ÉCOLE SUPÉRIEURE DES SCIENCES ET DE LA TECHNOLOGIE DE HAMMAM SOUSSE			
Matière : Java EE	**Date :** Mars 2019	**Durée :** 2 Heures	**EXAMEN**
Enseignant : Ala Eddine KHARRAT	**Niveau :** 3LFSI	**Session :** Principale	
Nombre total de pages : 7 pages	**Tous les documents sont autorisés sauf les appareils électroniques.**		

Un professeur de **SVT** (**S**cience de la **V**ie et de la **T**erre) veut trouver une idée originale et motivante permettant d'entraîner et préparer, au mieux, ses élèves du **BAC** à affronter tous les exercices de type **QCM**. A cet égard, il vous demande de lui aider à créer un **site web dynamique** en ***Java EE*** qui répond bien à son besoin.

Dans l'intention d'aider le professeur à créer un site web parfaitement simple à utiliser et accessible par tout les étudiants, le présent sujet est organisé en deux parties : La première partie ***« Présentation du projet »*** éclaircie toutes les informations importantes concernant le projet (la base de données, les ressources, etc.). La deuxième partie ***« Développement du projet »*** admet **4 exercices** qui répondent au besoin du professeur. Les réponses de ces exercices doivent être rédigées sur votre feuille d'examen qui doit être remise à la fin de l'épreuve.

I. Présentation du projet

Présentation de la base de données

La base de données englobe 4 tables : ***« Students »***, ***« Marks »***, ***« Questions »*** et ***« Categories »***. Ces dernières sont représentées de la manière suivante :

- **Students** (idStudent, fullName, login, password)
- **Marks** (#idStudent, date, mark)
- **Questions** (idQuestion, #idTopic, question, proposition1, proposition2, proposition3, proposition4, solution)
- **Topics** (idTopic, wording)

Tables	Descriptions des tables
Students	Contient les informations de chaque étudiant : son identifient, son nom complet, son login et son mot de passe.
Marks	Contient les notes des tests passés par les étudiants.
Questions	Contient plus que **500 questions** ; chaque question est définie par un identifiant unique, une catégorie (le thème), la question à poser, les quatre propositions et la solution. Une solution peut qu'une seule proposition correcte (***"A"***, ***"B"***, ***"C"*** ou ***"D"***).
Topic	Contient les thèmes étudiés au cours de l'année. Chaque thème est représenté par un identifiant unique et un libellé qui décrit le titre du thème.

Les ressources du projet

- Le projet respecte les normes du pattern **MVC** de **Java EE**. Il est constitué de **4 packages** : ***« controllers »***, ***« dao »***, ***« models »*** et ***« extra »***.

Packages	Descriptions des packages
controllers	Contient les servlets du projet : ***« Login.java »***, ***« Logout.java »***, ***« Result.java »***, ***« Test.java »*** et ***« Profil.java »***.
models	Contient des JavaBeans qui correspondent aux tables de la base de données : ***« Student.java »***, ***« Mark.java »***, ***« Question.java »*** et ***« Topic.java »***.
dao	Contient 4 interfaces java : ***« StudentDAO.java »***, ***« MarkDAO.java »***, ***« QuestionDAO.java »*** et ***« TopicDAO.java »***. Chaque interface possède une implémentation Java sous la forme ***« xxxImpl.java »*** où ***« xxx »*** désigne le nom de l'interface.
extra	Contient deux classes : - ***« DataBaseConnection.java »*** : cette classe permet la connexion et la déconnexion à la base de données. Cette classe sera appelée dans chaque constructeur d'un **DAO**. - ***« Strings.java »*** : contient un ensemble de messages de type ***« String »*** qui peuvent être utilisés dans le projet.

➢ Les méthodes de chaque interface :

Interfaces	Méthodes
StudentDAO	`public int verifyLogin(String login, String password);` `public Student getStudentByLogin(String login);`
MarkDAO	`public List<Mark> getMarksByUserId(int id);` `public void addNewMark (Mark mark);`
QuestionDAO	`public Question getQuestionById(int id);`
TopicDAO	`public Topic getTopicById(int id);`

➢ Contenu du fichier ***« Strings.java »*** :

```
public class Strings {

   public static String ERROR_LOGIN = "Login ou mot de passe incorrect !";

   public static String ERROR_DB_PROBLEM = "Désolé un problème technique est survenu.
   Veillez réssayer plus tard.";

   public static String RESULT_MSG = "Votre note finale est :";

   public static String REDIRECTION_MSG = "Vous allez être redirigé vers la page
   d'accueil dans 5 secondes !";

}
```

Fichier 1 - "Strings.java"

Démarche du projet

Au début, l'étudiant doit se connecter en accédant à la page ***« login.jsp »***. Il fait la saisie de son login et son mot de passe et clique sur le bouton ***« Se connecter »*** **(voir figure 1).**

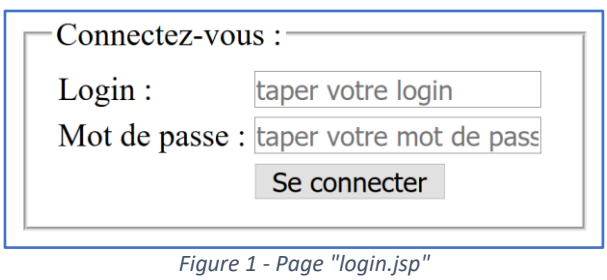

Figure 1 - Page "login.jsp"

Par la suite, si les informations saisies par l'étudiant sont correctes, il doit être rediriger vers la page d'accueil ***« index.jsp »***. Cette page contient 3 boutons principaux : ***« Consulter mon profil »***, ***« Télécharger ma fiche de résultats »*** et ***« Passer un test »*** **(voir figure 2).**

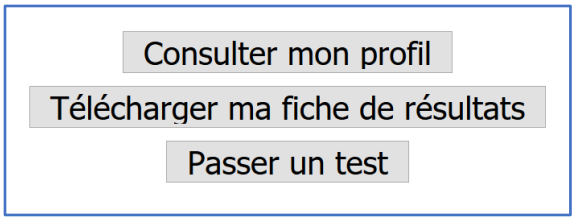

Figure 2 - Page "index.jsp"

En cliquant sur le bouton ***« Télécharger ma fiche de résultats »***, un fichier **Excel** contenant les notes de chaque test passé par l'étudiant doit être chargé. Si l'étudiant n'a passé aucun test, rien ne sera téléchargé **(voir figure 3).**

	A	B
1	**Etudiant**	Fedi Gaied
2		
3	**Date**	**Note /20**
4	13/3/2019 09:32	18
5	13/3/2019 10:41	13
6	15/3/2019 08:35	15
7		
8	**Moyenne**	**15.33**

Figure 3 - Exemple d'un fichier Excel

En cliquant sur le bouton ***« Passer un Test »***, la servlet ***« Test.java »*** sera appelée afin de charger à chaque fois une question dans la page ***« test.jsp »***. L'étudiant peut passer à la question suivant en cliquant sur le bouton ***« suivant »*** **(voir figure 4).**

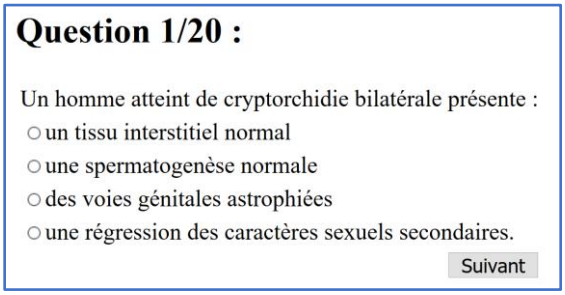

Figure 4 - Page "test.jsp"

Si l'étudiant termine toutes les questions, la note finale du test sera affichée et il sera redirigée vers la page d'accueil après 5 secondes **(voir figure 5).**

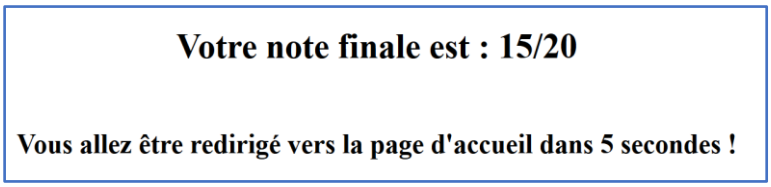

Figure 5 - Note finale

Les informations suivantes peuvent vous aider à mieux résoudre le sujet :

- Pour générer un nombre aléatoire appartenant à l'intervalle [**min**..**max**] :

```
Random random = new Random();
int r = random.nextInt((max - min) + 1) + min;
```

- Exemple permettant de définir un input de type ***« radio »*** :

```
<input name="proposition" type="radio" value="A" />Vrai
</br>
<input name="proposition" type="radio" value="B" />Faux
```

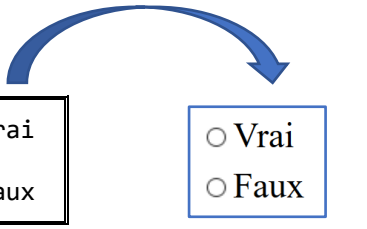

Figure 6 - Buton Radio

II. Développement du projet

Exercice 1 : Configuration du fichier « *web.xml* » (3 points)

Soit le fichier ***« web.xml »*** situé dans le dossier ***« WebContent/WEB-INF »*** :

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd" id="WebApp_ID" version="4.0">
   <display-name>Projet SVT</display-name>
   <welcome-file-list>
       <welcome-file>index.jsp</welcome-file>
   </welcome-file-list>
</web-app>
```

Fichier 2 - "web.xml"

1- Est-ce que le fichier ***« web.xml »*** représente un fichier **optionnel** ou **obligatoire** dans un projet **Java EE** ? Expliquer votre réponse d'une manière brève.

2- Ecrire les lignes nécessaires dans le fichier ***« web.xml »*** permettant de :

   a. Déclarer la servlet ***« Profil.java »***.

   b. Définir un mapping pour la servlet ***« Profil.java »*** avec les deux URLs : ***« /Profil/* »*** et ***« /profil/* »***. Expliquer la signification de ces deux URLs.

   c. Rediriger les pages d'erreur **404** vers la page ***« 404.jsp »***.

3- Indiquer la ligne de codes à ajouter dans la servlet ***« Profil.java »*** permettant de déclarer avec une autre méthode un mapping avec les deux URL : ***« /Profil »*** et ***« /profil »***.

Exercice 2 : Connexion et déconnexion d'un étudiant (7 points)

1. Soit une partie du code de la servlet ***« Login.java »*** et le code source de la balise **« body »** de la page ***« login.jsp »*** :

```
public class Login extends HttpServlet {
      private StudentDAOImpl studentDAOImpl;
      private RequestDispatcher rd;

    public void init(ServletConfig config) throws ServletException {
            studentDAOImpl = new StudentDAOImpl();
    }

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

    }

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

    }
}
```

Fichier 3 - "Login.java"

```
<form>
  <fieldset>
    <legend>Connectez-vous :</legend>
    <table>
      <tr>
        <td>Login :</td>
        <td><input type="text" name="login"/></td>
      </tr>
      <tr>
        <td>Mot de passe :</td>
        <td><input type="password" name="password"/></td>
      </tr>
      <tr>
        <td> </td>
        <td><input type="submit" value="Se connecter"/></td>
      </tr>
    </table>
  </fieldset>
</form>
```

Fichier 4 - "login.jsp"

a. Expliquer brièvement la tactique derrière l'instanciation de l'objet ***« studentDAOImpl »*** dans la méthode ***« init »***.

b. Ecrire les lignes de codes à ajouter dans la méthode ***« doPost »*** de la servlet ***« Login.java »*** permettant d'établir l'étape de la connexion d'un étudiant, en tenant compte de ces deux cas :

- **Cas n°1 :** *Le login et le mot de passe de l'étudiant sont correctes :*
  Créer une session avec un attribut nommé ***« etudiant »*** qui stocke un objet de type ***« Etudiant »*** contenant les informations d'un étudiant donné.
- **Cas n°2 :** *Les informations saisies par l'étudiant sont incorrectes, ou un problème de connexion avec la base de données s'est établit :*
  Sauvegarder, dans l'objet ***« request »*** des messages d'erreur.

  NB. Utiliser les messages d'erreurs existant dans la classe ***« Strings.java »***.

Important : *N'oublier pas de rediriger l'étudiant vers la page correspondante pour chaque cas !*

c. Indiquer les attributs à ajouter dans la balise ***« form »*** du fichier ***« login.jsp »***.

d. Indiquer les lignes de codes à ajouter au-dessus de la balise ***« table »*** du fichier ***« login.jsp »*** afin d'afficher en rouge et en gras les messages d'erreurs s'ils existent.

e. Indiquer les lignes de codes à ajouter dans le fichier ***« login.jsp »*** dans le but de respecter la contrainte suivante : *« Un étudiant ne peut pas accéder à la page de connexion s'il est déjà connecté »*.

2. On veut créer une nouvelle servlet nommé ***« Logout.java »*** permettant d'effectuer la tache de déconnexion d'un étudiant.

   a. Ecrire les lignes de code permettant d'effectuer la déconnexion.

   b. Dans quelle méthode doit-on écrire ces lignes de codes ? Justifier votre réponse.

Exercice 3 : Gérer la fiche des résultats d'un étudiant (3 points)

Soit une partie du code de la servlet ***« Result.java »*** :

```
public class Result extends HttpServlet {
    private MarksDAOImpl marksDAOImpl;

    public void init(ServletConfig config) throws ServletException {
            marksDAOImpl = new MarksDAOImpl();
    }
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    }
}
```

Fichier 5 - "Result.java"

Travail demandé :

Ecrire les lignes de codes à inscrire dans la méthode ***« doPost »*** permettant de charger un fichier **Excel**, exactement comme montré dans la ***figure 3***, contenant le résultat de chaque test passé par l'étudiant de la session courante, sachant que le fichier ***Excel*** ne doit pas être téléchargé dont le cas où l'étudiant n'a passé aucun test.

NB. *N'oubliez pas de formater les nombres à virgule flottante.*

Exercice 4 : Chargement de questions (7 points)

Un étudiant peut passer autant de fois qu'il veut des tests. Un test contient **20 questions** de type **QCM** qui respectent les contraintes suivantes :

- Les questions doivent être générées d'une manière **totalement aléatoire**.
- Il est impossible d'avoir la **même question** plus qu'une seule fois.
- Il est impossible d'avoir **plus que 5 questions** du même thème.

On veut développer cette partie en respectant la démarche suivante :

- Au début, la méthode ***« doGet »*** de la servlet ***« Test.java »*** est appelée. Cette méthode permet de charger une question, qui respecte les contraintes citées ci-dessus, dans un objet de type ***« Question »***. Ce dernier sera stocké dans l'objet ***« request »*** afin de l'envoyer avec le ***« RequestDispatcher »*** vers la page ***« test.jsp »***.

- La page ***« test.jsp »*** affiche la question stockée dans l'objet ***« request »*** comme c'est représenté dans la ***figure 4***.

- Quand l'étudiant clique sur le bouton ***« suivant »***, la servlet ***« Test.java »*** doit être appelée avec la méthode ***« doPost »***. Cette méthode doit calculer la note de la réponse de l'étudiant, stocker cette note, l'additionner avec la note précédente et charger une nouvelle question qui sera stockée est envoyée vers la page ***« test.jsp »***. Si on arrive à la dernière question **(question n°20)**, la méthode **« doPost »** doit afficher la note finale du test et rediriger l'étudiant vers la page d'accueil **(voir figure 5)**.

Travail demandé :

Soit la partie du code de la servlet *« Test.java »* :

```
public class Test extends HttpServlet {
    private MarkDAOImpl markDAOImpl;
    private QuestionDAOImpl questionDAOImpl;

    public void init(ServletConfig config) throws ServletException {
            markDAOImpl = new MarkDAOImpl();
            questionDAOImpl = new QuestionDAOImpl();
    }

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

    }

    private Question getNewQuestion() {

    }

    private int VerifyQuestion (Question question, String proposition) {

    }
}
```

Fichier 6 - "Test.java"

1- Ecrire les lignes de codes à mettre dans la méthode *« getNewQuestion »* permettant de générer une question d'une manière aléatoire et qui respecte les contraintes citées précédemment.

2- Ecrire les lignes de codes à mettre dans la méthode *« VerifyQuestion »* qui permet de donner une note sur la réponse d'un étudiant. *« 1 point »* si la réponse est correcte, *« 0 point »* dans le cas contraire.

3- En supposant que la méthode *« doGet »* est déjà développée, écrire les lignes de codes à ajouter dans la méthode *« doPost »*.

NB. Utiliser les deux méthode *« getNewQuestion »* et *« VerifyQuestion »*.

4- Ecrire le code à ajouter dans la balise *« body »* de la page *« test.jsp »* permettant d'afficher une question avec ses quatre propositions *(voir figure 4)*.

Exercice 5 : BONUS : Déploiement du projet (2.25 points)

Avant d'empaqueter l'application web *« Projet SVT »* dans un fichier livrable, le professeur veut faire quelques tests afin de garantir le bon fonctionnement de toutes les tâches du projet. Dans ce contexte, il essaye d'exécuter le site web sur sa propre machine en utilisant l'IDE *« Eclipse »* et le serveur d'application *« Apache Tomcat »*. En lançant ce projet, une boîte de dialogue d'erreur s'est affichée *(voir figure 7)*.

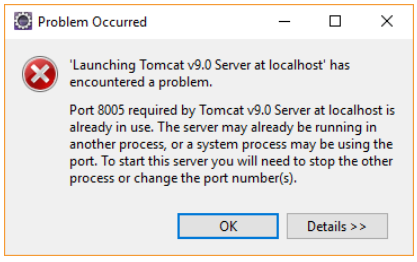

Figure 7 - Boîte de dialogue d'erreur

Travail demandé :

Citer 3 problèmes pouvant générer ce type d'erreur. Proposer une solution pour chaque problème.

« Amusez-vous bien et soyez créatif ! »