The following document outline the entire approach through which the classifier was deployed

1. **Step 1: Data Preprocessing and Model Building:**

**Loading and preprocessing data**

A screen shot of a computer program

AI-generated content may be incorrect.

**Model Training**

Data Loading & Preprocessing – Reads the Iris dataset from data/Iris.csv, cleans column names, splits into features (X) and target (y), and then into train/test sets (80/20 split).

MLflow Setup – Creates a models directory, sets up an MLflow experiment (iris\_experiment) to track parameters, metrics, and models.

LightGBM Training & Tuning – Uses RandomizedSearchCV with a restricted hyperparameter grid for LightGBM, trains the best model, evaluates accuracy, saves it with joblib, and logs model, parameters, metrics, and signature to MLflow.

Random Forest Training & Tuning – Repeats the same process for a RandomForestClassifier with its own hyperparameter search, accuracy evaluation, model saving, and MLflow logging.

Error Handling & Output – Wraps each training block in try/except for error reporting, suppresses LightGBM logs for cleaner output, and confirms when training is completed successfully.

1. **Step 2: API Development with FastAPI**

**Setup & Model Loading** – Uses nest\_asyncio for Jupyter compatibility, loads a pre-trained RandomForest model (random\_forest\_model.pkl), and retrieves feature names.

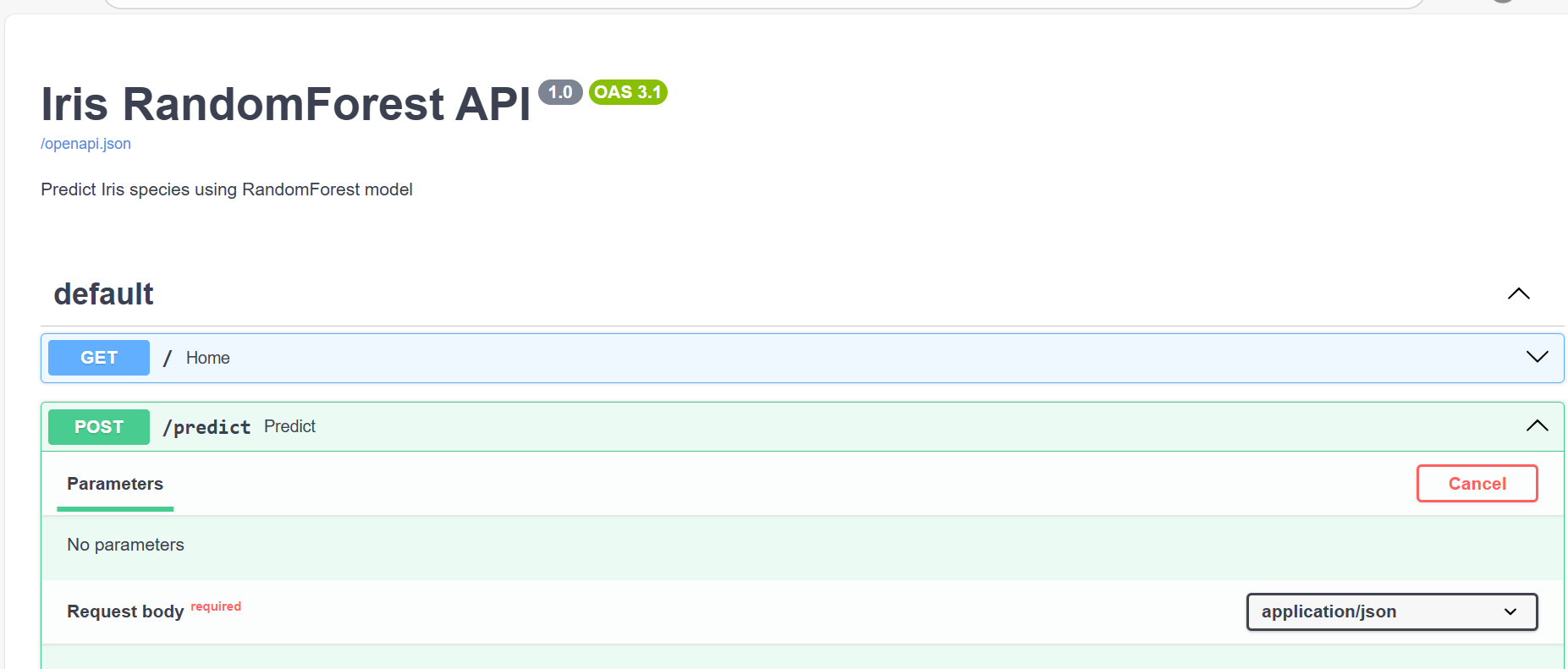
**FastAPI Initialization** – Creates a FastAPI app with metadata (title, description, version).

**Data Model** – Defines a Pydantic model (IrisInput) for validating incoming JSON with four Iris features.

**Endpoints** – / returns a health check message, /predict takes input features, converts them to a DataFrame, and returns the predicted species.

**Server Execution** – Runs the FastAPI app locally on port 7860 via uvicorn.

The Fast API app screenshot is attached on the next page

****

**STEP 3: App testing with swagger**

**Sample Input**{

"sepal\_length\_cm": 6.7,

"sepal\_width\_cm": 3.1,

"petal\_length\_cm": 4.7,

"petal\_width\_cm": 1.5

}

A black and white rectangle

AI-generated content may be incorrect.

**Part 4 – CI/CD with GitHub Actions**

The CI/CD pipeline automates:

1. **Linting & Testing**
   * Triggered on each push or pull request to main.
   * Runs flake8 for style checks and pytest for unit tests.
   * Ensures code quality before deployment.
2. **Build & Push Docker Image**
   * Builds the FastAPI + model Docker image.
   * Tags with commit SHA for version tracking.
   * Pushes image to Docker Hub using secrets DOCKERHUB\_USERNAME and DOCKERHUB\_TOKEN.
3. **Deployment (Local or Cloud)**
   * Deployment scripts can pull the latest image and run:
   * docker run -p 8000:8000 yourdockeruser/mlops-iris:<tag>
   * Supports deployment on local machine, AWS EC2, or LocalStack for testing.

**Workflow File – .github/workflows/ci.yml:**

A screen shot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

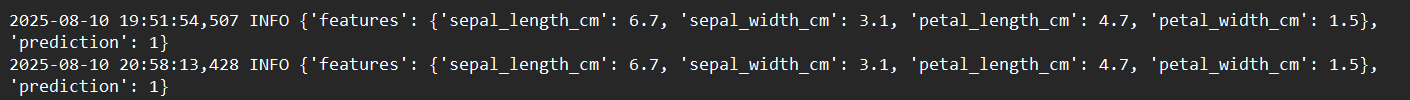
A screen shot of a computer program

AI-generated content may be incorrect.

**Part 5 – Logging and Monitoring**

**Implementation:**

1. **File Logging**
   * All prediction requests and outputs are appended to logs/predictions.log using Python’s logging module.
   * Example log entry:



1. **SQLite Logging**
   * logs/predictions.db stores structured logs with columns: timestamp, features, prediction.
   * Enables querying history and generating analytics.
2. **Monitoring with Prometheus**
   * /metrics endpoint exposes:
     + api\_request\_count{endpoint="/predict",status="200"}
     + Can be scraped by Prometheus for dashboarding in Grafana.
   * Change in Prometheus.yml:

A computer screen with green text

AI-generated content may be incorrect.

1. **Optional /logs endpoint**
   * Returns last N prediction logs from SQLite.

**API Snippet – Logging + Metrics**

**A screen shot of a computer code

AI-generated content may be incorrect.**

**A computer screen shot of a program code

AI-generated content may be incorrect.**

**Architecture Diagram**

Below is an updated diagram showing **CI/CD**, **logging**, **monitoring**, and **deployment** flow.

A diagram of a data flow

AI-generated content may be incorrect.

**Part 6 Summary:**   
The MLOps pipeline integrates:

* **GitHub Actions** for CI/CD automation.
* **Docker** for consistent packaging & deployment.
* **FastAPI** for serving the ML model as a REST API.
* **Logging** to both files and SQLite for traceability.
* **Prometheus/Grafana** for observability.
* **MLflow** for experiment tracking and model registry.