# Object Detection Microservice — Test & Verification Guide

Prepared for assignment submission • Python 3.11 • YOLOv3-tiny • Flask + OpenCV

## Overview

This document explains how to run the repository locally, test the object detection flow, and verify that the system produces both: **(A)** annotated image with bounding boxes, and **(B)** a JSON file describing detections. Follow each step in a Windows PowerShell terminal (or WSL if you prefer).

## Prerequisites

- Windows OS (PowerShell). The commands below use PowerShell syntax.
- Python 3.11 installed and added to PATH.
- Git installed and configured with access to the repository.
- Docker & Docker Compose (optional — for containerized tests).
- Repository extracted at: `F:\PROJECTS\object-detection-microservice` (adjust paths if different).

## Files & Structure (what to expect)

> **Main folders:**
>
> - `ai-backend/` — Flask service that runs YOLO and saves annotated images + JSON outputs.
> - `ui-backend/` — Flask UI proxy that uploads images to AI backend and exposes results.
> - `models/` — model files (YOLO .weights and .cfg). Ensure these are present.
> - `test_images/` — sample images (e.g. `sample.jpg`, `sample2.jpg`).
> - `outputs/` — generated annotated images and JSON results (created at runtime).
> - `docker-compose.yml` — optional container orchestration file (if you want Docker).

## Part A — Run locally (virtualenv method)

### 1. Start AI backend (port 5001)

Open a PowerShell window and run:

```
cd F:\PROJECTS\object-detection-microservice\ai-backend
python -m venv venv
.\venv\Scripts\Activate.ps1
pip install -r requirements.txt
python app.py
```

**Expected startup lines:**

```
* Serving Flask app 'app'
* Debug mode: on
* Running on http://127.0.0.1:5001
```

### 2. Start UI backend (port 5000) — new PowerShell

```
cd F:\PROJECTS\object-detection-microservice\ui-backend
python -m venv venv
.\venv\Scripts\Activate.ps1
pip install -r requirements.txt
python app.py
```

**Expected:** `* Running on http://127.0.0.1:5000`

### 3. Health check (optional)

```
Invoke-WebRequest -Uri "http://127.0.0.1:5000/health" | Select-Object -Expand Content | ConvertFrom-Json
```

**Expected JSON:**

```
{
  "ui": "ok",
  "ai": { "status": "ok" }
}
```

> If the AI shows **"down"**, ensure the AI backend terminal is still running and not stuck loading the model or throwing errors.

## Part B — Single image detection (curl)

Test a single image upload using PowerShell. Use the file that exists on disk. Note extension (`.jpg` vs `.jpeg`).

### PowerShell-safe curl command (use single quotes)

```
curl.exe -X POST -F 'image=@"F:/PROJECTS/object-detection-microservice/test_images/sample.jpg"' http://127.0.0.1:5000/de
```

**What the UI backend will do:**

1. Receive uploaded image and forward to the AI backend.
2. AI backend runs detection and produces:
   - Annotated image saved to `outputs/result_`
   - JSON saved to `outputs/result_.json`
3. UI backend responds with detection JSON and includes `output_url` and `output_json_url`.

### Example expected HTTP response (JSON)

```
{
  "detections": [
    {
      "bounding_box": { "height": 207, "width": 193, "x": 4, "y": 20 },
      "class": "dog",
      "confidence": 0.9906169772148132
    }
  ],
  "output_filename": "result_sample.jpeg",
  "output_json": "result_sample.json",
  "output_url": "/output/result_sample.jpeg",
  "output_json_url": "/output/result_sample.json"
}
```

### Verify files on disk

```
dir F:\PROJECTS\object-detection-microservice\outputs
```

You should see both `result_sample.jpeg` (annotated image) and `result_sample.json` (detections data).

### View annotated image in browser

Open in browser or navigate to the UI backend endpoint:

```
http://127.0.0.1:5000/output/result_sample.jpeg
```

## Part C — Testing multiple images (batch)

To test all images in `test_images/`, run this one-liner in PowerShell from the project root:

```
Get-ChildItem "F:\PROJECTS\object-detection-microservice\test_images" -Include *.jpg,*.jpeg,*.png -File | ForEach-Objec
```

```
  $p = $_.FullName -replace '\\','/';
  Write-Host "Testing $p";
  curl.exe -s -X POST -F ("image=@{0}" -f $p) http://127.0.0.1:5000/detect | Out-String;
  Start-Sleep -Milliseconds 300;
}
```

This loops through every image and sends it to the UI backend. After the loop, check `outputs/` folder for corresponding image + JSON pairs.

## Part D — Expected JSON format (schema)

Each JSON saved to disk follows this structure:

```
{
  "detections": [
    {
      "bounding_box": { "x": int, "y": int, "width": int, "height": int },
      "class": "string",
      "confidence": float
    },
    ...
  ],
  "output_filename": "result_.",
  "output_json": "result_.json"
}
```

Confidence is a float between 0.0 and 1.0, bounding box coordinates are pixel values relative to the input image size.

## Part E — Docker-based test (optional)

If you prefer to run the services in containers (recommended for consistent environments), use docker-compose:

```
cd F:\PROJECTS\object-detection-microservice
docker-compose up --build
```

Then run the same `curl` test shown earlier (UI still on `127.0.0.1:5000`).

## Troubleshooting & common issues

#### 1) `curl: (26) Failed to open/read local data`

- Most often: wrong path or file extension. Verify with `Test-Path "F:\...\sample2.jpg"`.
- Use PowerShell-safe quoting: `curl.exe -X POST -F 'image=@"F:/.../sample2.jpg"' http://127.0.0.1:5000/detect`.

#### 2) UI health shows AI as "down"

- Check AI backend terminal — is it starting fine? Look for model-loading errors.
- If AI takes time to load model, increase the UI health-check timeout (in `ui-backend/app.py` health route).

#### 3) Python module import errors (e.g., requests not found)

- Activate the correct virtualenv and run `pip install -r requirements.txt` for the service folder (ai-backend and ui-backend may have their own venvs).

#### 4) Indentation or syntax errors after applying automated patches

- Open the file in an editor and ensure consistent 4-space indentation for Python blocks.

## Submission checklist (what to include with the assignment)

1. Link to GitHub repository or zipped project.
2. Short README describing how to run locally and with Docker (this doc can be included).
3. Screenshots of:
    - Terminal showing both services running on `127.0.0.1:5000` and `:5001`.

    - Example curl response JSON.
    - Annotated output image showing bounding boxes.
4. Include sample outputs from `outputs/` — both `result_sample.jpeg` and `result_sample.json`.

Prepared for: **Ullas G H**
Notes: This guide assumes default host/port settings in the repo (UI: 5000, AI: 5001). If you changed ports, update curl and health-check commands accordingly.