# Building an API server

## Foreword

In my experience, collecting and interfacing with data are usually by far the most difficult part of a project, and the single most significant factor which determine whether some downstream application will work or not.

## What do we need from a basic backend?

A backend contains much of the business logic of an online application. Tasks which require more computing power than a clients device can provide should take place on a backend, as well as tasks which require sensitive data that we do not want to expose to the user. The backend may include a database and any database queries.

## Building an image classifier backend with flask and yolo

First, we'll set up the basic structure of our Flask application. Create a directory `yolo`. Inside this directory, create the following files.

### Step 1: Setting up the Flask application

Create a file named `__init__.py` with the following content:

```python
import os
from flask import Flask
from flask_cors import CORS

def create_app(test_config=None):
    app = Flask(__name__, instance_relative_config=True)

    # Register blueprints
    from . import database, classifier
    app.register_blueprint(database.database_service, url_prefix="/database")
    app.register_blueprint(classifier.classifier_service, url_prefix="/classifier")

    # Load configuration
    if test_config is None:
        app.config.from_pyfile('config.py', silent=True)
    else:
        app.config.from_mapping(test_config)

    # Ensure instance folder exists
    try:
        os.makedirs(app.instance_path)
    except OSError:
        pass

    # Simple route for testing
    @app.route('/hello')
    def hello():
        return 'Hello, World!'
    # Enable CORS
    CORS(app)

    return app
```

This sets up the basic Flask application, registers blueprints for database and classifier services, and enables CORS.

## Step 2: Creating the database service

Next, we'll create a `database.py` file to handle database operations:

```
from flask import Blueprint, request, jsonify, current_app
import psycopg
import psycopg.rows

database_service = Blueprint("database", __name__)

def connect_postgres() -> psycopg.Connection[psycopg.rows.TupleRow]:
    database_client = psycopg.connect(
        host=current_app.config["POSTGRES_HOST"],
        port=current_app.config["POSTGRES_PORT"],
        user=current_app.config["POSTGRES_USER"],
        password=current_app.config["POSTGRES_PASSWORD"],
        dbname=current_app.config["POSTGRES_DB"],
        row_factory=psycopg.rows.dict_row
    )
    return database_client

@database_service.route("/create_table", methods=["POST"])
def create_table():
    with connect_postgres() as client:
        cursor = client.cursor()
        cursor.execute(
            "CREATE TABLE classes (id UUID PRIMARY KEY, name TEXT)"
        )

@database_service.route("/get_table", methods=["GET"])
def get_table():
    with connect_postgres() as client:
        cursor = client.cursor()
        cursor.execute("SELECT * FROM classes")
        return jsonify(cursor.fetchall())

@database_service.route("/add_class", methods=["POST"])
def add_class():
    with connect_postgres() as client:
        cursor = client.cursor()
        data = request.form
        cursor.execute(
            "INSERT INTO classes (id, name) VALUES (gen_random_uuid(), %s)",
            (data.get("class"),)
        )
    return jsonify({"message": "Class added successfully"}), 201
```

This service provides routes for creating a table, getting all classes, and adding a new class.

### Step 3: Implementing the classifier service

Create a `classifier.py` file for the image classification functionality:

```python
from flask import Blueprint, request, jsonify
from ultralytics import YOLO
import io
from PIL import Image
import ultralytics.engine.results

classifier_service = Blueprint("classifier", __name__)

model = YOLO("yolo11n-cls.pt")

@classifier_service.route("/classify", methods=["POST"])
def classify():
    file = request.files["image"]
    img_bytes = file.read()
    img = Image.open(io.BytesIO(img_bytes))

    results: list[ultralytics.engine.results.Results] = model(img)

    formatted_results = []
    for r in results:
        name = r.names
        probs = r.probs
        top_probs = probs.top5

        formatted_result = {
            "top_classes": [
                {"name": name[top_probs[i]], "probability": float(probs.top5conf[i])}
                for i in range(5)
            ]
        }
        formatted_results.append(formatted_result)

    return jsonify(formatted_results)
```

This service provides a route for classifying images using the YOLO model.

## Setting up the environment and running the services

### Step 4: Docker Compose file

Back in the top-level directory, create a docker-compose.yml file to set up the required services:

```yaml
services:
  postgrest:
    image: postgrest/postgrest
    ports:
      - "3000:3000"
    environment:
      PGRST_DB_URI: postgres://app_user:password@db:5432/app_db
      PGRST_OPENAPI_SERVER_PROXY_URI: http://127.0.0.1:3000

  postgres:
    image: postgres
    ports:
      - "5432:5432"
    environment:
      POSTGRES_DB: app_db
      POSTGRES_USER: app_user
      POSTGRES_PASSWORD: password

  pgadmin:
    image: dpage/pgadmin4
    ports:
      - "5050:80"
    environment:
      PGADMIN_DEFAULT_EMAIL: your_email@example.com
      PGADMIN_DEFAULT_PASSWORD: your_password
```

This sets up PostgreSQL, PostgREST, and pgAdmin services.

## Step 5: Install the package manager UV

For this lab, we will use the package manager UV to manage our virtual environment and python packages. See https://docs.astral.sh/uv/getting-started/installation/ for installation instructions. If you are on Mac or Linux, you will run:

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

After UV is installed, we can download all the dependencies specified in the `pyproject.toml` file by running `uv sync`. Then activate the environment with `source .venv/bin/activate`.

There is one package which requires a little more, so run `uv pip install "psycopg[binary]"`

### Alternate method

If uv has errors, you can do this manually.

```
pip install "psycopg[binary]"
```

then

```
pip install flask flask-cors ultralytics pillow
```

## Step 6: Running the application

Start the Docker services:

```
docker-compose up -d
```

Set up the environment variables (you may want to create a .env file):

```
export FLASK_APP=your_app_name
export FLASK_ENV=development
export POSTGRES_HOST=localhost
export POSTGRES_PORT=5432
export POSTGRES_USER=app_user
export POSTGRES_PASSWORD=password
export POSTGRES_DB=app_db
```

Run the Flask application:

`flask --app yolo run` (or `python3 -m flask --app yolo run`)

In a new terminal tab, run the frontend. Navigate to the frontend directory, then do `npm install` and `npm run dev`. It will tell you the URL to go to, mine was `http://localhost:5173/`

Your image classifier backend is now up and running! Navigate to that url, plug in your favorite image and see what it classifies it as.