IDENTIFANT DE CONNEXION :

Identifiant : admin

Mot de passe : admin

I. Description des fonctionnalités utilisées dans mon projet

1. Utilisation de l'ORM DJANGO

❖ De quoi la fonctionnalité protège le code :

L'utilisation d'un ORM supprime les dépendances de notre code sur un dialecte SQL particulier. Au lieu d'interagir directement avec la BD, nous interagissons avec une couche d'abstraction qui fournit une isolation entre notre code et l'implémentation de la BD. De plus l'ORM offre une protection contre les injections SQL en construisant des requêtes paramétrées.

❖ Comment la fonctionnalité protège le code :

Django fournit un ORM qui permet de se protéger de ce risque en échappant les variables avant de soumettre la requête à la base de données. Tout modèle Django est automatiquement équipé d'un **manager** d'objets, implémenté via la classe « Model » de laquelle hérite tous nos modèles. Ce manager est en fait un attribut du modèle (un composant au sens UML) fournissant des méthodes utiles pour retrouver des objets, créer, supprimer, filtrer etc.

2. Protection contre le « Cross site scripting » (XSS)

❖ De quoi la fonctionnalité protège le code :

Les attaques XSS permettent à un intrus d'injecter des scripts clients des les navigateurs des utilisateurs. Le principe général est de stocker les scripts malveillants dans la base de données d'où ils seront repris et affichés pour d'autres utilisateurs, ou encore d'amener les utilisateurs à cliquer sur un lien qui va provoquer l'exécution du javascript de l'attaquant dans le navigateur des utilisateurs. Cependant les attaques XSS peuvent provenir de n'importe quelle source de données non fiable comme les cookies ou les services web, à chaque fois que les données ne sont pas suffisamment nettoyées avant d'être incluse dans une page.

❖ Comment la fonctionnalité protège le code :

L'utilisation des gabarits de Django protège de la majorité des attaques de XSS. Cependant il est important de comprendre les protections appliquées et leurs limites.

3. Protection contre le « Cross site request forgery » (CSRF)

❖ De quoi la fonctionnalité protège le code :

Les attaques CSRF permettent à une personne malveillante d'exécuter des actions en utilisant des données d'authentification d'un autre utilisateur sans que ce dernier ne s'en rende compte

❖ Comment la fonctionnalité protège le code :
La protection CSRF fonctionne en contrôlant un jeton dans chaque requête POST. Cela garantit qu'un utilisateur malveillant ne peut « rejouer » un envoi de formulaire POST sur votre site Web tout en faisant soumettre ce formulaire de manière involontaire par un autre utilisateur connecté. L'utilisateur malveillant devrait connaître le jeton qui est spécifique à l'utilisateur (en utilisant un cookie).

4. Protection contre l'injection SQL

❖ De quoi la fonctionnalité protège le code :

L’injection SQL est un type d’attaque où un utilisateur malveillant est capable d’exécuter du code SQL arbitraire sur une base de données. Il peut en résulter des suppressions d’enregistrements ou des divulgations de données.

- Comment la fonctionnalité protège le code :

Les jeux de requête de Django sont prémunis contre les injections SQL car leurs requêtes sont construites à l’aide de la paramétrisation des requêtes. Le code SQL d’une requête est défini séparément de ses paramètres. Comme ceux-ci peuvent provenir de l’utilisateur et donc non sécurisés, leur échappement est assuré par le pilote de base de données sous-jacent.

II. Mécanisme et principe de sécurité que mon code implique

1. Mécanismes de sécurités

Comme mécanisme de sécurité, nous avons utilisé dans notre projet l’authentification, les permissions, la validation de données, le chiffrement.

- L’authentification : pour avoir accès à l’application vous devez vous authentifier avec un identifiant et un mot de passe. En fonction de votre profile vous accéder la page qui correspond.
- Les permissions : Django contient un système intégré de permissions. Il permet d’attribuer des permissions à des utilisateurs ou des groupes d’utilisateurs particuliers.
- La validation de données : la fonctionnalité des formulaires de Django peut simplifier et automatiser de larges portions de gestion de formulaires. Ces fonctionnalités sont sécurisées et Django gère trois parties distinctes du travail induit par les formulaires à savoir :
    - Préparation et restructuration des données en vues de leur présentation
    - Création des formulaires HTML pour les données
    - Réception et traitement des formulaires et données envoyés par le client.

- ❖ Le chiffrement : Django ne stocke pas de mots de passe bruts dans le modèle utilisateur, mais uniquement une empreinte hachée. Par défaut Django utilise l'algorithme PBKDF2 avec une fonction de hachage SHA256.

L'implémentation de ces mécanismes de sécurités nous a permis d'établir le principe de moindre confiance et le principe de moindre privilège et de minimiser notre surface d'attaque.

Test : On pourra essayer d'accéder à une page en saisissant l'url de la page mais avec la fonction « login_required » de django et l'authentification cette action ne pourrait pas aboutir.

Pour notre projet nous avons créer un super utilisateur qui a tous les droits et pour accéder à l'application vous aurez besoin des paramètres suivant :

Identifiant : admin

Mot de passe : admin