Architecture du Projet

Structure des Fichiers

```
.
├── notebooks/                  # Notebooks d'expérimentation
│   └── organe-maladie.ipynb    # Ex: blood-malaria.ipynb
│
├── models/
│   ├── blood/                  # Dossier spécifique à l'organe "sang"
│   │   ├── main.py             # Point d'entrée central pour toutes les maladies du sang
│   │   └── malaria_model.py    # Classe pour le modèle "paludisme"
│   │       └── predict(image) → JSON
│   │
│   ├── lung/                   # Dossier pour l'organe "poumon"
│   │   ├── main.py
│   │   └── pneumonia_model.py
│   └── ...
│
├── healthbox/                  # Interface Web + Backend
│                              # Logique métier (routes, appels aux modèles)
│
├── main.py                     # Orchestrateur racine (appelle les main.py des organes)
└── docker/
    ├── Dockerfile              # Configuration pour conteneuriser l'app
    └── docker-compose.yml      # Déploiement multi-services
```

Détails des Composants

1. Dossier `notebooks`

- **Format** : `organe-maladie.ipynb` (ex: `blood-malaria.ipynb`).
- **But** : Expérimenter, entraîner et valider les modèles.

2. Dossier `models`

Structure d'un organe (ex: `models/blood`) :

```
# models/blood/malaria_model.py
class MalariaModel:
    def predict(self, image) -> dict:
        # Logique de prédiction (classification/détection/segmentation)
        return {
            "model_type": "classification",
            "results": {
                "disease": "malaria",
                "confidence": 0.95,
                "bbox": [],  # Optionnel (détection)
                "mask": []   # Optionnel (segmentation)
            }
        }
```

`main.py` d'un organe (ex: `models/blood/main.py`) :

```
# Charge tous les modèles de l'organe "sang"
from .malaria_model import MalariaModel
from .anemia_model import AnemiaModel

class BloodAnalyzer:
    def __init__(self):
        self.malaria_model = MalariaModel()
        self.anemia_model = AnemiaModel()

    def run(self, image):
        results = {
            "malaria": self.malaria_model.predict(image),
            "anemia": self.anemia_model.predict(image)
        }
        return results
```

---

3. Fichier `main.py` Racine

- Orchestre les appels aux différents organes :

```
from models.blood.main import BloodAnalyzer
from models.lung.main import LungAnalyzer

def analyze_image(image, organ):
    if organ == "blood":
        return BloodAnalyzer().run(image)
    elif organ == "lung":
        return LungAnalyzer().run(image)
    # ...
```

---

4. Dossier `healthbox`

- **Frontend** :
    - Interface utilisateur pour uploader des images.
    - Affiche les résultats en format JSON ou visuel (masques/boîtes de détection).
- **Backend** :
    - Reçoit les images, appelle `main.py` racine, et renvoie les résultats au frontend.

---

5. Dockerisation

- **Dockerfile** :

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "main.py"]
```

- **docker-compose.yml** :

```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
  # Ajouter d'autres services (ex: base de données) si besoin
```

Workflow

1. **Utilisateur** : Upload une image via l'interface `healthbox`.
2. **Backend** : Détermine l'organe cible et appelle `analyze_image()` dans `main.py` racine.
3. **Modèles** : Le `main.py` de l'organe exécute tous les modèles associés.
4. **Sortie** : Les résultats agrégés sont renvoyés au frontend en JSON.

Avantages

- **Modulaire** : Ajoutez des organes/maladies sans impacter le code existant.
- **Standardisé** : Format JSON commun pour tous les modèles.
- **Évolutif** : Déploiement simplifié via Docker.

Remarques supplémentaires

- La structure est conçue pour être facilement extensible. Par exemple, ajouter un nouvel organe revient à créer un nouveau dossier dans `models/` avec son propre `main.py` et ses modèles spécifiques.
- Le format JSON standardisé facilite l'intégration avec d'autres systèmes ou API.
