :

```
<!-- ===== -->
<!-- Build -->
<!-- ===== -->
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <configuration>
        <archive>
          <manifest>
            <mainClass>org.exemple.demo.batch.App</mainClass>
            <addClasspath>true</addClasspath>
            <!-- Le préfixe du chemin vers tous les éléments du classpath -->
            <!-- Les JAR des dépendances étant dans le même répertoire que le -->
            <!-- le JAR ticket-batch, on a pas de préfixe à ajouter -->
            <classpathPrefix></classpathPrefix>
          </manifest>
        </archive>
      </configuration>
    </plugin>
  </plugins>
</build>
```

- Assembler le tout dans une archive
  - On utilise le plugin apache.maven.plugins:maven-assembly-plugin:3.6.0
  - La définition des assemblages se fait via des fichiers xml dans le répertoire src/assembly. On crée donc le répertoire ticket/ticket-batch/src/assembly et le fichier ticket/ticket-batch/src/assembly/archive-deploy.xml :

```
mkdir ~/env/maven/maven-workspace/ticket/ticket-batch/src/assembly
touch ~/env/maven/maven-workspace/ticket/ticket-batch/src/assembly/archive-deploy.xml
```

- Dans /env/maven/maven-workspace/ticket/ticket-batch/src/assembly/archive-deploy.xml :

```
<assembly xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://maven.apache.org/ASSEMBLY/2.0.0"
  xsi:schemaLocation="http://maven.apache.org/ASSEMBLY/2.0.0 http://maven.ap
```

```
<!-- Identifiant de l'assemblage -->
<id>archive-deploy</id>
```

```
<!-- Les formats d'archive à générer -->
<formats>
  <format>tar.gz</format>
  <format>zip</format>
</formats>
```

```
<!-- lib : dépendances + JAR ticket-batch -->
<dependencySets>
  <dependencySet>
    <outputDirectory>lib</outputDirectory>
    <scope>runtime</scope>
  </dependencySet>
</dependencySets>
```

```
<fileSets>
  <!-- Scripts shell de lancement -->
  <fileSet>
    <directory>src/data/scripts</directory>
    <outputDirectory>bin</outputDirectory>
    <!-- Droits UNIX sur les fichiers (-rwx-rx-rx) -->
    <fileMode>0755</fileMode>
  </fileSet>
```

```
<!-- Fichiers de configuration -->
<fileSet>
  <directory>src/data/conf</directory>
  <outputDirectory>conf</outputDirectory>
</fileSet>
```



```
</fileSets>
</assembly>
```

- On ajoute le plugin `apache.maven.plugins:maven-assembly-plugin:3.6.0` dans la section `pluginManagement` du POM parent `ticket/pom.xml` :

```
<!-- ===== -->
<!-- Build -->
<!-- ===== -->
<build>
  <pluginManagement>
    <!-- Assemblage -->
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-assembly-plugin</artifactId>
        <version>3.6.0</version>
      </plugin>
    <...>
```

- Dans le POM du module pour les batchs `ticket/ticket-batch/pom.xml`, on configure le plugin `apache.maven.plugins:maven-assembly-plugin:3.6.0` :

```
<!-- ===== -->
<!-- Build -->
<!-- ===== -->
<build>
  <plugins>
    <!-- Assemblage -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-assembly-plugin</artifactId>
      <configuration>
        <!-- fichiers de description des assemblages que l'on veut réaliser -->
        <descriptors>
          <descriptor>src/assembly/archive-deploy.xml</descriptor>
        </descriptors>
      </configuration>
      <executions>
        <execution>
          <id>assembly-archive-deploy</id>
          <phase>package</phase>
          <goals>
            <goal>single</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  <...>
```

### C.1.16 Générer un site pour un projet

#### Généralités

- Par convention Maven, les sources servant à la génération du site du projet se trouvent dans le répertoire `src/site/`. On le crée :

```
mkdir ~/env/maven/maven-workspace/ticket/src/site
```

- Dans ce dossier, on édite un fichier `ticket/src/site/site.xml`, c'est le site descriptor ; il permet de définir la structure du site :

```
<site xmlns="http://maven.apache.org/SITE/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/SITE/4.0.0 https://maven.apache.org/xsd/site-4.0.0.xsd">

  <!-- Affichage de la date et de la version à droite dans le bandeau du haut -->
  <publishDate position="right"/>
  <version position="right"/>

  <!-- Utilisation du template de site fluido -->
  <skin>
    <groupId>org.apache.maven.skis</groupId>
    <artifactId>maven-fluido-skin</artifactId>
```

```

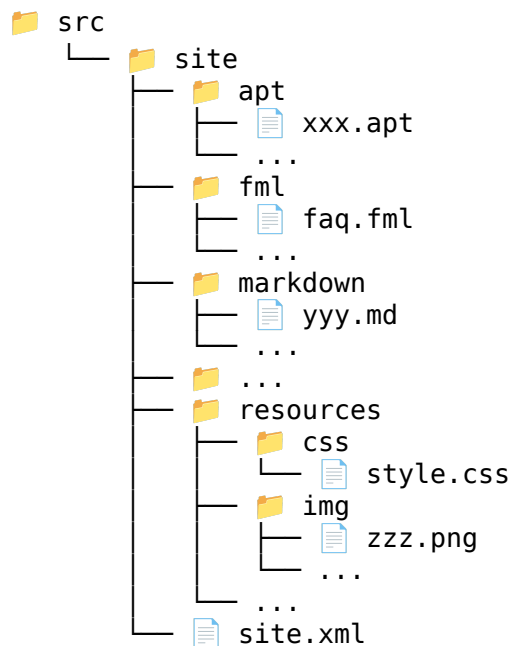
<version>1.6</version>
</skin>

<body>
  <!-- Ajout d'un fil d'ariane -->
  <breadcrumbs>
    <item name="Accueil" href="index.html"/>
  </breadcrumbs>

  <!-- ===== Menus ===== -->
  <!-- Ajout d'un menu vers le projet parent -->
  <menu ref="parent" inherit="top"/>
  <!-- Ajout d'un menu vers les différents modules du projet -->
  <menu ref="modules" inherit="top"/>
  <!-- ===== Menus ===== -->
  <!-- Ajout d'un menu vers la documentation -->
  <menu name="Documentation">
    <!-- Entrée de menu vers la page Architecture -->
    <item name="Architecture" href="architecture.html"/>
  </menu>
</body>
</site>

```

- Le répertoire ticket/src/site/site.xml devra avoir la structure suivante :



- Le fichier src/site/site.xml est le site descriptor. Il permet de définir la structure du site.
- Les répertoires apt, markdown, fml... hébergent les fichiers source des pages du site qui seront converties en HTML par Maven. Ces répertoires accueillent des fichiers de leur format respectif :
  - apt : Format Almost Plain Text
  - fml : Format FAQ Markup Language
  - markdown : Format Markdown
- Le répertoire resources pour les autres ressources :
  - css/style.css : fichier CSS permettant d'ajuster le style par défaut
  - ...
- Dans le POM du projet parent ticket/pom.xml, on définit l'url de déploiement du site :

```

<!-- ===== -->
<!-- DistributionManagement -->
<!-- ===== -->
<distributionManagement>
  <site>
    <id>site-projet</id>
    <url>scp://localhost/tmp/</url>
  </site>
</distributionManagement>

```

- On utilise org.apache.maven.plugins:maven-site-plugin:4.0.0-M12
- Dans ticket/pom.xml, on déclare la configuration de ce plugin :

```

<!-- ===== -->
<!-- Build -->
<!-- ===== -->
<build>
  <pluginManagement>
    <plugins>
      <!--
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-project-info-reports-plugin</artifactId>
        <version>3.5.0</version>
      </plugin>
      -->
      <!-- Génération du site -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-site-plugin</artifactId>
        <version>4.0.0-M12</version>
        <dependencies>
          <dependency>
            <groupId>org.apache.maven.doxia</groupId>
            <artifactId>doxia-module-markdown</artifactId>
            <version>2.0.0-M8</version><!-- ne marche pas -->
          </dependency>
          <!--
          <dependency>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-project-info-reports-plugin</artifactId>
            <version>3.5.0</version>
            <type>maven-plugin</type>
          </dependency>
          -->
        </dependencies>
      <configuration>
        <locales>default,fr</locales>
      </configuration>
    </plugin>

```

Problème rencontré : Dans la section suivante, ma première tentative de faire générer par Maven le fichier `ticket/target/site/architecture.html` à partir du fichier `ticket/src/site/markdown/architecture.md` a échoué. Pour pallier cet échec, j'ai dû ajouter la partie commentée du morceau de code précédent. Au vu des retours de Maven lors de l'exécution de l'instruction « `mvn clean package site site:stage -Ptarget-dev` », je pense que cet échec était causé par un conflit de version. Ainsi, quand j'ai ajouté la partie commentée du code précédent, j'ai précisé une version du plugin `org.apache.maven.plugins:maven-project-info-reports-plugin` qui ne rentrait plus en conflit avec les plugins `org.apache.maven.plugins:maven-site-plugin` et `org.apache.maven.doxia:doxia-module-markdown`. La version que Maven choisissait par défaut sans cette précision, elle, rentrait probablement en conflit avec ces deux plugins. Cependant, lors de la première génération de site avec cette précision, Maven a téléchargé la version 3.5.0 de `maven-project-info-reports-plugin` et, même en recommandant cette précision de version, Maven semble maintenant choisir cette nouvelle version puisque le fichier markdown continue d'être convertie en fichier html sans cette précision de version.

- Pour générer le site, on utilise le build lifecycle `site`. Comme il s'agit d'un projet multi-modules, la génération du site est un peu plus complexe que pour un projet simple. En effet :
  - Il faut que Maven génère un site pour le projet parent et chaque module. Ceci correspond au goal `site` du plugin `site`.
  - Ensuite il faut qu'il agrège tous ces sites en un seul. Cette étape est réalisée automatiquement par la phase `site-deploy`.
  - Cependant, on ne veut pas dans cet exemple déployer le site mais seulement avoir une copie en local pour pouvoir soit le tester avant de le déployer soit simplement avoir un site généré. Pour cela on utilise le goal `stage` du plugin

site.

```
mvn package site site:stage
```

On enchaîne ainsi trois exécutions. La première est un package classique. La deuxième génère ensuite le site. Et la troisième exécute le goal stage du plugin site.

Remarque : Il n'est donc pas nécessaire d'activer le plugin maven-site-plugin ni dans le projet parent ni dans aucun des sous-projets. Maven se charge en effet de cette activation lorsqu'on lui soumet l'instruction « `mvn clean package site site:stage -Ptarget-dev` »

- Le site se trouve ensuite dans `ticket/target/site/fr/`.

### Ajout d'une page de documentation à partir du format markdown

- On crée une page de documentation expliquant comment est organisé le projet.
- Dans `ticket/scr/site/markdown/architecture.md` :

```
## Architecture du projet
```

```
### Généralités
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit,  
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Une image :

```
![[Dépendances entre les modules](img/dependances_modules.png)]
```

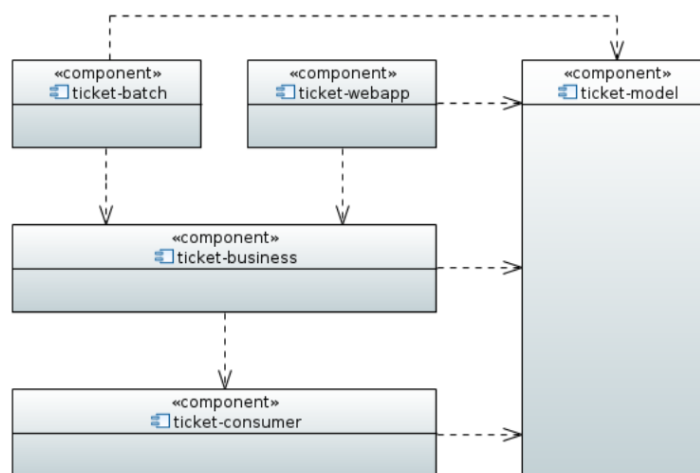
```
### L'application web
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit,  
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

```
### Les batches
```

Lorem ipsum dolor sit amet, consectetur adipisicing elit,  
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

- On copie l'image suivante dans `ticket/scr/site/resources/img/dependances_modules.png` :



Dépendances entre les modules

- On ajoute une entrée de menu dans `ticket/scr/site/site.xml` qui pointe sur la page de documentation susmentionnée :

```
[...]
<body>
  <!-- Ajout d'un fil d'ariane -->
  <breadcrumbs>
```

```

    <item name="Accueil" href="index.html"/>
</breadcrumbs>

<!-- ===== Menus ===== -->
<!-- Ajout d'un menu vers le projet parent -->
<menu ref="parent" inherit="top"/>
<!-- Ajout d'un menu vers les différents modules du projet -->
<menu ref="modules" inherit="top"/>

<!-- Ajout d'un menu vers la documentation -->
<menu name="Documentation">
    <!-- Entrée de menu vers la page Architecture -->
    <item name="Architecture" href="architecture.html"/>
</menu>
</body>
</project>

```

Maven va convertir tous les markdown/\*.md en .html

- Il ne reste qu'à régénérer le site :

```
mvn clean package site site:stage
```

### Ajout d'une FAQ à partir du format fml

- On édite le fichier ticket/src/site/fml/faq.fml :

```

<?xml version="1.0" encoding="UTF-8"?>
<faqs xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://maven.apache.org/fml/1.0"
    xsi:schemaLocation="http://maven.apache.org/FML/1.0.1 http://maven.apache.org/xsd/fml-1.0.xsd">

    <title>Foire Aux Questions</title>
    <toplink>false</toplink>

    <part id="general">
        <title>Général</title>
        <faq id="why-x">
            <question>
                Pourquoi ... ?
            </question>
            <answer>
                <p>
                    Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquam aperiam
                    consequuntur delectus deleniti dolores dolorum ex excepturi explicabo
                    modi
                    nemo sint, sit veniam voluptate.
                </p>
                <p>...</p>
            </answer>
        </faq>

        <faq id="how-x">
            <question>
                Comment ... ?
            </question>
            <answer>
                <p>...</p>
            </answer>
        </faq>
    </part>

    <part id="install">
        <title>Installation</title>
        <faq id="how-install">
            <question>
                Comment installer ... ?
            </question>

```

```

        </question>
        <answer>
            <p>
                Lorem ipsum dolor sit amet, consectetur adipisicing elit. Aliquam aperiam
                consequuntur delectus deleniti dolores dolorum ex excepturi explicabo
                modi
                nemo sint, sit veniam voluptate.
            </p>
            <source>apt-get install xxx</source>
        </answer>
    </faq>
</part>
</faqs>

```

- On ajoute une entrée de menu pour la FAQ dans ticket/src/site/site.xml :

```

[... ]
<body>
    <!-- Ajout d'un fil d'ariane -->
    <breadcrumbs>
        <item name="Accueil" href="index.html"/>
    </breadcrumbs>

    <!-- ===== Menus ===== -->
    <!-- Ajout d'un menu vers le projet parent -->
    <menu ref="parent" inherit="top"/>
    <!-- Ajout d'un menu vers les différents modules du projet -->
    <menu ref="modules" inherit="top"/>

    <!-- Ajout d'un menu vers la documentation -->
    <menu name="Documentation">
        <!-- Entrée de menu vers la page Architecture -->
        <item name="Architecture" href="architecture.html"/>
        <!-- Entrée de menu vers la page FAQ -->
        <item name="FAQ" href="faq.html" />
    </menu>
</body>
</project>

```

- Il ne reste qu'à régénérer le site :

```
mvn clean package site site:stage
```

## Générer des rapports

- Générer des rapports :
  - Maven permet de générer, de base, un certain nombre de rapports sur le projet :
    - Projet :
      - résumé du projet (nom, version, description, organisation...)
      - liste des modules du projet
      - membres du projet (contributeur, développeurs...)
      - licence du projet (section <licenses>)
      - item gestion de la distribution du projet (section <distributionManagement>)
      - listes de diffusion (section <mailingLists>)
      - dépôt des sources du projet (section <scm>)
      - intégration continue (section <ciManagement>)
      - gestion des anomalies (section <issueManagement>)
    - Plugins :
      - gestion des plugins (section <pluginManagement>)
      - liste des plugins utilisés dans le projet/module
    - Dépendances :
      - gestion des dépendances (section <dependencyManagement>)

- liste des dépendances utilisées dans le projet/module
- convergence des dépendances entre les différents modules du projet
- On ajoute un menu dans ticket/src/site/site.xml afin de pouvoir accéder aux rapports :

```
[...]
<body>
  <!-- Ajout d'un fil d'ariane -->
  <breadcrumbs>
    <item name="Accueil" href="index.html"/>
  </breadcrumbs>

  <!-- ===== Menus ===== -->
  <!-- Ajout d'un menu vers le projet parent -->
  <menu ref="parent" inherit="top"/>
  <!-- Ajout d'un menu vers les différents modules du projet -->
  <menu ref="modules" inherit="top"/>

  <!-- Ajout d'un menu vers la documentation -->
  <menu name="Documentation">
    <!-- Entrée de menu vers la page Architecture -->
    <item name="Architecture" href="architecture.html"/>
    <!-- Entrée de menu vers la page FAQ -->
    <item name="FAQ" href="faq.html" />
  </menu>

  <!-- Ajout d'un menu vers les différents rapport -->
  <menu ref="reports" inherit="top"/>
</body>
</project>
```

- On ajoute le plugin de génération des rapports dans le POM parent ticket/pom.xml :

```
<!-- ===== -->
<!-- Gestion des rapports -->
<!-- ===== -->
<reporting>
  <plugins>
    <!-- ===== Rapport d'information général sur le projet ===== -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-project-info-reports-plugin</artifactId>
      <version>3.5.0</version>
    </plugin>
  </plugins>
</reporting>
```

- Il ne reste qu'à regénérer le site :

```
mvn clean package site site:stage
```

- Sélectionner certains rapports seulement :

- Comme vu précédemment, les plugins de rapport comme org.apache.maven.plugins:maven-project-info-reports-plugin:3.5.0 sont ajoutés au sein des balises <reporting> et non au sein des balises <pluginManagement> elles mêmes au seins des balises <build>.
- Il existe d'autres plugins permettant de créer des rapports.
- Chaque rapport est fourni par un goal d'un de ces plugins.
- Par défaut, quand on ajoute un plugin de rapport, tous ses goals sont exécutés et donc tous ses rapports sont générés.
- On peut n'ajouter que certains goals en ajoutant une section <reportSets> comme ci-dessous :

```
<!-- ===== -->
<!-- Gestion des rapports -->
<!-- ===== -->
<reporting>
  <plugins>
    <!-- ===== Rapport d'information général sur le projet ===== -->
```

```

<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-project-info-reports-plugin</artifactId>
  <version>3.5.0</version>
  <reportSets>
    <reportSet>
      <reports>
        <report>index</report>
        <report>summary</report>
        <report>plugins</report>
      </reports>
    </reportSet>
  </reportSets>
</plugin>
</plugins>
</reporting>

```

- Ajouter d'autres rapports :

- On peut ajouter d'autres rapports en ajoutant les plugins de rapport correspondant.
- Par exemple, l'extrait suivant ajoute un rapport sur les tests, en n'ajoutant que le goal report-only du plugin de rapport org.apache.maven.plugins:maven-surefire-report-plugin:3.2.2 :

```

<!-- ===== -->
<!-- Gestion des rapports -->
<!-- ===== -->
<reporting>
  <plugins>
    [...]
    <!-- ===== Rapport sur les tests ===== -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-report-plugin</artifactId>
      <version>3.2.2</version>
      <configuration>
        <linkXRef>false</linkXRef>
      </configuration>
      <reportSets>
        <reportSet>
          <reports>
            <report>report</report>
          </reports>
        </reportSet>
      </reportSets>
    </plugin>
  </plugins>
</reporting>

```

- Agréger des rapports :

- Dans le POM du projet parent, d'une part on définit un ensemble de rapports non héréditaire que l'on configure de façon à ce qu'il génère une aggrégation de rapports et, d'autre part, on définit un ensemble de rapports héréditaires que l'on configure de façon à ce qu'il génère un unique rapport par module :

```

<!-- ===== Rapport sur les tests ===== -->
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-surefire-report-plugin</artifactId>
  <version>3.2.2</version>
  <configuration>
    <linkXRef>false</linkXRef>
  </configuration>
  <reportSets>
    <reportSet>
      <id>aggregate</id>
      <reports>
        <report>report</report>

```



```

        </reports>
        <inherited>false</inherited>
        <configuration>
            <aggregate>true</aggregate>
        </configuration>
    </reportSet>
    <reportSet>
        <id>module</id>
        <inherited>true</inherited>
        <reports>
            <report>report</report>
        </reports>
        <configuration>
            <aggregate>false</aggregate>
        </configuration>
    </reportSet>
</reportSets>
</plugin>

```

- Ajout d'un rapport sur la qualité du code :
  - On utilise org.apache.maven.plugins:maven-checkstyle-plugin:3.3.1.
  - Dans le POM parent ticket/pom.xml :

```

<!-- ===== -->
<!-- Gestion des rapports -->
<!-- ===== -->
<reporting>
    <plugins>
        [...]
        <!-- ===== Rapport sur la qualité du code ===== -->
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-checkstyle-plugin</artifactId>
            <version>3.3.1</version>
            <configuration>
                <configLocation>src/build/checkstyle.xml</configLocation>
                <linkXRef>false</linkXRef>
            </configuration>
            <reportSets>
                <reportSet>
                    <id>checkstyle</id>
                    <inherited>true</inherited>
                    <reports>
                        <report>checkstyle</report>
                    </reports>
                </reportSet>
                <reportSet>
                    <id>checkstyle-aggregate</id>
                    <reports>
                        <report>checkstyle-aggregate</report>
                    </reports>
                </reportSet>
            </reportSets>
        </plugin>
    </plugins>
</reporting>

```

#### Générer la documentation de Java

- Dans le POM parent ticket/pom.xml :

```

<!-- ===== -->
<!-- Gestion des rapports -->
<!-- ===== -->
<reporting>

```

```
<plugins>
[... ]
<!-- ===== Génération de la Javadoc ===== -->
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-javadoc-plugin</artifactId>
  <version>3.6.3</version>
  <configuration>
    <quiet>>true</quiet>
    <locale>fr</locale>
  </configuration>
  <reportSets>
    <!-- reportSet exécuté dans tous les modules -->
    <reportSet>
      <reports>
        <report>javadoc</report>
      </reports>
    </reportSet>
    <!-- reportSet d'agrégation des rapports des sous-modules -->
    <reportSet>
      <id>aggregate</id>
      <inherited>>false</inherited>
      <reports>
        <report>aggregate</report>
      </reports>
    </reportSet>
  </reportSets>
</plugin>
</plugins>
</reporting>
```

## D Arborescence d'un projet Maven

Répertoire	Contenu
/src	les sources du projet (répertoire qui doit être ajouté dans le gestionnaire de sources)
/src/main	les fichiers sources principaux
/src/main/java	le code source (sera compilé dans /target/classes)
/src/main/resources	les fichiers de ressources (fichiers de configuration, images, ...). Le contenu de ce répertoire est copié dans target/classes pour être inclus dans l'artéfact généré
/src/main/webapp	les fichiers de la webapp
/src/test	les fichiers pour les tests
/src/test/java	le code source des tests (sera compilé dans /target/test-classes)
/src/test/resources	les fichiers de ressources pour les tests
/target	les fichiers générés pour les artéfacts et les tests (ce répertoire ne doit pas être inclus dans le gestionnaire de sources)
/target/classes	les classes compilées
/target/test-classes	les classes compilées des tests unitaires
/target/site	site web contenant les rapports générés et des informations sur le projet
/pom.xml	le fichier POM de description du projet

## E Un exemple minimal de serveur Back-end derrière Apache 2 avec la bibliothèque Python websocket-server

création d'un environnement python virtuel isolé myenv pour y installer la bibliothèque python websocket-server et exécution du script /home/workaholic/site\_test2/websocket/websocket-server.py qui fait usage de cette bibliothèque :

```
sudo apt-get install python3-venv
python3 -m venv myenv
source myenv/bin/activate
pip install websocket-server
python /home/workaholic/site_test2/websocket/websocket-server.py
```

Plus tard pour désactiver l'environnement virtuel isolé :

deactivate

À droite index.html contenant un div avec onclick="toggleColor()" et du code javascript créant une websocket se connectant à 127.0.0.1:12345, affectant les messages en provenance de 127.0.0.1:12345 à la propriété backgroundColor du div (pour lui changer sa couleur) et implémentant toggleColor() de façon à envoyer le message "toggle" via la websocket. À gauche le serveur websocket écoutant 127.0.0.1:12345

```
#!/usr/bin/env python3

# site_test2/websocket/websocket-server.py

from websocket_server import WebsocketServer

def client_left(client, server):
    print("Client(%d) disconnected" % client['id'])

def new_client(client, server):
    print("New client(%d) connected" % client['id'])
    server.send_message_to_all("red")

def message_received(client, server, message):
    if message == "toggle":
        new_color = "blue" if server.last_color == "red" else "red"
        server.last_color = new_color
        server.send_message_to_all(new_color)

server = WebsocketServer(port=12345, host="127.0.0.1")
server.set_fn_new_client(new_client)
server.set_fn_client_left(client_left)
server.set_fn_message_received(message_received)
server.last_color = "red"
server.run_forever()
```

```
<!DOCTYPE html>

<!-- site_test2/index.html -->

<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>WebSocket Color Toggle</title>
</head>
<body>
  <div
    id="colorBox"
    style="width: 100px; height: 100px;"
    onclick="toggleColor()">
  </div>

  <script>
    let ws = new WebSocket("ws://127.0.0.1:12345/");
    let colorBox = document.getElementById("colorBox");

    ws.onopen = function(event) {
      console.log("Connected to WebSocket server.");
    };

    ws.onmessage = function(event) {
      colorBox.style.backgroundColor = event.data;
    };

    function toggleColor() {
      ws.send("toggle");
    }
  </script>
</body>
</html>
```

## F Exemple de conversion manuelle d'un projet Jakarta EE existant en un projet Maven

Le projet `/env/eclipse/eclipse-workspace/test_DAO` a été développé avec Eclipse et sans Maven dans la section « Étude du cours « Développez des sites web avec Java EE » de Mathieu Nebra sur [openclassrooms.com](https://openclassrooms.com) pour développer une application web proposant les services de lecture et d'écriture dans une table d'une base de données MySQL tout en respectant MVC et DAO ».

Les commandes suivantes génèrent le projet Maven packagé correspondant `/env/maven/maven-workspace/test_DAO` et donc en particulier son fichier WAR `/env/maven/maven-workspace/test_DAO/target/test_DAO.war` que l'on va utiliser dans la section « Exemple de déploiement manuel d'une l'application web sur un serveur Tomcat » :

```
cp -r ~/env/eclipse/eclipse-workspace/test_DAO ~/env/maven/maven-workspace
cd ~/env/maven/maven-workspace/test_DAO/
cat > pom.xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.octest</groupId>
  <artifactId>test_DAO</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>war</packaging>
  <name>Test DAO Application</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>
    <!-- Dépendance pour Servlet API -->
    <dependency>
      <groupId>jakarta.servlet</groupId>
      <artifactId>jakarta.servlet-api</artifactId>
      <version>6.1.0-M1</version>
      <scope>provided</scope>
    </dependency>
  </dependencies>

  <build>
    <finalName>test_DAO</finalName>
    <plugins>
      <!-- Plugin pour gérer les projets web -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.3.1</version>
      </plugin>
    </plugins>
  </build>
</project>
EOF
mvn clean package
```

## G Exemple de déploiement manuel d'une l'application web sur un serveur Tomcat

Dans cette section, on déploie manuellement l'application web développé dans la section « Étude du cours « Développez des sites web avec Java EE » de Mathieu Nebra sur [openclassrooms.com](https://openclassrooms.com) pour développer une application web proposant les services de lecture et d'écriture dans une table d'une base de données MySQL tout en respectant MVC et DAO » dans le répertoire `env/eclipse/eclipse-workspace/test_DAO`.

Ce déploiement se fait sur le serveur Tomcat dont les sources sont situés dans le répertoire `env/tomcat/apache-tomcat-10.1.16/`.

Pour ce faire, on a besoin du fichier WAR (Web ARchive) `env/maven/maven-workspace/test_DAO/target/test_DAO.war` correspondant à ce projet.

Ce fichier WAR a été obtenu à l'issue de la section « Exemple de conversion manuelle d'un projet Jakarta EE existant en un projet Maven ».

Les variables d'environnement concernées relatives à Java et Tomcat sont définies à la fin du script `/.bashrc` via les lignes suivantes :

```
# developpement directory ~/env/ configuration
# Java
export JAVA_HOME="/home/workaholic/env/java/jdk-21.0.1"
export MAVEN_HOME="/home/workaholic/env/maven/apache-maven-3.9.5"
export PATH="$JAVA_HOME/bin:$MAVEN_HOME/bin:$PATH"
# Tomcat
export CATALINA_HOME="/home/workaholic/env/tomcat/current"
```

Le déploiement en question se fait via les commandes suivantes :

```
~/env/tomcat/current/bin/startup.sh
cp ~/env/maven/maven-workspace/test_DAO/target/test_DAO.war
→ ~/env/tomcat/apache-tomcat-10.1.16/webapps/

# Pour arrêter le serveur
~/env/tomcat/current/bin/shutdown.sh
```

L'application web répond via l'url : `http://localhost:8080/test_DAO/guidb`

# H Sources

SSS

# I Fichiers de sortie

000