1 - Commentaire suite à la correction des projets précédents

Vous trouverez ci-dessous quelques remarques générales et non exhaustives notées lors de la correction des projets des années précédentes. Je laisse chacun(e) faire le bilan des points qui ont été ou non respectés dans vos travaux, et donc des évolutions à envisager.

Remarque préliminaire : L'organisation du code doit respecter les concepts abordés en UML notamment sur le rôle/responsabilité de chaque classe, mais aussi l'éventuelle utilisation de design pattern, …

1.1 - Rapport

La qualité des rapports rendus est très variable, et se décline entre un simple fichier texte d'une dizaine de ligne à un rapport complet, avec explication et copie d'écran de la version déployée, …

Il est important de vous améliorer dans cet exercice et d'être en mesure de présenter correctement votre travail, avec une rédaction claire exprimée dans un français correct. Le respect des termes techniques et la pertinence des informations fournies, attestent de votre compréhension des contenus présentés, …. au même titre que pour un entretien professionnel.

Exemple : Le choix technique ne peut se résumer à "cette solution permet d'écrire moins de lignes de code .... "!!!!

Lors de la rédaction d'un rapport, un étudiant de master est censé avoir assimilé les 2 notions suivantes :

- Un collègue de même niveau doit être capable d'exploiter efficacement mon code à l'aide des explications fournies et des commentaires insérés dans le code,
- Mon rapport me permet de persuader mon interlocuteur du niveau de qualité du travail fourni.

Rappel de la définition de qualité: Adéquation de l'élément (code, objet, ...) à répondre à l'expression des besoins définis lors de sa conception.

1.2 - Notices d'utilisation et de déploiement

Ces 2 notices complémentaires et pouvant être intégrées dans le rapport, doivent fournir les informations nécessaires à ces 2 opérations.

Un point souvent absent concerne la structure de la base de données (script de création de la base, ...), indispensable pour pouvoir assurer un déploiement réel et rapide !

1.3 - Projet maven

L'intérêt de ce format réside dans la gestion des dépendances et permet de simplifier le travail collaboratif, en permettant à tous de travailler sur les mêmes versions.

Également pratique pour réaliser des tests, il faut néanmoins penser à "nettoyer" ce contenu afin de ne conserver que les éléments nécessaires.

Exemple: Ne pas conserver 3 drivers de SGBD différents: mysql, postgresql, mongodb, ...!

1.4 - Pages HTML

- Privilégier les outils permettant de travailler directement avec des formats HTML
  Remarque: Fichier HTML pour pages statiques, ou JSP, Thymeleaf, .. pour pages dynamiques
- Limiter l'utilisation de bibliothèques externes non indispensables (bootstrap, jquery, …)
  Exemples :
  - Lien bootstrap externe pour une mise en page de tableau dont l'effet peut être obtenu avec quelques (<10) règles CSS basiques!
  - ou lien avec JQuery ... non utilisé dans le fichier!
- Stocker les images en local
  Remarque: Il est impératif de proscrire les liens vers des sites externes, ou réseaux sociaux de partage d'images,…

1.5 - Modèle

Le respect du modèle complet du projet conformément à la description du format qui vous a été fournie, est un impératif dans la réalisation du projet.
Le contrôleur doit gérer directement le modèle, sans avoir à se soucier de son organisation. La classe XXX (package model) est en charge de l'organisation interne du modèle !

1.6 - DAO

Plusieurs solutions peuvent être utilisées, tant au niveau du SGBD(R) ou de la couche ORM, ... à condition toutefois de respecter au final le modèle et les bonnes pratiques de codage.

- Il est possible d'avoir une différenciation entre le modèle XML, et l'organisation des données DAO, à condition de prévoir les méthodes permettant de réaliser une conversion stricte dans les 2 sens
  Exemple : Plusieurs formats sont autorisés pour les numéros de téléphone. Il est néanmoins possible de n'utiliser qu'un seul format pour la sauvegarde.

1.7 - Contrôleur

Le contenu des contrôleurs doit être organisés, notamment du fait des gestion différenciées entre le retour de pages HTML ou XML!

Quelques erreurs **à ne pas** commettre:

- Ne pas définir plusieurs méthodes distinctes pour gérer la même route, mais avec des méthodes distinctes GET / POST.
- Ne pas réaliser la création de la page HTML dans le code par génération d'une chaîne de caractères : Il faut utiliser des pages HTML déjà créées dans le cas de pages statiques, ou avec JSP, Thymeleaf, ... dans le cas de pages dynamiques c'est à dire dont une partie du contenu est modifiable.
  De même le retour des ces pages s'effectue en 1 seule commande!
- Ne pas recréer les fonctionnalités déjà existantes : 200 lignes de code pour gérer (parfois partiellement) un insert, alors qu'une couche DAO existe et doit être obligatoirement utilisée, ce qui d'ailleurs est une meilleure garantie de cohérence du modèle.

1.8 - Gestion des erreurs

Bien qu'il s'agisse d'un projet pédagogique, l'organisation du code doit respecter les concepts qui vous sont présentés dans votre formation, en respectant les patterns usuels (MVC, Singleton, ...), en gérant les erreurs (toutes les erreurs), ainsi que par un minimum de commentaires insérés dans le code.

1.9 - Tests

Cette partie a été largement ignorée, hormis un rendu postman généralement incomplet dans la majorité des cas. Les tests doivent également être réalisés sur la version déployée pour en valider le fonctionnement.
Les tests unitaires (junit) sont également utiles dans la version de développement.