# **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 :**

Training model and serialization  
(train.py & iris\_model.pl

Interface FastAPI   
(main.py )

Containerization (Dockerfile)

**Data Flow :**

**Client Request (JSON)**

**↓**

**FastAPI Endpoint (/predict)**

**↓**

**Load iris\_model.pkl**

**↓**

**Model.predict\_proba(features)**

**↓**

**Response (Predicted class + probability)**

1. **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** |

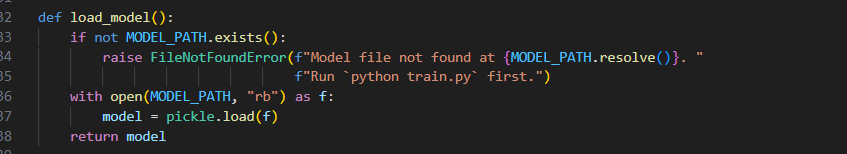
1. **Data Flow**

**Model Training (train.py)**

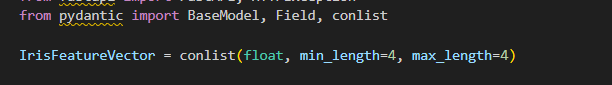
* A set of numeric features (16 parameters) representing patient attributes relevant to thyroid detection.
* Loads the Iris dataset from sklearn.datasets.
* Constructs a Pipeline:
  + - StandardScaler for normalization
    - LogisticRegression for classification
* Splits data into train/test sets (80/20).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

****Interface(main.py)****

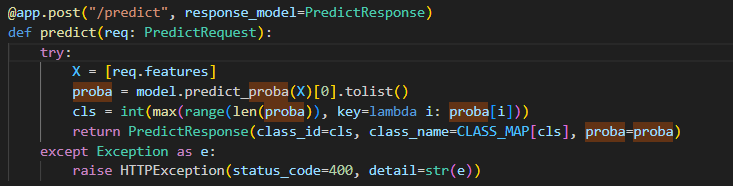
* Loads the model (iris\_model.pkl) is loaded via the pickle module.



* Validates incoming JSON requests with Pydantic.



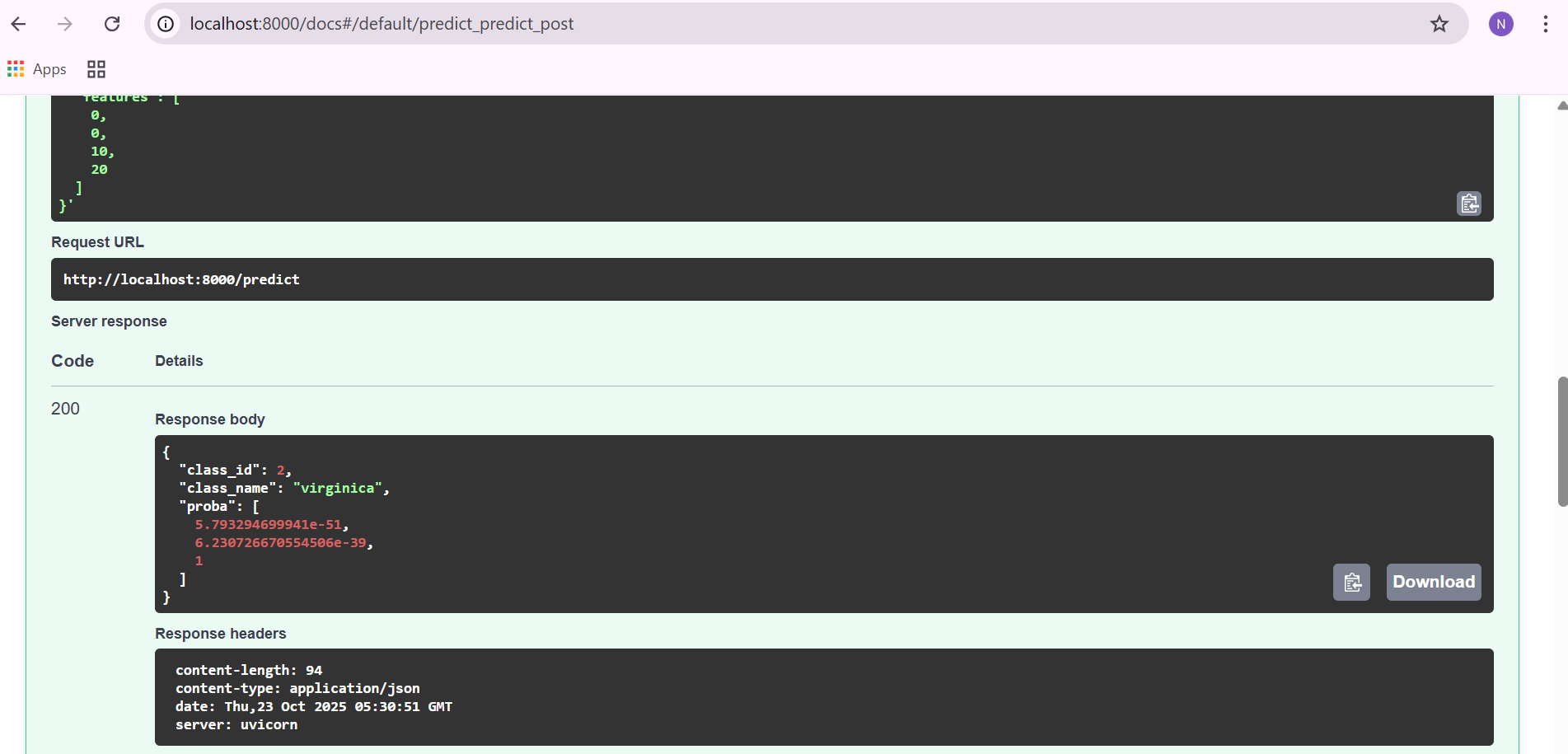
* Passes the feature vector to model.predict\_proba().



* 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



1. **API Specification**

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

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

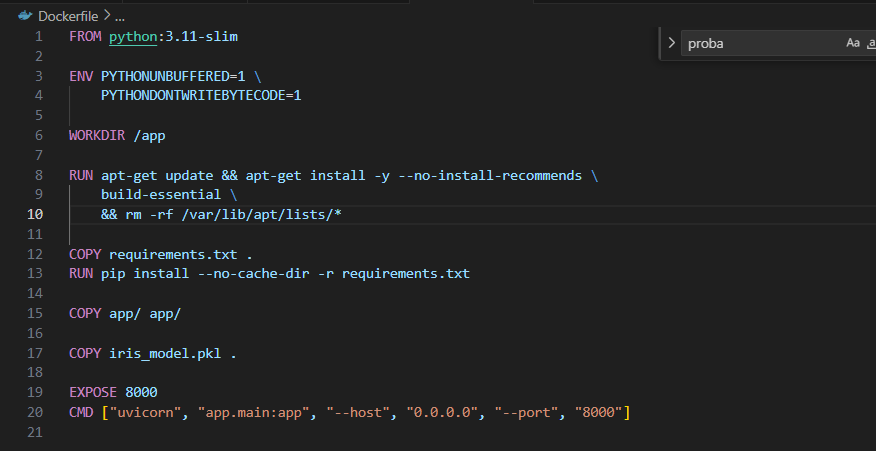
1. **Environmental Variables**

MODEL\_PATH : Path to serialized model - iris\_model.pkl

PORT : FastAPI Port : 8000

1. **Dockerisation**

**Container - iris\_api**

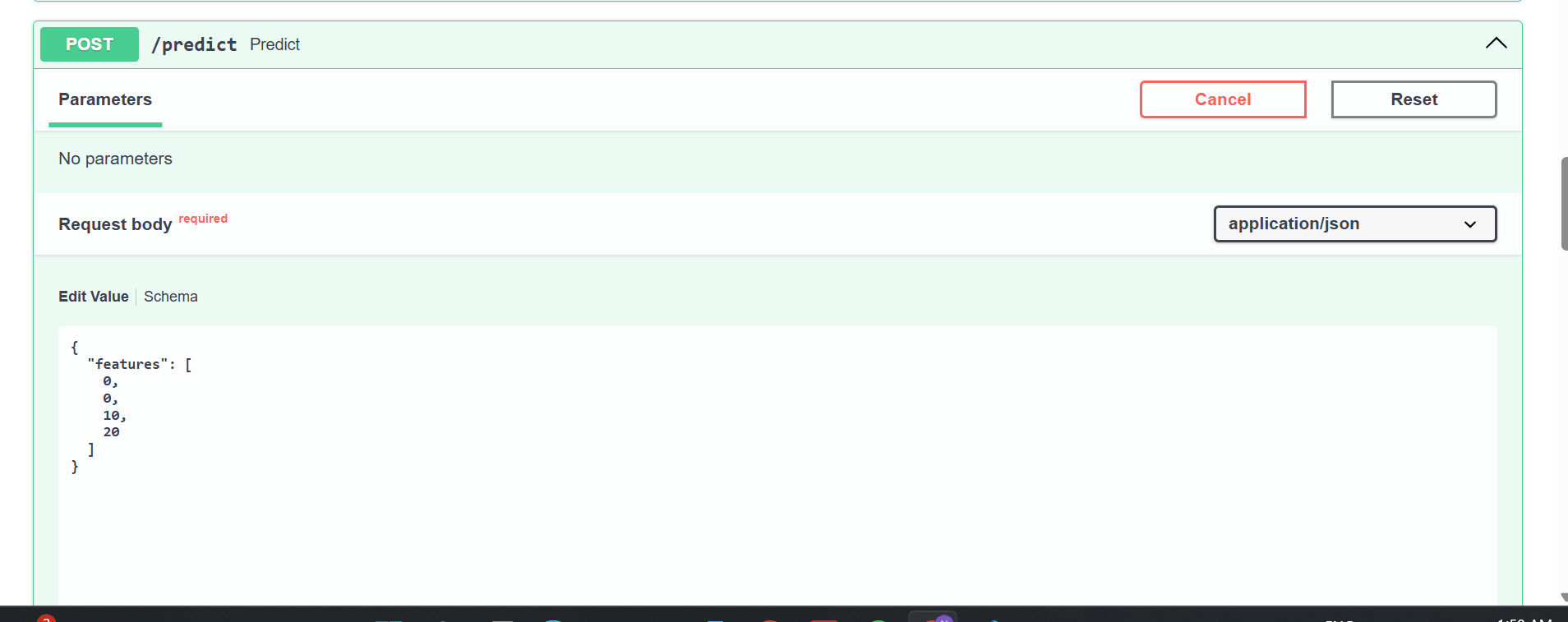


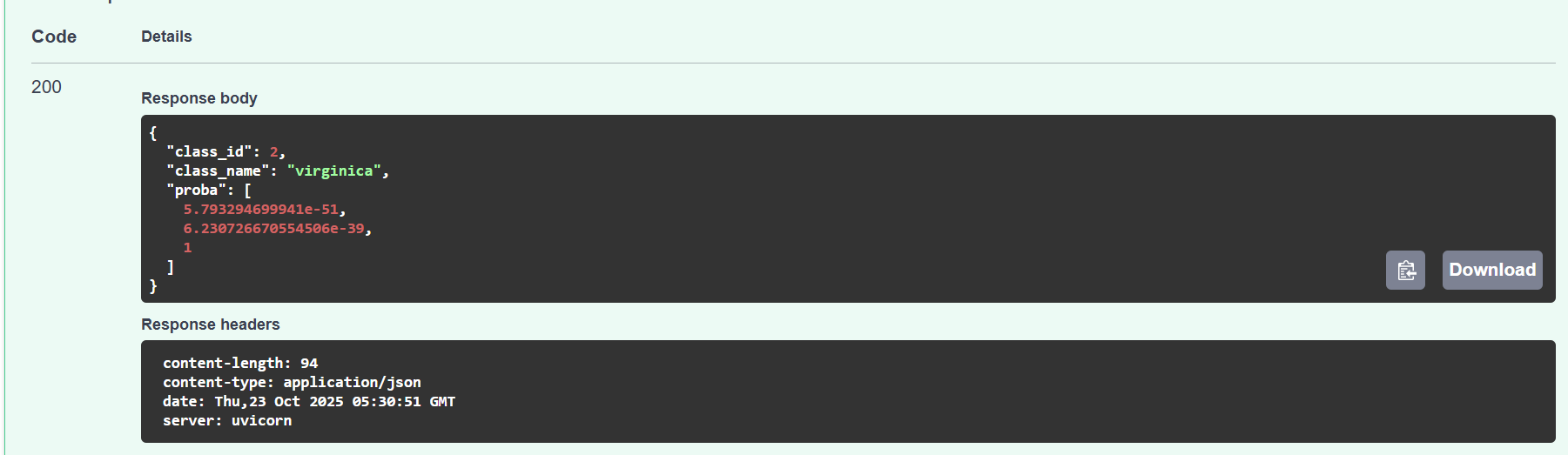
1. **Testing & Validation**

**Launch - Check Health :** Returns Status: ok



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





Validate Swagger UI at → [http://localhost:8000/docs](http://localhost:8000/docs" \t "_new)

1. **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