

# API REST en Python

# Table des matières

<b>I. API</b>	<b>3</b>
A. Utilité d'une API REST .....	3
B. Installation Django REST .....	4
<b>II. Exercice : Quiz</b>	<b>5</b>
<b>III. Django REST</b>	<b>6</b>
A. Sérialiseur .....	6
B. Affichage de l'API .....	8
<b>IV. Exercice : Quiz</b>	<b>11</b>
<b>V. Sécurité OWASP</b>	<b>12</b>
A. Introduction à la cybersécurité.....	12
B. Django-REST et sécurité.....	14
<b>VI. Exercice : Quiz</b>	<b>16</b>
<b>VII. Essentiel</b>	<b>17</b>
<b>VIII. Auto-évaluation</b>	<b>17</b>
<b>Solutions des exercices</b>	<b>18</b>

## I. API

### Contexte

Une API Web permet aux informations ou aux fonctionnalités d'être manipulées par d'autres programmes via Internet. Par exemple, avec l'API Web de Twitter, vous pouvez écrire un programme dans un langage comme Python, qui peut effectuer des tâches telles que la création de favoris de tweets ou la collecte de métadonnées de tweets.

De manière générale, en programmation, le terme API (*Application Programming Interface*) fait référence à une partie d'un programme informatique qui a pour fonction d'être utilisée par un autre programme. Au contraire d'une interface qui, elle, est conçue pour être utilisée ou manipulée par un humain. Les API permettent de faire communiquer entre eux les différents programmes informatiques ou avec le système d'exploitation.

### A. Utilité d'une API REST

En général, on utilise une API Web quand :

- L'ensemble de données est trop volumineux pour un être humain, ce qui rend le téléchargement coûteux en ressources.
- Vos utilisateurs devront accéder à vos données en temps réel, comme pour Twitter ou Instagram.
- Vos données sont mises à jour fréquemment.

Si vous avez des données que vous souhaitez partager avec le monde, une API est un moyen de les mettre entre les mains d'autres personnes. Cependant, les API ne sont pas toujours le meilleur moyen de partager des données avec les utilisateurs. Si la taille des données que vous fournissez est relativement petite, vous pouvez à la place fournir un « *vidage de données* » sous la forme d'un fichier JSON, XML, CSV ou SQLite téléchargeable.

### Remarque

Lors de l'utilisation ou de la création d'API, vous rencontrerez fréquemment le terme HTTP (*HyperText Transfer Protocol*) : c'est le principal moyen de communication des données sur le Web. HTTP implémente un certain nombre de fonctions qui indiquent comment les données se déplacent et ce qui devrait leur arriver. Les deux méthodes les plus courantes sont **GET** et **POST**, que vous avez déjà dû rencontrer.

### Définition

L'URL (*Uniform Resource Locator*) est une adresse pour une ressource sur le Web. Une URL se compose de 3 éléments :

- Un protocole (*http://*)
- Un domaine (*programmationhistorian.org*)
- Un chemin facultatif (*/about*)

Une URL décrit l'emplacement d'une ressource spécifique, telle qu'une page web. Lors de la lecture sur les API, vous pouvez voir les termes « *URL* », « *demande* », « *URI* » ou « *point de terminaison* » utilisés pour décrire les idées adjacentes. Vous pouvez suivre une URL ou effectuer une demande **GET** dans votre navigateur.

JSON (*JavaScript Object Notation*) est un format de stockage de données basé sur du texte conçu pour être facile à lire pour les humains et les machines. JSON est généralement le format le plus courant pour renvoyer des données via une API, XML étant le deuxième.

REST (*Representational State Transfer*) est une théorie qui décrit quelques bonnes pratiques pour l'implémentation d'API. Les API conçues avec ces principes sont appelées API REST. Bien que l'API qu'on va construire utilise certains principes REST, il y a beaucoup de désaccords autour de ce terme.

#### Exemple

Imaginons que notre domaine de recherche soit la presse : la couverture journalistique des grands événements aux États-Unis est-elle devenue plus ou moins sensationnelle au fil du temps ? En précisant le sujet, nous pourrions nous demander si la couverture médiatique, par exemple, des incendies urbains a augmenté ou diminué avec les rapports du gouvernement sur les dépenses de secours liées aux incendies.

La documentation est le point de départ d'un utilisateur lorsqu'il travaille avec une nouvelle API, des URL bien conçues permettent aux utilisateurs de trouver plus facilement des ressources de manière intuitive, car cela facilite l'accès aux informations via notre API.

## B. Installation Django REST

Django REST permet de faciliter la sérialisation. En effet, dans Django, vous définissez vos modèles pour votre base de données à l'aide de Python. Bien que vous puissiez écrire du SQL brut, la plupart du temps, Django ORM gère toutes les migrations et requêtes de base de données. Le framework Django REST fonctionne avec l'ORM Django qui s'occupe de la partie interrogation de base de données. Grâce à quelques lignes de code, en utilisant Django REST framework, vous pouvez sérialiser vos modèles de base de données aux formats RESTful.

#### Définition

Les sérialiseurs de Django REST convertissent les instances de modèle en dictionnaires Python, qui peuvent ensuite être rendus dans divers formats appropriés pour l'API, tels que JSON ou XML. Semblable à la classe Django **ModelForm**, Django REST est livré avec un format concis pour ses sérialiseurs, la classe **ModelSerializer**.

#### Exemple

Nous allons commencer par installer Django REST. Il faut aller dans le dossier racine de votre *virtual environment* :

```
pip install djangorestframework
```

Une fois cette première étape effectuée, il vous suffit d'aller dans votre projet Django et de l'ajouter dans **setog.py** « **INSTALLED\_APP** » :

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'rest_framework',  
]
```

## Exercice : Quiz

[solution n°1 p.19]

### Question 1

Quand est-ce qu'on utilise une API Web ?

- ☐ Quand l'ensemble des données est trop volumineux pour un être humain, ce qui rend le téléchargement coûteux en ressources.
- ☐ Quand vos utilisateurs devront accéder à vos données en temps réel.
- ☐ Quand vos données sont mises à jour fréquemment..
- ☐ Il est préférable de ne jamais utiliser API Web.

### Question 2

Qu'est-ce que HTTP ?

- ☐ Hyperlink Text Protocol
- ☐ Hypertext Transport Protocol
- ☐ Hypertext Transbordement Protocol
- ☐ Hypertext Transfer Protocol

### Question 3

Que veut dire URL ?

- ☐ Uniform Resource Locator
- ☐ Unique Reforme Location
- ☐ Unique Resource Location
- ☐ Uniform Reforme Locator
- ☐ Uniform Resource Location

### Question 4

Sélectionnez ce qui est juste sur l'URL

- ☐ Une URL se compose d'un protocole (http://).
- ☐ Une URL possède forcément un chemin.
- ☐ Une URL décrit l'emplacement d'une ressource spécifique.
- ☐ Lors de la lecture sur les API, il est impossible de voir les termes URL.

Question 5

REST (*Representational State Transfer*) est une théorie qui décrit quelques bonnes pratiques pour l'implémentation d'API.

- ☐ Vrai
- ☐ Faux

### III. Django REST

#### A. Sérialiseur

Nous allons créer une base de données de livres. Voici quelques détails : pour chacun des livres, nous avons un titre, un numéro de volume, ainsi qu'un auteur. Les informations ci-dessous nous permettent de construire notre modèle :

```
from django.db import models
from django.core.validators import MinValueValidator

# Create your models here.
class Livre(models.Model):
    titre = models.CharField(max_length=120)
    volume = models.IntegerField(validators=[MinValueValidator(0)])
    auteur = models.CharField(max_length=120)

    def __str__(self):
        return "%s %s" % (self.titre, self.volume)
```

#### Définition

Les sérialiseurs de Django REST permettent de convertir les instances de modèle en dictionnaires Python. Par la suite, ils peuvent être affichés sur plusieurs formats appropriés pour l'API comme JSON ou XML. Tout comme la classe Django **ModelForm**, Django REST dispose d'un format concis pour ses sérialiseurs : la classe **ModelSerializer**.

Les avantages des sérialiseurs sont les suivants : gain de temps, de lisibilité, de solidité, mais également et surtout, une simplicité d'utilisation.

## Fondamental

Il suffit d'indiquer les champs que vous souhaitez utiliser à partir du modèle. Commençons par créer un fichier `serializers.py` dans notre application, ensuite nous allons créer notre premier sérialiseur :

```
from rest_framework import serializers
from .models import Livre

class LivreSerializer(serializers.Serializer):
    titre = serializers.CharField(max_length=120)
    volume = serializers.IntegerField()
    auteur = serializers.CharField(max_length=120)

    def create(self, data):
        return Livre.objects.create(data)

    def update(self, obj):
        return obj
```

Ici, notre sérialiseur va servir à transmettre les données de notre classe `Livre`. Nous allons maintenant utiliser `LivreSerializer` pour sérialiser un livre ou une liste de livres. Regardons dans le shell :

```
>>> from rest_framework.parsers import JSONParser
>>> from livres.serializers import LivreSerializer
>>> from livres.models import Livre
>>> from rest_framework.renderers import JSONRenderer
>>> from rest_framework.parsers import JSONParser
>>> Livre.objects.create(titre="Le Nom du vent", volume=1, auteur="Patrick Rothfuss")

>>> a = Livre.objects.get(id=1)
>>> s = LivreSerializer(a)
>>> s.data
{'titre': 'Le Nom du vent', 'volume': 1, 'auteur': 'Patrick Rothfuss'}
```

Nous avons converti nos données de sorte à ce que Python puisse comprendre ce qu'il reçoit. L'avantage de Django REST est le formatage de données à partir d'objets natifs de Python. `JSONReoder` va nous servir à focaliser la sérialisation au format JSON, un format assez courant :

```
>>> gson = JSONRenderer().render(s.data)
>>> gson
b'{"titre": "Le Nom du vent", "volume": 1, "auteur": "Patrick Rothfuss"}'
```

Lorsque vous allez récupérer les données, elles arriveront donc au format JSON et vous serez donc obligé de dé-sérialiser les données. C'est ici que `JSONParser` va nous être utile :

```
>>> import io
>>> content = JSONParser().parse(io.BytesIO(gson))
>>> content
{'titre': 'Le Nom du vent', 'volume': 1, 'auteur': 'Patrick Rothfuss'}
```

La meilleure manière de créer des sérialiseurs qui correspondent étroitement aux définitions du modèle Django est la classe `ModelSerializer`, qui fournit un raccourci qui vous permet de créer automatiquement une classe `Serializer` avec des champs qui correspondent aux champs du modèle.

```
from rest_framework import serializers
from .models import Livre

class LivreSerializer(serializers.ModelSerializer):
    class Meta:
        model = Livre
        fields = ['titre', 'volume', 'auteur']
```

### Méthode

Avec cette méthode, vous pouvez afficher toutes les caractéristiques des champs :

```
>>> from livres.models import Livre
>>> from livres.serializers import LivreSerializer
>>> s = LivreSerializer()
>>> print(repr(s))
LivreSerializer():
  titre = CharField(max_length=120)
  volume = IntegerField(min_value=0)
  auteur = CharField(max_length=120)
```

Les sérialiseurs sont très utiles pour générer de la documentation d'API, les outils comme Swagger ou la Browsable API s'en servent pour faire de l'auto-documentation, par exemple pour les signatures d'entrées / sorties. Django possède aussi un système de gestion d'exception, qui est plus simple avec l'utilisation d'un sérialiseur.

## B. Affichage de l'API

### Fondamental

Nous allons ici afficher nos données sur une page. Pour ce faire, nous utilisons un style de vue qui est implémenté par Django REST.

Le framework Django REST fournit une classe `APIView`, qui sous-classe la classe `View` de Django. Les classes `APIView` diffèrent des classes `View` standards.

### Méthode

Tout d'abord, les requêtes passées aux méthodes du gestionnaire ne sont pas des instances `HttpRequest` de Django, mais des instances `Request` du framework REST.

Les méthodes de gestion peuvent renvoyer une réponse du framework REST, au lieu d'une `HttpResponse` de Django classique. De plus, la vue s'occupe de gérer le format du contenu automatiquement et de définir le bon rendu de la réponse. Chaque exception `APIException` est interceptée et donne une réponse au format approprié.

Pour utiliser la classe `APIView`, nous pouvons reprendre l'utilisation d'une classe `View` standard. En effet, le principe est similaire : la requête entrante est envoyée à une méthode de gestion appropriée, telles que `.get()` ou `.post()`. D'autre part, un grand nombre d'attributs peuvent être définis sur la classe, ce qui offre la possibilité d'être plus agile vis-à-vis des principes REST.



**Exemple**

Nous allons commencer par créer notre classe, qui va hériter de la classe `APIView`, et nous allons lui donner une méthode `get`. Nous allons l'effectuer dans le fichier `view.py` comme ci-dessous :

```
from django.shortcuts import render
from rest_framework.views import APIView
from .models import Livre
from .serializers import LivreSerializer
from rest_framework.response import Response
# Create your views here.

class MyView(APIView):
    def get(self, request):
        livres = Livre.objects.all()
        serializer = LivreSerializer(livres, many=True)
        return Response(serializer.data)
```

Nous allons refaire exactement ce que nous avons fait sur notre shell. La seule différence, c'est qu'ici, nous retournons un objet `Response` de Django REST avec les données du sérialiseur. Si nous lançons notre serveur et que nous essayons d'accéder aux données, nous pourrions voir ce qu'il y a dans notre API. Pour ce faire, nous allons ajouter un lien dans notre fichier `url.py`:

```
from django.contrib import admin
from django.urls import path
from livres.views import MyView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('livre/', MyView.as_view())
]
```

Nous allons tout faire migrer et lancer notre serveur pour avoir accès à la page web :

```
python3 manage.py makemigrations
python3 manage.py migrate
```

```
python3 manage.py runserver
```

Si vous vous rendez sur la page « `.../livre` », vous devriez voir tout ce qui se trouve dans la table `livre`. Créons une nouvelle classe `MyViewDetailed` : nous allons ajouter certaines fonctions suivant les principes REST. Chaque action des `APIView` porte le même nom de la requête SQL qui leur est propre.

Voici ce que cela donne :

```
class MyViewDetailed(APIView):
    def get_object(self, id):
        try:
            return Livre.objects.get(id=id)
        except Livre.DoesNotExist:
            return Response(Livre.objects.get(id=id).errors)

    def get(self, request, id):
        livre = self.get_object(id)
        serializer = LivreSerializer(livre, many=False)
        return Response(serializer.data)

    def put(self, request, id):
        livre = self.get_object(id)
        serializer = LivreSerializer(livre, data=request.data)

        if(serializer.is_valid()):
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    def delete(self, request, id):
        livre = self.get_object(id)
        livre.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```

Pour récupérer l'ID Django REST, nous pouvons aller la chercher dans l'URL. Pour cela, il faut lui dire comment procéder. Il suffit d'ajouter dans `url.py`, comme ceci :

```
path('detail/<int:id>', MyViewDetailed.as_view()),
```

#### Définition

Nous retrouvons dans cette classe 4 fonctions, chacune d'entre elles ayant une fonction bien spécifique :

- `get_object()` : cette fonction récupère un objet dont l'ID est placé dans l'URL.
- `get()` : cette méthode va juste utiliser `get_object` et renvoyer la réponse au serveur.
- `put()` : cette méthode sert à mettre à jour les données de l'objet.
- `delete()` : elle permet de supprimer l'objet.

Il existe également une autre méthode que `APIView` : la classe `ViewSet`. Une classe `ViewSet` se définit par un type de `View` basé sur une classe, elle ne fournit pas de gestionnaire de méthode comme `.get()` ou `.post()`, mais plutôt des actions comme `.list()` et `.create()`. Les gestionnaires de méthode pour un `ViewSet` sont entièrement liés aux actions qui correspondent au moment de la finalisation de la vue. Cela est permis grâce à la méthode `.as_view()`.

#### Remarque

En général, au lieu d'enregistrer les vues au sein d'un ensemble de vues dans l'`urlpatterns`, il faut enregistrer l'ensemble de vues avec une classe de routeur. Celle-ci détermine de manière automatique l'`urlpatterns`. Il est également possible de trouver des implémentations similaires dans d'autres frameworks, appelés « Ressources » ou bien « Contrôleurs ».

**Fondamental**

Les `ViewSet` possèdent des méthodes différentes des requêtes SQL que vous pouvez remplacer (Override). Si vous oubliez ces actions, vous pouvez aller sur le site de Django REST pour les retrouver :

```
class MyViewSet(viewsets.ViewSet):

    def list(self, request):
        pass

    def create(self, request):
        pass

    def retrieve(self, request, pk=None):
        pass

    def update(self, request, pk=None):
        pass

    def partial_update(self, request, pk=None):
        pass

    def destroy(self, request, pk=None):
        pass
```

Définissons un `viewset` simple qui peut être utilisé pour lister ou récupérer tous les livres du système. On va donc combler les fonctions `list` et `retrieve` :

```
def list(self, request):
    livres = Livre.objects.all()
    serializer = LivreSerializer(livres, many=True)

    return Response(serializer.data)

def retrieve(self, request, id=None):
    livres = Livre.objects.all()
    livre = get_object_or_404(livres, id=id)
    serializer = LivreSerializer(livre)
    return Response(serializer.data)
```

Une particularité de cette version est que l'on utilise une fonction de Django `get_object_or_404` qui est utile quand on ne sait pas si l'objet existe.

**Exercice : Quiz**

[solution n°2 p.20]

## Question 1

Les sérialiseurs de Django REST :

- ☐ Servent à convertir les instances de modèle en dictionnaire Python.
- ☐ Ne sont pas conseillés.
- ☐ Créent un modèle en Django.
- ☐ Créent une vue en Django.

## Question 2

À quoi sert la méthode **put ()** ?

- ☐ Cette méthode sert à mettre à jour les données de l'objet.
- ☐ Supprime l'objet.
- ☐ Cette méthode va juste utiliser **get\_object** et renvoyer la réponse au serveur.
- ☐ Cette fonction récupère un objet dont l'ID est placé dans l'URL.

## Question 3

À quoi sert la méthode **get ()** ?

- ☐ Cette méthode sert à mettre à jour les données de l'objet.
- ☐ Supprime l'objet.
- ☐ Cette méthode va juste utiliser **get\_object** et renvoyer la réponse au serveur.
- ☐ Cette fonction récupère un objet dont l'ID est placé dans l'URL.

## Question 4

Qu'est-ce que JSON ?

- ☐ JavaScript Object Notation
- ☐ Java Object Notation
- ☐ JavaScript Object Note

## Question 5

**JSONRender** sert à finaliser la sérialisation au format JSON.

- ☐ Vrai
- ☐ Faux

## V. Sécurité OWASP

### A. Introduction à la cybersécurité

**Fondamental**

La sécurité web, également connue sous le nom de « *cybersécurité* », consiste essentiellement à la protection d'un site web ou d'une application web en détectant, prévenant et répondant aux cybermenaces. Les sites web et les applications web sont tout aussi sujets aux failles de sécurité que les maisons physiques, les magasins et les sites gouvernementaux. Malheureusement, la cybercriminalité se produit tous les jours et de grandes mesures de sécurité web sont nécessaires pour empêcher que les sites web et les applications web ne soient compromis.

La mise en place de pratiques sécurisées dans une entreprise peut coûter très cher. Toutes les entreprises ont des actifs à protéger. Les actifs peuvent être des équipements physiques tels que des serveurs et des téléphones, mais il peut également s'agir de données très importantes que l'entreprise doit protéger. La gestion des risques en entreprise consiste à déterminer la valeur d'un actif et le coût de sa sécurisation.

**Conseil**

L'*Open Web Application Security Project (OWASP)* est une fondation à but non-lucratif qui a pour but d'améliorer la sécurité des logiciels en prônant une philosophie d'ouverture pour toute personne. Cette fondation ou communauté en ligne est une source pour chaque développeur afin de sécuriser le Web.

Il existe trois principes de base en sécurité : confidentialité, intégrité et disponibilité.

- **La confidentialité** est l'assurance que les personnes non autorisées n'accèdent pas aux informations sensibles.
- **L'intégrité** est l'assurance que les données sont fiables et n'ont pas été modifiées par des personnes non autorisées.
- **La disponibilité** signifie qu'il n'y a pas de perturbation d'un service ou d'accessibilité aux données.

Toute application qui gère des données sensibles risque d'être attaquée et nécessite une sécurité standardisée. D'autres entreprises qui ne gèrent pas de données sensibles, mais qui ont besoin d'une plateforme web pour faire des affaires, sont également exposées au risque d'attaque. Si une attaque faisait tomber le site, cela affecterait également les revenus de l'entreprise.

Le top 10 de l'OWASP est une liste qui permet de visualiser les 10 principales vulnérabilités pour les applications web. Ce document permet de sensibiliser à la sécurité et regroupe les risques les plus répandus sur les applications web.

Nous allons regarder ensemble chacune des vulnérabilités que nous pouvons rencontrer :

**L'injection SQL** ou A1 : il s'agit ici d'une faille liée à l'injection, comme l'injection SQL, NoSQL, OS ou encore LDAP. Ce type de faille se produit lorsque certaines données non approuvées sont transmises à un interpréteur dans le cadre d'une requête ou d'une commande. Le hacker peut ici inciter l'interpréteur à exécuter des commandes de manière involontaire ou bien à accéder directement à des informations sans aucune autorisation préalable.

**Exemple Injection SQL**

```
SELECT salarié.nom, entreprise.nom  
FROM salarié JOIN entreprise  
ON salarié.num_entreprise = entreprise.num_entreprise  
AND (entreprise.nom = 'Studi' OR entreprise.nom = 'Ditesco')
```

- **Le piratage de session** ou A2 : dans ce cas de figure, les fonctions qui permettent de s'authentifier, comme les mots de passe, les clés ou encore les jetons de sessions, sont en général mal implantés. Cela permet au hacker de les compromettre afin de prendre l'identité d'un autre utilisateur.
- **L'exposition aux données sensibles** ou A3 : un grand nombre d'applications web et d'API ne sécurisent pas les données sensibles. Ces données sensibles sont principalement les données financières, comme les données de carte de crédit, ou bien les données médicales et autres informations personnelles. Le hacker peut donc s'en emparer pour utiliser les données d'identité ou bancaires et ainsi effectuer une fraude. Ce type de données dites sensibles nécessitent donc une protection supplémentaire, en utilisant notamment des cryptages.
- **L'entité externe XML / XXE** ou A4 : certains anciens processeurs XML ou avec une mauvaise configuration ne permettent pas de bien évaluer les références d'entités externes dans les documents XML. Ces entités externes peuvent alors être utilisées pour transférer et divulguer des fichiers internes. Ceci grâce à différents éléments : le gestionnaire d'URI de fichier, l'analyse des ports internes, le partage de fichiers internes, l'exécution de code à distance, ainsi que les attaques par déni de service.

- **La déviation des contrôles d'accès** ou A5 : parfois, certaines restrictions appliquées aux utilisateurs authentifiés sur ce qu'ils ne peuvent pas faire ne sont pas appliquées correctement. Les hackers peuvent exploiter cette faille pour s'introduire et avoir accès aux différentes fonctionnalités, voire données, non autorisées. Nous avons par exemple les comptes d'utilisateurs, l'affichage de fichiers sensibles ou encore la modification des droits d'accès.
- **La configuration incorrecte de la sécurité** ou A6 : ce qui est lié à une mauvaise configuration de sécurité est extrêmement courant. En général, c'est le résultat de configurations effectuées par défaut totalement non sécurisées ou bien, incomplètes. Cela peut également être le résultat d'un stockage dans le Cloud ouvert, d'entêtes HTTP mal configurés ou encore de messages d'erreurs détaillés et qui contiennent des informations sensibles. Pour faire face à ce problème, les systèmes d'exploitation, frameworks, applications et autres bibliothèques doivent être configurés et mis à niveau afin d'être sécurisés.
- **Cross Site Scripting** ou A7 : ce type de faille appelée XSS fonctionne comme suit : un hacker exécute des scripts malveillants dans le navigateur web ou application de la personne ciblée. L'attaque se déclenche une fois que la victime ouvre sa page web qui exécute ce code malveillant. La page web ou le site web est donc utilisé pour transmettre le script malveillant.
- **Désérialisation non sécurisée** ou A8 : ce type de faille se produit lorsque des données malveillantes sont utilisées pour corrompre et venir abuser de la logique d'une API ou application.
- **L'utilisation de composants avec des vulnérabilités connues** ou A9 : les composants, tels que les bibliothèques, les frameworks et d'autres modules logiciels, s'exécutent avec les mêmes privilèges que l'application. Si un composant vulnérable est exploité, une telle attaque peut causer de graves pertes de données ou une prise de contrôle du serveur.
- **Journalisation et surveillance insuffisante** ou A 10 : une journalisation et une surveillance insuffisante, associées à une intégration manquante ou inefficace avec la réponse aux incidents, permettent aux hackers d'attaquer davantage les systèmes, de maintenir la persistance, de basculer vers plus de systèmes et de falsifier, extraire ou encore détruire les données.

#### Remarque

Si vous utilisez des pratiques de codage sécurisées et que vous êtes bien renseigné sur ces attaques, vous avez déjà une longueur d'avance sur le hacker. Il ne faut pas oublier que 80 % des logiciels de production présentent des vulnérabilités communes, comme indiqué dans OWASP. Apprendre à les connaître et utiliser des techniques de codage sécurisées simples permettra de nous aider à créer un code de qualité.

## B. Django-REST et sécurité

#### Définition

L'injection se produit lorsqu'un hacker injecte du code, un script ou une commande à l'aide d'une application web. Certaines injections courantes sont des requêtes SQL pour manipuler la base de données, des scripts JavaScript et HTML exécutés dans l'application web et des injections de système d'exploitation qui contrôlent le système d'exploitation.

#### Fondamental

##### • A1 Injection

Les jeux de requête de Django sont totalement immunisés contre les injections SQL. En effet, leurs requêtes sont construites à l'aide de la paramétrisation des requêtes. Le code SQL d'une requête est défini indépendamment de ses paramètres.



- **A2 Piratage de sessions**

Il existe plusieurs techniques qui permettent de se protéger, en voici quelques-unes :

- N'acceptez que les mots de passe contenant des minuscules, des majuscules et des caractères spéciaux.
- Informez les utilisateurs de changer régulièrement de mot de passe.
- Empêchez les utilisateurs d'effectuer un trop grand nombre d'essais pour s'identifier.
- Ne jamais stocker les mots de passe, toujours les hacher avant.

**Remarque**

Django.contrib.sessions présente également certaines limites. Les sous-domaines d'un site peuvent définir des cookies pour le client, valables pour tout le domaine. Cela rend possibles les attaques par fixation de session si des cookies peuvent être créés par des sous-domaines qui ne sont pas sous le contrôle de personnes de confiance.

- **A3 Exposition aux données sensibles**

- N'utilisez **GET** que pour récupérer des informations.
- Utilisez **POST** pour les informations qui seront manipulées. Toutes les requêtes **POST** doivent utiliser HTTPS / SSL pour garantir que le corps est crypté.
- Vérifiez tous les modules tiers que vous utilisez pour créer des requêtes **GET** / **POST** et utilisez HTTPS pour tous ! Utilisez les COR si vous sortez du domaine.
- Sécurisez votre base de données avec un cryptage.
- Utilisez des algorithmes de hachage.
- N'utilisez pas uniquement des algorithmes de hachage SHA et MD5 sans randomisation de sel.
- Le masquage des données peut être utilisé pour sécuriser les données sensibles sur la base de données.

- **A4 Entités externes XML (XEE)**

Il existe plusieurs XML en Python pour gérer ce problème, comme Sax, Etree, Minidom, Pulldom, Xmlrpc, Lxml, Genshi.

Etree, Minidom, Xmlrpc et Genshi sont fiables contre les attaques d'extension d'entités externes locales et distantes. Cependant, il existe un package plus sécurisé appelé **defusedxml**. Vous pouvez utiliser l'un des analyseurs ci-dessus via defusedxml et être à l'abri des attaques XEE.

Defusedxml empêche les attaques XEE :

- Il interdit le XML avec les déclarations `< ! ENTITY>` dans la DTD et lève une exception `EntitiesForbidden` lorsqu'une entité est déclarée.
- Il interdit tout accès aux ressources distantes ou locales dans des entités externes ou DTD et déclenche une exception `ExternalReferenceForbidden` lorsqu'une DTD ou une entité fait référence à une ressource externe.

- **A5 Déviation de contrôle d'accès**

Voici quelques bonnes pratiques pour éviter ce genre d'attaque. Au lieu de nommer vos pages cibles avec un sens, utilisez un tableau de valeurs clés qui font référence à vos objets. Modifiez les noms par défaut de vos pages web. Assurez-vous que toutes les pages ont un contrôle d'authentification. Personnalisez vos exceptions et vos codes d'erreur.

- **A7 Cross-Site Scripting (XSS)**

L'utilisation des gabarits Django vous protège de la majorité des attaques XSS. Cependant, il est important de comprendre les protections appliquées, ainsi que leurs limites. Les gabarits de Django échappent aux caractères spécifiques qui sont particulièrement dangereux en HTML. Bien que cela protège les utilisateurs de la plupart des saisies malveillantes, cela ne constitue pas une protection absolue. Il est aussi important d'être particulièrement prudent lors de l'utilisation de `is_safe` avec des balises de gabarit personnalisées, avec la balise de gabarit `safe`, avec `mark_safe` et quand l'échappement automatique est désactivé. De plus, si vous utilisez le système des gabarits pour produire du contenu autre que du HTML, les caractères et les mots nécessitant l'échappement peuvent être totalement différents. Vous devez aussi être très prudent lorsque vous stockez du code HTML dans la base de données, en particulier quand ce contenu est sélectionné et affiché.

**Exercice : Quiz**

[solution n°3 p.21]

## Question 1

Que signifie OWASP ?

- ☐ Open Web Application Security Project
- ☐ Online Web Access Security Protocol
- ☐ Online Web Authority System Profile
- ☐ Open Web Automated System of Protection

## Question 2

Vous devez utiliser une Black List dans la mesure du possible. N'utilisez les White List que comme défense secondaire.

- ☐ Vrai
- ☐ Faux

## Question 3

Lequel des éléments suivants est une attaque par injection ?

- ☐ Cross-Site Scripting
- ☐ Falsification de requêtes intersites
- ☐ Références d'objet directes non sécurisées
- ☐ Authentification et gestion de sessions interrompues

## Question 4

Qu'est-ce que Django apporte contre le Cross-Site Scripting ?

- ☐ Des gabarits
- ☐ Des modèles
- ☐ Des réponses sécurisées

## Question 5



Lequel des éléments suivants est le meilleur moyen d'empêcher les entrées malveillantes d'exploiter votre application ?

- ☐ Validation d'entrée à l'aide d'une liste d'autorisations
- ☐ Utilisation de cryptage
- ☐ Utilisation de l'indirection de table
- ☐ Utilisation de **GET** / **POST**

## VII. Essentiel

Les API REST sont une interface pour les applications tierces qui leur permet d'impacter la base de données. Pour ne pas perdre de temps, il faut garder à l'esprit que Django fournit déjà de puissants modules qui font le plus gros travail. Enfin, bien qu'il existe un très grand nombre de failles en Web et donc d'attaques potentielles, grâce à Django, la majorité d'entre elles sont bloquées sans que vous n'ayez rien de plus à mettre en place.

## VIII. Auto-évaluation

### Exercice 1 : Quiz

[solution n°4 p.22]

#### Question 1

Que dois-je faire pour pouvoir utiliser le framework d'API REST de Django ?

- ☐ Utiliser la commande `pip install django djangorestframework`
- ☐ Ajouter `rest_framework` à la variable `INSTALLED_APPS`
- ☐ Modifier le fichier `manage.py`
- ☐ Utiliser la commande `python manage.py shell`

#### Question 2

Pour notre API, quel composant de Django REST est responsable de la validation des champs de saisie (et JSON) ?

- ☐ Les sérialiseurs
- ☐ Les vues
- ☐ Les URL
- ☐ Les modèles

#### Question 3

Pour créer un sérialiseur, quelle classe dois-je utiliser ?

- ☐ `rest_framework.serializers.Serializer`
- ☐ `json.Serializer`
- ☐ `Django.serializers.Serializer`
- ☐ `rest_framework.Serializer`

#### Question 4

Pour définir mes vues, quel type d'objet puis-je utiliser ?

- ☐ `rest_framework.views.APIView`
- ☐ `rest_framework.viewsets.Viewset`
- ☐ `rest_framework.serializers.Serializer`
- ☐ `rest_framework.View`

#### Question 5

Quel nom de méthode ne sera pas pris en compte par le dispatcher de requête si on la crée sur un objet **APIView** ?

- ☐ `new()`
- ☐ `get()`
- ☐ `delete()`
- ☐ `put()`

#### Question 6

Quelle méthode pourriez-vous remplacer pour changer le comportement d'un objet **ViewSet** afin de créer un objet ?

- ☐ `create(self, request)`
- ☐ `post(self, request)`
- ☐ `perform_create(self, request)`
- ☐ `put(self, request)`

#### Question 7

Quelle est la bonne méthode pour récupérer un ID dans une URL ?

- ☐ `path("object/<int:id>", MyView.as_view())`
- ☐ `path("object/%s", MyView.as_view())`
- ☐ `path("object/get(id=id)", MyView.as_view(id))`
- ☐ `path("object/<id>", MyView.as_view())`

#### Question 8

Un utilisateur veut transmettre une entrée malveillante afin de faire exécuter du code à notre serveur. Quel est ce type d'attaque ?


- ☐ Une injection
- ☐ Un défaut d'accès URL
- ☐ Un piratage de session
- ☐ Une exposition aux données sensibles

## Solutions des exercices

**Exercice p. 5 Solution n°1****Question 1**

Quand est-ce qu'on utilise une API Web ?


- ☒ Quand l'ensemble des données est trop volumineux pour un être humain, ce qui rend le téléchargement coûteux en ressources.
- ☒ Quand vos utilisateurs devront accéder à vos données en temps réel.
- ☒ Quand vos données sont mises à jour fréquemment..
- ☐ Il est préférable de ne jamais utiliser API Web.

 Dans sa définition, une API va servir d'interface entre un programme externe et un serveur contenant, par exemple, une base de données. Dès que l'on a besoin de rendre accessibles certaines données ou opérations, une API Web est la solution parfaite.

**Question 2**

Qu'est-ce que HTTP ?


- ☐ Hyperlink Text Protocol
- ☐ Hypertext Transport Protocol
- ☐ Hypertext Transbordement Protocol
- ☒ Hypertext Transfer Protocol

 Le sigle HTTP signifie *HyperText Transfer Protocol*, c'est un des protocoles principaux, utilisé pour afficher des pages Internet (avec le HTTPS).

**Question 3**

Que veut dire URL ?

- ☒ Uniform Resource Locator
- ☐ Unique Reforme Location
- ☐ Unique Resource Location
- ☐ Uniform Reforme Locator
- ☐ Uniform Resource Location

 Le sigle URL signifie *Uniform Resource Locator*. C'est une norme d'écriture de chemin d'accès principalement utilisée pour identifier une ressource précise via un certain protocole Internet.

**Question 4**

Sélectionnez ce qui est juste sur l'URL

- ☒ Une URL se compose d'un protocole (http://).
- ☐ Une URL possède forcément un chemin.
- ☒ Une URL décrit l'emplacement d'une ressource spécifique.
- ☐ Lors de la lecture sur les API, il est impossible de voir les termes URL.

- Q Comme précisé dans la question précédente, une URL représente un chemin d'accès utilisé pour identifier une ressource précise via un certain protocole Internet. Elle se compose donc bien d'un protocole et d'un emplacement de ressource.

### Question 5

REST (*Representational State Transfer*) est une théorie qui décrit quelques bonnes pratiques pour l'implémentation d'API.

☒ Vrai

☐ Faux

- Q Une API REST (ou RESTful) est une API qui va respecter une liste de restrictions ou de définitions. Par exemple, les API REST ont une définition bien précise pour chaque méthode HTTP (**GET**, **POST**, **PUT**, etc.).

## Exercice p. 11 Solution n°2

### Question 1

Les sérialiseurs de Django REST :

☒ Servent à convertir les instances de modèle en dictionnaire Python.

☐ Ne sont pas conseillés.

☐ Créent un modèle en Django.

☐ Créent une vue en Django.

- Q Les sérialiseurs de Django REST convertissent les instances de modèle en dictionnaire Python. Le but d'un sérialiseur est de convertir un groupe d'informations d'une forme complexe et massive vers une forme générique et plus simple. Dans notre cas, le sérialiseur permet de transmettre une instance d'objet (par exemple, un modèle de base de données) vers une instance de dictionnaire. Les attributs et leurs valeurs sont transformés en paires de clé et valeur dans le dictionnaire.

### Question 2

À quoi sert la méthode `put()` ?

☒ Cette méthode sert à mettre à jour les données de l'objet.

☐ Supprime l'objet.


☐ Cette méthode va juste utiliser `get_object` et renvoyer la réponse au serveur.

☐ Cette fonction récupère un objet dont l'ID est placé dans l'URL.

- Q La fonction `put()` sert à mettre à jour les données de l'objet. La méthode `put`, comme pour une API REST, permet la mise à jour d'un élément en base de données. C'est la méthode qui sera utilisée lors d'une requête **PUT** visant à modifier un élément dans la base de données (par exemple, une modification de profil utilisateur).

### Question 3

À quoi sert la méthode `get()` ?


- ☐ Cette méthode sert à mettre à jour les données de l'objet.
  - ☐ Supprime l'objet.
  - ☒ Cette méthode va juste utiliser **get\_object** et renvoyer la réponse au serveur.
  - ☐ Cette fonction récupère un objet dont l'ID est placé dans l'URL.
-  La méthode **get**, comme pour une API REST, permet de récupérer un élément précis depuis la base de données (généralement grâce à un ID). C'est la méthode qui sera appelée lors d'une requête **GET** visant à récupérer, par exemple, les informations d'un profil utilisateur. Cette méthode utilise généralement **get\_object** pour atteindre son objectif.

#### Question 4

---

Qu'est-ce que JSON ?

- ☒ JavaScript Object Notation
- ☐ Java Object Notation
- ☐ JavaScript Object Note


 Le sigle JSON signifie *JavaScript Object Notation*. C'est un format d'information très utilisé pour communiquer un groupe d'informations en HTTP. Même si son sigle fait référence au JavaScript, l'utilisation de ce format est commune pour énormément de langages.

#### Question 5

---

**JSONRender** sert à finaliser la sérialisation au format JSON.

- ☒ Vrai
- ☐ Faux

 Grâce à **JSONRender**, on peut transformer un groupe de données en JSON. De cette manière, on peut échanger des informations complexes via un format de données commun et simple.


### Exercice p. 16 Solution n°3

#### Question 1

---

Que signifie OWASP ?


- ☒ Open Web Application Security Project
- ☐ Online Web Access Security Protocol
- ☐ Online Web Authority System Profile
- ☐ Open Web Automated System of Protection

 C'est le sigle de l'association présentée plus tôt. Pour rappel, cette association a pour but de veiller à la sécurité des logiciels.

#### Question 2


---

Vous devez utiliser une Black List dans la mesure du possible. N'utilisez les White List que comme défense secondaire.

- ☐ Vrai
- ☒ Faux
-  Il n'y aucune différence entre l'utilisation exclusive de White List ou de Black List.


**Question 3**

Lequel des éléments suivants est une attaque par injection ?

- ☒ Cross-Site Scripting
- ☐ Falsification de requêtes intersites
- ☐ Références d'objet directes non sécurisées
- ☐ Authentification et gestion de sessions interrompues
-  Le but de cette attaque est d'injecter du contenu généralement malveillant dans une page web. Sur les sites qui ne s'en protègent pas, on peut par exemple, via un système de commentaire, écrire un commentaire contenant du code HTML ou JS qui sera exécuté lorsque le commentaire sera affiché.


**Question 4**

Qu'est-ce que Django apporte contre le Cross-Site Scripting ?

- ☒ Des gabarits
- ☐ Des modèles
- ☐ Des réponses sécurisées
-  Les gabarits Django permettent d'échapper les caractères étant susceptibles d'être interprétés comme du code. C'est de cette manière qu'ils aident à contrer les attaques XSS.

**Question 5**

Lequel des éléments suivants est le meilleur moyen d'empêcher les entrées malveillantes d'exploiter votre application ?


- ☒ Validation d'entrée à l'aide d'une liste d'autorisations
- ☐ Utilisation de cryptage
- ☐ Utilisation de l'indirection de table
- ☐ Utilisation de **GET** / **POST**
-  Avec une liste d'autorisations, votre site sera en mesure de bloquer les contenus malveillants, qui, du coup, apparaîtront comme non-autorisés.

**Exercice p. 17 Solution n°4**

### Question 1

Que dois-je faire pour pouvoir utiliser le framework d'API REST de Django ?


- ☒ Utiliser la commande `pip install django djangorestframework`
- ☒ Ajouter `rest_framework` à la variable `INSTALLED_APPS`
- ☐ Modifier le fichier `manage.py`
- ☐ Utiliser la commande `python manage.py shell`

 Afin d'utiliser ce type de framework, il est indispensable d'utiliser la commande `pip install django djangorestframework` et d'ajouter par la suite `rest_framework` à la variable `INSTALLED_APPS`.

### Question 2

Pour notre API, quel composant de Django REST est responsable de la validation des champs de saisie (et JSON) ?


- ☒ Les sérialiseurs
- ☐ Les vues
- ☐ Les URL
- ☐ Les modèles

 Les vues n'ont rien à faire avec les données, elles ne font que les afficher ; les modèles ne vérifient pas les données, ils ne font que les transporter. Seuls les sérialiseurs font cette vérification lorsqu'ils convertissent les données en dictionnaire python.

### Question 3

Pour créer un sérialiseur, quelle classe dois-je utiliser ?


- ☒ `rest_framework.serializers.Serializer`
- ☐ `json.Serializer`
- ☐ `Django.serializers.Serializer`
- ☐ `rest_framework.Serializer`

 Pour créer un sérialiseur Django, il faut créer une classe héritant de la classe **Serializer** contenue dans le module `rest_framework.serializers`.

### Question 4


Pour définir mes vues, quel type d'objet puis-je utiliser ?

- ☒ `rest_framework.views.APIView`
- ☒ `rest_framework.viewsets.Viewset`
- ☐ `rest_framework.serializers.Serializer`
- ☐ `rest_framework.View`

 Comme utilisées précédemment, les deux types de classes valides sont les **APIView** et **ViewSet**. Ces deux classes proviennent du framework Django REST et sont là pour simplifier la conception de l'API.


### Question 5

Quel nom de méthode ne sera pas pris en compte par le dispatcher de requête si on la crée sur un objet **APIView** ?

- ☒ `new()`
  - ☐ `get()`
  - ☐ `delete()`
  - ☐ `put()`
-  Tout comme l'objet Django **View**, l'objet **APIView** va accepter les méthodes **get**, **post**, **delete**, **put**, etc. Ces noms de méthodes correspondent aux différents types de requête HTTP qui existent. Et le type **new** n'existe pas. Pour créer un élément, on utilise généralement une requête de type **post**.


### Question 6

Quelle méthode pourriez-vous remplacer pour changer le comportement d'un objet **ViewSet** afin de créer un objet ?

- ☒ `create(self, request)`
  - ☐ `post(self, request)`
  - ☐ `perform_create(self, request)`
  - ☐ `put(self, request)`
-  La méthode **create** est la seule fonction permettant de créer un objet et qui est déjà défini par la classe **ViewSet**. Les méthodes **post** et **put** ne sont définies que pour la classe **APIView**.


### Question 7

Quelle est la bonne méthode pour récupérer un ID dans une URL ?

- ☒ `path("object/<int:id>", MyView.as_view())`
  - ☐ `path("object/%s", MyView.as_view())`
  - ☐ `path("object/get(id=id)", MyView.as_view(id))`
  - ☐ `path("object/<id>", MyView.as_view())`
-  Seulement la première méthode fonctionne pour atteindre notre but. Avec cette syntaxe, on signifie à Django qu'après le caractère « / », on s'attend à récupérer un nombre entier.

### Question 8

Un utilisateur veut transmettre une entrée malveillante afin de faire exécuter du code à notre serveur. Quel est ce type d'attaque ?

- ☒ Une injection
  - ☐ Un défaut d'accès URL
  - ☐ Un piratage de session
  - ☐ Une exposition aux données sensibles
-  Une injection est le terme pour parler d'injection de code par un attaquant, par exemple, dans la base de données. Le but de cette attaque est de faire exécuter du code malicieux à notre serveur pour forcer certaines actions.