Programmation Web avancée	**420-02E-FX**
Travail pratique 1 : Java Web et MVC	**25%**
Prof. : Stéphane Lapointe	***Automne 2016***

Travail en équipe de deux obligatoirement.

Le professeur se garde le droit de modifier les équipes et de vérifier le travail effectué par les membres de l'équipe. La note d'équipe n'est donc pas assurée pour les deux membres de l'équipe.

Date de remise:

- ***Au plus tard jeudi le 22 septembre 2016 avant la fin de journée sur LEA.***
- ***Remise électronique uniquement (aucune copie papier).***

Objectif et présentation général du projet

L'objectif de ce travail est de compléter la construction d'une application Web basée sur l'architecture et la méthode de conception Modèle-Vue-Contrôleur (MVC) tel que présenté en classe. L'application Web étudiée porte sur un réseau social appelé ***twitface***. L'étudiant devra partir de l'application initiale présentée par le professeur et faire les modifications et ajouts demandées tout en respectant la structure initiale de l'application Web et la méthode de conception MVC.

Directives générales

- Dans ce travail, vous devrez **compléter** le code initial qui vous est fourni et en **corriger** certaines parties.
- Toutes les **fonctionnalités déjà implémentées** doivent **demeurées fonctionnelles** après les modifications que vous allez effectuer.
- Vous devez **comprendre** et **respecter** à la lettre les **spécifications** données aux **annexes A et B**. **L'annexe A** fournit les **détails des ressources** (non statiques) fournies par l'application Web selon MVC et **l'annexe B** présente le **diagramme de navigation** entre les ressources.

Éléments à modifier et à ajouter

- Inscrivez votre nom dans le pied de page à l'endroit approprié.

1. (25 points) Programmation de la ressource « /rech-amis » :

- Compléter la programmation de **la ressource « /rech-amis »** (méthode HTTP **GET**) qui permet de rechercher de nouveaux amis, et ce, même pour les utilisateurs non connecté.
- Cette ressource présente la **liste des membres trouvés** (en **ordre de nom complet** du membre) correspondant aux quatre **critères** de recherche qui sont **tous optionnels**. Si l'utilisateur est connecté, il faut ignorer dans le résultat de recherche les amis actuels et le membre lui-même.
- Si un critère de recherche est absent (champ vide ou espaces seulement), il faut l'ignorer dans la recherche. Si aucun critère n'est fourni ou si aucun membre ne correspond aux critères, on affiche un message indiquant qu'il n'y a aucun résultat. De plus, l'utilisateur peut fournir une partie seulement de la valeur d'un champ (**correspondance partielle**); par exemple, pour la ville d'origine, une recherche avec le terme « mont » fait correspondre, entre autres, les villes

« Montebello » et « El Monte ». De plus, si l'utilisateur décoche les deux options pour le sexe, on doit ignorer ce critère.

- Utilisez le même **type d'affichage** que celui de la page de suggestions d'amis avec les modifications suivantes :
  - Bien sûr, on ne doit pas afficher le nombre d'amis en commun.
  - Il faut afficher le **lien pour devenir ami** seulement si l'utilisateur est **connecté** en tant que « membre ».
  - Il faut afficher le nom de la **ville actuelle** en dessous du nom de l'ami.
  - Il faut afficher le nom de la **ville d'origine** entre **parenthèses** en dessous du nom de l'ami.
- Il est important que le formulaire de recherche demeure présent après l'affichage des résultats; il faut donc simplement ajouter une section à la vue « **rech-amis.jsp** » pour l'affichage des résultats si cela est nécessaire (formulaire soumis). De plus, vous devez **conserver l'état du formulaire** pour **tous les champs** de recherche incluant le sexe (**boîtes à cocher**).
- Les **espaces inutiles** doivent être **retirés** pour la recherche (un espace de trop ne doit pas faire échouer celle-ci) et pour la réinitialisation des champs du formulaire.
- Cette fonctionnalité doit obligatoirement être implantée de manière similaire à la fonctionnalité (déjà fournie) qui suggère des amis. Pour ce faire, vous devrez **créer un nouveau modèle** (une classe de type ***Java bean***) appelé « **ModeleRechAmis** ». Ce modèle doit créer une liste générique d'amis « ArrayList<Ami> » et la conserver dans un de ses attributs. Vous devez utiliser la classe existante « Ami » sans utiliser le champ « nbAmisEnCommun ».
- Attention : Le modèle de recherche d'amis ne doit être utilisé que lorsque le formulaire est soumis; autrement, il est inutile.
- Il ne faut pas faire de requêtes à la BD en JSTL à partir de la vue.

2. (15 points) Programmation de la ressource « /membre/mes-amis » :

- Programmer au complet la ressource « **/membre/mes-amis** » (méthode HTTP **GET**) qui affiche les demandes d'amitié en attente ainsi que la liste des amis actuels (en ordre de nom complet de membre pour les deux listes).
- Pour chaque membre, on doit afficher la **photo** et le **nom complet** :
  - Pour les demandes d'amitié, on doit aussi afficher la **date de la demande** ainsi qu'un **lien** permettant **d'accepter la demande d'amitié**; le nom de la ressource à appeler est « **/membre/accept-dem-ami** » (méthode HTTP **GET**) qui a un paramètre dont le nom est « **no-ami** ». Voir plus bas pour les détails concernant cette ressource que vous devrez développer.
  - Pour les amis, on doit aussi afficher un **lien** permettant de **supprimer l'ami**; le nom de la ressource est « **/membre/sup-ami** » (méthode HTTP **GET**) qui a un paramètre dont le nom est « **no-ami** ». Note : vous ne devez pas développer cette ressource.
- Faites les requêtes à la BD avec JSTL directement (pas de modèles) en complétant la page JSP existante « **mes-amis.jsp** ». Bien sûr, vous devrez modifier le contrôleur pour les membres « ControleurMembre ».

3. (15 points) Programmation de la ressource « /membre/accept-dem-ami » :

- Programmer au complet la ressource « **/membre/accept-dem-ami** » (méthode HTTP **GET**) qui permet d'accepter une demande d'amitié (le numéro d'ami dont on accepte l'amitié est reçu dans le paramètre GET « **no-ami** »).
- Il faut bien sûr faire les modifications nécessaires dans la base de données (retirez la demande d'amitié et ajoutez la relation d'amitié).
- La gestion complète de cette ressource (incluant les validations) doit être effectuée avec à l'aide du modèle existant (une classe de type *Java Bean*) appelé « **ModeleGestionAmi** ». Ajoutez les attributs et méthodes nécessaires dans cette classe.
- Après les traitements, on doit réafficher la même vue (« mes-amis.jsp ») avec un **message de confirmation ou bien d'erreur**. Une erreur survient lorsqu'un membre tente de devenir ami avec une personne qui ne lui a pas fait une demande d'amitié (en modifiant l'URL directement, par exemple). Le modèle doit donc valider si la demande d'amitié existe vraiment avant d'ajouter la relation d'amitié.
- Il ne faut pas faire de requêtes à la BD en JSTL à partir de la vue.

4. (30 points) Programmation des ressources « /connexion » et « /deconnexion »:

- Vous devez **programmer la connexion** qui doit être gérer par le contrôleur général « ControleurGeneral.java ». Le nom de la ressource associée à la connexion est « **/connexion** » (méthode HTTP **POST**). Voir le formulaire de connexion existant qui est présent à deux endroits.
- Il y a deux types d'utilisateur, soient « **membre** » et « **administrateur** »; voir la base de données pour les tables correspondantes et le code Java pour l'énumération correspondante.
- Pour le traitement complet de la demande de connexion (incluant la validation des paramètres de connexion, côté serveur uniquement), vous devez créer un **nouveau modèle** (une classe Java) ayant comme nom « **ModeleConnexion** ». Voici des détails concernant le fonctionnement de ce modèle qui contient **deux attributs** :
  - Un des attributs est de type « **ConnexionBean** »; il sera créé et utilisé uniquement lorsque la tentative de connexion est réussie. Il permettra de conserver **l'information sur l'utilisateur connecté**.
  - L'autre attribut est de type « **String** » et servira à conserver le **message d'erreur** lorsque la tentative de connexion échoue.
  - Lorsqu'un de ces attributs n'est pas utilisé, il doit contenir la valeur « null ». Ainsi, il y aura toujours au moins un de ces deux attributs qui sera à « null ».
  - Pour la validation du formulaire par le modèle (côté serveur uniquement), assurez-vous que les champs ne sont pas laissés vides et ignorer les espaces inutiles.
  - Si la tentative de **connexion a échoué**, le modèle doit simplement **conserver un message d'erreur** approprié dans son attribut de type « String » et cesser les opérations immédiatement. Vous devez avoir deux types de message d'erreur pour les cas suivants :
    - Paramètres de connexion non fournis (un des deux ou les deux). Un paramètre ne contenant que des espaces sera considéré comme vide donc non fourni.

        - Paramètres de connexion invalides; il n'est pas possible de valider la connexion en tant que « membre » ou bien en tant que « administrateur ».
    - Si la tentative de **connexion** a **réussi**, le modèle doit **créer un *bean* de connexion** (classe « ConnexionBean »), le configurer correctement et le conserver dans son attribut dédié à cette fin.
    - Assurez-vous de bien crypter le mot de passe en **SHA2 (256 bits)** dans les requêtes SQL préparée.
- Le contrôleur général « **ControleurGeneral** » qui est responsable de la gestion de la connexion doit effectuer les opérations suivantes lors d'une tentative de connexion :
    - **Créer le modèle** de connexion « ModeleConnexion » et **appeler la méthode** appropriée tout en lui passant les **paramètres nécessaires**.
    - Par la suite, il doit **vérifier** si la tentative de **connexion** a réussi :
        - Si celle-ci a **réussi**, il doit **conserver le *bean* de connexion** (créé par le modèle) dans un attribut de la **session** et faire une **redirection côté client** vers la bonne ressource, soit « /**membre/** », soit « /**admin/** ».
        - Si celle-ci a **échoué**, il doit **conserver** le **message d'erreur** créé par le modèle dans un **attribut de la requête** et, par la suite, il doit **afficher la bonne vue**.
- La **gestion des interfaces** suite à une tentative de **connexion qui a échoué** doit respecter ce qui suit :
    - La page où l'erreur s'est produite doit être réaffichée (page d'accueil ou bien page de recherche de nouveaux amis). Entre autres, s'il y a erreur à partir de la page de recherche d'amis, il ne faut pas retourner sur la page d'accueil.
    - Le message d'erreur produit par le modèle doit être affiché en dessous du formulaire de connexion concerné.
    - L'état du champ pour le nom d'utilisateur doit être conservé dans le formulaire de connexion (retirer les espaces inutiles); le mot de passe ne doit pas être conservé.
- La **gestion des interfaces** doit respecter ce qui suit pour tenir compte de l'état de la connexion :
    - **L'en-tête des vues** (sauf celle de la page d'accueil, soit « index.jsp ») doit s'adapter en fonction de l'état de la connexion. Si la personne n'est pas connectée, on doit présenter le formulaire de connexion; par contre, si elle est connectée, on doit plutôt afficher le nom complet de l'utilisateur connecté, son nom d'utilisateur (entre parenthèses) ainsi qu'une option pour se déconnecter. Pour les administrateurs du site Web, il n'y a pas de photos mais on doit afficher le texte « Administrateur du site Web » en-dessous de son nom.
    - Le **menu** doit s'adapter en fonction de l'état de la connexion de sorte à ce que les **bonnes options** soient affichées pour chacun des trois états possibles (non connecté, connecté comme « membre » et connecté comme « administrateur »).
    - Les **liens** dans le bas des **quatre pages d'erreurs** doivent être modifiés de sorte à pouvoir revenir à la bonne page après une erreur (accueil, section « membre » ou section « administrateur »).

- Vous devez **programmer la déconnexion** au niveau du contrôleur général (« ControleurGeneral.java »). Le nom de la ressource associée à la déconnexion est « **/deconnexion** » (méthode HTTP **GET**). Après la déconnexion, on doit revenir à la page d'accueil (« index.jsp ») tout en affichant un message confirmant la déconnexion.
- La connexion ne fonctionne pas à partir de la page de recherche d'amis; corrigez le problème.

5. (15 points) Programmation de la section pour les administrateurs :

- Développer la section pour les administrateurs associée à la ressource principale « **/admin/** » (méthode HTTP **GET**) à l'aide d'un nouveau contrôleur-répartiteur (une servlet) appelée « **ControleurAdmin** ».
- Modifier toutes les vues qui doivent l'être, entre autres, celle pour l'en-tête des pages. Pour un administrateur, il n'y a pas de photo mais il y a le texte « Administrateur du site Web » en dessous de son nom.
- Effectuer les ajustements nécessaires au menu de sorte à avoir uniquement l'option « **Gestion des publications** » lorsque l'utilisateur est connecté en tant qu'administrateur. L'option « Recherche des amis » ne doit pas y être.
- Ne développer que la page d'accueil pour les administrateurs qui doit simplement contenir la liste des publications de tous les membres en ordre inverse de date de publication (les plus récentes en premier). De plus, pour chaque publication, il doit y avoir un lien (ou une image, un « X » par exemple) permettant de la détruire. Le nom de la ressource à appeler est « **admin/supp-pub** » (méthode HTTP **GET**) qui a un paramètre dont le nom est « **no-pub** ». Ne développer pas cette ressource mais afficher simplement la vue qui indique qu'elle est en construction (« en-construction.jsp ») lorsqu'on la demande.
- Faites les requêtes à la BD avec JSTL directement (pas de modèles). Ajoutez la vue "**accueil-admin.jsp**".

Directives spécifiques

1. Vous devez toujours **valider** la **présence** et le **contenu** des **paramètres des requêtes HTTP**.
2. Pour toutes les ressources, vous devez **gérer correctement la méthode HTTP utilisée** pour la demande de la ressource. Ainsi, si on demande une ressource par la méthode POST alors qu'elle ne doit accepter que la méthode GET, on doit retourner une erreur HTTP 405 (méthode non permise) et non pas une erreur HTTP 404 (ressource inexistante). Pour tester à partir d'un navigateur Web, je vous recommande d'utiliser l'extension ***HttpRequester*** pour ***Firefox*** ou bien ***Postman*** pour ***Google Chrome***.
3. Pour les **attributs de la requête**, vous devez **respecter** à la lettre la **dernière colonne du tableau de l'annexe A**. Il est interdit d'utiliser d'autres noms ou d'ajouter d'autres attributs.
4. La seule information qu'il est permis de mettre dans la session utilisateur est « **connBean** » de type « **ConnexionBean** », et ce, uniquement lorsque l'utilisateur est connecté.
5. Dans tous les **modèles**, il est strictement **interdit** d'utiliser des objets en lien avec le contexte Web de l'application. En particulier, il n'est pas permis de passer à une méthode d'un modèle les objets « **HttpRequest** » et « **HttpResponse** »

6. Pour ce travail, vous devez partir du projet de départ en le renommant. Le nom du nouveau projet et dossier doit être « **twitface-sl-jpb »** en remplaçant « sl » et « jpb » par vos initiales. Par la suite, modifiez aussi le ***context path*** (*context root*) en lui donnant la même valeur que le nom du projet.
7. Pour les requêtes à la BD à partir de code Java, vous devez utiliser la classe Java « **ReqPrepBdUtil.java** » déjà présente dans le projet initial. Ainsi, vous ne devez utiliser que des **requêtes préparées** avec des **paramètres**.
8. Les nouvelles **classes Java** créées doivent être placées dans les **bons *packages***.
9. Pour la génération des **vues**, vous devez seulement utiliser **EL-JSTL**. Vous ne pouvez pas utiliser les éléments de scripts en JSP.
10. Tous les documents produits dans le cadre de ce travail doivent s'afficher correctement avec au moins les navigateurs Web suivants : Firefox, Chrome et Internet Explorer.
11. Respectez les règles de base de la programmation: utilisation de méthodes, utilisation des instructions appropriées, **noms des identifiants significatifs**, **présentation du code,** etc.
12. Mettez suffisamment de commentaires dans toutes les parties de votre code (Java et JSP). De plus, les **commentaires** pour **Javadoc** doivent être présents pour tout le code Java (classes, attributs, constructeurs, getter/setter et méthodes).
13. Portez une attention particulière à la structure de votre programme; pensez à la ***parcimonie*** et à la ***simplicité*** avant tout !

Précisions sur l'évaluation

Outre les éléments à réaliser dans le cadre du travail, les aspects suivants viendront influencer la note finale:

- Structure du code (modularité, clarté, simplicité, concision, uniformité, etc.)
- Présentation du code: saut de ligne, décalage, etc.
- Nom des identifiants de toutes sortes (variables, méthodes, classes CSS, etc.)
- Commentaires.
- Qualité du français (structure des phrases et fautes).
- D'autres éléments pourraient s'ajouter au besoin.

À remettre

- Aucune remise papier.
- Le dossier de votre projet *Eclipse* (contenant tous les fichiers du projet) doit porter le nom « twitface-sl-jpb » (avec vos initiales) et doit être placé dans un autre dossier portant le nom suivant « **TP1-P1-NomFamille1-P2-NomFamille2** » (utilisez les initiales de vos prénoms et vos noms de famille complet).
- Ajoutez un fichier texte portant le nom « **InfoPourProfesseur.txt** » dans le dossier principal « TP1-P1-NomFamille1-P2-NomFamille2 » qui précise si des éléments n'ont pas pu être implémentés correctement. Le fichier doit être présent (avec un message approprié) même si tout fonctionne correctement.
- **Remettez le dossier compressé sur LÉA**; il est important de conserver le dossier que vous venez de créer lors de la compression des fichiers avant la remise.

Annexe A
Ressources (non statiques) fournies par l'application Web selon MVC

Nom de la ressource (URL)	Signification	Contrôleur et modèles	Paramètres GET/POST	Attributs utilisés (niveau requête)
/	Page d'accueil de l'application Web (non connecté); permet de se connecter, de s'inscrire et de faire une recherche d'amis simplifiée (un seul champ).	ControleurGeneral	Aucun	Aucun
/connexion	Permet de gérer la demande de connexion.	ControleurGeneral ModeleConnexion	POST: nom-util, mot-passe, source	msgErrConn (String)
/deconnexion	Permet de gérer la demande de déconnexion.	ControleurGeneral	Aucun	msgConfDeconn (String)
/rech-amis	Permet de faire une recherche d'amis en fonction de différents critères : nom, ville d'origine, ville actuelle et sexe.	ControleurGeneral ModeleRechAmis	GET : nom-ami, ville-origine, ville-actuelle, sexe	modRechAmis (ModeleRechAmis)
/membre/	Page d'accueil de la section membre; affiche le fil de nouvelles du membre. ***RESTERA EN CONSTRUCTION***	ControleurMembre	Aucun	Aucun
/membre/profil	Affiche le profil du membre ainsi que les publications sur son babillard.	ControleurMembre	Aucun	Aucun
/membre/mes-amis	Affiche les demandes d'amitié en suspens ainsi que les amis actuels.	ControleurMembre	Aucun	Aucun
/membre/sugg-amis	Suggère des amis en fonction du nombre d'amis en commun.	ControleurMembre ModeleGestionAmi	GET : indice-prem, nb-amis-sugg	modSuggAmis (ModeleGestionAmi)
/membre/accept-dem-ami	Permet d'accepter une demande d'amitié. Réaffiche la page « Mes amis ».	ControleurMembre ModeleGestionAmi	GET : no-ami	modAcceptDemAmi (ModeleGestionAmi)
/membre/dem-ami	Permet d'effectuer une demande d'amitié. ***RESTERA EN CONSTRUCTION***	ControleurMembre ModeleGestionAmi	GET : no-ami	modDemAmi (ModeleGestionAmi)
/membre/supp-ami	Permet de supprimer un ami. ***RESTERA EN CONSTRUCTION***	ControleurMembre ModeleGestionAmi	GET : no-ami	modSuppAmi (ModeleGestionAmi)
/admin/	Page d'accueil de la section administrateur; affiche les publications de	ControleurAdmin	Aucun	Aucun

	tous les membres.			
/admin/supp-pub	Permet de supprimer une publication. ***RESTERA EN CONSTRUCTION***	`ControleurAdmin` `ModeleGestionPub`	`GET : no-pub`	`modSuppPub` `(ModeleGestionPub)`

Entrées dans la session utilisateur :

1. Lorsque l'utilisateur est connecté : « **connBean** » de type « **ConnexionBean** »

Annexe B
Diagramme de navigation entre les ressources