Les normes OWASP

Les normes OWASP sont un ensemble de recommandations de sécurité pour les développeurs web. Elles sont divisées en plusieurs catégories, dont la catégorie "Applications Web" contient des recommandations spécifiques pour les API RESTful.

Dans une API RESTful Django, les normes OWASP se traduisent par les mesures de sécurité suivantes :

•**Authentification et autorisation:** Il est important d'authentifier et d'autoriser les utilisateurs avant de leur permettre d'accéder aux ressources de l'API. Django fournit des outils pour implémenter l'authentification et l'autorisation, tels que le framework d'authentification Django et le modèle d'autorisation Django.

•**Contrôle d'accès:** Il est important de limiter l'accès aux ressources de l'API aux utilisateurs autorisés. Django fournit des outils pour implémenter le contrôle d'accès, tels que les permissions et les rôles.

•**Sécurité des données:** Il est important de protéger les données sensibles stockées dans l'API, telles que les mots de passe et les informations de carte de crédit. Django fournit des outils pour implémenter la sécurité des données, tels que le cryptage et la validation des données.

•**Filtrage des entrées:** Il est important de filtrer les entrées des utilisateurs pour éviter les attaques par injection SQL et d'autres attaques. Django fournit des outils pour implémenter le filtrage des entrées, tels que le modèle de requête Django.

•**Caching:** Le caching peut être utilisé pour améliorer les performances de l'API et réduire le risque d'attaques. Django fournit des outils pour implémenter le caching, tels que le cache de session Django.

•**Tests de sécurité:** Il est important de tester régulièrement la sécurité de l'API. Django fournit des outils pour implémenter les tests de sécurité, tels que le framework de tests Django.
Voici quelques exemples concrets de mesures de sécurité OWASP appliquées à une API RESTful Django :

- **Utiliser le framework d'authentification Django pour authentifier les utilisateurs avant de leur permettre d'accéder aux ressources de l'API.**
- **Utiliser le modèle d'autorisation Django pour limiter l'accès aux ressources de l'API aux utilisateurs autorisés.**
- **Crypter les mots de passe des utilisateurs stockés dans l'API.**
- **Valider les données des utilisateurs pour éviter les entrées malveillantes.**
- **Utiliser le cache de session Django pour améliorer les performances de l'API et réduire le risque d'attaques.**
- **Exécuter régulièrement des tests de sécurité sur l'API pour identifier et corriger les vulnérabilités.**

En suivant ces recommandations, les développeurs peuvent créer des API RESTful Django plus sécurisées.

Pour voir ces mesures OWASP dans mon code, voici quelques extraits :

Authentification et autorisation

Fichier : permissions.py ligne 158 à 166

```
157
158             # Vérifie si l'utilisateur est l'auteur du projet
159             if user == project.author:
160                 print("# User is the project author : Author is authorized")
161                 return True  # Retourne True pour autoriser l'accès
162
163             # Vérifie si l'utilisateur est un contributeur du projet
164             if Contributor.objects.filter(user=user, project=project).exists():
165                 print("# L'utilisateur est un contributeur du projet : autorisé à lister")
166                 return True  # Retourne True pour autoriser l'accès
```

Fichier : permissions.py ligne 249 à 257

```
249         # Vérifie si l'utilisateur est l'auteur du projet
250         if user == project.author:
251             print("# L'utilisateur est l'auteur du projet : autorisé")  # Affiche un message de débogage
252             return True  # Retourne True pour autoriser l'accès
253
254         # Vérifie si l'utilisateur est un contributeur du projet
255         if Contributor.objects.filter(user=user, project=project).exists():
256             print("# L'utilisateur est un contributeur du projet : "
257                   "autorisé à lister")  # Affiche un message de débogage
```

Fichier : permissions.py ligne 349 à 357

```
349         # Vérification si l'utilisateur est l'auteur du projet
350         if user == project.author:
351             print("# User is the project author. Project author is authorized")
352             return True  # Autorisation accordée
353
354         # Vérification si l'utilisateur est un contributeur du projet
355         if Contributor.objects.filter(user=user, project=project).exists():
356             print("# User is a project contributor : Contributor is authorized to list")
357             return True  # Autorisation accordée
```

Fichier : permissions.py ligne 637 à 645

```
637         # Autorisation si l'utilisateur est l'auteur du projet
638         if user == project.author:
639             print("# User is the project author. User is authorized")
640             return True
641
642         # Autorisation si l'utilisateur est l'auteur du commentaire
643         if user == obj.author:
644             print("# User is the comment's author. User is authorized")
645             return True
```

Ces extraits montrent que j’ai mis en place des mécanismes d'authentification et d'autorisation, et ce en conformité avec l’OASWP

Mécanismes de contrôle d'accès

Fichier : permissions.py ligne 158 à 171

```
158             # Vérifie si l'utilisateur est l'auteur du projet
159             if user == project.author:
160                 print("# User is the project author : Author is authorized")
161                 return True  # Retourne True pour autoriser l'accès
162 
163             # Vérifie si l'utilisateur est un contributeur du projet
164             if Contributor.objects.filter(user=user, project=project).exists():
165                 print("# L'utilisateur est un contributeur du projet : autorisé à lister")
166                 return True  # Retourne True pour autoriser l'accès
167 
168             # Si l'utilisateur n'est ni l'auteur ni un contributeur, il n'est pas autorisé à lister les utilisateurs
169             print("# L'utilisateur n'est pas l'auteur ni un contributeur : non autorisé")  # Affiche un message de
170             # débogage
171             return False  # Retourne False pour refuser l'accès
```

Fichier : permissions.py ligne 336 à 357

```
336         # Vérification si l'action est de lister les 'issues'
337         if view.action == "list":
338             print("# Action : List")
339             print("# Project author or contributor are able to read the issue")
340 
341             # Récupération de l'ID du projet depuis les arguments de la vue
342             project_id = view.kwargs.get('project_id')
343             # Récupération de l'objet Project correspondant à l'ID
344             project = get_object_or_404(Project, id=project_id)
345 
346             # Affichage de l'ID du projet pour le débogage
347             print("# Project_id :", project_id)
348 
349             # Vérification si l'utilisateur est l'auteur du projet
350             if user == project.author:
351                 print("# User is the project author. Project author is authorized")
352                 return True  # Autorisation accordée
353 
354             # Vérification si l'utilisateur est un contributeur du projet
355             if Contributor.objects.filter(user=user, project=project).exists():
356                 print("# User is a project contributor : Contributor is authorized to list")
357                 return True  # Autorisation accordée
```

Fichier : permissions.py ligne 451 à 476

```
451         # Vérification si l'action est de récupérer un 'issue'
452         if view.action == "retrieve":
453             print("# Action : Retrieve")
454             print("# Project author, contributors or issue's author can retrieve")
455 
456             # Récupération de l'ID du projet
457             project_id = view.kwargs.get('project_id')
458             # Récupération du projet associé à cet ID
459             project = get_object_or_404(Project, id=project_id)
460 
461             print("# Project_id :", project_id)
462 
463             # Autorisation si l'utilisateur est l'auteur du projet
464             if user == project.author:
465                 print("# User is the project author. User is authorized")
466                 return True
467 
468             # Autorisation si l'utilisateur est l'auteur de l'issue
469             if user == obj.author:
470                 print("# User is the issue author. User is authorized")
471                 return True
472 
473             # Autorisation si l'utilisateur est un contributeur du projet
474             if Contributor.objects.filter(user=user, project=project).exists():
475                 print("# User is a project contributor. User is authorized")
476                 return True
```

Ces extraits montrent que j’ai mis en place des contrôles d'accès basés sur le rôle de l'utilisateur (auteur du projet, contributeur, etc.), ce qui est en accord avec les meilleures pratiques de sécurité.

Sécurité des données

Utilisation des permissions pour restreindre l'accès aux données :

Fichier : views.py ligne 20 à 25

```
20
21  # Importation des permissions personnalisées
22  from .permissions import PermissionProject
23  from .permissions import PermissionProjectsUsers
24  from .permissions import PermissionIssue
25  from .permissions import PermissionComment
26
27
28  # Vue pour l'enregistrement des utilisateurs
29  # 1. Enregistrement de l'utilisateur - POST - /signup/
30  class SignupAPIView(APIView):
31
32      # Définir le serializer et les permissions pour cette vue
33      serializer_class = SignupSerializer
34      permission_classes = (AllowAny,)
35
```

Fichier : views.py ligne 268 à 298

```
268     permission_classes = [IsAuthenticated, PermissionIssue]
269
270     # 11. GET - Récupère la liste des problèmes liés à un projet
271     def get_queryset(self):
272         print("# IssueViewset: get()")
273
274         # Récupère l'ID du projet depuis les arguments de l'URL
275         project_id = self.kwargs.get('project_id')
276
277         # Filtre les problèmes en fonction de l'ID du projet
278         queryset = Issue.objects.filter(project_id=project_id)
279         return queryset
280
281     # 12. POST - Crée un problème dans un projet
282     def create(self, request, *args, **kwargs):
283         print("# IssueViewset: create()")
284
285         # Vérifie les permissions
286         self.check_permissions(request)
287
288         # Récupère l'ID du projet et l'objet projet correspondant
289         project_id = self.kwargs.get('project_id')
290         project = Project.objects.get(id=project_id)
291
292         # Valide les données envoyées
293         serializer = self.get_serializer(data=request.data)
294         serializer.is_valid(raise_exception=True)
295
296         # Ajoute le projet et l'utilisateur assigné aux données validées
297         serializer.validated_data['project'] = project
298         serializer.validated_data['assignee'] = request.user
```

Ces extraits montrent que j'ai utilisé des permissions pour restreindre qui peut accéder à certaines données, ce qui est une bonne pratique. Cependant, cela ne couvre pas tous les aspects de la "Sécurité des données" tels que le chiffrement des données en transit ou au repos.

Filtrage des entrées et validation par les Serializers

Fichier : views.py ligne 415 à 419

```
414
415         # Initialise le serializer avec les données existantes et les nouvelles données
416         serializer = self.get_serializer(instance, data=request.data, partial=True)
417
418         # Vérifie si les données sont valides
419         serializer.is_valid(raise_exception=True)
```

Fichier : views.py ligne 36 à 48

```
36      # Méthode POST pour créer un nouvel utilisateur
37      def post(self, request):
38          # Récupération des données envoyées dans la requête
39          user = request.data
40          # Initialisation du serializer avec ces données
41          serializer = SignupSerializer(data=user)
42
43          # Vérification de la validité des données
44          if serializer.is_valid():
45              # Enregistrement du nouvel utilisateur dans la base de données
46              serializer.save()
47              # Renvoie une réponse HTTP 201 avec les données de l'utilisateur
48              return Response(serializer.data, status=status.HTTP_201_CREATED)
```

Ces extraits montrent que j'ai utilisé des serializers pour valider les données entrantes, ce qui est une étape importante dans le filtrage des entrées. Cependant, cela ne couvre pas tous les aspects du "Filtrage des entrées", tels que l'assainissement des entrées pour prévenir les attaques comme l'injection SQL.

Le « Caching »

Fichier : views.py ligne 73 à 81

```
73      # 3. GET - Récupérer la liste de tous les projets associés à l'utilisateur connecté
74      def get_queryset(self):
75          print("################################################")
76          print("# Views.py : ProjectsViewset: get()")
77          print("# L'utilisateur doit être authentifié.")
78          print("# Classe de permission limitée : PermissionProject")
79
80          # Récupération de l'utilisateur connecté
81          user = self.request.user
```

Fichier : views.py ligne 342 à 354

```
342  # 15. POST - Créer des commentaires sur un problème
343  # 16. GET - Récupérer la liste de tous les commentaires liés à un problème
344  # 17. PUT - Éditer un commentaire
345  # 18. DELETE - Supprimer un commentaire
346  # 19. GET - Obtenir un commentaire via son identifiant
347
348
349  # Classe pour gérer les vues associées aux commentaires d'un problème (Issue)
350  class CommentViewset(ModelViewSet):
351
352      # Définit les objets et le serializer
353      queryset = Comment.objects.all()
354      serializer_class = CommentSerializer
```

Ces extraits montrent que j’ai utilisé des vues basées sur des classes pour gérer les requêtes HTTP, mais il n'y a pas de mention explicite de la mise en cache des données.

Les tests de sécurité :

J’ai créé une collection de texts complete sur POSTMAN avec sa documentation mesurant plus de 70 tests confirmés afin de réaliser que plus de doute sur la sécurisation de cette API RESTful de DJANGO.
Voici l’url de ma documentation sur POSTMAN

En résumé, mon code démontre qu’il est bien structuré et suit plusieurs des meilleures pratiques de sécurité OWASP, mais il manque des éléments spécifiques liés à la "Sécurité des données", comme le "Caching",

Implémentation du RGPD

Pour implémenter le RGPD dans une API RESTful Django, il faut suivre les recommandations suivantes :

•**Respecter les principes du RGPD:** Les principes du RGPD sont la base de l'ensemble de la réglementation. Il est important de les respecter dans la conception et la mise en œuvre de l'API.

•**Mettre en place des mesures techniques et organisationnelles appropriées:** Le RGPD exige que les responsables du traitement des données mettent en place des mesures techniques et organisationnelles appropriées pour protéger les données personnelles.

•**Informer les personnes concernées:** Les personnes concernées ont le droit d'être informées de leurs droits et de la manière dont leurs données personnelles sont traitées.

•**Permettre aux personnes concernées d'exercer leurs droits:** Les personnes concernées ont le droit d'accéder à leurs données personnelles, de les rectifier, de les effacer, de limiter leur traitement, de s'opposer à leur traitement et de demander la portabilité de leurs données.

Voici quelques exemples concrets de mesures RGPD appliquées à une API RESTful Django :

- **Utiliser un modèle de données pour représenter les données personnelles.**

- **Mettre en place une authentification et une autorisation appropriées pour limiter l'accès aux données personnelles.**
- **Crypter les données personnelles lors de leur stockage et de leur transfert.**
- **Générer des journaux d'activité pour suivre les accès aux données personnelles.**
- **Mettre en place un mécanisme pour permettre aux personnes concernées d'exercer leurs droits.**

En suivant ces recommandations, les développeurs peuvent créer des API RESTful Django conformes au RGPD.

Voici quelques mesures spécifiques que vous pouvez prendre pour implémenter le RGPD dans votre API RESTful Django :

- **Ajoutez une politique de confidentialité à votre API.** Cette politique doit expliquer comment vous collectez, utilisez et stockez les données personnelles.
- **Mettez en place un mécanisme pour permettre aux personnes concernées de demander l'accès à leurs données personnelles.** Ce mécanisme peut prendre la forme d'une API ou d'un formulaire web.
- **Mettez en place un mécanisme pour permettre aux personnes concernées de demander la rectification de leurs données personnelles.** Ce mécanisme peut prendre la forme d'une API ou d'un formulaire web.
- **Mettez en place un mécanisme pour permettre aux personnes concernées de demander l'effacement de leurs données personnelles.** Ce mécanisme peut prendre la forme d'une API ou d'un formulaire web.
- **Mettez en place un mécanisme pour permettre aux personnes concernées de s'opposer au traitement de leurs données personnelles.** Ce mécanisme peut prendre la forme d'une API ou d'un formulaire web.
- **Mettez en place un mécanisme pour permettre aux personnes concernées de demander la portabilité de leurs données personnelles.** Ce mécanisme peut prendre la forme d'une API ou d'un formulaire web.

Il est important de noter que ces mesures ne sont que des exemples. Le niveau de conformité RGPD de votre API dépendra de la nature des données personnelles que vous collectez et traitez.

GREEN CODE

Pour démontrer que le code de votre projet (API RESTful Django) respecte la philosophie « Green Code », vous pouvez suivre les étapes suivantes :

1.**Évaluez votre code en fonction des principes de la philosophie « Green Code ».** Les principes de la philosophie « Green Code » sont les suivants :

- **Efficacité:** Le code doit être efficace, c'est-à-dire qu'il doit utiliser les ressources de manière optimale.
- **Simplicité:** Le code doit être simple, c'est-à-dire qu'il doit être facile à comprendre et à maintenir.
- **Maintenabilité:** Le code doit être maintenable, c'est-à-dire qu'il doit être facile à modifier et à améliorer.
- **Robustesse:** Le code doit être robuste, c'est-à-dire qu'il doit être capable de résister aux erreurs et aux changements.
- **Portabilité:** Le code doit être portable, c'est-à-dire qu'il doit pouvoir être utilisé sur différents environnements.

2.**Identifiez les domaines d'amélioration.** Une fois que vous avez évalué votre code, identifiez les domaines d'amélioration. Ces domaines d'amélioration peuvent être des points de non-conformité aux principes de la philosophie « Green Code », ou des domaines qui pourraient être améliorés pour rendre le code plus efficace, simple, maintenable, robuste ou portable.

3.**Appliquez les améliorations.** Une fois que vous avez identifié les domaines d'amélioration, appliquez les améliorations. Vous pouvez utiliser des outils et des techniques pour vous aider à améliorer votre code.

4.**Testez le code.** Une fois que vous avez appliqué les améliorations, testez le code pour vous assurer qu'il fonctionne correctement.

5.**Documentez les améliorations.** Documentez les améliorations que vous avez apportées au code. Cela vous aidera à suivre les changements et à vous assurer que les améliorations sont maintenues.

Voici quelques exemples concrets de mesures que vous pouvez prendre pour rendre votre code plus respectueux de l'environnement :

- **Utiliser des modèles de données efficaces.** Les modèles de données doivent être conçus de manière à utiliser les ressources de manière optimale. Par exemple, vous pouvez utiliser des types de données appropriés pour stocker les données.
- **Éviter les répétitions.** Les répétitions dans le code peuvent rendre le code moins efficace et plus difficile à maintenir. Vous pouvez utiliser des techniques de réutilisation du code pour éviter les répétitions.
- **Utiliser des bibliothèques et des frameworks éprouvés.** Les bibliothèques et les frameworks éprouvés sont généralement plus efficaces et plus maintenables que le code personnalisé.
- **Développer des tests unitaires.** Les tests unitaires vous aideront à identifier les erreurs et les problèmes de performances dans votre code.
- **Utiliser un système de gestion de version.** Un système de gestion de version vous aidera à suivre les changements dans votre code et à revenir à une version précédente si nécessaire.

En suivant ces étapes, vous pouvez démontrer que le code de votre projet (API RESTful Django) respecte la philosophie « Green Code ».

Voici quelques outils et techniques que vous pouvez utiliser pour améliorer votre code :

•**Linter:** Un linter est un outil qui analyse le code pour détecter les erreurs et les problèmes potentiels.

•**Code coverage:** La couverture de code est une mesure de la quantité de code qui est testée par les tests unitaires.

•**Static analysis:** L'analyse statique est une technique qui analyse le code sans le compiler. Elle peut être utilisée pour détecter les erreurs et les problèmes potentiels.

•**Code review:** Une revue de code est une procédure dans laquelle un code est examiné par un autre développeur.

En utilisant ces outils et techniques, vous pouvez améliorer la qualité et la performance de votre code.