



APPLICATION/TODOLIST

DOSSIER D'ARCHITECTURE TECHNIQUE

Objectif du document

Ce document est un dossier d'architecture technique destiné aux développeurs en charge du développement, de la mise en production et de la maintenance de l'application TODOLIST développée à l'aide du framework **Symfony**.

Il a pour objectif de permettre aux nouveaux développeurs intégrant l'équipe d'avoir rapidement une vue d'ensemble sur l'architecture de l'application, d'expliquer dans le détail quelques mécanismes propres à l'application et de fournir une direction claire sur les méthodes et les normes de développement préconisées.

Ce document fournit aussi en annexe une liste d'axes d'améliorations possibles de l'application.

Version 1.0

Sommaire

OBJECTIF DU DOCUMENT.....	1
1. PREAMBULE.....	3
1.1 Lexique	3
1.2 Signalétique.....	3
2. ARCHITECTURE TECHNIQUE GENERALE	4
2.1 Schéma global d'architecture.....	4
2.2 Plateforme technique.....	4
2.3 Flux	5
3. VERSION DU FRAMEWORK SYMFONY.....	5
3.1 Version actuelle	5
3.2 Calendrier des maj.....	5
4. DESCRIPTION DES DONNEES	6
4.1 Modèle physique.....	6
5. DESCRIPTION DU CODE.....	7
5.1 Gestion de version	7
5.2 Architecture du code – Diagrammes UML	7
5.3 Arborescence du projet.....	10
5.4 Le fichier security.yaml.....	11
5.5 Mécanismes d'authentification	15
5.6 Mécanismes d'accréditation	16
6. NORMES ET STANDARDS DE REALISATION	16
6.1 Développement de l'application.....	16
6.2 Développement orienté par les tests.....	16
6.3 Normes de codage	17
7. QUALITE DU CODE ET PERFORMANCES DE L'APPLICATION	17
7.1 Audit de la qualite du code	17
7.2 Métrique des performances de l'application	18
8. ANNEXE : AXES D'AMELIORATIONS.....	19
8.1 Introduction	19
8.2 A court terme.....	19
8.3 A moyen terme	20
8.4 A long terme	20

1. PREAMBULE

1.1 LEXIQUE

Dans ce document, les abréviations suivantes seront utilisées:

maj : mise à jour



bdd : base de données

v : version

CRUD : create, read, update, delete

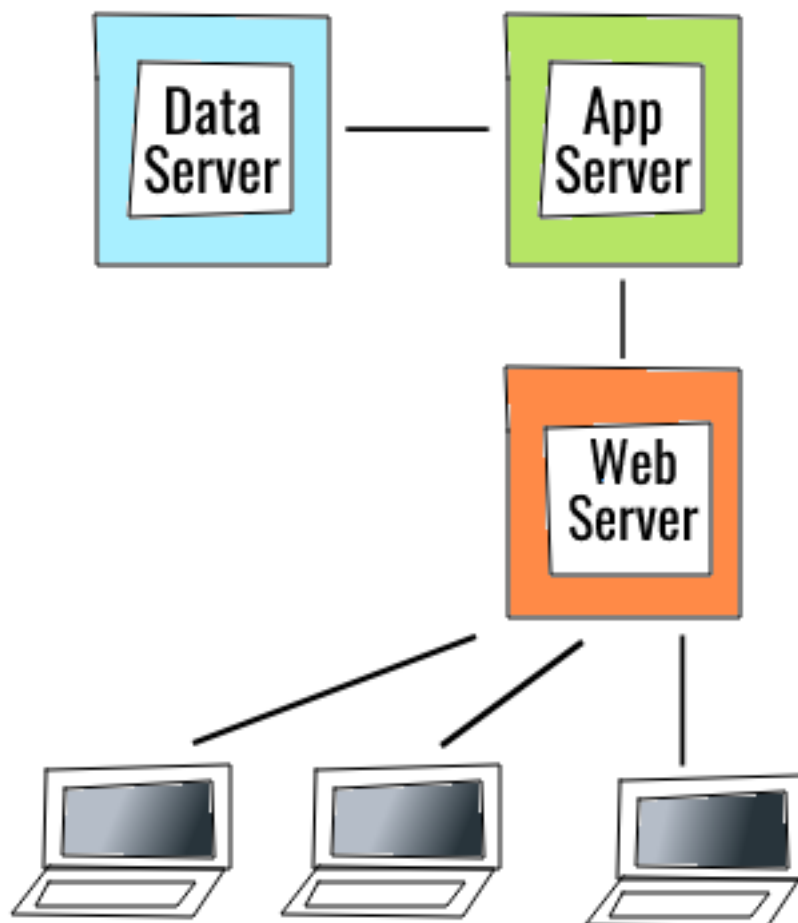
1.2 SIGNALÉTIQUE

Tout au long de ce document, les pictogrammes ci-dessous sont utilisés afin de souligner des points ou des notions importantes.

	Information importante
	Action obligatoire

2. ARCHITECTURE TECHNIQUE GENERALE

2.1 SCHEMA GLOBAL D'ARCHITECTURE



2.2 PLATEFORME TECHNIQUE

Type	OS/Plateforme	Logiciel	Version
Serveur Web	Linux	Apache	Apache >2.4
Serveur d'application	Linux	Tomcat	n/a
Serveur de base de données	Linux	MySQL	MySQL >5.7
Langage	PHP		> 7.1.3

2.3 FLUX

De	Vers	Visibilité / Protocole	Port
Client	Apache	Internet / HTTPS	A configurer

3. VERSION DU FRAMEWORK SYMFONY

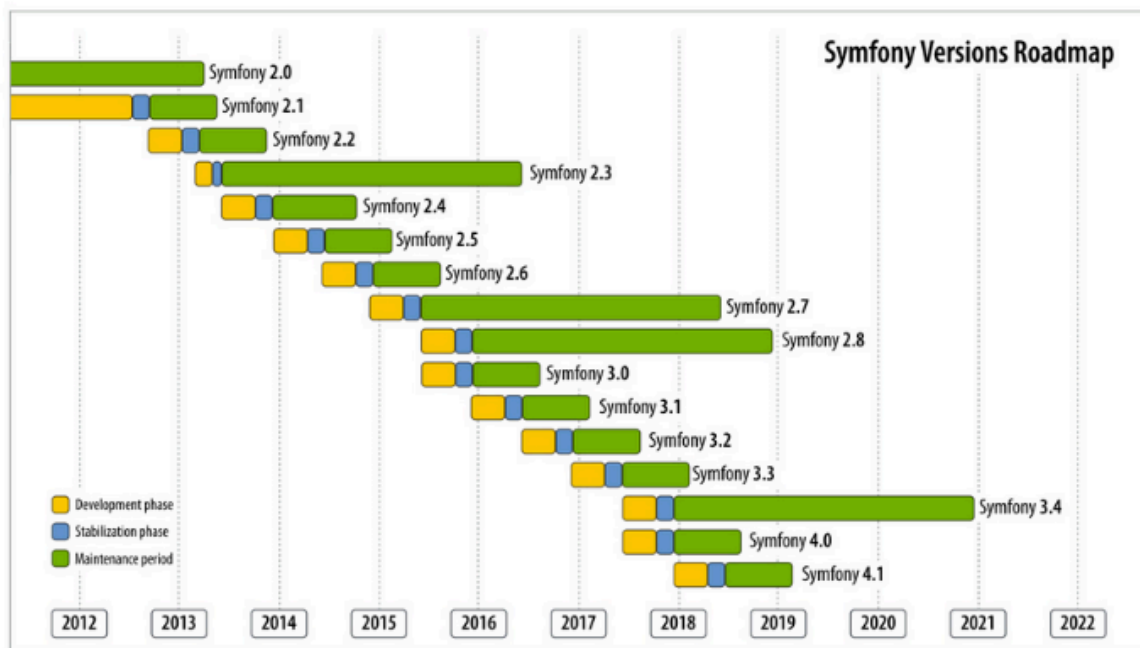
3.1 VERSION ACTUELLE



La version du framework a été montée de la version 3.1 à 4.1, c'est à dire la dernière version stable du framework (à la date de rédaction du document).

La version 3.1 n'était plus maintenue par SensioLabs.

3.2 CALENDRIER DES MAJ



- **Yellow** represents the Development phase
- **Blue** represents the Stabilization phase
- **Green** represents the Maintenance period

[Calendrier des mises à jour de Symfony](#)



Il est important de continuer à maintenir le framework à jour afin d'atteindre la prochaine version majeure 4.4 qui sera maintenue pendant trois ans.

Nous avons choisi de passer à la version 4 du framework afin d'assurer le meilleur développement de l'application sur le long terme.

Les maj mineures devront se faire dès que la version suivante entre dans sa phase de maintenance.

Promesse de compatibilité ascendante

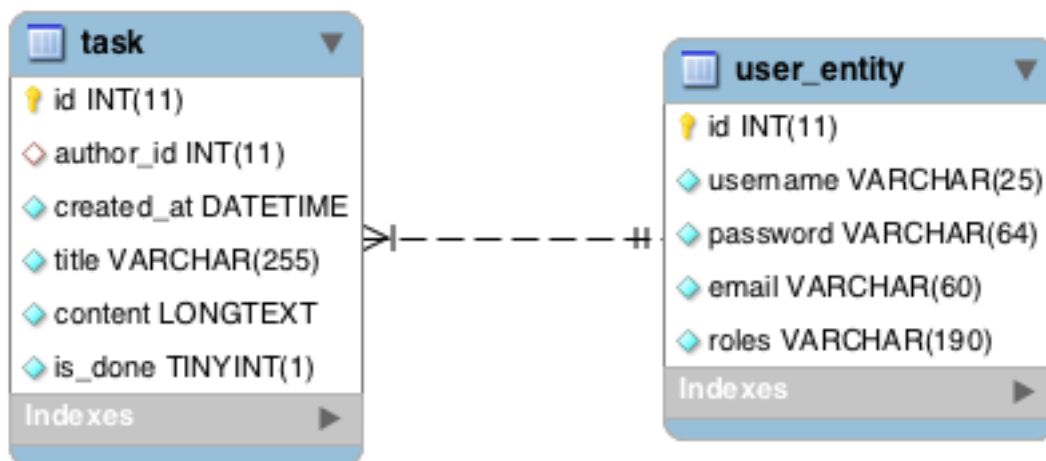
En se tenant informés des maj du framework, l'application sera plus facilement maintenue et les failles, dépréciations seront minorées.

Toute montée de version peut cependant entraîner un crash de l'application, c'est pourquoi la poursuite d'écriture de tests automatisés au cours du développement de l'application doit rester une priorité (voir la section normes et standards de réalisation).

4. DESCRIPTION DES DONNEES

4.1 MODELE PHYSIQUE

v 1.0



5. DESCRIPTION DU CODE

5.1 GESTION DE VERSION

L'ensemble du code et des différentes étapes ayant mené à son état actuel est disponible sur le [repository Github de l'application](#).

Une explication complète pour installer l'application est aussi disponible sur le repository de l'application.

5.2 ARCHITECTURE DU CODE – DIAGRAMMES UML

Diagramme de cas d'utilisation v 1.0

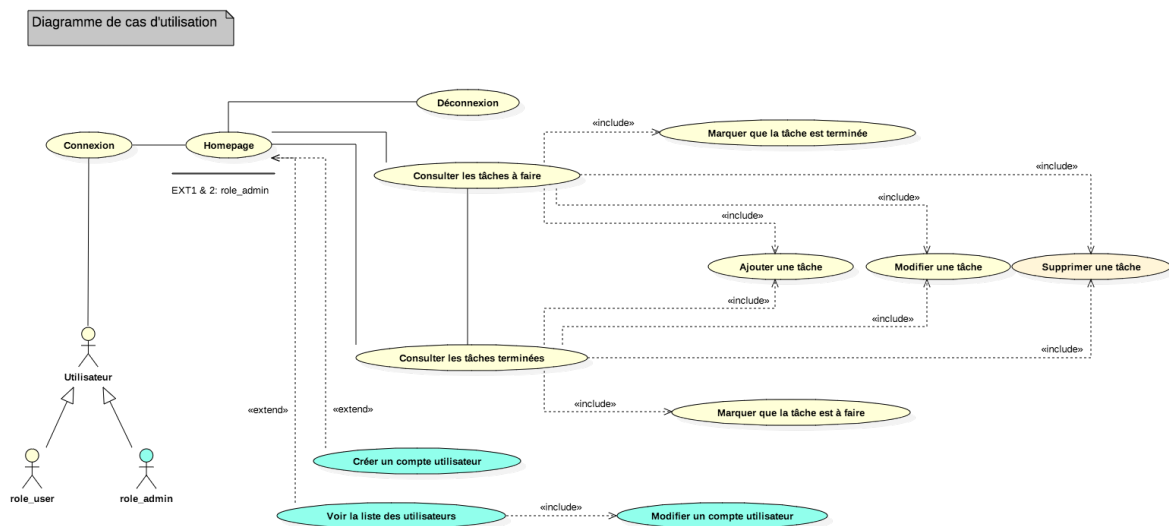
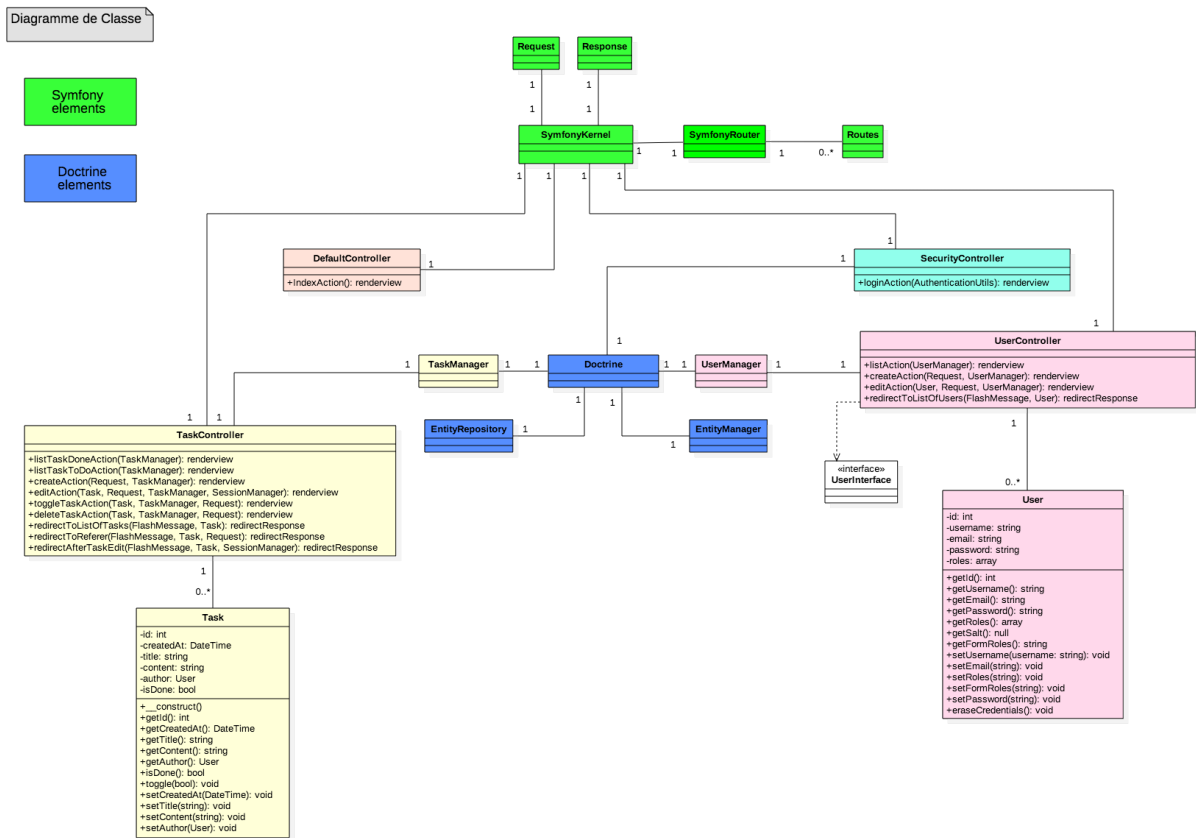


Diagramme de classe v 1.0



Principaux diagrammes de séquences

Diagrammes de séquence – homepage v 1.0

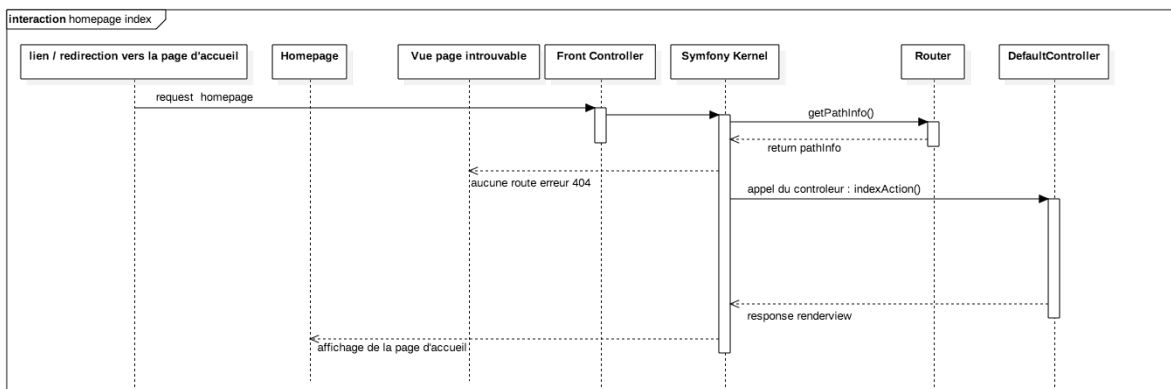


Diagramme de séquence – liste des tâches à faire v 1.0

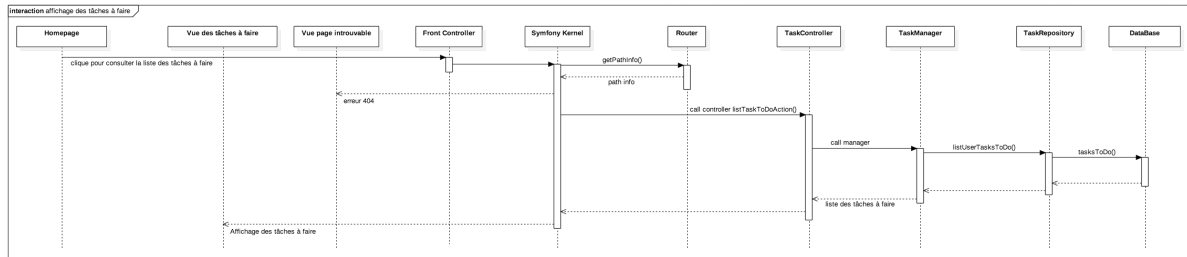


Diagramme de séquence – liste des tâches terminées v 1.0

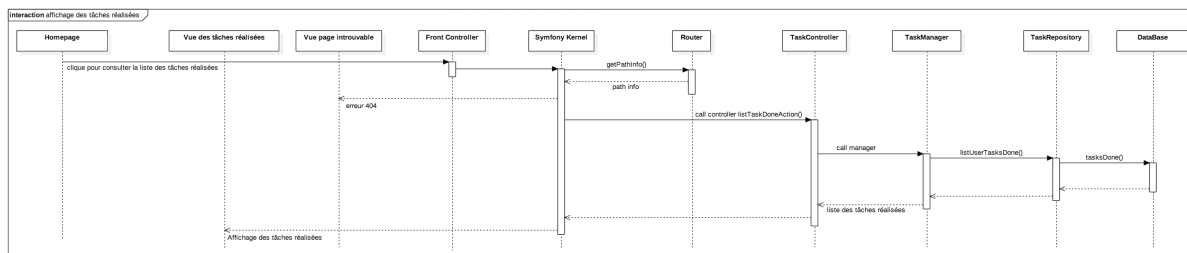


Diagramme de séquence – ajouter une tâche v 1.0

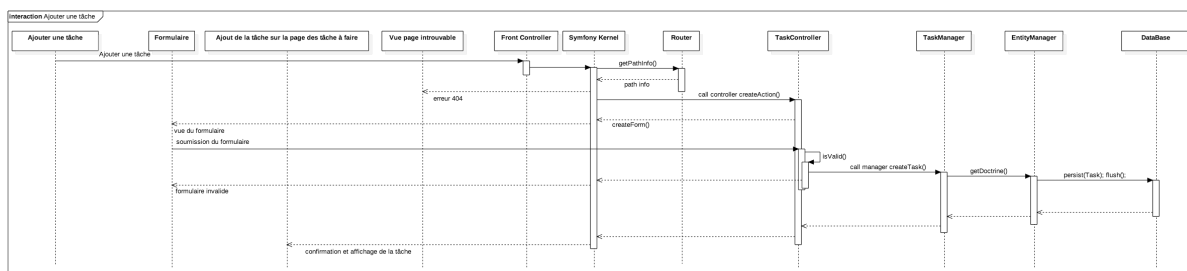


Diagramme de séquence – supprimer une tâche v 1.0

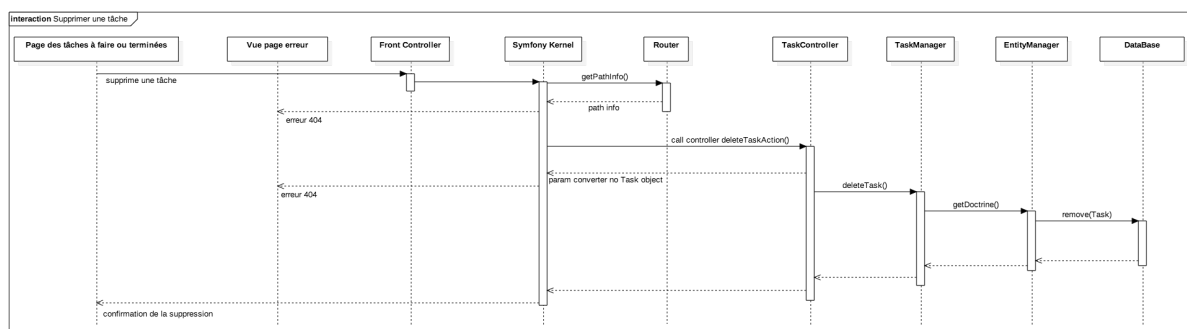
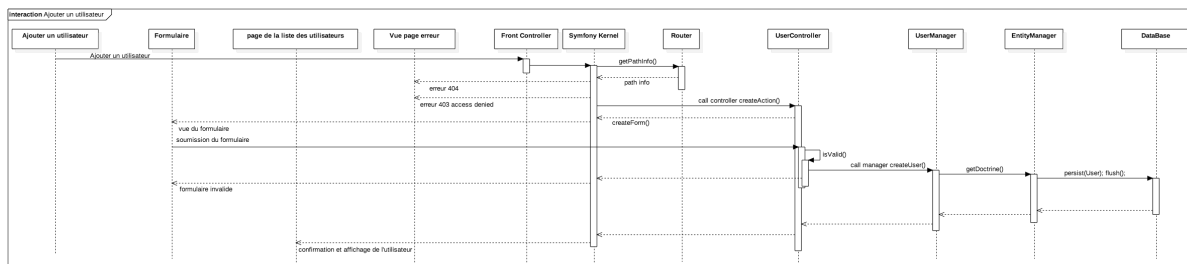
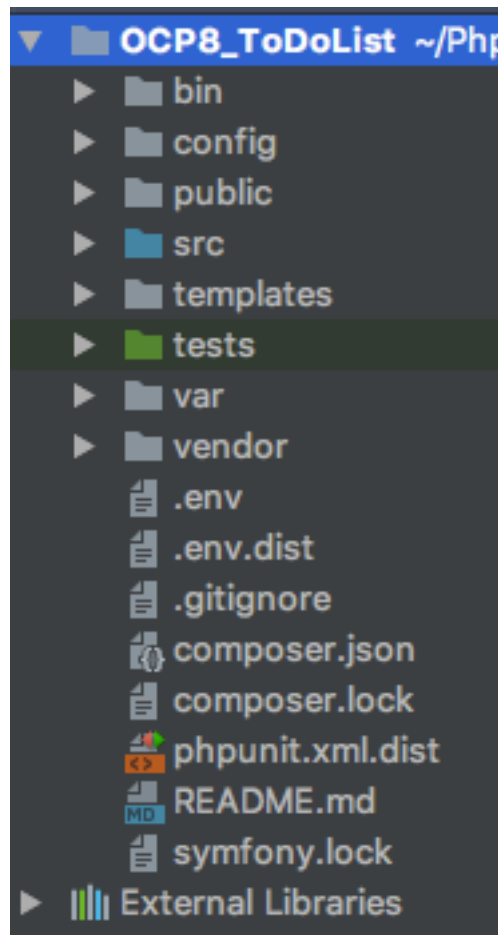


Diagramme de séquence – ajouter un utilisateur v 1.0



5.3 ARBORESCENCE DU PROJET



/bin contient les fichiers qui gèrent les exécutables en ligne de commande.

/config contient tous les fichiers de configuration du projet.

/public contient les fichiers accessibles par le client : CSS, JS, photos etc.. et le fichier index.php qui permet de lancer l'application.

/src contient le code source de l'application.

/templates contient les différents templates, les vues de l'application.

/tests contient nos fichiers de tests.

/var contient un dossier avec les fichiers de log et un dossier pour ceux du cache.

/vendor contient toutes les dépendances, bundles utilisés dans le projet.



- Le fichier .env permet de définir les variables d'environnement, c'est ici par exemple que l'on renseigne l'URL de la base de données et que l'on définit de manière globale l'environnement : développement ou production.

- Le fichier composer.json liste les dépendances, bundles utilisés dans le projet.

5.4 LE FICHIER SECURITY.YAML



La configuration de la sécurité de l'application s'effectue dans le fichier :

config/packages/security.yaml

Ce fichier permet de configurer tous les éléments touchant à la sécurité de l'application et aux mécanismes d'authentification et d'autorisation des utilisateurs.

La photo ci-dessous présente la configuration actuelle du fichier security.yaml de l'application :

Fichier security.yaml v 1.0

security:		
encoders:		encodeurs
App\Entity\User: bcrypt		
providers:		fournisseurs d'utilisateurs
my_provider:		
entity:		
class: App\Entity\User		
property: username		
firewalls:		
dev:		
pattern: ^/(_(profiler wdt) css images js)/		
security: false		
main:		
anonymous: ~		
pattern: ^/		type d'authentification des utilisateurs
form_login:		
login_path: login		
check_path: login		
always_use_default_target_path: true		
default_target_path: /		
logout:		
path: /logout		
target: login		
provider: my_provider		
access_control:		
- { path: ^/login, roles: IS_AUTHENTICATED_ANONYMOUSLY }		
- { path: ^/users, roles: ROLE_ADMIN }		limitation d'accès selon le rôle de l'utilisateur
- { path: ^/, roles: ROLE_USER }		
role_hierarchy:		hiérarchie des rôles utilisateurs
ROLE_ADMIN: ROLE_USER		

Explication détaillée des paramètres de configuration v 1.0

encoders:

Permet de configurer le choix des algorithmes de cryptage des données sensibles, comme les mots de passes, selon la classe concernée.

Dans la configuration actuelle nous utilisons l'algorithme « `bcrypt` » pour crypter les mots de passes de nos utilisateurs.

providers:

Permet de configurer le(s) fournisseur(s) d'utilisateurs.

Dans la configuration actuelle nous avons déclaré un provider : `my_provider`. Celui-ci utilise la classe `App\Entity\User` et sa propriété `username` comme fournisseur d'utilisateur. Cette configuration est donc implicitement liée à la persistance de nos utilisateurs en base de données.

firewalls:

Le but principal des pare-feu est de configurer la manière dont les utilisateurs vont s'authentifier. C'est d'une certaine manière le cœur de la configuration de notre sécurité.

Dans la configuration actuelle il y a deux firewalls :

dev: ce pare-feu n'est pas très important, il s'assure juste que notre système de sécurité ne bloque pas les outils de développement Symfony qui utilisent les URLs `/_profiler` ou `/_wdt`.

main: ce pare-feu est en revanche essentiel. C'est ici que nous allons déterminer de quelle manière les utilisateurs vont s'authentifier.

Détails du firewall **main:**

anonymous: `~`

Ici nous spécifions que les utilisateurs anonymes, ceux qui ne se sont pas encore authentifiés, ont accès à notre pare-feu général.

patern: `^/`

Ici nous spécifions que toutes les URLs de notre application sont soumises au pare-feu principal. (Exceptées celles définies précédemment dans le pare-feu dev.)

form_login:

Nous spécifions que nos utilisateurs s'identifieront avec un formulaire de connexion.

login_path: `login`

Ici nous spécifions le nom de la route ou le chemin vers lequel l'utilisateur sera redirigé s'il tente d'accéder à une ressource protégée nécessitant une authentification complète préalable.

Un utilisateur non complètement authentifiés (tant qu'un utilisateur n'est pas complètement authentifié il est considéré comme anonyme) sera systématiquement redirigé vers cette route quelque soit la ressource qu'il cherche à atteindre.

Dans la configuration actuelle, la route est « login ».

Check_path: `login`

Ici nous spécifions la route ou le chemin utilisé pour la soumission du formulaire de connexion.

Dans la configuration actuelle, la route est « login ».

Always_use_default_target_path: `true`

Permet de déterminer si l'utilisateur est systématiquement redirigé vers le « default_target_path » après son authentification. (voir paramètre suivant.)

Dans la configuration actuelle la valeur est à « true ».

default_target_path: `/`

Définit la page de redirection de l'utilisateur après son authentification.

Dans la configuration actuelle l'utilisateur est redirigé sur la page d'accueil.

logout:

Ce paramètre de configuration permet la prise en charge automatique du processus de déconnexion par le firewall.

path: `/logout`

Nous définissons le chemin de déconnexion.

Il faudra dès lors créer la route correspondante dans le fichier config/routes.yaml, mais il n'est pas nécessaire de créer un contrôleur.

target: `/`

Nous définissons la page de redirection après la déconnexion.

provider: `my_provider`

Nous spécifions que le fournisseur d'utilisateurs est « my_provider » que nous avons configuré précédemment dans la section providers.

access_control:

Permet de sécuriser des sections entières de l'application, en restreignant l'accès de ces sections à certaines catégories d'utilisateurs en fonction de leurs rôles.

Dans la configuration actuelle seules les pages définies par un chemin débutant par « **/login** » sont accessibles à un utilisateur anonyme (non formellement authentifié).

Les sections définies par un chemin débutant par « **/users** » sont limitées aux utilisateurs ayant le rôle ADMIN.

L'ensemble de l'application hormis la section « **/login** » est limitée aux utilisateurs ayant le rôle USER. (Un ADMIN a également les accès d'un USER, voir paramètre suivant).

role_hierarchy:

ROLE_ADMIN: ROLE_USER

Nous attribuons ici toutes les autorisation du rôle USER au rôle ADMIN.

5.5 MECANISMES D'AUTHENTIFICATION



Synthèse du processus d'authentification définit dans la configuration

Les paramètres définis et configurés dans le fichier de configuration security.yaml que nous venons de détailler permettent de résumer le processus d'authentification d'un utilisateur.

1 – Lorsqu'un utilisateur arrive sur l'application il est automatiquement redirigé vers le formulaire d'authentification.

2 – Après soumission du formulaire d'authentification qui contient un champ « nom d'utilisateur » et un champ « mot de passe », l'application va rechercher en base de données l'utilisateur ayant pour nom d'utilisateur celui fourni lors de la soumission du formulaire.

Si aucun utilisateur n'est enregistré en base sous ce nom, l'utilisateur est redirigé vers le formulaire d'authentification, l'application l'informe que les identifiants de connexion soumis sont invalides.

Si l'application a bien récupéré un utilisateur lié à ce nom, elle vérifie que le mot de passe enregistré en base de données correspond bien à celui fourni lors de la soumission du formulaire.

Si le mot de passe fourni lors de la soumission du formulaire ne correspond pas à celui de l'utilisateur récupéré en base de données, l'utilisateur est redirigé vers le formulaire d'authentification, l'application l'informe que les identifiants de connexion soumis sont invalides.

Si le mot de passe soumis dans le formulaire correspond bien à celui de l'utilisateur récupéré en base, l'utilisateur est alors pleinement authentifié et l'application le redirige sur la page d'accueil. Il a alors accès à toutes les ressources de l'application définit par son rôle.

5.6 MECANISMES D'ACCREDITATION

Comme vu dans la section précédente, il existe pour l'instant deux rôles attribuables aux utilisateurs :

- ROLE_USER
- ROLE_ADMIN

De plus un utilisateur ayant le ROLE_ADMIN a aussi tous les droits alloués au ROLE_USER.



La possibilité d'ajouter un utilisateur n'est allouée qu'aux utilisateurs ayant le ROLE_ADMIN.

Le choix du rôle pour le nouvel utilisateur leur revient donc, lors de la création d'un compte utilisateur.

Il leur est également possible de modifier le rôle d'un utilisateur lors de l'édition d'un compte utilisateur.

6. NORMES ET STANDARDS DE REALISATION

6.1 DEVELOPPEMENT DE L'APPLICATION

Le repository GitHub contenant le code complet de l'application est disponible via le lien fourni à la section 5.1 du document.

Il faudra en premier lieu faire un **fork**, dupliquer, le répertoire du projet. De cette manière le repository original ne sera pas directement modifié.

Chaque développeur disposant d'une copie de l'application, le développement de nouvelles fonctionnalités sera implémenté tout d'abord sur une nouvelle branche du repository copié, afin de préserver l'intégrité de sa branche master.

Quand la fonctionnalité satisfera le développeur, il effectuera une **pull-request** de sa branche de développement vers la branche master du répertoire original de l'application.

Si la nouvelle fonctionnalité satisfait toutes les attentes, nous procéderons à un **merge**, une fusion, de la fonctionnalité dans la branche master du repository original.

6.2 DEVELOPPEMENT ORIENTE PAR LES TESTS

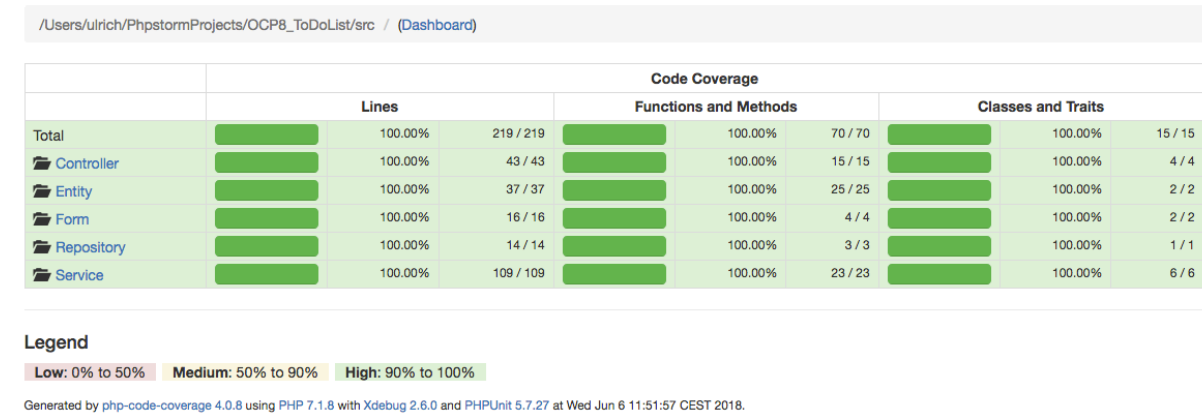


L'écriture de tests automatisés à l'aide de PHPUnit doit être systématique lors de chaque ajout d'une nouvelle fonctionnalité.

Il est conseillé de tout d'abord écrire le test, constater qu'il échoue, de développer la fonctionnalité puis de repasser le test et de s'assurer qu'il est valide.

Cette manière de procéder oblige à réfléchir en amont, lors de l'écriture du test, sur la logique fonctionnelle de la future implémentation.

Etat actuel de la couverture du code de l'application par les tests automatisés v1.0



6.3 NORMES DE CODAGE

Le code de l'application doit respecter le standard de codage PSR-2 et les normes de standard de Symfony.

[Normes PSR-2](#) | [normes de codage de Symfony](#).

Selon les normes de codage de Symfony, les paramètres des méthodes de classe doivent être écrits sur la même ligne que le nom de la méthode. Dans ce cas un dépassement du nombre de caractères recommandés devra être toléré.

7. QUALITE DU CODE ET PERFORMANCES DE L'APPLICATION

7.1 AUDIT DE LA QUALITE DU CODE

La qualité du code de l'application a été vérifiée par [SensioLabsInsight](#).

Cet outil permet d'analyser la qualité du code sur demande ou de manière automatisée à chaque **push** de code dans une branche ou lors d'un **merge**.

De plus, il permet d'être informé si une faille a été détectée dans une version du framework ce qui permet de rapidement mettre à jour la version avec le patch correctif.

Dans l'état actuel la qualité du code de l'application est jugée optimale par cet outil.

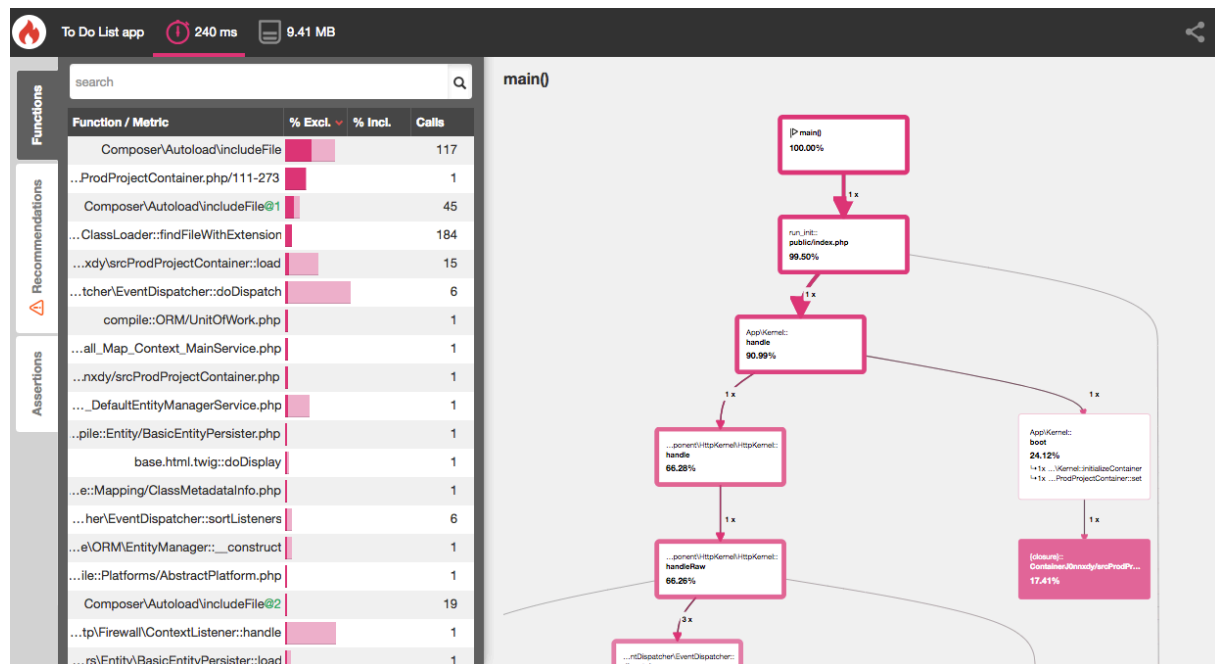


7.2 METRIQUE DES PERFORMANCES DE L'APPLICATION

Les performances de l'application peuvent être mesurées avec l'outil [BlackFire](#) de SensioLabs.

Cet outil permet le profilage des pages d'une application en mesurant sa vitesse d'exécution, la mémoire allouée, et fournit un graphique des résultats.

Exemple avec le profilage de la page d'accueil



Le graphique de droite représente le code exécuté sous forme de diagramme dans lequel chaque nœud représente une fonction ou une méthode et leurs relations permettent de représenter le flux de code.

BlackFire affiche les nœuds du graphique d'appel en utilisant un code « couleur » où le rouge indique les nœuds d'un chemin chaud. Plus cette couleur rouge est intense plus le nœud a d'importance dans le chemin chaud.

Les chemins chauds sont les premières sources potentielles de pertes de performances.

Les nœuds avec un fond coloré sont ceux qui sont les plus sollicités. Ils doivent aussi être analysés avec attention.

La partie de gauche affiche la liste complète des fonctions ou méthodes exécutées ainsi que leurs principales informations de profilage (temps exclusif / inclusif, appels de fonctions, etc.).

Après le profilage d'une page, BlackFire fournit des recommandations qui permettent d'améliorer sensiblement les performances de l'application.

Etat actuel des performances de l'application v 1.0 :

La vitesse moyenne d'affichage d'une page se situe entre 200 et 330 ms.

La taille de la mémoire allouée varie entre 9 et 12 MB.



Sur cette base, l'affichage d'une page résultant du développement d'une nouvelle fonctionnalité ne devrait pas excéder **400 ms**.

Pour conclure, rappelons juste que tout le travail effectué pour optimiser les performances d'une application est motivé par la nécessité d'offrir la meilleure expérience possible à l'utilisateur.

8. ANNEXE : AXES D'AMÉLIORATIONS

8.1 INTRODUCTION

Cette partie a pour but de proposer différents axes d'améliorations de l'application.

Indépendamment de l'évolution que prendra l'application, les axes d'améliorations proposés dans la section 8.2 – « A court terme » ayant un rapport avec la protection des données, nous semblent importants d'être rapidement implémentés.

8.2 A COURT TERME

Afin d'être en adéquation avec le nouveau règlement général de protection des données fixées par l'Union Européenne (RGPD), plusieurs modifications nous semblent importantes :

- Après avoir reçu ses identifiants de connexion, un utilisateur devrait être redirigé sur une page lui demandant de redéfinir ses identifiants de connexion. Tant que cela n'est pas fait, son compte devrait être désactivé.
- Un utilisateur dont le compte est actif devrait pouvoir redéfinir son mot de passe en cas d'oubli.
- Un utilisateur ayant le rôle ADMIN ne devrait pas pouvoir redéfinir les identifiants de connexion d'un autre utilisateur.

Afin d'améliorer la fluidité du développement de l'application, **un outil d'intégration continue** devrait être mis en place.

Cet outil permettra de s'assurer que tous les points de contrôles (tests, performances, qualité de code, montée de charge) sont valides lors des différents stades d'évolution de l'application, que ce soit au moment d'une **pull-request** en phase de développement, ou lors du passage en (pré)production.

Les outils d'intégration continue sont nombreux :

- [Travis](#)
- [Jenkins](#)
- [AppVeyor](#)
- etc....

8.3 A MOYEN TERME

Mise en place d'une couche de services entre l'application et le serveur de données.

Actuellement, l'application traite directement avec la couche de données.

Un micro service dédié, chargé d'être seul en lien avec la bdd pour effectuer les opérations CRUD, offrirait une interopérabilité avec de nouveaux clients, typiquement une version mobile Android / IOS de l'application.

8.4 A LONG TERME

Il faut déjà considéré que le trafic généré par l'application pourrait à moyen, long terme devenir conséquent, voire très important.

Il est donc intéressant d'envisager des solutions permettant de gérer un nombre important de connexions simultanées.

Démultiplication des serveurs web et mise en place d'un répartiteur de charges

La mise en place d'un **load balancer** (répartiteur de charges) permet de répartir sur plusieurs servers les multiples requêtes entrantes, en fonction d'un algorithme de distribution.

Les avantages d'une répartition de charge équilibrée sont nombreux :

- Temps d'accès plus court, même si de nombreuses demandes arrivent simultanément.
- Meilleure sécurité contre les crash, puisque le trafic d'un serveur plus lent est transmis automatiquement à un autre serveur.
- Si un server est indisponible, l'application reste fonctionnelle.
- La maintenance des serveurs est facilitée.

Fin du document