

Documentation technique

Simplex-Immo

A propos de cette documentation :

Cette documentation technique a pour but d'aider les développeurs travaillant sur l'application à la :

- développer,
- tester, et
- maintenir,

de manière collaborative et efficace.

Cette documentation doit être un outil facile et rapide à utiliser. Pour atteindre cet objectif, elle devra être exhaustive, tout en restant concise.

C'est-à-dire que :

- Tous les sujets devront être abordés (de la méthodologie de travail jusqu'aux fonctionnalités).
- On documentera de manière la plus courte et la plus explicite possible (on préférera les schémas et tableaux aux longs textes explicatifs).

Mise à jour

03/03/2022 : version 0.1

23/03/2022 : version 0.2

30/03/2022 : version 0.3

11/04/2022 : version 1.1

14/04/2022 : version 1.2

Table des matières

I)	Rappel sur le fonctionnement de l'application	3
1)	Description de l'application.....	3
2)	Utilisateurs de l'application et use cases	3
II)	Description de l'architecture	7
1)	Architecture de l'application	7
2)	Diagramme de classes et modèle physique de données	8
3)	Bundles utilisés.....	9
4)	API et tierces sollicités.....	10
III)	Méthode de travail.....	11
1)	Methodologie scrum	11
2)	GitFlow	13
3)	Campagne de tests	15
4)	Bonnes pratiques.....	16
IV)	Sécurité.....	19
1)	Protocole sécurisé https.....	19
2)	Les failles du web	19
3)	L'authentification dans Symfony.....	19
V)	Procédure d'installation et de déploiement	21
1)	Procédure d'installation	21
2)	Procédure de déploiement.....	21
VI)	Déroulement des sprints	22
1)	Sprint 1	22
VII)	Bug connus	24

I) Rappel sur le fonctionnement de l'application
















1) Description de l'application

Simplex-Immo est une application de gestion immobilière créée pour les propriétaires-bailleurs qui souhaitent gérer leur bien seul sans recourir aux services d'une agence immobilière.

En automatisant les tâches de gestion locative, Simplex-Immo permet aux bailleurs de reprendre la main sur tous les aspects de leurs locations, sans dépense de temps et d'énergie supplémentaires par rapport à une agence classique et surtout sans les coûts financiers inhérents à celle-ci.

2) Utilisateurs de l'application et use cases

Les utilisateurs de l'application

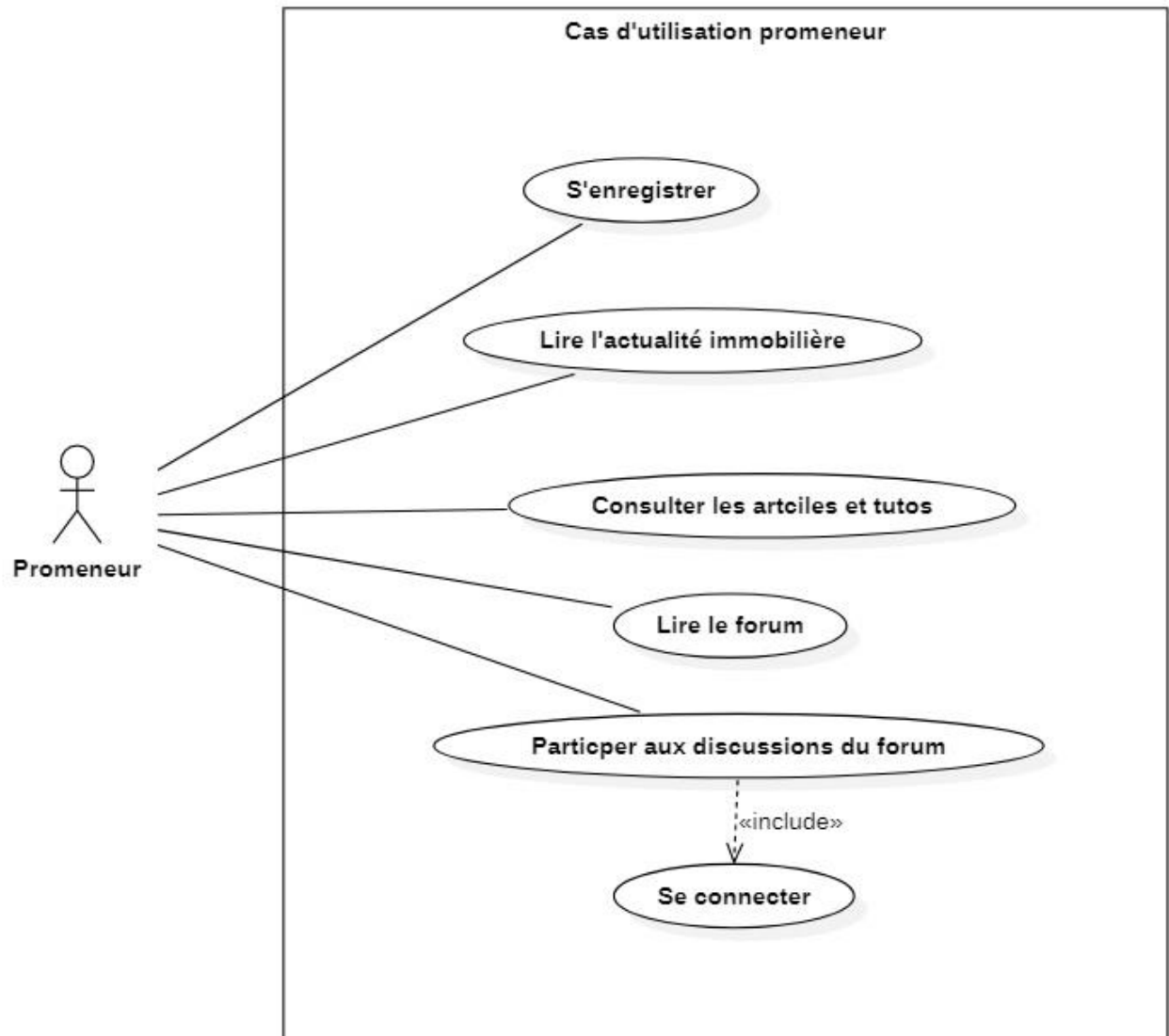
 ( ?) Promeneur	Le promeneur est un utilisateur qui peut consulter anonymement et gratuitement le contenu éditorial et le forum. Il doit s'enregistrer et se connecter pour participer aux discussions du forum.
  Bailleur	Le promeneur devient bailleur lorsqu'il a souscrit un abonnement payant pour profiter des fonctionnalités de gestion immobilière locative.
  Locataire	Le locataire est invité par le bailleur à valider son compte pour profiter de son espace locataire qui lui facilitera ses démarches et la communication avec le bailleur.
   Super admin	Le super admin est un employé de Simplex-Immo. Il possède tous les droits d'administration du site pour gérer les autres admins et les bailleurs.
   Juriste (< Super Admin)	Le juriste est un employé de Simplex-Immo dont le profil a été créé par le super admin. Il a accès aux fonctionnalités éditoriales pour créer du contenu et peut répondre aux questions juridiques du locataire et du bailleur.
   Editoraliste (< Juriste)	L' éditorialiste est un employé de Simplex-Immo dont le profil a été créé par le super admin. Il a accès aux fonctionnalités éditoriales pour créer du contenu.

Les diagrammes de cas d'utilisation

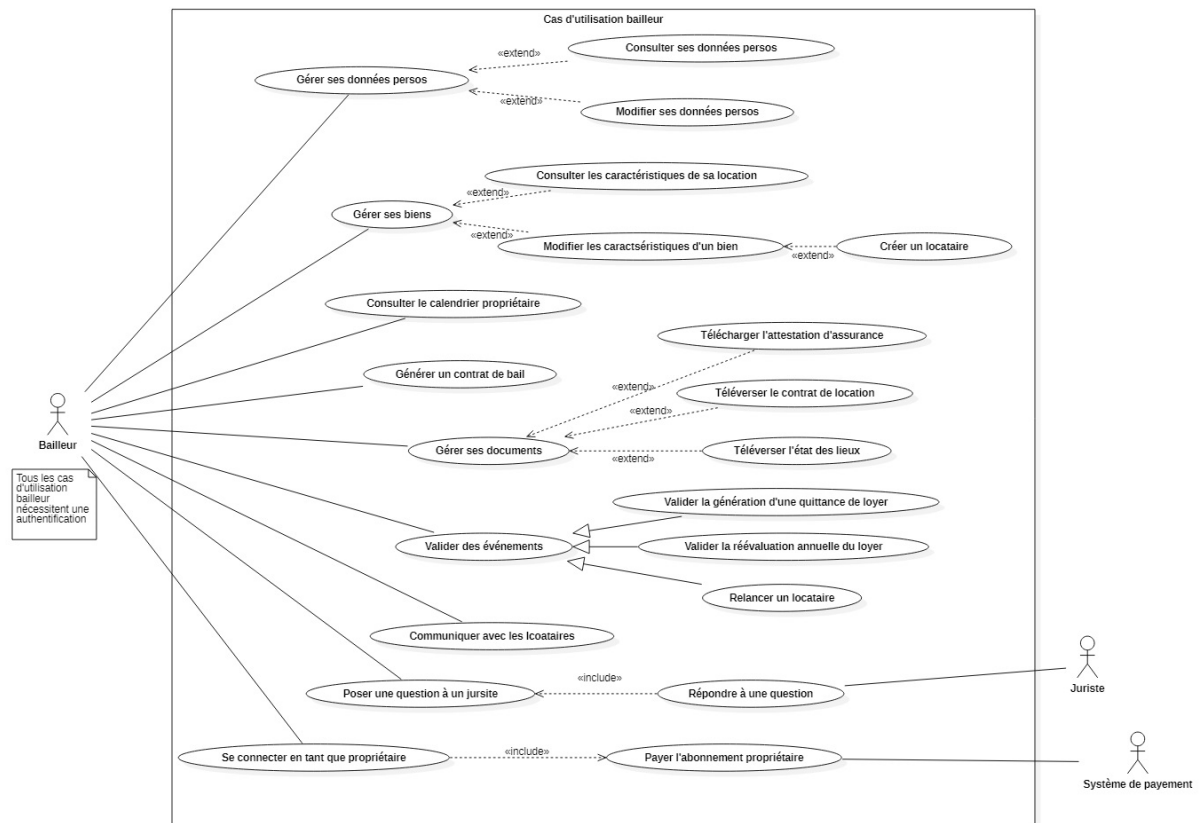
Travaillant en scrum, les besoins utilisateur sont au centre du projet. C'est donc assez naturellement que le découpage par lot de l'application s'est fait en fonction des besoins de chaque utilisateur : Promeneur, Bailleur, Locataire et Admin.

Les besoins de chaque utilisateur sont représentés dans les diagrammes de cas d'utilisation ci-dessous :

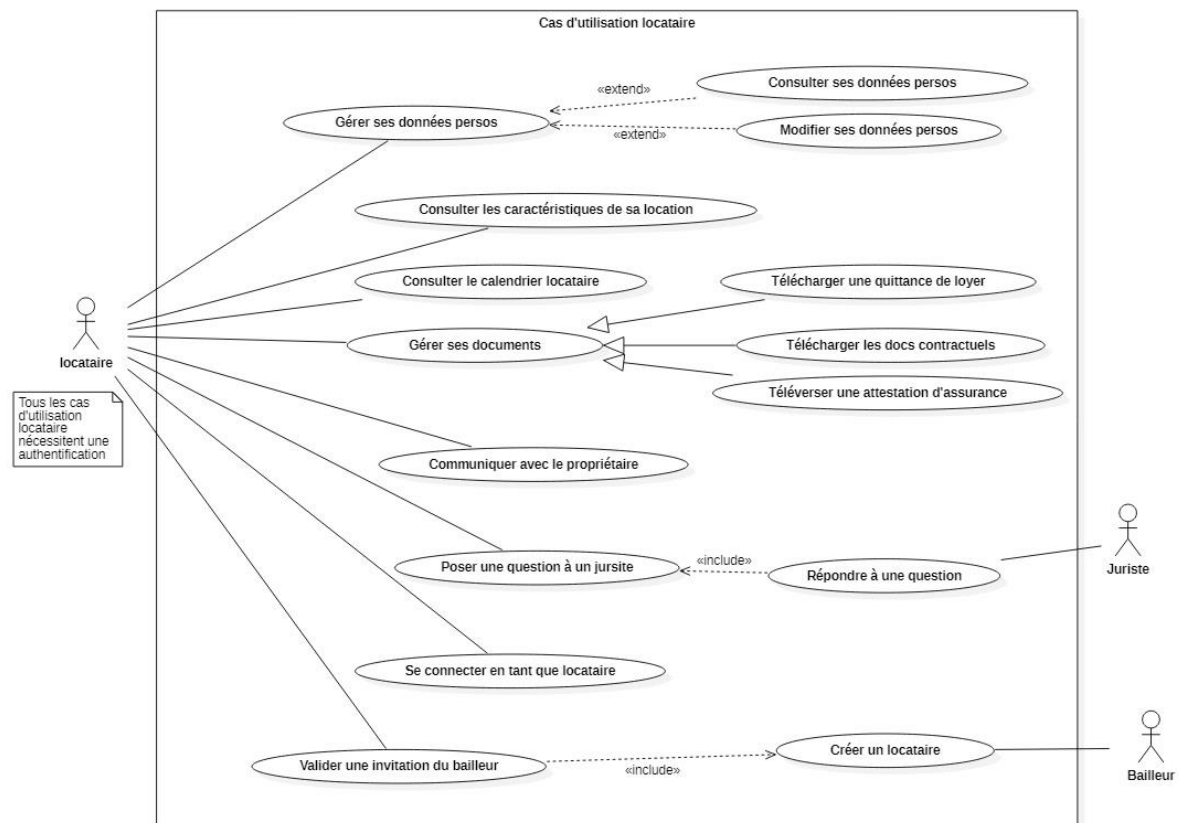
Lot n°1 : Vitrine de l'application



Lot n°2 : Cœur de l'application – espace bailleur

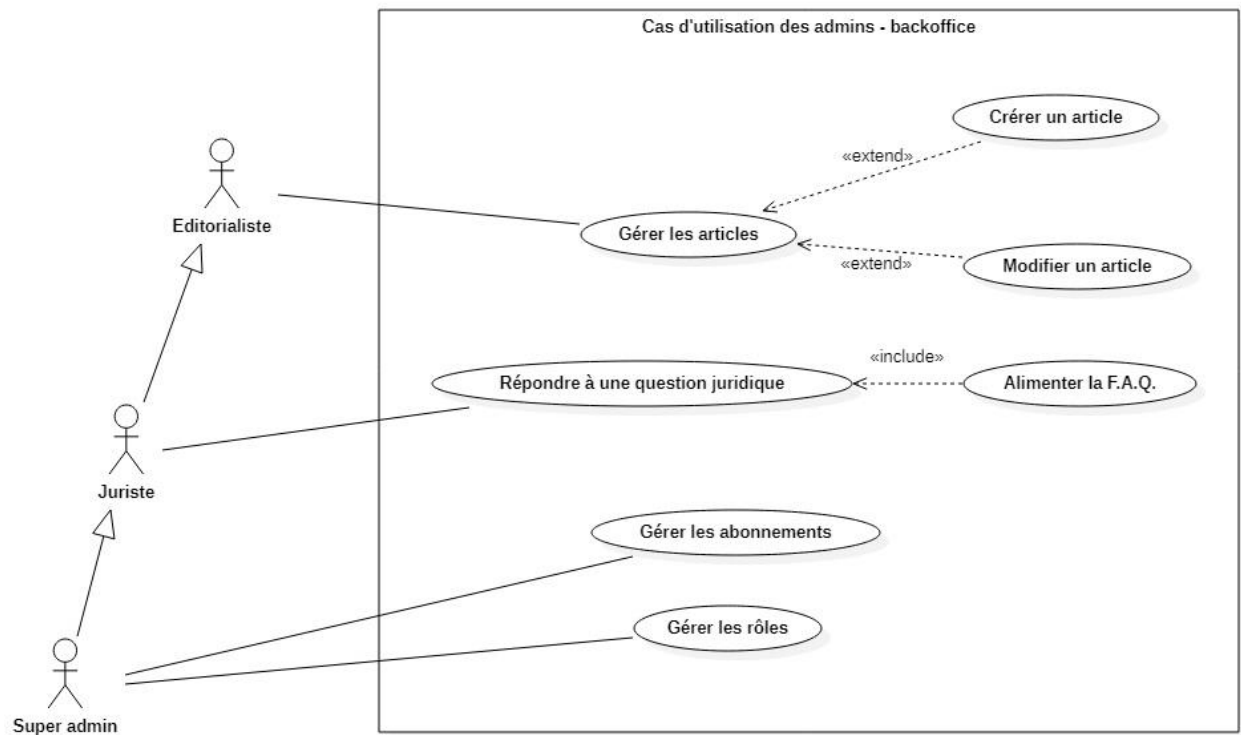


Lot n°3 : Cœur de l'application – espace locataire



Lot n°4 : Interface d'administration de l'application

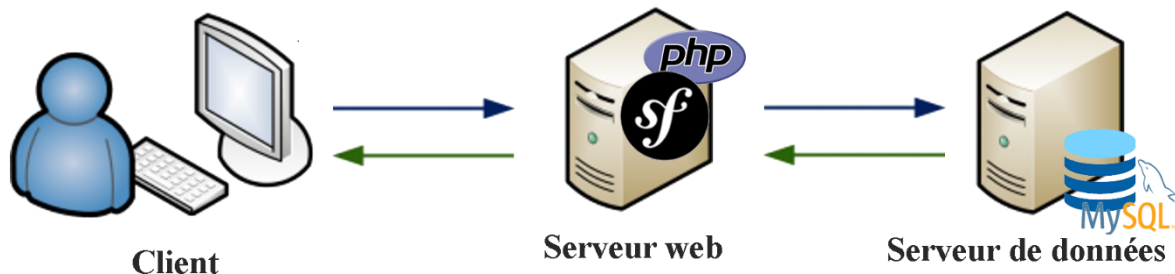
Le Super Admin, le Juriste et l'Editorialiste sont tous des admins.



II) Description de l'architecture

1) Architecture de l'application

Architecture globale



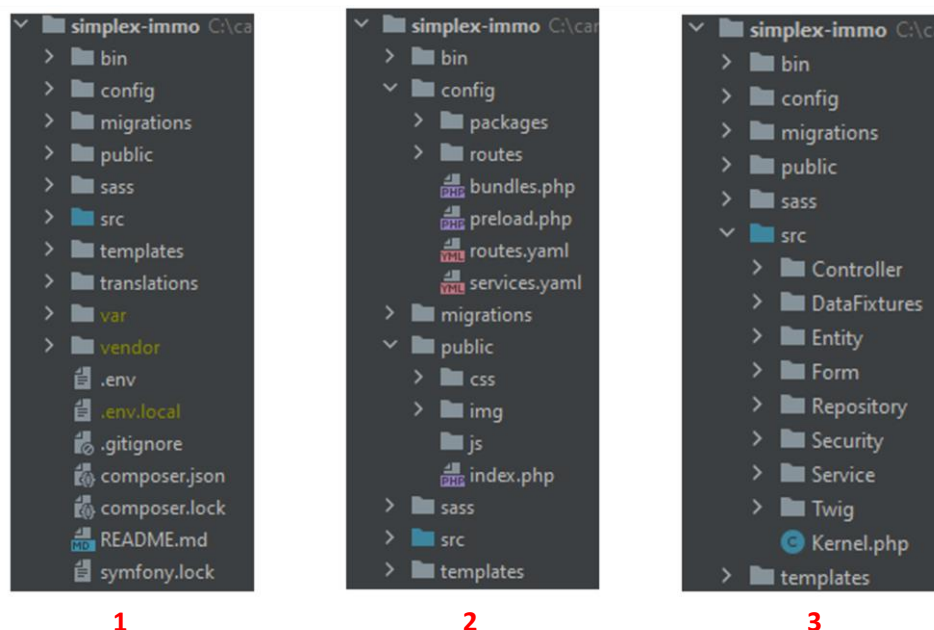
Versions :

- PHP : 8.0.11
- Symfony : 5.4
- Mysql : 10.4.21-MariaDB

Architecture de Symfony

Nous utiliserons pour notre projet l'architecture MVC (Modèle-Vue-Contrôleur) proposée par Symfony.

Architecture de fichier de Symfony

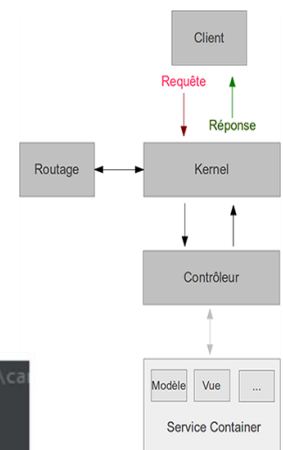


1 : Premier niveau de l'architecture de fichier.

2 : Deuxième niveau pour « config » et « public »

3 : Deuxième niveau pour « src »

⚠ : le dossier « template » qui contient les vues est placé à la racine du projet en dehors du dossier « src »



2) Diagramme de classes et modèle physique de données

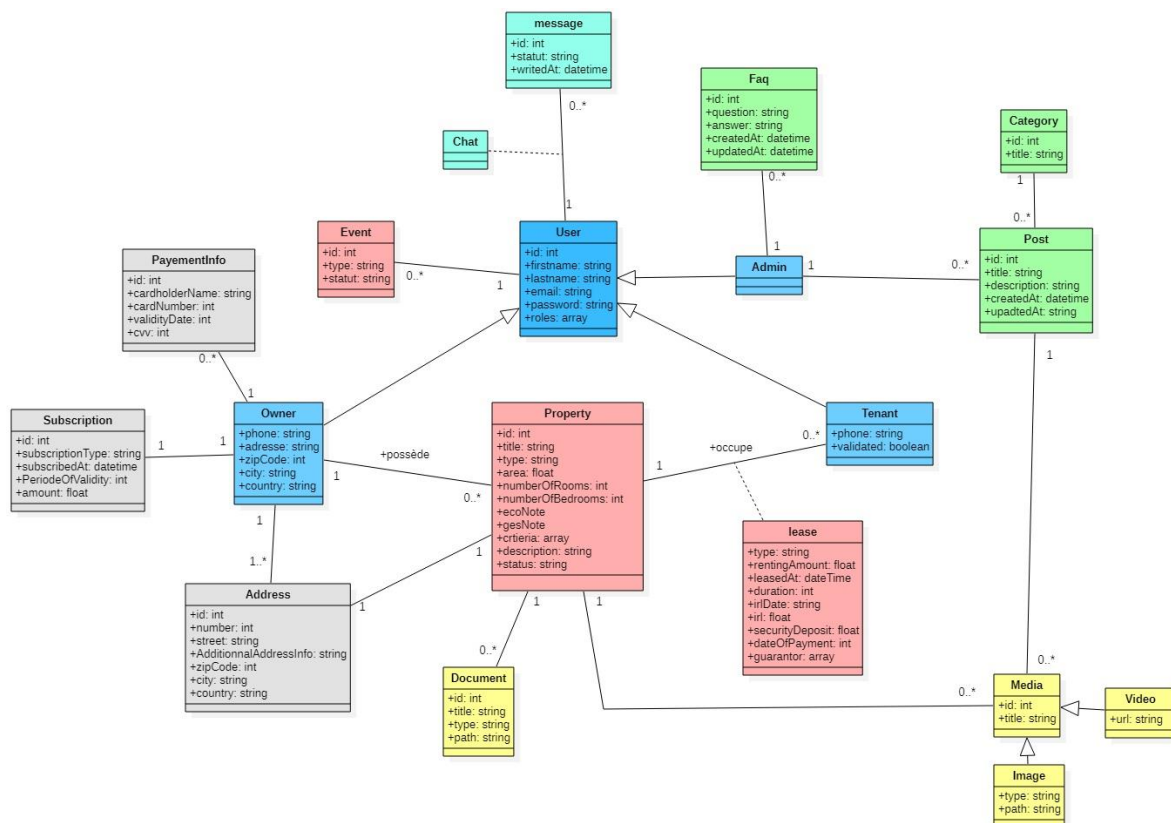
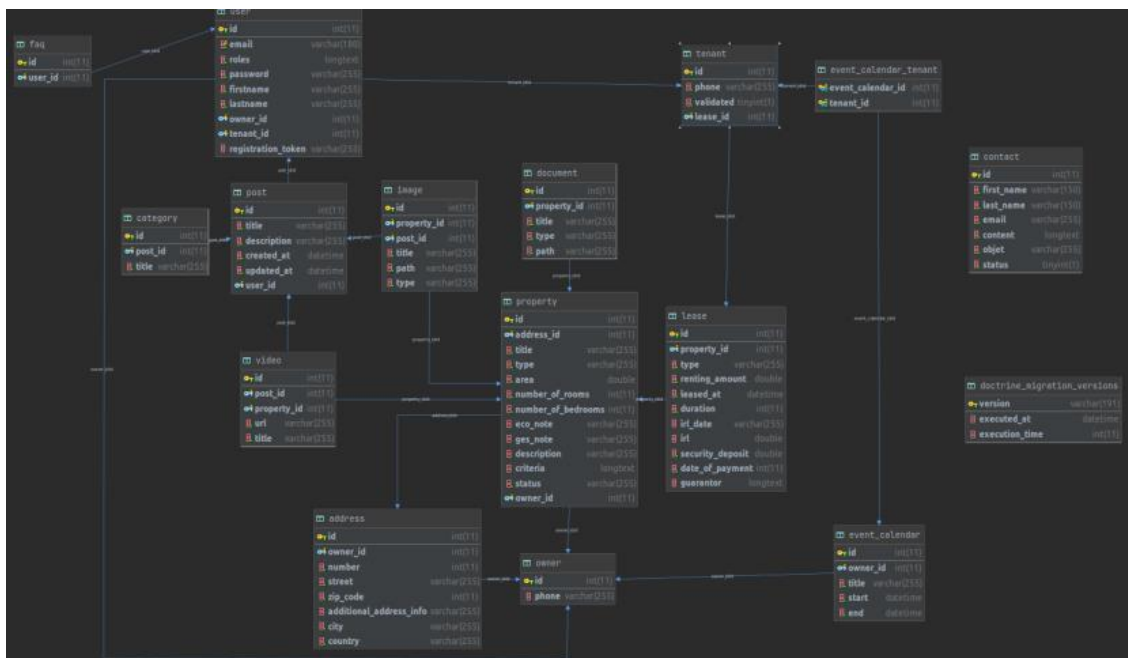


Diagramme de classes entités



Modèle physique de données

3) Bundles utilisés

Symfony a été installé dans sa version micro-service. Seuls les composants Symfony utiles seront installés au cours du développement. Voir le *require* du *composer.json* à la racine du projet.

```
"require": {
    "php": ">=8.0",
    "ext-ctype": "*",
    "ext-iconv": "*",
    "doctrine/doctrine-bundle": "^2.6",
    "doctrine/doctrine-fixtures-bundle": "^3.4",
    "doctrine/doctrine-migrations-bundle": "^3.2",
    "doctrine/orm": "^2.11",
    "fakerphp/faker": "^1.19",
    "sensio/framework-extra-bundle": "^6.2",
    "symfony/asset": "5.4.*",
    "symfony/cache": "5.4.*",
    "symfony/console": "5.4.*",
    "symfony/dotenv": "5.4.*",
    "symfony/flex": "^1.17|^2",
    "symfony/form": "5.4.*",
    "symfony/framework-bundle": "5.4.*",
    "symfony/monolog-bundle": "^3.0",
    "symfony/proxy-manager-bridge": "5.4.*",
    "symfony/runtime": "5.4.*",
    "symfony/security-bundle": "5.4.*",
    "symfony/translation": "5.4.*",
    "symfony/twig-bundle": "5.4.*",
    "symfony/validator": "5.4.*",
    "symfony/yaml": "5.4.*",
    "twig/extra-bundle": "^2.12|^3.0",
    "twig/twig": "^2.12|^3.0"
},
```

Twig (installé)

- Moteur de template
- Version : 2.12
- Lien : <https://twig.symfony.com/doc/2.x/>
-

Doctrine (installé)

- ORM
- Version : 2.6
- Lien : <https://www.doctrine-project.org/projects/doctrine-orm/en/2.11/index.html>

PHPUnit (installé)

- Framework de tests
- Version : 9.5
- Lien : <https://phpunit.readthedocs.io/fr/latest/>
- Utilisation : voir [paragraphe III-3](#)

CKeditor 5 (à installer)

- Editeur de texte (WYSIWYG) à utiliser dans la partie admin pour écrire des articles et des F.A.Q.
- Installation : *composer require ckeditor/ckeditor*
- Version : 4.18
- Lien : <https://packagist.org/packages/ckeditor/ckeditor>

DomPdf (à installer)

- Transformer du HTML en PDF
- Installation : *composer require dompdf/dompdf*
- Version : 1.2
- Lien : <https://github.com/dompdf/dompdf/>

4) API et tierces sollicités

Pour chaque solution finale retenue, indiquer en quelques lignes :

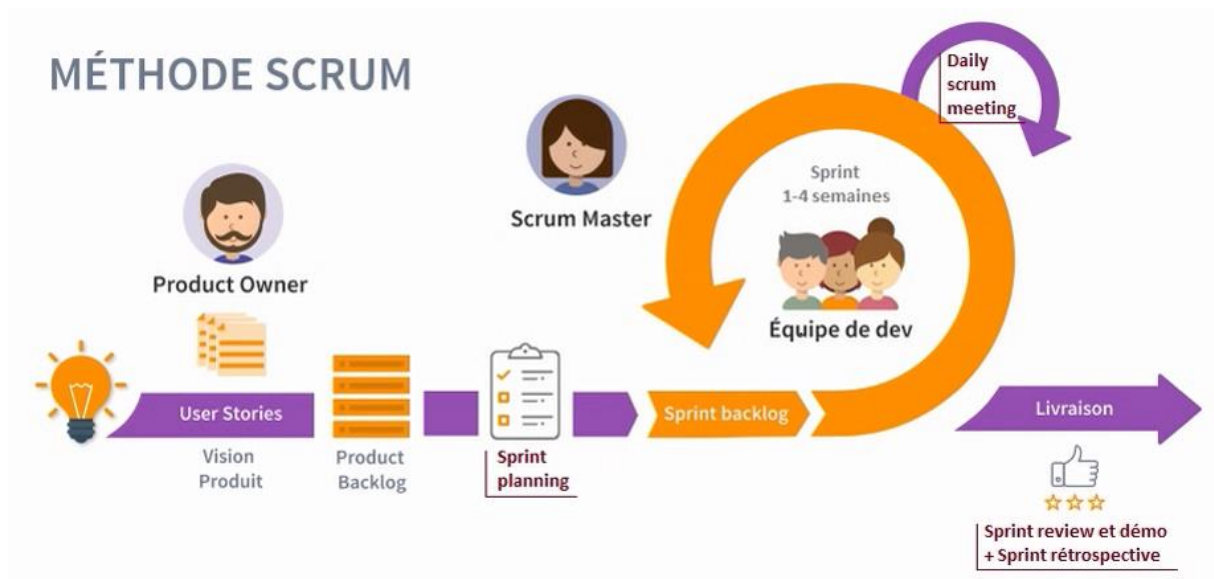
- Description de l'API
- Infos de connexion
- Version
- URL vers doc officiel

Piste de recherche

- Villes en zone tendue
Explorer la possibilité d'interroger les sites :
 - o <https://www.service-public.fr/simulateur/calcul/zones-tendues>
 - o <https://www.yanport.com> (49€/mois)
- Vérification des avis d'imposition des candidats à la location
Api ou intégration d'une fenêtre ou lien direct vers le site :
 - o <https://cfsmsp.impots.gouv.fr/secavis/>
- Indice de référence des loyers :
Explorer la possibilité d'interroger les sites :
 - o <https://www.anil.org/outils/indices-et-plafonds/tableau-de-lirl/>
 - o <https://www.yanport.com> (49€/mois)

III) Méthode de travail

1) Méthodologie scrum



Les principes de la méthode Scrum

La priorité numéro 1 est de satisfaire le client grâce à une livraison rapide et continue d'un prototype opérationnel pour l'utilisateur, afin qu'il puisse l'évaluer. C'est une approche dynamique et participative qui implique un engagement de la scrum team à réorienter le projet au fil de son avancement.

Les acteurs de la méthode Scrum

La **scrum team** est composée :

- D'un **scrum master** qui est le leader et le garant de la mise en place de la méthode scrum et de l'efficacité de la scrum team.
- D'un **Product Owner** qui détient la vision du produit à réaliser. Il représente les utilisateurs et le commanditaire.
- D'une **équipe de développeurs** qui transforme une sélection du Product Backlog en un incrément (= résultat utilisable par l'utilisateur).



Les événements de la méthode Scrum

Le projet est divisé en **sprint** (de 1 à 4 semaines chacun).

Chaque sprint est ponctué de 4 événements :

- Le **sprint planning** : c'est une réunion qui lance le sprint. Son but est de définir ses objectifs et d'organiser son déroulement (2h maximum par semaine de sprint).
- Le **Daily Scrum meeting** : c'est une réunion quotidienne (5-10 minutes) ayant pour but de maintenir le lien dans l'équipe et de favoriser la collaboration et l'entraide en son sein.

- Le **sprint review** : c'est une réunion de fin de sprint ayant pour but de présenter les incréments réalisés durant le sprint et de réfléchir à son déroulement afin de réorienter si besoin le produit (2h maximum par semaine de sprint).
- Le **sprint retrospective** : c'est une réunion ayant pour but d'analyser le sprint terminé, afin de mettre en place des stratégies pour améliorer les suivants (45 minutes par semaine de sprint).

La définition du « Done »

Un incrément est une tâche entièrement terminée produisant une fonctionnalité utilisable par l'utilisateur.

Une tâche est dite « done » et du sprint backlog, si et seulement si, la checklist ci-contre est entièrement cochée.

Définition du Done

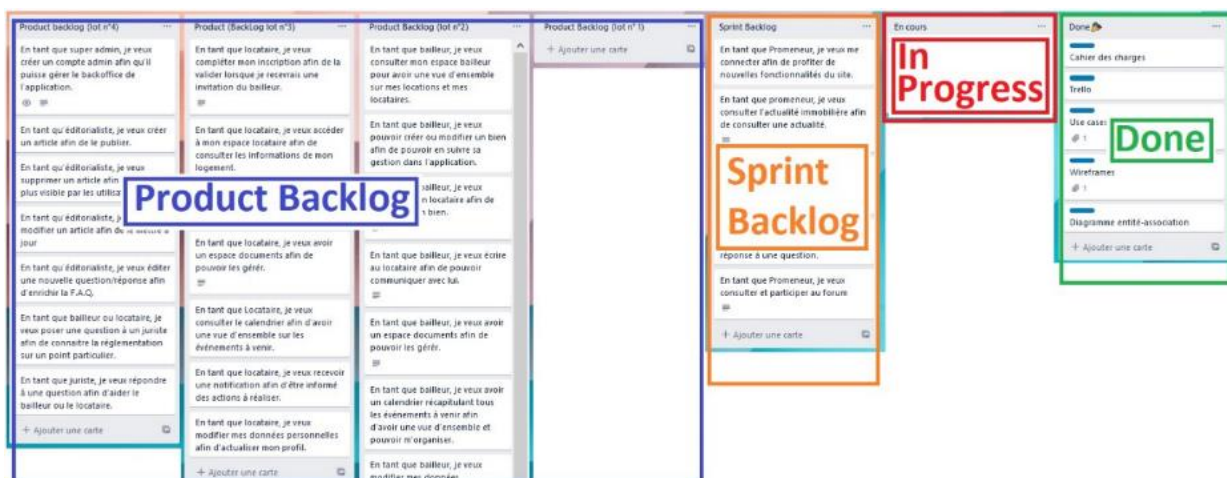
- ☐ Développement terminé
- ☐ Test unitaire réalisé
- ☐ Code revu
- ☐ Code documenté
- ☐ Campagne de test validée

Outil de gestion de projet

Nous utilisons Trello comme outil de gestion de projet.

<https://trello.com/b/OPz8EdJV/simplex-immo>

Dans notre Trello, nous utilisons un kanban pour lister et planifier les user stories à développer. Elles y sont toutes listées : celles à développer, en développement ou développées.



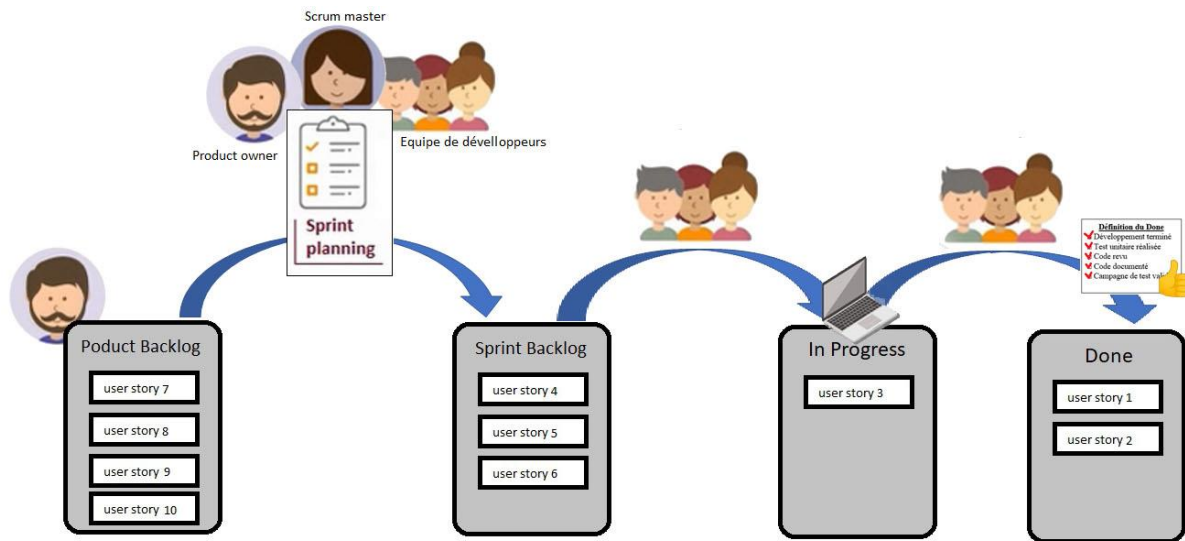
Au début du projet, le *product owner* place toutes les user stories dans la liste « **Product Backlog** ».

Par souci de lisibilité, le « product backlog » a été divisé en lots.

Lors du sprint planning, la *scrum team* déplace collégialement dans la liste « **Sprint Backlog** » les user stories qui seront développées lors du sprint à venir.

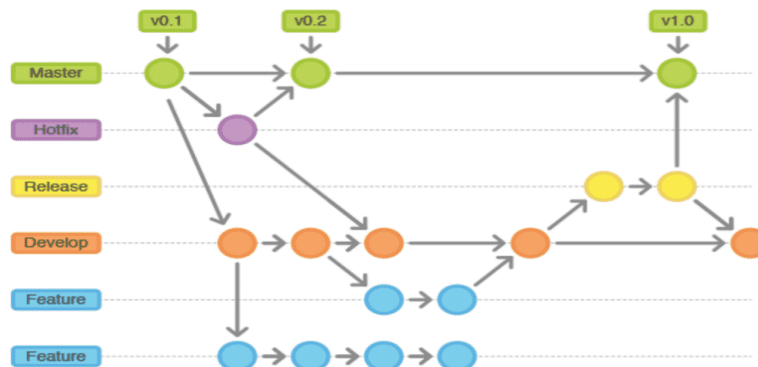
Lors du développement, lorsqu'une user stories est prise en charges par un *développeur*, il la déplace dans la liste « **In Progress** ».

Une fois qu'une user stories a été développée et satisfait aux critères de la définition du « Done », elle est placée dans la liste « **Done** ».



2) GitFlow

Les branches du GitFlow



La branche master : c'est la branche de production.

Chaque merge sur cette branche donnera lieu à une nouvelle version :

- Mineure pour les merges de hotfix
- Majeure pour les merges de release

Ce qui sera mergé sur cette branche devra faire preuve de la plus grande attention et sera de la responsabilité unique du lead développeur.

La branche dev : C'est la branche de développement qui recevra les fonctionnalités achevées, revues, documentées, testées et approuvées par le lead développeur.

Les branches feature : Ce sont des branches éphémères.

Chacune sera consacrée au développement d'une fonctionnalité précise. Une fois la fonctionnalité achevée, revue, documentée et testée, le développeur qui l'a traitée proposera un merge de sa branche sur la branche dev au lead dev.

- Si le merge est accepté et réalisé, la branche feature est alors supprimée.

- Si le merge est refusé, le lead dev indique les points à améliorer par le développeur avant qu'il puisse la proposer à nouveau au merge.

La branche release : C'est la branche qui permet de développer et tester une nouvelle version de l'application avant sa mise en production.

La branche hotfix : C'est une branche destinée à résoudre des bugs de la branche master.

Création d'une nouvelle fonctionnalité

Création d'une branche

Les développeurs travaillent sur les branches feature en partant de la branche dev.

- Créer une nouvelle branche en se plaçant dessus à partir de la branche « dev » :
`git checkout -b feat/myNewFeature dev`
- Faire un premier commit vide pour activer cette branche :
`git commit -m « new branche feat/myNewFeature » --allow-empty`

Développement de la fonctionnalité

Développer la fonctionnalité :

- en respectant les standards de code (voir [paragraphe III-4](#) : standard de code)
- en écrivant OBLIGATOIREMENT les tests unitaires et fonctionnels nécessaires (voir [paragraphe III-3](#) : campagne de tests)
- en commitant les changements importants (voir [paragraphe III-4](#) : règle de commit)
`git add <file name>` #ajouter le fichier au stage
`git commit` #commiter les changements
- En documentant et commentant votre code (voir [paragraphe III-4](#): commentaires)

Revue de la fonctionnalité

- Vérifier les standards de code avec code sniffer etc.
- Vérifier que vos tests et les autres passent correctement (voir [paragraphe III-3](#) : campagne de tests)
- Vérifier la qualité
- Vérifier la performance du code (requête sql, etc.)

Merge de la fonctionnalité sur la branche dev

- Envoyer votre code sur le dépôt distant github
`git push origin feat/myNewFeature`
- Proposer une pull-request au lead dev
 - o Si la pull-request est refusée, effectuer les modifications attendues par le lead développeur.
 - o Si elle est acceptée, le lead développeur mergera la pull-request.
- Si le lead dev vous demande de merger la pull-request à sa place
`git checkout dev` #se placer sur la branche dev
`git rebase feat/myNewFeature` #linéariser l'historique
`git merge feat/myNewFeature` (si nécessaire) #merger votre branche sur la branche dev

→ Résoudre les conflits éventuels
`git branch -d feat/myNewFeature`

#supprimer votre branche

Particularité des hotfixs et des releases :

Pour résoudre une hotfix ou créer une realease, l'esprit et la méthode sont identiques avec quelques variations listées dans le tableau ci-dessous.

	hotfix	release
Création de la branche à partir de	master	dev
Merge de la branche sur	dev et master	dev et master
Ajouter un tag de version	git tag V.1.2	git tag V.2.0

3) Campagne de tests

On ne recherche pas 100% de couverture de code par les tests.

Par contre, DOIVENT ETRE TESTES :

- L'intégralité du code métier.
- Les points critiques de l'application
- Le maximum de cas limite

Pour tester l'API, on utilise PHPUnit dans le dossier \tests

Avant de lancer les tests, préparer la BDD de test :

Composer prepareDB-test

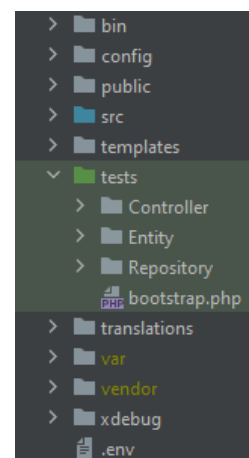
Vérifier que tous les tests (ceux existants et les vôtres) passent au vert.

Pour cela, lancer la commande : *php bin/phpunit*

PhpUnit propose un outil de contrôle de la couverture de code.

Dans la console : *php bin/phpunit --coverage-html public/test-coverage*

Dans le navigateur : *https://127.0.0.1:8000/test-coverage*



	Code Coverage								
	Lines			Functions and Methods			Classes and Traits		
Total	<div></div>	100.00%	264 / 264	<div></div>	100.00%	69 / 69	<div></div>	100.00%	18 / 18
Controller	<div></div>	100.00%	64 / 64	<div></div>	100.00%	11 / 11	<div></div>	100.00%	4 / 4
DataFixtures	<div></div>	100.00%	42 / 42	<div></div>	100.00%	5 / 5	<div></div>	100.00%	2 / 2
Entity	<div></div>	100.00%	50 / 50	<div></div>	100.00%	30 / 30	<div></div>	100.00%	2 / 2
Form	<div></div>	100.00%	18 / 18	<div></div>	100.00%	2 / 2	<div></div>	100.00%	2 / 2
Repository	<div></div>	100.00%	10 / 10	<div></div>	100.00%	3 / 3	<div></div>	100.00%	2 / 2
Security	<div></div>	100.00%	45 / 45	<div></div>	100.00%	11 / 11	<div></div>	100.00%	3 / 3
Service	<div></div>	100.00%	35 / 35	<div></div>	100.00%	7 / 7	<div></div>	100.00%	3 / 3

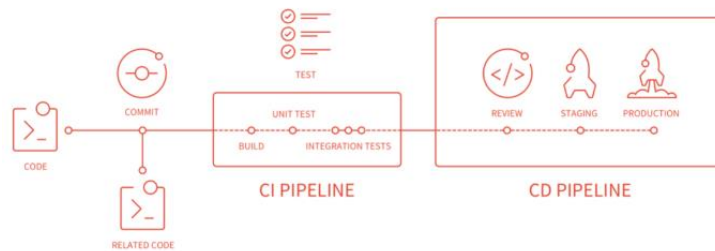
Intégration continue

Le script d'intégration continue se trouve dans le fichier **.github/workflow/symfony.yml**

Ce script indique que les tests seront lancés automatiquement sur github dès que :

- Un push sera réalisé sur la branche dev

- Un merge sera fait sur la branche dev
- Toute action sur la branche dev devra être validée par le lead dev.



Les résultats de l'intégration continue sont disponibles sur

<https://github.com/CamileGhastine/simplex-immobilier> en cliquant sur le bouton « action »  Actions .

4) Bonnes pratiques

Standard de code

Le code doit suivre quelques règles :

Les commentaires

- Les classes et les méthodes doivent être commentées en suivant les standards de phpDoc.
<https://www.phpdoc.org/>
- Sans en abuser, ne pas hésiter à commenter certaines parties du code si cela vous semble nécessaire.

Lisibilité du code

Votre code doit être bien présenté et bien indenté.

Au besoin installer PHP-CS-Fixer :

composer global require friendsofphp/php-cs-fixer

Et régler automatiquement les problèmes avec la commande :

php-cs-fixer fix src/monDossier/monFichier.php

Standard de code

Votre code doit respecter les standards de code de Symfony basés sur les standards [PSR-1](https://symfony.com/doc/4.4/contributing/code/standards.html), [PSR-2](https://symfony.com/doc/4.4/contributing/code/standards.html), [PSR-4](https://symfony.com/doc/4.4/contributing/code/standards.html) et [PSR-12](https://symfony.com/doc/4.4/contributing/code/standards.html) (<https://symfony.com/doc/4.4/contributing/code/standards.html>)

Au besoin, installer PHP-CodeSniffer :

composer global require "squizlabs/php_codesniffer="*

Et corriger pour chaque standard les problèmes soulevés par PHP-CodeSniffer :

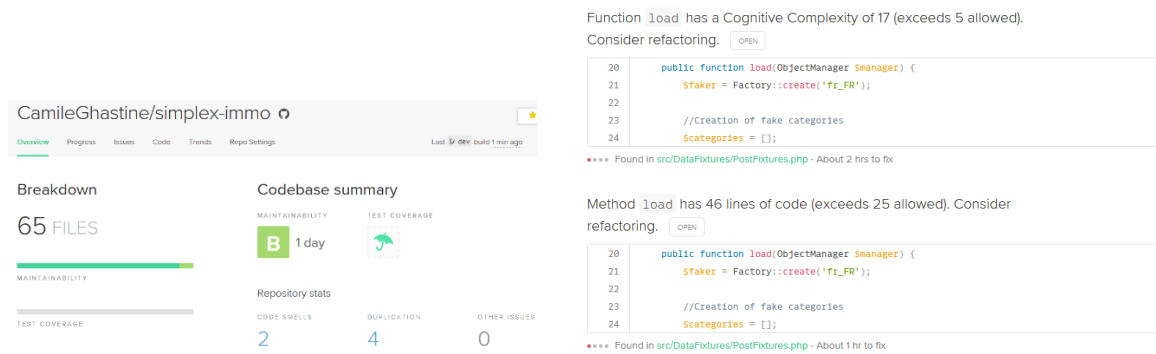
phpcbf --standard=PSR12 src/monDossier/monFichier

⚠ Ne jamais corriger les standards d'un code que vous n'avez pas écrit !!!

Qualité du code

Utiliser **CodeClimat** pour soumettre votre code à un contrôle de qualité :

<https://codeclimate.com/github/CamileGhastine/simplex-immobilier>



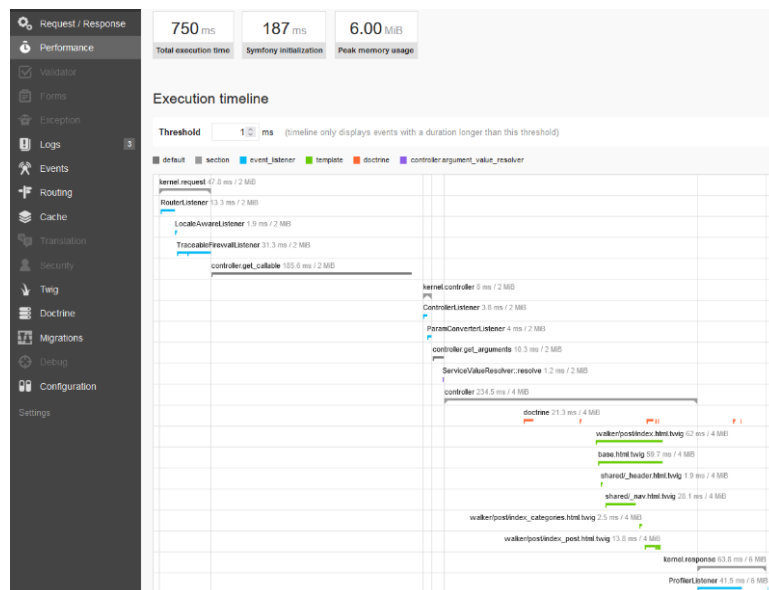
Le cas échéant, modifier votre code afin de régler les problèmes soulevés par CodeClimat.

Rester critique sur les propositions de CodeClimat. Dans la capture d'écran ci-dessus :

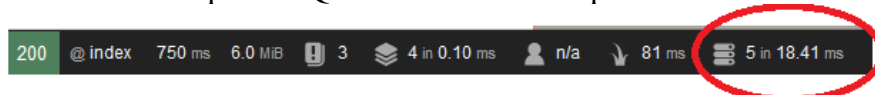
- Les 2 code smells concernent les fixtures qui ont une complexité trop élevée. Il est inutile de perdre du temps à résoudre cela.

Performance du code

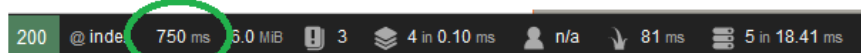
Pensez à utiliser le profiler et la debug toolbar de Symfony pour contrôler les performances de l'application.



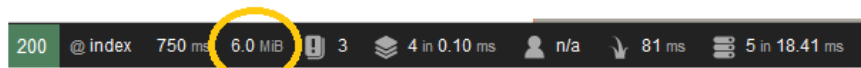
Minimiser le nombre de requêtes SQL en utilisant des requêtes customisées.



Minimiser le temps de chargement des pages en utilisant la mise en cache par exemple.



Penser aussi à contrôler l'utilisation de la mémoire

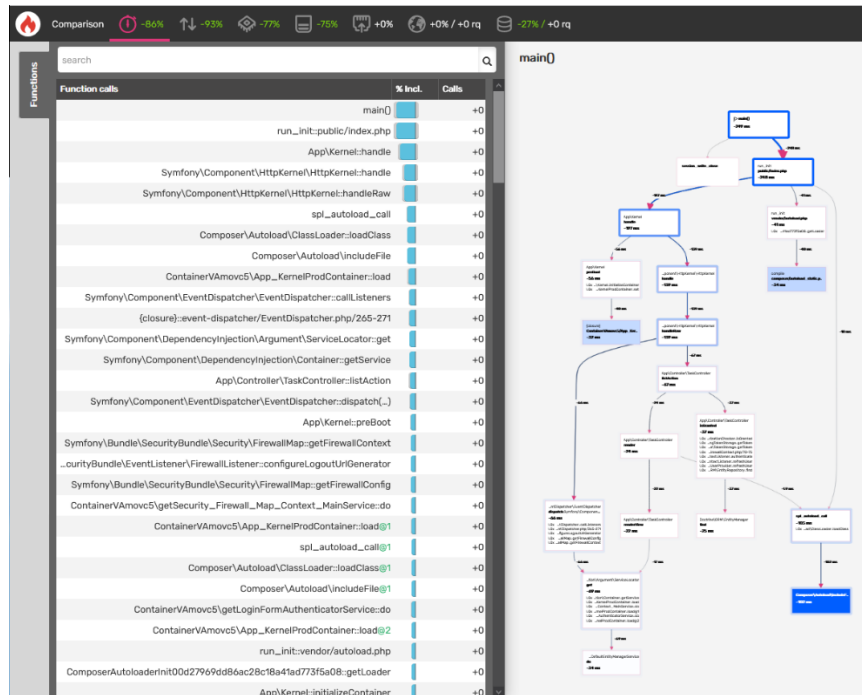


Il ne sera pas toujours possible de faire baisser ces trois paramètres. Il faut parfois faire des arbitrages en fonction du contexte sur les optimisations à réaliser.

Il est aussi possible d'utiliser la version gratuite de Blackfire (<https://www.blackfire.io/>) pour soumettre votre code à un contrôle de performance plus pointu.



Blackfire permet un contrôle plus fin des performances que le profiler de symfony. Il permet notamment d'avoir des informations sur l'utilisation des ressources du CPU serveur. Il permet aussi de comparer les performances de manière détaillée de deux profils.



En cas de dégradation de la performance, trouver les solutions d'optimisation avant de proposer une pull-request.

Règle de commit

A moins d'un commit évident, on évitera la commande `git commit -m « mon message »` pour commiter.

On préférera la commande `git commit`.

Une fois dans l'invite de commande, on écrira :

- Un titre explicite, pas trop long (49 caractères maximum)
- Un texte de description (pertinent et clair)

IV) Sécurité

1) Protocole sécurisé https

Afin de créer un certificat d'authentification en local, à l'initialisation d'un projet Symfony, toujours penser à effectuer la commande :

Symfony server:ca:install

Cela permet :

- D'éviter les problèmes lors d'un déploiement en https.
- D'éviter les problèmes avec les libraires ne fonctionnant qu'en https.

2) Les failles du web

La faille XSS

Même si toutes les libraires modernes (Twig) gèrent cette faille, lors du développement il faudra toujours l'avoir en tête afin de s'en prémunir.

La faille csrf

Même si Symfony gère cette faille en introduisant dans ces formulaires des tokens. Il est indispensable de vérifier qu'ils en contiennent bien tous un et ne pas hésiter à en ajouter dès que nécessaire.

L'injection sql

Là encore cette faille est gérée automatiquement par l'ORM doctrine. Gardez néanmoins toujours en tête ce risque lors du développement.

Les mots de passes

Symfony hash automatiquement les mots de passe.

Se prémunir des attaques par force brute.

Mettre en place des règles de validation par regex coté entité avec l'annotation :

@Assert\Regex(« /^\w+ / »)

Les mots de passe devront :

- Contenir entre 8 et 20 caractères.
- Contenir au moins une minuscule.
- Contenir au moins une majuscule.
- Contenir au moins un chiffre.
- Contenir au moins un caractère spécial.

3) L'authentification dans Symfony

La classe User

La class **User** étendant l'interface **UserInterface**.

<https://github.com/symfony/symfony/blob/5.2/src/Symfony/Component/Security/Core/User/UserInterface.php>

Chaque utilisateur possède au minimum les attributs ci-dessous :

- *id* : identifiant auto-incrémenté
- *email* : courriel
- *password* : mot de passe d'authentification hashé
- *role* : **ROLE_USER** , **ROLE_OWNER** , **ROLE_TENANT**, **ROLE_ADMIN**

Ces informations sont stockées par doctrine dans la table **user** de la base de données.

Le fichier de configuration de l'authentification

La configuration de l'authentification se trouve dans le fichier *config/packages/security.yaml*

- encoders : <https://symfony.com/doc/current/security.html#c-encoding-passwords>
Avant d'enregistrer le mot de passe en base de données, il est OBLIGATOIRE de le hasher à l'aide du service :
Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface.
<https://github.com/symfony/security-core/blob/5.x/Encoder/UserPasswordEncoderInterface.php>
- providers : <https://symfony.com/doc/current/security.html#b-the-user-provider>
- firewalls : <https://symfony.com/doc/current/security.html#a-authentication-firewalls>
On utilise le système d'authentification **guard** afin d'exercer un plus grand contrôle sur le processus d'authentification.
https://symfony.com/doc/current/security/guard_authentication.html
- acces_control : <https://symfony.com/doc/current/security.html#add-code-to-deny-access>
- role_hierarchy : <https://symfony.com/doc/current/security.html#hierarchical-roles>

Le SecurityController

Le contrôleur de sécurité se situe dans le fichier *src/Controller/security.php* et contient les méthodes :

- **login()** appelée par l'URL : **/login** nommée **app_login**.
- **logout()** appelée par l'URL : **/logout** nommée **app_logout**.

Il n'y a pas de logique dans la méthode **login()** pour vérifier le formulaire de connexion et authentifier l'utilisateur. Symfony gère cela en interne avec un **listener** qui va traiter le formulaire. <https://symfony.com/doc/current/components/security/authentication.html>

En cas de réussite, la méthode **onAuthenticationSuccess()** du *LoginFormAuthenticator* est appelée et redirige l'utilisateur vers la page d'accueil (**homepage**).

```
Return new RedirectResponse($this->urlGenerator->generate('homepage'));
```

Les Voters

Utiliser les **voters** pour gérer les autorisations en centralisant toute la logique d'autorisation hors des contrôleurs.

Ces voters pourront être interrogés à loisir avec :

- **isGranted()** pour autoriser une action
- **denyAccessUnlessGranted()** pour organiser une redirection, afin qu'un code non autorisé soit lu.

<https://symfony.com/doc/current/security/voters.html>

V) Procédure d'installation et de déploiement

1) Procédure d'installation

La procédure d'installation est également disponible dans le *README.md* à la racine du projet.

- 1) Ouvrir la console et choisir un dossier pour installer le projet.
- 2) Télécharger le projet avec la commande :
git clone https://github.com/CamileGhastine/simplex-immo.git
- 3) Vous rendre dans le dossier du projet avec la commande :
cd simplex-immo
- 4) Installer composer et ses dépendances avec la commande :
composer install
- 5) Ouvrir le fichier *.env* et configurer la connexion à la base de données
- 6) Créer la base de données et charger les fixtures avec la commande :
Composer prepareDB

Avant de commencer à coder, créer votre branche à partir de la branche dev en la nommant avec un nom explicite, reflétant la fonctionnalité développée (voir [paragraphe III-2](#)) :

git checkout -b feat/leNomDeMaBranche

Vous pouvez vous connecter en tant qu'utilisateur enregistré avec les identifiants ci-dessous :

- email: user@user.fr
- mot de passe: password

2) Procédure de déploiement

Le déploiement de l'application sera optimisé et facilité grâce à sa containerisation avec Docker.

Sur PlanetHoster, on configurera deux serveurs :

- Un serveur de production qui hébergera le site web.
- Un serveur de préprod qui permettra de tester l'application et son déploiement avant sa mise en production finale sur le serveur de production.

VI) Déroulement des sprints

1) Sprint 1

Résumé du sprint planning

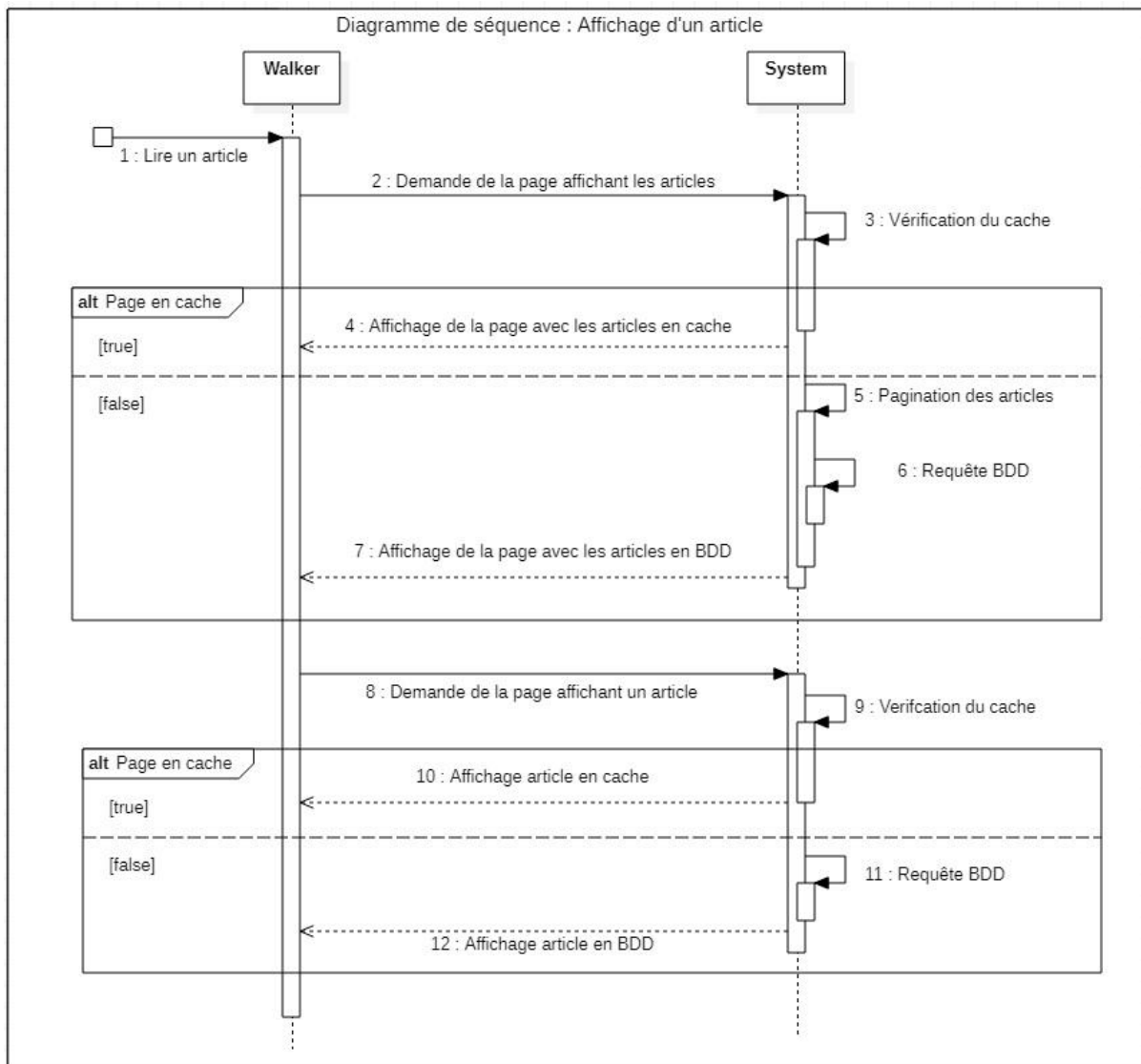
- Sprint goal : Avoir une vitrine du site fonctionnelle et navigable.
- Sprint backlog : voir Trello <https://trello.com/b/OPz8EdJV/simplex-immo>
- De la user story à la tâche :
 - « En tant que promeneur, je veux consulter l'actualité immobilière afin de m'informer. »
 - Mise en place du projet
 - Structuration du html du site (base.html.twig, header, nav, footer)
 - Structuration du html de la homepage (promotion et news)
 - Entités Post et Image avec une relation OneToMany
 - Fixtures posts
 - Requête customisée dans PostRepository (post + image.poster)
 - « En tant que promeneur, je veux consulter la liste des articles et tutoriels afin de consulter un article ou un tutoriel. »
 - Structuration html de la page index
 - Entité catégorie relation ManyToOne avec Post
 - Service Pagination
 - Fixtures Categories
 - Affichage des catégories
 - Affichage des posts par catégories
 - Requête customisée (post par catégorie avec image)
 - Structure html de la page show
 - Entité Video relation ManyToOne avec Post + fixtures videos + Repository
 - « En tant que promeneur, je veux consulter la F.A.Q afin d'avoir une réponse à une question. »
 - Structuration html de la page faq
 - Entity faq
 - Affichage de la faq
 - « En tant que Promeneur, je veux créer un compte afin de posséder un compte utilisateur. »
 - Authentification avec Guard de Symfony
 - Mot de passe oublié
 - « En tant que Promeneur, je veux me connecter afin de profiter de nouvelles fonctionnalités du site. »
 - Register-form de Symfony
- ➔ Mettre en place un cache pour améliorer les performances.
- ➔ Mettre en place des requêtes AJAX lorsque la pagination ou un affichage par catégorie est utilisée.

Remarque : Pas de difficulté technique particulière repérée sur ce sprint

Le sprint et ses incréments

- Les incréments du sprint

- Les documents produit lors du sprint



VII) Bug connus