

Exemple de recettage — Application web

Recette d'une petite application web PHP / MariaDB d'inscription à des ateliers : cas complet rédigé, du cahier de recette au procès-verbal

§ Un exemple à adapter

Ce document présente un **exemple rédigé** de recettage, appliqué à un cas concret. Il illustre la démarche décrite dans le livret « Le recettage informatique » : il ne s'agit pas d'un modèle unique à recopier.

Le nombre de tests, leur formulation et le niveau de détail doivent être **adaptés** à chaque application et à chaque cahier des charges. Les noms, adresses, extraits de code et données sont fictifs ou indicatifs.

1. Présentation du cas

Une association souhaite une petite application web permettant à ses adhérents de **s'inscrire à des ateliers**. Un **espace d'administration** protégé par identifiant et mot de passe affiche la liste des inscrits. L'application est développée en PHP, avec une base MariaDB.

Environnement de recette

Élément	Valeur	Remarque
Serveur	web-dev01 (Debian 12)	Apache 2.4, PHP 8.2, MariaDB 10.11
URL de recette	http://inscriptions.lab.local	Environnement interne de recette
Poste de test	dev-pc01	Firefox, terminal avec curl
Version testée	v1.0.0 (commit a3f9c2e)	Version identifiée dans le dépôt Git
Jeu de données	3 ateliers, 12 inscrits fictifs	Aucune donnée réelle

Exigences retenues

Réf.	Exigence du cahier des charges
E1	Un adhérent peut s'inscrire à un atelier ; l'inscription est enregistrée en base.
E2	Les saisies sont contrôlées côté serveur (champs obligatoires, format de l'e-mail), et pas seulement dans le navigateur.
E3	L'application résiste aux injections SQL.
E4	Les données saisies par les utilisateurs sont affichées sans pouvoir exécuter de script (protection XSS).
E5	Les pages d'administration ne sont accessibles qu'après authentification.
E6	Les mots de passe des administrateurs sont stockés sous forme d'empreinte (hachage), jamais en clair.

Réf.	Exigence du cahier des charges
E7	Les échanges avec l'application sont chiffrés (HTTPS).

Vigilance : tests de sécurité et données personnelles

Les tests d'injection (tests 3 et 4) ne se pratiquent que sur **sa propre application**, dans un environnement de recette. Les mener sur un site tiers sans autorisation est illégal.

Le jeu de test utilise des **données fictives** : on ne recette jamais avec de vraies données personnelles d'adhérents (RGPD). Les empreintes de mots de passe sont masquées dans les preuves.

2. Fiches de test et preuves

Test 1 — Inscription valide à un atelier

FICHE DE TEST N° 1	
Exigence couverte	E1
Objectif du test	Vérifier qu'une inscription correcte est acceptée et enregistrée.
Type de test	Fonctionnel
Prérequis	Application déployée ; atelier « Initiation Linux » ouvert.
Procédure	1. Ouvrir la page d'inscription dans Firefox. 2. Saisir : Martin / Léa / lea.martin@example.org / Initiation Linux, puis valider. 3. Contrôler l'enregistrement dans la table inscrit.
Résultat attendu	Message de confirmation affiché ; une nouvelle ligne en base avec les valeurs saisies et la date du jour.
Résultat obtenu	Conforme.
Preuve(s)	Captures 1a et 1b
Statut	☒ Conforme
Anomalie / commentaire	Aucune.
Re-test après correction	Sans objet.

Capture 1a — Réponse de l'application (code source de la page, extrait)

```
<main>
  <p class="succes">Votre inscription à l'atelier « Initiation Linux » est
  enregistrée.</p>
  <a href="index.php">Retour à la liste des ateliers</a>
</main>
```

 Explicitation

Contexte : poste dev-pc01, Firefox, page `http://inscriptions.lab.local/inscription.php` après validation du formulaire (affichage du code source avec Ctrl+U).

Action : envoi du formulaire avec un jeu de données valide.

À observer : le paragraphe de classe « succès » qui confirme l'inscription.

Interprétation : l'application accepte la saisie et l'indique clairement à l'utilisateur. Cette preuve seule ne suffit pas : il faut vérifier la base (capture 1b).

Données masquées : aucune donnée sensible.

Capture 1b — Contrôle en base de données

```
MariaDB [inscriptions]> SELECT id, nom, prenom, email, id_atelier, date_inscription  
-> FROM inscrit ORDER BY id DESC LIMIT 1;
```

id	nom	prenom	email	id_atelier	date_inscription
13	Martin	Léa	lea.martin@example.org	2	2026-10-15 14:02:11

 Explicitation

Contexte : serveur web-dev01, client MariaDB, base inscriptions.

Action : lecture de la dernière ligne de la table inscrit.

À observer : la ligne 13 : valeurs identiques à la saisie (accents compris), atelier 2 (Initiation Linux) et date du jour.

Interprétation : l'inscription est réellement enregistrée, avec un encodage correct des caractères accentués. **Test conforme.**

Données masquées : aucune ; données fictives (domaine example.org réservé aux exemples).

Test 2 — Contrôle des saisies côté serveur

FICHE DE TEST N° 2	
Exigence couverte	E2
Objectif du test	Vérifier que le serveur refuse une saisie invalide, même si les contrôles du navigateur sont contournés.
Type de test	Fonctionnel / Sécurité
Prérequis	Test 1 conforme. 13 inscrits en base.
Procédure	1. Envoyer directement au serveur, avec curl, un formulaire sans prénom et avec un e-mail invalide. 2. Compter les inscrits en base.
Résultat attendu	Deux messages d'erreur ; aucune nouvelle ligne en base (toujours 13).
Résultat obtenu	Conforme.
Preuve(s)	Capture 2
Statut	☒ Conforme
Anomalie / commentaire	Aucune.
Re-test après correction	Sans objet.

Capture 2 — Envoi d'une saisie invalide sans passer par le navigateur

```
dev@dev-pc01:~$ curl -s -X POST http://inscriptions.lab.local/inscription.php \
-d 'nom=Test&prenom=&email=pas-un-email&id_atelier=2' | grep 'class="erreur"'
<p class="erreur">Le prénom est obligatoire.</p>
<p class="erreur">L'adresse e-mail n'est pas valide.</p>

MariaDB [inscriptions]> SELECT COUNT(*) FROM inscrit;
| COUNT(*) | 13 |
```

 Explicitation

Contexte : poste dev-pc01, terminal ; puis client MariaDB sur web-dev01.

Action : requête POST envoyée avec curl, ce qui contourne les attributs `required` et `type="email"` du formulaire HTML.

À observer : les deux messages d'erreur renvoyés par le serveur, puis le nombre d'inscrits inchangé.

Interprétation : la validation est bien réalisée côté serveur : un utilisateur malveillant ne peut pas l'éviter en contournant le navigateur. **Test conforme.**

Données masquées : aucune donnée sensible.

Test 3 — Résistance à l'injection SQL

FICHE DE TEST N° 3	
Exigence couverte	E3
Objectif du test	Vérifier qu'une saisie piégée dans l'identifiant ne permet pas de contourner l'authentification.
Type de test	Sécurité
Prérequis	Compte administrateur « gestionnaire » existant.
Procédure	1. Envoyer au formulaire de connexion l'identifiant admin' OR '1'='1' -- et un mot de passe quelconque. 2. Relire la requête de connexion dans le code source.
Résultat attendu	Connexion refusée ; requête préparée avec paramètre lié.
Résultat obtenu	Conforme.
Preuve(s)	Captures 3a et 3b
Statut	☒ Conforme
Anomalie / commentaire	Aucune.
Re-test après correction	Sans objet.

Capture 3a — Tentative d'injection

```
dev@dev-pc01:~$ curl -s -X POST http://inscriptions.lab.local/admin/login.php \
--data-urlencode "login=admin' OR '1'='1' -- " --data-urlencode 'mdp=x' \
| grep -E 'erreur|Tableau de bord'
<p class="erreur">Identifiant ou mot de passe incorrect.</p>
```

 Explicitation

Contexte : poste dev-pc01, terminal, environnement de recette.

Action : envoi d'un identifiant contenant une injection SQL classique.

À observer : le message d'erreur, et l'absence de « Tableau de bord » (page affichée après connexion).

Interprétation : la saisie piégée est traitée comme un simple identifiant inexistant : l'authentification n'est pas contournée.

Données masquées : aucune donnée sensible.

Capture 3b — Requête de connexion (admin/login.php, extrait)

```
$req = $pdo->prepare('SELECT id, mdp_hash FROM utilisateur WHERE login = :login');
$req->execute(['login' => $_POST['login']]);
$user = $req->fetch();
```

 Explicitation

Contexte : dépôt Git, fichier admin/login.php, version v1.0.0.

Action : relecture du code de la requête.

À observer : la requête préparée et le paramètre : `login` lié à l'exécution.

Interprétation : la saisie n'est jamais concaténée dans la requête SQL : c'est ce qui explique le résultat de la capture 3a. Le test de comportement est confirmé par la relecture du code. **Test conforme.**

Données masquées : aucune donnée sensible.

Test 4 — Protection contre l'injection de script (XSS)

FICHE DE TEST N° 4	
Exigence couverte	E4
Objectif du test	Vérifier qu'un script saisi dans le nom n'est pas exécuté lors de l'affichage dans l'espace d'administration.
Type de test	Sécurité
Prérequis	Tests 1 à 3 conformes.
Procédure	1. S'inscrire avec le nom <code><script>alert('XSS')</script></code>. 2. Se connecter à l'espace d'administration et ouvrir la liste des inscrits. 3. Afficher le code source de la page.
Résultat attendu	Aucune fenêtre d'alerte ; le nom apparaît sous forme de texte, avec les caractères <code><</code> et <code>></code> échappés.
Résultat obtenu	1er passage : fenêtre « XSS » affichée (non conforme). 2e passage : texte échappé.
Preuve(s)	Captures 4a et 4b
Statut	☒ Conforme après correction
Anomalie / commentaire	Anomalie bloquante (sécurité) : faille XSS stockée dans admin/liste.php, les noms étaient affichés sans échappement. Corrigée avec <code>htmlspecialchars()</code> .
Re-test après correction	15/10/2026 — Conforme (capture 4b), et test 1 rejoué sans régression.

Capture 4a — Premier passage : code source de admin/liste.php (extrait)

```
<tr>
  <td><script>alert('XSS')</script></td>
  <td>example</td>
  <td>test.xss@example.org</td>
</tr>
```

 Explicitation — résultat non conforme

Contexte : poste dev-pc01, Firefox, connecté à l'espace d'administration ; une fenêtre d'alerte « XSS » s'est ouverte à l'affichage de la liste.

Action : affichage du code source de la page (Ctrl+U).

À observer : la balise `<script>` insérée telle quelle dans la cellule du tableau.

Interprétation : le navigateur exécute le script saisi par un visiteur lorsqu'un administrateur consulte la liste. Un attaquant pourrait ainsi agir avec les droits de l'administrateur. **Test non conforme** ; anomalie bloquante.

Données masquées : aucune donnée sensible.

Correction appliquée dans admin/liste.php (même traitement pour le prénom et l'e-mail)

```
// Avant
<td><?= $inscrit['nom'] ?></td>

// Après : échappement de toutes les données affichées
<td><?= htmlspecialchars($inscrit['nom'], ENT_QUOTES, 'UTF-8') ?></td>
```

Capture 4b — Re-test : code source après correction (extrait)

```
<tr>
  <td>&lt;script&gt;alert(&#039;XSS&#039;)&lt;/script&gt;</td>
  <td>example</td>
  <td>test.xss@example.org</td>
</tr>
```

✓ Explicitation

Contexte : poste dev-pc01, Firefox, même page après déploiement de la correction (v1.0.1).

Action : rechargement de la liste puis affichage du code source.

À observer : les caractères < > et ' remplacés par leurs entités HTML (< > ').

Interprétation : le nom s'affiche désormais comme du texte, sans être exécuté ; aucune fenêtre d'alerte. Le test 1 a été rejoué pour vérifier l'absence de régression. **Test conforme.**

Données masquées : aucune donnée sensible.

Test 5 — Accès à l'espace d'administration

FICHE DE TEST N° 5	
Exigence couverte	E5
Objectif du test	Vérifier qu'une page d'administration n'est pas accessible sans être connecté.
Type de test	Droits et accès
Prérequis	Aucune session ouverte sur le poste de test.
Procédure	1. Demander directement la page admin/liste.php avec curl, sans cookie de session.
Résultat attendu	Redirection (code 302) vers login.php ; aucune donnée d'inscrit renvoyée.
Résultat obtenu	Conforme.
Preuve(s)	Capture 5
Statut	☒ Conforme
Anomalie / commentaire	Aucune.
Re-test après correction	Sans objet.

Capture 5 — Accès direct sans authentification

```
dev@dev-pc01:~$ curl -s -i http://inscriptions.lab.local/admin/liste.php | head -5
HTTP/1.1 302 Found
Date: Thu, 15 Oct 2026 14:31:07 GMT
Server: Apache
Location: login.php
Content-Length: 0
```

 Explicitation

Contexte : poste dev-pc01, terminal, sans session.

Action : demande directe de l'URL de la liste des inscrits, en affichant les en-têtes de la réponse.

À observer : le code 302, l'en-tête Location vers login.php et un contenu vide (Content-Length: 0).

Interprétation : connaître l'adresse de la page ne suffit pas pour y accéder, et aucune donnée n'est envoyée avant la redirection. **Test conforme.** Remarque : l'en-tête Server pourrait être réduit pour ne pas révéler le logiciel (piste d'amélioration).

Données masquées : aucune donnée sensible.

Test 6 — Stockage des mots de passe

FICHE DE TEST N° 6	
Exigence couverte	E6
Objectif du test	Vérifier que les mots de passe des administrateurs sont hachés.
Type de test	Sécurité
Prérequis	Compte « gestionnaire » créé via l'application.
Procédure	1. Lire la colonne mdp_hash de la table utilisateur. 2. Relire le code de création et de vérification du mot de passe.
Résultat attendu	Empreinte bcrypt (préfixe \$2y\$) ; utilisation de password_hash() et password_verify().
Résultat obtenu	Conforme.
Preuve(s)	Capture 6
Statut	☒ Conforme
Anomalie / commentaire	Aucune.
Re-test après correction	Sans objet.

Capture 6 — Contenu de la table et code associé

```
MariaDB [inscriptions]> SELECT login, mdp_hash FROM utilisateur;
+-----+-----+
| login      | mdp_hash                                     |
+-----+-----+
| gestionnaire | $2y$10$[masqué]                           |
+-----+-----+

// admin/creer_utilisateur.php
$hash = password_hash($_POST['mdp'], PASSWORD_DEFAULT);
// admin/login.php
if ($user && password_verify($_POST['mdp'], $user['mdp_hash'])) {
```

 Explicitation

Contexte : serveur web-dev01, client MariaDB ; puis dépôt Git, version v1.0.1.

Action : lecture de la table utilisateur, puis relecture des deux fonctions concernées.

À observer : le préfixe \$2y\$10\$ (algorithme bcrypt, coût 10) et les appels à password_hash() et password_verify().

Interprétation : aucun mot de passe n'est stocké en clair ; même en cas de fuite de la base, ils ne sont pas directement lisibles. **Test conforme.**

Données masquées : empreinte masquée après le préfixe (■) : même hachée, elle ne doit pas être diffusée.

Test 7 — Chiffrement des échanges (HTTPS)

FICHE DE TEST N° 7	
Exigence couverte	E7
Objectif du test	Vérifier que l'application est servie en HTTPS.
Type de test	Sécurité
Prérequis	Aucun.
Procédure	1. Demander l'application en HTTPS avec curl.
Résultat attendu	Réponse HTTPS valide avec un certificat reconnu ; redirection automatique de HTTP vers HTTPS.
Résultat obtenu	Connexion HTTPS refusée : le serveur de recette n'écoute qu'en HTTP.
Preuve(s)	Capture 7
Statut	☒ Conforme avec réserve
Anomalie / commentaire	Anomalie majeure : HTTPS non configuré. Acceptée en réserve, car l'environnement de recette est interne et isolé ; mise en place d'un certificat obligatoire avant la mise en production.
Re-test après correction	À réaliser lors de la levée de la réserve.

Capture 7 — Tentative de connexion en HTTPS

```
dev@dev-pc01:~$ curl -sS -I https://inscriptions.lab.local/  
curl: (7) Failed to connect to inscriptions.lab.local port 443: Connection refused
```

 Explicitation — résultat non conforme

Contexte : poste dev-pc01, terminal.

Action : demande des en-têtes de la page d'accueil en HTTPS.

À observer : l'erreur « Connection refused » sur le port 443.

Interprétation : aucun service HTTPS n'est disponible : identifiants et données personnelles circuleraient en clair. **Non conforme** ; compte tenu du contexte de recette interne, l'anomalie est acceptée **en réserve**, avec une échéance de levée.

Données masquées : aucune donnée sensible.

3. Synthèse des tests

N°	Test	Statut	Anomalie / réserve	Re-test
1	Inscription valide	Conforme	—	—
2	Contrôle des saisies côté serveur	Conforme	—	—
3	Injection SQL	Conforme	—	—
4	Injection de script (XSS)	Conforme après correction	Bloquante : XSS stockée	Conforme
5	Accès à l'administration	Conforme	—	—
6	Stockage des mots de passe	Conforme	—	—
7	Chiffrement HTTPS	Conforme avec réserve	Majeure : HTTPS absent	À faire

4. Procès-verbal de recette

PROCÈS-VERBAL DE RECETTE	
Projet / installation	Application web d'inscription aux ateliers — version v1.0.1
Bilan des tests	7 tests réalisés : 6 conformes (dont 1 après correction), 0 non conforme, 1 avec réserve
Décision	☐ Recette prononcée sans réserve ☒ Recette prononcée avec réserves ☐ Recette refusée
Réserves éventuelles	R1 — Mettre en place HTTPS (certificat valide et redirection HTTP vers HTTPS) avant toute mise en production. Échéance : avant le déploiement. La mise en production est conditionnée à la levée de cette réserve.
Observations	Pistes d'amélioration hors exigences : jeton anti-CSRF sur les formulaires, réduction de l'en-tête Server, limitation des tentatives de connexion.
Testeur	Nom Prénom — 15/10/2026 — signature
Valideur	Nom Prénom — 15/10/2026 — signature

 Ce que montre cet exemple

- Une application peut **fonctionner** parfaitement (test 1) et rester **vulnérable** (test 4) : la recette doit inclure des tests de sécurité.
- Un test de comportement est renforcé par la **relecture du code** (tests 3 et 6).
- Les contrôles du navigateur se contournent facilement : on teste aussi **directement le serveur** (test 2).
- Une recette peut être prononcée **avec réserves** : l'anomalie est acceptée provisoirement, avec une échéance claire.
- La **version testée** est identifiée, ce qui permet de savoir exactement ce qui a été validé.