PREMIER MINISTRE
Liberté
Égalité
Fraternité

Secrétariat général du Gouvernement

Direction des services
administratifs et financiers

Annexe 6 au cahier des clauses techniques particulières

-

Guide d'architecture DILA

Développement et architecture applicative

Version 1.1.6

Table des matières

Suivi des versions 4
1 Introduction 5
2 Modularité des développements 5
2.1 Privilégier une architecture en service 5
2.2 Exemples de cas d'usage 6
2.2.1 Assurer une scalabilité ciblée à un périmètre fonctionnel 6
2.2.2 Isolation d'une partie critique de l'application 6
2.2.3 Isolation de la partie administration 6
2.3 Responsabilité du traitement et des données 6
2.4 Adapter la notion de service / micro-service au contexte métier 7
3 Mise en place d'une API 8
3.1 Description des ressources et verbes associés 8
3.2 Documentation de l'API 9
3.3 Gestion des erreurs 9
3.4 Sécurisation de l'API 9
3.5 Appropriation de l'API 10
4 Aspects sécurité 10
4.1 Règles de portée générale 10
4.2 Dispositions relatives au suivi de la sécurité au sein des projets 11
4.3 Instances et processus de validation 12
4.4 Règles relatives au développement sécurisé d'applications 12
4.5 Règles relatives aux interfaces web 13
4.6 Règles relatives aux services de courriels 13
4.7 Configuration TLS 13
4.8 Gestion des mots de passe 13
4.8.1 Complexité des mots de passe 14
4.8.2 Stockage du mot de passe 14
4.8.3 Réinitialisation du mot de passe 14
4.8.4 Traçabilité des accès 15
5 Attendu sur le code produit 15

5.1 Qualité du code 15
5.2 Sécurité du code 15
5.3 Respect des différents référentiels 15
6 Pile technologique 16
6.1 Principes généraux appliqués à la DILA 16
6.2 Focus sur le développement Web 17
7 Références 17
8 Annexe : Mesures de développement sécurisées 18

Suivi des versions

Le tableau ci-dessous décrit l'historique du document. Chaque modification doit y être consignée et conduit à une montée de version du document.

Les versions sont exprimées sur 3 digits :

- le premier digit traduit l'ajout d'une nouvelle règle à prendre en compte ou d'évolutions plus conséquentes ;
- le second digit exprime une modification mineure comme une précision apportée à une règle, un exemple…
- le troisième digit correspond à des modifications de formes ou des ajouts qui ne changent pas la portée et l'interprétation du document.

Version	*Date*	*Auteur*	*Commentaires*
0.1.0	20/12/2021	Laurent DENIS	Initialisation du document
0.2.0	05/01/2022	Laurent DENIS	Ajout d'une mention sur l'obligation d'utiliser les guides de sécurité ANSSI
0.2.1	06/01/2022	Laurent DENIS	Ajout du tableau de suivi des versions Ajout de précisions sur la partie sécurité
0.3	10/01/2022	Laurent DENIS	Ajout des sections suivantes : • dispositions relatives au suivi de la sécurité au sein des projets • règles de développement sécurisé
0.3.1	11/01/2022	Laurent DENIS	Mise en forme
1.0.0	11/01/2022	Laurent DENIS	Version diffusable
1.1.0	12/01/2022	Laurent DENIS	Ajout d'élément de validation sur la sécurité des architectures
1.1.1	21/03/2022	Laurent DENIS	Ajout d'une section sur la prise en main des API
1.1.2	03/03/2022	Laurent DENIS	Apport de précision sur l'utilisation de microservices
1.1.3	11/07/2022	Laurent DENIS	Utilisation de PISTE pour les clients des API internes
1.1.4	21/07/2022	Laurent DENIS	Ajouts de points concernant la sécurité • configuration TLS • gestion des mots de passe
1.1.5	06/12/2022	Laurent DENIS	Intégration du guide CNIL de sécurisation des API dans le cadre de mise à disposition de données personnelles. Ajout de la conformité au « Référentiel général d'écoconception de services numériques »
1.1.6	05/01/2023	Laurent DENIS	Le code livré par un prestataire doit faire l'objet d'un audit automatique de sécurité

1 Introduction

Ce document décrit les grandes orientations à prendre en compte dans le cadre du développement d'une application. Ces dernières permettent d'une part d'assurer la cohérence des développements réalisés et d'autre part de faciliter les évolutions et la maintenance des applications concernées.

Les règles données dans ce document complètent celles mentionnées dans le guide d'architecture DILA (1) édité par le DIS et focalisé sur la partie infrastructure et cloud.

La majorité de ces recommandations restent très générales et doivent faire l'objet d'une déclinaison technique au sein de chaque projet.

Ce document s'applique à l'ensemble des développements réalisé ou piloté par la DILA. Dans le cas où les développements sont réalisés dans le cadre d'un marché, le titulaire aura au préalable détaillé la mise en application de ce document à travers la mise en place d'un « Plan Assurance Qualité » (PAS)[1]. Il s'inscrira aussi dans les exigences de sécurité ANSSI. Ces dernières sont annexées au marché sous forme d'une annexe regroupant toutes les clauses de sécurité demandée par l'ANSSI.

2 Modularité des développements

Les applications monoblocs sont difficiles à maintenir et à faire évoluer. En raison de leur taille, elles sont aussi souvent complexes et difficiles à appréhender par le développeur. C'est pourquoi la DILA souhaite privilégier une approche modulaire qui permet de réduire cette complexité et faciliter une approche collaborative dans les développements.

2.1 Privilégier une architecture en service

L'approche retenue pour mettre en place la modularité des développements repose sur une architecture en service (ou micro service selon les projets).

La granularité retenue pour le découpage en services doit prendre en compte les axes suivants :

- *Simplicité* : le service doit couvrir un *ensemble fonctionnel cohérent*, participant ainsi à une démarche d'urbanisation métier ;
- *Isolation* : dans l'idéal le dysfonctionnement d'un service ne doit pas être bloquant et entraîner un dysfonctionnement total de l'application. Cette dernière doit pouvoir continuer à fonctionner en mode dégradé le temps de palier le problème. Des mécanismes ou des procédures de reprise doivent alors permettre de revenir à un mode nominal ;
- *Indépendance* : un service doit garantir un certain niveau d'indépendance et englober tout ce qui est nécessaire à son bon fonctionnement. Ceci implique d'avoir des briques techniques qui lui sont dédiées. Cette approche permet ainsi d'augmenter la résilience globale de l'application en diminuant les points de contention (Single Point Of Failure) ;

[1] Lorsqu'un PAS est demandé, les soumissionnaires devront renseigner le cadre de réponse PAS qui est annexé au marché.

- *Scalabilité* : un service doit permettre de répondre à un pic de charge à travers des mécanismes de scalabilité limité au périmètre fonctionnel concerné. *Ce point implique aussi une approche sans état (stateless).*

Durant la phase de conception, il conviendra de chalenger les différentes solutions à minima selon les critères ci-dessus.

2.2 Exemples de cas d'usage

Ce paragraphe a pour objectif de mettre en avant quelques cas d'usage qui peuvent guider le découpage d'une application et il y associe des patterns d'architecture applicative à même de répondre aux besoins induits. Ces cas d'usages ne sont pas exhaustifs, ils ont uniquement vocation à illustrer les principes décrits dans la section 2.1.

2.2.1 Assurer une scalabilité ciblée à un périmètre fonctionnel

Toutes les fonctionnalités d'une application ne sont pas sollicitées au même niveau sur une période définie. Par exemple le temps de traitement d'un dossier peut impliquer des traitements de fond consommateurs de ressources alors que la constitution de ce dernier peut être faite beaucoup plus rapidement. Des approches de type « producteur / consommateur » permettent par exemple d'adapter le nombre de ressources disponibles en fonction des besoins du moment.

2.2.2 Isolation d'une partie critique de l'application

En raison des traitements effectués certaines parties d'une application doivent bénéficier de conditions particulières (manipulation de données sensibles, besoin d'une qualité de service particulière (SLA…). Un découpage applicatif adapté peut fournir la qualité de service nécessaire au périmètre métier le requérant sans impacter les exigences sur le reste de l'application (coût, localisation, astreintes…).

2.2.3 Isolation de la partie administration

Sur un principe similaire à celui présenté section 2.2.2, afin de pouvoir mettre en place des mesures de sécurité différentiées, l'interface d'administration sera isolée de l'application elle-même. Pour cela, il est possible d'opter pour l'une des deux approches suivantes :

- mettre en place une application d'administration indépendante de l'application elle-même
  (deux fichiers « war » par exemple dans le cas de développement java) ;
- accéder à l'application et à l'administration à travers deux sous-domaines différents.

2.3 Responsabilité du traitement et des données

Comme mentionné à la section 2.1, le découpage applicatif doit garantir un niveau d'indépendance suffisant. Les données et leur gestion sont généralement un entrant important pour déterminer le bon niveau de granularité d'un module ou d'un service. Dans ce cadre, les règles suivantes doivent être prises en compte :

- *une donnée n'est stockée qu'à un seul endroit.* Pour des raisons de performance il peut être envisageable de mettre en place un cache, mais ce dernier ne doit pas impacter la cohérence fonctionnelle des données ;

- un module / service à l'entière responsabilité de la cohérence globale des données qu'il gère. Avant de stocker ces dernières, il doit s'assurer que l'ensemble des règles métiers sont respectées. Le cas échéant, il doit lever une erreur ou mettre la donnée en cohérence avec les contraintes fonctionnelles ;
- une donnée entrante est toujours considérée comme non sûre. Le module / service doit obligatoirement vérifier le format de cette dernière avant de l'intégrer (pattern, plage de valeur…). On prendra aussi soin de se prémunir d'injection de code en « échappant » ou en « encodant » les caractères interprétables lors des traitements réalisés.

2.4 Adapter la notion de service / micro-service au contexte métier

La notion de service et microservices peut différer d'une personne à une autre. Cette section a pour objectif de mettre en avant les conséquences d'un découpage fin des services et ainsi de donner des éléments concrets permettant de proposer des architectures ou la complexité reste en adéquation avec les bénéfices obtenus notamment en termes de scalabilité et d'indépendance des équipes amenés à travailler sur le sujet.

Si le découpage en microservices implique une forte interdépendance, cela va se traduire par une communication accrue entre les différents services et potentiellement par des latences dues aux protocoles de communication de type REST. Une telle situation ralentit de facto l'ensemble du système.

La mise en place d'un microservices implique qu'il ait une base de données qui lui est propre. Certains traitements métiers nécessitent la mise en place d'une transaction qui implique deux microservices. Même si les données des microservices sont disjointes, il faut être à même de gérer la transaction au niveau de l'appelant. Bien qu'il existe des patterns classiques pour ce type de besoin (pattern saga[2], Two-phase commit[3]…), cette option reste complexe à mettre en œuvre et à maintenir dans le temps.

Multiplier le nombre de service implique d'augmenter le nombre d'application à déployer et à intégrer entre elles. Là encore, il est important de trouver un bon équilibre entre les avantages (application plus simple, cycle de vie dissociée, scalabilité…), la charge de travail complémentaire que cela peut apporter et le nombre de VM à déployer[4].

La mise en place d'une application constituée de beaucoup de microservices implique d'être en mesure de corréler les différents logs pour restituer un comportement utilisateur ou pour débuguer. Ce point est d'autant plus structurant si les microservices sont fortement couplés.

Tous ces éléments conduisent à mener une réelle réflexion sur le découpage en services afin de mettre en balance les avantages et les contraintes apportées. D'une manière générale, on mettra en place des services de petites tailles mais en prenant comme ligne directrice de *minimiser les interdépendances*. L'idée générale serait d'avoir un service qui *rassemble les principaux éléments fonctionnels présentant une forte interdépendance* (quitte à ce que ce dernier s'approche du monolithe) et des *services satellites de plus petite taille à même de remplir leur rôle sans appels vers d'autres services*. Cependant, il reste primordial de garder une

[2] https://microservices.io/patterns/data/saga.html

[3] https://en.wikipedia.org/wiki/Two-phase_commit_protocol

[4] Pour le moment, les containers ne sont pas autorisés en production. Augmenter le nombre de service passe donc nécessairement par une augmentation du nombre de VM.

architecture modulaire dans chaque service afin *d'être en mesure de le découper facilement dans un second temps*[5].

3 Mise en place d'une API

Le mode d'implémentation privilégié pour la réalisation de service est la mise en œuvre de services REST. Ce type de mécanisme est très largement décrit dans la littérature qui propose de nombreuses approches pour répondre aux différentes problématiques qui peuvent apparaître lors des développements. Dans ce chapitre, nous nous focaliserons uniquement sur les points saillants de la conception de l'API ou les points qui ont été remontés par les projets.

3.1 Description des ressources et verbes associés

Les URLs permettant d'utiliser une API REST doivent être suffisamment significatives pour que les développeurs n'aient pas à consulter systématiquement la documentation. En lisant l'URL, il doit être possible de comprendre immédiatement de quelles ressources il est question. Le verbe http associé donne le type d'opération qui s'applique.

- le nom d'une ressource est décrit par un nom commun (clients, adresses…). Il convient de ne pas utiliser de verbe ;
- pour rester cohérent, il est préconisé d'utiliser systématiquement le nom de la ressource au pluriel (le nom désigne alors la collection) ;
- les URI sont écrites avec la convention « snake_case » (mots en minuscules séparés par le caractère « _ »). Cette notation est très largement utilisée par les principaux acteurs du web (Google, Facebook, tweeter, GitHub…) ;
- le verbe http décrit l'action réalisée : « *GET* : lecture », , « *POST* : création », « *PUT* : modification / création », « *PATCH* : modification », « *DELETE* : suppression » ;
- le visionnement de l'API se fait dans l'URL. La mention de la version doit être explicite.

Sur la base de ces recommandations, voici quelques exemples d'URL :

- get /v1/client/5 : retourne un objet décrivant le client « n°5 »
- patch /V1/client/34/adresse/ {"numero" : "57", "rue" :"Varenne", "code_postal" : "75007", "ville" :"Paris"} : modifie l'adresse passée dans le corps de la requête pour le client 34
- delete /v2/client/12/adresse/2 : supprime l'adresse ayant l'identifiant « 2 » pour le client « 12 »

Pour faciliter si besoin la mise en place d'un WAF, l'API ne doit pas proposer dans son URL les motsclefs qui pourrait déclencher des règles de filtrage. Sans être exhaustif, on ne doit donc pas retrouver les mots clefs : date, select, for, between…

[5] https://martinfowler.com/bliki/MonolithFirst.html

3.2 Documentation de l'API

Toute API doit obligatoirement être documenté avec OpenAPI[6]. Dans la mesure du possible, il est souhaitable de donner dans la documentation des exemples que l'on puisse directement dérouler sur un environnement de démonstration de l'API (particulièrement utile pour les API à destination de l'extérieur).

3.3 Gestion des erreurs

L'API doit s'appuyer sur les codes HTTP afin que le client puisse statuer sur la validité de sa requête. Les codes HTTP renvoyés doivent être le plus précis possible :

- famille 2xx : la requête a été exécutée avec succès ;
- famille 3xx : redirection ;
- famille 4xx : problème au niveau de la requête ;
- famille 5xx : problème au niveau du serveur.

En parallèle des codes d'erreur HTTP, en cas de problème, l'API doit retourner un objet « erreur » permettant de mieux diagnostiquer le problème. Ce type d'objet contient à minima un code d'erreur permettant de classifier cette dernière (erreur technique, fonctionnelle…), un libellé court compréhensible par l'utilisateur et une description plus complète de l'erreur là aussi compréhensible par l'utilisateur.

3.4 Sécurisation de l'API

Toute API doit utiliser des mécanismes de sécurité permettant de limiter leur utilisation aux clients légitimes. Dans le cas d'API ouvertes, l'obtention des droits d'utilisation de l'API peut se faire simplement à travers un formulaire d'enregistrement (fonctionnalité offerte par PISTE, utilisation de DataPass[7] mis à disposition par la DiNum).

La sécurisation de l'API (authentification ou autorisation) doit être réalisée sur la base d'une méthode basée sur OAuth2 (OpenId Connect[8], JWT[9]…).

Une API publique ouverte sur l'extérieure, doit être positionnée derrière une brique « API Management » afin de gérer les aspects sécurité, quota et gestion des comptes. La brique retenue à la DILA est le produit PISTE[10] de l'AIFE[11]. L'utilisation de cette même API par une application interne DILA doit aussi passer par « l'API Management « qui gère les aspects sécurité (quota, gestion du token…). « L'API management » doit ainsi permettre de ne pas exposer des informations qui permettraient de contourner PISTE (le code des clients de type « Single Page Application » écrit en javascript est accessible à tous).

[6] OpenAPI : https://swagger.io/specification/
[7] DataPass : https://datapass.api.gouv.fr/
[8] OpenId Connect : https://openid.net/connect/
[9] JWT (JSON Web Tokens) : https://jwt.io/
[10] PISTE : https://developer.aife.economie.gouv.fr/
[11] AIFE : Agence pour l'Informatique Financière de l'Etat

Sur le périmètre des données personnelles, la CNIL met à disposition un guide des bonnes pratiques[12] à mettre en œuvre. Ces dernières sont à prendre en compte dans la sécurisation de ces dernières. En particulier, il faudra veiller au cloisonnement des API servant des données personnelles, la garantie des secrets d'authentification ainsi qu'à la possibilité de révoquer rapidement des accès compromis. Afin de pouvoir répondre aux impératifs du RGPD, les différents accès aux API doivent être journalisés en conservant l'auteur individuellement identifié, l'horodatage, la nature de l'opération ainsi que la référence aux données accédées.

3.5 Appropriation de l'API

Au-delà des aspects documentation, l'appropriation de l'API par un développeur passe aussi par la possibilité qu'on lui offre de tester cette dernière, soit directement à travers son navigateur ou via « curl ». La mise à disposition de ce type de fonctionnalité sur la page de documentation de l'API est souhaitable. Si l'API à vocation à être largement partagée, il est indispensable de proposer des méthodes afin que le développeur puisse l'exploiter via un portail web.

4 Aspects sécurité

La conception et l'exploitation du système d'information doivent se faire conformément à l'état de l'art en matière de sécurité des systèmes d'information. Il doit se reporter systématiquement aux guides de recommandations de l'ANSSI pour être à jour de l'état de l'art en la matière.

Les développements réalisés doivent à minima tenir compte des règles de sécurité déclinées dans cette section.

4.1 Règles de portée générale

Ces règles couvrent les aspects généraux concernant la sécurité. Ces dernières doivent être prises en compte par le projet.

- l'architecture projet doit être conforme à la PSSIe DILA ;
- l'équipe de développement doit prendre en compte les vulnérabilités les plus courantes en s'assurant que leur code n'est pas touché par le Top 10 de OWASP [13] (https://owasp.org/wwwproject-top-ten) ;
- il est souhaitable que l'équipe projet s'organise selon une approche Dev / Sec / Ops afin de produire du code « sécurisé by design », notamment en incluant les mécanismes de protection et de vérification à chaque étape du développement ;
- l'architecture choisie et l'outillage utilisé doivent permettre de facilement mettre en place une politique de patch management. Ce point couvre à la fois les briques applicatives et les bibliothèques / librairies de développement ;

[12] https://www.cnil.fr/sites/default/files/atoms/files/consultation_publique_-_projet-derecommandation_interfaces-programmation-applicatives-api.pdf

[13] OWASP : Open Web Application Security Project

- *l'équipe projet doit être à tout moment en mesure de donner une liste exhaustive des versions des briques applicatives et des bibliothèques utilisées*. Cette liste doit être construite automatiquement via les outils de construction et ne doit surtout pas devoir être maintenue à la main ;
- la définition des versions des bibliothèques utilisées doit être centralisée dans un fichier unique qui adresse tous les modules du projet. Par exemple, dans le cas de « maven », il est possible de déclarer les versions des bibliothèques utilisées dans un fichier « pom.xml parent » ;
- les versions des bibliothèques utilisées doivent toutes être explicitement déclarées. Les mécanismes permettant d'utiliser la dernière version disponible sans avoir à la mentionner sont proscrits ;
- les bibliothèques utilisées doivent être en version stable (pas de snapshot), faire partie du processus de veille sécurité et rentrer dans un processus de MCS;
- dans la mesure du possible, il est demandé d'appliquer les mêmes règles pour les briques applicatives à travers une approche « infra as code ».
- le développement au sein d'un projet de brique ou de code liée à la sécurité est interdit. Il est demandé de se reposer sur des bibliothèques éprouvées et développées par des spécialistes ; • le code mort dans une application est proscrit ;
- toute donnée entrante dans un système doit systématiquement être vérifiée (format, plage de valeur…). Afin de se prémunir d'attaque de type injection, il est demandé de systématiquement « échapper » ou « encoder » les chaines de caractères ;
- le recours aux API doit systématiquement être sécurisé à minima via l'utilisation d'un jeton (cf. section 3.4 Sécurisation de l'API) ;
- les pièces jointes téléversées par les usagers / utilisateurs doivent faire l'objet d'une vérification antivirale ;
- les environnements de développement et de recette ne doivent pas contenir de données nominatives ou confidentielles issues de la production. S'il y a un réel besoin d'importer de telles données, ces dernières doivent être entièrement anonymisées ;

4.2 Dispositions relatives au suivi de la sécurité au sein des projets

- l'équipe projet doit réaliser une veille sur les composants du projet afin de détecter les failles qui pourraient conduire à des vulnérabilités applicatives et sur leur qualification. La veille doit aussi permettre d'évaluer les impacts sur l'application ;
- l'équipe projet doit réaliser une veille sur l'obsolescence des composantes et des bibliothèques utilisée afin que l'application ne comporte pas d'élément non maintenu (maintenance au sens correction de bug et sécurité). Cette gestion de l'obsolescence doit aussi permettre à ce qu'une application à la date de sa mise en production bénéficie d'une durée de support la plus importante possible. *Une durée d'un an semble un strict minimum* ;

- l'équipe projet en collaboration avec le pôle sécurité doit mettre en place une politique de patch management permettant d'assurer un maintien constant de l'application condition de sécurité. Les vulnérabilités majeures doivent être corrigées tout de suite. Les autres corrections se font dans le calendrier de patch management associé au projet. Le patch management est pris en compte y compris dans les phases de développement. Dans l'urgence, il est possible de mettre en place des solutions de contournement, mais à terme, ces dernières doivent impérativement être remplacés par une solution pérenne ;
- l'équipe projet devra mettre en place des pratiques de développement lui permettant de s'assurer que les développements réalisés n'exposent pas l'application à des vulnérabilités induites par un développement qui ne serait pas réalisé dans les règles de l'art. Pour les développements DILA, il sera nécessaire de s'appuyer sur l'outillage mis à disposition. Dans le cas d'un développement externe, le titulaire du marché doit expliciter les moyens techniques, organisationnel et humain mis en place pour répondre à cette exigence ;

4.3 Instances et processus de validation

Toute architecture doit être validée par un architecte sécurité et le pôle sécurité et urbanisme avant déploiement.

4.4 Règles relatives au développement sécurisé d'applications

Afin de se conforter à la PSSIE DILA (2), le projet doit se conforter à un ensemble de règles de développement. Ces dernières sont issues des bonnes pratiques et leur application est obligatoire.

Les règles de développements sécurisées sont disponibles en annexe en fin de document. Ces dernières couvrent les thèmes suivants :

- journalisation des erreurs ;
- protection des données ;
- configuration et opérations ;
- authentification ;
- contrôle et gestion des sessions ;
- contrôle des entrées / sorties ;
- contrôle d'accès ;
- protection des formulaires ;

4.5 Règles relatives aux interfaces web[14]

- les développements ne doivent pas générer d'adhérence avec des modules spécifiques (Flash, Silverlight, JRE, etc.) ou une technologie en particulier ;
- les mécanismes cryptographiques TLS (https) doivent être systématiquement activés pour identifier et authentifier la source et protéger les communications ; l'utilisation de la technologie HSTS est fortement recommandée ;
- les mécanismes de protection des cookies de session (HttpOnly, Secure, SameSite) sont mis en œuvre pour se protéger des vols ou exploitation de sessions déjà ouvertes ;
- une politique de sécurité des contenus (CSP, SRI) et des navigateurs (emploi d'en-têtes de sécurité (X-Content-Type-Options, X-Frame-Options, X-XSS-Protection, Referrer-Policy) est élaborée pour se protéger contre les injections de contenus actifs malicieux ;
- les obligations légales sont renseignées sur les sites Internet et un point de contact est publié via le fichier /.well-known/security.txt pour permettre des signalements directement auprès des points de contact identifiés.

4.6 Règles relatives aux services de courriels[15]

- les mécanismes de chiffrement TLS sont mis en œuvre pour l'authentification, la lecture et la distribution des messages (STARTTLS, SMTPS, IMAPS, etc.) ;
- la mise en œuvre des mécanismes permettant de garantir l'authenticité des émetteurs est systématiquement envisagée (contrôle des noms de domaines associés aux serveurs (SPF), signature numérique (DKIM), politique de sécurité liant le tout (DMARC)).

4.7 Configuration TLS

Comme mentionné dans la section « 4.5 Règles relatives aux interfaces web », les applications Web doivent utiliser TLS conformément aux recommandations de l'ANSSI :

- La version TLS cible est la 1.3 ;
- La version 1.2 reste autorisée, mais en désactivant les cyphers les moins sécurisés.

La liste des cyphers autorisée est donnée par l'ANSSI. Elle est disponible dans son guide de recommandations de sécurité relatives à TLS[16].

4.8 Gestion des mots de passe

L'annexe 8 « Annexe : Mesures de développement sécurisées » donne un certain nombre d'indications autour de la gestion des mots de passe. Cette section se concentre sur les points les plus structurants.

---

[14] Règles concernant les interfaces web issue des préconisations ANSSI

[15] Règles concernant les services de courriels issues des préconisation ANSSI [16]

Guide TLS ANSSI https://www.ssi.gouv.fr/uploads/2017/07/anssi-guiderecommandations_de_securite_relatives_a_tls-v1.2.pdf

4.8.1 Complexité des mots de passe

On distingue deux cas d'utilisation : les mots de passe « utilisateur » et les mots de passe pour les « utilisateurs avec pouvoir » (comme les administrateurs).

Les mots de passe des « utilisateurs » doivent suivre les recommandations CNIL[16] :

- 10 caractères avec au moins 3 caractères parmi minuscules, majuscules, chiffres, caractères spéciaux ;
- La saisie des mots de passe doit être accompagnée par un système de type « captcha » ou « système anti-brute force ».

Les mots de passe des « utilisateurs avec pouvoir » doivent respecter les recommandations ANSSI[18] :

- 12 caractères avec 4 caractères parmi minuscules, majuscules, chiffres, caractères spéciaux ;
- La saisie des mots de passe doit être accompagnée par un système de type « captcha » ou « système anti-brute force ».
- Un système d'authentification à double facteur doit être mis en place.

4.8.2 Stockage du mot de passe

Seule une version hachée du mot de passe (condensa) peut être stockée. Cela permet de garantir la confidentialité du mot de passe même en cas d'accès à l'espace de stockage. Les algorithmes de hachage et d'une manière plus générale, tous les algorithmes en lien avec la sécurité doivent faire appel à une bibliothèque éprouvée comme Argon2 ou scrypt, et dans une moindre mesure PBKDF2.

Pour éviter d'établir la correspondance entre le condensa qui est stocké dans l'application et la valeur effective du mot de passe, il est demandé de faire appel à un sel[17] et à un poivre.

Le sel est une information supplémentaire concaténée au mot de passe avant le calcul du condensa. L'information utilisée est la même pour tous les mots de passe.

Le poivre fonctionne de manière similaire au sel. En revanche l'information utilisée est propre à chaque utilisateur.

La combinaison du sel et du poivre rend le calcul de table de correspondance très coûteuse et garantie une bonne confidentialité du mot de passe, même si un attaquant venait à accéder au système.

4.8.3 Réinitialisation du mot de passe

La réinitialisation d'un mot de passe doit se faire à travers un écran de saisie spécifique. Les contraintes pour définir le mot de passe sont les mêmes que celles exigées pour sa définition initiale (complexité, capcha, système anti-force brute).

---

[16] Remandations CNIL pour les mots de passe : https://www.cnil.fr/fr/mot-de-passe [18] Recommandation ANSSI pour la gestion des mots de passe : https://www.ssi.gouv.fr/guide/recommandations-relatives-a-lauthentification-multifacteur-et-aux-mots-depasse/

[17] Sel ; https://fr.wikipedia.org/wiki/Salage_%28cryptographie%29

En aucun cas, un mot de passe doit être envoyé par mail, même si ce dernier est temporaire et n'a une durée de validité réduite.

4.8.4 Traçabilité des accès

L'accès à l'application est très fortement couplé à la saisie du mot de passe. Aussi, il est impératif de pouvoir surveiller ces derniers en traçant les connexions à la fois sur le périmètre « utilisateur » et « utilisateur avec pouvoir » à travers un système de log ou équivalant (les accès doivent être horodatés).

Afin de disposer d'une surveillance effective, l'application doit offrir une interface pour accéder à ces données. Cette dernière peut être l'interface d'accès à un système de log centralisé.

5 Attendu sur le code produit

5.1 Qualité du code

Le code produit est analysé dans des outils de qualimétrie de type SonarQube. Il est demandé que par défaut, le code produit ne lève aucune alerte critique. Il est possible de déroger à cette règle sur accord du responsable du développement DILA qui aura lui-même fixé son périmètre d'autonomie avec le pôle sécurité. SonarQube[18] ainsi que sa configuration est directement disponible sur les plateformes d'intégration continue de la DILA.

5.2 Sécurité du code

Le code livré par un prestataire doit faire l'objet d'un audit de sécurité statique (SAST[19]). L'outil utilisé est à la discrétion du prestataire. Le prestataire doit être en mesure de produire le rapport de sécurité à la demande de la DILA.

Pour certains projets jugés sensibles, le code devra être soumis à une analyse Checkmarx[20] afin de remonter les vulnérabilités potentielles du code (la nécessité de devoir s'appuyer sur Checkmarx sera mentionnée dans le CCTP). Afin de prendre en charge les rapports produits par Checkmarx, il est important que les prestataires répondants aux marchés portant sur du développement soit en mesure de fournir un expert à même d'interpréter et de chalenger ces rapports.

D'une manière générale, la DILA se réserve le droit de réaliser des audits de code (notamment durant les phases d'homologation). Le prestataire à l'obligation de corriger l'ensemble des vulnérabilités remontées dans le cadre de la garantie.

5.3 Respect des différents référentiels

Le code livré doit être conforme aux différents référentiels suivis par l'administration :

- RGAA[21] : Référentiel Général d'Amélioration de l'Accessibilité niveau visé : 80% (sauf mention contraire dans le CCTP) ;

[18] Sonar Qube : https://www.sonarqube.org/
[19] SAST : https://en.wikipedia.org/wiki/Static_application_security_testing
[20] Checkmarx : https://checkmarx.com/
[21] RGAA : https://www.numerique.gouv.fr/publications/rgaa-accessibilite/

- RGI[22] : Référentiel Général d'Interopérabilité ;
- RGS [23]: Référentiel Général de Sécurité ;
- PSSIE [24]: Politique de Sécurité du Système d'Information de l'Etat ;
- RGESN[25] : Référentiel général d'écoconception de services numériques.

6 Pile technologique

6.1 Principes généraux appliqués à la DILA

Les technologies utilisables par les projets sont définies dans la « matrice technologique (3) ». Cette dernière liste les briques et bibliothèques autorisées ainsi que leur cadre d'utilisation. Les informations contenues dans cette matrice reflètent les règles actuellement en vigueur à la DILA et doivent être considérées comme contraignantes. Toutefois, sur la base d'impératifs projet il est possible de proposer des évolutions. Le pôle sécurité se chargera alors de soumettre ces dernières en comité d'architecture afin de statuer sur ces évolutions. Cette possibilité doit être considérée comme exceptionnelle et ne doit pas remettre en question la direction générale prise par la DILA en termes de gouvernance technologique.

Les versions des briques applicatives et des bibliothèques évoluent rapidement. Il est donc possible que la matrice ne soit pas à jour avec les dernières versions disponibles sur le marché. La politique mise en place à la DILA consiste (sauf exception) à utiliser les versions stables les plus récentes en version LTS si le projet le propose.

La matrice technologique met en avant différents statuts :

- **Non standard** : la technologie n'est pas utilisée à la DILA. Demande à étudier ;
- **En cours d'évaluation** : une étude est en cours pour statuer sur l'utilisation ;
- **À venir** : version pas encore disponible d'une technologie utilisée à la DILA ;
- **En cours d'introduction** : un POC est en cours ;
- **Standard** : technologie utilisable sans procédure particulière ;
- **Standard - utilisation limité** : technologie dont on souhaite limiter l'utilisation à des cas d'usages particuliers (performance, cloud…) ;
- **En cours de retrait** : technologie ou version dont la DILA souhaite sortir ;
- **Retiré** : technologie ou version qui n'est plus au catalogue DILA ;
- **Refusé** : technologie que l'on ne souhaite pas utiliser à la DILA ;

[22] RGI : https://www.numerique.gouv.fr/publications/interoperaibilite/
[23] RGS : https://www.ssi.gouv.fr/entreprise/reglementation/confiance-numerique/le-referentiel-general-desecurite-rgs/
[24] PSSIE : https://www.ssi.gouv.fr/entreprise/reglementation/protection-des-systemes-dinformations/lapolitique-de-securite-des-systemes-dinformation-de-letat-pssie/
[25] RGESN : https://ecoresponsable.numerique.gouv.fr/publications/referentiel-general-ecoconception/

6.2 Focus sur le développement Web

Le développement d'applications Web étant majoritaire, cette section fait un focus plus particulier sur ce type d'application et les technologies associées.

Le parc applicatif de la DILA est majoritairement réalisé en Java. Ce langage reste donc celui à privilégier pour tout nouveau développement. Les briques techniques de développement associées sont celles que l'on retrouve classiquement dans de nombreuses entreprises, à savoir : *Java*, *Spring*, *Hibernate*, *PostgreSQL*.

Le développement des IHM est majoritairement réalisé en *JSP* (Java Server Page) avec une utilisation régulière de la bibliothèque *Thymeleaf*. Ce type de technologie est encore possible pour de nouveaux projets. Toutefois, la DILA commence à adopter de plus en plus des technologies JavaScript qui devraient au fur et à mesure devenir le standard.

La pile Javascript retenue est *REACT JS* avec une utilisation de *TypeScript*. Aucune dérogation ne sera donnée pour partir sur un framework JavaScript différent de celui choisi.

7 Références

1. **DILA / DIS.** *Guide d'architecture DILA*. 2020.

2. **DILA.** *Déclinaison DILA PSSIe.* 2015.

3. **DILA / D2S.** *Matrice technologique.* 2019.

8 Annexe : Mesures de développement sécurisées

Ref. interne	Ref. PSSIE DILA	Journalisation des erreurs	Mesures	Observations
SecDev-01-001	DEV-FUITES	Affichage de messages génériques	non divulgation d'information : mot de passe, version, erreur éditeur, etc.	
SecDev-01-002	DEV-FUITES	Gestion des exceptions	Intercepter, filtre ou personnaliser les messages des composants logiciels	
SecDev-01-003	DEV-FUITES EXP-CONFIG	Suppression des messages d'erreurs natifs des framework	Intercepter, filtre ou personnaliser les messages d'erreurs	
SecDev-01-004	EXP-JOUR-SUR	Journaliser les opérations d'authentification	Journaliser les opérations d'authentification réussies ou en échec	
SecDev-01-005	EXP-JOUR-SUR	Journaliser les changements de privilèges	Journaliser toutes opérations de ce type	
SecDev-01-006	EXP-JOUR-SUR	Journaliser les opérations d'administration fonctionnelle	Journaliser toutes opérations de ce type	
SecDev-01-007	EXP-JOUR-SUR	Journaliser l'accès aux données sensibles	Journaliser toutes opérations de ce type	
SecDev-01-008	EXP-JOUR-SUR	Ne pas journaliser les données protégées ou sensibles	En dehors de mécanisme de chiffrement les données sensibles (mot de passe, cookies, etc.) ou personnelles (nom, prénom, etc.) ne doivent pas journalisées	
SecDev-01-009	EXP-CONS-JOUR	Sécuriser le stockage des données journalisées	Les données des journaux doivent être protégées en intégrité (perte, modification) et disponibilité	

			afin de respecter des exigences métiers ou d'enquêtes	
Ref. interne	**Ref. PSSIE DILA**	**Protection des données**	**Mesures**	**Observations**
SecDev-02-001	Bonnes pratiques dont guides ANSSI	Protéger l'accès aux applications	L'usage d'https avec des ciphers fort est obligatoire sur les pages d'authentification. Il est fortement recommandé pour les autres	Porter une attention particulière aux performances et a de possible exploitation en déni de service
SecDev-02-002	Bonnes pratiques dont guides ANSSI	Désactiver l'accès aux pages en http ou les rediriger en https	Une même URL protéger en https ne doit pas être accessible en http	Cette mesure permet de limiter les tentatives de MITM
SecDev-02-003	Bonnes pratiques dont guides ANSSI	Utiliser l'entête Strict-TransportSecurity	Ce mode assure que les navigateurs n'accèdent pas au site https en http	
SecDev-02-004	DEV-LOG-PASS	Chiffrer et saler les mots de passe stockés	Chiffrer avec des algorithmes conforme au RGS	
SecDev-02-005	EXP-PROT-INF	Echange de secrets	Tout échange de clés ou de données secrètes doit être réalisé via un canal sécurisé	Outre la possible utilisation d'un second canal (courrier postal, téléphone, SMS, etc.), la DILA dispose d'outils pour chiffrer les données : de ZED! Sur tous les postes utilisateurs et d'ACID pour certains besoins spécifiques

SecDev-02-006	EXP-PROT-INF	Protéger les secrets	Lorsque des secrets doivent être stockés, les droits d'accès doivent être dictés par le droit d'en connaître et une protection appropriée est mise en place	L'utilisation de produits certifiés comme Keystore (installé sur tous les postes DILA) doivent être privilégiés
SecDev-02-007	Bonnes pratiques dont guides ANSSI	Utiliser des ciphers forts	Utiliser des algorithmes conformes avec le RGS. Une note de A+ sur le site Qualys SSL Labs doit être visée	Valider la compatibilité avec les versions des navigateurs des utilisateurs
SecDev-02-008	EXP-CERTIFS	Utiliser des certificats valides et d'AC fiables	Utiliser des certificats RGS sur le bon niveau d'étoiles	Une étoile en standard
SecDev-02-009	EXP-GEST-PASS	Désactiver les caches "Cache Control Headers" et "Autocomplete"	Désactiver le cache des navigateurs avec l'entête ad-hoc. Les champs sensibles (mot de passe) ne doivent pas être auto renseignés	
SecDev-02-010	EXP-GEST-PASS EXP-CONF-AUTH	Limiter le stockage de données sensibles	Utiliser si besoin les jetons (token) pour éviter le transport ou le stockage de données sensibles	
SecDev-02-011	EXP-DROITS	Anonymisation des données de production	En cas de réutilisation de données de production, les données sensibles (empreintes de mot de passe, certaines données personnelles (CNI, etc.), etc.) doivent être anonymisées pour être réutilisable sur d'autres environnements (développement, qualification, préproduction)	Cette opération d'anonymisation doit être réalisée au plus près des données de production (à l'export vs à l'import sur un autre environnement)
SecDev-02-012	EXP-DROITS	Protection des données d'indexation des sites	Les protections de type robot.txt mise en œuvre contre l'indexation des URL ne doivent pas permettre de divulguer des informations confidentielles ou sensibles (URL d'administration, utilisation de cgi-bin, etc.)	Des alternative existe via le paramétrage d'Apache (ajout de XMeta dans le header des pages) ou l'écriture des pages (balise Meta)

SecDev-02-013	EXP-PROTMALV	Protéger les données contre les codes malveillants	Les données diffusées doivent être de confiance vis-à-vis d'une éventuelle présence de code malveillant : un contrôle antiviral des données est réalisé sur site ou minima sur la source de confiance qui aliment directement le site. Les données uploadées doivent être contrôlées par un antivirus au plus près de la fonction avec si possible une possibilité de notifier l'usager du rejet des fichiers déposés	
Ref. interne	**Ref. PSSIE DILA**	**Configuration et opérations**	**Mesures**	**Observations**
SecDev-03-001	Bonnes pratiques dont guides ANSSI	Utiliser de préférence des outils de déploiement applicatifs automatiques	Facteur de sécurité sur la consistance des modifications et de la répétabilité des opérations de déploiement	
SecDev-03-002	EXP-POL-COR	Etablir un process rigoureux des déploiements	Les nouvelles versions doivent suivre un process formalisé (environnements de développement, de qualification et de recettes, de préproduction, de production, etc.)	
SecDev-03-003	INT-HOMOLOG-SSI	Formaliser les exigences de sécurité	Les métiers formalisent leurs besoins et exigences de sécurité	
SecDev-03-004	R-ORG-04-01 INT-SSI DEV-INTEGR-SECLOC DEV-LOG-CYCLE	Intégrer la sécurité dans le cycle projet	Intégrer la sécurité au plus tôt pour réduire les coûts SSI. Synchroniser la démarche d'analyse des risques avec la démarche projet et ses grands jalons	

SecDev-03-005	DEV-LOG-CYCLE DEV-LOG-WEB	Réaliser des revues de codes	S'assurer en cours de développement et en fin de développement du code, de sa qualité en terme SSI (absence de vulnérabilité du Top 10 de l'OWASP à minima). Utiliser des librairies valides et non vulnérables.	
SecDev-03-006	DEV-LOG-CYCLE DEV-LOG-WEB	Réaliser des tests de sécurité	Conduire des tests de sécurité pendant et en fin de projet, avant la mise en production. Exemple : en même temps que les recettes applicatives	
SecDev-03-007	cf. lien sur les guides ANSSI	Composants applicatifs et techniques	Les composants doivent être configurés selon les bonnes pratiques de la sécurité, en tenant comptes des "guides de durcissement" et des guides publiés par l'ANSSI. (routeurs, pare-feu, switches, OS, serveurs web, serveurs d'application, Bases de données et frameworks applicatifs)	
SecDev-03-008	Bonnes pratiques dont guides ANSSI	Formaliser un plan de réponse sur incident	Un plan doit être formalisé, mis à jour et testé régulièrement avec l'identification des contacts ad-hoc.	
SecDev-03-009	DEV-SOUS-TRAIT	Sensibiliser l'équipe à la SSI	Etendre la sensibilisation au de-là des équipes de développement ou projet	

Ref. interne	Ref. PSSIE DILA	Authentification	Mesures	Observations
SecDev-04-001	EXP-PROT-INF	Ne pas stocker en dur les secrets	Les mots de passe ne sont pas stockés dans le code de l'application, même pour des besoins de tests ou de développement	

SecDev-04-002	EXP-INIT-PASS	Mettre en œuvre une procédure de réinitialisation de mot de passe robuste	Ne pas s'appuyer sur un système de questions/réponses simplistes exposé à des attaques par force brute. Par exemple : Ne jamais envoyer par courriel un mot de passe en clair mais privilégier un lien avec une URL unique de confirmation de changement de mot de passe, valide sur une durée bornée	
SecDev-04-003	EXP-QUAL-PASS EXP-POL-PASS	Mettre en œuvre une politique de mot de passe robuste	Prendre en compte les exigences de la PSSIE DILA et des recommandations ANSSI en termes de politiques de mots de passe. Les implémentations divergentes doivent justifiées auprès de la chaîne SSI DILA	
SecDev-04-004	Bonnes pratiques dont guides ANSSI	Mettre en place des mesures contre les attaques par force brute	Le mécanisme de protection peut être s'appuyer : - sur un blocage du compte après un nombre bornée de tentatives infructueuses (3, 5, etc.) - sur une temporisation après un nombre bornée de tentatives infructueuses (3 ou 5). (exemple : 15 secondes au 5ème échec puis un doublement du délai jusqu'au 15 -ème échec qui réinitialise les compteurs. - sur un Captcha - etc. Il est important de veiller à la compatibilité de la procédure de déblocage avec les exigences métiers pour éviter une attaque en déni de service et de journaliser les opérations avec remontée d'alerte en cas d'enclenchement du système de protection.	
SecDev-04-005	DEV-FUITES	Ne pas divulguer trop d'information dans les messages d'erreur de connexion ou d'authentification	Le message doit être neutre les raisons de l'échec de connexion. En particulier, le message doit être identique en cas d'erreur sur le mot de ou de l'identifiant utilisateur	

SecDev-04-006	EXP-PROT-INF	Le stockage des éléments d'authentification (mot de passe) aux bases de données doivent sécurisés	Une centralisation des mots de passe comme le propose certains framework est à utiliser en priorité. Le stockage dans le code applicatif est à bannir.	
SecDev-04-007	EXP-DROITS EXP-PROFILS	Les applications et les middlewares doivent être exécutés avec des droits minimaux	En cas de compromission, il est important d'avoir octroyer que les privilèges nécessaires afin de limiter les impacts de la compromission	
SecDev-04-008	EXP-QUAL-PASS EXP-POL-PASS	Valider la robustesse des mots de passe	Mettre en place une validation du niveau de robustesse des mots de passe	
Ref. interne	**Ref. PSSIE DILA**	**Contrôle et gestion des sessions**	**Mesures**	**Observations**
SecDev-05-001	Bonnes pratiques dont guides ANSSI	Générer des jetons de session robuste	S'assurer du niveau aléatoire des jetons et de leur nonprédictibilité (longueur suffisante et aléatoire)	
SecDev-05-002	Bonnes pratiques dont guides ANSSI	Renouvellement du jeton de session	Le jeton doit être renouvelé/supprimé lorsque l'utilisateur se connecte ou se déconnecte, et doit être renouvelé lors d'une élévation de privilège (accès à une fonction sensible par exemple)	
SecDev-05-003	Bonnes pratiques dont guides ANSSI	Paramétrer le timeout du jeton en cas de session inactive	L'utilisateur doit être déconnecté automatiquement après une durée d'inactivité à déterminer selon les besoins métier mais raisonnablement inférieur à 30'. Attention aux application Ajax qui peuvent générer des requêtes qui réinitialisent ce délai d'inactivité	

SecDev-05-004	Bonnes pratiques dont guides ANSSI	Paramétrer le timeout du jeton de session	Pour se protéger des attaques de vol de jetons, mettre en place un timeout du jeton compatible avec les exigences métiers (4h, 8h, etc.) qui induit une demande de ressaisie du mot de passe	
SecDev-05-005	Bonnes pratiques dont guides ANSSI	Invalider les jetons de session au premier signe de falsification	A moins que l'application nécessite plusieurs sessions simultanées pour un seul utilisateur, implémentez des fonctionnalités pour détecter les tentatives de clonage de session. Si un signe de clonage de session est détecté, la session doit être détruite, ce qui oblige l'utilisateur réel à se ré-authentifier.	
SecDev-05-006	Bonnes pratiques dont guides ANSSI	Supprimer les éléments de session à la déconnexion	Quand l'utilisateur se déconnecte, les données correspondantes sur le serveur, dont le jeton de session, doivent être supprimées	
SecDev-05-007	Bonnes pratiques dont guides ANSSI	Mettre à disposition de l'utilisateur, une capacité de déconnexion	Prévoir une fonction de déconnexion facilement visualisable sur chaque page (bouton, etc.) lorsque l'utilisateur est en mode authentifié	
SecDev-05-008	Bonnes pratiques dont guides ANSSI	Paramétrer les sessions avec les options de sécurité disponibles	Les sessions doivent être couvertes par les modes http only et le mode secure pour celles sous https	
SecDev-05-009	Bonnes pratiques dont guides ANSSI	Paramétrer correctement les cookies "Domain" et "Path"	Ces cookies doivent être positionnés sur le niveau le plus restrictif. Tout usage de caractère "joker" (*) doit être dûment justifié	

SecDev-05-010	Bonnes pratiques dont guides ANSSI	Paramétrer un délai de fin de validité du jeton	La durée doit être raisonnable. Une durée illimitée est à proscrire.	
SecDev-05-011	Bonnes pratiques dont guides ANSSI	Vérifier les dernières connexions	- Rappeler à l'utilisateur sa date et heure de dernière connexion ;	
Ref. interne	**Ref. PSSIE DILA**	**Contrôle des entrées / sorties**	**Mesures**	**Observations**
SecDev-06-001	Bonnes pratiques dont guides ANSSI	Adapter l'encodage des données envoyer par le serveur	Par exemple les données d'une URL sont encoder différemment de celle portées par un javascript: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet	
SecDev-06-002	Bonnes pratiques dont guides ANSSI	Privilégier un filtrage par listes blanches des données envoyées au serveur	Accepter uniquement des données qui répondent à des critères attendus (format date, numériques, texte alphanumérique, etc.). Les listes noires moins efficaces peuvent être utilisées pour interdire l'envoi de certains motifs de caractères au serveur	
SecDev-06-003	Bonnes pratiques dont guides ANSSI	Utiliser des requêtes SQL paramétrées	Les requêtes SQL ne doivent pas être générées dynamiquement en utilisant une concaténation de chaînes de caractères, mais être construites à l'avance pour être exécutée à la demande	
SecDev-06-004	Bonnes pratiques dont guides ANSSI	Utiliser des jeton anti-rejeu	Les jetons anti-rejeu uniques doivent être utilisés pour valider chaque requête. En particulier lorsqu'il s'agit d'opérations sensibles : changement de mot de passe, lancement de traitement métier critique, etc.	Cette protection est désormais intégrée comme un paramètre de la plupart des framework de développement

SecDev-06-005	Bonnes pratiques dont guides ANSSI	Définir formellement le mode d'encodage des pages HTML	Cette information portée dans les entêtes HTTP ou les meta balises HTML permet de limiter les risques du XSS	
SecDev-06-006	Bonnes pratiques dont guides ANSSI	Contrôler l'upload de fichiers	Il est important de valider la taille, le type, le contenu du fichier et de vérifier qu'il n'est pas possible de modifier le chemin d'écriture du fichier	
SecDev-06-007	Bonnes pratiques dont guides ANSSI	Paramétrer l'entête Nosniff pour les fichiers uploadés et accessibles sur le site	Il est recommandé de laisser le serveur ou l'application déterminer le type de données pour la diffusion de fichiers uploadés. A cette fin, utiliser l'en-tête X-Content-TypeOptions: nosniff.	
SecDev-06-008	Bonnes pratiques dont guides ANSSI	Valider les entrées	N'accepter les entrées que sur la méthode attendue (POST vs GET)	
SecDev-06-009	Bonnes pratiques dont guides ANSSI	Paramétrer l'entête X-Frame	Ce paramètre protège le chargement de contenu à partir d'un site tiers dans une frame et couvre les attaques de type Clickjacking.	
SecDev-06-010	Bonnes pratiques dont guides ANSSI	Paramétrer les entêtes HTTPResponse	Les entêtes Content Security Policy (CSP), X-XSS-Protection renforce la protection contre les attaques de type Cross-Site Scripting (XSS) and Man-in-the-Middle (MITM)	
Ref. interne	**Ref. PSSIE DILA**	**Contrôle d'accès**	**Mesures**	**Observations**

SecDev-07-001	Bonnes pratiques dont guides ANSSI	Appliquer un contrôle d'accès cohérent	Mettre la brique de contrôle des droits et accès en contrôle permanant de toutes les connexions entrantes / appel de page / JS que l'utilisateur soit authentifié ou non. Les droits peuvent évoluer en cours de session (suppression de compte), la session peut être fermé ou ouverte sur un autre onglet...	
SecDev-07-002	EXP-DROITS	Appliquer le principe du moindre privilège	Ce principe traduit que ce qui n'est pas explicitement autorisé est interdit	
SecDev-07-003	Bonnes pratiques dont guides ANSSI	Interdire l'accès directs à des fonctions d'attribution de droits	Ne pas faire porter la gestion de droits sur des informations pouvant être manipulées (variable get ou post correspondant à un group utilisateur par exemple). Les droits d'accès octroyés doivent toujours correspondre à un utilisateur authentifié et des informations de droits obtenue et vérifié côté serveur.	
SecDev-07-004	Bonnes pratiques dont guides ANSSI	Ne pas coder des redirections non validées	Valider et contrôler les redirections pour se protéger d'un renvoi vers des sites malveillants.	
Ref. interne	**Ref. PSSIE DILA**	**Protection des formulaires**	**Mesures**	**Observations**
SecDev-08-001	Bonnes pratiques dont guides ANSSI	Protéger les formulaires non authentifiés d'une soumission massive par des robots	Mettre en place un système de protection type Captcha ou équivalent	**Attention à valider sa "compatibilité" avec le RGPD et le RGAA**