

# Compte rendu de mini projet

Sujet : **QCM**

Enseignant : Sylvain JOYEAU

Torien TORIEN & Zeineb AMARA



[Lien GitHub du projet :](https://github.com/Torien-TORIEN/MiniProjet_Jakarta_EE)

[https://github.com/Torien-TORIEN/MiniProjet\\_Jakarta\\_EE](https://github.com/Torien-TORIEN/MiniProjet_Jakarta_EE)



# Sommaire

<b>Introduction.....</b>	<b>3</b>
<b>Description du projet.....</b>	<b>4</b>
<b>Spécifications fonctionnelles.....</b>	<b>5</b>
<b>1. API REST.....</b>	<b>6</b>
<b>2. Front End.....</b>	<b>8</b>
2.1 Les Filters.....	8
2.2 La Structure MVC (Modèles et Services).....	9
2.3 Les Servlets et les page JSP.....	10
2.4 Les captures des différentes vues de l'application.....	11
<b>Conclusion.....</b>	<b>15</b>



# Introduction

Dans le cadre de ce projet, nous avons consolidé l'ensemble des connaissances acquises en cours afin de réaliser un projet fonctionnel et utilisable. Notre objectif principal est de tirer le meilleur parti de ce que nous avons appris durant le cours sur Java Entreprise Edition. Ce projet vise à créer un site web permettant de réaliser des QCM en utilisant les différentes technologies proposées par JEE.

Ce projet se divise en deux sous-projets distincts. Le premier est une API REST(TestAPI) qui utilise JPA/DAO pour accéder à une base de données MySQL, ainsi que JAX-RS et JAXB, services web pour fournir les éléments nécessaires. Le second est le front-end(QCM\_FrontEnd), qui utilise des servlets/JSP pour traiter les requêtes des clients (navigateurs).



# Description du projet

Pour ce projet, nous avons défini quatre (04) entités principales :

- **Utilisateur :**

Il existe deux types d'utilisateurs :

- ❖ Les utilisateurs simples (rôle : **USER**) peuvent créer un compte, s'authentifier et accéder à la page d'accueil pour passer différents tests (QCM) proposés par notre site web. Ils peuvent également consulter leurs scores pour les tests effectués.
- ❖ Les administrateurs (rôle : **ADMIN**) ont les mêmes fonctionnalités que les utilisateurs simples, mais en plus, ils disposent de privilèges administratifs. Cela inclut la possibilité de réaliser des opérations CRUD sur les tests et les questions, en plus de passer les tests.

- **Test :**

Un test est conçu pour classer et organiser les questions. Chaque test est identifié par un ID et appartient à une catégorie spécifique. Il est composé de plusieurs questions, et lorsqu'un utilisateur décide de passer un test, dix (10) questions sont sélectionnées au hasard parmi celles associées à ce test dans la base de données. Cette approche garantit que chaque passation de test peut présenter des questions différentes, offrant ainsi une expérience unique à chaque utilisateur.

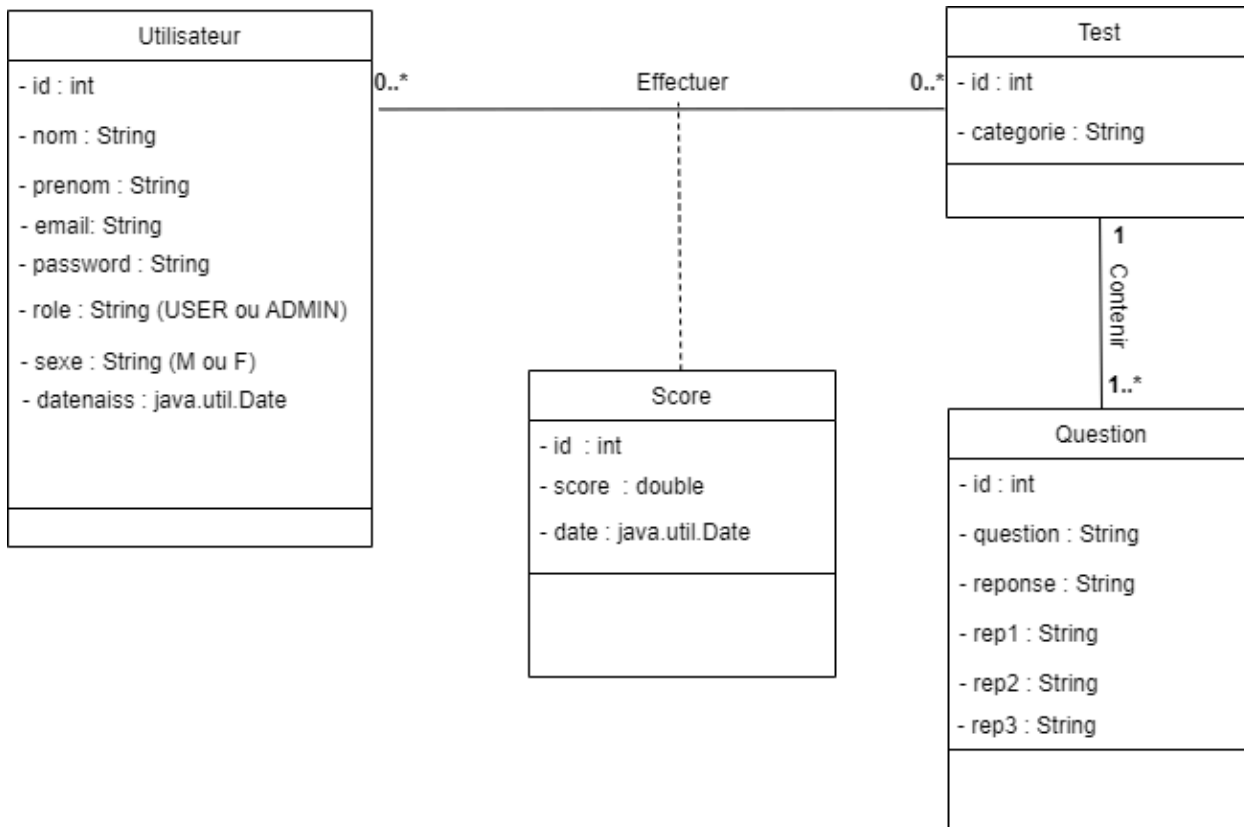
- **Question :**

Une entité Question se compose d'une question, d'une réponse correcte et de trois (03) réponses incorrectes. Lorsque la question est affichée, les quatre (04) réponses doivent être présentées dans un ordre aléatoire, afin de rendre le processus d'examen plus dynamique et stimulant pour l'utilisateur.

- **Score:**

Un utilisateur peut passer le même test plusieurs fois, car les questions sélectionnées peuvent varier d'une session à l'autre. Par conséquent, l'entité Score est formée par un ID, un score et la date à laquelle le test a été effectué. Chaque question ne peut avoir qu'une seule réponse correcte. Cependant, nous avons choisi de permettre aux utilisateurs de sélectionner plusieurs réponses.

Dans ce cas, si plus d'une réponse est sélectionnée, la question est considérée comme incorrecte et le score pour cette question est zéro.

**Figure 01** : Diagramme de classe

## Spécifications fonctionnelles

### Cas 1 : Utilisateur Simple (USER)

- Lorsque l'application est lancée, l'utilisateur est redirigé vers la page d'authentification où il peut se connecter ou créer un compte s'il n'en a pas déjà un. Le compte créé sera un compte utilisateur simple, sans privilèges administratifs.
- Une fois connecté, l'utilisateur est dirigé vers la page d'accueil où il peut effectuer des tests. Sur cette page, il peut également consulter ses scores précédents et sélectionner le type de test qu'il souhaite effectuer.
- La barre de navigation latérale permet à l'utilisateur de se déconnecter, de consulter ses scores et de gérer son compte.
- Après avoir sélectionné un test, dix (10) questions aléatoires de ce test sont présentées à l'utilisateur une par une. Le score est calculé à la fin en fonction des réponses



correctes. L'ordre d'affichage des quatre réponses possibles pour chaque question est aléatoire.

### Cas 2 : Administrateur (ADMIN)

- L'administrateur est un utilisateur spécifique créé dans la base de données avec le rôle "ADMIN".
- Une fois connecté, l'administrateur dispose d'un accès supplémentaire à la page d'administration, en plus des fonctionnalités disponibles pour les utilisateurs simples.
- L'administrateur peut afficher, modifier et supprimer tous les tests et questions. Ces fonctionnalités d'administration sont protégées par un filtre :
- Les utilisateurs non connectés ont uniquement accès à la page de connexion et de création de compte.
- Les utilisateurs connectés qui ne sont pas administrateurs n'ont pas accès à la page d'administration.
- Les utilisateurs connectés ont un accès restreint à la page de connexion et de création de compte.
- Les utilisateurs ont également la possibilité de se déconnecter, ce qui met fin à leur session et les redirige vers la page de connexion.

## 1. API REST

L'implémentation de la couche DAO dans le projet web dynamique a été un défi majeur pour nous, et nous avons rencontré plusieurs difficultés qui ont considérablement ralenti notre progression dans cette partie du projet.

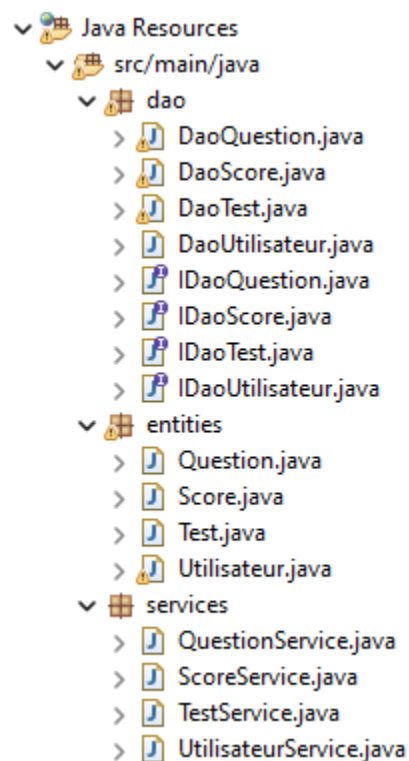
Tout d'abord, nous avons tenté d'intégrer le DAO dans le projet web dynamique, mais le serveur Tomcat ne parvenait pas à trouver le fichier persistence.xml. Nous avons alors créé un dossier "classes" dans le répertoire WEB-INF et y avons placé le fichier persistence.xml, mais cela n'a pas résolu le problème.

Ensuite, nous avons créé un projet Java classique distinct pour le DAO, que nous avons ensuite intégré dans le projet web dynamique. Cependant, le serveur a continué à rencontrer des problèmes de dépendances même après avoir ajouté toutes les dépendances du projet DAO dans le dossier lib du serveur Tomcat.



Pour surmonter ces problèmes de dépendances, nous avons décidé de transformer notre projet web dynamique en projet Maven après sa création. Nous avons simplement spécifié les dépendances dans le fichier pom.xml, et ajouté le fichier persistence.xml dans le dossier WEB-INF/classes et celles-ci ont été correctement interprétées par le serveur. Cette approche nous a permis de résoudre efficacement les problèmes liés à l'implémentation de la couche DAO.

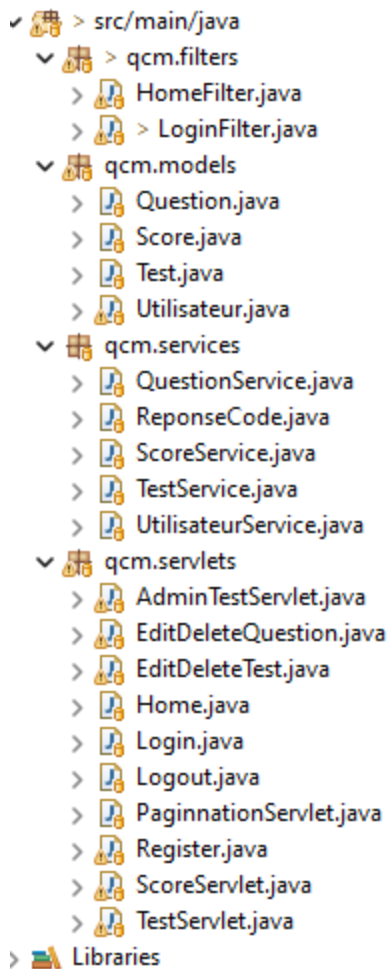
Ainsi, nous avons réussi à mettre en place les entités, les couches DAO et les services pour ce projet, en surmontant les obstacles rencontrés lors de l'intégration de la couche DAO dans le projet web dynamique.



**Figure 02** : Structure de projets API REST



## 2. Front End



Pour mener à bien ce projet, nous avons opté pour la création d'un projet Web dynamique afin d'implémenter les servlets et les JSP pour offrir une expérience utilisateur optimale. En plus de couvrir l'ensemble des sujets abordés en classe, notre objectif principal était de concevoir un site web attrayant et convivial.

Pour cela, nous avons développé quatre pages JSP (Login, Register, Home, Admin et Error) que nous avons placées dans le dossier WEB-INF pour des raisons de sécurité. Nous avons également mis en place des servlets pour traiter les requêtes des clients, des filtres pour restreindre l'accès à certaines pages, des services pour appeler l'API REST, et des modèles pour structurer efficacement les objets.

Dans notre démarche, nous avons utilisé des bibliothèques JAR telles que Jackson et JSON pour réaliser le mapping entre les classes de notre modèle et le format JSON à envoyer ou recevoir de l'API REST. Cette approche nous a permis de garantir une communication fluide et efficace entre notre application et l'API REST sous-jacente, tout en assurant une expérience utilisateur optimale sur notre site web.

**Figure 03** : Structure de projets Front End

### 2.1 Les Filters

Dans le cadre de ce projet, nous avons mis en place deux filtres (LoginFilter et HomeFilter) afin de réguler l'accès aux différentes pages en fonction de la session de l'utilisateur.

- Filtre de Connexion (LoginFilter) : Ce filtre supervise l'accès aux pages de connexion et de création de compte. S'il détecte qu'un utilisateur est déjà connecté, c'est-à-dire si la session contient déjà un attribut "user", l'accès à ces pages est refusé. L'utilisateur doit préalablement se déconnecter pour accéder à nouveau à ces fonctionnalités.





- Filtre d'Accueil (HomeFilter) : Ce filtre contrôle l'accès aux pages d'accueil et d'administration. Lorsqu'un utilisateur n'est pas authentifié, il se voit interdire l'accès à ces deux pages. De plus, s'il n'est pas administrateur, l'utilisateur est automatiquement redirigé loin de la page d'administration.

Ces filtres assurent une gestion efficace des servlets qui renvoient les pages JSP, en effectuant les redirections nécessaires en fonction du statut de l'utilisateur et de ses privilèges d'accès.

## 2.2 La Structure MVC (*Modèles et Services*)

Afin d'organiser notre projet de manière optimale, nous avons adopté une architecture MVC (Modèle-Vue-Contrôleur) pour garantir la clarté du code. Cette approche nous permet de séparer la logique métier de la présentation des données.

Nous avons donc créé deux packages distincts : "*models*" et "*services*", en plus des servlets et des filtres. Cette division nous permet de maintenir une structure de code modulaire et évite que les servlets ne deviennent trop volumineux et difficiles à comprendre.

Dans le package "*services*", chaque classe de service est associée à un modèle spécifique. Ces services agissent comme des intermédiaires entre les servlets et les appels à l'API REST. Ils encapsulent la logique métier et communiquent avec l'API pour effectuer les opérations nécessaires. Les services renvoient ensuite une réponse encapsulée dans un objet de type "**ReponseCode**", une classe que nous avons également créée pour structurer de manière cohérente les réponses de l'API.

Dans les servlets, nous construisons les objets à partir des requêtes des clients et faisons appel aux services appropriés pour effectuer les opérations requises. Nous traitons ensuite la réponse reçue du service et prenons les mesures nécessaires en fonction de son contenu. Cette approche favorise la réutilisation du code, la maintenabilité et la gestion efficace des interactions avec l'API REST.



## 2.3 Les Servlets et les page JSP

Dans notre projet, nous utilisons des servlets pour afficher les pages JSP, qui sont placées dans le dossier WEB-INF afin de restreindre l'accès direct des clients et de garantir une gestion sécurisée des pages.

Nous avons mis en place différentes servlets pour gérer les différentes pages JSP. Notre approche consiste à stocker les objets ou les variables nécessaires dans une session ou un cookie s'ils sont fréquemment requis, tels que l'utilisateur connecté, les listes de tests et les questions. En revanche, si un objet ou une variable est utilisé une seule fois lors de la requête du client, nous passons ces variables dans la requête (request) vers la JSP. Par exemple, lorsqu'un administrateur clique sur un bouton de modification, seul l'objet cliqué est transmis dans le formulaire pour permettre à l'administrateur de le modifier.

Nous utilisons également des variables pour dynamiser la page. Comme nous disposons de seulement quatre pages JSP, nous évitons d'afficher toutes les rubriques simultanément pour ne pas surcharger l'écran de l'utilisateur. Pour offrir à l'utilisateur l'impression de naviguer sur différentes pages, nous avons mis en place une barre latérale de menus permettant de choisir le contenu à afficher. Toutefois, nous maintenons une cohérence dans l'expérience utilisateur ; par exemple, pendant un QCM, l'utilisateur ne peut pas accéder au bouton pour consulter ses scores, et vice versa.

De plus, nous avons créé une page dédiée à l'affichage des erreurs spécifiques au serveur qui ne sont pas liées à l'action des utilisateurs. Cependant les erreurs et notifications survenant lors d'actions sont affichées directement sur la page de client elle-même, telles qu'un mot de passe incorrect ou une adresse e-mail lors de la connexion, une saisie incorrecte ou invalide, ou encore un e-mail redondant lors de la création du compte. De plus, des notifications sont également affichées en cas de succès ou d'échec lors d'une opération d'administration, assurant ainsi une expérience utilisateur complète.



## 2.4 Les captures des différentes vues de l'application

- La page de connexion:

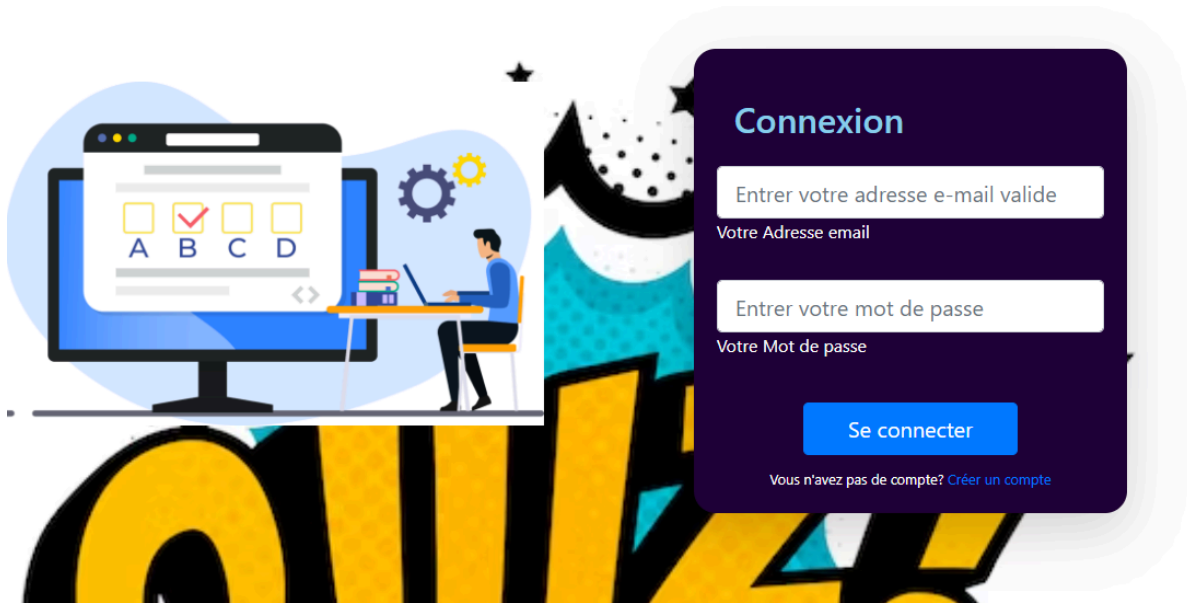


Figure 04 : Capture de la page login

- Si l'utilisateur n'a pas de compte:

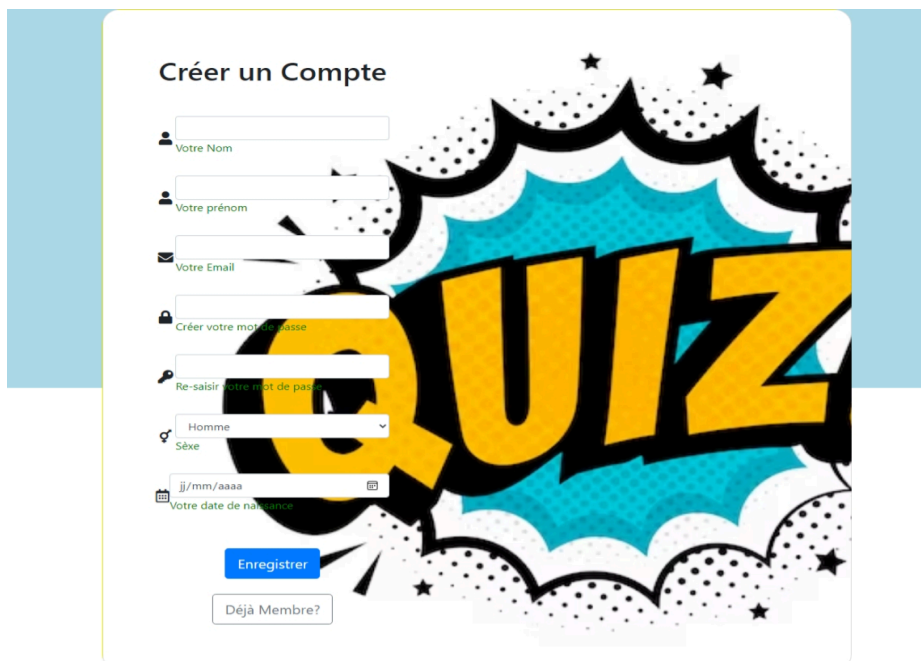


Figure 05 : Capture de la page de création de compte

- Après l'authentification:

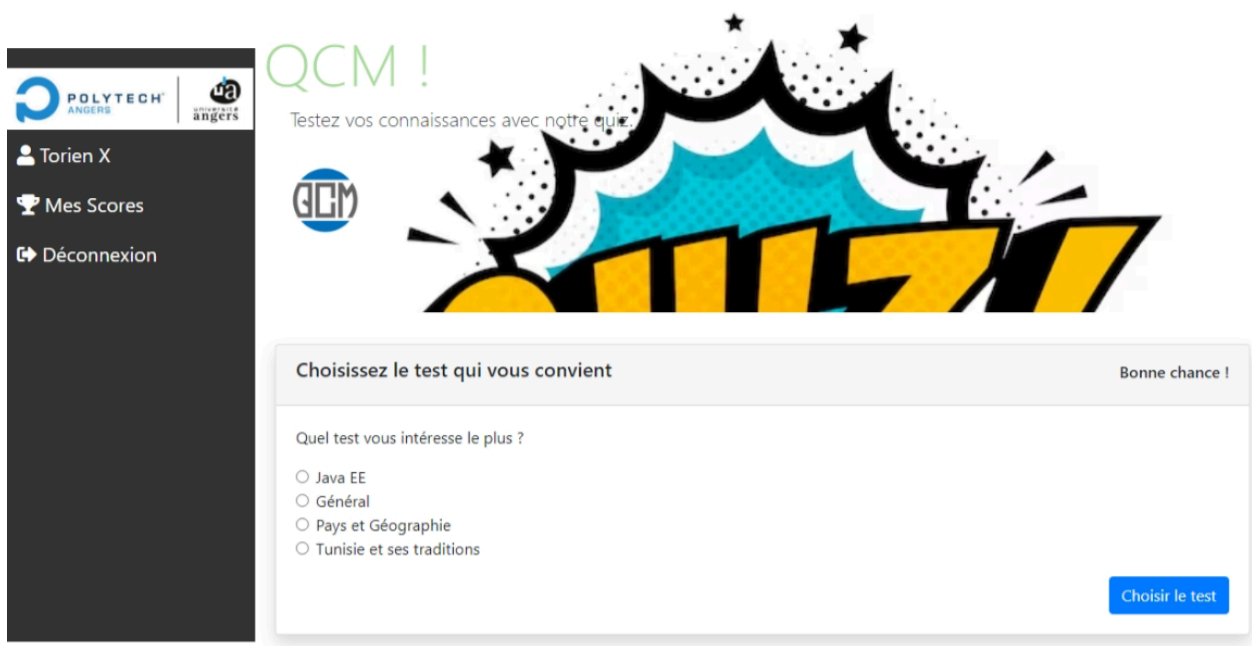


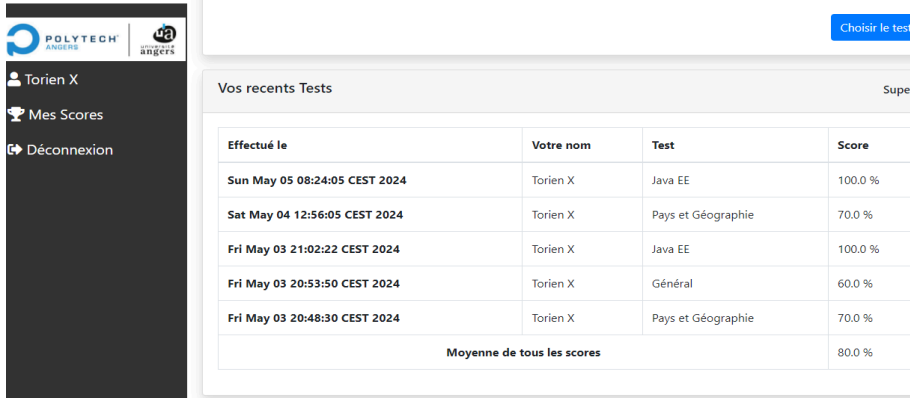
Figure 06 : Capture de la page d'accueil

- L'utilisateur peut choisir un test et effectuer le QUIZ:



Figure 07 : Page lors de QCM , question par question

- L'utilisateur peut voir ses scores:



○ Tunisie et ses traditions

Choisir le test

Vos recents Tests Super

Effectué le	Votre nom	Test	Score
Sun May 05 08:24:05 CEST 2024	Torien X	Java EE	100.0 %
Sat May 04 12:56:05 CEST 2024	Torien X	Pays et Géographie	70.0 %
Fri May 03 21:02:22 CEST 2024	Torien X	Java EE	100.0 %
Fri May 03 20:53:50 CEST 2024	Torien X	Général	60.0 %
Fri May 03 20:48:30 CEST 2024	Torien X	Pays et Géographie	70.0 %
Moyenne de tous les scores			80.0 %

**Figure 08** : Capture d'affichage des scores pour l'utilisateur

- Si c'est l'admin qui est connecté:



QCM !

Testez vos connaissances avec notre quiz.

Choisissez le test qui vous convient Bonne chance !

Quel test vous intéresse le plus ?

☐ Java EE

☐ Général


☐ Pays et Géographie

☐ Tunisie et ses traditions

Choisir le test

**Figure 09** : Capture de la page d'accueil pour l'administrateur

- L'admin peut voir toutes les questions et les modifier ou les supprimer:



ADMIN Page !

Testez vos connaissances avec notre quiz.

Sélectionner un Test

Java EE

Afficher les questions

**Figure 10** : Capture de la page d'admin

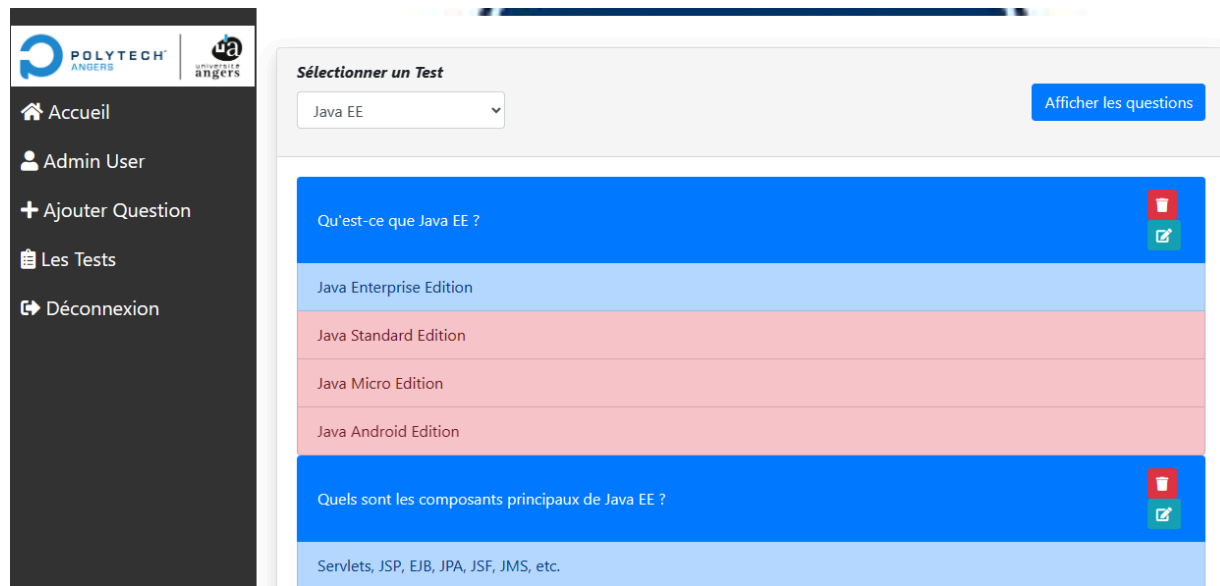


Figure 11 : Capture de la liste des questions pour l'admin

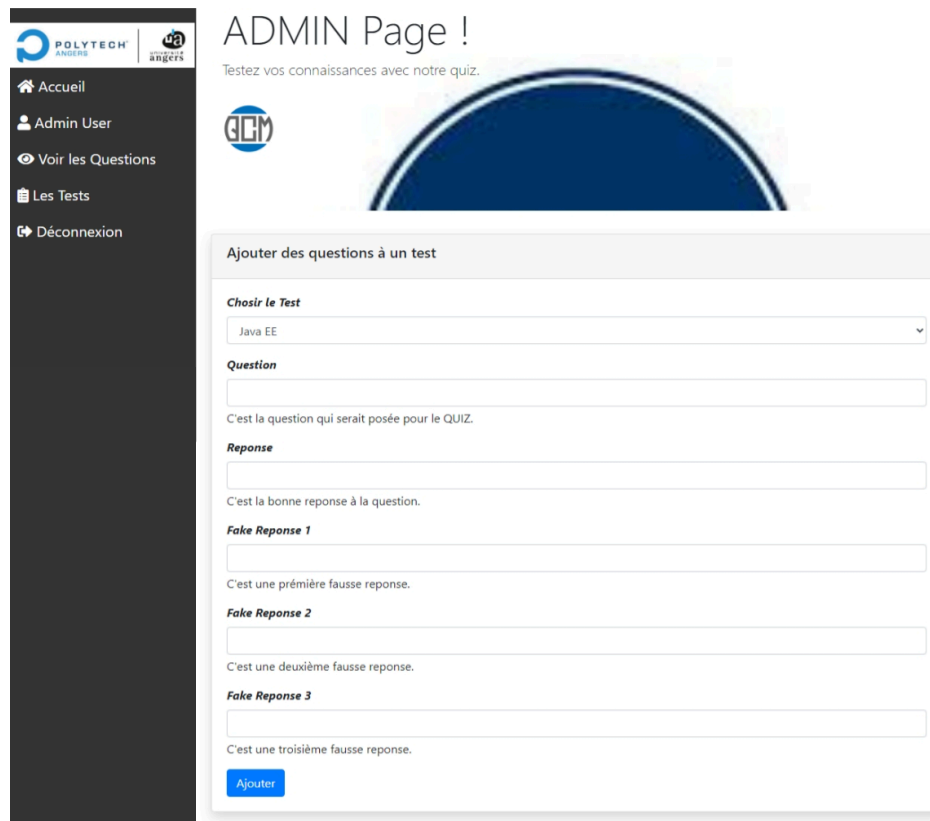


Figure 12 : Capture de la page d'ajout des questions pour l'admin



- L'admin peut ajouter, modifier ou supprimer un test:

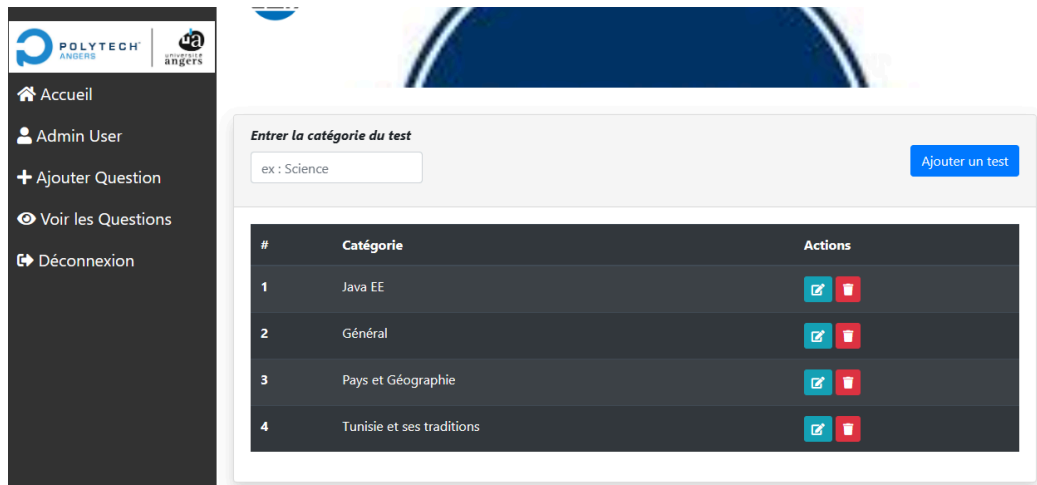


Figure 13 : Capture de page d'ajout , de la liste des tests

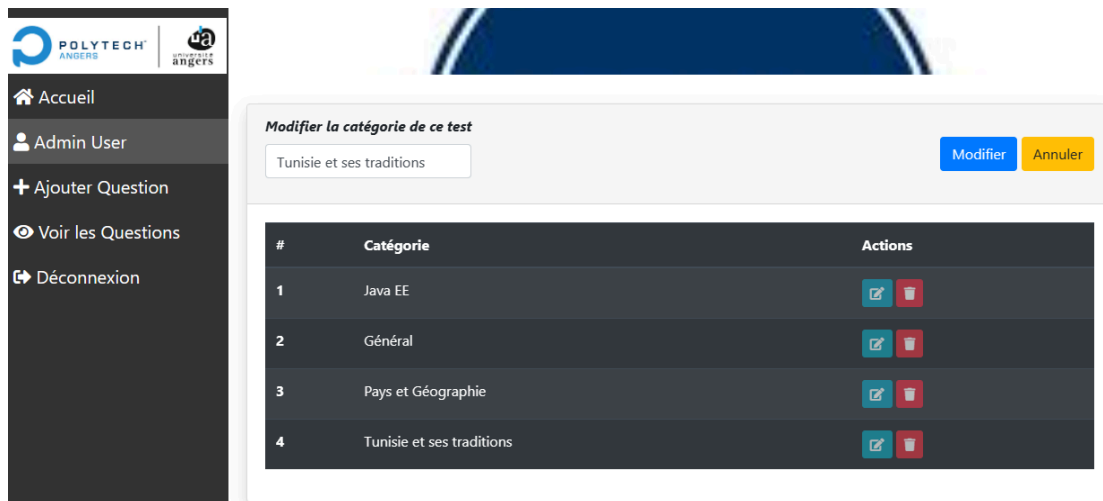


Figure 14 : Capture de page de modification de test

NB : Lors de la modification les boutons editier et supprimer sont de la liste sont désactivés

## Conclusion

Grâce aux différentes approches que nous avons utilisées dans ce projet, nous avons pu créer un site web dynamique et convivial, offrant ainsi une expérience utilisateur très satisfaisante. Nous avons également mis en place, exploité, combiné et même enrichi toutes les techniques apprises durant les travaux pratiques de JEE. Ce projet nous a permis d'acquérir des compétences incomparables en Java Enterprise Edition.