UNIVERSITE FERHAT ABBAS - SETIF 1

FACULTE DES SCIENCES – DEPARTEMENT D'INFORMATIQUE

Niveau : 3ème Licence Académique

Epreuve : Génie Logiciel

Date : 29/01/2023

Corrigé-type

1ère partie : QCM (5 points)

// *bonne réponse* = 0.5, *réponse incomplète* = 0.25, *mauvaise réponse* = -0.25, *sans réponse* = 0

1. **Des sous-activités de la planification du projet**
   a. développer le produit logiciel
   b. évaluer les ressources, les coûts et les délais
   c. élaborer un calendrier du projet
   d. aucune des réponses précédentes
2. **L'analyse des besoins détermine**
   a. les possibilités de réalisation du projet
   b. ce qu'il faut faire pour réaliser les besoins du système
   c. comment réaliser les besoins du système
   d. aucune des réponses précédentes
3. **La décomposition des besoins du système en versions caractérise**
   a. le développement par prototypage
   b. le développement incrémental
   c. le modèle RAD
   d. aucune des réponses précédentes
4. **La modélisation doit permettre**
   a. l'abstraction et la compréhension
   b. la précision
   c. l'exhaustivité (représenter tous les détails)
   d. aucune des réponses pré cédentes
5. **L'abstraction et la décomposition dans la modélisation signifient**
   a. la représentation du système à base de modèles
   b. la représentation du système par rapport à un point de vue
   c. l'utilisation de notation claire et expressive
   d. aucune des réponses précédentes
6. **Les classes entités sont identifiées sur la base**
   a. des noms et des expressions nominales
   b. de la liste des classes candidates épurée
   c. de la réalisation de cas d'utilisation
   d. aucune des réponses précédentes
7. **L'héritage de cas d'utilisation définit**
   a. l'adoption du comportement de base
   b. la possibilité d'ajout ou de modification du comportement de base
   c. un comportement additionnel
   d. aucune des réponses précédentes
8. **Le modèle MVC adapté aux applications web**
   a. le contrôleur est toujours l'intermédiaire
   b. la vue est toujours l'intermédiaire
   c. la communication directe entre modèle et vue est toléré (possible)
   d. aucune des réponses précédentes
9. **Une implémentation possible du modèle MVC**
   a. une vue contenant des références du modèle et des contrôleurs
   b. un contrôleur contenant des références du modèle et des vues
   c. un modèle contenant des références des vues et des contrôleurs
   d. aucune des réponses précédentes
10. **Les patterns utilisés pour implémenter le modèle MVC**
    a. State, Composite et Singleton
    b. Façade et Simple Factory
    c. Observer et Strategy
    d. aucune des réponses précédentes

2ème partie : Principes SOLID (5 points)

- Identifier le ou les principes SOLID qui ne sont pas vérifiés sur le code suivant et proposer la solution appropriée :

// La classe *TotalSalariesCalculator* est responsable du calcul du total des salaires payables aux employées sur la base des types d'employés (Salaire employé permanent = index * 500, Salaire employé temporaire : hours * 100)

```php
<?php
interface EmployeeIF // 1.0 point
{
    public function getSalary();
}
class PermanentEmployee implements EmployeeIF
{
    protected $index;
    public function getIndex(){
       return $this->index;
    }
    public function getSalary(){
       return $this->index * 500;
    }
}
```

Essentiellement :
// **OCP** : l'ajout d'un nouveau type d'employés implique le changement du code de la classe *TotalSalariesCalculator*
En plus :
// **SRP** : la classe *TotalSalariesCalculator* possède deux responsabilités ; le calcul du salaire de chaque employé et le calcul du total des salaires
// **DIP** : la classe de haut niveau *TotalSalariesCalculator* dépend de classes de bas niveau `PermanentEmployee ou TemporaryEmployee`

```php
class TemporaryEmployee implements EmployeeIF // 0.5 point
{
    protected $hours;
    public function getDuration(){
       return $this->hours;
    }
    public function getSalary(){
       return $this->hours * 100; // 0.5 point
    }
}
class TotalSalariesCalculator
{
    public function calculate(EmployeeIF ...$workers){ // 1.0 point
         foreach ($workers as $worker)
        {
            $salary[] = $worker->getSalary(); // 1.0 point
             if ($worker instanceOf PermanentEmployee)
            {
                $salary[] = $worker->getIndex()*500;
            }
            else
            {
                $salary[] = $worker->getDuration()*100;
             }
        }
        return array_sum($salary);
    }
}
```

3ème partie : Modélisation UML (10 points : 5+2+2+1)

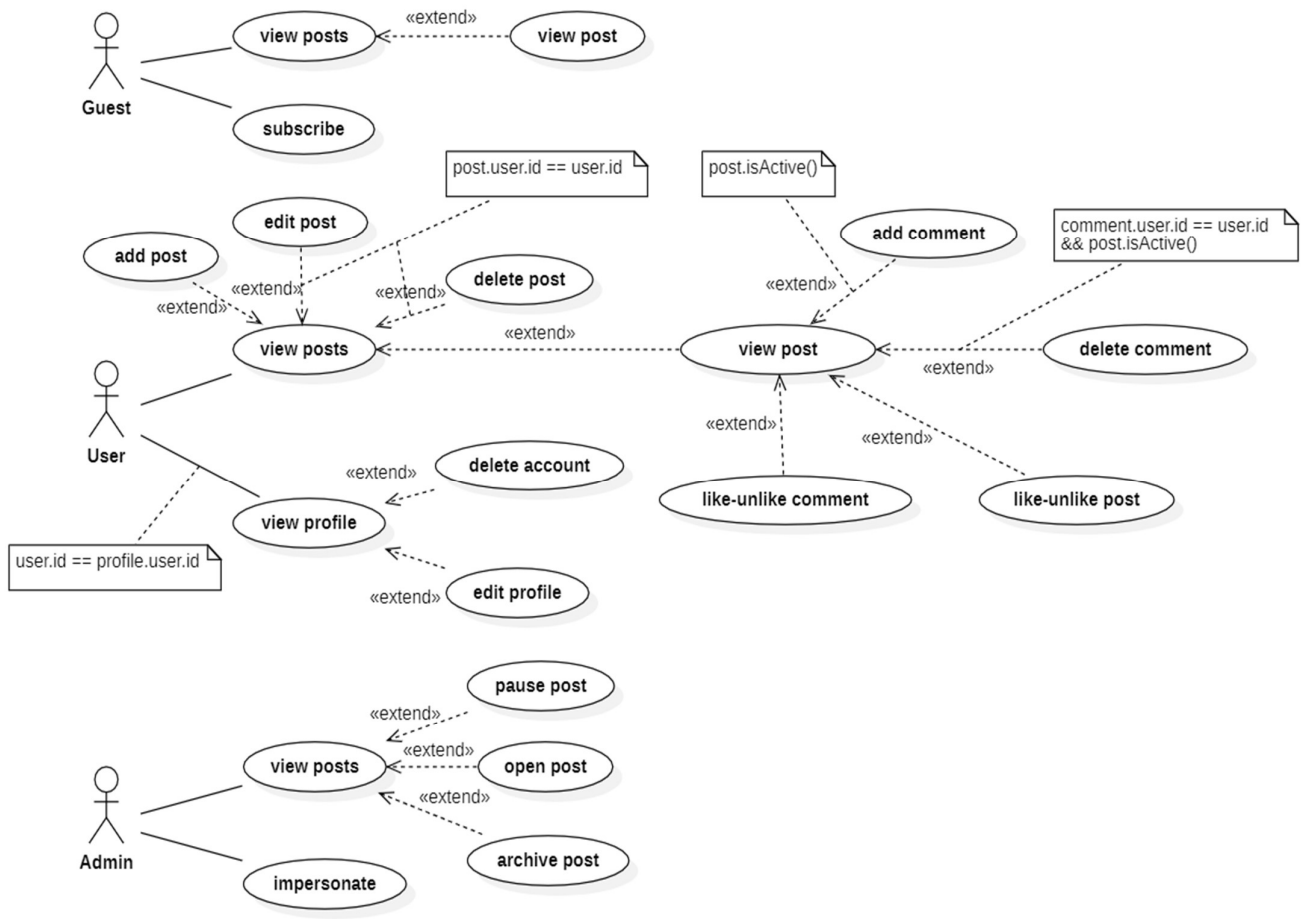
3.1 Fonctionnalités : Un *Blog* offre à ses utilisateurs les possibilités de publier leurs *posts* (articles), de lire ou de commenter des posts ; trois types d'acteurs sont identifiés ; *visiteur (Guest)*, *utilisateur (User)* et *administrateur (Admin)*.

Un *visiteur* peut explorer la liste des posts publiés (*view posts*) et demander par la suite de consulter un post donné (*view post*). Il peut également s'inscrire (*subscribe*) et devenir ainsi un utilisateur.

Un *utilisateur* peut lors de l'exploration de la liste des posts (*view posts*), consulter un post (*view post*), ajouter un nouveau post (*add post*), modifier (*edit post*) ou supprimer (*delete post*) l'un de ses posts. Pendant la consultation d'un post (*view post*), l'utilisateur peut ajouter un commentaire (*add comment*), annuler l'un de ses commentaires sur ce post (*delete comment*), aimer/annuler_aimer le post (*like/unlike post*), aimer/annuler_aimer un commentaire (*like/unlike comnent*). Un utilisateur peut également voir son profil (*view profile*) et ensuite modifier ce profil (*edit profile*) ou supprimer son compte (*delete account*).

Un administrateur peut, à partir de l'exploration de la liste des posts (*view posts*), suspendre les commentaires d'un post (*pause post*), activer les commentaires d'un post suspendu (*open post*) ou archiver un post (*close post*). Il peut également prendre le rôle d'un utilisateur donné ou d'un visiteur (*impersonate*).

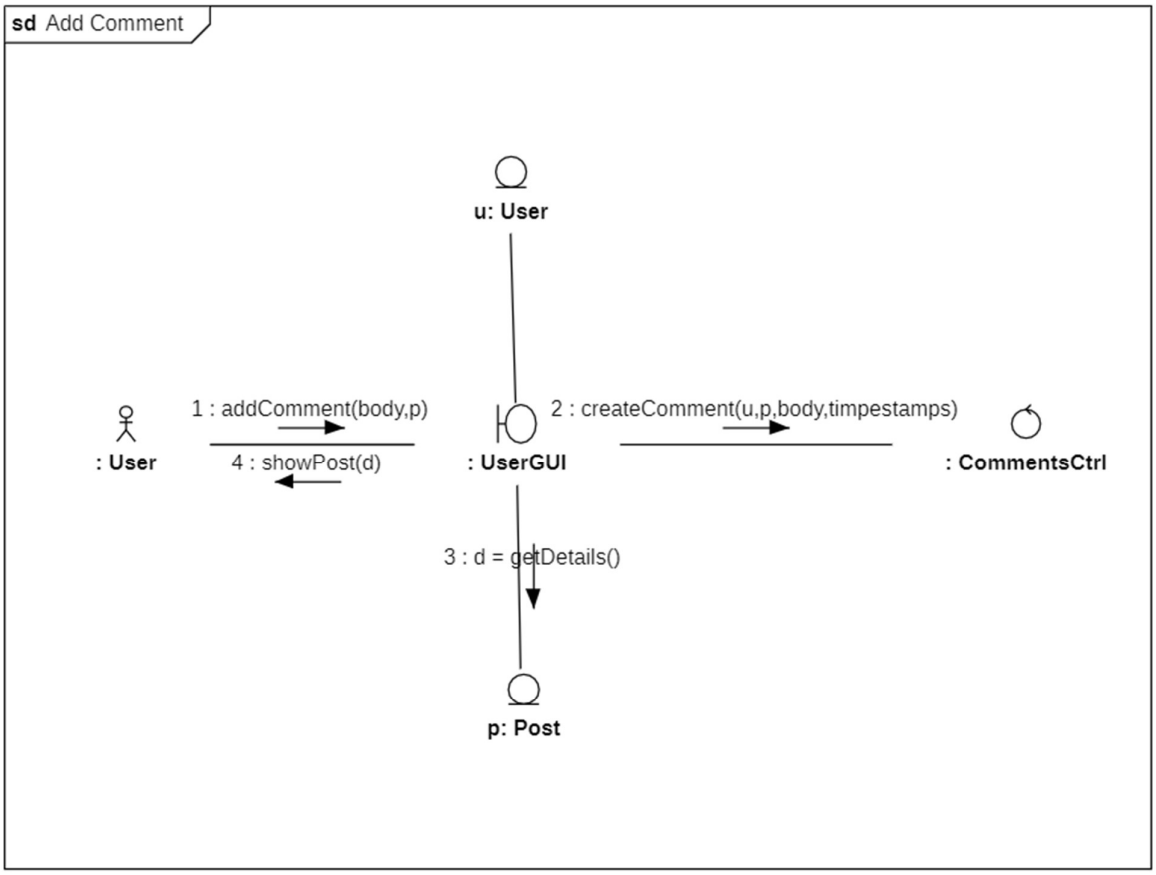
- donner les trois diagrammes de cas d'utilisation associés aux différents acteurs



// 0.25 pour chaque relation entre cas d'utilisation ou lien entre acteur et cas d'utilisation

3.2 Interaction : Pour ajouter un commentaire (*Comment*) à un article (*Post*), l'utilisateur authentifié saisit le contenu du commentaire (*body*) et demande au système d'enregistrer le nouveau commentaire. Le système créé alors un nouvel objet commentaire sur la base des objets de l'article concerné (post) et de l'utilisateur authentifié (*user*), la date du système (*timestamps*) ainsi que le contenu saisi (*body*) et affiche ensuite les détails du post sujet de ce commentaire.

- donner le diagramme de communication associé au cas d'utilisation "*add comment*"



// 0.25 pour chaque objet / message

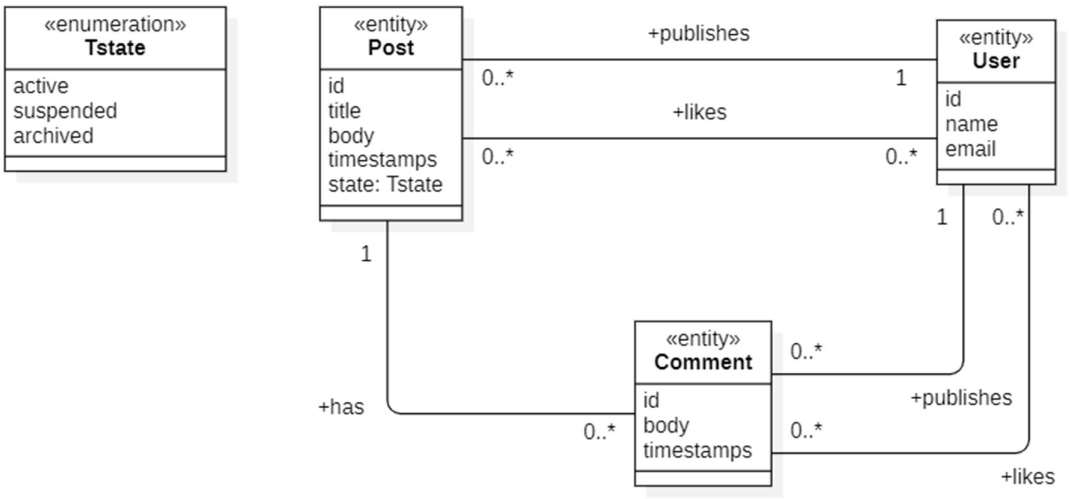
3.3 Structure : Un *Post* est caractérisé par *id*, *title*, *body* et *timestamps* et un utilisateur (*User*) est identifié par *id*, *name* et e*mail*. Un post est publié par un seul utilisateur et un utilisateur peut publier plusieurs posts. Un commentaire (*Comment*) est caractérisé par *id*, *body* et *timestamps*. Il est écrit par un utilisateur et concerne un post. L'utilisateur (ou le post) peut avoir plusieurs commentaires. Un utilisateur peut aimer un ou plusieurs posts (ou commentaires). Un post (ou un commentaire) peut être aimé par plusieurs utilisateurs. Finalement, Un post peut être active, suspendu ou archivé.

- donner le diagramme de classes correspondant

// le diagramme de classes de base

// ébauche après réalisation du cas d'utilisation "*add comment*" selon la description 3.2 (0.75 pts **bonus**)

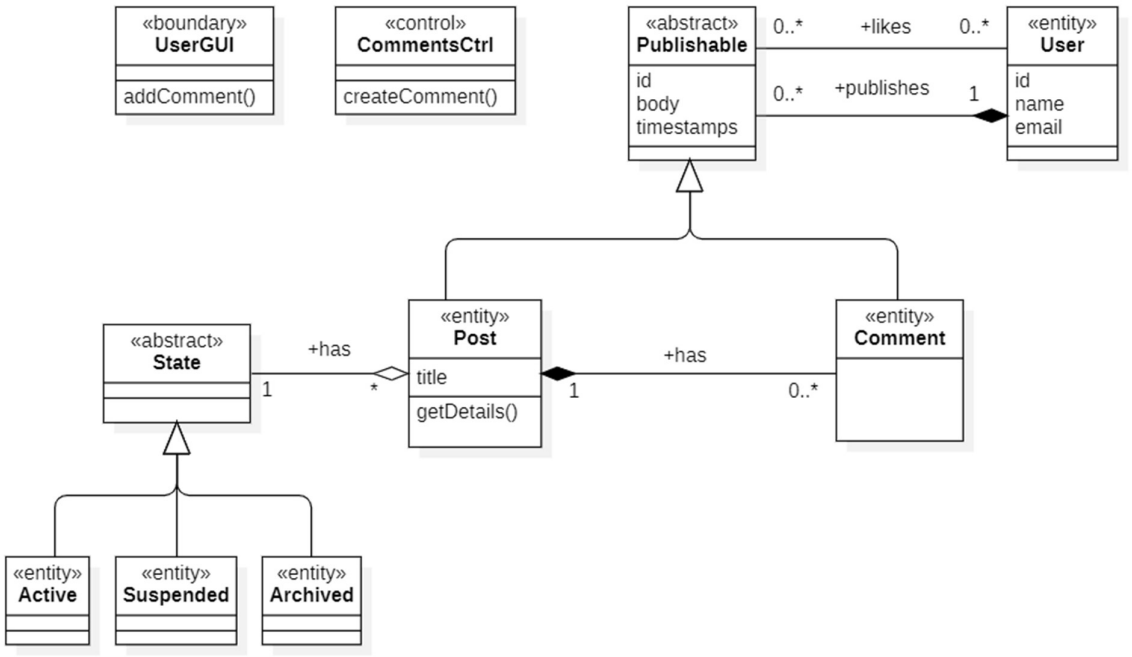
// le pattern identifié dans la partie 3.4 ainsi que toute forme d'abstraction utile (1.25 pts **bonus**)



// les formes agrégation ou composition sont acceptées pour les associations *publishes* (User-Post et User-Comment) et *has* (Post-Comment).

// 0.25 pour chaque classe entité/association

Solution avec ébauche, factorisation et pattern State :



// 2.0 points bonus : 0.75 pour les classes boundary et control et la méthode *getDetails*()+ 0.75 point pour le pattern State + 0.5 pour la factorisation Post-Comment.

3.4 Dynamique :

- quelles sont les classes qui doivent avoir des machines à états et pourquoi ?

La classe Post parce qu'elle change d'état : *active*, *suspended* et *archived* **(0.5 point)**

- quel est le pattern à utiliser dans ce genre de situations ?

State Pattern **(0.5 point)**