

Rapport de Stage Insy2S

4 FEVRIER – 4 MAI

Afpa CDI
Josse Moncheaux



Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage, qui m'ont accompagné et aidé tout au long de ces 3 mois.

Je tiens à remercier vivement mon maitre de stage, Monsieur TORCHE Nezar, Directeur de Insy2S , pour son accueil, le temps passé ensemble et le partage de son expertise au quotidien.

Grâce aussi à sa confiance j'ai pu m'accomplir dans mes missions et ainsi développé mes connaissances.

Je remercie également toutes les personnes de l'équipe de Insy2S pour leur accueil et leur esprit d'équipe.

Je les remercie aussi pour m'avoir guidés sur la rédaction, les rendus finaux, ainsi que pour leurs écoutent et leurs soutiens lorsque j'avais des questions plus techniques.

2. Sommaire

| | | |
|----|---|------------------------------|
| 1. | Remerciements..... | Error! Bookmark not defined. |
| 2. | Sommaire. | 3 |
| 3. | Entreprise. | 4 |
| 4. | Le Client. | 5 |
| | 4.1 Travail réalisé. | 6 |
| 5. | Outils et Technologies employée | 8 |
| 6 | Architecture de l'application | 34 |
| | 6.1 Architecture N-tiers | 34 |
| | 6.2 MVC | 36 |
| | 6.3 Conception de La Base De Données..... | 49 |
| | 6.3.1 les outils | 49 |
| | 6.3.2 MCD | 49 |
| | 6.3.3 MLD | 51 |
| | 6.3.4 Diagramme de classe | 52 |
| | CONCLUSION | 54 |

3.Entreprise

Insy2S est une entreprise de services en ingénierie digitale, elle a été créée en 2013 par Monsieur TORCHE Nezar, elle apporte des solutions à ses clients et partenaires en France comme à l'international.

Acteur majeur de la gestion applicative (Application Management) sur les marchés européen (INSY2S) et au Moyen-Orient (Watad Alhijaz), Insy2S propose à ses clients des solutions visant de simplifier et améliorer le maintien de leur système et d'accompagner les entreprises dans l'adaptation de leur patrimoine applicatif à la transformation numérique.

Avec les nouvelles ruptures technologiques, la performance « business » des entreprises dépend de plus en plus de leurs systèmes d'informations.

Les applications IT se doivent donc d'être agiles et performantes afin d'accompagner et de soutenir la transformation numérique.

4. Travail pour le client

L'association Olympique Lille Sud (OLS) est une Association Multisports régit sous la loi 1901, dont le siège social se situe au 84 rue du Faubourg des Postes à Lille, en plein cœur du quartier de Lille-Sud, servant même de petite frontière existant, entre les sous quartiers de celui.



Elle a pour objectifs principaux de proposer aux habitants du quartier de Lille-Sud, un éventail d'activités sportives non existantes dans le quartier, avec une tarification abordable à toutes les tranches sociales, et de favoriser le développement du sport au féminin.

Ces membres fondateurs sont **Monsieur Jean Claude Sabre**, qui à cette époque était Président du Conseil de Quartier de Lille-Sud, et **Monsieur Joël Comblez**, chef de Projet au Développement Social Urbain (D.S.U) de Lille-Sud en 1995.

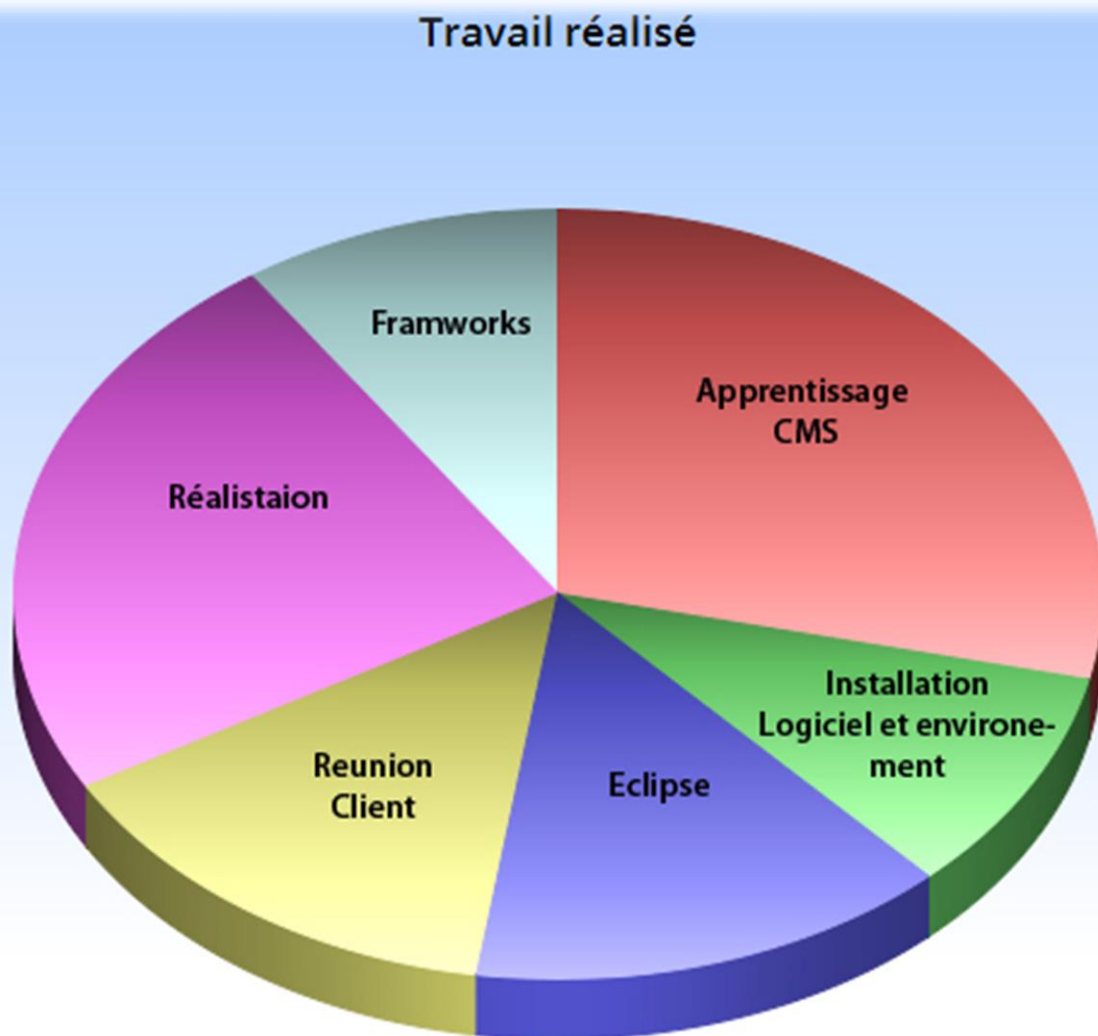
Durant ces 3 mois de stage j'ai eu pour demande de la part du client «OLS » (**Olympique Lille Sud**) de développer une application web permettant de gérer ces adhérents ainsi que leurs préinscriptions par ce même client.

Etant une association sportive il était primordial de pouvoir, en cas d'annulation d'une activité, prévenir l'adhérent. Il m'a était donc demandé aussi d'ajouter un outil permettant de contacter automatiquement ces clients pour cette situation.

Vers la fin de mon stage il m'a était aussi demandé de créer un moyen de prévenir les clients pour des évènements sportifs et aussi de gérer et visualiser chaque tâches liés à leurs employés.

4.1 Travail réalisé

Pour avoir une idée plus précise de mon travail tout au long du stage, voici un diagramme qui récapitule le déroulé des missions qui m'ont été demandées.



1. Installation logiciels et environnements :

En tout début de stage c'est une phase de découverte et d'installation.
Durant plusieurs jours, en local, j'ai installé et configuré de nombreux modules.

2. Eclipse :

Deuxièmement, j'ai commencé la conception du « *moteur* » de l'application via le logiciel **Eclipse** et de la base donnée via **PostgreSQL**.

3. Framework :

Par la suite, j'ai installé et utilisé de nombreux Framework sur **Eclipse** pour améliorer « *le moteur* » de l'application web.

4. Réalisation :

Pour finir, j'ai effectué les corrections de bug et le déploiement de l'application web.

5. Réunion Client :

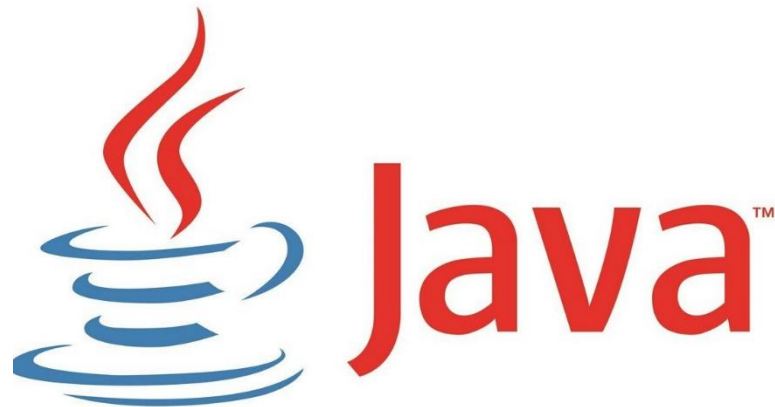
Travailler en **Agilité**, toutes les deux semaines, j'avais une réunion avec le client pour discuter de l'évolution de l'application et voir les changements à effectuer.

6. Apprentissage/Conception du CMS :

Faisant la découverte du **CMS** J'ai décidé de dédier une partie dans les outils de ce rapport et donc de mettre la conception **CMS** en dernier sur cette légende.

5. Outils et Technologies employées

1. Langage Java :



Java est un langage de programmation orienté objet créé par James Gosling et Patrick Naughton, employés de Sun Microsystems, avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au SunWorld.

2. Eclipse :

Eclipse est un logiciel Utilisant le langage **Java**, composé d'une plateforme de base, et d'un très grand nombre de *plugins* aussi nommé *module d'extension*, *module externe*, *greffon*, *plugiciel*, ainsi que *add-in* ou *add-on* en France. Ces *plugins* apportent des fonctionnalités à la plateforme de base, dans de très nombreux domaines, y compris en dehors des technologies Java.



Jusqu'à présent, je vous ai présenté la version **Eclipse Classique**. Maintenant nous verrons la version **Eclipse pour Java EE**. De plus, cette version inclut des éditeurs html et jsp qui facilitent le travail. Le nom de cette version est **Eclipse IDE**

Photon.



Initié par **IBM** puis ouvert à la communauté **Open Source**, l'**environnement de développement intégré (IDE) Eclipse** pour développeurs, Java présente l'énorme avantage de disposer d'un nombre impressionnant de plugins l'enrichissant dans tous les domaines de la programmation.

Dorénavant, avec la version d'**Eclipse Photon**, un *Marketplace Client* propose de faciliter la vie des développeurs dans leurs recherches de plugins. Une interface dédiée rend leur intégration bien plus rapide.

Fonctionnant en Java et avec la bibliothèque graphique SWT d'IBM, cet **IDE** "tourne" indépendamment du système d'exploitation, pourvu qu'une JVM soit installée.

Des projets tels que **Lotus Notes** ou **Symphony** ont été développés grâce à cet atelier. L'IDE open source de référence supporte les projets EGit qui retrace tout l'historique des modifications effectuées sur le code. Et si Java fut l'un des premiers langages à avoir été utilisé, il a depuis été enrichi par le C/C++, l'Ada, le Python, le Perl, le Ruby, le Cobol, le Pascal, le PHP, l'XML et HTML, l'Action Script, ColdFusion, etc.

3. PostgreSQL

PostgreSQL est un **Système de Gestion de Bases de Données Relationnel-Objet (SGBDRO)** libre et gratuit. Son travail consiste à gérer les accès concurrents aux données. C'est-à-dire de gérer les transactions (Atomicité et isolation), la Consistance de ces données et d'assurer leur Durabilité dans un contexte où n clients peuvent modifier ces données en même temps. Il s'agit des fameux principes ACID communs aux SGBDR.



PostgreSQL est largement reconnu pour son comportement stable, proche de **Oracle**, mais aussi pour ses possibilités de programmation étendues, directement dans le moteur de la base de données, via **PL/pgSQL**. Le traitement interne des données peut aussi être couplé à d'autres modules externes compilés dans d'autres langages.

Qu'elles sont ces points forts ?

1. Déploiement illimité

Vous pouvez déployer **PostgreSQL** sur autant de serveurs avec autant de **CPU** que vous le souhaitez. Non seulement le coût d'investissement est nul, mais il n'y a pas de maintenance annuelle à payer ! Sur le long terme, l'économie est très importante.

2. Excellent support

Le support assuré par la communauté **PostgreSQL** est excellent et gratuit. De plus, de nombreuses **SSL** peuvent vous offrir un contrat de support formel sur mesure.

2. Economies significatives sur les coûts de personnel

PostgreSQL nécessite beaucoup moins de maintenance et de paramétrage que les grandes bases de données commerciales, tout en proposant la plupart de leurs fonctionnalités, et surtout la fiabilité et les performances que l'on attend d'un tel produit.

Contrairement à beaucoup de bases de données commerciales, **PostgreSQL** ne nécessite pas de suivre plusieurs semaines de formation, ni d'avoir un administrateur de bases de données à plein temps. C'est un avantage majeur pour beaucoup de **PME**.

3. Fiabilité et stabilité légendaires

Il est très courant que des sociétés rapportent que **PostgreSQL** n'a jamais crashé, même pendant des années. Pas une seule fois. Tous les **SGBDR** ne peuvent pas en dire autant. En particulier, **PostgreSQL** ne craint pas les coupures électriques.

4. Conçu pour une grande capacité

De par sa conception, **PostgreSQL** ne craint pas les bases de données de grande taille ou ayant un grand nombre d'utilisateurs simultanés.

Plusieurs organisations l'utilisent pour des bases de données de plus d'un Teraoctet.

Le système mondial d'enregistrement des noms de domaine en « .org » est géré avec une base de données **PostgreSQL** par **Afilias**.

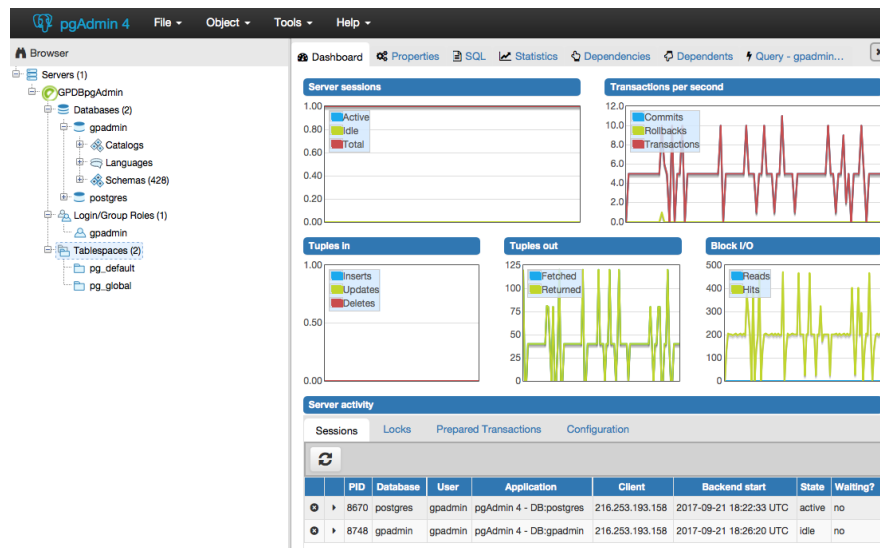
5. Outils graphiques de modélisation et d'administration

Plusieurs outils graphiques existent pour administrer les bases de données. Citons **pgadminIII**, **pgaccess**, **phppgadmin**, ect...

6. Très bonne compatibilité SQL

Comme beaucoup d'outils *Open Source*, **PostgreSQL** met un point d'honneur à suivre les normes, et en particulier les normes SQL 92 et 99. C'est un gage de pérennité et de portabilité.

L'Outil d'interface utilisé : pgAdmin.



pgAdmin est un outil d'administration graphique pour **PostgreSQL** distribué selon les termes de la licence **PostgreSQL**.

4. Apache Tomcat

Apache Tomcat est un conteneur web libre de servlets et JSP Java EE. Il implémente les spécifications des servlets et des JSP du **JavaCommunity Process**, est paramétrable par des fichiers XML et des



propriétés, et inclut des outils pour la configuration et la gestion. Il comporte également un serveur HTTP.

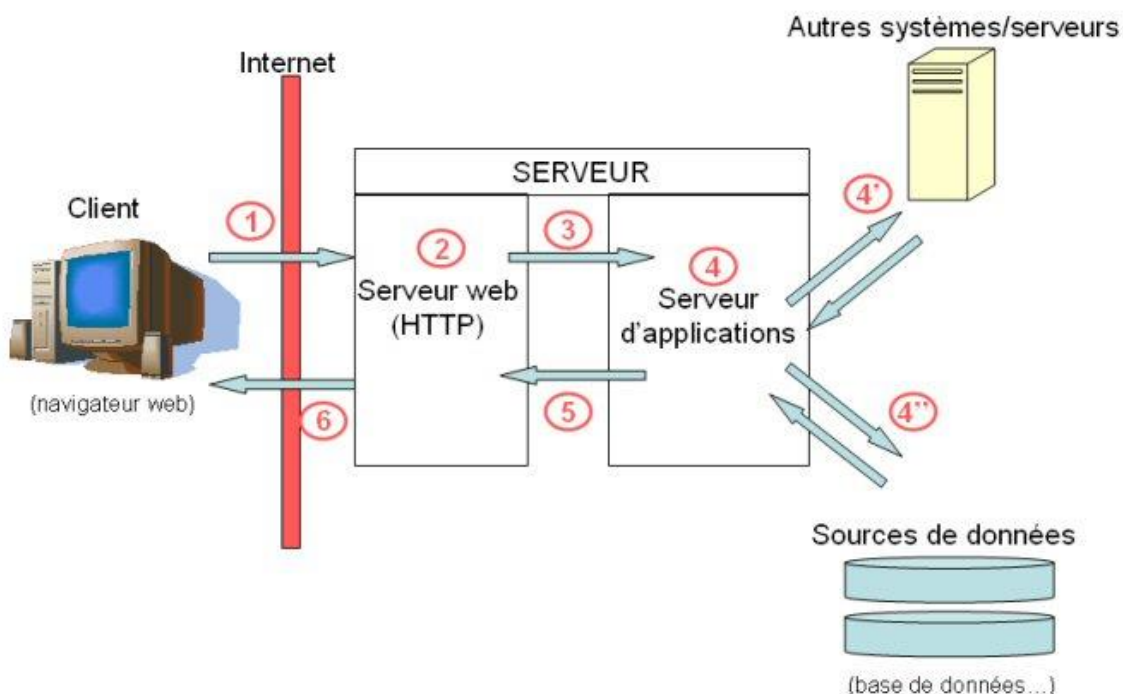
Le logiciel **Apache Tomcat** est développé dans un environnement ouvert, participatif et est distribué sous la [licence Apache version 2](#) . Le projet **Apache Tomcat** se veut une collaboration entre les meilleurs développeurs du monde entier.

Celui-ci prend en charge de nombreuses applications Web critiques à grande échelle et de grande envergure dans un large éventail de secteurs et d'organisations. Certains de ces utilisateurs et leurs histoires sont répertoriées sur la page wiki de [PoweredBy](#) .

3.1 Centrons-nous sur Tomcat.

Tomcat est un serveur d'applications Java. Servant principalement pour les [applications web](#). Elle permet de générer une réponse HTML à une requête après avoir effectué un certain nombre d'opérations (*connexion à une base de données, à un annuaire **LDAP** et ect...*). Pour le client (*un navigateur web en général*), il n'y a pas de différence avec une page web statique : il reçoit toujours du **HTML**, seul langage qu'il comprend. Seule la manière dont la réponse est formée côté serveur change.

Les requêtes, pour le client, ne diffèrent pas non plus. Qu'il souhaite accéder à une ressource statique ou à une application web, il utilise toujours une URL au même format (**standard HTTP**). C'est donc côté serveur que la distinction doit s'opérer. Le schéma suivant montre le déroulement classique d'une requête vers un serveur d'applications :



1) Le client émet une requête (i.e. appelle une URL) pour demander une ressource au serveur.

Exemple : **http://leserveur.com/welcome**. Il ne sait pas si la réponse qui va lui parvenir est statique (page HTML simple) ou dynamique (générée par une application web). Dans notre cas, il s'agit d'une application répondant à l'adresse **welcome** sur le serveur **leserveur.com**.

2) Côté serveur, c'est le serveur web (exemple : Apache) qui traite les requêtes HTTP entrantes. Il traite donc toutes les requêtes, qu'elles demandent une ressource statique ou dynamique. Seulement, un serveur HTTP ne sait répondre qu'aux requêtes visant des ressources statiques. Il ne peut que renvoyer des pages HTML, des images,... existantes.

3) Ainsi, si le serveur HTTP s'aperçoit que la requête reçue est destinée au serveur d'applications, il la lui transmet. Les deux serveurs sont reliés par un canal, nommé connecteur.

4) Le serveur d'applications (exemple : Tomcat !) reçoit la requête à son tour. Il est, lui, en mesure de la traiter. Il exécute donc le morceau d'application (la servlet) auquel est destinée la requête, en fonction de l'URL. Cette opération est effectuée à partir de la configuration du serveur. La servlet est donc invoquée, et le serveur lui fournit notamment deux objets Java (Tomcat est un serveur d'applications Java) exploitables : un représentant la requête, l'autre représentant la réponse. La servlet peut maintenant travailler, et générer la réponse à la demande. Cela peut passer par la consultation de sources de données, comme des bases de données (4'' sur le schéma). Ou bien par l'interrogation d'autres serveurs ou systèmes (4' sur le schéma), l'environnement Java web permettant de se connecter à de nombreux systèmes.

5) Une fois sa réponse générée, le serveur d'applications la renvoie, par le connecteur, au serveur web. Celui-ci la récupère comme s'il était lui-même allé chercher une ressource statique. Il a simplement délégué la récupération de la réponse, et celle-ci a été générée, mais ce n'est plus le problème.

6) La réponse est dorénavant du simple code HTML, compréhensible par un navigateur. Le serveur HTTP peut donc retourner la réponse au client.

4.Spring

Spring est un framework libre pour construire et définir l'infrastructure d'une application **java**, dont il facilite le développement et les tests.



Spring est considéré comme un conteneur dit « *léger* ». C'est-à-dire une infrastructure similaire à un **serveur d'applications J2EE**. Il prend donc en charge la création d'objets et la mise en relation d'objets par l'intermédiaire d'un fichier de configuration qui décrit les objets à fabriquer et les relations de dépendances entre ces objets. Le gros avantage par rapport aux serveurs d'application est qu'avec **Spring**, les classes n'ont pas besoin d'implémenter une quelconque interface pour être prises en charge par le framework (au contraire des **serveurs d'applications J2EE et des EJBs**). C'est en ce sens que **Spring** est qualifié de conteneur « *léger* ».

Spring s'appuie principalement sur l'intégration de trois concepts clés :

1. l'inversion de contrôle est assurée de deux façons différentes : la recherche de dépendances et l'injection de dépendances
2. la programmation orientée aspect
3. une couche d'abstraction.

La couche d'abstraction permet d'intégrer d'autres frameworks et bibliothèques avec une plus grande facilité. Cela se fait par l'apport ou non de couches d'abstraction spécifiques à des frameworks particuliers. Il est ainsi possible d'intégrer un module d'envoi de mails plus facilement.

L'inversion de contrôle :

1. La recherche de dépendance : consiste pour un objet à interroger le conteneur, afin de trouver ses dépendances avec les autres objets. C'est un cas de fonctionnement similaire aux EJBs.
2. L'injection de dépendances : cette injection peut être effectuée de trois manières possibles :

-
- L'injection de dépendance via le constructeur.
 - L'injection de dépendance via les modificateurs (setters).
 - L'injection de dépendance via une interface.

Les deux premières sont les plus utilisées par **Spring**.

Ce framework, grâce à sa couche d'abstraction, ne concurrence pas d'autres frameworks dans une couche spécifique d'un modèle architectural **Modèle-Vue-Contrôleur (MVC)** mais s'avère être un framework multicouches pouvant s'insérer au niveau de toutes les couches ; modèle, vue et contrôleur. Ainsi il permet d'intégrer **Hibernate** pour la couche de persistance ou encore **Struts** et **JavaServer Faces** pour la couche présentation.

4.0.1 Composition de Spring

Le noyau de Spring est basé sur :

- une fabrique générique de composants informatiques, composants nommés **beans** (en anglais : *haricots*, et dans le contexte Java : *grain de café*) ;
- un conteneur capable de stocker ces **beans**.

De plus, le noyau de **Spring** permet de forcer le contrôle de ces composants de leur extérieur, par la technique nommée inversion de contrôle.

Le principal avantage est de composer **les beans** de façon plus déclarative plutôt que de façon impérative dans le programme. **Les beans** peuvent être définis par le biais de fichiers de configuration **en Java ou XML**.

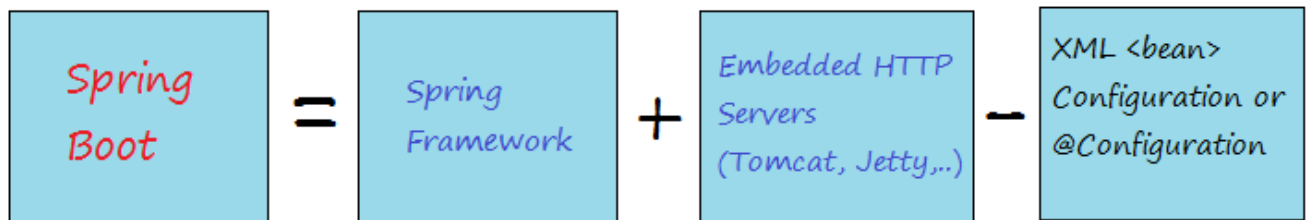
4.1. Spring Boot



Spring Boot est un projet qui se trouve sur la couche de **IO Execution** la couche de **Spring IO Framework**.

Spring Boot est une étape avancée qui simplifier le démarrage et le développement de nouvelles applications **Spring**. Avec **Spring Boot**, des configurations de **Spring** sont atténuées. **Spring Boot** soutient des conteneurs embarqués .Cela permet des application web d'exécuter indépendamment sans déploiement sur Web Server.

On peut utiliser **Spring boot** afin de créer l'application **Java Web application** qui exécute par la ligne de commande `'java -jar'` ou exporter le fichier **war** pour déployer sur le Web Server comme d'habitude. **Spring Boot** nous donne un "*CLI Tool*" pour exécuter le scénario du Spring (*spring scripts*). **Spring Boot** peut s'expliquer simplement par l'illustration ci-dessous:



Les avantages du Spring Boot sont:

1. Il est facile de développer l'application basée sur **Spring** avec **Java ou Groovy**.
2. Il diminue énormément du temps et augmente la productivité.
3. Il évite d'écrire plusieurs de codes d'expression standard, des Annotations et des configurations **XML**.
4. Il est très facile d'intégrer des applications **Spring Boot** avec ses écosystèmes de **Spring** comme **Spring JDBC, Spring ORM, Spring Data, Spring Security** etc.
5. Il suit l'approche "Configuration par défaut afin de diminuer le temps et l'effort de développement.
6. Il fournit des serveurs intégrés (Embedded HTTP servers) comme Tomcat, afin de développer et de tester des applications web à la manière la plus facilement.
7. Il fournit l'**outil CLI** (*Command Line Interface*) afin de développer et de tester des applications **Spring Boot** à partir de l'invite de commande facilement et rapidement.

-
8. Il fournit beaucoup de plugins afin de développer et de tester des applications **Spring Boot** rapidement en utilisant les outils de Build comme **Maven et Gradle**.
 9. Il fournit beaucoup de plugins afin de travailler avec des serveurs intégrés et la base de donnée stockée dans la mémoire Databases facilement.

4.2 Spring Security

Spring Security est un Framework de sécurité léger qui fournit une authentification et un support d'autorisation fiable, robuste et facile à intégrer afin de sécuriser les applications Spring. Il est livré avec des implémentations d'algorithmes de sécurité populaires.

4.2.1 Mettre en place Spring Security

L'installation du **Spring Security** dans une application **Spring** est simple. Elle repose sur la définition d'un fichier de configuration de ce premier framework. Le fichier doit être ensuite rajouté dans la liste des fichiers de configuration lus pendant l'initialisation du contexte. La configuration comprend les URLs vers les formulaires, la définition des rôles, la gestion des sessions et de gestionnaires d'autorisation et un filtre de sécurité qu'on doit préciser dans notre web.xml.

4.2.2 Connexion dans Spring Security

La gestion d'authentification sous **Spring Security** se fait avec l'élément `<http />`. Ce fragment permet de spécifier les **URLs** spécifiques auxquels on peut accéder uniquement avec un rôle déterminé (`intercept-url`). Les autres balises définissent les pages de connexion ou de déconnexion ainsi que la gestion des utilisateurs connectés.

4.2.3 Se souvenir de moi

La fonctionnalité *"se souvenir de moi"* se cache sur la balise `<remember-me />`.

Le principe de fonctionnement est simple. Si la case "**se souvenir de moi**" (appelé "`_spring_security_remember_me`" dans le code) est cochée, un cookie est envoyé et stocké chez l'utilisateur. Maintenant, après sa nouvelle visite sur le site, il sera automatiquement authentifié.

4.2.4 Gestionnaires d'authentification

Les gestionnaires d'authentification, **authentication-manager** dans notre fichier de configuration, précisent toute la configuration qui doit être utilisée dans le processus d'authentification de chaque balise `<http />`.

Tout d'abord, on définit le gestionnaire avec l'attribut **id**. Cet attribut est ensuite utilisé dans le paramètre **authentication-manager-ref** du `<http />`. Toute la configuration crée l'instance de la classe

`org.springframework.security.authentication.ProviderManager`.

Concernant la balise suivante, **authenciation-provider**, il s'agit d'un fournisseur des données d'authentification. C'est lui qui va récupérer l'utilisateur de la base de données et comparer ses valeurs (le login et le mot de passe) avec celles saisies dans le formulaire de connexion. La précision du service à utiliser peut se faire avec un des 2 attributs : **jdbc-user-service** ou **user-service-ref**.

Un seul enfant de la balise **authentication-provider**, **password-encoder**, définit la classe utilisée pour l'encodage du mot de passe. Il contient un enfant, **salt-source**. Cette balise définit, grâce à l'attribut **ref**, le **bean** qui se chargera de "saler" le mot de passe. *il s'agit de rajouter des caractères supplémentaires au mot de passe saisi par l'utilisateur afin de le rendre plus difficile à deviner par un utilisateur malveillant.*

4.2.5 Gestion de session sous Spring Security

La balise `<http />` permet aussi de définir la gestion des sessions sous Spring. Cela se fait grâce à la balise `<session-management />`

Les attributs de cette balise garantissent une meilleure protection de l'utilisateur contre les attaques visant la session.

Le premier attribut de la balise **session-management**, **invalid-session-url**, indique l'URL qui est appelé quand la session est invalide.

Le deuxième attribut, **session-fixation-protection**, aide à protéger notre application contre les attaques du type session fixation.

(Il s'agit des problèmes où un utilisateur malveillant crée la session manuellement sur le site et l'envoi sous forme d'URL le plus souvent à sa victime).

Trois paramètres existent pour protéger contre cette attaque :

- **migrateSession** : une nouvelle session est créée pour la victime. La session contient les paramètres de l'ancienne session. (celle de l'utilisateur malveillant)
- **none** : ne fait rien, la session de l'utilisateur malveillant est gardée.
- **newSession** : une nouvelle session est créée au propre. Aucune copie des données n'est effectuée.

L'enfant de la balise **session-management** concerne l'accès concurrentiel d'un utilisateur au système. **<concurrency-control />** permet de spécifier combien de sessions actives ouvertes dans son navigateur peut avoir un utilisateur. L'attribut **max-sessions** spécifie ce nombre tandis qu'**error-if-maximum-exceeded** indique si l'exception **SessionAuthenticationException** doit être lancée quand ce nombre sera dépassé.

4.2.6 Gestion de rôle sous Spring Security

Dans la partie **intercept-url** de la balise http il y a un attribut que je n'ai pas encore expliqué. Il s'agit de **access**. Il se charge d'établir une liste des rôles qui peuvent accéder à une ressource donnée.

Comment sont définis les rôles d'un utilisateur ? Pour le savoir, on doit revenir vers le **UserDetailsService** .

Les parties importantes sont : la création d'une collection contenant les instances de la classe **GrantedAuthority** et son rajout dans le constructeur **AuthenticationFrontendUserDetails**. Il s'agit donc d'une liste d'autorités (rôles) associées à l'utilisateur connecté.

Les rôles sont un élément de base d'ACL. Ils permettent de définir les ressources auxquelles peut accéder un utilisateur dans une application basée sur les rôles différents. Dans notre cas, le backoffice implémentera ce mécanisme.

4.2.7 Annotations d'autorisation

Maintenant quand on a découvert les fonctionnalités importantes du Spring Security, on doit savoir comment l'implémenter dans le code de nos contrôleurs et des services. Pour ce faire on peut soit utiliser l'attribut **access** de la balise **intercept-url**, soit utiliser les **annotations**.

Pour savoir si un utilisateur connecté peut "exécuter" une méthode du service ou du contrôleur, on utilisera l'annotation **@PreAuthorize**. Cette annotation peut appeler des méthodes d'autorisation différentes.

La méthode **isAnonymous()** vérifie si l'utilisateur n'est pas connecté. En occurrence, on veut que seulement l'utilisateur pas connecté puisse accéder au formulaire d'enregistrement.

La méthode **hasRole()** assure que l'utilisateur connecté possède un rôle **ROLE_USER**. Si c'est le cas, la méthode du contrôleur sera exécutée. C'est logique car la fonction en question affichera le formulaire d'édition des données de l'utilisateur authentifié.

Il existe une méthode utilisant **Spring Expression Language (SpEL)**. Le fragment **#subscriber.login** se réfère à l'instance de la classe **Subscriber** passée en paramètre de cette méthode. Il s'agit de l'instance de **UserDetails**. Le mot **or** signale l'exclusion; pour que la méthode du service puisse être exécutée, soit la condition de gauche doit être vraie, soit celle de droite.

La méthode **hasAnyRole** liste des rôles qui peuvent accéder à la méthode. Dans notre cas, les rôles admis pour pouvoir modifier un écrivain, sont **ROLE_ADMIN** et **WRITER_ADD**.

Une autre annotation, **@PostAuthorize**, permet d'effectuer l'action de validation après l'exécution d'une méthode. Cependant, on ne va pas l'employer dans notre application.

4.2.8 Récupérer l'utilisateur connecté

Dans certaines applications on a besoin de récupérer les données de l'utilisateur connecté.

En ce qui concerne la partie de vue, on peut l'achever grâce à la librairie des tags mise à disposition avec **Spring Security**.

La récupération se fera alors avec `<sec:authentication property="principal.username"/>`. Cependant, la récupération dans le Demande beaucoup de temps . Elle nécessite la création d'un **resolver**.

Cependant, cette notion a déjà été expliquée dans la partie consacrée aux annotations.

On peut récupérer l'instance **UserDetails** grâce à un seul paramètre passé dans la méthode. Il doit être annoté avec **@LoggedInUser**.

5. Thymeleaf

Thymeleaf est un moteur de template Java moderne côté serveur pour les **environnements Web** et autonomes.



L'objectif principal de **Thymeleaf** est d'intégrer des **modèles** naturels élégants à notre flux de travail de développement avec un code **HTML** pouvant être correctement affiché dans les navigateurs et fonctionnant également en tant que **prototypes statiques**, ce qui permet de renforcer la collaboration au sein des équipes de développement.

Avec des modules pour **Spring Framework**, une multitude d'intégrations pour nos outils et la possibilité d'intégrer nos propres fonctionnalités, **Thymeleaf** est idéal pour le développement **Web JVM HTML5 moderne** - bien qu'il puisse faire beaucoup plus.

5.1 Modèles naturels

Les modèles **HTML** écrits dans **Thymeleaf** ont toujours le même aspect et le même fonctionnement **HTML**, ce qui permet aux modèles exécutés dans l'application de continuer à fonctionner comme des artefacts de conception utiles.

```
<table>
  <thead>
    <tr>
      <th th:text="#{msgs.headers.name}">Name</th>
      <th th:text="#{msgs.headers.price}">Price</th>
    </tr>
  </thead>
  <tbody>
    <tr th:each="prod: ${allProducts}">
      <td th:text="${prod.name}">Oranges</td>
      <td th:text="${#numbers.formatDecimal(prod.price, 1,
2) }">0.99</td>
    </tr>
  </tbody>
</table>
```

5.2 Les intégrations et écosystème

Eclipse, IntelliJ IDEA, Spring, Play, voire l'**API Model-View-Controller** pour **Java EE 8.***
On peut écrire **Thymeleaf** dans ces outils à l'aide d'infrastructure de développement Web.

L'écosystème Thymeleaf est composé d'outils et d'extensions de deux types:

- *Extensions officielles* (alias *Thymeleaf Extras*): créées par l'équipe Thymeleaf et prises en charge dans le cadre du projet.
-
- *Extensions non officielles* ou *communautaires*: développées et mises à jour par des utilisateurs de Thymeleaf en dehors du projet Thymeleaf et distribuées sous leurs propres conditions de licence et de support.

5.2.1 Spring+Thymeleaf

Thymeleaf propose un ensemble d'intégrations **Spring** qui nous permettent de l'utiliser comme substitut complet du logiciel **JSP** dans les applications **Spring MVC**.

Ces intégrations nous permettront de:

1. Appliquez les méthodes mappées de vos **@Controller** objets **Spring MVC** aux modèles gérés par **Thymeleaf**, exactement comme vous le faites avec les **JSP**.
2. Utilisez **Spring Expression Language (Spring EL)** au lieu de **OGNL** dans vos modèles.
3. Créez des formulaires dans vos modèles complètement intégrés à vos *beans* de sauvegarde de formulaire et à vos liaisons de résultats, y compris l'utilisation d'éditeurs de propriétés, de services de conversion et du traitement des erreurs de validation.
4. Affiche les messages d'internationalisation des fichiers de messages gérés par **Spring** (à travers les **MessageSource** objets habituels).
5. Résolvez vos modèles à l'aide des mécanismes de résolution des ressources propres à **Spring**.

5.3 Qui utilise Thymeleaf ?

Across , Auchan, Broadleaf , Apereo CAS, Connect Group, Enerko Informatik, Enonic , Lagerwey , PPI AG, Sahibinden.com, Trabe, Travelcompositor, VEDA GmbH, YobiDrive et biens d'autres !

6.Bootstrap



Bootstrap est une boîte à outils open source avec **HTML, CSS et JS** pour le développement de sites réactifs destinés au mobile. Il permet de construire notre application avec des variables et des combinaisons **Sass** (*langage d'extension CSS extrêmement stable*) avec un système de grille réactif, de nombreux composants prédéfinis et de puissants plug-ins basés sur **jQuery**.

6.Jenkins

Jenkins, premier serveur d'automatisation open source, fournit des centaines de plug-ins permettant de créer, déployer et automatiser n'importe quel projet.



Jenkins peut être utilisé pour automatiser toutes sortes de tâches liées à la création, au test, à la livraison ou au déploiement de logiciels.

Jenkins peut être installé via des packages système natifs, Docker, ou même fonctionner de manière autonome sur n'importe quel ordinateur sur lequel un environnement d'exécution Java (JRE) est installé.

7. Sonar

Sonar offre une solution performante du contrôle de la qualité du logiciel. Mais qu'est-ce que la qualité d'un logiciel, et en quoi est-il important de la contrôler ?

En paraphrasant Wikipédia, « *la gestion de la qualité est l'ensemble des activités qui concourent à l'obtention de la qualité dans un cadre de production de biens ou de services (dans notre cas, d'un logiciel). Plus largement, c'est aussi un moyen que se donnent certaines organisations, dans des buts tels que la mise en conformité par rapport aux standards du marché.* »

Dans le monde informatique en général, et en Java en particulier, la qualité d'une application va être directement liée à la qualité du code. De nombreux outils s'affairent à contrôler certains aspects de cette qualité du code : **exécution de tests unitaires, analyse de la couverture du code par ces tests, vérifications du respect des règles de codage, etc.** Il est donc possible de contrôler la qualité de son code grâce à ces outils, et d'avoir une confiance accrue en son application !

Le contrôle de la qualité va donc pousser l'équipe de développement à adopter et à respecter certains standards de développement. Le but de tout cela étant bien entendu de rendre le code plus sûr, mais de permettre d'y **déceler les erreurs le plus rapidement possible...** et donc de les corriger !

Le "**Toyota Way**" correspond à une méthodologie extrêmement appréciée aujourd'hui, aussi appelée le "**Lean**". Celle-ci est basée sur 14 principes dont l'un d'eux est le "*Build a culture of stopping to fix problems, to get quality right the first time*". Ce principe est là pour nous rappeler **qu'il est impossible d'accélérer et de soutenir une fréquence de production sans que la qualité soit au coeur de toutes les actions**. Autrement dit, il n'est pas possible d'aller vite sans qualité, mais qu'il n'est pas possible non plus de faire de la qualité sans vitesse. C'est aussi pour cela qu'il est aujourd'hui primordial de disposer d'une intégration continue et d'un processus itératif et incrémental.

L'un des intérêts de la surveillance de la qualité est la détection précoce des éventuels problèmes. Or lorsque l'on sait que le coût de la correction d'une erreur augmente considérablement avec le temps (voir image ci-dessous), on comprend très vite l'importance de la détection rapide des erreurs...

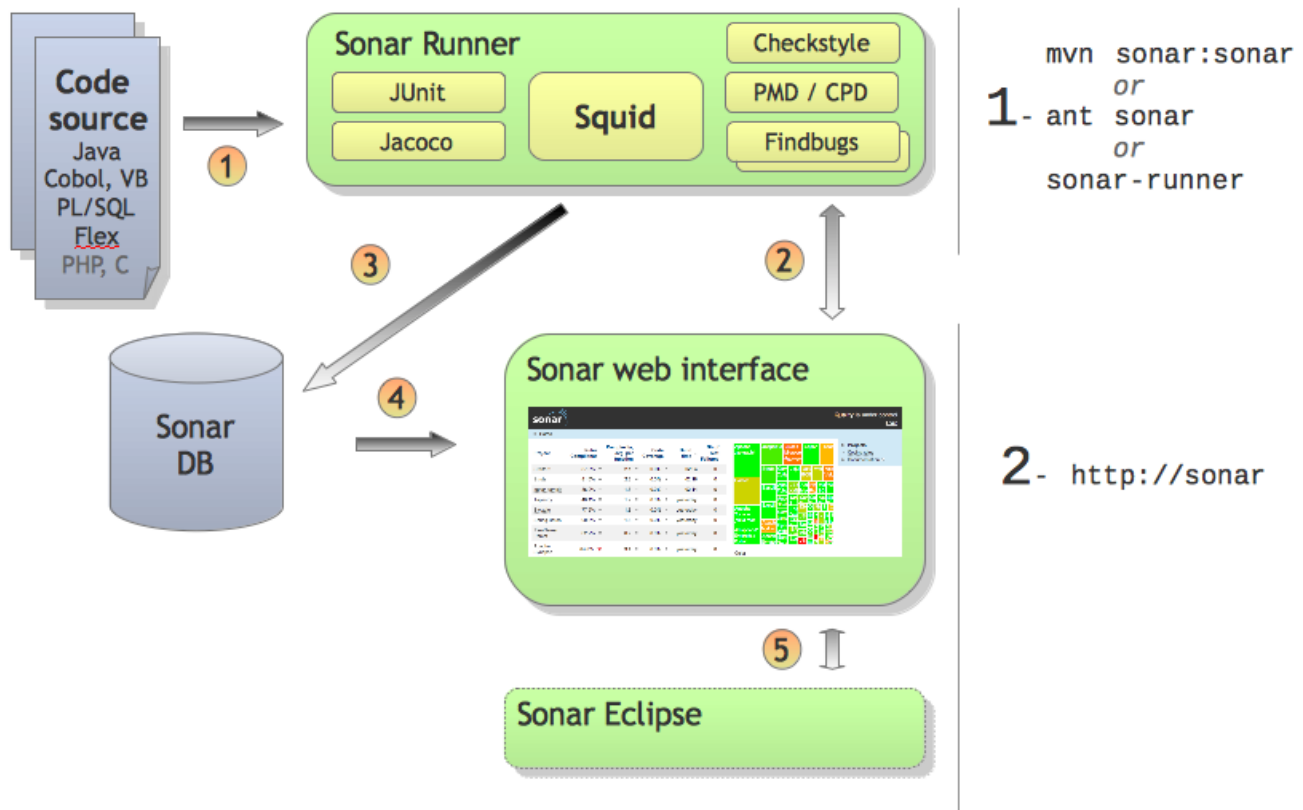
7.1 Description de Sonar



Sonar est un outil open source initialement développé par la société suisse **Hortis**. Depuis novembre 2008, c'est la société suisse **SonarSource** qui se charge du développement et du support de Sonar. Le but principal de cet outil est de fournir une analyse complète de la qualité d'une application en fournissant de nombreuses statistiques (ou **métriques**) sur ses projets. Ces données permettent ainsi d'évaluer la qualité du code, et d'en connaître l'évolution au cours du développement.

D'un point de vue architectural, Sonar est composé de plusieurs couches :

- un exécuteur (basé sur Maven 2/3, Ant ou un exécuteur Java) dont le but sera de lancer un certain nombre d'outils d'analyse, et d'en agréger les résultats ;
- une base de données, qui stocke et historise les informations sur les projets surveillés par Sonar ;
- le serveur web qui permet la navigation et la consultation des analyses réalisées sur les projets ;
- éventuellement un plugin pour Eclipse qui offre une meilleure intégration des données de Sonar dans son outil de développement.



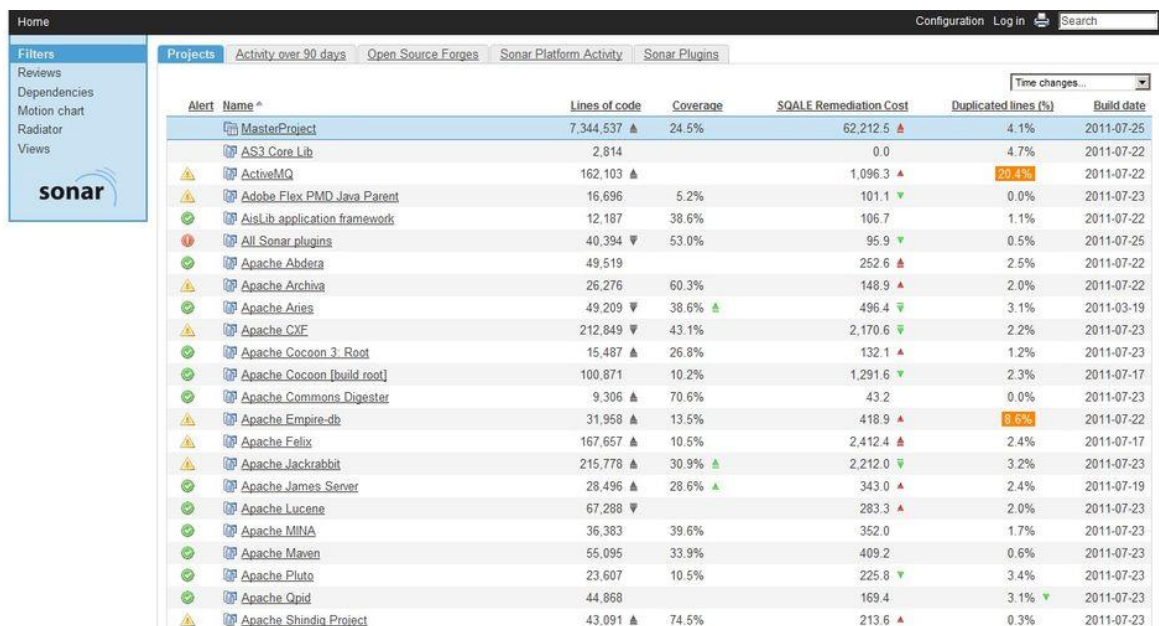
7.2 Les fonctionnalités de Sonar

Voici les principales fonctionnalités de l'outil Sonar :

- Tableau de bord complet des différents projets suivis.
- Détection rapide du code à risque.
- Mesures quantitatives : nombre de classes, duplication de code, etc.
- Mesures qualitatives : couverture et taux de réussite des tests, complexité du code, respect des règles de codage...
- Historiques des statistiques, pour en voir l'évolution au cours du temps.
- Support de plus de 600 règles de qualité.
- Gestion de profils pour les règles de codage.
- Visualisation du code source, surlignant les violations des règles de codage qui s'y trouvent.
- Fonction "Time machine" permettant de comparer plusieurs versions d'une même application.
- Identification des points faibles d'un projet.
- Support des plugins.

7.3 Démonstration de Sonar

SonarSource met à disposition le site <http://nemo.sonar.codehaus.org>, qui regroupe un certain nombre de projets open source. Cela permet de voir les différentes fonctionnalités de l'outil, sans nécessairement l'installer soi-même. De plus, cette version publique est toujours mise à jour, ce qui permet également de tester les nouvelles fonctionnalités !



| Alert | Name | Lines of code | Coverage | SQALE Remediation Cost | Duplicated lines (%) | Build date |
|-------|------------------------------|---------------|----------|------------------------|----------------------|------------|
| | MasterProject | 7,344,537 ▲ | 24.5% | 62,212.5 ▲ | 4.1% | 2011-07-25 |
| | AS3 Core Lib | 2,814 | | 0.0 | 4.7% | 2011-07-22 |
| ⚠ | ActiveMQ | 162,103 ▲ | | 1,096.3 ▲ | 28.4% | 2011-07-22 |
| ⚠ | Adobe Flex PMD Java Parent | 16,696 | 5.2% | 101.1 ▼ | 0.0% | 2011-07-23 |
| ✓ | AisLib application framework | 12,187 | 38.6% | 106.7 | 1.1% | 2011-07-22 |
| ⚠ | All Sonar plugins | 40,394 ▼ | 53.0% | 95.9 ▼ | 0.5% | 2011-07-25 |
| ✓ | Apache Abdera | 49,519 | | 252.6 ▲ | 2.5% | 2011-07-22 |
| ⚠ | Apache Archiva | 26,276 | 60.3% | 148.9 ▲ | 2.0% | 2011-07-22 |
| ✓ | Apache Aries | 49,209 ▼ | 38.6% ▲ | 496.4 ▼ | 3.1% | 2011-03-19 |
| ⚠ | Apache CXF | 212,849 ▼ | 43.1% | 2,170.6 ▼ | 2.2% | 2011-07-23 |
| ✓ | Apache Cocoon 3 Root | 15,487 ▲ | 26.8% | 132.1 ▲ | 1.2% | 2011-07-23 |
| ✓ | Apache Cocoon [build root] | 100,871 | 10.2% | 1,291.6 ▼ | 2.3% | 2011-07-17 |
| ✓ | Apache Commons Digester | 9,306 ▲ | 70.6% | 43.2 | 0.0% | 2011-07-23 |
| ⚠ | Apache Empire-db | 31,958 ▲ | 13.5% | 418.9 ▲ | 8.6% | 2011-07-22 |
| ⚠ | Apache Felix | 167,657 ▲ | 10.5% | 2,412.4 ▲ | 2.4% | 2011-07-17 |
| ⚠ | Apache Jackrabbit | 215,778 ▲ | 30.9% ▲ | 2,212.0 ▼ | 3.2% | 2011-07-23 |
| ✓ | Apache James Server | 28,496 ▲ | 28.6% ▲ | 343.0 ▲ | 2.4% | 2011-07-19 |
| ✓ | Apache Lucene | 67,288 ▼ | | 283.3 ▲ | 2.0% | 2011-07-23 |
| ✓ | Apache MINA | 36,383 | 39.6% | 352.0 | 1.7% | 2011-07-23 |
| ✓ | Apache Maven | 55,095 | 33.9% | 409.2 | 0.6% | 2011-07-23 |
| ✓ | Apache Pluto | 23,607 | 10.5% | 225.8 ▼ | 3.4% | 2011-07-23 |
| ✓ | Apache Qpid | 44,868 | | 169.4 | 3.1% ▼ | 2011-07-23 |
| ⚠ | Apache Shindig Project | 43,091 ▲ | 74.5% | 213.6 ▲ | 0.3% | 2011-07-23 |

8. Crafter CMS



Crafter CMS est un système de gestion de contenu Web Open Source basé sur Java, conçu pour les sites Web, les applications mobiles, la réalité virtuelle, tous cela pour faciliter le développement et la mise à l'échelle. **Crafter CMS** est composé de plusieurs composants indépendants basés sur un microservice, notamment:

- un référentiel de contenu principal
- une application de création pour les éditeurs / gestionnaires de contenu (*Crafter Studio*)
- un système de diffusion de contenu dynamique (*Crafter Engine*)
- un magasin de profils utilisateur
- un serveur de personnalisation (*Crafter Profile*)
- un serveur de requête / recherche (*Crafter Search*)
- un magasin de contenu social
- un serveur (*Crafter Social*).

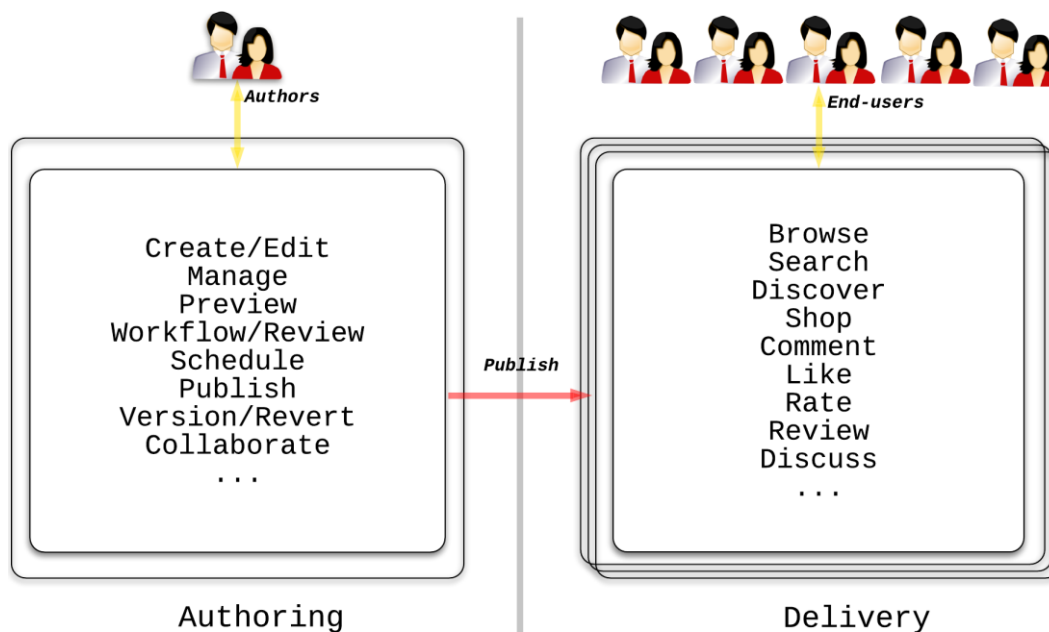
8.1 Architecture de Crafter

Crafter CMS se distingue par son architecture moderne, ce qui permet:

- Truly Decoupled CMS (livraison globale déconnectée), composé de composants indépendants basés sur un microservice
- Livraison dynamique / personnalisée de chaque demande à la vitesse
- API premier CMS (contenu en tant que service)
- CMS basé sur Git (permet une excellente cadence de développement)
- Architecture de diffusion sans partage (échelle extrême)

- Agnostique front-end (apportez votre framework d'interface utilisateur préféré ou utilisez-le comme un CMS sans tête)
- Support égal pour les trois parties prenantes du CMS: auteurs de contenu, développeurs et administrateurs système

Crafter CMS est un véritable système de gestion de contenu découplé, mais il prend en charge une diffusion de contenu dynamique et personnalisée. Pour mieux comprendre cela, les systèmes de gestion de contenu découplés typiques ont l'architecture générale suivante:



La plupart des systèmes de gestion de contenu découplés compilent le contenu dans les artefacts finaux et le transfèrent au niveau de livraison. Bien que cela permette une diffusion déconnectée et une évolutivité extrême (*il suffit d'ajouter des serveurs sur le niveau de diffusion qui diffusent le contenu ou d'utiliser un CDN*), cette approche ne permet pas la diffusion **dynamique / personnalisée du contenu**.

De nombreux CMS réellement couplés prétendent être découplés. Ces systèmes nous permettent d'avoir un niveau de création distinct du niveau de livraison, mais ces derniers sont effectivement connectés via une synchronisation de base de données. Cela signifie que les niveaux de livraison ne peuvent pas s'exécuter sans un certain niveau de connectivité avec le maître de création et que, par conséquent, l'échelle du niveau de livraison est limitée.

Un système véritablement découplé prend en charge les livraisons déconnectées (pensez à un niveau de livraison qui fonctionne dans un sous-marin ou un navire de croisière). Bien que l'exécution de nœuds de livraison déconnectés soit un exemple extrême, c'est un bon test de la véritable évolutivité du niveau de livraison d'un CMS.

Crafter CMS est véritablement découplé et seuls les actifs sont publiés du niveau création au niveau publication. Ces actifs comprennent des **fichiers XML** et des **actifs statiques** (*images, CSS, code Groovy, etc.*). Le niveau de livraison ingère ces artefacts et peut ensuite fournir l'expérience souhaitée.

Comment **Crafter CMS** peut-il offrir une expérience dynamique?

Lors de l'ingestion, le niveau de diffusion indexe le contenu dans un moteur de recherche local et construit une représentation en mémoire des éléments de contenu pour aider à définir le comportement dynamique. Le moteur de recherche et le magasin en mémoire sont locaux et ne partagent donc rien avec les autres nœuds. Cependant, nous pouvons désormais rechercher et créer des réponses dynamiques.

Qu'en est-il de la personnalisation et du ciblage?

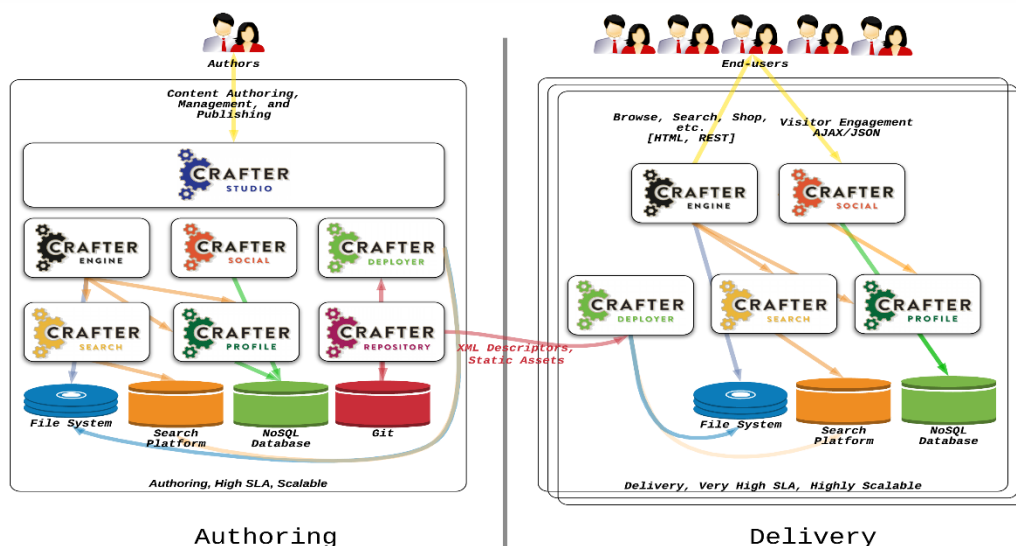
Crafter CMS a deux sous-systèmes qui sont sauvegardés par une base de données **NoSQL** pour aider à la personnalisation et **UGC (User Generated Content): Crafter Profile et Crafter Social**. Ces informations collectées sur l'utilisateur (*connecté ou non*) et peuvent générer un comportement **dynamique** et permettre à l'utilisateur de dialoguer avec le site (*commentaires, notes, etc.*).

Livraison déconnectée!?

Crafter Profile et Crafter Social ont en effet besoin d'une base de données, mais:

- 1) le CMS ne les impose pas, vous pouvez fournir du contenu sans ces fonctionnalités
- 2) le choix que NoSQL vous aide en géo-distribution, à grande échelle et avec une certaine déconnexion pour cohérence éventuelle.

Ci-dessous, un diagramme montrant le CMS Crafter, ainsi que tous les microservices de création et de diffusion:

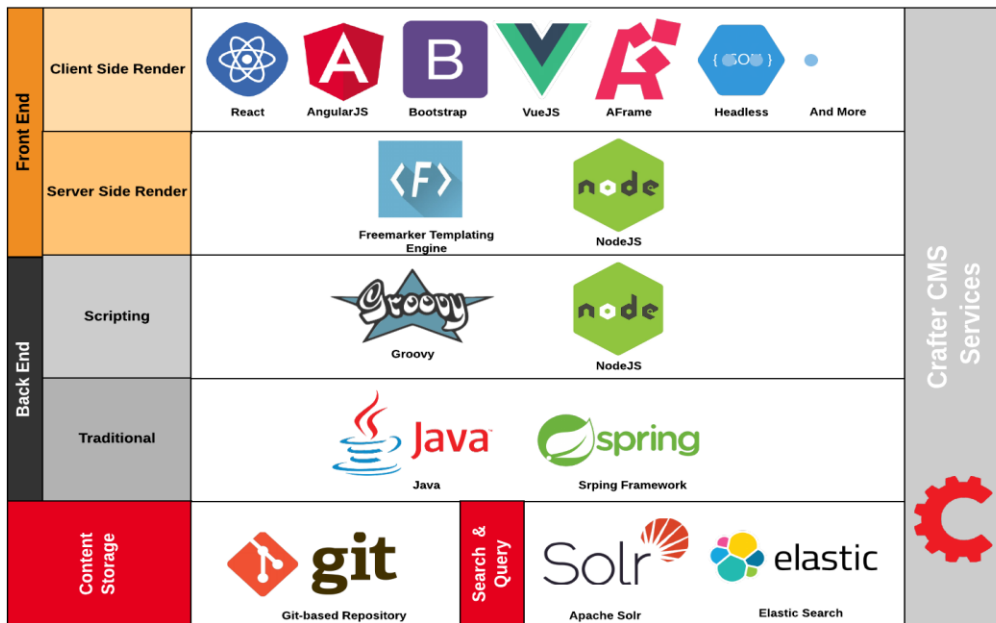


8.2 Qu'utilise Crafter CMS ?

Crafter est un CMS dynamique capable de restituer du balisage ou de diffuser du contenu via une API. Nous pouvons utiliser n'importe quel framework frontal sur le serveur de contenu sans tête de Crafter: **React, Angular, Vue, Bootstrap, ect....**

Mais ce n'est pas tous !

Crafter peut aussi créer des applications modernes avec des outils, langages, frameworks multiples et diverses que nous voyons ici :



8.3 Documentation de Crafter.

La documentation du Logiciel se trouve sur <https://docs.craftercms.org/en/index.html> (en anglais ;)

9. Maven

Maven est un outil de construction de projets (build) open source développé par la fondation Apache, initialement pour les besoins du projet Jakarta Turbine. Il permet de faciliter et d'automatiser certaines tâches de la gestion d'un projet Java.



Il permet notamment :

- d'automatiser certaines tâches : compilation, tests unitaires et déploiement des applications qui composent le projet
- de gérer des dépendances vis-à-vis des bibliothèques nécessaires au projet
- de générer des documentations concernant le projet

Au premier abord, il est facile de croire que **Maven** fait double emploi avec **Ant**.

Ant et Maven sont tous les deux développés par le groupe **Jakarta**, ce qui prouve bien que leur utilité n'est pas aussi identique que cela. **Ant**, dont le but est d'automatiser certaines tâches répétitives, est plus ancien que **Maven**. **Maven** propose non seulement les fonctionnalités d'**Ant** mais en propose de nombreuses autres.

Pour gérer les dépendances du projet vis-à-vis de bibliothèques, **Maven** utilise un ou plusieurs dépôts qui peuvent être locaux ou distants.

Maven est extensible grâce à un mécanisme de plugins qui permettent d'ajouter des fonctionnalités.

6. Architecture de l'application

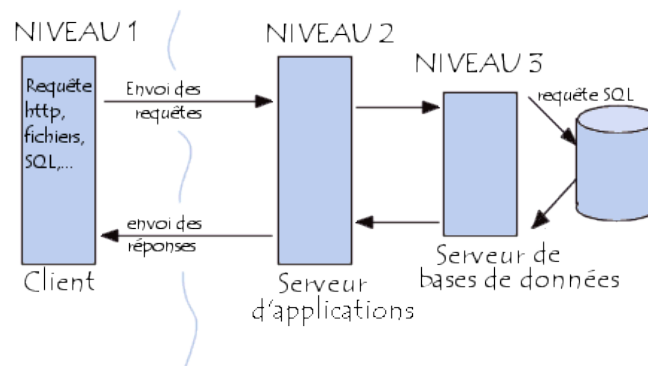
1. Architecture N-tiers:

L'**architecture N-tier** (*en anglais tier : étage, niveau*), ou encore appelée **multi-tier**, est une architecture client-serveur dans laquelle une application est exécutée par plusieurs composants logiciels distincts.

Voici comment se comporte mon architecture 3-tier:

- **Tier de présentation** : interfaces utilisateurs sur un PC poste de travail, qui s'adressent à des applications serveur
- **Tier de l'application**: applications serveur qui contiennent le **moteur** du site et accèdent aux données stockées dans des bases de données
- **Tier de base de données** : serveurs de bases de données

N-tier : Le lien entre les niveaux est défini et limité à des interfaces. Les interfaces assurent la modularité et l'indépendance technologique et topologique de chaque niveau



1.1 Pourquoi utiliser ce type d'architecture ?

Dans les applications à 2 niveaux (2-tiers) :

- Orienté clients, la plupart des traitements sont effectués sur le poste du client qui possède une Interface Homme Machine dite **"lourde"**.
- Dans celles orientés serveurs c'est le serveur qui effectue tous les traitements. Le client lui possède une **IHM légère**.

Qu'elle soit légère ou lourde l'**IHM** est propriétaire est nécessite donc une installation sur le poste client.

Ceci rend l'installation d'une telle application très lourde si le nombre de client est important. De plus ont peu très difficilement réutiliser ce type d'application car l'IHM est dédié et composée d'un seul bloc. Enfin les performances réseaux ne sont pas très élevés car de nombreuses requêtes doivent transiter par le réseau.

Pour pallier au différent inconvénient que nous avons vues, on a décidé de séparer le traitement applicatif à la fois des données et de l'interface, c'est ce qui a amené l'architecture 3-tier (3-tiers ou plus = **N-tiers**).

Le tableau ci-dessous présente les différences qu'apportent les architectures 3-tier aux anciennes architectures 2-tier.

| | | |
|---------------------------|--|--|
| Administration du système | Complexe (la couche application est physiquement répartie sur plusieurs postes clients) | Moins complexe (les applications peuvent être gérées centralement sur le serveur) |
| Sécurité | Faible (sécurité au niveau des données) | Elevée (raffinée au niveau des services ou des méthodes) |
| Encapsulation des données | Faible (les tables de données sont directement accessibles) | Elevée (le client fait appel à des services ou méthodes) |
| Performance | Faible (plusieurs requêtes SQL sont transmises sur le réseaux, les données sélectionnées doivent être acheminées vers le client pour analyse) | Bonne (seulement les appels de services et les réponses sont mis sur le réseau) |
| Extensibilité | Faible (gestion limitée des liens réseaux avec le client) | Excellente (possibilité de répartir dynamiquement la charge sur plusieurs serveurs) |

| | | |
|---|---|--|
| Réutilisation | Faible (application monolithique sur le client) | Excellente (réutilisation des services et des objets) |
| Facilité de développement | Elevée | En progression (des outils intégrés pour développer la partie du client et du serveur) |
| Lien Serveur-serveur | Non | Oui (via le middleware Serveur/Serveur) |
| Intégration des systèmes déjà en place | Non | Oui (via des passerelles encapsulées par les services ou objets) |
| Soutien Internet | Faible (les limitations de la bande passante pénalisent le téléchargement d'applications de type "fat-client") | Excellente (les applications de type "thin-client" sont facilement téléchargeable et les appels aux services repartissent la charge sur un ou plusieurs serveurs) |
| Sources de données hétérogènes | Non | Oui (les applications 3-tier peuvent utiliser plusieurs bases de données dans la même transaction) |
| Choix de communications de type "riche" | Non (synchrone et RPC) | Oui (gestion asynchrone de messages , files de livraison, publication et abonnement, "broadcast") |
| Flexibilité d'architecture matériel | Limitée | Excellente (possibilité de faire résider les couches 2 et 3 sur une ou plusieurs machines) |
| Relève en cas de pannes | Faible | Excellente (possibilité d'avoir la couche du centre "middle-tier" sur plusieurs serveurs) |

On peut constater que l'utilisation d'une architecture 3-tier ou n-tier apportent des correctifs a une grande partie des problèmes des architectures 2-tier.

2. MVC:

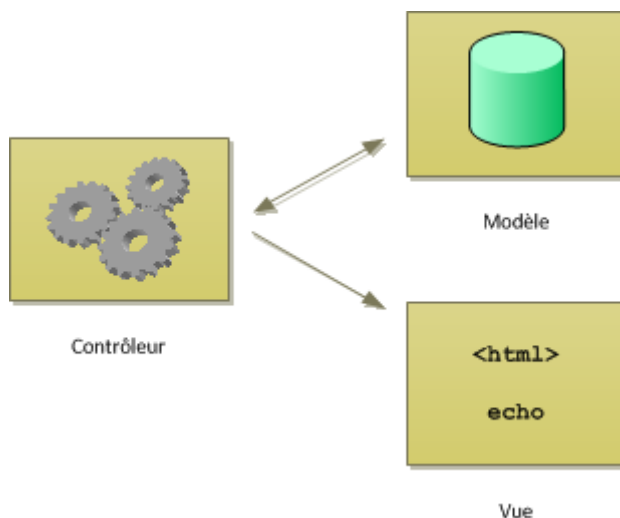
Modèle-vue-contrôleur ou **MVC** est un motif d'architecture logicielle destiné aux interfaces graphiques très populaire pour les applications web. Le motif est composé de trois types de modules ayant trois responsabilités différentes : **les modèles, les vues et les contrôleurs**.

- **Un modèle (Model)** cette partie gère les *données* de votre site. Son rôle est d'aller récupérer les informations « brutes » dans la base de données, de les organiser et de les assembler pour qu'elles puissent ensuite être traitées par le contrôleur. On y trouve donc entre autres les requêtes SQL.

- **Une vue (View)** cette partie se concentre sur l'*affichage*. Elle ne fait presque aucun calcul et se contente de récupérer des variables pour savoir ce qu'elle doit afficher. On y trouve essentiellement du code HTML mais aussi du JavaScript.
- **Un contrôleur (Controller)** cette partie gère la logique du code qui prend des *décisions*. C'est en quelque sorte l'intermédiaire entre le modèle et la vue : le contrôleur va demander au modèle les données, les analyser, prendre des décisions et renvoyer le texte à afficher à la vue. C'est notamment lui qui détermine si le visiteur a le droit de voir la page ou non (gestion des droits d'accès).

Ce motif est utilisé par de nombreux frameworks pour applications web tels que **Ruby on Rails, Grails, ASP.NET MVC, Spring, Struts, Symfony, Apache Tapestry, Laravel, ou AngularJs**.

Il est important de bien comprendre comment ces éléments s'agencent et communiquent entre eux. Regardez bien la figure suivante.



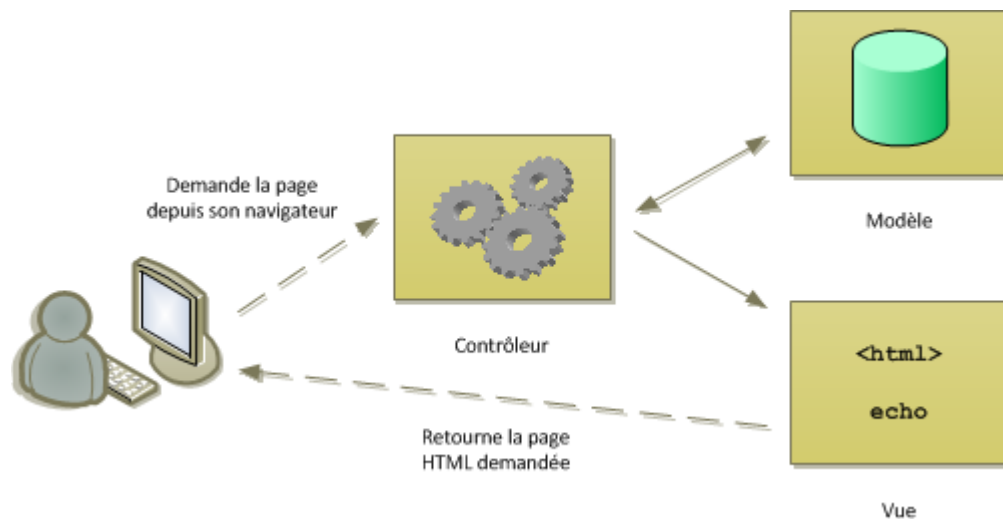
Nous pouvons voir sur cette image que le contrôleur est le chef d'orchestre : c'est lui qui reçoit la requête du visiteur et qui contacte d'autres fichiers (le modèle et la vue) pour échanger des informations avec eux.

Le fichier du contrôleur demande les données au modèle sans se soucier de la façon dont celui-ci va les récupérer.

Par exemple : « Donne-moi la liste des 30 derniers messages du forum numéro 5 ». Le modèle traduit cette demande en une requête SQL, récupère les informations et les renvoie au contrôleur.

Une fois les données récupérées, le contrôleur les transmet à la vue qui se chargera d'afficher la liste des messages.

Concrètement, le visiteur demandera la page au contrôleur et c'est la vue qui lui sera retournée, comme schématisé sur la figure suivante.



Bien entendu, tout cela est transparent pour lui, il ne voit pas tout ce qui se passe sur le serveur. c'est pourtant sur ce type d'architecture que repose un grand nombre de sites professionnels et donc c'est ce type d'architecture que j'ai travaillé.

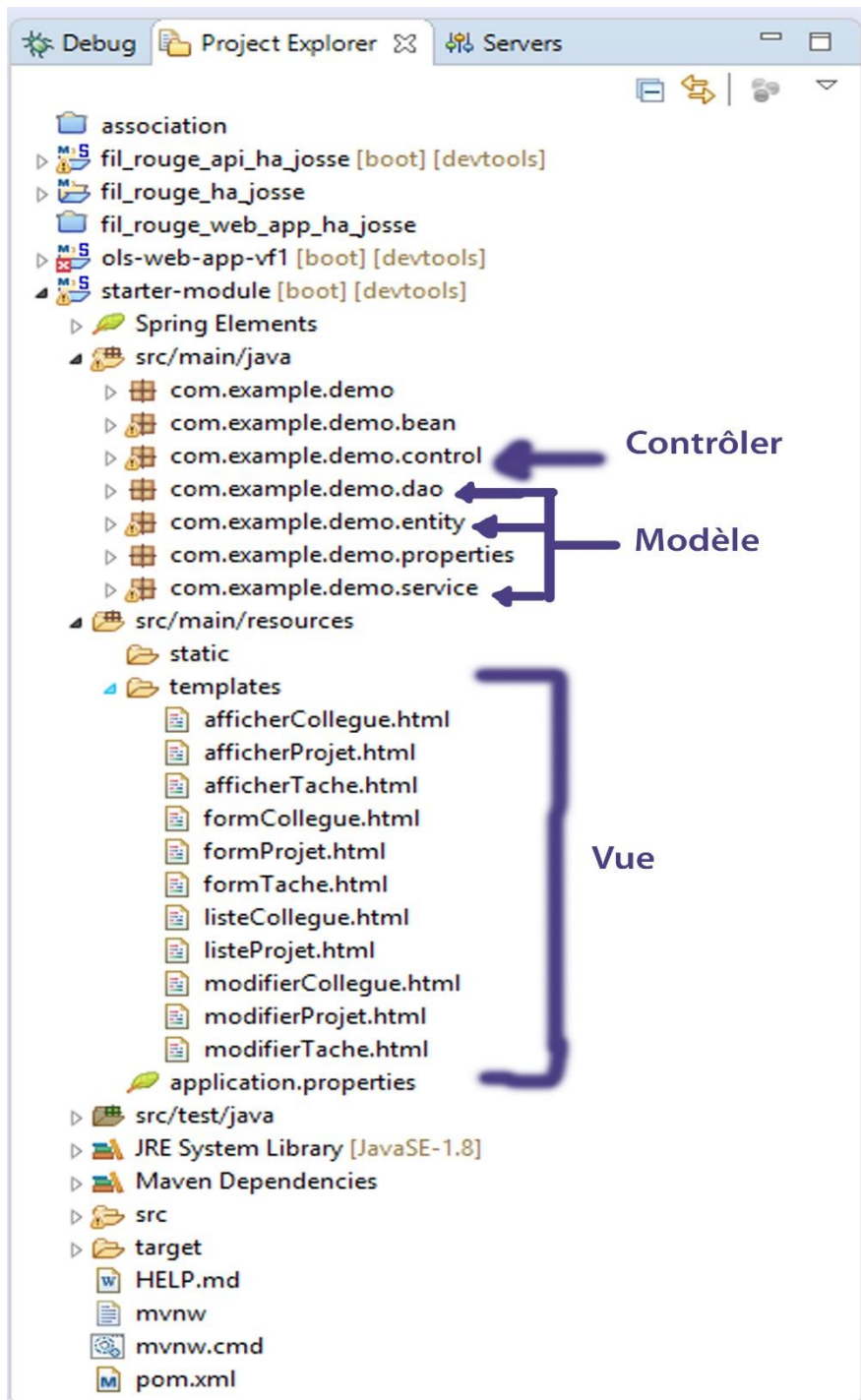
2.1 MVC avec Eclipse

Alors comment cela fonctionne concrètement sur **Eclipse** ?

Tout d'abord, nous devons prendre connaissances et comprendre qu'elles packages seront liés à qu'elle partit du **MVC** :

- Le package Control seras logiquement le **Controller**.
- Les fichiers Html situé dans le dossier Template seront les **Vues** .
- Pour le **Model**, se seras tous ce qui touche à la base de données (Services, DAO, Entités).

Puis si tous cela se passent bien nous devons retrouver logiquement des Packages se rapprochant de cette image :



2.2 MVC avec Springboot

J'ai utilisais, lors de la création du projet, le Framework Springboot (accès à Spring et tomcat) et installé des **Dépendances** tel que Devtools, Security, Jersey, Mail et Web simplifiant le démarrage et le développement de l'application.

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-devtools</artifactId>
  <scope>runtime</scope>
</dependency>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-test</artifactId>
  <scope>test</scope>
</dependency>
```

La facilitée de développer avec SpringBoot .

Grace à Spring il plus facile de développer une application avec ces **annotations et les injections de Dépendances**. Pour faire **l'injection de dépendance** il faut utiliser l'annotation **@Autowired**.

L'injection de dépendance(ID) est un design pattern qui nous permet de supprimer les dépendances entre les objets et de faire l'injection automatiquement lors du démarrage du conteneur Spring, ce dernier facilite la gestion de l'application, les tests et maintenance de l'application avec un couplage du code faible. Donc on peut changer l'application sans faire beaucoup de modifications dans le code.

```
package fr.ols.webapp.controller;

import java.util.List;

@Controller
@RequestMapping("/secured")
public class ControllerSection {

    @Autowired
    public ISectionService sectionService;

    @GetMapping("/listeSections")
    public String listeSections(Model model) {
```

2.3 La Vue

Pour la Vue j'ai tout d'abord utilisais le moteur de template Thymeleaf qui est très extensible et nous permet de définir nos propres ensembles d'attributs de modèle (ou même de balises) avec les noms de notre choix, en évaluant les expressions souhaitées dans la syntaxe souhaitée et en appliquant la logique souhaitée. Cela ressemble plus à un *framework de moteur de template*. Par la suite j'ai utilisé Bootstrap afin de faciliter la responsivité (moyen de calibrer la vue selon n'importe quel écran utilisé) et le placement des éléments de la vue.

2.3.1 Les Syntaxe d'expressions et attributs de base Thymeleaf

La plupart des attributs de Thymeleaf permettent à leurs valeurs d'être définies ou contenant des *expressions*, que nous appellerons des *expressions standard* en raison des dialectes dans lesquels elles sont utilisées. Celles-ci peuvent être de cinq types:

- **\${...}** : Expressions variables.
- ***{...}** : Expressions de sélection.
- **#{...}** : Expressions de message (i18n).
- **@{...}** : Expressions de lien (URL).
- **~{...}** : Expressions de fragment.

Thymeleaf utilise aussi des attributs fondamentaux du dialecte standard :

- **th:text** : qui remplace simplement le corps d'une balise
- **th:each** : qui répète l'élément dans autant de fois que spécifié par le tableau créant également une variable interne pour l'élément itération avec une syntaxe équivalente à celle d'une expression Java *foreach*.
- **th:action** : est utilisé pour fournir l'URL d'action du formulaire
- **th:object** : est utilisé pour spécifier un objet auquel les données de formulaire soumises seront liées
- **Et bien d'autres** (*th: switch, th: case, th: field,.....*)

Nous devons utiliser ces expressions et ces attributs pour que celle-ci soient lues dans un fichier modèle et le combine avec des objets Java pour générer (generate) un autre document.

```
<!-- Début pour personnalisation -->
<form th:method="post" th:action="ajouterCollegue">
    Civilité : <select class="browser-default custom-select"
        name="civilité">
        <option selected value="">Selectionner civilité</option>
        <option value="Monsieur">Monsieur</option>
        <option value="Madame">Madame</option>
    </select>
    Nom : <input type="text" name="nom" class="form-control" required /> <br>
    Prenom : <input type="text" name="prenom" class="form-control" required /> <br>
    Date Naissance : <input type="date" name="dateNaissance" class="form-control" required /> <br>
    Adresse : <input type="text" name="adresse" class="form-control" required /> <br>
    Numero Telephone : <input type="text" name="telephone" class="form-control" required /> <br>
    Email : <input type="email" name="mail" class="form-control" required /> <br>
    <br>

    <div class="tile-footer">
        <button class="btn btn-primary" type="submit" value="Ajouter">Ajouter</button>
    </div>
</form>
```

2.3.2 Le placement de Bootstrap.

Les conteneurs dans Bootstrap

Les conteneurs sont les éléments de présentation les plus élémentaires de Bootstrap et sont **indispensables pour l'utilisation du système de grille par défaut**. Nous devons choisir parmi un conteneur sensible à largeur fixe (max-width) ou fluid (signifiant qu'il est à 100% de la largeur max de l'écran tout le temps).

Bien que les conteneurs *puissent* être imbriqués, la plupart des mises en page ne nécessitent pas de conteneur imbriqué.

Tous ceci appel des fichiers CSS qui renvoie les informations du conteneur (tailles, largeur, couleurs, ect...)

Exemple d'appel du CSS en Html :

```
<!-- Sidebar toggle button-->
<a class="app-sidebar toggle" th:href="@{#}" data-toggle="sidebar" aria-label="Hide Sidebar"></a>
```

Dans le fichier CSS voilà ce que nous trouvons en exemple.

```
13468 @media (min-width: 768px) {
13469     .app.sidenav-toggled .app-content {
13470         margin-left: 0;
13471     }
13472     .app.sidenav-toggled .app-sidebar {
13473         left: -230px;
13474     }
13475     .app.sidenav-toggled .app-sidebar__overlay {
13476         visibility: hidden;
13477     }
13478 }
```

2.4 Le Contrôleur

2.4.1 Le REST avec Spring

REST (representational state transfer) est un style d'architecture logicielle définissant un ensemble de contraintes à utiliser pour créer des services web. Les services web conformes au style d'architecture REST, aussi appelés services webRESTful, établissent une interopérabilité (Possibilité de communication entre deux ou plusieurs systèmes, appareils ou éléments informatiques.) entre les ordinateurs sur Internet.

REST adhère aux préceptes du Web, y compris son architecture, ses avantages et tout le reste. Ce n'est pas une surprise si son auteur, **Roy Fielding**, était impliqué dans une douzaine de spécifications qui régissent le fonctionnement du Web.

Quel est son avantage?

Le Web et son protocole principal, HTTP, fournissent une pile de fonctionnalités:

- GET : Accède à une ressource
- POST : Ajoute une ressource
- DELETE : Supprime une ressource
- PUT : Met à jour une ressource complète en la remplaçant par une nouvelle version (99% des cas).

Exemple : Nous avons des itinéraires pour chaque opération (@GetMapping, @PostMapping, @PutMapping et @DeleteMapping, correspondant aux appels HTTP GET, POST, PUT et DELETE).

```
@GetMapping("/ajouterSection")
public String ajouterSection(SectionDto sectionDto) {
    return "formSection";
}

@PostMapping("/ajouterSection")
public String ajouterSection(Model model, @ModelAttribute("sectionDto") @Valid SectionDto sectionDto, BindingResult bindingResult) {
    if (bindingResult.hasErrors()) {
        return "formSection";
    }
    sectionService.ajouterSection(sectionDto);
    return "redirect:listeSections";
}
```

REMARQUE : il est utile de lire chaque méthode et de comprendre ce qu'elle fait. En exemple celle-ci font :

GetMapping : appel le formulaire Ajouter section pour que le client aperçoit la vue.

PostMapping : Envoie les informations remplies dans le formulaire « ajouter section » par le client.

2.5 Le Model

Le Model est la seule partie du MVC qui communique avec la base de données, il organise les données et les assemble pour qu'elles puissent ensuite être traitées.

Les Entité et les POJO

Les entités dans les spécifications de l'API Java Persistence permettent d'encapsuler les données d'une occurrence d'une ou plusieurs tables. Ce sont de simples POJO (Plain Old Java Object). Un POJO est une classe Java qui **n'implémente aucune interface particulière ni n'hérite d'aucune classe mère spécifique**.

Un objet Java de type POJO mappé vers une table de la base de données grâce à des méta data via l'API Java Persistence est nommé bean entité (Entity bean).

Un bean entité doit obligatoirement avoir un constructeur sans argument et la classe du bean doit obligatoirement être marquée avec l'annotation @javax.persistence.Entity ou être importée par javax.persistence.Entity. Ceci possède un attribut optionnel nommé name qui permet de préciser le nom de l'entité dans les requêtes. Par défaut, ce nom est celui de la classe de l'entité.

En tant que POJO, le **bean entity** n'a pas à implémenter d'interface particulière mais il doit respecter les règles de tous Java beans :

- Être déclaré avec l'annotation `@javax.persistence.Entity` ou importer le `javax.persistence.Entity`
- Posséder au moins une propriété déclarée comme clé primaire avec l'annotation `@Id`

Le bean entity est composé de propriétés qui seront mappées sur les champs de la table de la base de données sous-jacente. Chaque propriété encapsule les données d'un champ d'une table. Ces propriétés sont utilisables au travers de simples accesseurs (getter/setter).

Une propriété particulière est la clé primaire qui sert d'identifiant unique dans la base de données mais aussi dans le POJO. Elle peut être de type primitif ou de type objet. La déclaration de cette clé primaire est obligatoire.

```
package fr.ols.webapp.beans;

import java.io.Serializable;
import java.sql.Date;
import java.util.ArrayList;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Embedded;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.SequenceGenerator;

import fr.ols.webapp.dto.AdherentDto;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Adherent implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(generator = "adherent_generator")
    @SequenceGenerator(name = "adherent_generator", s
    private Long id;
    private String civilite;
    private String nom;
```

Les annotations du bean entity :

Les annotations peuvent être divisées en deux catégories, les annotations de mapping logique (nous permettant de décrire le modèle objet, les associations de classe, etc) et les annotations de mapping physique (décrivant le schéma physique, les tables, les colonnes, les index, etc).

Voici, en exemple, les définitions des annotations utilisé pour l'image ci-dessus :

@Entity : déclare la classe comme un entity bean

@Id : déclare la propriété identifiant de cet entity bean.

@GeneratedValue : indique que la clé primaire est générée de façon automatique lors de l'insertion en base de donnée. Sans cette annotation, la valeur de l'identifiant de la clé primaire doit être affecté avant .Elle est utilisée avec une autre annotation **@Id** qui permet de mapper une clé primaire sur un champ unique.

@SequenceGenerator : Définit un générateur de clé primaire qui peut être référencé par son nom lorsqu'un élément générateur est spécifié pour l'annotation **GeneratedValue** . Un générateur de séquence peut être spécifié sur la classe d'entité ou sur le champ ou la propriété de clé primaire. La portée du nom du générateur est globale pour l'unité de persistance (pour tous les types de générateurs).

@AllArgsConstructor : génère un constructeur avec 1 paramètre pour chaque champ de votre classe.

@NoArgsConstructor : va générer un constructeur sans paramètre.

Le JpaRepository et Spring Data :

Spring Data offre des facilités pour écrire le code d'accès aux données. Le module JPA de Spring Data contient un espace de noms personnalisé permettant de définir des beans de référentiel. Il contient également certaines caractéristiques et attributs d'éléments spécifiques à JPA.


```

package com.example.demo.dao;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.example.demo.entity.ColleagueEntity;

@Repository
public interface ColleagueDao extends JpaRepository<ColleagueEntity, Long> {

    public void save(Long id);

    // public Adherent rechercherAdherentParNom(String string);

}

```

Spring Data permet d'écrire des requêtes à partir des noms de méthode : vous écrivez la méthode avec certains mots-clef, et il se charge de traduire ce nom de méthode en requête puis de l'exécuter au moment voulu, comme un grand :

```

@Override
public void deleteById(Long id) {
    colleagueDao.deleteById(id);
}

```

Exemple de suppression d'un collègue.

Pour bénéficier d'opérations CRUD de base, il suffit donc tout simplement d'étendre une interface de Spring Data pour y avoir accès.

Connexion de la base de données avec mon API :

En utilisant, dans les dépendances, spring-boot-starter-data-jpa et postgresql nous allons pouvoir générer nos tables, transformer nos classes en entité gérée par JPA (vue juste avant). De plus l'avantage est que la base de données est complètement auto-configurée par Spring Boot.

```

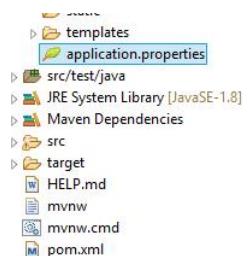
</dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jersey</artifactId>
    </dependency>
<!--        <dependency -->
<!--            <groupId>org.springframework.boot</groupId> -->
<!--            <artifactId>spring-boot-starter-security</artifactId> -->
<!--        </dependency> -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <scope>runtime</scope>
    </dependency>

```

La configuration de DataSource est contrôlée par des propriétés de configuration externes dans `spring.datasource.*`.

Ensuite, nous devons compléter le fichier **application.properties** afin d'ajouter les configurations requise pour PostgreSQL.



```

20
21 spring.datasource.url=jdbc:postgresql://localhost:5433/olabase
22 spring.datasource.username=postgres
23 spring.datasource.password=josse
24 spring.datasource.driver-class-name=org.postgresql.Driver
25 spring.datasource.initialization-mode=always
26
27 # The SQL dialect makes Hibernate generate better SQL for the chosen database
28 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
29 spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
30
31 # Hibernate ddl auto (create, create-drop, validate, update)
32 spring.jpa.hibernate.ddl-auto = create
33 spring.jpa.show-sql = true

```

3. Conception de La Base De Données :

Tout d'abord, une base de données bien conçue permet à ses utilisateurs d'accéder à des informations essentielles.

Une base de données bien structurée doit :

- libère de l'espace en éliminant les données redondantes ;
- préserve l'exactitude et l'intégrité des données ;
- permet d'accéder efficacement aux données.

Pour concevoir une base de données efficace et utile, vous devez suivre le bon processus, qui comprend les phases suivantes :

1. Analyse des besoins, c'est-à-dire l'identification de l'objet de votre base de données
2. Organisation des données en tables
3. Spécification des clés primaires et analyse des relations
4. Normalisation des tables

3.1 Les outils utilisés

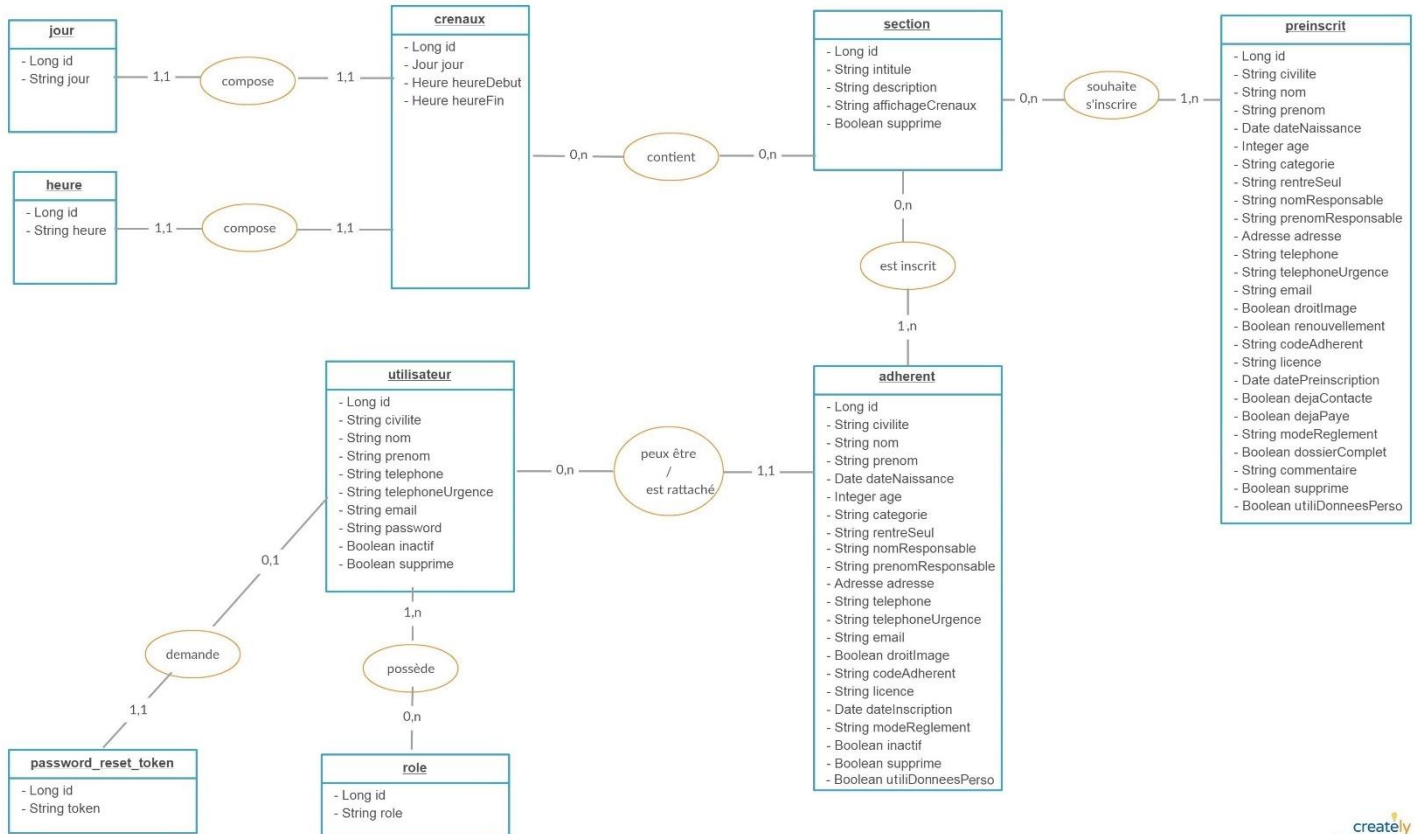
Les outils pour la conception de la base de données que j'ai utilisé sont :

- PostgreSQL
- PgAdmin
- whiteStarUml

3.2 MCD

Dans la méthodologie Merise destinée à créer des bases de données, il y a des outils dédiés aux traitements et aux données. Le **MCD (Modèle Conceptuel des Données)** est un des outils majeurs concernant les données.

Le MCD est une représentation graphique de haut niveau qui permet facilement et simplement de comprendre comment les différents éléments sont liés entre eux à l'aide de diagrammes codifiés dont les éléments suivants font partie :



- Les entités (1 rectangle = 1 objet) ;
- Les propriétés (la liste des données de l'entité) ;
- Les relations qui expliquent et précisent comment les entités sont reliées entre elles (les ovales avec leurs « pattes » qui se rattachent aux entités) ;
- Les cardinalités (les petits chiffres au dessus des « pattes »).

Cet outil permet d'échanger entre informaticiens et non-informaticiens sur l'outil à informatiser. On peut ainsi à partir d'un MCD valider et préciser des règles qui s'appliqueront à la future base de données (d'après le MCD en image ci-dessus) :

- Zéro ou plusieurs préinscrit souhaite s'inscrire Zéro ou plusieurs section.
- 1 ou plusieurs adhérent et inscrit a 0 ou plusieurs section

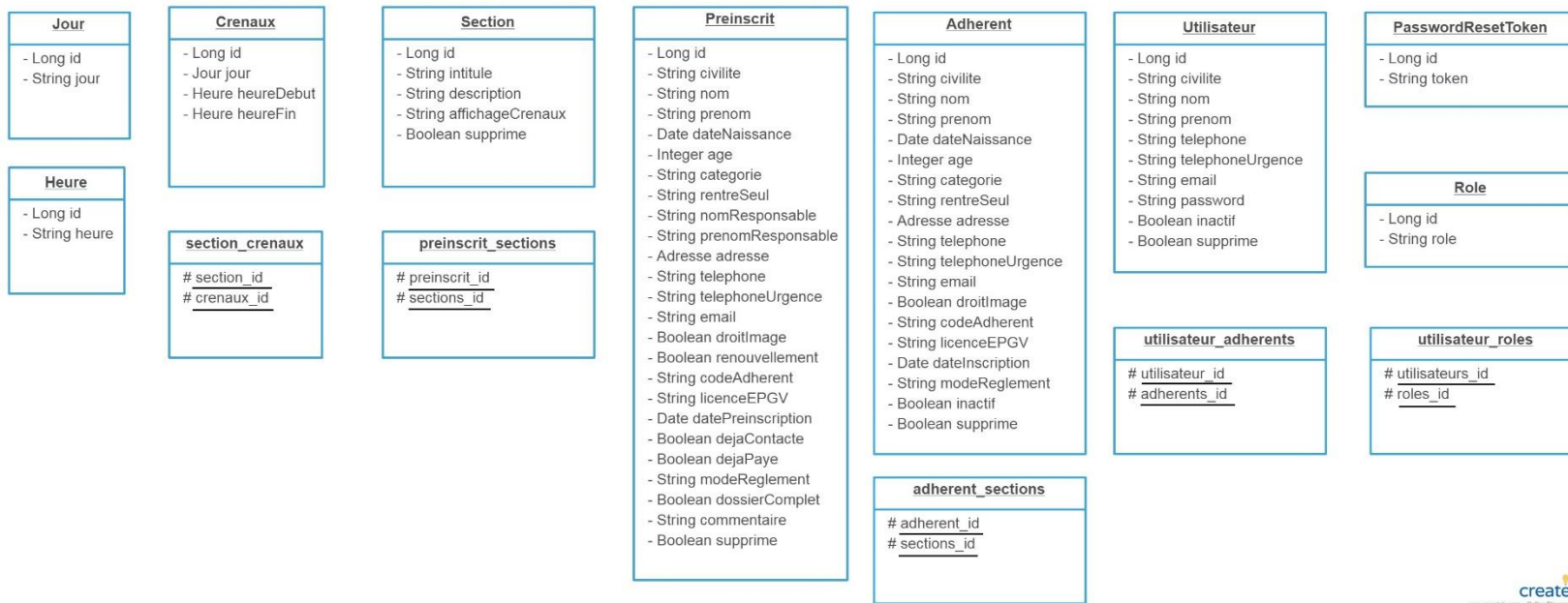
3.3 Transformation du MCD en MLD (Model Logique de Données)

Pour changer notre un **MCD** il faut connaitre les notions **MLD** à travers les Cardinalités.

C'est à dire :

- **L'entité ayant la cardinalité de type 1,1 ou 0,1** absorbe l'identifiant de l'entité la plus forte (0, n ou 1, n). *Cet identifiant est alors appelé la clé étrangère.*
- **Cardinalité (1,1) et (1,1)** Dans ce cas on choisit l'une des entités qui hérite l'ensemble des attributs de l'autre entité. En effet, la clé primaire restant celle de l'entité.
- **Relation (1, n) et (0,1)** Dans ces relations, ce sont les maximales qui comptent. Ceci est possible si l'entité dépend de l'autre entité et si elle est partiellement identifiée par cette dernière. Dans l'exemple suivant, l'entité client dépend de l'entité commande.
- Dans le MLD de la relation A et B de cardinalité (1,1) et (1,1), la clé étrangère migre : pas de migration.
- Dans le MLD de la relation A et B de cardinalité (0,1) et (0,n), la clé étrangère migre : vers A.
- Dans le MLD de la relation A et B de cardinalité (0,1) et (1,1), la clé étrangère migre : vers B, vers B uniquement.

Voici le résultat :

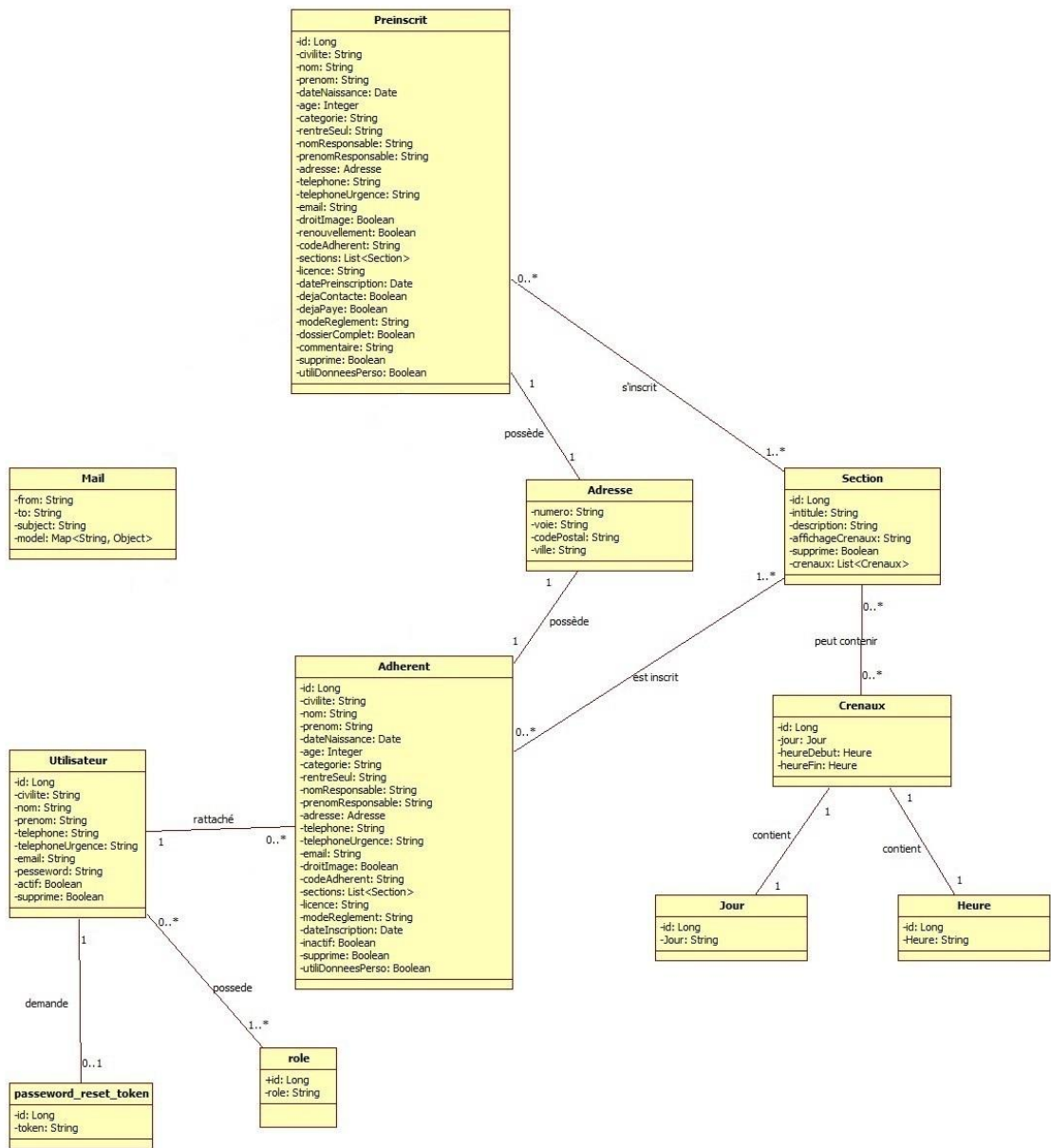


3.4 Diagramme de Classes

Le Diagramme de classes est considéré comme le plus important de la modélisation orientée objet, **il est le seul obligatoire lors d'une telle modélisation.**

Les principaux éléments de cette vue statique sont les classes et leurs relations : association, généralisation et plusieurs types de dépendances, telles que la réalisation et l'utilisation. Dans un Diagramme de classe une « classe » est représentée par un rectangle divisé verticalement entre trois à cinq compartiments. Le premier indique le **nom** de la classe, le deuxième **ses** compartiments. Le

premier indique le **nom** de la classe, le deuxième **ses attributs**, et le troisième **ses opérations**.(falcultatif)



Conclusion :

Durant ces 3 mois de Stage j'ai appris à concevoir, dans son intégralité, une application web. Les parties gestion des préinscriptions, des adhérents, des notifications par e-mails et Sportive sont fonctionnels et répond au critère de la cliente. Pour la suite, le projet à était rattaché un autre stagiaire, qui devras répondre aux attentes et nouvelle demande de la cliente. (Finalisation du système événementiel, rattacher la fonctionnalité gestion de tarif.) J'ai appris aussi as concevoir, en parallèle, un front dynamique en prenant connaissance du CMS.