Déploiement d’un modèle de prédiction de Churn

Richard HUGOU

Objectif de la mission

1. Industrialiser un modèle de Machine Learning existant (Prédiction de turnover).
2. Développer une API sécurisée et documentée avec FastAPI.
3. Mettre en place les bonnes pratiques MLOps (Tests, CI/CD).
4. Déployer la solution dans le Cloud (Hugging Face Spaces).

Architecture du projet

Langage :

- Python 3.13

API :

- PostgeSQl + Pydantic

API :

- FastAPI + Pydantic

Data :

- PostgreSQL + SQLAlchemy + Alembic

Sécurité :

- Bcrypt + Pydantic Settings

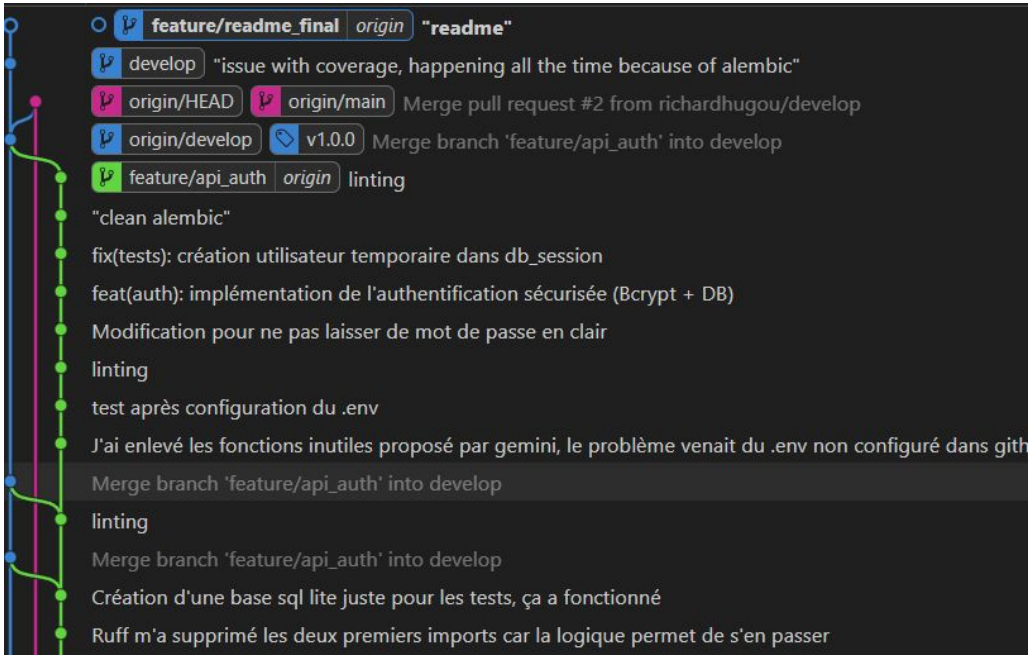
Ops :

- GitHub Actions + Docker (Service) + uv

Structure du git et du workflow

Utilisation de GitFlow pour structurer le développement :

- main : Code de production (Déployé automatiquement).
- develop : Branche d'intégration (Tests CI).
- feature/* : Développement des fonctionnalités (ex: feature/huggingFaceSpace).



Structure du dossier

.github/workflows pour la logique ci/cd

app contient le code source, la configuration et la structure de la base

data contient les données joblib (le modèle) ainsi que la database

tests contient les tests unitaires utilisé tout au long du développement

.toml contient les informations pour construire le projet (en utilisant uv de astra)

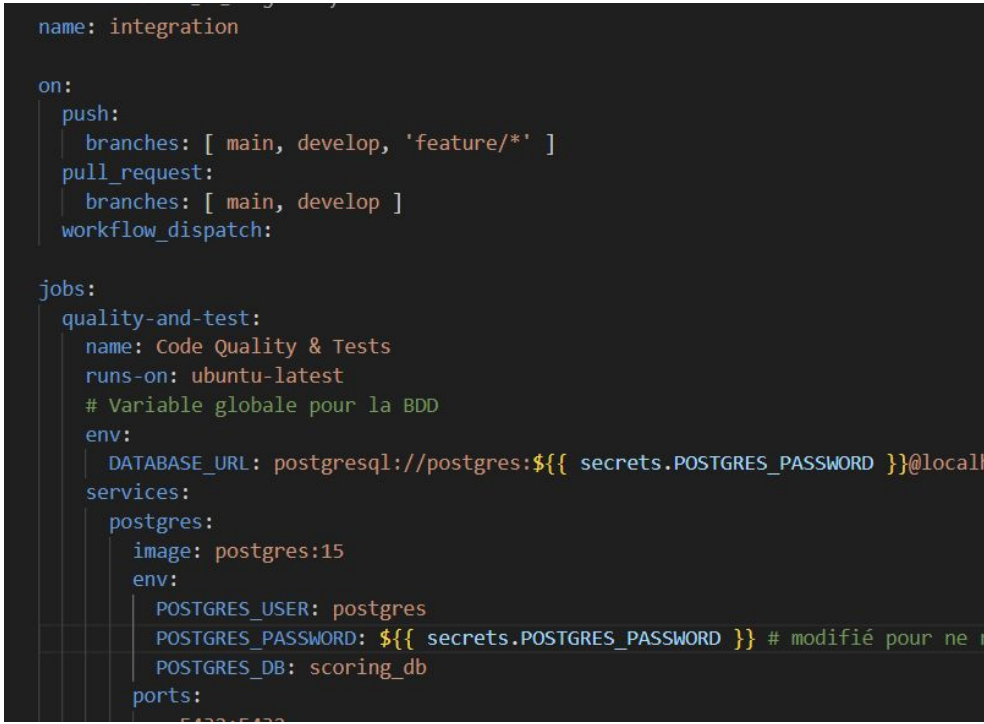
Pipeline CI/CD

Intégration continue

- Linting : Vérification du style avec Ruff.
- Migrations : Test de la création de BDD.
- Tests : Exécution de Pytest avec couverture.

Déploiement sur hugging face

- Déclenchement sur push main.
- Build & Push vers Hugging Face Spaces.



```yaml
name: integration

on:
  push:
    branches: [ main, develop, 'feature/*' ]
  pull_request:
    branches: [ main, develop ]
  workflow_dispatch:

jobs:
  quality-and-test:
    name: Code Quality & Tests
    runs-on: ubuntu-latest
    # Variable globale pour la BDD
    env:
      DATABASE_URL: postgresql://postgres:${{ secrets.POSTGRES_PASSWORD }}@localh
    services:
      postgres:
        image: postgres:15
        env:
          POSTGRES_USER: postgres
          POSTGRES_PASSWORD: ${{ secrets.POSTGRES_PASSWORD }} # modifié pour ne
          POSTGRES_DB: scoring_db
        ports:
```

Fast API

Choix de FastAPI pour sa rapidité et sa validation de données native (Pydantic).

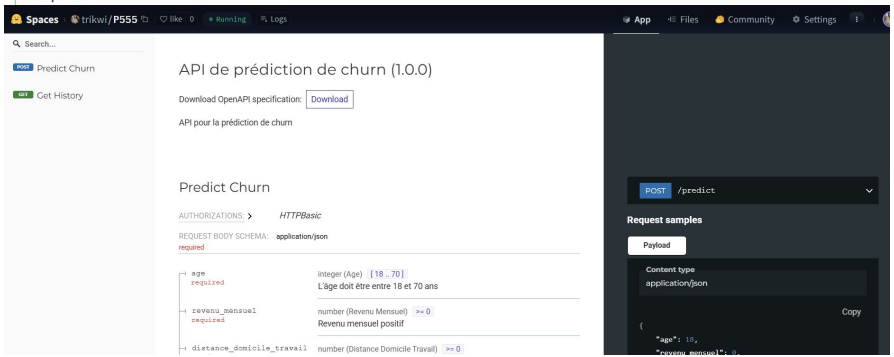
Points clés du code (app/main.py) :

- **Typage fort** : EmployeeInput valide les entrées (ex: age > 18).
- **Sécurité** : Hachage de mot de passe (Bcrypt) & Authentification Basic Auth.
- **Redirection UX** : Landing page HTML (réalisé à la fin, car l'app ne pouvait pas "fonctionner" sans) redirigeant vers la documentation.

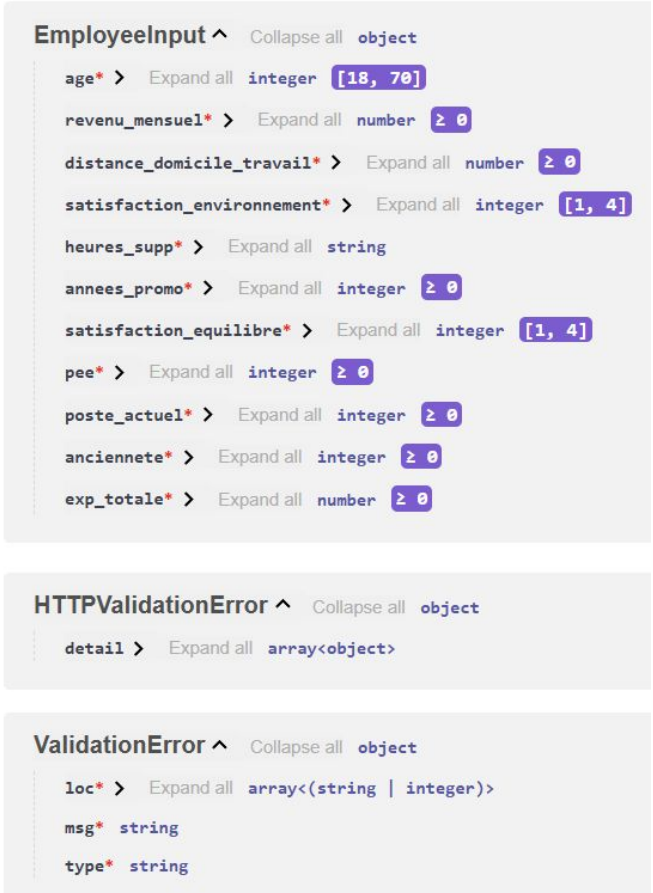
Documentation intégrée

L'API est auto-documentée (Standard OpenAPI).

- **Swagger UI (/docs)** : Interface interactive pour tester les endpoints en direct.
- **ReDoc (/redoc)** : Documentation technique statique pour les développeurs tiers.



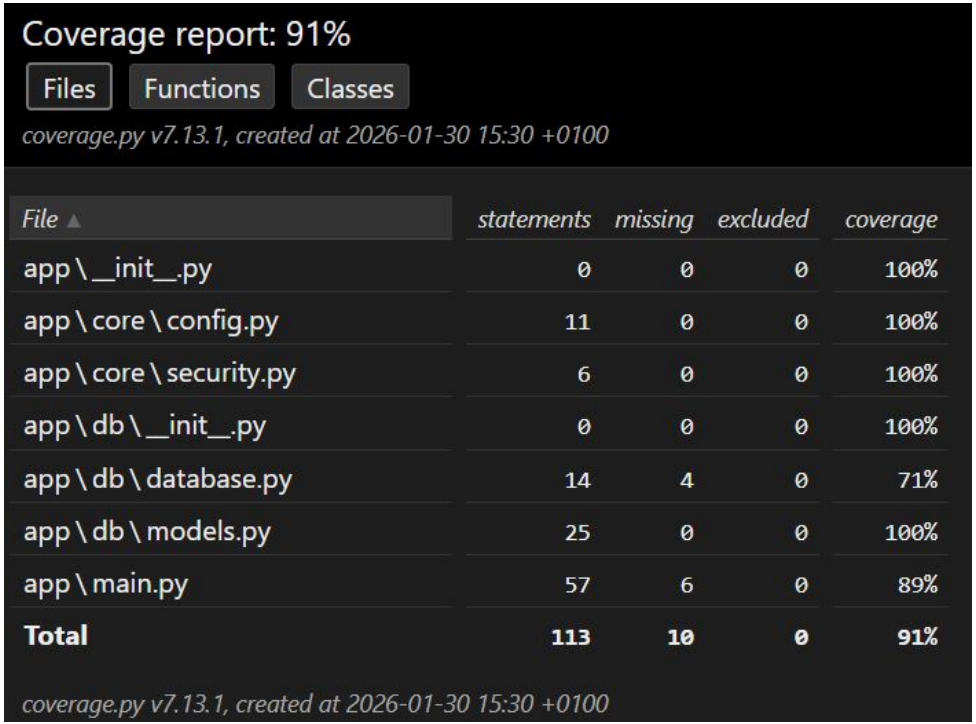
Modélisation des Données & Base de Données



Stratégie de test

Approche TDD/BDD pour garantir la fiabilité.

- **Isolation** : Utilisation de fixtures pour créer/détruire une base de données temporaire pour chaque test.
- **Scénarios couverts** :
  - Cas nominal (Prédiction OK).
  - Données invalides (Âge négatif, champs manquants).
  - Erreurs d'authentification.
- **Rapport** : Couverture de code > 80% (vérifié par pytest-cov).



Coverage report: 91%

Files | Functions | Classes

coverage.py v7.13.1, created at 2026-01-30 15:30 +0100

File ▲	statements	missing	excluded	coverage
app \ __init__.py	0	0	0	100%
app \ core \ config.py	11	0	0	100%
app \ core \ security.py	6	0	0	100%
app \ db \ __init__.py	0	0	0	100%
app \ db \ database.py	14	4	0	71%
app \ db \ models.py	25	0	0	100%
app \ main.py	57	6	0	89%
Total	**113**	**10**	**0**	**91%**

coverage.py v7.13.1, created at 2026-01-30 15:30 +0100

Base de donnée

Utilisation de SQLAlchemy pour ne pas avoir à réaliser de SQL.

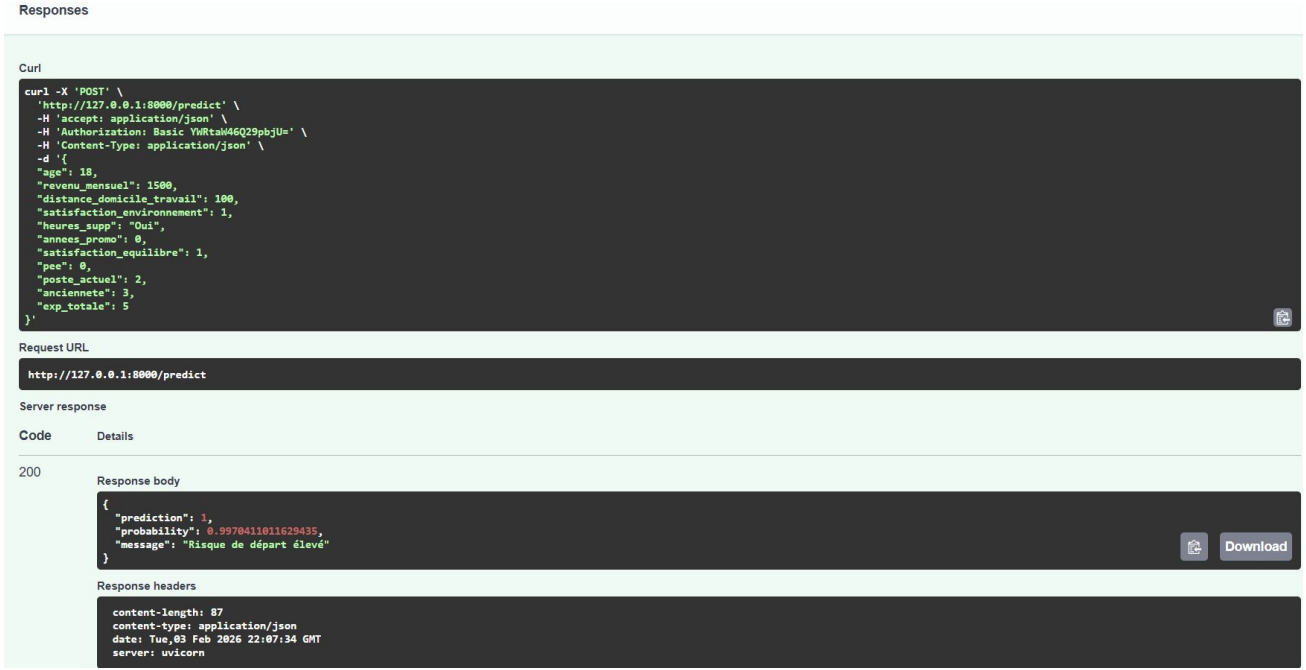
Modèles (app/db/models.py) :

- **User** :
  - username (Unique)
  - hashed_password (Sécurisé)
- **Historique** :
  - Stocke les inputs (age, revenu…)
  - Stocke la prediction et la probability.
  - Permet le monitoring du modèle (Data Drift).

Cas d’usage et résultat

Scénario de Démonstration :

- **Profil** : Employé jeune, faible revenu, loin du travail.
- **Requête** : Envoyée via Swagger.



Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/predict' \
  -H 'accept: application/json' \
  -H 'Authorization: Basic YWRtaW46Q29pbjU=' \
  -H 'Content-Type: application/json' \
  -d '{
  "age": 18,
  "revenu_mensuel": 1500,
  "distance_domicile_travail": 100,
  "satisfaction_environnement": 1,
  "heures_supp": "Oui",
  "annees_promo": 0,
  "satisfaction_equilibre": 1,
  "pee": 0,
  "poste_actuel": 2,
  "anciennete": 3,
  "exp_totale": 5
}'
```

Request URL

```
http://127.0.0.1:8000/predict
```

Server response

Code	Details
200	Response body

```
{
  "prediction": 1,
  "probability": 0.9970411011629435,
  "message": "Risque de départ élevé"
}
```

Download

Response headers

```
content-length: 87
content-type: application/json
date: Tue,03 Feb 2026 22:07:34 GMT
server: uvicorn
```