# Sprint 1 — Agent Skeleton & Fixtures (Compact)

**Goal**: establish a small, runnable agent skeleton using local fixtures (no external crawling). Keep files minimal so you can iterate quickly.

## What I added (compact)

**backend/**

* app/agents/base\_agent.py — abstract BaseAgent (ABC) that other agents inherit.
* app/agents/discovery.py — small fixture-based DiscoveryAgent that reads backend/fixtures/seed\_posts.json.
* app/agents/orchestrator.py — Orchestrator that runs discovery → (placeholder) pipeline and produces clusters + brief.
* fixtures/seed\_posts.json — two small post fixtures (captions, hashtags, image paths).
* app/main.py — simplified FastAPI entry that uses Orchestrator for /run.

## Quick dev notes

1. **Purpose**: keep everything local and deterministic so we can validate orchestration before adding parsing/vision/cluster complexity.
2. **Next steps after this sprint**: add parser.py, vision\_palette.py, and cluster.py one at a time and wire them into the Orchestrator.
3. **Requirements**: Sprint 1 needs only fastapi + uvicorn. Later steps will add pillow, scikit-learn, etc.

## How to test (commands)

# from project root  
cd backend  
python -m venv .venv && source .venv/bin/activate # Windows: .venv\Scripts\activate  
pip install fastapi uvicorn  
uvicorn app.main:app --reload --port 8000

Then POST a seed to /run (example):

curl -s -X POST http://localhost:8000/run \  
 -H "Content-Type: application/json" \  
 -d '{"season":"SS25","keywords":["olive","pleated"]}' | python -m json.tool

You should get a JSON response containing clusters, brief\_md, and items (the fixtures).

## Small reminder

I kept the canvas compact as requested. If you want the full code pasted into chat instead of copying from canvas, say so and I will paste the first file (BaseAgent) directly here. Otherwise, copy the files from the canvas into your backend/app/agents/ and run the server. When you have base\_agent in place and the /run returns fixture items, reply `` and I will push the next agent (parser) and instructions.

## Sprint 1.1 — BaseAgent (file added)

**File path (create this in your repo):** backend/app/agents/base\_agent.py

**What I added to the canvas:** the complete BaseAgent implementation that standardizes agent lifecycle (start/stop), logging, simple error handling, and a run\_agent wrapper that subclasses should call via run.

**File contents (already saved in the canvas):**

from abc import ABC, abstractmethod  
from typing import Any, Dict, Optional  
import logging  
import uuid  
from datetime import datetime  
  
logger = logging.getLogger(\_\_name\_\_)  
  
class BaseAgent(ABC):  
 """Minimal synchronous BaseAgent for the PoC.  
  
 Subclasses MUST implement `run(self, ctx: Dict[str,Any]) -> Dict[str,Any]`.  
 Use `run\_agent(ctx)` as the standard entrypoint (it wraps start/stop and  
 basic error handling).  
 """  
  
 def \_\_init\_\_(self, name: Optional[str] = None, config: Optional[Dict[str,Any]] = None):  
 self.name = name or self.\_\_class\_\_.\_\_name\_\_  
 self.config = config or {}  
 self.id = str(uuid.uuid4())  
 self.started\_at: Optional[datetime] = None  
 self.ended\_at: Optional[datetime] = None  
 self.logger = logger  
  
 def start(self) -> None:  
 self.started\_at = datetime.utcnow()  
 self.logger.info(f"[{self.name}] start id={self.id}")  
  
 def stop(self) -> None:  
 self.ended\_at = datetime.utcnow()  
 self.logger.info(f"[{self.name}] stop id={self.id}")  
  
 def run\_agent(self, ctx: Dict[str, Any]) -> Dict[str, Any]:  
 """Wrapper handling lifecycle and errors; returns a serializable dict."""  
 self.start()  
 try:  
 result = self.run(ctx)  
 if result is None:  
 result = {}  
 return result  
 except Exception as exc:  
 # Agents should not crash the orchestrator; surface error info.  
 self.logger.exception(f"[{self.name}] error: {exc}")  
 return {"error": str(exc), "\_agent": self.name}  
 finally:  
 self.stop()  
  
 @abstractmethod  
 def run(self, ctx: Dict[str, Any]) -> Dict[str, Any]:  
 """Implement this in subclasses. Return a JSON-serializable dict."""  
 raise NotImplementedError  
  
 def to\_dict(self) -> Dict[str, Any]:  
 return {  
 "id": self.id,  
 "name": self.name,  
 "started\_at": self.started\_at.isoformat() if self.started\_at else None,  
 "ended\_at": self.ended\_at.isoformat() if self.ended\_at else None,  
 "config": self.config,  
 }

**How to apply locally**

1. Create the directories if they don’t exist:

* mkdir -p backend/app/agents

1. Open the canvas document (Trent Concept To Catwalk — Po C (mvp Build Kit)), copy the base\_agent.py contents into backend/app/agents/base\_agent.py or download the canvas file.
2. No new pip packages are required for this file.
3. (Optional) Restart your backend server. The file itself won’t change /run behaviour until a subclass uses it — that’s next.

**Next (Step 1.2)** When you’re ready, reply `` and I will push the next agent: discovery (a small fixture-based DiscoveryAgent subclass that returns the sample posts) and an updated orchestrator that uses BaseAgent.run\_agent. That will make /run actually call the orchestrated agents and return items from fixtures.

If you prefer I paste the file contents directly into chat as a code block for quick copy-paste instead of using the canvas download, tell me and I’ll paste it here.