DOCKER-COMPOSE

TP DOCKERCOMPOSE DJANGO

Dans ce tp nous allons voir la mise en place d'une solution docker et docker-compose pour un projet django. Nous verrons la mise en place d'une image pour l'environnement de développement et une image pour l'environnement de production.

I/Mise en place

Vous allez fonctionner avec un dépôt git personnel dans lequel se trouve un projet Django. Vous pouvez réutiliser un ancien projet ou en démarrer un nouveau. Nous mettrons en place un environnement de développement, visant à tester rapidement et efficacement notre projet sur le serveur, et un environnement de production sécurisé.
Sur votre serveur Docker commencez par installer les outils pour interagir avec github et gitlab :

```
sudo apt install git
```

Dans votre répertoire personnel créez un dossier docker_django. Créez dedans deux dossiers, un dossier dev et un dossier prod. Dans chacun de ces répertoires initiez votre dépôt git. Créez une nouvelle branche par répertoire, une branche prod et une branche dev. Nous allons commencer par la mise en place de l'environnement de développement. Nous rédigerons un Dockerfile pour l'image python django que nous exécuterons avec une base de données sqlite3. Nous lierons ensuite notre service à un container mariadb et changerons la configuration du projet pour qu'il se connecte à cette base.

II/Environnement de développement

- **A/ L'image python**

Pour configurer notre image nous allons nous baser sur une image python, dans laquelle nous installerons django. Pour faciliter l'installation présente et futurs de paquets python nous allons rédiger un fichier requirements.txt dans lequel nous indiquerons quels paquets nous souhaitons installer. Pour l'instant indiquez juste `Django==4.0.4`.

Rédigez maintenant un dockerfile, qui se base sur l'image python:3. Déclarez dedans deux variables d'environnement `PYTHONWRITECODE=1` qui permet de faire en sorte que python n'écrivent pas les fichier binaires, ce qui peut faire grossir l'espace pris par votre projet, ainsi que la variable `PYTHONBUFFERED=1` qui permet d'avoir les logs django directement attaché aux logs python et qui seront donc visibles sur les logs du container. Indiquez ensuite que le répertoire de travail est `/code` et

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**			
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

DOCKER-COMPOSE

copiez tout le répertoire dans lequel vous vous trouvez dedans. L'image doit exécuter pour finir la commande `pip install –r requirements.txt`.

Pour démarrer l'image que vous venez de créer vous allez rédiger un fichier docker-compose dans lequel vous indiquerez certains paramètres.
Vous n'indiquez qu'un seul service pour l'instant, web, qui compilera le Dockerfile présent dans le dossier. Le service devra exécuter la commande `python manage.py runserver 0.0.0.0:8000` pour que votre projet django démarre au lancement du container, il attribuera le répertoire actuel au répertoire `/code` du container et attribuer le port 8000 de la vm au port 8000 du container.

Une fois le docker-compose rédigé vous pouvez l'exécuter avec la commande :

```
docker-compose up -d --build
```

Votre container web démarrer en utilisant l'image que vous avez rédigés dans le Dockerfile, exécute la commande pour démarrer votre serveur django sur le port 8000, qu'il attribue au port 8000 de votre machine virtuelle. Vous pouvez maintenant accéder à votre projet en allant sur

```
http://<ip_de_la_vm>:8000/<votre_application>
```

Si en accédant au site vous avez une erreur django vous indiquant que vous devez ajouter l'adresse de votre machine virtuelle à `ALLOWED_HOSTS` allez dans le fichier settings.py de votre projet. Dedans trouvez la ligne où la variable `ALLOWED_HOST` est déclarée en tant que liste vide. Mettez `"*"` entre les crochets de la liste afin d'autoriser toutes les adresse ip à publier votre site (attention ceci n'est pas recommandé pour une version de production). Vous pouvez maintenant redémarrer votre docker-compose et accéder à votre site.

Une fois que tout fonction faites un commit et push vers la branche dev de votre dépôt.

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

DOCKER-COMPOSE

B/ La base de données

Nous allons maintenant rajouter un service de base de données à notre docker-compose et nous allons modifier notre projet pour qu'il se connecte à celle-ci au lieu d'utiliser le fichier db.sqlite3.
Commencez par récupérer le contenu de votre base de données avec la commande :

```
docker exec dev_web_1 python manage.py dumpdata > db.json
```

Configurons maintenant notre service de base de données dans notre docker-compose.
Rajoutez sur le fichier un service que vous appellerez db dont l'image est mariadb.
Définissez-y des variables d'environnement suivantes :

- MARIADB_ROOT_PASSWORD: secret
- MARIADB_DATABASE: db
- MARIADB_USER: dev
- MARIADB_PASSWORD: Azerty77

Vous allez également créer un volume docker pour y stocker la base de données. Vous faites cela en rajoutant les lignes suivantes dans votre service db:

```
volumes:
 - mariadb_data:/var/lib/mysql/data
```

Avec cette instruction docker va stocker le contenue du répertoire `/var/lib/mysql/data` de votre container dans le répertoire docker `mariadb_data`. Il faudra également rajouter la déclaration du volume en fin de fichier :

```
volumes:
 -mariadb_data:
```

Nous devons également modifier notre image python pour y installer les outils de connexion à mysql.
Rajoutez dans votre fichier Dockerfile cette ligne :

```
RUN apt update && apt install -y default-mysql-client
```

Rajoutez également au fichier requirements.txt la ligne :

```
mysqlclient==2.1.0
```


Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

DOCKER-COMPOSE

Il faudra maintenant modifier le fichier settings.py de notre projet pour lui indiquer la connexion à la base de données. Le fichier settings.py aura besoin de 4 informations pour établir la connexion :

- Le nom de la base de donnée
- Le nom de l'utilisateur
- Le mot de passe de l'utilisateur
- Le nom d'hôte du serveur mariadb
- Le port du serveur mariadb

Nous pourrions remplir directement ces champs dans le fichier settings.py mais pour que des modifications futures soient plus facile nous allons utiliser des variables d'environnement que nous allons définir dans notre fichier docker-compose.

Dans le service web rajoutez ces variables d'environnement :

- MARIADB_DATABASE=db
- MARIADB_USER=dev
- MARIADB_PASSWORD=Azerty77
- MARIADB_HOST=db
- MARIADB_PORT=3306

Nous pouvons maintenant nous servir de ces variables dans notre fichier settings.py. Commencez par importer le module os. Remplacez ensuite la variable DATABASES par celle-ci:

```
DATABASES = {
        'default':{
                'ENGINE': 'django.db.backends.mysql',
                'NAME': os.environ.get("MARIADB_DATABASE"),
                'USER': os.environ.get("MARIADB_USER"),
                'PASSWORD': os.environ.get("MARIADB_PASSWORD"),
                'HOST': os.environ.get("MARIADB_HOST"),
                'PORT': os.environ.get("MARIADB_PORT"),
        }
}
```

Avec ce nouveau fichier settings.py de votre projet et le docker-compose correctement rédigé vous pouvez relancer votre solution avec la commande :

```
docker-compose up -d --build
```


Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

DOCKER-COMPOSE

Notre docker compose ne fait que démarrer nos services. Il faut donc maintenant écrire les données de notre projet dans notre base. Commencez par créer les tables dans la base en utilisant la commande migrate du manage.py

```
docker exec dev_web_1 python manage.py migrate
```

Vous devriez voir à l'écran la création dans la base des tables de Django et de votre projet. Il va falloir maintenant remplir votre base de données avec ce qui était présent dans votre projet quand il tournait avec une base sqlite3. Il va tout d'abord purger votre projet des informations qu'il aurait stocké de votre ancienne configuration de base de donnée. Pour ça démarrez un shell dans votre projet dans votre container :

```
docker exec -it dev_web_1 python manage.py shell
from django.contrib.contenttypes.models import ContentType
ContentType.objects.all().delete()
exit()
```

Vous pouvez maintenant charger les données que vous avez sauvegardé auparavant dans le fichier db.json :

```
docker exec dev_web_1 python manage.py loaddata db.json
```

Et enfin il vous faut démarrer votre serveur django :

```
docker exec –d dev_web_1 python manage.py runserver 0.0.0.0:8000
```

Votre projet est maintenant accessible à l'adresse http://<ip_de_votre_vm>:8000/<nom_application> avec toutes les données présentes dans votre base mariadb.

Une fois cette étape réussie vous pouvez faire un commit et push vers votre branche de développement.

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

DOCKER-COMPOSE

C/ L'automatisation

Votre solution fonctionne mais plusieurs étapes sont nécessaires pour qu'il soit opérationnel sur un autre serveur : démarrer le docker-compose, migrer la base, vider les informations de base de donnée, remplir la base avec les données, démarrer le serveur.
Nous allons voir comment rendre ce process plus simple en rédigeant un script qui va effectuer toutes ces actions en une seule commande. Pour se faire nous allons utiliser le paquet make.

```
apt install make
```

Make nous permet de rédiger un fichier Makefile dans lequel on va définir des nom d'action qui vont éffectuer nos taches. Nous avons par contre dans nos tâche l'interaction avec le shell pour vider les informations de base de données. Vous allez donc créer un fichier python script.py que vous enverrez dans le shell du container. Indiquez dans ce fichier python les deux lignes que vous aviez écrit dans le shell :

```
from django.contrib.contenttypes.models import ContentType
ContentType.objects.all().delete()
```

Fermez le fichier et rendez le exécutable avec la commande `chmod +x script.py`.
Nous allons maintenant voir la rédaction du Makefile qui va démarrer toutes nos tâches.

Créez un fichier Makefile dans votre répertoire de la branche dev et indiquez-y ci cette configuration

```
run:
	docker-compose up -d --build
	echo "Waiting mariadb to be up"
	sleep 10
	docker exec dev_web_1 python manage.py migrate
	docker exec dev_web_1 bash -c "python manage.py shell < script.py"
	docker exec dev_web_1 python manage.py loaddata db.json
	docker exec -d dev_web_1 python manage.py runserver 0.0.0.0:8000
```

Fermez et sauvegardez le fichier et coupez vos containers avec la commande

```
docker-compose down -v
```


Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

DOCKER-COMPOSE

Exécutez maintenant votre Makefile en tapant la commande make run, votre solution docker-compose devrait se déployer avec la bonne configuration.

Vous pouvez faire un dernier commit et push sur la branche dev de votre projet.

III/Environnement de production

Nous allons maintenant nous intéresse à la branche de production de notre projet. Certaines choses sont à changer par rapport à notre branche de développement. Tout d'abord le serveur généré par la commande python manage.py runserver n'est pas fait pour l'environnement de production. Comme nous le dit la documentation officielle de docker :

N'UTILISEZ PAS CE SERVEUR DANS UN ENVIRONNEMENT DE PRODUCTION. Il n'a pas fait l'objet d'audits de sécurité ni de tests de performance.

Nous le remplacerons par un serveur nginx, serveur web équivalent à apache, utilisant les outils wsgi permettant la gestion de code python.

Nous modifierons également la manière dont les informations de connexion sont transmises au container (le fait que les identifiants soient directement visibles dans le fichier docker-compose est un problème).

Nous changerons également l'utilisateur du container afin qu'il ne s'exécute pas en root.

Enfin nous inclurons le processus de démarrage de notre solution à l'image directement, sans avoir à passer par le fichier Makefile.

Commencez par récupérer les fichier db.json, docker-compose.yml, Dockerfile, requirements.txt et script.py de la branche de de développement pour les adaptez à cette branche.

A/ Les variables d'environnement

Jusqu'à maintenant les variables d'environnement étaient passées directement dans le fichier docker-compose et donc toutes les informations de connexion à votre base de données sont visibles par tout le monde qui récupèrerai votre dépôt. Pour pallier à ça nous allons rédiger un fichier .env dans lequel nous indiquerons nos variables d'environnement. Ce fichier sera exclu des fichiers à push sur votre dépôt. Nous créerons également un fichier .env.sample qui sera inclus au dépôt qui permettra

DOCKER-COMPOSE

d'indiquer les variables nécessaires au fonctionnement de votre projet. Voici la structure du fichier .env.sample :

```
DATABASE=<engine_of_your_database>
MARIADB_DATABASE=<name_of_your_databse>
MARIADB_USER=<user_for_your_database>
MARIADB_PASSWORD=<password_for_your_user>
MARIADB_ROOT_PASSWORD=<root_password_of_your_database>
MARIADB_HOST=<name_of_your_database_service>
MARIADB_PORT=<port_of_your_database_service>
```

Nous n'avons pas encore utilisé la variable DATABASE, nous nous en servirons par la suite.

Rédigez maintenant le fichier .env en remplaçant les valeurs des variables par celles indiqués dans votre docker-compose.

Dans le fichier docker-compose.yml remplacé les sections environnement des deux services par :

```
env_file:
  - .env
```

N'oubliez pas d'ajouter le fichier .env aux fichiers ignorés par git

B/ L'utilisateur du container

En l'état notre l'utilisateur de notre container est root, ce qui peut poser des problèmes de sécurité. Nous allons donc indiquer dans notre image web qu'il faut créer un utilisateur app, mettre app en propriétaire du dossier /code et de ses sous dossier et connecter l'utilisateur app. Pour les deux premières étapes il faudra utiliser l'instruction RUN afin d'exécuter les commandes nécessaires dans l'image. Pour connecter l'utilisateur vous vous servirez de l'instruction USER.

Avec cette modification l'utilisateur connecté dans notre container est l'utilisateur app qui a tous les droits sur le répertoire /code mais qui ne peux pas effectuer d'actions importante dans notre container.

DOCKER-COMPOSE

C/Le serveur web

Pour démarrer de manière sécurisé notre application à besoin d'un serveur web à qui transmettre les requêtes django. Vous utiliserez un container nginx. Il y a deux particularité dans le fait d'utiliser un serveur web pour django. Tout d'abord il faut utiliser le processus wsgi pour démarrer votre projet (au lieu de python manage.py runserver). Il faudra ensuite affecter un répertoire particulier aux fichiers statiques et medias.

Dans le répertoire de votre branche prod créez un dossier nginx. Dedans y figurera un simple Dockerfile ainsi que le fichier de configuration nginx de votre projet web.
Vous allez vous baser sur l'image nginx:1.21-alpine. Alpine est une distribution Linux faites pour être la plus légère possible, elle est donc particulièrement appropriée pour des images docker (l'image alpine fait 5MB alors que l'image Ubuntu en fait 188).
Vous indiquerez dans le Dockerfile d'éxecuter une commande pour supprimer le fichier `/etc/nginx/conf.d/default.conf` puis vous indiquerez de copier le fichier projet.conf, que vous allez créer dans le même dossier que le Dockerfile, dans le répertoire `/etc/nginx/conf.d` de l'image.

Voici ce qu'il faut indiquer dans le fichier projet.conf :

```
upstream <nom_projet> {
        server web:8000;
}
server {
        listen 80;
        location /static/ {
                alias /code/staticfiles/;
        }
        location / {
                proxy_pass http://<nom_projet>;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header Host $host;
                proxy_redirect off;
        }
}
```


Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

DOCKER-COMPOSE

Nous indiquons dans le fichier une définition pour un répertoire /code/staticfiles car nginx à besoin qu'on lui partage l'emplacement des fichiers statiques du projet. Django gère très bien ce cas grâce à deux choses : la définition d'une variable STATIC_ROOT et l'exécution d'une commande qui permet d'effectuer la migration des fichiers statiques dans le bon répertoire. Dans le fichier settings.py renseignez la ligne :

```
STATIC_ROOT = BASE_DIR / "staticfiles"
```

Ici BASE_DIR fait référence au répertoire de l'application (cette variable est définie au début du fichier). Vous indiquez donc, comme dans le fichier projet.conf et dans le Dockerfile, que les fichier statiques seront situés dans /code/staticfiles. Vous devez maintenant indiquer ce répertoire et le service nginx dans le docker compose.

Rajoutez donc un service nginx qui compilera l'image présente dans le dosier ./nginx.
Le service devra attribuer le port 80 de la vm au port 80 du container et sera lié au services web.
Renseignez ces lignes pour indiquer le volume du répertoire des fichiers statiques :

```
volumes:
  - static_volume:/code/staticfiles
```

Vous rajouterez également ce volume au service web.
Etant donné que nous sommes dans la branche de production nous ne voulons plus que les fichiers de codes soient modifiable vous allez donc retirer l'attribution du répertoire courant au répertoire /code du container du service web.
Il faudra également remplacer la commande de lancement de serveur python manage.py runserver 0.0.0.0:8000 par la commande gunicorn <nom_projet_django>.wsgi:application --bind 0.0.0.0:8000.
Cette commande permet de démarrer le projet en utilisant le processus wsgi gunicorn mais pour se faire il faut installer le module python pour gunicorn. Rajoutez donc cette ligne à votre fichier requirements.txt :

```
gunicorn==20.1.0
```


Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

DOCKER-COMPOSE

D/Automatisation

Nous allons maintenant voir une autre manière d'automatiser notre processus de déploiement. Vous allez remplacer le fichier Makefile par un script bash entrypoint.sh qui sera exécuté par l'image. Ce script va attendre que le service de base de données soit démarrer pour lancer le container web, puis va attribuer tous les droits à notre utilisateur, puis va charger migrer et remplir la base de données et enfin effectuer la commande permettant d'attribuer les fichiers statiques au répertoire indiqué dans le fichier settings.py. Voici comment rédiger le fichier entrypoint.sh :

```
#!/bin/bash

If [ "$DATABASE" = "mariadb" ]
then
        echo "Waiting for mariadb to be up"

        while ! nc -z $MARIADB_HOST $MARIADB_PORT; do
          sleep 0.1
        done

        echo "Mariadb started"
fi

echo "GRANT ALL on *.* to '${MARIADB_USER}'; "| mysql –u root --
password="${MARIADB_ROOT_PASSWORD}" –h "${MARIADB_HOST}"

python manage.py flush --no-input
python manage.py migrate
python manage.py loaddata db.json
python manage.py collectstatic --noinput

exec "$@"
```

Le script utilise le paquet nc lors de la vérification de l'état de la base de donnée. Il faut installer ce paquet dans l'image python.

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			

DOCKER-COMPOSE

Dans le Dockerfile du service python rajoutez le paquet netcat au paquet default-mysql-client déjà configuré. Il faut aussi indiquer d'exécuter le script entrypoint.sh au démarrage de l'image. Il faudra pour ça rendre le script exécutable. Rajoutez ces 2 lignes à la fin du Dockerfile :

```
RUN chmod +x /code/entrypoint.sh
ENTRYPOINT ["/code/entrypoint.sh"]
```

En indiquant ces lignes, l'image, une fois la compilation finie, lancera le script entrypoint.sh qui attendra que le service de base de données démarre avant de lancer les commandes permettant de mettre la base à jours. Vous pouvez donc démarrer votre recette docker-compose avec la commande:

```
docker-compose up -d --build
```

Si vos fichiers de configuration sont correctement rédigés vous pouvez accéder à votre site sur le port 80 de votre VM. Si c'est le cas vous pouvez effectuer un commit et un push vers votre branche prod.

Auteur(s)	Relu, validé et visé par :		Date de création :	Date dernière MAJ :
Maxime PRZYBYLO	Jérôme CHRETIENNE : **Resp. Secteur NUMERIQUE OCCITANIE**			
	Florence CALMETTES : **Coordinatrice Filière SYST. & RESEAUX**		*Toute reproduction, représentation, diffusion ou rediffusion, totale ou partielle, de ce document ou de son contenu par quelque procédé que ce soit est interdite sans l'autorisation expresse, écrite et préalable de l'ADRAR.*	
	Sophie POULAKOS : **Coordinatrice Filière WEBDESIGN / DEVELOPPEMENT**			