SoftDesk
Support

Exigences de sécurité et d'optimisation

Open Worldwide Application Security Project (OWASP) Top 10

L'OWASP Top 10 est la liste des vulnérabilités de sécurité des applications web les plus couramment rencontrées dans les applications et les API existantes. Les risques sont classés de A1 à A10, A1 étant le risque le plus fréquent.

Vous trouverez le top 10 des vulnérabilités actuelles sur le site officiel de l'OWASP.

Toutes ces vulnérabilités peuvent être encore regroupées selon le processus **AAA** *(*en anglais, *Authentication, Authorization, Accounting* ; en français, authentification, autorisation, traçabilité) des protocoles réseau.

Nous allons donc traiter ces 3 A en mettant en œuvre ces bibliothèques ou méthodes dans notre application :

1. **Authentification :** utilisez JWT (JSON Web Token) pour le back-end d'authentification du framework Django REST. Cela vise à couvrir les cas d'utilisation de JWT les plus répandus, en offrant par défaut un jeu de fonctionnalités prudent. Pour cela, vous pouvez utiliser cette ressource ;
2. **Autorisation :** la deuxième étape est l'autorisation, dans laquelle le back-end décide si *l'utilisateur authentifié* est autorisé à accéder à une ressource. Dans notre application, un utilisateur ne doit pas être autorisé à accéder à un projet pour lequel il n'est pas ajouté en tant que contributeur. De même, un contributeur ou un utilisateur doit toujours être connecté pour accéder à une fonctionnalité.
    1. Seuls les utilisateurs authentifiés doivent être en mesure d'accéder à quoi que ce soit dans l'application. Utilisez Django REST APIView et la classe d'autorisation pour vérifier que l'utilisateur est authentifié. Vous pouvez vous référer à la documentation Permissions de Django REST Framework.
    2. Vous aurez besoin d'ajouter des autorisations à vos modèles en spécifiant l'auteur (clé étrangère de la documentation Permissions d'utilisateur) de ce projet/problème/commentaire.

3. **Accès :** même après que l'utilisateur a été authentifié et autorisé à accéder à une ressource, la ressource elle-même peut nécessiter des autorisations spéciales, par exemple pour actualiser ou supprimer un commentaire.
    1. Les commentaires doivent être visibles par tous les contributeurs au projet et par le responsable du projet, mais seul leur auteur peut les actualiser ou les supprimer.
    2. Un problème ne peut être actualisé ou supprimé que par son auteur, mais il doit rester visible par tous les contributeurs au projet.
    3. Il est interdit à tout utilisateur autre que l'auteur de demander une mise à jour et de supprimer des demandes sur une question/un projet/un commentaire.
4. **Gestion des dépendances :** les dépendances Python peuvent contenir des failles et des vulnérabilités, et il est important de les garder à jour. Pour cela, nous utiliserons Pipenv ou Poetry afin de faciliter les mises à jour et l'interdépendance des bibliothèques tierces.

La protection des données utilisateur grâce au RGPD

Le **Règlement Général sur la Protection des Données** est un ensemble de règles qui encadre la collecte et l'utilisation des données utilisateur (nom, prénom, âge, numéro de téléphone…). Il s'agit avant tout de **sécuriser** les données personnelles et de **respecter le droit des utilisateurs** sur leurs données. Le RGPD concerne l'ensemble de l'Union européenne.

Vous trouverez ici le guide pratique du CNIL, qui est très complet.

Dans le cadre de notre application, nous respecterons quelques règles de sécurisation :

1. L'utilisateur a le **droit à l'accès et à la rectification** de son profil. Il doit pouvoir récupérer ses informations personnelles et les rectifier au besoin.
2. Le **droit à l'oubli** : un utilisateur doit pouvoir supprimer ses données personnelles sans qu'il reste aucune subsistance dans l'application.
3. Collecter le **consentement** : l'utilisateur peut donner ou non son consentement pour être contacté ou partager ses données personnelles.
4. **Vérification de l'âge**: l'âge légal pour donner son consentement seul est de 15 ans. Nous collectons et vérifions donc l'âge de l'utilisateur pour valider son inscription.

Green code

SoftDesk s'inscrira dans les enjeux décisifs de notre ère en proposant aux clients une application qui répondra à leurs attentes tout en réduisant son impact environnemental. En effet, l'application devra être pensée et structurée en considération des enjeux énergétiques et climatiques actuels.

Dans le cadre de notre application, il est tout de même possible, dans une démarche de green code, d'éviter les pièges les plus grossiers et prévisibles, en intégrant **la**

pagination des ressources. Nous savons que nos clients vont créer une multitude de ressources et sans pagination, notre serveur va vite surchauffer.

Attention toutefois au piège du **« greenwashing » :** du fait de ses datacentres et de sa consommation électrique, Internet pollue. SoftDesk a conscience de cette dimension indéniable et ne prétend pas la résoudre. L'application ne sera donc jamais « zéro émission », ce qui ne l'empêchera toutefois pas de questionner son approche afin de minimiser son impact environnemental quand c'est possible, chaque fois que c'est possible.

D'ailleurs, le concept de green code est certes récent, mais reprend en fin de compte des fondamentaux de la programmation traditionnelle à travers le concept crucial, à terme, **d'optimisation du code.** Voici une petite présentation de cette notion clef :

L'optimisation du code web ? Comprendre ses différentes applications :

- L'optimisation du **code source** : lorsque la taille des fichiers de code et des médias a été revue et minimisée ;
- L'optimisation des **requêtes web** : lorsque le parcours utilisateur est optimisé, les requêtes ne sont pas trop lourdes et/ou trop nombreuses ;
- Une gestion intelligente de l'hébergeur : lorsque l'hébergeur possède les ressources adaptées au site qu'il héberge ;
- Une **gestion intelligente de la CI** (Continuous Integration) : vous le verrez plus tard, mais l'intégration continue peut aussi prendre beaucoup de ressources lors de la génération de tests automatisés. Elle est à optimiser.

Par ailleurs, le domaine de l'optimisation du code a permis l'émergence d'outils et de concepts très variés. On pense notamment au **système de mise en cache des ressources** au niveau du navigateur ou du serveur web. Ce système permet de sauvegarder la ressource pour éviter de la régénérer.

Cependant, l'optimisation est souvent pointée du doigt comme le fléau du développeur. La PEP20 de Python (un ensemble de règles « philosophiques » sur le langage) le dit clairement : « l'optimisation est la racine du mal. » Il faut entendre par là qu'il est inutile – et même contre-productif – de **vouloir optimiser son code avant même de rencontrer des problèmes** liés à un manque d'optimisation. Comme beaucoup de pratiques en programmation, l'optimisation est la **solution correspondant à un problème donné**, et non une pratique à automatiser, car de fait, elle aura toujours tendance à **complexifier le code**.