Fiche 3 — Sécurité et respect des données

Introduction

Dans le développement web, la **sécurité** n'est pas une option, c'est une condition de confiance. Chaque site, chaque application, chaque API manipule des données — souvent personnelles, parfois sensibles.
Mal protéger ces données, c'est exposer les utilisateurs à des risques concrets : vol d'identité, usurpation, perte de vie privée.

Le **numérique responsable** inclut donc une exigence claire :
→ concevoir des services **sûrs par défaut**, respectueux des personnes et du cadre légal.

1. Comprendre les enjeux de la sécurité numérique

1.1 La sécurité, un pilier de la qualité logicielle

Un service sécurisé est :

- **fiable** (il fonctionne sans faille critique),
- **prévisible** (il résiste aux usages imprévus),
- **responsable** (il protège les données confiées).

La sécurité ne se limite pas au code : elle englobe les choix d'architecture, d'hébergement, d'outils et de gestion d'équipe.

1.2 Les principaux risques

- **Injection SQL** : modification des requêtes via les entrées utilisateur.
- **Cross-Site Scripting (XSS)** : injection de scripts malveillants dans le navigateur.

- **Cross-Site Request Forgery (CSRF)** : détournement d'une session authentifiée.
- **Fuite de données** : stockage ou partage non sécurisé.
- **Exposition d'API** : endpoints accessibles sans contrôle.
- **Mots de passe faibles ou mal stockés.**

Chaque faille non corrigée est une porte ouverte.

> Une ligne négligée peut compromettre une entreprise entière.

2. Sécurité by design : anticiper dès la conception

2.1 Le principe "security by design"

- Intégrer la sécurité **dès la phase de conception**, pas après coup.
- Identifier les données sensibles (mails, identifiants, historiques).
- Cartographier les flux : où les données circulent, qui y accède.
- Prévoir les **mécanismes de défense** à chaque niveau (frontend, backend, base de données).

2.2 Pratiques fondamentales

- **Validation stricte** des entrées utilisateurs.
- **Échappement des sorties** dans les vues et templates.
- **Hashage des mots de passe** (bcrypt, Argon2, sodium).
- **Authentification sécurisée** (tokens JWT, sessions avec expiration).
- **Gestion fine des rôles et autorisations.**
- **Chiffrement** des données sensibles en transit (HTTPS/TLS) et au repos (AES).

2.3 Sécuriser la configuration

- Ne jamais exposer les fichiers `.env` ou clés API.
- Ne pas stocker de secrets dans le code source.
- Séparer les environnements (dev, staging, prod).
- Maintenir les dépendances à jour (Composer, npm audit).
- Activer les journaux d'erreurs sécurisés (sans données sensibles).

3. Le RGPD et le respect des données personnelles

3.1 Qu'est-ce qu'une donnée personnelle ?

Toute information permettant d'**identifier directement ou indirectement une personne** : nom, email, photo, adresse IP, identifiant de session, localisation, etc.

3.2 Principes clés du RGPD (Règlement Général sur la Protection des Données)

Principe	Description	Exemple
Finalité	Collecter uniquement ce qui est nécessaire	Ne pas demander le téléphone pour un simple formulaire de contact
Consentement	L'utilisateur doit accepter clairement	Case à cocher explicite, non précochée
Transparence	Informer sur l'usage des données	Politique de confidentialité claire
Durée limitée	Ne pas conserver indéfiniment	Suppression automatique après inactivité
Sécurité	Protéger les données collectées	Cryptage, accès restreint
Droits utilisateurs	Permettre l'accès, la rectification, la suppression	Formulaire de demande RGPD

3.3 Privacy by design et by default

- **By design** : intégrer la protection des données dès la conception.
- **By default** : paramétrer par défaut les options les plus protectrices.

Exemples :

- Ne pas activer la géolocalisation sans demande explicite.

- Limiter la visibilité publique des profils par défaut.
- Chiffrer automatiquement les sauvegardes.

4. Bonnes pratiques de développement sécurisé

4.1 Validation et encodage

- Toujours valider côté **serveur**, même si la validation existe côté client.
- Utiliser les filtres intégrés (ex. `filter_var()` en PHP, `class-validator` en Node.js).
- Échapper systématiquement le HTML dans les vues (`{{ variable|escape }}`).

4.2 Gestion des sessions et cookies

- Utiliser des cookies **HttpOnly** et **Secure**.
- Définir un **temps d'expiration raisonnable**.
- Activer le **SameSite** pour limiter les attaques CSRF.
- Invalider les sessions après changement de mot de passe ou déconnexion.

4.3 Stockage des mots de passe

- Ne jamais stocker en clair.
- Utiliser un **algorithme de hachage** adapté (bcrypt, Argon2).
- Ajouter un **sel** unique à chaque mot de passe.
- Interdire les mots de passe trop faibles ou réutilisés.

4.4 Sécurité des API

- Authentifier chaque requête via token ou clé API.
- Limiter les permissions et la durée de validité.
- Ajouter du **rate limiting** pour éviter les abus.
- Masquer les endpoints inutiles ou non documentés.

5. Sécurité organisationnelle

5.1 Gestion des rôles et droits

- Mettre en place une politique claire : qui fait quoi, qui accède à quoi.
- Distinguer les environnements de test, de préproduction et de production.
- Mettre en place des revues de code axées sécurité.

5.2 Mise à jour et maintenance

- Surveiller les alertes de vulnérabilités (GitHub Security, npm audit, Snyk).
- Mettre à jour régulièrement les dépendances et frameworks.
- Documenter les changements et procédures d'incident.

5.3 Formation et sensibilisation

La sécurité est un **effort collectif** :

- Former les équipes aux réflexes essentiels.
- Sensibiliser aux risques d'ingénierie sociale (phishing, hameçonnage).
- Créer une culture de vigilance et de transparence.

6. Activités pratiques

Atelier 1 — Mini-audit de sécurité

- Analysez un petit projet (personnel ou open source).
- Vérifiez :
  - stockage des mots de passe,
  - validation des formulaires,
  - usage des cookies,
  - configuration des environnements.
- Dressez une liste de points faibles et d'actions correctives.

Atelier 2 — Mise en conformité RGPD

- Choisissez un projet contenant des formulaires utilisateurs.
- Rédigez :
  - une politique de confidentialité simplifiée,
  - un texte de consentement clair,
  - un mécanisme d'effacement ou de consultation des données.

Livrables attendus

- Rapport d'audit de sécurité (1 à 2 pages).
- Mini-charte RGPD du projet (document ou fichier `PRIVACY.md`).
- Correctifs appliqués ou propositions concrètes d'amélioration.

Évaluation

Critère	Description	Pondération
Analyse des vulnérabilités	Capacité à identifier les risques majeurs	40 %
Application des bonnes pratiques	Cohérence et rigueur des correctifs ou recommandations	40 %
Respect du cadre RGPD	Clarté et pertinence des éléments de conformité	20 %

Pour aller plus loin

- Consulter : https://owasp.org — Top 10 des vulnérabilités web.
- Lire : *Guide RGPD pour les développeurs* (CNIL, 2023).
- Explorer : *Security by Design Manifesto* (OWASP).
- Utiliser : *Mozilla Observatory*, *SonarQube*, *Snyk* pour l'analyse automatisée.

Message clé

> La sécurité n'est pas une contrainte, c'est un **devoir professionnel**.
> Un développeur responsable protège non seulement les données,
> mais aussi la **confiance** que les utilisateurs lui accordent.