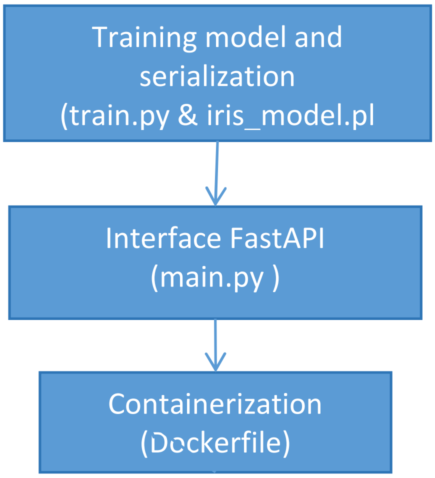
Project Title: Database & Python ETL with Reproducible Infrastructure

1. **Objective :**To deploy a Machine Learning model using FastAPI and Docker, providing a lightweight and reproducible setup for serving ML predictions through REST API endpoints. The project demonstrates how to load a trained model and deploy as an API endpoint fully reproducible via Docker.
2. **Architecture Overview :**



Data Flow :

Client Request (JSON)

↓

FastAPI Endpoint (/predict)

↓

Load iris_model.pkl

↓

Model.predict_proba(features)

↓

Response (Predicted class + probability)

3. **Technology Stack :**

Component	**Technology**	**Purpose**
Programming Language	Python 3.10	ML logic & API framework
Framework	FastAPI	RestfulAPI for prediction
Libraries	Scikit-learn,pydantic,numpy	Model training & inference
Serialization	Pickle	ML saved model
Web server	Uvicorn	Server for fastAPI
Containerization	Docker	Reproducible Deployment

4. Data Flow

Model Training (train.py)

- A set of numeric features (16 parameters) representing patient attributes relevant to thyroid detection.
- Loads the Iris dataset from sklearn.datasets.
- Constructs a Pipeline:
  - StandardScaler for normalization
  - LogisticRegression for classification
- Splits data into train/test sets (80/20).
- Trains the pipeline and evaluates accuracy (~95–98%).
- Serializes the trained model into models/iris_model.pkl using pickle

Interface(main.py)

- Loads the model (iris_model.pkl) is loaded via the pickle module.

```python
def load_model():
    if not MODEL_PATH.exists():
        raise FileNotFoundError(f"Model file not found at {MODEL_PATH.resolve()}. "
                                f"Run `python train.py` first.")
    with open(MODEL_PATH, "rb") as f:
        model = pickle.load(f)
    return model
```

- Validates incoming JSON requests with Pydantic.

```python
from pydantic import BaseModel, Field, conlist

IrisFeatureVector = conlist(float, min_length=4, max_length=4)
```

- Passes the feature vector to model.predict_proba().

```python
@app.post("/predict", response_model=PredictResponse)
def predict(req: PredictRequest):
    try:
        X = [req.features]
        proba = model.predict_proba(X)[0].tolist()
        cls = int(max(range(len(proba)), key=lambda i: proba[i]))
        return PredictResponse(class_id=cls, class_name=CLASS_MAP[cls], proba=proba)
    except Exception as e:
        raise HTTPException(status_code=400, detail=str(e))
```

- Returns the predicted class name (setosa, versicolor, or virginica) with probabilities.

Output

- The ML model is deployed an an API and runs succesfully with prediction

5. API Specification

/health : get() - Health check and returns the status

/predict : post() - Takes 4 features as input and returns prediction with probability

6. Environmental Variables

MODEL_PATH : Path to serialized model - iris_model.pkl

PORT : FastAPI Port : 8000

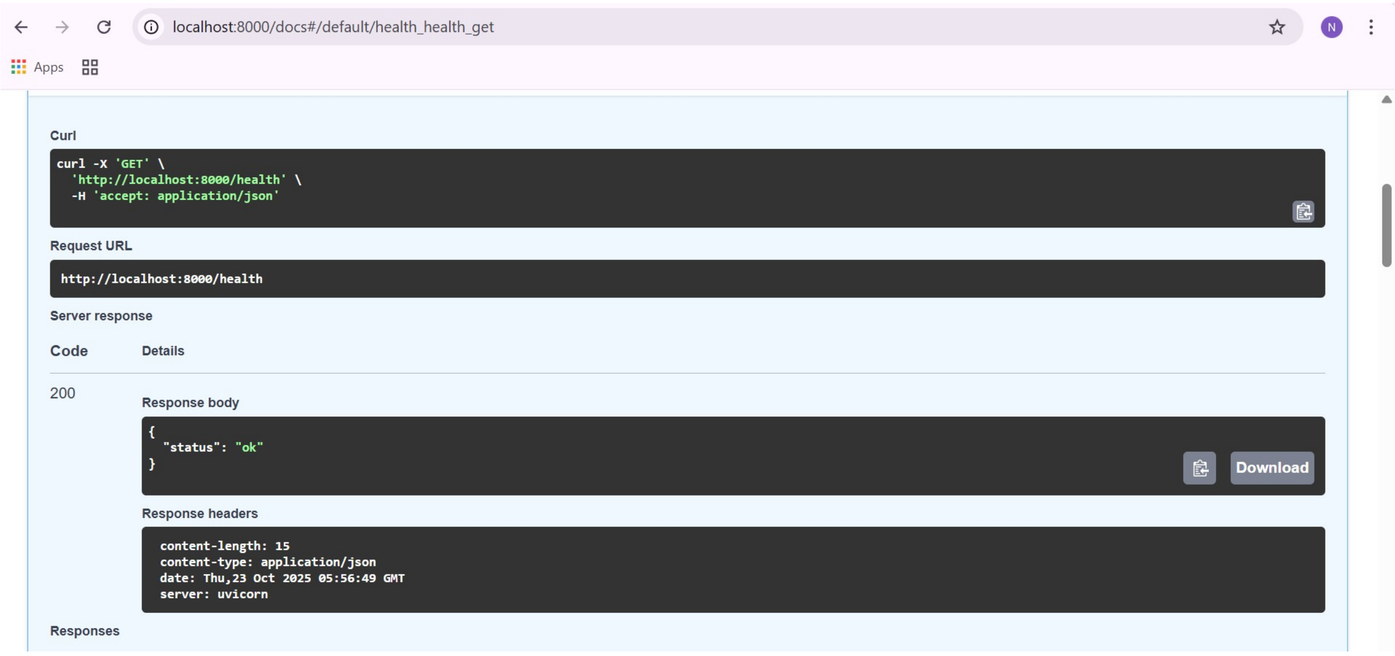
7. Dockerisation

Container - iris_api

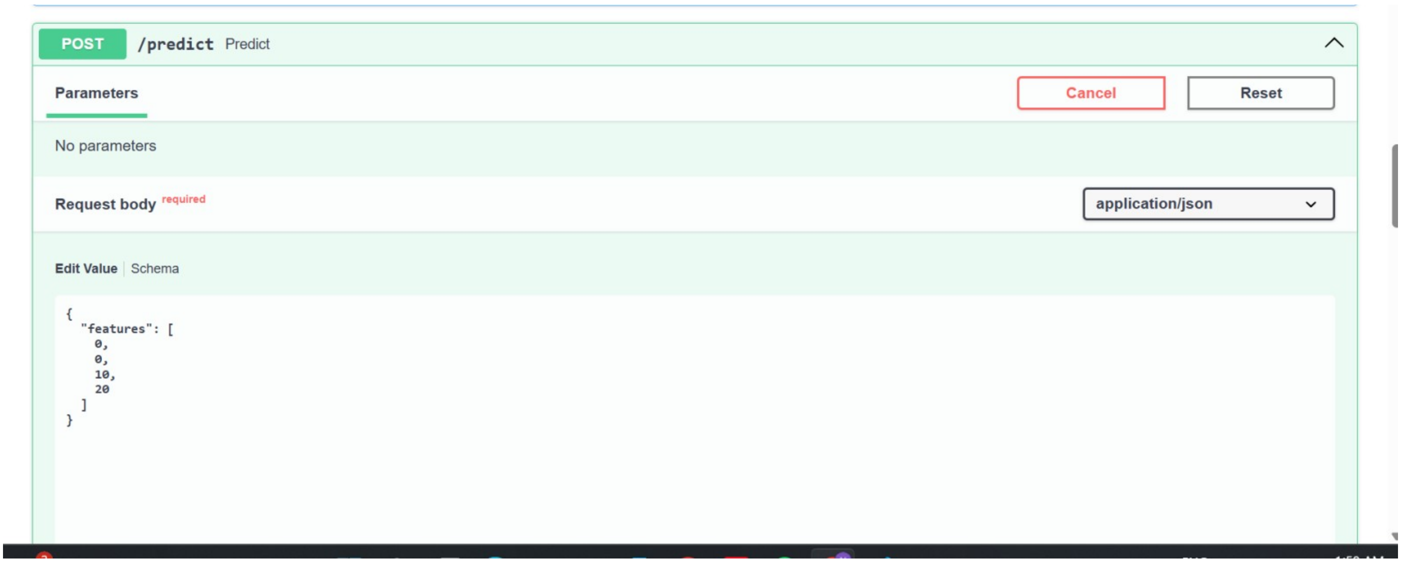
```dockerfile
FROM python:3.11-slim

ENV PYTHONUNBUFFERED=1 \
    PYTHONDONTWRITEBYTECODE=1

WORKDIR /app

RUN apt-get update && apt-get install -y --no-install-recommends \
    build-essential \
    && rm -rf /var/lib/apt/lists/*

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

COPY app/ app/

COPY iris_model.pkl .

EXPOSE 8000
CMD ["uvicorn", "app.main:app", "--host", "0.0.0.0", "--port", "8000"]

```

8. Testing & Validation

Launch - Check Health : Returns Status: ok



Prediction : localhost:8000/predict : Predict which class it belongs.



Validate Swagger UI at → http://localhost:8000/docs

9. Reproducibity

- All components (model, code, dependencies) are defined under Docker.
- requirements.txt fully pins package versions
- The solution can be build using the commands
  - Docker build --no cache -t iris-api
  - Git : GitHub