TP 2
La couche modèle d'un projet d'application Web

L'objectif de ce TP est de développer la couche modèle de l'architecture MVC présentée dans le TP1. Un modèle définit, sous la forme de classes, un ensemble d'entités relatives à l'application visée. Il pourra également contenir un ensemble de classes outils permettant la manipulation des données correspondantes.

Architecture MVC et Application :

Le dossier « model » contient les classes propres à l'application. Pour l'instant, seules les classes « utilisateur » et « utilisateurTable » sont présentes. La classe « utilisateur » propose un exemple d'annotation brute pour définir l'entité correspondante. Seules les propriétés sont définies. Aucune relation n'est spécifiée. La classe « utilisateurTable » contient la méthode permettant à l'utilisateur de se connecter à son profil par son identifiant et son mot de passe. Nous implémenterons dans ce TP les classes d'entités « utilisateur », « message », « post » et « chat », puis les classes fonctionnelles (classes outils) si nécessaire, soit « utilisateurTable », « postTable », « messageTable », « chatTable ».

Le diagramme de classe simplifié du modèle est présenté ci-dessous.

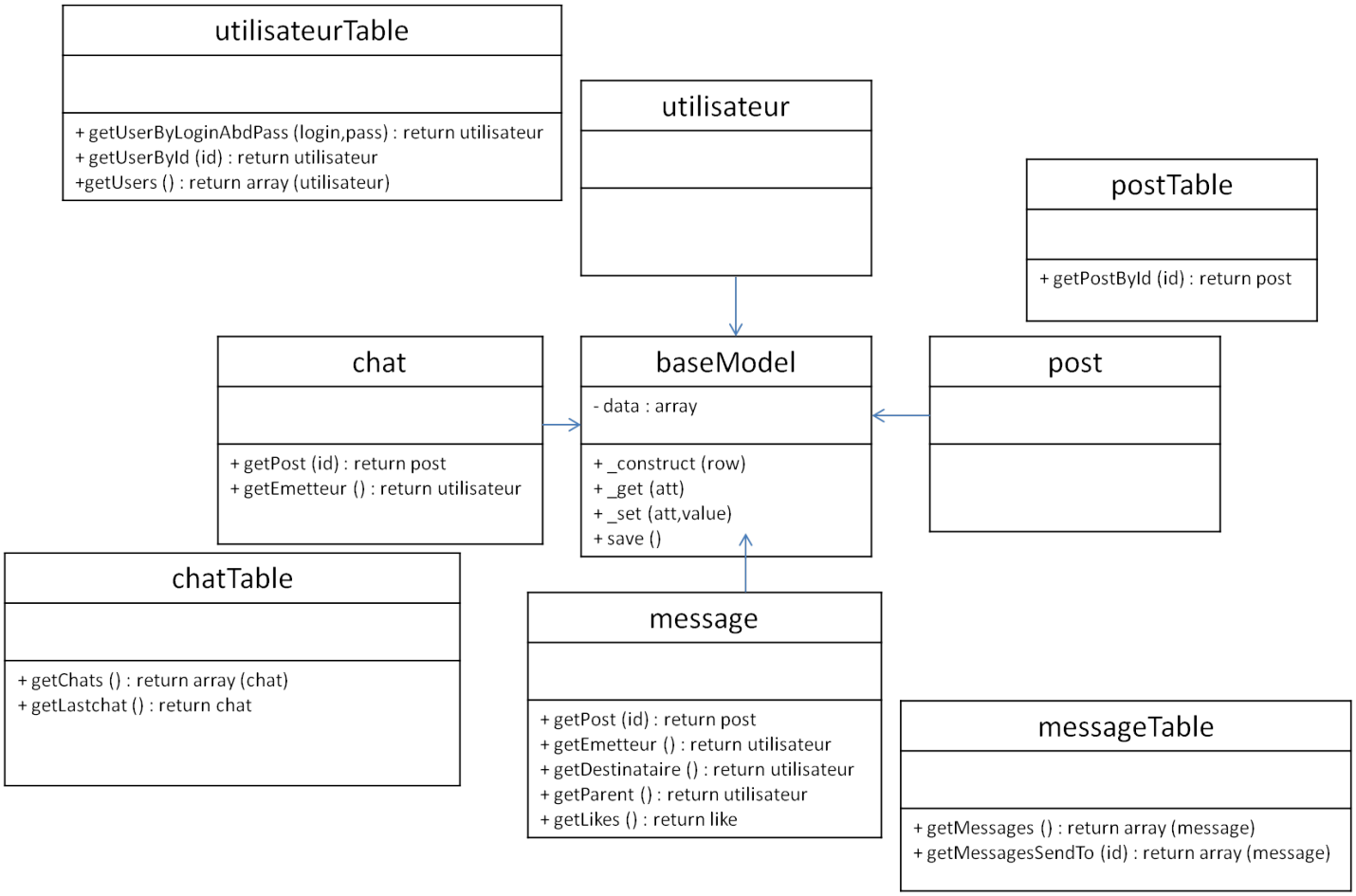

Figure 1: diagramme de classe simplifié du modèle

Ce diagramme simplifié contient un élément central, la classe « base », d'où vont hériter les classes *primitives*. Chacune de ces classes doit pouvoir être instanciées grâce aux éléments présents dans la table correspondante de la base de données. Concrètement, ces classes sont la correspondance entre le modèle de données et notre application.

* Un exemple : la classe *message* représente une instance (= un enregistrement dans la table) d'un message. La table *messageTable* est la classe qui permet de récupérer à travers la base de données des instances de message (= représente l'accès à la table).

Ces deux classes fournissent une couche d'abstraction à la base de données et apportent une structure métier qui colle au modèle de données. Ainsi on trouvera dans la classe messageTable, des méthodes (le plus souvent statiques) qui renverront des instances de message construites à partir des lignes récupérées en base.

* Un autre exemple : un objet de la classe utilisateur pourra être instancié avec les informations contenues dans la table utilisateur, sur un enregistrement donné. C'est à dire son identifiant, mot de passe, nom, prénom, date de naissance, etc.

Lorsqu'un objet utilisateur est instancié, d'après un enregistrement dans la table utilisateur, les valeurs des champs sont récupérées puis affectées à des attributs propres à la classe utilisateur. Cependant, il n'est pas nécessaire de déclarer ces attributs dans le code de la classe, ils peuvent être déclarés de manière dynamique. De ce fait, chaque valeur à affecter à un attribut de classe se fait grâce à la méthode *magique* « __set ». De la même philosophie découle la méthode *magique* « __get », permettant de récupérer les valeurs affectées aux attributs de classes.
Ces méthodes sont les mêmes que celles utilisées dans la classe "context" vue au TP1. Toutes les classes *primitives* de notre modèle utiliseront ces méthodes.

- Il est toutefois possible d'instancier un objet d'une classe du modèle sans récupérer les valeurs de la table correspondante dans la base de données, pour créer un nouvel utilisateur vide par exemple. De ce fait, la méthode « __set » permet d'affecter ultérieurement des valeurs aux attributs nécessaires.
- La méthode « save » de la classe « base » vous est fournie. Elle permet de mettre à jour ou d'insérer un enregistrement dans une table de la base de données correspondant à l'objet courant.
- La classe « post » définit de manière générique toute information envoyée sur la plateforme de réseau social. Comme précisé dans le fichier PDF « applicationOurface », la table post de la base de données centralise les saisies textuelles pouvant être accompagnées d'images. Les messages et le chat sont donc des posts, pouvant posséder des méthodes propres.

Implémentation :

1. La classe base
   - Vous devez implémenter un constructeur, prenant ou non un paramètre de type tableau (sous la forme array($key => $value, …)), contenant les valeurs provenant d'un enregistrement d'une table de la base de données. Si le paramètre est présent, chaque valeur du tableau sera affectée à un attribut de la classe par la méthode «__set».
   - Vous devez donc implémenter les méthodes «__get» et «__set».

2. La classe utilisateurTable
   - Comme vous pouvez le constater sur le diagramme de classe, la classe utilisateurTable possède trois méthodes. L'une d'elle est déjà implémentée et est utilisée pour la connexion d'un utilisateur à son profil, c'est la méthode «getUserByLoginAndPass».
   - Une seconde méthode (que vous devez implémenter) est destinée à récupérer les informations d'un utilisateur selon un identifiant. Nous pourrons ultérieurement afficher le profil de quelqu'un d'autre.
   - Enfin, une méthode est nécessaire afin de collecter l'ensemble des utilisateurs utilisant notre plateforme de réseau social.

3. La classe postTable
   - Pour cette classe, la seule méthode à implémenter permet de récupérer un post selon un identifiant. Comme précisé ci-dessus, tout contenu (multimédia, textuel, etc.) envoyé sur la plateforme de réseau social est un post.

4. La classe message
Pour cette classe primitive, vous devrez implémenter toutes les méthodes définies dans le diagramme de classes, comme par exemple ;
   - la méthode «getPost()». Cette méthode permet de récupérer l'objet *post* associé à un message.
   - la méthode «getParent()». Cette méthode permet de récupérer l'objet *utilisateur* correspond au rédacteur du message.
   - la méthode «getLikes()» qui permet de retourner le nombre d'utilisateurs ayant voté.

5. La classe messageTable
Pour cette classe outil, vous devrez implémenter :

- une méthode «getMessages()» permettant de collecter l'ensemble des messages, via une requête récupérant les données de la table tweet.
- une méthode «getMessagesSentTo(id)» permettant de collecter les n derniers messages postés sur le mur d'un utilisateur particulier. Le paramètre à lui passer est un identifiant d'utilisateur. Cette méthode doit faire appel à une fonction PL/SQL que vous allez créer permettant de trier les messages d'un utilisateur et ne renvoyer que les n derniers.

Ces deux méthodes retournent un tableau contenant des objets messages (pour cela vous devez implémenter la méthode doQueryObeject dans la classe dbconnection).

6. Les classe chat et chatTable

Pour la classe primitive chat, vous devrez implémenter la méthode "getPost(id)". Cette méthode permet de récupérer l'objet *post* associé à un chat.

Pour la classe outil messageTable, vous devrez implémenter :

- une méthode "getChats()" permettant de collecter l'ensemble des chats, via une requête récupérant les données de la table chat. Cette méthodes retournera un tableau contenant des objets chats.
- une une méthode "getLastChat()" permettant de récupérer le dernier message du chat.

7. Les classes primitives

Le diagramme de classes que nous vous avons donné est très simplifié, notamment au niveau des classes primitives. Vous pouvez les enrichir si vous le désirez.