# Compte rendu de TP — Déploiement d'une API de Credit Scoring

Master 1 Informatique

PUREN Mewen
LE MANS Université

4 février 2026

## 1 Introduction

Ce TP avait pour objectif de déployer une API de **Credit Scoring** basée sur :

- FastAPI pour la partie backend
- Authentification JWT
- Base de données PostgreSQL
- Modèle de Machine Learning
- Orchestration avec Docker Compose

La mission consistait à rendre l'ensemble fonctionnel avec la commande :

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

Cependant, plusieurs problèmes ont été rencontrés et corrigés au cours du déploiement.

## 2 Problèmes rencontrés et solutions

### 2.1 Conflit de conteneur Docker

Un ancien conteneur existait déjà avec le même nom.

**Solution :**

```
docker rm -f credit-scoring-api
```

### 2.2 Port 8000 déjà utilisé

Le port 8000 était occupé par Portainer.

**Solution :**

```
docker stop <id-du-conteneur-
   portainer>
```

### 2.3 Dépendance Python manquante

Le module `email-validator` était absent.

**Solution :** ajout dans `requirements.txt` :

```
email-validator==2.1.0
```

### 2.4 Mauvaise URL PostgreSQL

L'API se connectait à `localhost` au lieu du conteneur Docker `db`.

**Avant :**

```
postgresql://meo@localhost:5432/db
```

**Après :**

```
postgresql://credit_user:
   credit_password@db:5432/db
```

### 2.5 Endpoint d'inscription manquant

L'endpoint `/auth/register` n'existait pas.

**Solution :** Ajout de la route avec validation de l'email et du username.

### 2.6 Double préfixe dans les routes

Les routes étaient du type :

```
/auth/auth/login
```

**Correction :** Suppression du préfixe dans les fichiers routers.

### 2.7 Nom de champ incorrect

Le champ `model_version` ne correspondait pas au schéma Pydantic.

**Solution :** Renommage en `model_ver`.

### 2.8 Imports incorrects

Dans `crud.py`, la classe `User` était mal importée.

**Solution :** Correction pour utiliser directement `User` et `Prediction`.

# 3 Reconstruction des images Docker

À chaque modification du code Python, il était nécessaire de reconstruire les images :

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

# 4 Fichiers modifiés

1. requirements.txt
2. app/config.py
3. app/routers/auth.py
4. app/routers/predictions.py
5. app/routers/admin.py
6. app/crud.py

# 5 Tests de l'API

### 5.1 script d'automatisation

À la racine du projet, il y a un script qui permet de tester toutes les fonctionnalités de l'api,

```
./test_api.sh
```

ce dernier prend en compte tous les cas d'on le mail ou pseudo déjà utiliser.

### 5.2 Création d'utilisateur

```
curl -X POST http://localhost:8000/
   auth/register \
-H "Content-Type: application/json"
   \
-d '{
"email": "john@example.com",
"username": "john",
"password": "SecurePass123",
"full_name": "John Doe"
}'
```

### 5.3 Connexion

```
curl -X POST http://localhost:8000/
   auth/login \
-H "Content-Type: application/x-www-
   form-urlencoded" \
-d "username=john&password=
   SecurePass123"
```

### 5.4 Prédictions

Sans token :

```
curl -X POST http://localhost:8000/
   predictions/predict
```

Avec token :

```
curl -X POST http://localhost:8000/
   predictions/predict \
-H "Authorization: Bearer TOKEN"
```

# 6 Conclusion

Ce TP a permis de comprendre :
— le fonctionnement de Docker Compose
— la gestion des dépendances Python
— l'importance des routes FastAPI
— la rigueur imposée par Pydantic
— l'analyse des logs Docker