ToDo & Co

Audit de Code

Table des matières

Synthèse de l'audit **2**
Analyse critique de l'existant 2

Architecture technique **3**
Php 3
MySQL 3
Symfony 3

Bonnes pratiques Symfony **3**
Structure de l'application 3
Configuration 4
Logique Métier 4
Standard de développement PSR 4
Contrôleurs 4
Templates 5
Formulaires 5
Sécurité 5
Gestion des assets 5

Tests **6**

Performances **6**

Evolution **7**
Amélioration du système de tâches 7
Création de nouveaux rôles 7
Mise en place d'autorisations d'accès 7
Implémentation de tests automatisés 8

Synthèse de l'audit

Analyse critique de l'existant

Points forts	Points faibles
Les formulaires respectent entièrement les bonnes pratiques ; Le firewall est correctement configuré ;	Projet ancien ne recevant plus de correctifs; Aucune utilisation des fixtures ou des repositories;

La définition des entités et l'utilisation de doctrine respectent les bonnes pratiques.	Pas de contrôle d'accès pour les pages réservées aux administrateurs; Symfony flex n'est pas installé; Pas ou peu d'injection de dépendance.

Architecture technique

Php

Besoin applicatif minimum: 5.5.9 ou supérieur.
Version actuelle: 7.4.9 (serveur).
Commentaires: Un passage à Symfony 4 nécessite la version 7.1.9 au minimum.
Pistes d'amélioration: Configurer composer pour lui indiquer la version 7.1.9 comme version minimum requise.

MySQL

Version Actuelle: MySQL 5.7.31

Symfony

Version Actuelle: Symfony 3.1.6
Commentaires: La version 3.1.6 n'est plus maintenue et la version 3.4 bénéficie d'une maintenance jusqu'au 3ème trimestre 2021 seulement.
Pistes d'amélioration: Un passage à la version 4.4 de Symfony permettrait la maintenance de l'application jusqu'au 3ème trimestre 2023. Utiliser l'outil "requirement-checker" peut aider à faciliter la mise à niveau vers la version 4.

Bonnes pratiques Symfony

Structure de l'application

Recommandé:

- Utiliser la structure de dossier par défaut.

Etat actuel:

- Non Standard.

Pistes d'amélioration:

- Retirer dossier AppBundle
- Renommer le dossier resources en dossier templates ;
- Réorganiser le dossier web en un dossier assets et utiliser Webpack encore (voir la section Templates)

Configuration

Recommandé:

- Utiliser des variables d'environnement pour configurer l'infrastructure ;
- Utiliser des constantes pour définir des options qui changent rarement ;
- Préfixer les variables présentes dans les fichiers de configuration.

Etat actuel:

- Pas de variables d'environnement présentes.

Logique Métier

Recommandé:

- Éviter d'utiliser les bundles pour organiser l'application.
- Utiliser l'autowiring pour automatiser la configuration des services de l'application.
- Les services doivent, si possible, avoir un accès privé.
- La configuration des services personnalisés doit être définie dans un fichier yaml approprié.
- Utiliser les annotations pour définir le mapping des entités Doctrine.

Etat actuel:

- L'autowiring n'est pas activé
- Le code de l'application est organisé comme un bundle

Standard de développement PSR

Etat actuel:

- Commentaires et notations liés à la documentation du code non présent dans de nombreux fichiers :
    - AppBundle.php, DefaultController.php, SecurityController.php, TaskController.php, UserController, User.php, Task.php, TaskType, UserType.
- Mauvaises indentations :
    - User.php.
- Parenthèses mal positionnées :
    - SecurityController.php, TaskController, UserType.php.
- Fichiers utilisant les retours à la ligne au format CRLF plutôt que LF

Contrôleurs

Recommandé:

- Les contrôleurs doivent étendre la classe AbstractController ;
- Le routing, le cache et la sécurité doivent utiliser les annotations pour leur configuration ;
- Ne pas utiliser les annotations pour configurer le template du contrôleur ;
- Utiliser l'injection de dépendances pour récupérer un service ;

- Utiliser la classe ParamConverter de doctrine, si besoin.

Etat actuel:

- L'injection de dépendances n'est pas utilisée dans les contrôleurs.

Templates

Recommandé:

- Utiliser le snake_case pour les noms et variables de templates ;
- Préfixer le nom des fragments de templates avec un underscore.

Etat actuel:

- Certaines variables ne sont pas en snake_case (dans list.html.twig ou login.html.twig, par exemple).

Formulaires

Recommandé:

- Définir chaque formulaire sous la forme d'une classe PHP ;
- Les boutons de formulaires doivent figurer dans les templates plutôt que dans les classes PHP ;
- Définir les contraintes de validation sur l'entité plutôt que dans la classe de formulaire ;
- Utiliser une seule action pour l'affichage et le traitement du formulaire.

Etat actuel:

- Tout est en ordre.

Sécurité

Recommandé:

- Définir un firewall unique ;
- Utiliser l'encrypteur automatique de mots de passe ;
- Utiliser des voters pour implémenter des restrictions d'accès plus fines.

Etat actuel:

- L'encrypteur bCrypt est utilisé au lieu de l'encrypteur automatique.

Gestion des assets

Recommandé:

- Utiliser Webpack Encore pour la gestion des assets.

Etat actuel:

- Webpack Encore n'est ni installé, ni utilisé.
- Les assets se trouvent dans le dossier web plutôt que dans le dossier public
- Les assets sont directement référencés dans les templates par les balises script et link.

Tests

Existants:

- L'application dispose d'un test fonctionnel vérifiant le bon déroulement de la requête pour obtenir la page d'accueil (DefaultController->Index())

Besoins critiques:

- Les opérations de modification, suppression et ajout ont besoin d'être testées pour s'assurer qu'elles n'aient pas d'effets indésirable sur le bon fonctionnement de l'application après chaque évolution du code.

Performances

Avant désigne les tests de performance effectués avant modification du projet.
Après désigne les tests de performance effectués après ajout des nouvelles fonctionnalités et mise à niveau de symfony.

	Avant		Après	
Route	**Temps d'exé.**	**Conso. mémoire**	**Temps d'exé.**	**Conso. mémoire**
GET /	29.4ms	1.96MB	83.5ms	5.86MB
GET /tasks	8.28ms	1.66MB	104ms	6.14MB
GET /tasks/create	10.3ms	1.66MB	99.3ms	7.36MB
GET /tasks/{id}/edit	7.93ms	1.66MB	103ms	7.4MB
GET /tasks/{id}/delete	8.57ms	1.66MB	81ms	5.83MB
GET /login	12.6ms	1.76MB	84.4ms	5.89MB
GET /logout	9.1ms	1.6MB	54.4ms	4.75MB
GET /users/create	18.8ms	2.65MB	102ms	7.72MB
GET /users/{id}/edit	36.2ms	3.45MB	112ms	7.77MB
POST /tasks/create	52.1ms	3.42MB	161ms	8.13MB
POST /tasks/{id}/edit	46.3ms	3.13MB	127ms	8.1MB
POST /users/create	75.5ms	3.82MB	152ms	9.67MB
POST /users/{id}/edit	518ms	3.05MB	354ms	8.51MB

POST /login_check	486ms	2.24MB	304ms	5.69MB

Commentaires:

- L'autoload est responsable d'une grosse partie du temps d'exécution et peut être réduit significativement en mettant en place un autoload optimisé via la commande "composer dump-autoload -o".
- Les requêtes internes de doctrine sont responsables d'une grande partie de la consommation mémoire. Elles peuvent être réduites en mettant en place un cache doctrine ou en réalisant des requêtes plus précises par le biais des QueryBuilders.

Evolution

Amélioration du système de tâches

Rappel du besoin:

- L'utilisateur authentifié doit automatiquement être rattaché à une tâche qu'il a nouvellement créée.
- Lors de la modification de la tâche, l'auteur ne peut pas être modifié.
- Pour les tâches déjà créées, il faut qu'elles soient rattachées à un utilisateur "anonyme".

Commentaires:

Création de nouveaux rôles

Rappel du besoin:

- Lors de la création d'un utilisateur, il doit être possible de choisir un rôle pour celui-ci. Les rôles listés sont les suivants :
  - rôle utilisateur (*ROLE_USER*) ;
  - rôle administrateur (*ROLE_ADMIN*).
- Lors de la modification d'un utilisateur, il est également possible de changer le rôle d'un utilisateur.

Commentaires:

Mise en place d'autorisations d'accès

Rappel du besoin:

- Seuls les utilisateurs ayant le rôle administrateur (*ROLE_ADMIN*) doivent pouvoir accéder aux pages de gestion des utilisateurs.

- Les tâches ne peuvent être supprimées que par les utilisateurs ayant créé les tâches en question.
- Les tâches rattachées à l'utilisateur "anonyme" peuvent être supprimées uniquement par les utilisateurs ayant le rôle administrateur (*ROLE_ADMIN*).

Commentaires:

- Si besoin, vérifier les autorisations d'accès par le biais de voters pour désencombrer les contrôleurs.

Implémentation de tests automatisés

Rappel du besoin:

- L'application bénéficierait grandement d'une batterie complète de tests fonctionnels pour chaque action des principaux contrôleurs.

Commentaires:

- Aucun service ou fonction personnalisé n'étant présent, les tests unitaires n'ont pas lieu d'être