

Audit SPO

Rapport détaillé

→ 18 Mars 2024

onepoint.
beyond the obvious



Sommaire

1. Contexte et objectifs de l'audit
2. Audit de l'architecture
3. Analyse Performance
4. Analyse du code front
5. Analyse du code back
6. Chaine de fabrication et déploiement
7. Observabilité
8. Analyse des interviews
9. Bilan et synthèse des préconisations
10. SPO en 2024 et après
11. Conclusion



Sommaire

1. Contexte et objectifs de l'audit
2. Audit de l'architecture
3. Analyse Performance
4. Analyse du code front
5. Analyse du code back
6. Chaine de fabrication et déploiement
7. Observabilité
8. Analyse des interviews
9. Bilan et synthèse des préconisations
10. SPO en 2024 et après
11. Conclusion





AUDIT

DE L'APPLICATION

SPO

RAPPEL DU BESOIN

SHOM a sollicité onepoint afin de réaliser un audit complet de l'application SPO. Cette application permet de générer les documents PDF des ouvrages du SHOM.

L'objectif principal de l'audit est de faire un bilan sur la qualité du code source, et des constats sur l'architecture applicative et technique.

SHOM demande un regard extérieur qui pourrait à la fois faire des constats et préconisations sur le fonctionnement technique du produit, sur son implémentation dans le SI.

Cet audit se découpe en plusieurs parties :

1. Etude de l'architecture applicative et technique

- ☐ Regard sur l'architecture mise en en place par rapport aux besoins. Analyse de la solution sur les aspects logiciel et infrastructure

2. Etude de la performance applicative

- ☐ Analyse des requêtes les plus lentes et les pages les plus consommatrices de ressources
- ☐ Analyse des traces d'erreurs applicatives

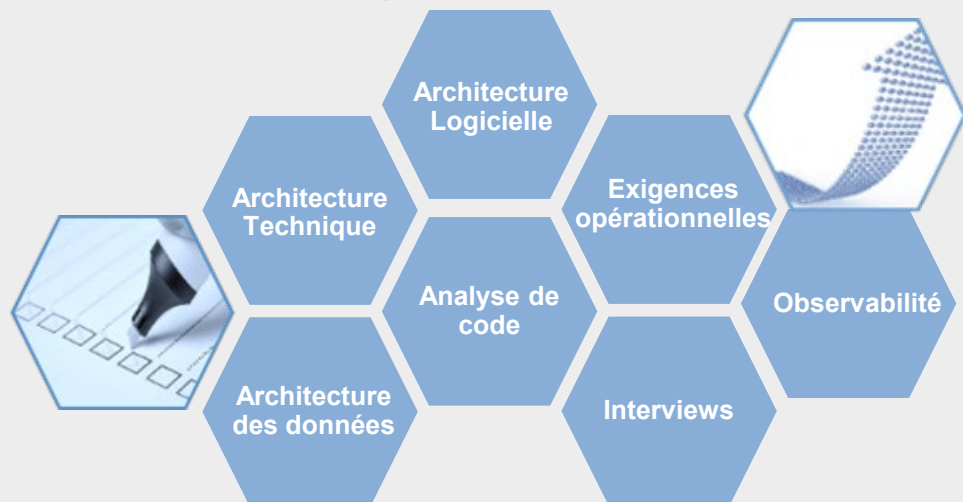
3. Etude de la documentation

- ☐ Vérification de la qualité de la documentation afin de vérifier si elle permet de réaliser une maintenance applicative et d'exploitation

4. Analyse du code applicatif, chaine de compilation et déploiement

- ☐ Contrôle de la qualité globale du code (conformité vis-à-vis des normes et standards de développement, analyse CVE performance, taux de commentaires, librairies obsolètes, bon usage de l'ORM).
- ☐ Recommandation technique d'amélioration court (quick win), moyen et long terme

Périmètre de l'Analyse :



Equipe Audit

Auditeurs

Arnaud GENRE-GRANDPIERRE

Architecte : responsable de mission
a.genre-grandpierre@groupeonepoint.com

Jean-Baptiste ALIGNÉ

Expert Python
jb.aligne@groupeonepoint.com

Pascal LUCAS

Leader Delivery
p.lucas@groupeonepoint.com

Les ateliers réalisés

Présentation générale, fonctionnelle et technique



25/01



MOE

Présentation technique



16/02



MOE

1. Contexte et objectifs de l'audit
2. Audit de l'architecture
3. Analyse Performance
4. Analyse du code front
5. Analyse du code back
6. Chaine de fabrication et déploiement
7. Observabilité
8. Analyse des interviews
9. Bilan et synthèse des préconisations
10. SPO en 2024 et après
11. Conclusion

Audit de l'architecture

Description générale

SPO repose sur une architecture classique front back en python avec une base Postgresql

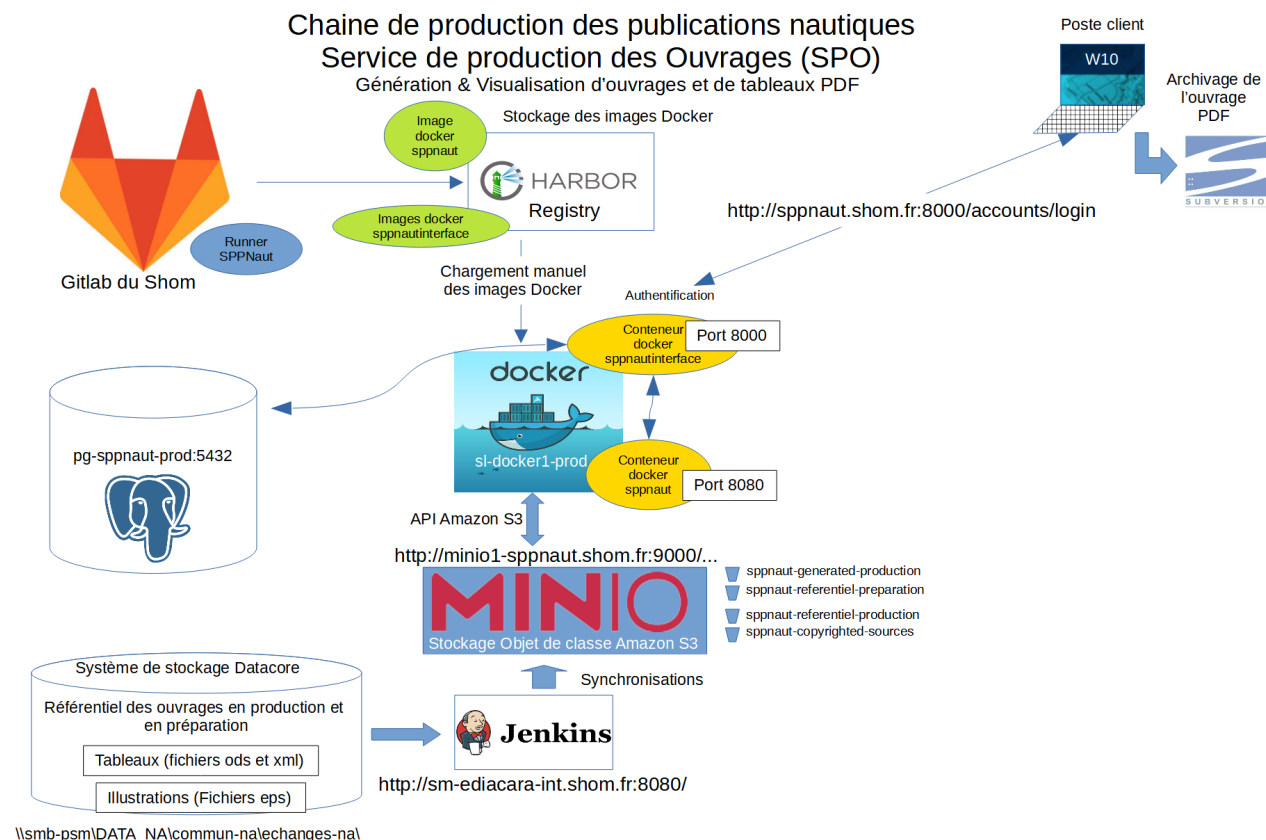
L'infrastructure technique est hébergée en interne.

Les modules frontend, backend sont fournis sous forme d'image Docker.

L'ouvrage est fait à partir de fichiers ODS, XML et d'illustrations, le tout stocké sur un espace réseau. Une chaîne logicielle permet de construire l'ouvrage au format PDF.

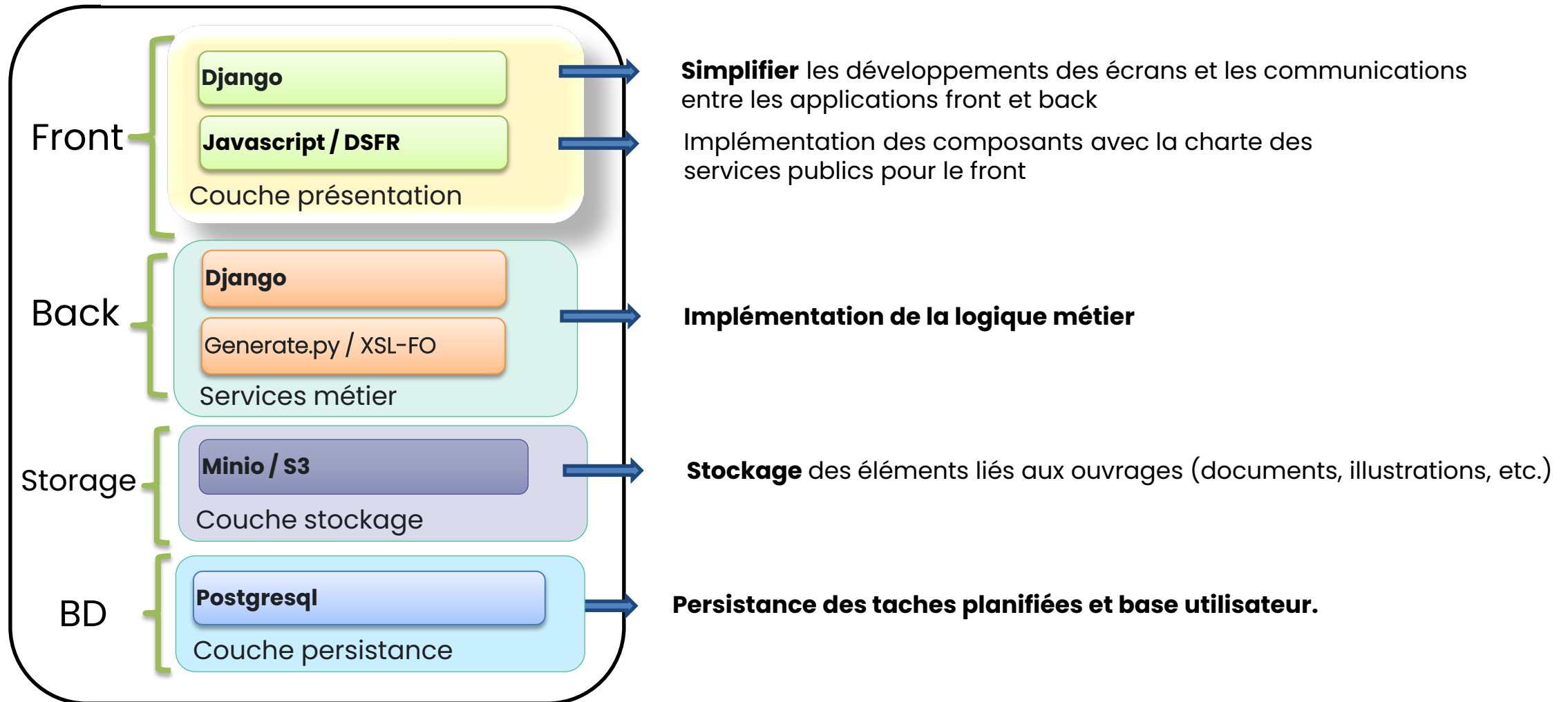
Les PDF des ouvrages générés sont stockés sur un espace stockage de type S3 (actuellement Minio).

Les utilisateurs peuvent ensuite télécharger les ouvrages disponibles.



Audit de l'architecture

Stack technique globale, principaux framework utilisé sur SPO



Analyse de la documentation projet

L'application n'est pas suffisamment documentée

Livrables		Commentaires
Livrables dev	Spécifications fonctionnelles	Inexistant
	Doc de conception, Spécifications techniques	Documentation as code
	Cahier de test	Inexistant
	Normes et procédures en vigueur	Incomplet
Livrables projet en lien avec la mise en production	Documents d'architecture technique	Incomplet
	Procédures d'installation	Incomplet
	Procédures d'exploitation	Pas de procédure venant de l'équipe dev
	Cartographie de la plateforme de production	Incomplet
Livrable à destination des utilisateurs	Manuel utilisateur	Inexistant

Audit de l'architecture

Constats globaux : aspects techniques

Observations

- **Réseau** : pas d'isolation réseau
- **Répartition de charges** : application simple : pas de répartiteur.
- **Modularité** : L'application est structurée sous la forme de modules fonctionnels interdépendants. L'application est modulaire avec un front et un back. On aurait pu faire plus simple au vu du nombre faible de fonctionnalités.
- **Redondance** : application simple : pas de redondance.
- **Stockage et données** : La base de données Postgres, est mutualisée, ce qui est normal vu la faible volumétrie. Les PDF sont stockés sur un espace de stockage de type S3. L'usage de S3 est discutable au vu de la faible volumétrie des PDF et le fait que l'appli est déployée sur VM. On aurait pu les stocker sur disque ou dans la base de données.
- **Environnements** : Différents environnements sont mis en place pour gérer les différentes phases du projet (développement, recette, production).
- **Cloisonnement** : Les modules sont cloisonnés. Containerisation via Docker.
- **Sécurité** : Les flux d'échanges vers S3 sont sécurisés. Le reste non.
- **Dimensionnement** : Le dimensionnement est adapté au fonctionnement de l'application, au vu du nombre d'utilisateurs
- **Ressource et allocation** : Les outils de surveillance de la consommation ne peuvent pas donner une vision précise de la consommation, permettant de relier la consommation aux événements et d'adapter le dimensionnement.
- **Cache** : Pas de cache, ni interne, ni externe.

Audit de l'architecture

Autres constats

Observations

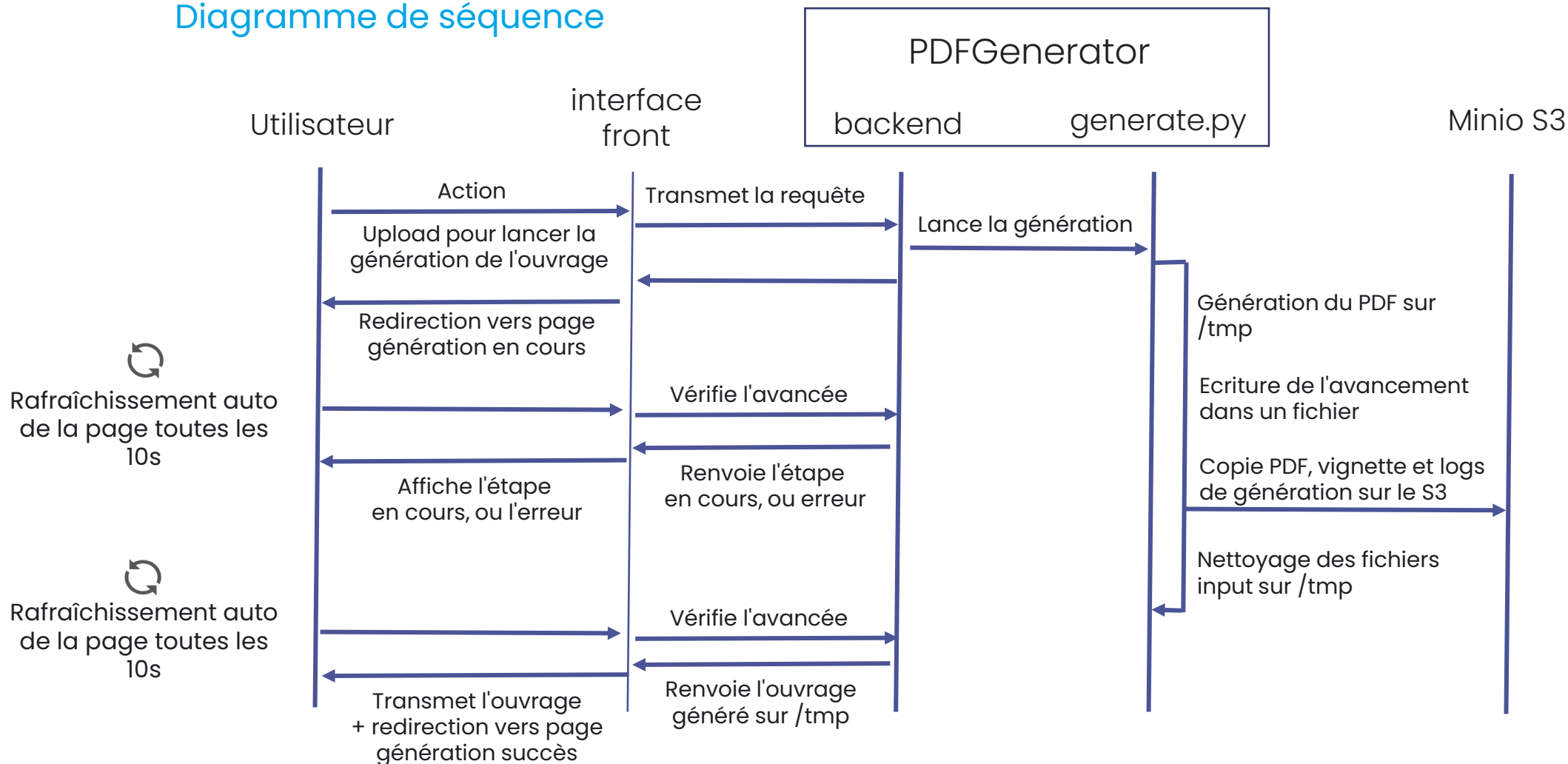
- **Exigences non fonctionnelles** : Les exigences fonctionnelles ne sont pas formalisées
- **Temps de réponse** : La majorité des temps de réponse sont globalement corrects du point de vue utilisateur. Seules les requêtes de génération de PDF peuvent créer des temps d'attente pour l'utilisateur.
- **SLA** : Les SLA ne sont pas formalisés d'une manière claire et accessible pour l'ensemble des parties prenantes.
- **Modèle de données** : Le modèle de données forme un tout rationalisé, adapté aux besoins de l'application.
- **Industrialisation** : La construction et déploiement ne sont pas automatisés.
- **Interopérabilité** : Le Système respecte les standards du marché. Des évolutions seraient possibles pour permettre au système d'échanger avec des Systèmes Tiers
- **Scalabilité** : Le système est potentiellement scalable en dehors de la tâche de génération PDF manuelle.
- **Documentation** : La documentation est le point noir du projet.
- **Sécurité** : Les moyens de sécurisation mis en œuvre sont efficaces. La gestion IAM aurait toutefois pu s'appuyer sur le SSO Shom.
- **Sauvegarde** : Existence d'une stratégie d'archivage des ouvrages.

1. Contexte et objectifs de l'audit
2. Audit de l'architecture
3. Analyse Performance
4. Analyse du code front
5. Analyse du code back
6. Chaine de fabrication et déploiement
7. Observabilité
8. Analyse des interviews
9. Bilan et synthèse des préconisations
10. SPO en 2024 et après
11. Conclusion



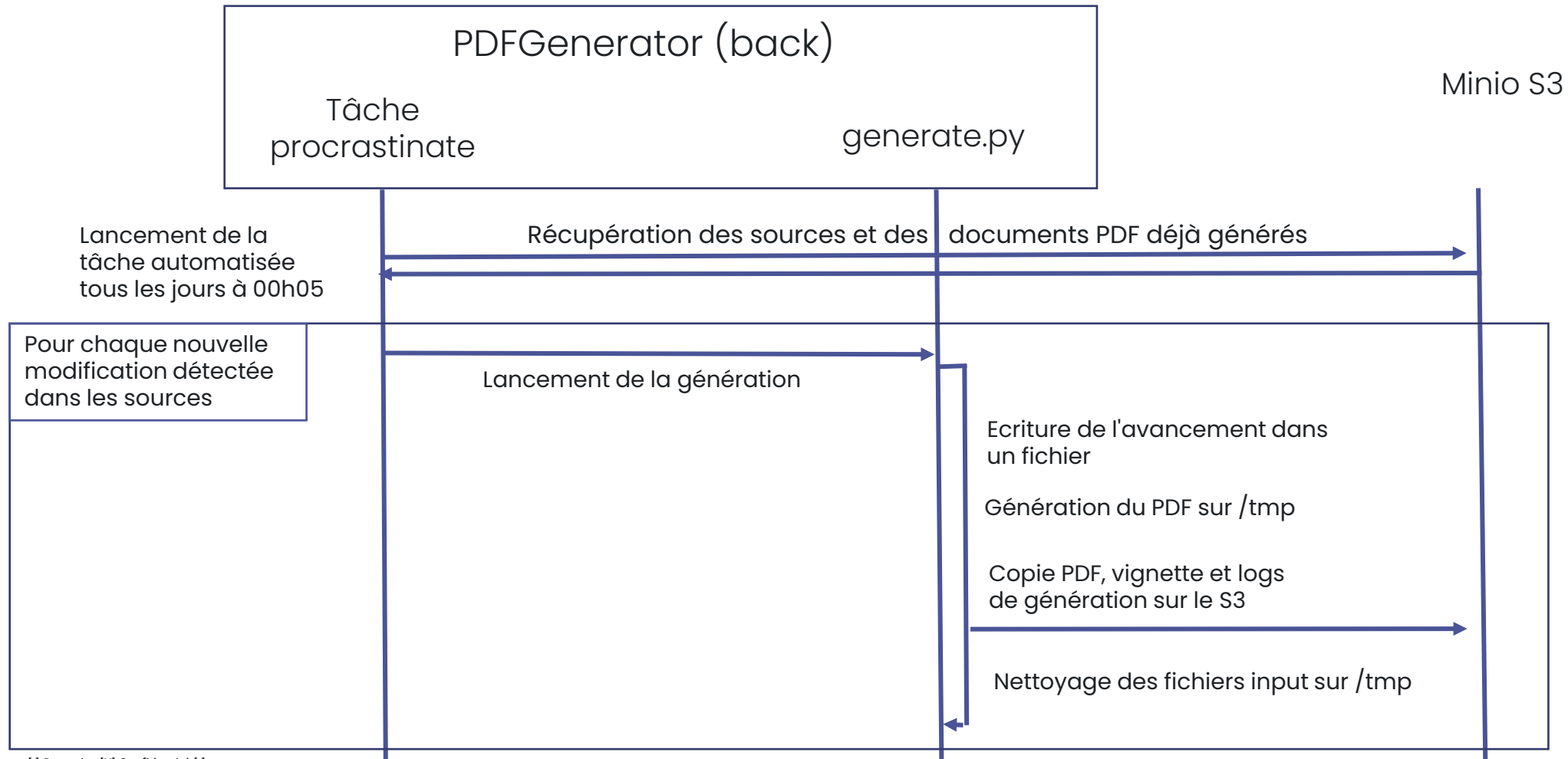
Génération d'un ouvrage par un utilisateur

Diagramme de séquence



Génération quotidienne d'un ouvrage

Diagramme de séquence



Détail des étapes de génération

La génération prend plusieurs dizaines de secondes, majoritairement lors de la génération de l'ouvrage et l'optimisation

Etapes	Temps d'exécution
Récupération des sources de l'ouvrage dans le référentiel	3,6s
Conversion des illustrations communes	9,9s
Conversion des illustrations de l'ouvrage	7,8s
Génération des fichiers intermédiaires	5,1s
Génération de l'ouvrage (PDF)	31,4s
Génération de la vignette	0,5s
Génération des métadonnées	1,26s
Compression du fichier PDF (optimisation)	29,7s
Sauvegarde de l'ouvrage	2,5s
Total	91,76s

A noter que les mesures ont été faites uniquement sur les étapes identifiables dans le fichier de log qui a été fourni. Les étapes de récupération et copies de ressources hors S3 ne passent pas sous forme de commandes bash et ne sont donc pas tracées.

Analyse des pages les plus lentes

Aucun constat ne peut être fait faute de source de données

L'application n'est pas reliée à un reverse proxy.

Faute de source de données comme un access_log, il n'est pas possible d'analyser les temps de réponse de l'application et les code de réponse HTTP

1. Contexte et objectifs de l'audit
2. Audit de l'architecture
3. Analyse Performance
4. Analyse du code front
5. Analyse du code back
6. Chaine de fabrication et déploiement
7. Observabilité
8. Analyse des interviews
9. Bilan et synthèse des préconisations
10. SPO en 2024 et après
11. Conclusion

Analyse de l'obsolescence des librairies utilisées

Pas d'obsolescence majeure constatée des librairies

Librairie Python	Version utilisée	Dernière version disponible	Commentaires
python3	3.10	3.12	Date de fin de support:officiel : octobre 2026
django	4.1.9	5.0	Support officiel arrêté depuis début 2024 Version 5.0 compatible avec Python 3.10 mais pourrait nécessiter d'autres mises à jour
django-extensions	3.2.1	3.2.3	Ajout de support pour Python 3.11 et psycopg3
requests	2.31.0	2.31.0	
python3-decouple	3.8	3.8	
psycopg2	2.9.6	2.9.9	Psycopg3 est disponible, ajout de fonctionnalités pour les dernières versions de Python et PostgreSQL
awscli	1.27.154	1.32.47	
boto3	1.24.58	1.34.47	
natsort	8.3.1	8.4.0	



Analyse de l'usage du framework Django

Les bonnes pratiques sont respectées

- Organisation du projet
 - La partie applicative Django (formulaires, vues, urls, etc.) a été séparée de la partie configuration centrale de Django, ce qui permet une bonne visibilité sur la constitution du projet.
- Utilisation des templates maîtrisée
 - Taille légère
 - Inclusion dans les templates pour éviter la duplication
 - Système de gabarit bien externalisé



Analyse de l'implémentation

Implémentation minimaliste et satisfaisante

Points positifs

- Code clair et compréhensible
- Tests unitaires présents
- Bonne séparation des formulaires et de la vue
- Utilisation de slugs pour les URLs associées à des ouvrages
- Utilisation de render pour optimiser l'utilisation des templates

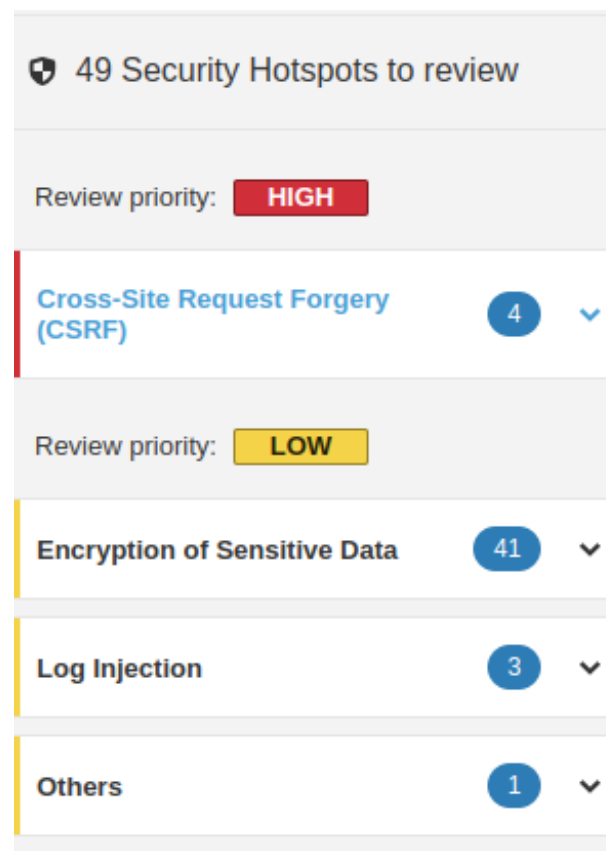
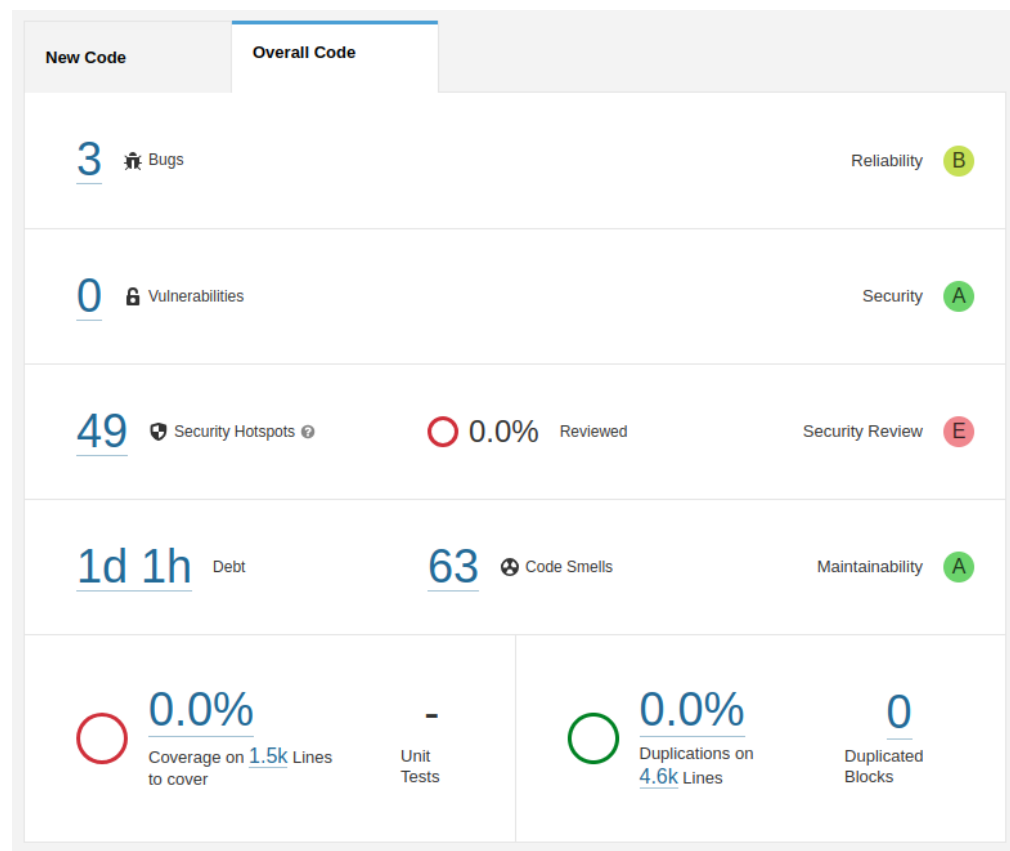
Points négatifs

- Gestion restreinte des erreurs et codes de réponse HTTP (seulement erreur interne serveur ou page non trouvée)
- Code factorisable au niveau des vues
- Absence de journaux d'application (sentry présent dans le code mais inutile car pas de serveurs sentry en interne Shom)



Analyse qualimétrie Sonar

Dans l'ensemble les bonnes pratiques sont respectées avec une mise en œuvre maîtrisée.



Les retours de Sonar sur la sécurité concernent en grande majorité

- soit le manque de token CSRF,
- soit la non utilisation de HTTPS sur certaines requêtes



Analyse qualimétrie Sonar

Dans l'ensemble les bonnes pratiques sont respectées avec une mise en œuvre maîtrisée.

▼ Type			
🐛	Bug		3
🔒	Vulnerability		0
☹️	Code Smell		63
▼ Severity			
🚫	Blocker	0	🟢 Minor 3
🚨	Critical	52	📘 Info 0
🚧	Major	11	

Il existe du code factorisable au niveau des vues, mais il s'agit d'une légère partie. La duplication de code reste faible.

Le score de "code smells" est dû au non-usage de constantes, cela représente la totalité des "code smells" jugés critiques.

Il reste également 4 « FIXME » concernant des ajouts mineurs dans le code.

Le formatter utilisé Black n'indique pas de modification nécessaire.



- Corriger les "code smells" et clarifier la pertinence des FIXME restants dans le code
- Mettre en place un linter (Sonarlint par exemple)

Tests

Constats

Points positifs

- Présence de tests malgré l'interface minimaliste
- Vérification de la bonne exécution des redirections selon réussite ou échec de la génération des ouvrages

Points négatifs

- Les tests des vues nécessitent une base de données en parallèle pour pouvoir s'exécuter car tous les tests sont réalisés avec la fixture *admin_client* qui impose l'usage d'une base de données



Ne pas passer par la fixture *admin_client* pour les tests des vues, mais plutôt passer par un utilisateur de test pour les requêtes nécessitant une authentification.

Exécution des tests

Bilan des tests exécutés **SANS** une base de données en parallèle

- 17 tests réalisés :
 - 8 tests validés, dont un warning
 - 9 erreurs
- **MAIS** toutes les erreurs et le warning sont dus à l'absence de base PostgreSQL lors des tests réalisés sur les vues, à cause de la fixture utilisée *admin_client*
- Tout est validé côté tests sur les formulaires et sur les ouvrages

} **47.16% de réussite**

```

===== warnings summary =====
unit_tests/spo/test_views.py::TestOuvragesByDate::test_group_and_desc_sort_by_document_date
  C:\Users\jb.aligne\AppData\Local\Programs\Python\Python310\lib\site-packages\django\db\backends\postgresql\base.py:336: RuntimeWarning: Normally Django will use a connection to the 'postgres' database to avoid running initialization
  queries against the production database when it's not needed (for example, when running tests). Django was unable to create a connection to the 'postgres' database and will use the first PostgreSQL database instead.
    warnings.warn(

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== short test summary info =====
ERROR unit_tests/spo/test_views.py::TestOuvragesByDate::test_group_and_desc_sort_by_document_date - django.db.utils.OperationalError: connection to server at "localhost" (::1), port 5434 failed: Connection refused (0x0000274D/10061)
ERROR unit_tests/spo/test_views.py::TestOuvragesByDate::test_sort_by_ouvrage_name - django.db.utils.OperationalError: connection to server at "localhost" (::1), port 5434 failed: Connection refused (0x0000274D/10061)
ERROR unit_tests/spo/test_views.py::TestOuvragesByName::test_sort_by_ouvrage_name - django.db.utils.OperationalError: connection to server at "localhost" (::1), port 5434 failed: Connection refused (0x0000274D/10061)
ERROR unit_tests/spo/test_views.py::TestPublicationGenerationInProgress::test_generation_failed - django.db.utils.OperationalError: connection to server at "localhost" (::1), port 5434 failed: Connection refused (0x0000274D/10061)
ERROR unit_tests/spo/test_views.py::TestPublicationGenerationInProgress::test_generation_in_progress - django.db.utils.OperationalError: connection to server at "localhost" (::1), port 5434 failed: Connection refused (0x0000274D/10061)
)
ERROR unit_tests/spo/test_views.py::TestPublicationGenerationInProgress::test_generation_success - django.db.utils.OperationalError: connection to server at "localhost" (::1), port 5434 failed: Connection refused (0x0000274D/10061)
ERROR unit_tests/spo/test_views.py::TestPublicationGenerationEnded::test_generation_success - django.db.utils.OperationalError: connection to server at "localhost" (::1), port 5434 failed: Connection refused (0x0000274D/10061)
ERROR unit_tests/spo/test_views.py::TestPublicationGenerationEnded::test_generation_failed - django.db.utils.OperationalError: connection to server at "localhost" (::1), port 5434 failed: Connection refused (0x0000274D/10061)
ERROR unit_tests/spo/test_views.py::TestPublicationGenerationEnded::test_generation_in_progress - django.db.utils.OperationalError: connection to server at "localhost" (::1), port 5434 failed: Connection refused (0x0000274D/10061)
===== 8 passed, 1 warning, 9 errors in 16.85s =====

```



Exécution des tests

Bilan des tests exécutés **AVEC** une base de données en parallèle

- 17 tests réalisés :
 - 17 tests validés
 - 9 warnings
- } **100% de réussite**
- Les erreurs liées à la fixture *admin_client* ont disparu grâce à la présence de la base de données
 - Tous les warnings sont dus à l'absence d'un répertoire pour la librairie whitenoise

```
===== warnings summary =====
unit_tests/spo/test_views.py::TestOuvragesByDate::test_group_and_desc_sort_by_document_date
unit_tests/spo/test_views.py::TestOuvragesByDate::test_sort_by_ouvrage_name
unit_tests/spo/test_views.py::TestOuvragesByName::test_sort_by_ouvrage_name
unit_tests/spo/test_views.py::TestPublicationGenerationInProgress::test_generation_failed
unit_tests/spo/test_views.py::TestPublicationGenerationInProgress::test_generation_in_progress
unit_tests/spo/test_views.py::TestPublicationGenerationInProgress::test_generation_success
unit_tests/spo/test_views.py::TestPublicationGenerationEnded::test_generation_success
unit_tests/spo/test_views.py::TestPublicationGenerationEnded::test_generation_failed
unit_tests/spo/test_views.py::TestPublicationGenerationEnded::test_generation_in_progress
C:\Users\jb.aligne\AppData\Local\Programs\Python\Python310\lib\site-packages\whitenoise\base.py:109: UserWarning: No directory at: C:\Users\jb.aligne\OneDrive - ONEPOINT\Documents\Audit - SHOM\sources\sppnaut-spo-AHFormatterV73\sppnaut-spo-AHFormatterV73\interface\static\collected\
  warnings.warn(f"No directory at: {root}")

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html

===== 17 passed, 9 warnings in 4.75s =====
```

Observations sécurité front

- Django offre nativement de bonnes protections de sécurité
 - Renforcement de la protection native contre les injections XSS dans les templates
- Utilisation d'un "intergiciel" natif Django pour les attaques CSRF, il pourrait être intéressant de contrôler explicitement le moment de la validation CSRF côté vue
- L'application ne vérifie que l'extension des fichiers uploadés par l'utilisateur.
- Pas d'utilisation de HTTPS
- L'authentification se fait selon le mode basic auth



- Vérifier le MIME Type des fichiers uploadés
- Mettre en place HTTPS que ce soit en configuration et dans le code (exemple : utiliser le décorateur `@secure_require` pour forcer l'usage du SSL pour certaines actions)

Analyse du Frontend

Conclusion générale

- Le code est globalement de bonne qualité. Il reste clair malgré le manque de documentation.
- L'usage de Django est bien réalisé. La séparation entre la partie configuration centrale de Django et la partie applicative métier rajoute de la clarté.
- Les tests unitaires présents sont pertinents, même si l'utilisation obligatoire d'une base de données pour tous les passer reste contraignante.
- La gestion restreinte des erreurs de réponse HTTP ainsi que l'absence de logs sur les différentes actions réalisées par les utilisateurs impactent la maintenabilité.
- D'un point de vue sécurité, l'application est à un niveau correct, il serait tout de même préférable de mettre en place HTTPS via un reverse proxy.



Synthèse des préconisations

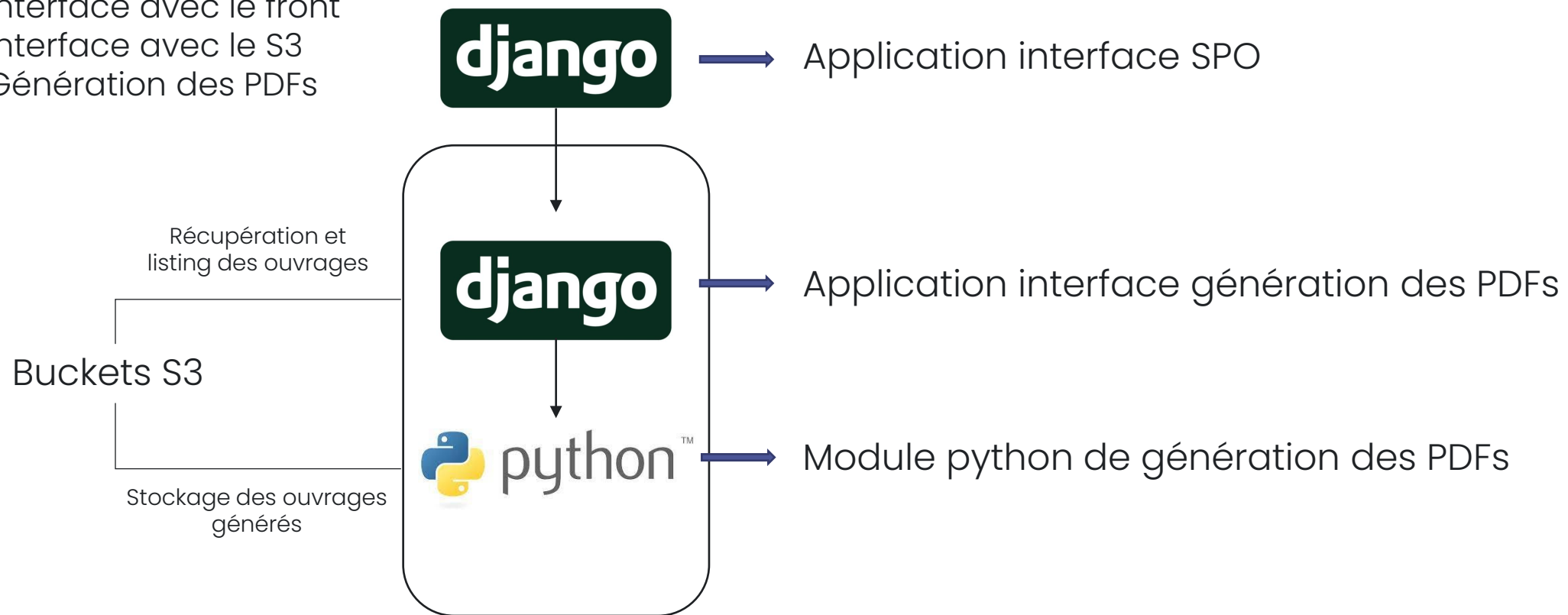
Thématiques	Priorité	Effort à engager	Impact perf	acteur	Description
Sécurité	Haute	Faible	Faible	Dev / Infra	Mise en place HTTPS
Sécurité	Moyenne	Faible	Faible	Dev	Vérifier le contenu des fichiers ajoutés par l'utilisateur
Maintenabilité	Moyenne	Faible	Faible	Dev	Ajouter des logs sur les actions réalisées
Maintenabilité	Moyenne	Faible	Faible	Dev	Etoffer la gestion des erreurs HTTP
Maintenabilité	Faible	Faible	Faible	Dev	Corriger les "code smells" et clarifier la pertinence des FIXME dans le code
Maintenabilité	Faible	Faible	Faible	Dev	Mettre en place un linter
Contrôle	Faible	Faible	Faible	Dev	Ajouter des tests vérifiant la conformité du MIME Type
Contrôle	Faible	Faible	Faible	Dev	Ne plus utiliser admin_client comme fixture pour les tests des vues

1. Contexte et objectifs de l'audit
2. Audit de l'architecture
3. Analyse Performance
4. Analyse du code front
5. Analyse du code back
6. Chaîne de fabrication et déploiement
7. Observabilité
8. Analyse des interviews
9. Bilan et synthèse des préconisations
10. SPO en 2024 et après
11. Conclusion

Architecture logique

3 sources de code à analyser :

- Interface avec le front
- Interface avec le S3
- Génération des PDFs



Architecture back

Constats globaux

- Réutilisation de Django pour que l'application front interface puisse communiquer avec l'application back de génération des ouvrages
- Squelette Django respecté
- Génération des PDFs par un module Python
- Ecriture des ouvrages dans le S3
- Utilisation de procrastinate pour lancer les tâches automatisées de génération
- Pas de journaux d'application
- La communication front back se fait sans authentification
- L'accès au stockage objet S3 se fait via un couple ACCESS_KEY_ID et ACCESS_KEY_SECRET



Analyse du back

Le code est globalement de bonne facture

Points positifs

- Code métier concernant la génération des PDFs bien isolé
- Même chose pour l'interface avec Minio (S3)
- Code en majorité explicite
- Identification des actions longues nécessitant une exécution asynchrone, notamment pour la génération des PDF

Points négatifs

- Maintenabilité un peu complexe des URLs de contact pour lancer les actions de génération ou de récupération car codées en « dur » côté application génération PDF et interface front
- Gestion des erreurs pas toujours évidente
- Légère duplication de code présente dans les vues
- Présence de chemins relatifs menant à des scripts et de "magic number/string"
- Présence de fichiers n'étant plus utiles à l'application (ancienne version AHFormatter)



Analyse qualimétrie Sonar

Dans l'ensemble les bonnes pratiques sont respectées avec une mise en œuvre maîtrisée.

Les retours qualimétriques Sonar sont les mêmes que pour l'application interface

Il existe du code factorisable au niveau des vues, mais il s'agit d'une légère partie. La duplication de code reste faible.

Le score de "code smells" est dû au non-usage de constantes, cela représente la totalité des "code smells" jugés critiques.
Il reste également des FIXME dans le code.

Le formatter utilisé Black n'indique pas de modification nécessaire.



Identifier les parties du code pouvant être factorisées, et clarifier la pertinence des FIXME restants dans le code

Tests

Présence de tests automatisés et manuels

- Les tests sont lancés à chaque lancement de pipeline (donc à chaque synchronisation du code)
- Présence de tests d'intégration vérifiant la bonne génération d'un ouvrage à partir d'un document XML réel
- Tests unitaires complets couvrant :
 - tous les différents composants de la génération d'ouvrage (résultat de génération, écriture, nettoyage, etc.)
 - la synchronisation avec le S3
 - les tâches lancées par procrastinate

Aspects non testés :

- Performance (temps d'exécution)
- Couverture



Utiliser plusieurs documents réels pour tester la génération sur différents cas (document correct, mal formé, mauvais type, etc.)

Exécution des tests

Bilan des tests exécutés

- 53 tests réalisés :
 - 50 tests validés
 - 3 tests en erreur

94.34% de réussite
- Erreurs au niveau des tests suivants :
 - Test de compression du document PDF généré
 - Test de la gestion d'erreur de génération d'un document
 - Test de vérification du type du fichier lorsque le document généré est une archive

```
===== short test summary info =====
FAILED unit_tests/test_generator.py::TestGenerator::test_compress - FileExistsError: [WinError 183] Impossible de créer un fichier déjà existant: 'C:\\Users\\jb.aligne\\AppData\\Local\\Temp\\pytes...
FAILED unit_tests/test_views.py::TestPublication::test_generation_failed - zoneinfo._common.ZoneInfoNotFoundError: 'No time zone found with key UTC'
FAILED unit_tests/test_views.py::TestPublication::test_calmar_generation_done - AssertionError: assert 'application/x-zip-compressed' == 'application/zip'
===== 3 failed, 50 passed in 24.91s =====
```

NB : l'installation de pyyaml 5.4.1 posait problème, les tests ont été réalisés avec pyyaml 5.3.1



Traitements batch

Quelques éléments sur les traitements batchs

1 traitement batch principal

- Génération de tous les ouvrages dont les tableaux ont été mis à jour.

Exécution du batch

- Lancé dans la même instance que pour le backend
- Configuration par procrastinate
- Batch lancé quotidiennement, à 00h05



Modélisation des données

Analyse MCD et MDD

Observations :

- La base de données est composée d'une table dédiée à Django pour l'authentification Basic Auth
- Une seconde table est dédiée au stockage des tâches pour la librairie procrastinate



Gestion des traces applicatives

Analyse des logs (configuration, bonnes pratiques, analyse des erreurs remontées)

Configuration

- Tous les logs sortent en sortie erreur
- Pas de log de debug
- Tous les logs au niveau INFO

Bonnes pratiques

- Pas de séparation des types de log
- Pas d'indication sur les étapes en cours, ne contient uniquement les commandes bash exécutées

Analyse des erreurs

- Erreurs sur la tentative d'accès à un fichier XML, pas de vérification de présence du fichier

```
Building tree for file:/tmp/sppnaut/e18dbc95-61bc-451f-af13-5423acc1845f/g4/xml/document.xml using class net.sf.saxon.tinytree.TinyBuilder
Tree built in 33 milliseconds
Tree size: 10648 nodes, 143910 characters, 2851 attributes
Loading net.sf.saxon.event.MessageEmitter
Recoverable error on line 128 of document.xml:
  FODC0005: java.io.FileNotFoundException:
    /tmp/sppnaut/e18dbc95-61bc-451f-af13-5423acc1845f/www/xml/document.xml (No such file or directory)
Using parser com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl$JAXPSAXParser
Building tree for file:/tmp/sppnaut/e18dbc95-61bc-451f-af13-5423acc1845f/g4/illustrations/svg/ing4_0.0.svg using class net.sf.saxon.tinytree.TinyBuilder
Tree built in 10 milliseconds
Tree size: 1217 nodes, 3358 characters, 1145 attributes
Recoverable error on line 482 of ouvrage.xml:
  FODC0005: Document has been marked not available:
    file:/tmp/sppnaut/e18dbc95-61bc-451f-af13-5423acc1845f/www/xml/document.xml
Recoverable error on line 482 of ouvrage.xml:
  FODC0005: Document has been marked not available:
    file:/tmp/sppnaut/e18dbc95-61bc-451f-af13-5423acc1845f/www/xml/document.xml
```

Recommandation

- Mettre en place des logs de type debug afin de faciliter le débogage
- Indiquer de manière explicite à quelle étape est la chaîne de génération PDF

Analyse du Backend

Conclusion générale

- Le code est globalement de bonne qualité.
- Il est lisible et clair malgré l'absence de documentation. La conception paraît également adaptée au besoin actuel de l'application.
- L'utilisation de asyncio afin d'avoir des traitements asynchrones sur les actions longues et potentiellement concurrentielles (génération d'un ouvrage) est cohérente.
- La mise en place d'un sémaphore personnalisé afin de gérer la concurrence a été mis en place. Cela ne pose pas de problème de scalabilité vu qu'il n'y a qu'un utilisateur chargé de lancer la génération.
- Absence de logs explicite sur le déroulement de la génération, seules les commandes bash exécutées sont présentes.



Synthèse des préconisations

Thématiques	Priorité	Effort à engager	Impact perf	acteur	Description
Maintenabilité	Faible	Faible	Faible	Dev	Mise en place de constantes pour éviter la duplication de chaînes
Maintenabilité	Faible	Faible	Faible	Dev	Rajouter des logs de debug
Maintenabilité	Moyenne	Faible	Faible	Dev	Rajouter des logs explicites lors des actions importantes
Test	Faible	Faible	Faible	Dev	Utiliser des fichiers réels pour les tests (fichiers corrects, mal formés, etc.)

1. Contexte et objectifs de l'audit
2. Audit de l'architecture
3. Analyse Performance
4. Analyse du code front
5. Analyse du code back
6. Chaine de fabrication et déploiement
7. Observabilité
8. Analyse des interviews
9. Bilan et synthèse des préconisations
10. SPO en 2024 et après
11. Conclusion

Chaine de fabrication et déploiement

Gestion du développement

Gestion des sources et du build

- Développement sur github et mise à disposition des sources sur Gitlab (<https://gitlab.com/GitShom/spo/sppnaut-spo>)

Le projet suit une architecture *monorepo*. Il est découpé en trois applications distinctes :

- **PDFGenerator** : le serveur backend assurant la génération manuelle ou périodique d'ouvrages.
- **interface** : le serveur permettant de consulter ou lancer la génération d'ouvrages.
- **referentiel-sync** : le démon recopiant un système de fichiers au SHOM dans des buckets S3

Bonne gestion des branches et version.

Notes :

- Absence de tag sur Gitlab
- Absence de changelog

Process de développement des features

Mode de développement par sprint

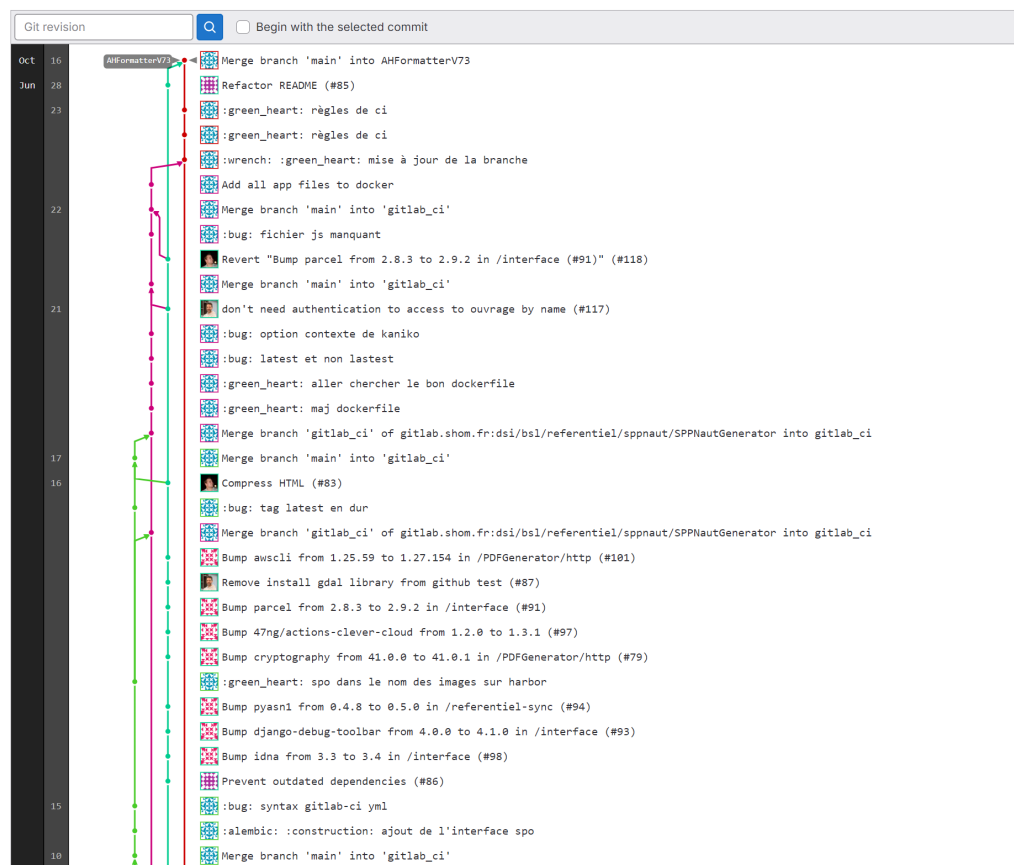
- Développement des fonctionnalités par incréments
- Démo
- Merge vers le Gitlab
- Rebuild des artefacts depuis SHOM
- **Déploiement manuel**

Versionner le code source



Chaine de fabrication et déploiement

Bonne gestion des branches et version sur Git



Chaine de fabrication et déploiement

Gestion du build et automatisations

Intégration continue

- Gitlab-ci
 - Usage de gitlab actions
 - Pipeline non passant actuellement
- Les images sont compilées via un exécuteur kaniko
- Exécution des tests unitaires sur Github mais non reproduit sur Gitlab
- Pas de push de l'image sur le registry
- Pas de procédure de déploiement

DockerFile

- Bonne pratique sécurité. Le service est lancé avec un user non root
- Présence de commentaires inutiles

Docker compose

- Présence de docker compose uniquement pour le dev.



Mettre en place un docker compose pour la production
Utiliser Gitlab-CI pour compiler et mettre l'image dans une registry
Reproduire les tests unitaires sur Gitlab.
Ajouter une step dans Gitlab-CI pour déployer

Analyse CVE des images avec trivy

Les images contiennent de nombreuses vulnérabilités

Analyse de l'image sppnautgenerator-spo

Total: 986 : UNKNOWN: 7, LOW: 546, MEDIUM: 280, HIGH: 141, CRITICAL: 12 (CURL, ghostscript, libarchive13, libdb5.3, libcurl, libgs9, libgs9-common, linux-libc-dev, zlib1g)

Analyse de l'image sppnautinterface-spo

Total: 1060 : UNKNOWN: 7, LOW: 583, MEDIUM: 270, HIGH: 179, CRITICAL: 21 (libaom0, libblas3, libcurl3-gnutls, libcurl4-gnutls-dev, libdb5.3, libjson-c5, liblapack3, libminizip1, libpython3.9-minimal, libpython3.9-stdlib, linux-libc-dev, python3.9, python3.9-minimal, zlib1g)



Mettre à jour les « dockerfile » pour éliminer les CVE

Synthèse des préconisations

Thématiques	Priorité	Effort à engager	Criticité	acteur	Description
Build	faible	faible	faible	ops	Mettre en place un docker compose pour la production
Build	faible	Moyen	faible	ops	Utiliser Gitlab CI pour compiler et mettre l'image dans un registry
Test	Faible	faible	faible	ops	Reproduire les tests unitaires sur Gitlab
Déploiement	Faible	Moyen	faible	ops	Ajouter une step dans Gitlab CI pour déployer
Sécurité	Forte	Moyen	forte	ops	Elimination des CVE

1. Contexte et objectifs de l'audit
2. Audit de l'architecture
3. Analyse Performance
4. Analyse du code front
5. Analyse du code back
6. Chaine de fabrication et déploiement
7. Observabilité
8. Analyse des interviews
9. Bilan et synthèse des préconisations
10. SPO en 2024 et après
11. Conclusion



Opérabilité

Constats globaux

Observations

- **Exploitabilité** : Le système est exploitable à faible coût malgré son faible niveau d'automatisation.
- **Stabilité de la plateforme** : La plateforme ne souffre pas de problèmes particuliers.
- **Incidentologie**: Pas d'incident majeur remonté.



Accès aux journaux

Constats

- Actuellement, seuls les logs de génération des ouvrages sont conservés. Ils sont copiés en même temps que les ouvrages sur Minio (S3).
- Aucune trace d'exécution pour les applications n'est conservée, que ce soit en échec ou en succès, même chose pour les batchs lancés quotidiennement via la librairie procrastinate.
- SPO utilise sentry pour la gestion des journaux, mais il n'existe aujourd'hui pas de serveur dédié.
- On peut constater que des configurations pour les loggers sont présentes dans les fichiers settings des applications.



Utiliser la sortie standard des conteneurs Docker utilisés pour l'exécution de l'application pour créer ces journaux de log
Ou indiquer les fichiers de log de sortie dans les configurations Django et monter les volumes adéquats sur les conteneurs

Supervision applicative et technique

Constats

- SPO est déployé sur une VM qui elle-même est supervisée. Il n'y a cependant pas de « health check » qui permet de vérifier que l'application répond.
- Les traces ne font l'objet d'aucune alerte.
- L'application n'est pas située derrière un reverse proxy. Cela permettrait de mesurer les temps de réponses de l'application.



- Mettre en place à minima un health check
- Mettre en place traefik comme reverse proxy

Synthèse des préconisations

Ou

Thématiques	Priorité	Effort à engager	Sévérité	Qui	Description
Journalisation	Moyenne	Faible	Moyenne	Dev / Infra	Utiliser la sortie standard des conteneurs Docker pour créer ces journaux de log
Journalisation	Moyenne	Faible	Moyenne	Dev / Infra	Indiquer les fichiers de log de sortie et monter les volumes adéquats sur les conteneurs
Supervision	Moyenne	Faible	Moyenne	Infra	Mettre en place un health check
Supervision	Moyenne	Moyenne	Moyenne	Infra	Mettre en place traefik comme reverse proxy

1. Contexte et objectifs de l'audit
2. Audit de l'architecture
3. Analyse Performance
4. Analyse du code front
5. Analyse du code back
6. Chaine de fabrication et déploiement
7. Observabilité
8. Analyse des interviews
9. Bilan et synthèse des préconisations
10. SPO en 2024 et après
11. Conclusion

Analyse des interviews MOA

LES THEMES ABORDES ET LES QUESTIONS POSEES

- Quels sont les exigences en termes de temps de réponses, disponibilité, traçabilité, sécurité ?
- Quels sont les natures de problèmes avec l'application vu de l'utilisateur (page blanche, erreur 503, erreur 500...) ?
- Y'a-t-il eu beaucoup d'anomalies remontées ?
- Quelle est la roadmap prévue ?
- Etes-vous content de l'application SPO dans son ensemble ?

POINTS POSITIFS

- L'application donne satisfaction au métier
- La complexité de l'application n'est pas considérable
- Facilité d'utilisation
- Très peu d'erreurs rencontrées à l'utilisation

POINTS DE DOULEURS

- Très peu de documentation
- Difficultés liées à l'intégration à l'environnement local SHOM (car l'application est conçue pour utiliser le cloud)

AXES D'AMELIORATION

- Améliorer la gestion de la journalisation

ATTENTES / BESOINS

- Avoir des pistes pour la mise en place du SSO
- Préconisations sur la partie Gitlab CI
- Enjeux de sécurité

1. Contexte et objectifs de l'audit
2. Audit de l'architecture
3. Analyse Performance
4. Analyse du code front
5. Analyse du code back
6. Chaine de fabrication et déploiement
7. Observabilité
8. Analyse des interviews
9. Bilan et synthèse des préconisations
10. SPO en 2024 et après
11. Conclusion



Evaluation de SPO : axe architecture

Industrialisation

Ce critère évalue le niveau d'industrialisation de la solution.

Pas industrialisé

L'application repose sur des solutions spécifiques, peu utilisées dans le SI ministériel et mal maîtrisées.

Industrialisé

Les couches basses et middleware de l'application reposent sur des socles techniques ministériel, connus et maîtrisés par les équipes.

Automatisé

Le déploiement de l'application se fait entièrement dans les standards SHOM, ce qui permet une automatisation de la majorité des actions.

Modularité

Ce critère évalue le découpage et l'autonomie des modules du domaine.

Monolithique

L'application est packagée et livrée sous la forme d'un monolithe. Il n'est pas possible de faire évoluer ou livrer qu'un périmètre fonctionnel réduit de l'application.

Modulaire

L'application est structurée sous la forme de modules fonctionnels interdépendants. Livrer un module de façon complètement indépendante n'est pas possible.

Microservices

L'application est structurée sous la forme de modules indépendants. Il est possible de faire évoluer un périmètre fonctionnel réduit de l'application sans impacts pour le reste de l'application ou des modules.

Interopérabilité

Ce critère évalue le niveau de standardisation de la solution et sa capacité à s'interconnecter dans le SI.

Pas interopérable

Le Système n'est pas conçu pour échanger avec des Systèmes Tiers, ou supporter d'autres socles techniques. Une refonte du Système doit être réalisée pour le rendre interopérable.

Potentiellement interopérable

Le Système respecte les standards et des évolutions sont possibles pour permettre au Système d'échanger avec des Systèmes Tiers, ou bien le transposer sur des socles techniques différents.

Interopérable

Le Système respecte les standards et est prévu pour échanger avec des Systèmes Tiers. Il est naturellement compatible avec une grande variété de socles techniques

Exigences non Fonctionnelles

Ce critère évalue le niveau de formalisation des exigences non fonctionnelles et leur niveau de partage entre les acteurs du projet.

Non exprimées

Les exigences fonctionnelles ne sont pas formalisées d'une manière claire et accessibles pour l'ensemble des parties prenantes.

Formalisées

Les exigences fonctionnelles sont formalisées dans les documents de référence liés au projet.

Partagées

Les exigences fonctionnelles formalisées sont le résultat d'échanges entre les parties prenantes, prenant en compte les contraintes métier, techniques ainsi que les standards de l'Ademe.

Sécurité

Ce critère évalue le niveau de sécurité du SI à travers les moyens mis en œuvre pour prévenir, détecter et traiter les vulnérabilités.

Non sécurisé

Les moyens de sécurisation mis en œuvre sont inexistantes ou obsolètes ou aucun moyen de détection des incidents de sécurité n'est mis en place, soit des incidents de sécurité graves ont été détectés.

Sécurisé partiellement

Les moyens de sécurisation mis en œuvre sont efficaces mais incomplets. Les incidents sont détectés et des mesures correctives sont prises. Quelques incidents de sécurité de faible gravité ont pu avoir lieu

Sécurisé

Les moyens de sécurisation mis en œuvre sont efficaces. Des moyens de détections des incidents sont mis en place et des mesures correctives sont réalisées en cas de détection.

Obsolescence

Ce critère évalue le niveau d'obsolescence de la solution.

Obsolète

Le système est obsolète et nécessite une mise à jour urgente pour assurer son fonctionnement, sa sécurité et sa pérennité.

Obsolète partiellement

La pile technologique est variable. Certaines bibliothèques /framework sont à jour alors que d'autres n'ont pas été mis à jour depuis plusieurs années.

A jour

L'application est à jour. Aucune obsolescence n'est observée

Evaluation de SPO : axe logiciel

Qualité du code source

Ce critère évalue la qualité du code à travers des outils de qualimétrie accompagné d'une revue manuelle par

Mauvaise

Indicateurs de qualimétrie mauvais 1/5 à 2/5.
Absence de standard de développement

Moyenne

Indicateurs de qualimétrie 3/5 à 4/5

Bonne

Indicateurs de qualimétrie 5/5
Code de bonne facture

Modèle de Données

Ce critère évalue le modèle de données de l'application, sa complexité, son niveau de rationalisation, la présence d'index...

Non adapté

Le modèle de données n'est pas adapté au fonctionnement de l'application, car trop complexe, directement repris d'autres sources de données ou bien non standardisé.

Rationalisé

Le modèle de données forme un tout rationalisé, adapté aux besoins de l'application.

Optimisé

Le modèle de données a été optimisé pour répondre aux besoins de l'application.

Documentation

Ce critère évalue la qualité de la documentation et la qualité de capitalisation (nomenclatures, templates...)

Pas de documentation

Il n'y aucune démarche pour documenter et capitaliser le savoir.
De la documentation a pu être produite mais elle n'est pas accessible ou utilisable.

Documentation incomplète

Une démarche de capitalisation est mise en place et la documentation est accessible. De la documentation est produite mais pas mise à jour. La documentation existante est donc partiellement utilisable.

Documentation à jour

Une démarche de capitalisation est mise en place, la documentation produite est accessible et régulièrement mise à jour.

Tests unitaires

Ce critère évalue le niveau de couverture des tests, les scénarios de tests et la capitalisation réalisée.

Non représentatifs

Absence de tests, ou bien tests réalisés dans des conditions trop éloignées des conditions réelles.

Tests partiels

Quelques tests sont présents mais avec une trop faible couverture.

Tests complets

Chaque aspect de la solution est testé, une capitalisation est réalisée pour observer l'évolution des résultats de test.

CI/CD

Ce critère la chaîne de construction et de déploiement de l'applicatif

Manuelle

Aucune politique de gestion de configuration
Procédure de construction des livrables laborieuses
Procédure d'installation manuelle

Semi industrialisé

Politique de gestion de configuration
Intégration continue.
Déploiement semi automatisé

Full automatisé

Usine logicielle complète CI/CD.
Déploiement complet à chaud

Requêtes

Ce critère évalue les requêtes de l'application à sa base de données.

Non adaptées

Les requêtes ne sont pas standardisées, récupèrent trop de données ou bien sont trop nombreuses pour permettre un fonctionnement fluide.

Rationalisées

Les requêtes sont rationalisées, récupèrent les données nécessaires en gâchant un minimum de ressources.

Optimisées

Les requêtes sont optimisées pour répondre aux besoins de l'application.



Evaluation de SPO: axe infra

SLA

Ce critère évalue le niveau de formalisation des exigences non fonctionnelles et leur niveau de partage entre les acteurs du

Non exprimés

Les SLA ne sont pas formalisés d'une manière claire et accessibles pour l'ensemble des parties prenantes.

Formalisés

Les SLA sont formalisés dans les documents de référence liés au projet.

Partagés

Les SLA formalisés sont le résultat d'échanges entre les parties prenantes, prenant en compte les contraintes métier, techniques ainsi que les standard de l'Ademe.

Ressources et Allocation

Ce critère évalue la consommation des ressources allouées par les différents modules de l'application.

Non mesuré

Les outils de surveillance de la consommation ne permettent qu'une mesure parcellaire de la consommation des ressources, créant un flou sur le réel besoin en termes dimensionnement.

Surveillé

Les outils de surveillance de la consommation donnent une vision précise de la consommation, permettant de relier la consommation aux événements et d'adapter le dimensionnement.

Adapté

La consommation de ressources est surveillée, et permet de vérifier que le dimensionnement est adapté aux besoins de l'application, et d'extrapoler sur les besoins en cas de pic/de montée en charge.

Dimensionnement

Ce critère évalue l'adéquation entre le dimensionnement de l'application, les recommandations éditeurs et la volumétrie.

Inadapté

Le dimensionnement est sous-estimé/surestimé, sans lien avec les besoins réel de l'application.

Adapté

Le dimensionnement est adapté au fonctionnement de l'application, avec suffisamment de marge pour permettre un fonctionnement fluide en toutes circonstances.

Evolutif

Le dimensionnement peut être adapté au plus près des besoins réels de l'application, en pic comme en creux de charge.

Temps de Réponse

Ce critère évalue les temps de réponse de l'application, vus par l'utilisateur.

Bloquant

Un nombre important de fonctionnalités présentent des temps de réponse longs, ce qui impacte négativement l'expérience utilisateur.

Acceptable

La majorité des temps de réponse sont globalement corrects du point de vue utilisateur. Seules les requêtes les plus gourmandes peuvent créer des temps d'attente pour l'utilisateur.

Fluide

La solution propose une navigation entièrement fluide, sans délais de réponse perceptible pour l'utilisateur.

Evaluation de SPO : axe Opérabilité

Incidentologie

Ce critère évalue le nombre et la fréquence des incidents, ainsi que les procédures mises en place pour les résoudre.

Incidents non surveillés

Les incidents sont observés directement par les utilisateurs, sans remontée préalable par d'autres canaux.

Incidents détectés

La détection des incidents est assurée, leur prise en charge rapide et les causes des incidents sont identifiées par les parties prenantes

Résolution procédurée

Chaque nouvel incident est capitalisé, permettant une résolution procédurée. Les causes d'incidents sont identifiées, et l'occurrence d'incidents similaires induit une priorisation de leur traitement.

Robustesse et résilience

Ce critère évalue le niveau de stabilité de la solution, sa capacité à rester dans le cadre des SLA à tout moment.

Instabilité permanente

Même en fonctionnement nominal, l'application est instable et ne répond pas aux SLA.

Instable lors des opérations admin

En mode nominal, l'application est stable. Certaines opérations ponctuelles (Sauvegarde, redémarrage, mise à jour) peuvent impacter la stabilité, et doivent donc être réalisées à froid.

Stabilité permanente

L'application est stable en permanence. Les opérations ponctuelles peuvent être réalisées à chaud sans conséquence pour les utilisateurs.

Disponibilité

Ce critère évalue le niveau de disponibilité de l'application durant les 12 derniers mois.

Taux d'indisponibilité forte

L'application a été indisponible régulièrement ou a connue des pannes d plusieurs heures ou n'a pas respecté ses exigences .

Haute dispo

L'application a connu quelques indisponibilités, sans toutefois dépasser quelques minutes.

Très haute disponibilité

L'application n'a connu aucune indispo ou de quelques secondes sur un an.

Scalabilité

Capacité à supporter les montées en charges, en termes d'utilisateurs, de volumétrie...

Non scalable

Le Système est dans l'impossibilité de monter en charge. Une refonte importante doit être réalisée pour permettre au Système de monter en charge.

Potentiellement Scalable

Le Système pourrait monter en charge sans refonte mais au prix de quelques adaptations comme l'attribution de plus de ressources.

Scalable

Le Système peut absorber une montée en charge sans impacts.

Performances

Ce critère évalue les performances de l'application

Lenteurs sur toute l'application

Les requêtes ne sont pas standardisées, récupèrent trop de données ou bien sont trop nombreuses pour permettre un fonctionnement fluide.

Quelques pages lentes

Les temps de réponse sont globalement satisfaisant sauf pour quelques pages qui ne sont pas optimisées.

Fluide

Les temps de réponses sont inférieurs à la seconde
Les requêtes sont optimisées pour répondre aux besoins de l'application.

Exploitabilité

Ce critère évalue les process et outillage mis en place pour maintenir les outils dans état opérationnel stable.

Exploitation non industrialisée

L'exploitation du système repose sur des tâches manuelles. Les pratiques d'exploitation ne sont pas industrialisées : pas de rationalisation des process, peu d'outillage.

Exploitation industrialisée

Les pratiques d'exploitation sont industrialisées : rationalisation des process, outillage adapté, documentation existante et à jour. Beaucoup de tâches manuelles.

Exploitation automatisée

Le Système est exploitable à faible coûts grâce à un fort niveau d'automatisation. Les pratiques d'exploitation sont industrialisées : rationalisation des process et outils, documentation existante et à jour.



1. Contexte et objectifs de l'audit
2. Audit de l'architecture
3. Analyse Performance
4. Analyse du code front
5. Analyse du code back
6. Chaine de fabrication et déploiement
7. Observabilité
8. Analyse des interviews
9. Bilan et synthèse des préconisations
10. SPO en 2024 et après
11. Conclusion



L'architecture actuelle pourrait-elle tenir si elle est ouverte sur internet ? Une autre architecture serait-elle plus adaptée ?

L'architecture actuelle paraît adaptée, moyennant quelques évolutions :

- Renforcer la sécurité, avec la mise en place d'un reverse proxy, la mise en place de HTTPS, la vérification des fichiers importés.
- Dédier une instance front back pour la génération PDF.



Peut-on raccorder le front à l'outil IAM SHOM Lemon LDAP ?

OUI : le raccordement est simple

Contexte

Afin d'unifier les moyens d'authentications sur toutes les applications utilisées par le SHOM, l'implémentation de la gestion d'authentification SSO pour SPO est possible.

Coté SSO

LemonLDAP propose plusieurs protocoles mis à disposition (CAS, SAML, OIDC) pour s'accoster.

En dehors de ces protocoles, il y a toujours la possibilité de s'interfacer via basic auth. Lemon LDAP propose un Connector pour Django <https://github.com/rcsilver/django-lemondap> dans ce sens.

Coté Django

Il existe des librairies pour s'accoster à LemonLDAP via les protocoles CAS, SAML, OIDC.



Quid de la base de données en cas de raccordement à l'outil SSO ?

La base de données pourrait être enlevée

La base de données ne sert aujourd'hui que pour 2 choses : l'authentification locale et le stockage des tâches déléguées à procrastinate.

Cette dernière librairie a besoin d'une base PostgreSQL pour stocker les tâches.

Une réflexion pourrait donc avoir lieu pour passer par une autre librairie Python pour avoir ces tâches en mémoire ou autre moyen (par exemple crontab) pour éviter d'avoir une base de données PostgreSQL dédiée à ce simple usage.



Conclusion

→ 7 mars 2024

Conclusion de l'audit SPO

L'application SPO **repose sur de bonnes fondations**. Le code dans sa globalité est de bonne facture et l'architecture est évolutive. Les indicateurs qualimétriques sont au vert.

L'application répond positivement aux besoins métier.

Le choix de Django est cohérent. L'architecture aurait pu toutefois être plus simple avec un seul module intégrant le front et le back.

L'analyse du code montre quelques points à améliorer. L'effort pour corriger les recommandations n'est pas énorme.

Un effort doit être fait pour améliorer la documentation et l'industrialisation à travers l'usine logicielle Gitlab.



Conclusion de l'audit SPO

Quelles sont les préconisations au niveau Logiciel et/ou infra permettant d'améliorer la sécurité, la tierce maintenance ?

Préconisations à court terme

- Remédier aux problématiques CVE des images.
- Mettre en place la CI sur Gitlab
- Brancher le SSO
- Enrichir la documentation

Préconisations en cas d'évolution

- Traiter les autres recommandations sur l'analyse front back



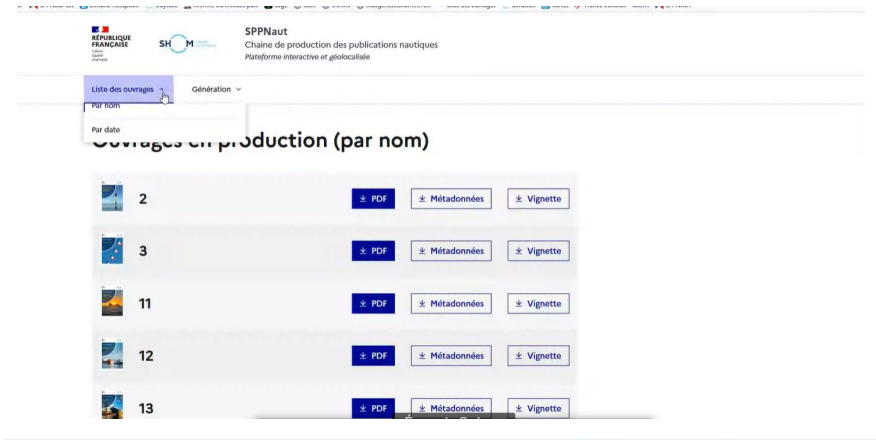
Merci.

→ 7 mars 2024

onepoint.
beyond the obvious

Annexe

Liste des écrans SPO



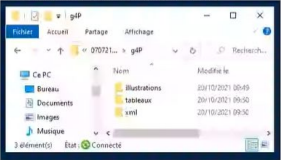
Génération d'ouvrage téléversé

Je souhaite visualiser le rendu final d'une publication en PDF à partir de l'ensemble des sources de ma publication.

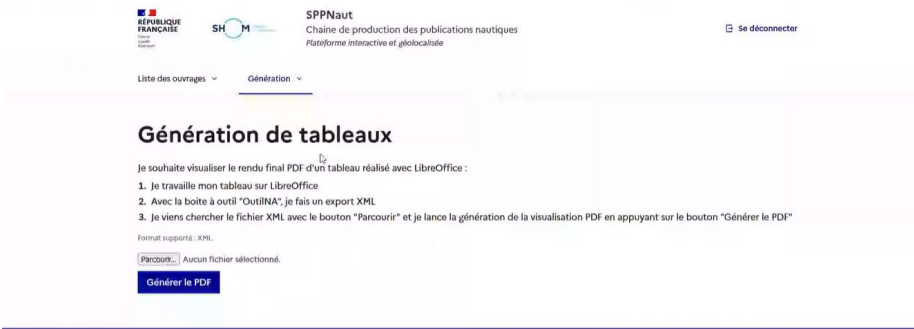
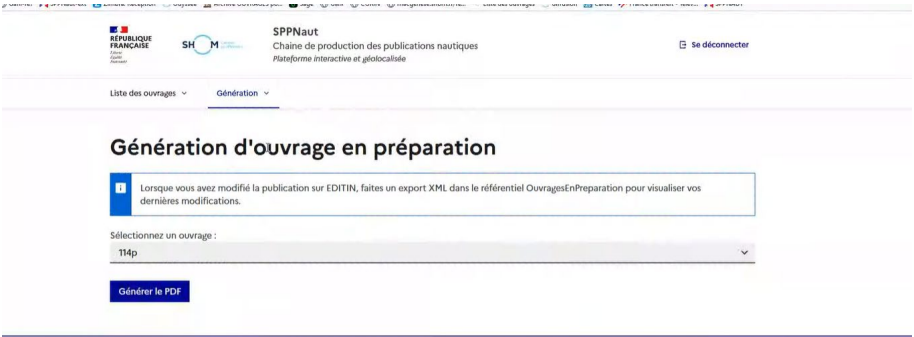
Attention, le dossier téléversé doit contenir :

- 1. Les illustrations dans les sous dossiers //illustration/eps, //illustration/svg et //illustration/pdf (si nécessaire)
- 2. Les tableaux au format XML dans //tableaux
- 3. Le contenu de la publication dans le fichier //xml/document.xml

> Par exemple, téléverser le dossier nommé 'g4P'



Lorsque vous avez modifié la publication sur EDITIN, n'oubliez pas de faire un export XML pour visualiser vos dernières modifications.



Annexe

Stockage des PDF sur Minio

