DEVELOPPEMENT D'UN BLOG

Concevoir une application en PHP

Résumé

https://github.com/sebastien-chomy/oc_blog

Sébastien CHOMY

Cadre du projet

Dans le cadre du parcours de ma formation de développeur d'application Web PHP/Symfony, l'objectif du cinquième projet est de développer son propre blog professionnel en PHP sans l'usage d'un framework. Le but est de convaincre les futurs employeurs/clients de mes talents de développeur d'application Web en environnement PHP.

La maitrise du langage PHP n'est pas le seul point de compétences à atteindre, une bonne organisation des tâches et une parfaire analyse et compréhension du projet sont tout aussi important, de part une démarche UML et la prise en main de GitHub.

Il est temps de s'approprier le projet, alors venez apprécier les nombreux diagrammes UML accompagnés d'explication dans cet exposé.

TABLE DES MATIERES

PARTIE I - ANALYSE

1 - ANALYSE DU BESOIN

1 | 1 - IDENTIFICATION DES ACTEURS

Les acteurs pour le blog sont les suivants :

VISITEUR

Un visiteur parcourant et consultant les différents articles du blog avec la possibilité de donner son avis par le biais de commentaires.

AUTEUR (DE COMMENTAIRES)

Un visiteur qui commente un article doit s'identifier en fournissant son adresse Email. L'adresse Email du visiteur internaute devient le liant entre les différents commentaires d'un même auteur.

BLOGUEUR

Rédacteur des articles du blog ; un droit alloué lors de la création de son compte par l'administrateur du site. Il peut y avoir plusieurs rédacteurs.

ADMINISTRATEUR

Gestionnaire de l'application sur l'ensemble des fonctionnalités de celle-ci.

1 | 2 - LES INTENTIONS METIER DES ACTEURS SUR LE SYSTEME

Nos acteurs, que souhaite-t-il faire sur le site de vente en ligne ?

Le visiteur

- Doit pouvoir consulter les articles du blog.

L'auteur (de commentaire)

- Doit pouvoir faire un commentaire sur un article.

Le blogueur

- Doit pouvoir s'authentifier
- Doit pouvoir créer un nouvel article.
- Doit pouvoir modifier un article.

L'administrateur pourra

- Gérer les commentaires laissés par les auteurs de commentaires (validation, modification, suppression)
- Gérer les articles (Publication, modification, suppression, archivage)
- Gérer les utilisateurs

1 | 3 - HIERARCHIE DES ROLES DES ACTEURS DU SYSTEME

Chaque acteur est lié à un rôle qui lui confère un ensemble de droit de consulter ou de modifier les données du système. Afin de ne pas répéter les relations entre les cas d'utilisation et acteur, il est établi une hiérarchie entre les différents rôles décrits ci-dessous, représenté par une relation de généralisation (flèche fermé vide) entre acteur sur le diagramme de cas d'utilisation exposé ci-après.

Par exemple : Un « Blogueur » à ses droits alloués plus ceux du rôle du « Auteur ».

Acteurs : Visiteur < Auteur < Blogueur < Administrateur

Rôles : Guest < Author < Blogger < Admin

2 - APERÇU DES CONTENUS

Nous rappelons les exigences fonctionnelles de notre client et ses choix en termes présentations des interfaces. Seules les interfaces remarquables sont exposées ici.

2 | 1 - PIED DE PAGE « MENU »

- L'ensemble des liens vers les réseaux sociaux où l'on peut vous suivre (Github, LinkedIn, Twitter…).
- Lien pour accéder à l'administration du blog (Rôle « Administrateur »)

2 | 2 - PAGE « ACCUEIL »

Le contenu principal de la page d'accueil se distingue en deux parties :

EN ENTETE

- Votre nom et prénom
- Une photo et/ou un logo
- Une phrase d'accroche qui vous ressemble (exemple : "Martin Durand, le développeur qu'il vous faut !")
- Lien vers votre CV au format PDF

FORMULAIRE DE CONTACT

- Un formulaire de contact (à la soumission de ce formulaire, un email avec toutes ces informations sera envoyé à l'administrateur) avec les champs suivants :
    - Nom/prénom
    - Email de contact
    - Message

2 | 3 - PAGE « DES ARTICLES »

Cette page présente une liste des articles (du plus récent au plus ancien). Il faut afficher les informations suivantes pour chaque article :

- Le titre
- La date de dernière modification
- Le chapô (brève description de l'article)
- Et un lien vers la page article.

2 | 4 - PAGE « ARTICLE »

Sur la page « Article » le contenu principal se décompose en trois parties :

L'ARTICLE

- Le titre
- Le chapô (brève description de l'article)
- Le contenu
- L'auteur
- La date de dernière mise à jour
- Un lien pour modifier l'article.

FORMULAIRE D'AJOUT D'UN COMMENTAIRE EN RELATION A UN ARTICLE

Un simple formulaire composé d'une zone de texte sur plusieurs ligne (textarea) d'un bouton de soumission pour validation. Un commentaire laissé par un visiteur doit être validé par le rôle « administrateur » avant d'être publié à l'ensemble des visiteurs.

LES COMMENTAIRES PUBLIES

Liste des commentaires validés et publiés dans l'ordre du plus récent au plus ancien. Pour chaque commentaire les informations suivantes seront visibles : le contenu du commentaire, la date et heure de rédaction, l'auteur sous son pseudo.

2 | 5 - PAGE « UN NOUVEL ARTICLE »

Il s'agit de la page de rédaction d'un nouvel article, prenant l'apparence d'un formulaire.

Les différents champs de saisies seront :

- Titre
- Chapô
- Contenu

Note : L'auteur de l'article est le blogueur authentifié courant.

2 | 6 - PAGE « MODIFIER UN ARTICLE »

Il s'agit de la page de modification du contenu d'un article, sous une apparence identique du formulaire de création « un nouvel article ». Notez que seules les informations suivantes sont modifiables :

- Titre
- Chapô
- Contenu

2 | 7 - PAGE « GESTION DES ARTICLES »

Page d'administration listant l'ensemble des articles sur lesquelles les actions suivantes sont possibles :

- Modifier un article
- Supprimer un article
- Archiver un article
- Publier un article

Cette page fait partie du back-office

2 | 8 - PAGE « GESTION DES COMPTES UTILISATEURS »

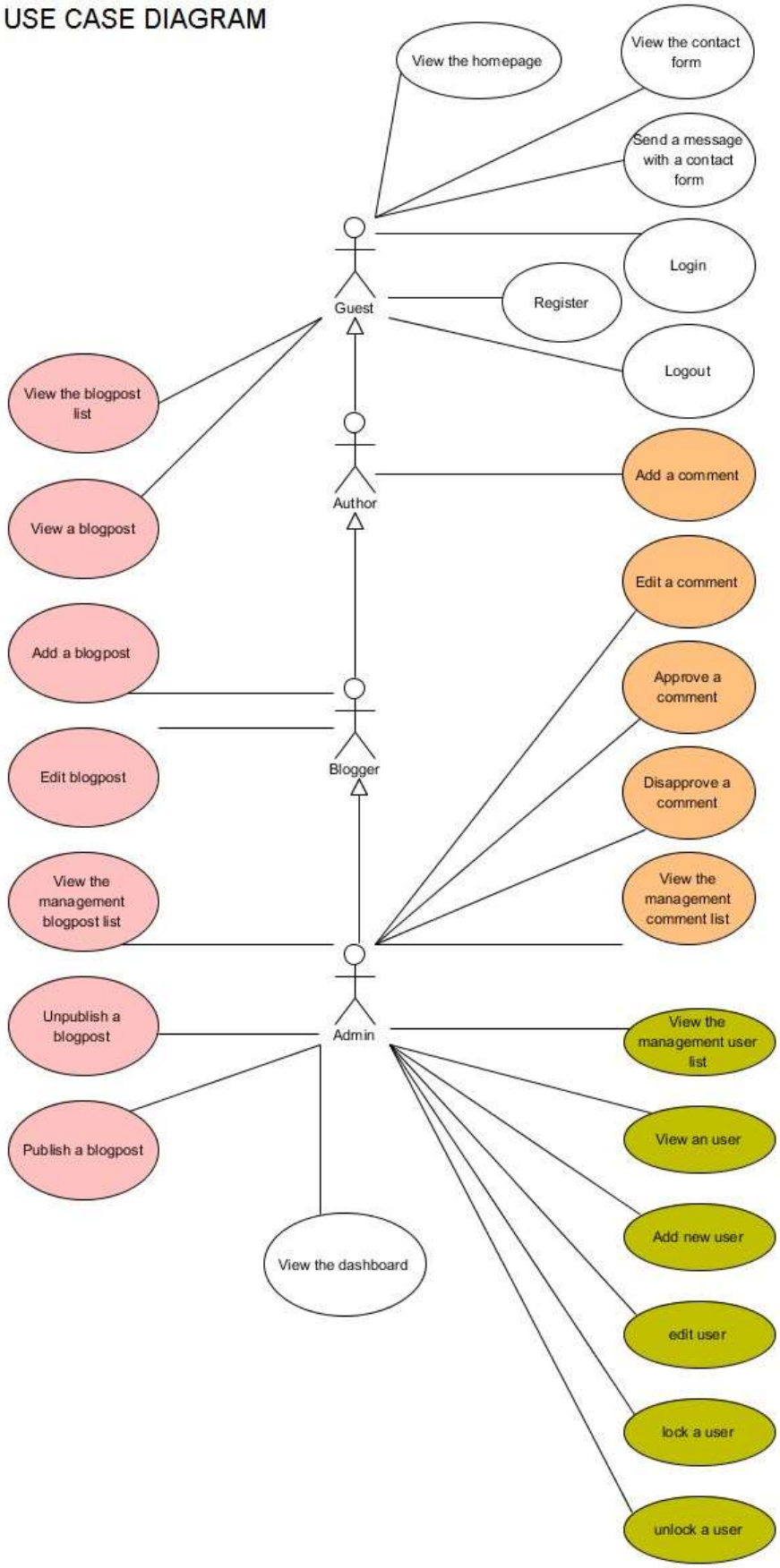
Page d'administration listant l'ensemble des utilisateurs sur lesquelles les actions suivantes sont possibles :

- Verrouiller ou déverrouiller un utilisateur
- Modifier les données de l'utilisateur (Nom, Prénom, Email, Rôle)
- Un lien pour créer un nouvel utilisateur

3 - DIAGRAMME DE CAS D'UTILISATION

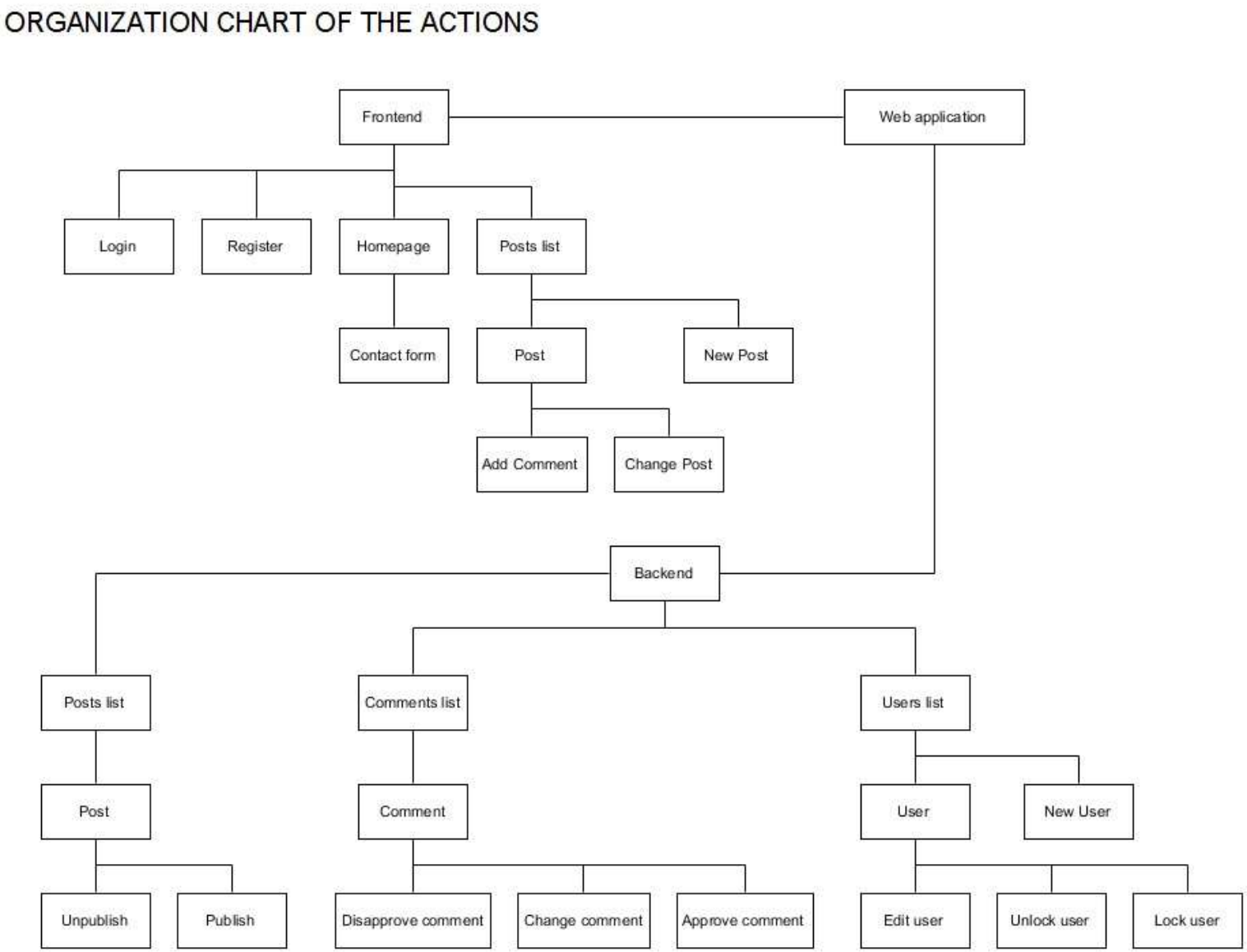
Le diagramme de cas d'utilisation (Use Case) représente un ensemble de séquences d'actions qui sont réalisées par le système et qui produisent un résultat observable intéressant pour un acteur en particulier.

Afin de rapidement représenter les intentions métier de chaque acteur, nos différents cas d'utilisation sont réduits à une simple action.



4 - DIAGRAMME DE NAVIGATION SIMPLIFIE

Ce diagramme montre l'arborescence des différents cas d'utilisation présents dans les interfaces de l'application



4 | 1 - TABLEAU DE SYNTHESE

Ce tableau liste l'ensemble des cas d'utilisation avec un rapprochement avec l'architecture du code. Rappelons que nos cas d'utilisation sont réduits à une simple action et non à une séquence d'actions.

Domaine	Controller	Action	Route	View	Use case
App	Homepage	homepage	/	homepage/homepage	View the homepage
App	ContactUs	contactUs	contact	contactUS/contactUS	View the contact form
App	Dashboard	index	admin	dashboard/index	View the dashboard
User	Security	login	login	security/login	login
User	Security	logout	logout	No render	logout
User	Registration	register	registration	registration/register	Registrer
User	Management	getUsers	admin/users	management/userList	View the management users list
User	Management	getUser	admin/user/{userID}	management/user	View an user
User	Management	postUser	admin/addUser	management/newUser	Add an user
User	Management	putUser	admin/changeUser/{userID}	management/changeUser	Edit an user
User	Management	lock	admin/user/lock/{userID}	No Render	Lock an user
User	Management	unlock	admin/user/unlock/{userID}	No Render	Unlock an user
Blogpost	GetBlogposts	getBlogposts	posts	blogpost/blogpostList	View the blogpost list
Blogpost	GetBlogpost	getBlogpost	post/{postID}	blogpost/blogpost	View a blogpost
Blogpost	PostBlogpost	postBlogpost	newPost	blogpost/newPost	Add a blogpost
Blogpost	PutBlogpost	putBlogpost	changePost/{postID}	blogpost/changePost	Edit a blogpost
Blogpost	Management	getPosts	admin/posts	management/postList	View the managmement blogposts list
Blogpost	Management	publish	admin/post/publish/{postID}	No render	Publish a blogpost
Blogpost	Management	unpublish	admin/post/unpublish/{postID}	No render	Unpublish a blogpost
Comment	PostComment	postComment	newComment	comments/newComment	Add a comment
Comment	Management	getComments	admin/comments	management/commentList	View the management comments list
Comment	Management	putComment	admin/comment/change/{commentID}	management/changeComment	Edit a comment
Comment	Management	approve	admin/comment/approve/{commentID}	No render	Approve a comment
Comment	Management	disapprove	admin/comment/disapprove/{commentID}	No render	Disapprove a comment

Partie II - Conception

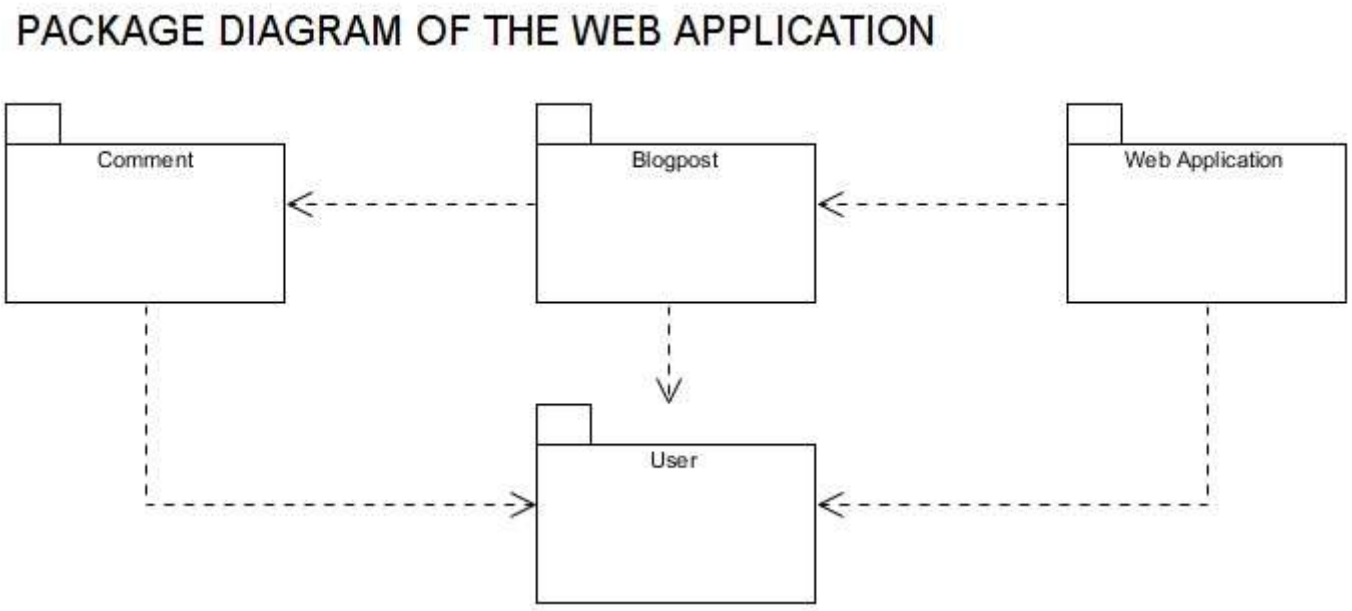
5 - INFRASTRUCTURE DE L'APPLICATION

Lors de la construction d'une application sans l'appui d'un Framework qui vient avec sa structure de répertoires, il est important de choisir une structure.

5 | 1 - DIAGRAMME DE CONTEXTE DE L'APPLICATION

L'application se découpe en quatre grands contextes, chacun responsable d'une seule logique métier (bounded context)

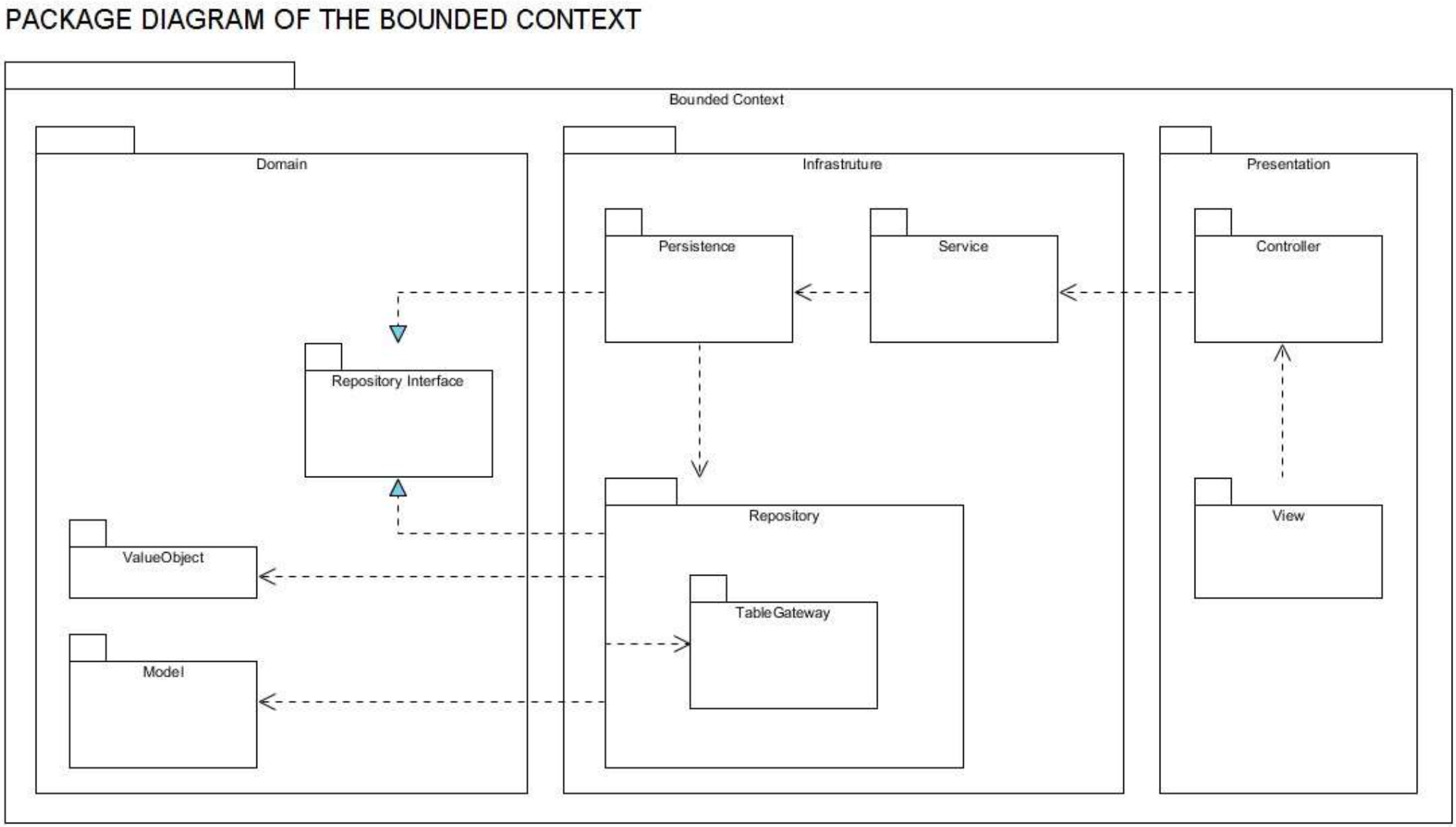
- Contexte « Web application » orchestre l'application.
- Contexte « Blogpost » pour la gestion des articles.
- Contexte « Comment » gère les commentaires des visiteurs relatifs aux articles du blog.
- Contexte « User » pour la gestion des utilisateurs et droit d'accès.



Les flèches en pointillés représentent les dépendances logiques d'usage entre les contextes.

5 | 2 - DIAGRAMME DE PACKAGE D'UN CONTEXTE

Chaque contexte est lui-même découpé en package hiérarchiser visible sur ce diagramme.



ARCHITECTURE EN COUCHE

Il est important de donner des explications sur ce diagramme de package d'un contexte. Tout d'abord, un contexte se compose de trois couches.

Couche présentation	Cette couche est responsable de présenter une interface utilisateur basé sur HTML, permettant aux utilisateurs d'utiliser la logique métier. La couche de présentation contient toutes les ressources concernées par la création d'une interface utilisateur rendue côté serveur pour l'utilisateur final.
Couche domaine	Cette couche contient les informations sur le domaine métier du contexte. C'est le cœur du logiciel métier. L'état des objets métier est renfermé ici. La persistance des objets métier et peut-être aussi leur état est délégué à la couche infrastructure.
Couche infrastructure	Cette couche agit comme une bibliothèque de soutien pour toutes les autres couches. Elle fournit la communication entre les couches, implémente la persistance des objets métier.

Découvrons plus en détail les aspects de notre diagramme de package d'un contexte.

COUCHE PRESENTATION

Cette couche est responsable de présenter une interface utilisateur basé sur HTML, c'est donc tout naturellement que l'on y retrouve les contrôleurs (Controller) et les vues (View). Dans notre cas nous suivons le motif de conception « Modèle-Vue-Contrôleur ». Le modèle est délégué à la couche infrastructure.

COUCHE DOMAINE

Cette couche contient les informations sur le domaine métier, composé de trois sous-dossiers.

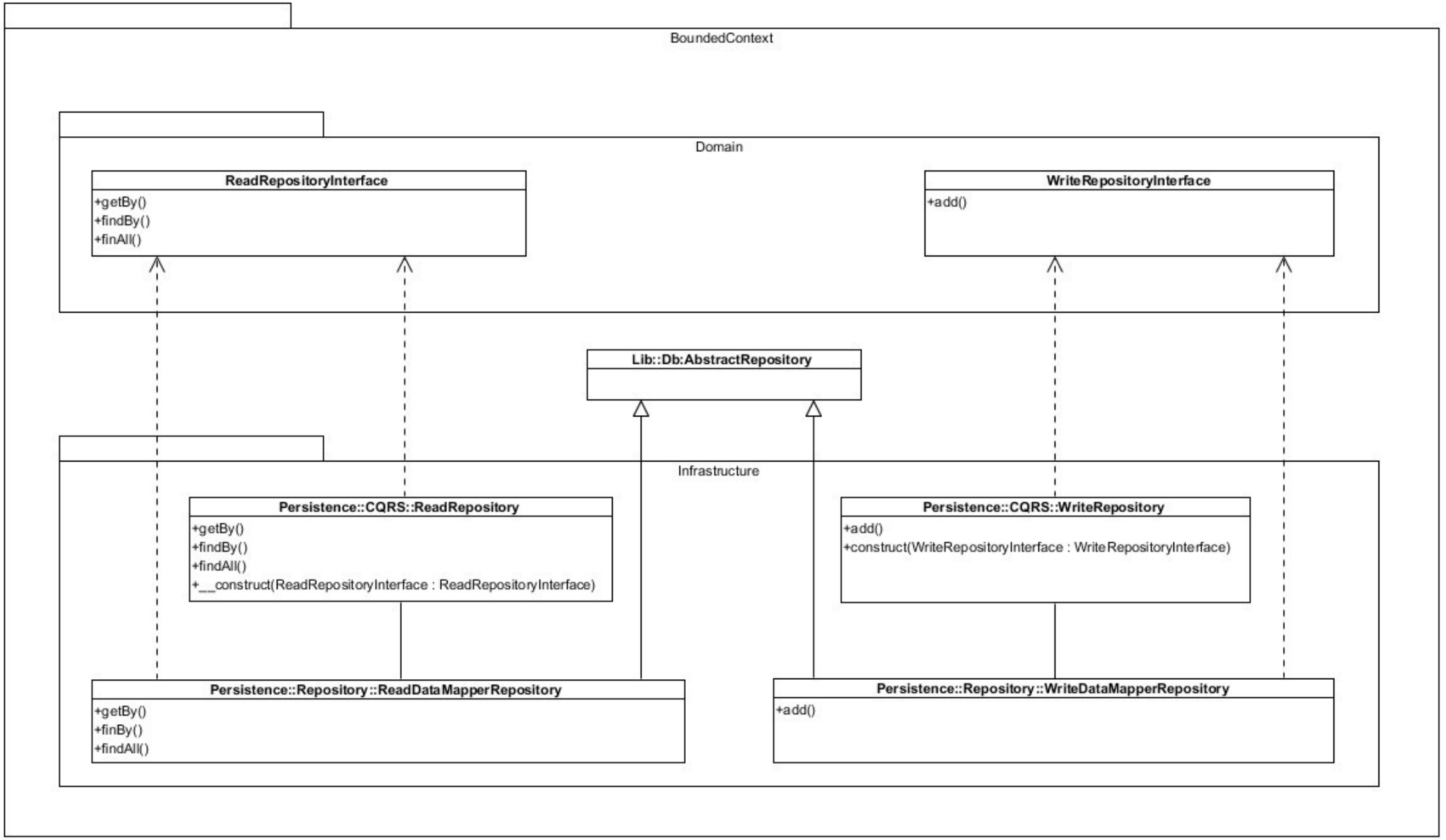
Model	Dossier des seules classes de la logique métier. Les classes suffixés « Aggregate » jouent le rôle d'agrégateur d'objet métier mais ne sont pas persistés.
ValueObject	Dossier des seules classes jouant le rôle d'identificateur unique (UUID) entre objet métier du dossier **Model**. En effet nous souhaitons générer un lien unique entre objets métier sans coordination externe. L'intérêt est que nous ne sommes pas dans l'attente d'un identifiant unique de type clé primaire de la base de données, puisque nos UUID sont générés côté applicatif.
Repository	Ce dossier contient toutes classes d'interface qui définit la manière dont les objets du domaine métier et les sources de données doivent communiquer. Notre couche domaine n'a pas besoin de savoir quelle source de données est implémentée et comment elle est implémentée, c'est le rôle de la couche infrastructure. La couche domaine doit définir via des interfaces son contrat de communication.

COUCHE INFRASTRUCTURE

Cette couche contient le mécanisme de consultation et persistance des sources de données de nos objets métier de la couche domaine. Elle est composée de quatre sous dossiers.

Persistence	Nous avons souhaité définir une logique d'ignorance de la persistance de la source de données qui permet au code de la couche domaine de rester indifférent à la technologie utilisée de persistance. Sur cette logique, toutes les classes de ce dossier implémenteront les interfaces définies dans la couche domaine (Domain\Repository). Nous souhaitons également pourvoir permuter sur le choix d'un dépôt (repository). Le dépôt peut être défini par son mécanisme de persistance en mémoire, fichier ou traditionnellement en base de données. Pour notre code, le constructeur d'une classe de persistance attend en argument le dépôt (repository) choisi. L'avantage est que nous ne sommes donc pas tributaires du choix final du mode de stockage des sources des données.
Repository	Nous avons souhaité au niveau de la persistance de retarder le choix de notre dépôt. Notre choix final sur le mode de stockage est une base de données relationnelles. Nous suivons ici un motif de conception dit *Data mapper* https://martinfowler.com/eaaCatalog/dataMapper.html Les objets métiers de la couche domaine et notre base ont des mécanismes différents pour structurer les données. De nombreuses parties d'un objet, telles que les collections et l'héritage, ne sont pas présentes dans les bases de données relationnelles. Le Data Mapper va faire le transfert des données entre le schéma structurel d'objet et le schéma relationnel. De plus, les classes devront implémenter les interfaces définies dans notre couche domaine *BoundedContext\Domain\Repository* pour respecter le contrat de communication entre ces deux couches.
Repository \TableGateway	Ce dossier contient toutes les classes d'objets qui utilise un motif de conception dit *Table Data Gateway*. https://martinfowler.com/eaaCatalog/tableDataGateway.html. Ce motif agit comme une passerelle entre une table de base de données et une classe. Chaque classe contient tout le code SQL au travers de ses méthodes qui offre une manière simple d'effectuer des opérations sur la table en question. Les opérations SQL comme *Insert* et *update*, ont un comportement généralisé pour toutes les tables. Pour cette raison nous faisons usage d'une class abstraire Lib\Db\AbstractTableGateway embarquant notamment de telle méthode.
Service	Ce dossier contient toutes les classes de service. Un service est la partie visible de la couche structure. Il apporte également une simplification de la complexité de la structure de cette couche. Ces classes de services sont essentiellement utilisés par la couche présentation ou un autre service du même contexte.

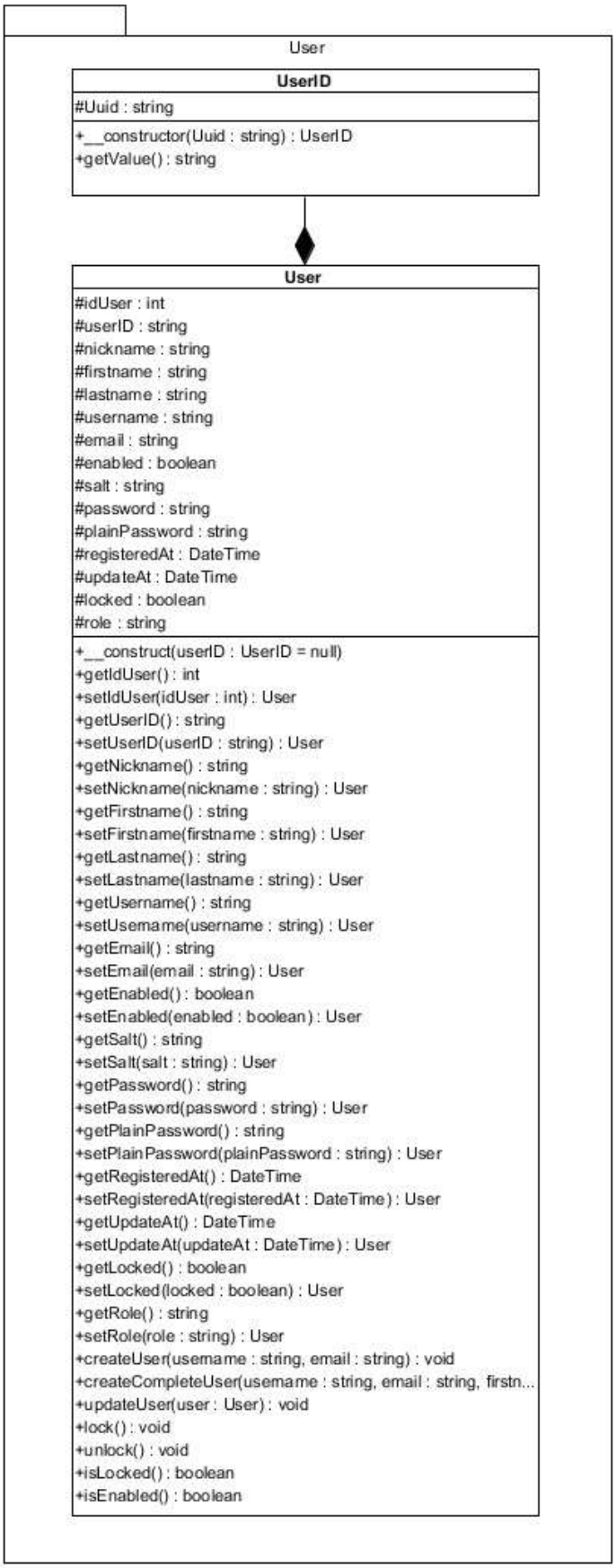
5 | 3 - DIAGRAMME SIMPLIFIE DE LA COUCHE STRUCTURE



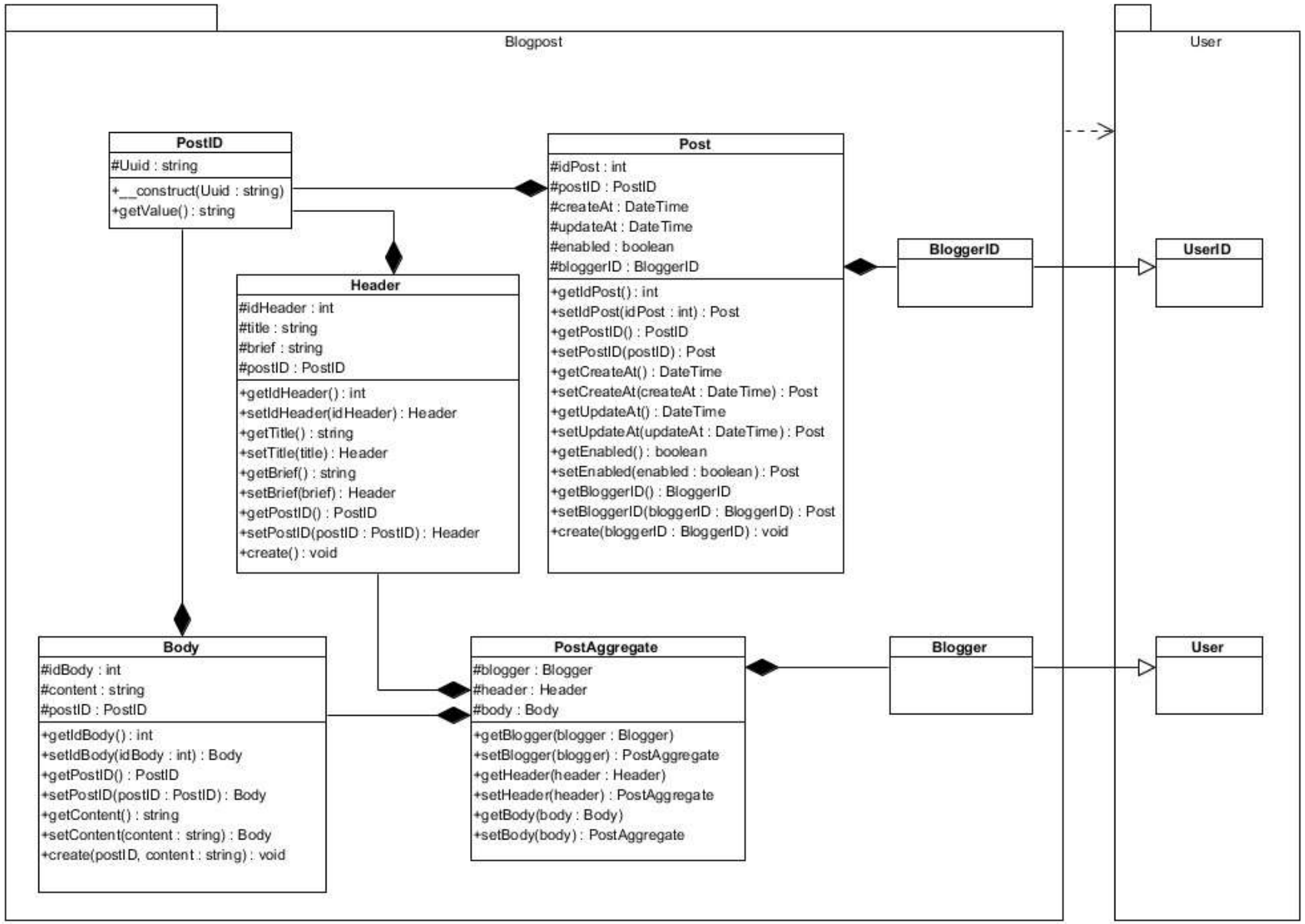
6 - LOGIQUE METIER

6 | 1 - CONTEXTE USER

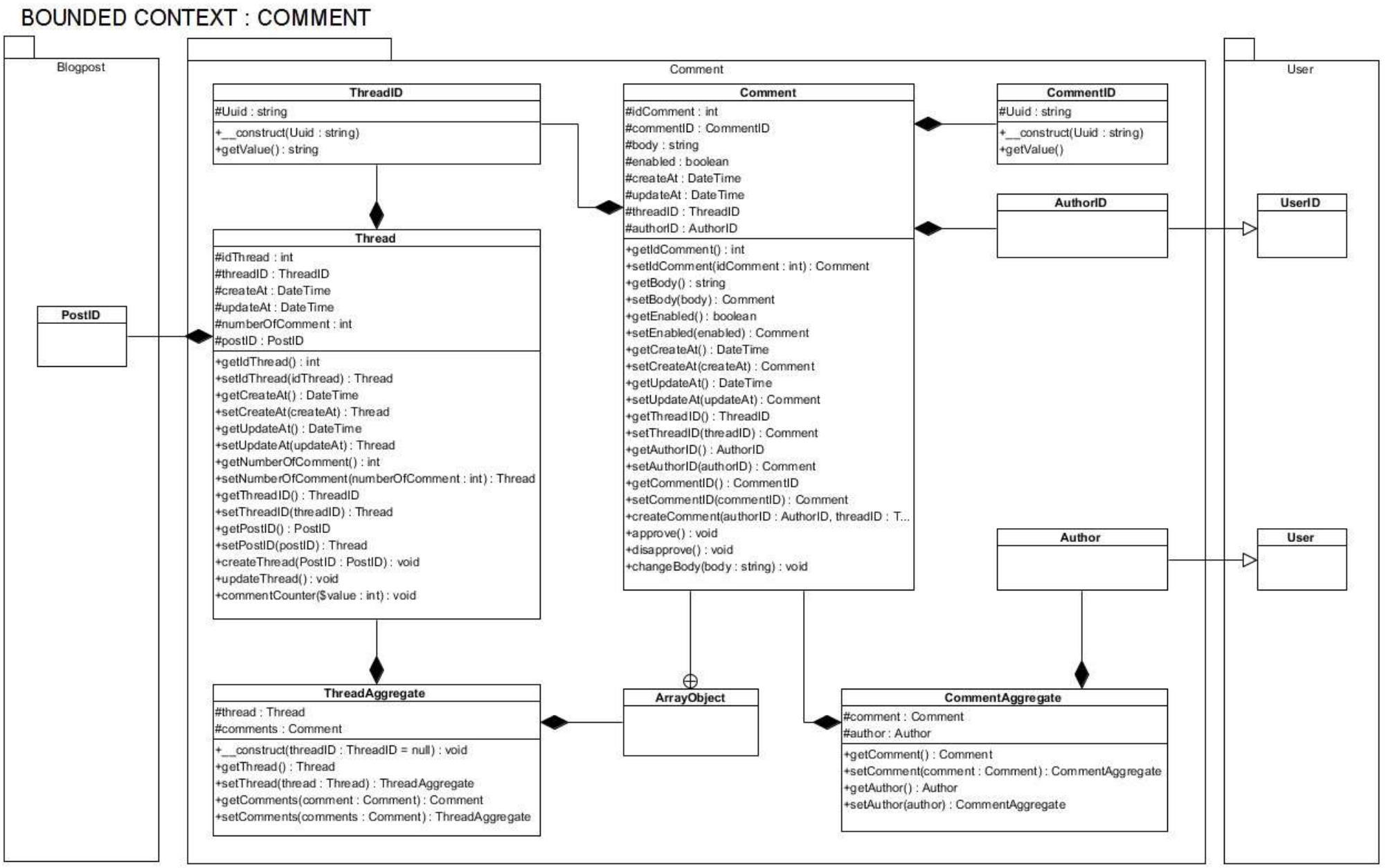
BOUNDED CONTEXT : USER



BOUNDED CONTEXT : BLOGPOST



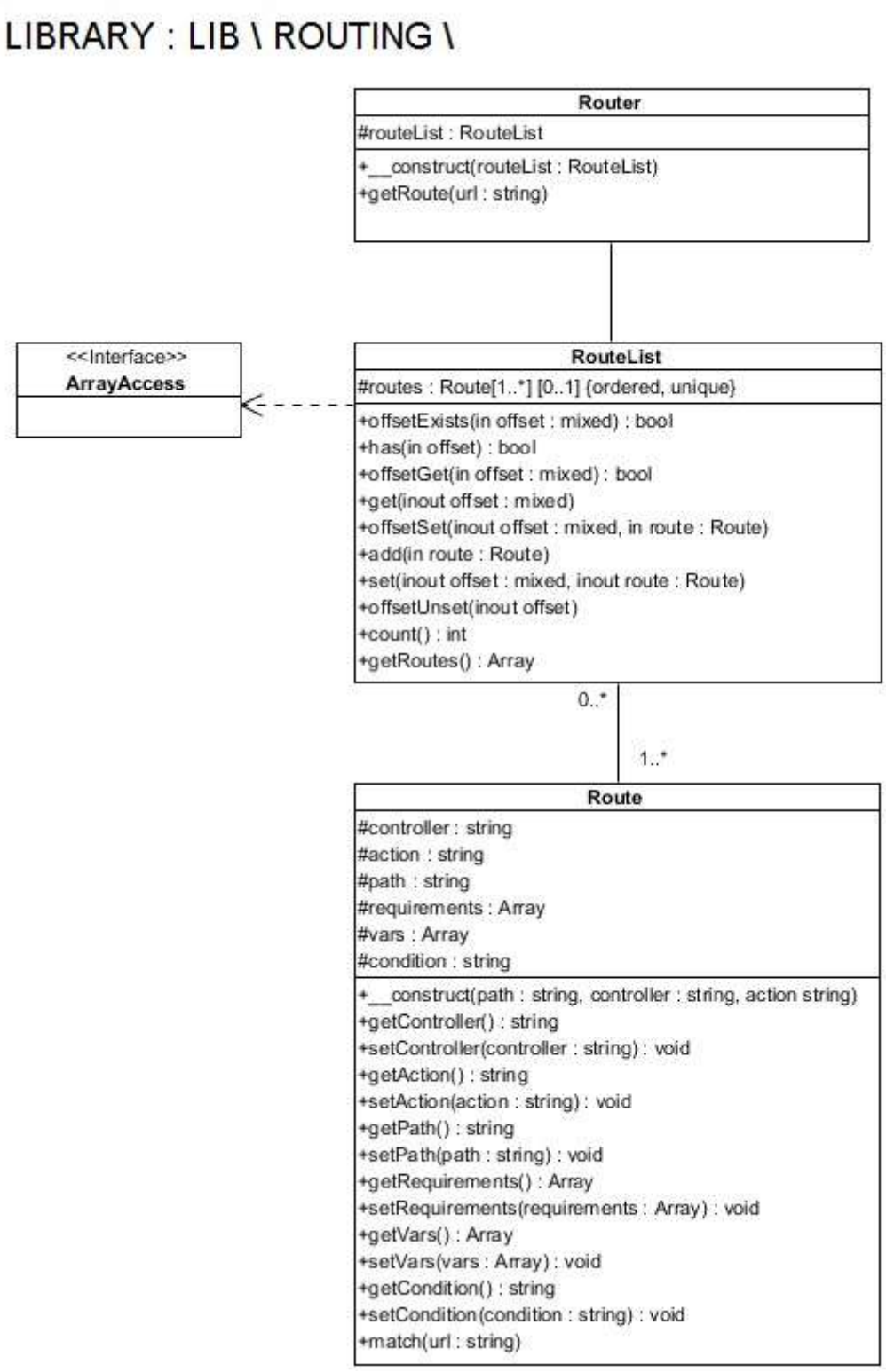
6 | 3 - CONTEXTE COMMENT



7 - LIBRAIRIES

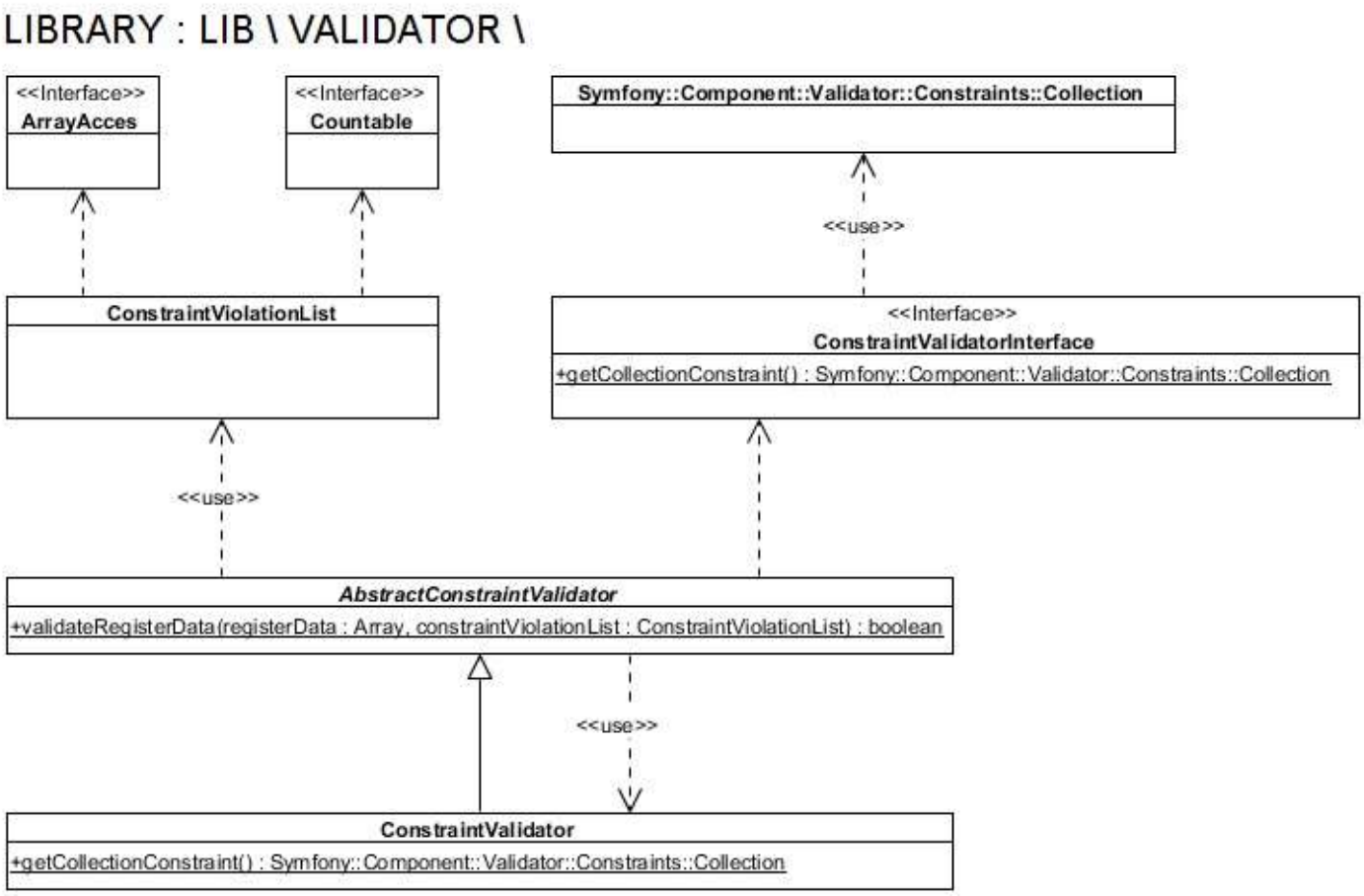
7 | 1 - ROUTEUR

Le routeur est l'objet qui va nous permettre de savoir quelle page nous devons exécuter. Pour en être capable, le routeur (Router) aura à sa disposition une liste (RouteList) de route (Route) pointant chacune vers un module et une action. En passant en paramètre l'URL à la méthode Router::getRoute(), celle-ci retournera un objet Route



7 | 2 - VALIDATEUR

Le validateur (ConstraintValidator) est la classe qui va permettre de valider les données reçues de nos formulaires. A la méthode (ValidateRegisterData) nous passons en premier arguments ces données issues du formulaire et comme le validateur par sa méthode (GetCollectionConstraint).est conscient des contraintes de validation, le validateur rapproche les données avec les contraintes et retourne un booléen de résultat pour la méthode (ValidateRegisterData). Dans le cas où la validation est fausse l'objet (ConstraintViolationList) passé en second argument est renseigné avec le détail des erreurs de validations.



8 - LES LIBRAIRIES EXTERNES

L'application fait usage de librairies externes pour son backend et frontend. Les gestionnaires de dépendances Composer et Yarn sont respectivement utilisés.

8 | 1 - BACKEND

Nous ferons usage de Composer.org https://getcomposer.org/ pour intégrer au backend nos dépendances suivantes :

twig/twig	Moteur de template
Guzzlehttp/psr7	Ce référentiel contient une implémentation complète PSR-7 : http message interfaces.
Ramsey/uuid	Générateur Universal Unique Identifier
Symfony/dotenv	Accéder aux variables d'environnements
Swiftmailer/swiftmailer	Envoyer des mails
Symfony/validator	Pour valider des valeurs en fonction de contraintes définies

8 | 2 - FRONTEND

Nous ferons usage de **Yarn** https://yarnpkg.com/fr/ pour intégrer au frontend nos dépendances suivantes :

JQuery https://yarnpkg.com/fr/package/jquery	JQuery
Bootstrap https://yarnpkg.com/fr/package/bootstrap	Le Framework CSS Bootstrap version 4
Startbootstrap-clean-blog https://yarnpkg.com/fr/package/startbootstrap-clean-blog	Pour le front office, thème s'appuyant sur Bootstrap V.4
Boosted - orange-opensource https://yarnpkg.com/fr/package/boosted http://boosted.orange.com/docs/4.0/examples/	Pour le back-office, thème s'appuyant sur Bootstrap V.4

9 - QUALITE DU CODE ET BONNE PRATIQUE

9 | 1 - PHP 7.1 / PSR-2 : CODING STYLE GUIDE

Cela passe par la mise en place d'un typage fort lors de l'écriture du code PHP, le respect du standard PSR-2 http://www.php-fig.org/psr/psr-2/

9 | 2 - CODACY

Coté backend, la qualité du code du projet est suivie par CODACY https://app.codacy.com

9 | 3 - GOOGLE TOOLS : LIGHTHOUSE

Coté Frontend, la qualité du code du projet est suivie par Lighthouse

https://developers.google.com/web/tools/lighthouse/

10 - GESTION DE VERSION DU CODE, LES REGLES APPLIQUES

Le projet n'a pas été versionné lors de sa phase de développement de la première version, toutefois nous rappelons les règles

https://semver.org/lang/fr/

Gestion sémantique de version 2.0.0

En bref

Étant donné un numéro de version MAJEUR.MINEUR.CORRECTIF, il faut incrémenter :

- Le numéro de version MAJEUR quand il y a des changements non rétro compatibles,
- Le numéro de version MINEUR quand il y a des changements rétro compatibles,
- Le numéro de version de CORRECTIF quand il y a des corrections d'anomalies rétro compatibles

Des libellés supplémentaires peuvent être ajoutés pour les versions de pré-livraison et pour des métadonnées de construction sous forme d'extension du format MAJEURE.MINEURE.CORRECTIF.

Pour gérer les différentes versions de son application sur un dépôt Git, plusieurs branches seront nécessaires :

- La branche principale « master » qui reçoit tous les changements.
- Des branches nommées sur le modèle des numéros de version MAJEUR.MINEUR (exemple : 1.0, 1.1, 2.0)

➢ Des références sur le sujet

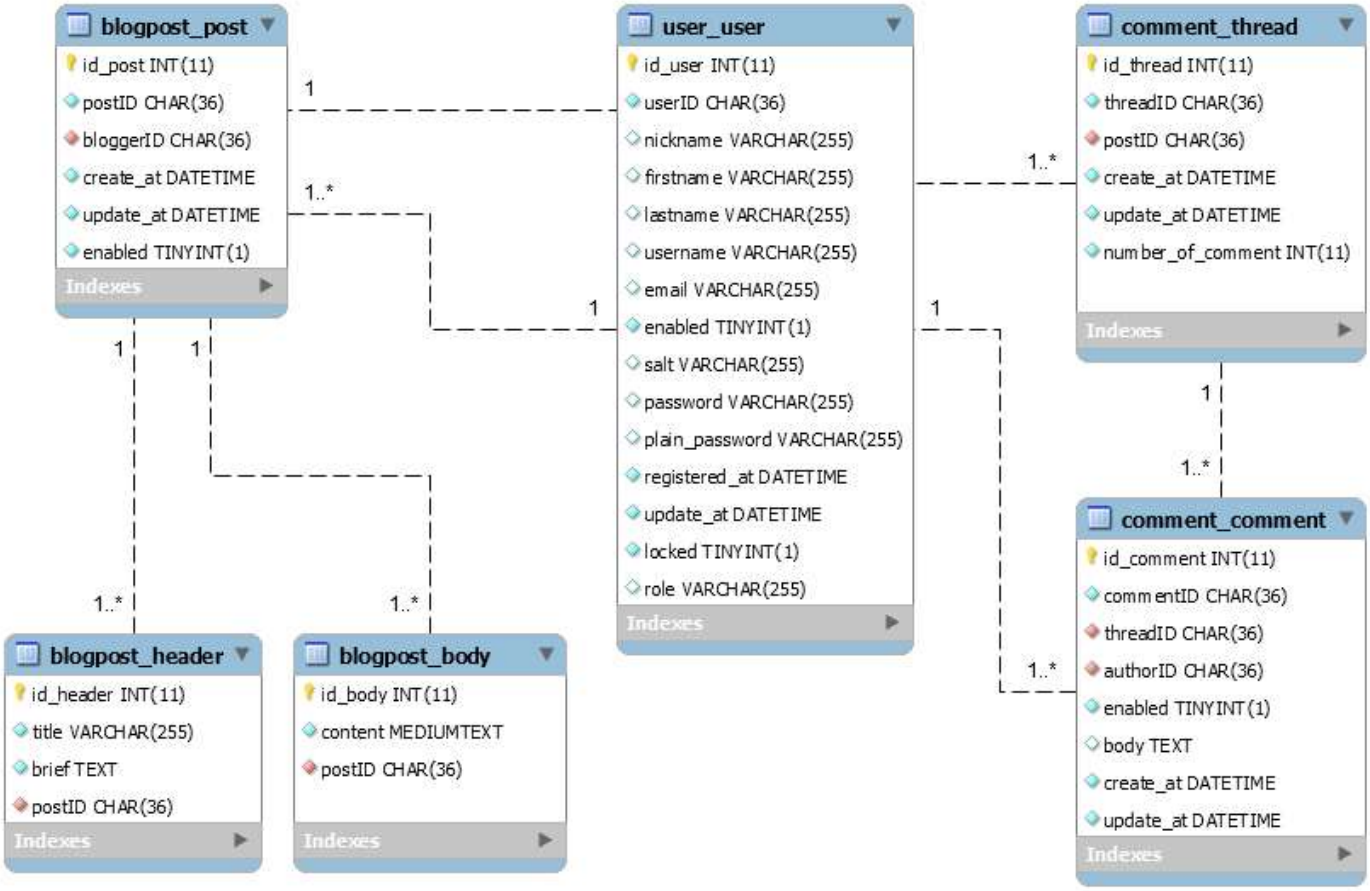
Tester les contraintes de version.

https://semver.mwl.be/

Composer.org

https://getcomposer.org/doc/articles/versions.md

11 - MODELE PHYSIQUE DE DONNEES



12 – DIAGRAMME DE SEQUENCE « AJOUT D'UN COMMENTAIRE »