The Winning Architecture: "The Hybrid Monorepo"

Instead of two separate projects, we will build **one project** where Django handles the data/admin and FastAPI handles the AI Agents.

- **Django:** Manages the Database (PostgreSQL/SQLite), User Auth, and the Admin Panel (to view Agent logs/results).
- **FastAPI:** Mounts *inside* or runs alongside Django. It exposes the endpoints that trigger the CrewAI agents.
- **The Bridge:** FastAPI will import Django models directly. This means your Agents can save data to the DB **without** making slow HTTP requests. This is the secret to winning.

🛠️ Step-by-Step Implementation Guide

Phase 1: Setup the Hybrid Core (Hours 0-2)

We need to make FastAPI and Django talk to the same database.

Create Django Project:

Bash

```
django-admin startproject hackathon_core
cd hackathon_core
python manage.py startapp core_db
```

1.

Define Models (core_db/models.py):

Create the table where Agents will save their data.

Python

```
from django.db import models

class AgentLog(models.Model):
    machine_id = models.CharField(max_length=100)
    status = models.CharField(max_length=50)
    risk_score = models.FloatField()
    recommendation = models.TextField()
    timestamp = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.machine_id} - {self.status}"
```

2.

Run Migrations:

Bash

```
python manage.py makemigrations
python manage.py migrate
```

3.

Phase 2: The FastAPI Bridge (Hours 2-4)

Now, create the FastAPI app that uses Django's database.

1. **Create api.py** in the same folder as manage.py.

The "Magic Glue" Code:
You must initialize Django inside the FastAPI script so it can access the database.
Python

```
# api.py
import os
import django
from fastapi import FastAPI, BackgroundTasks
from pydantic import BaseModel

# 1. INIT DJANGO
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'hackathon_core.settings')
django.setup()

# NOW we can import models!
from core_db.models import AgentLog

# 2. INIT FASTAPI
app = FastAPI()

# Pydantic Model for incoming data
class AgentResult(BaseModel):
    machine_id: str
    status: str
    risk_score: float
    recommendation: str

# 3. ENDPOINT: Agents call this to save data
@app.post("/api/save_log")
def save_log(data: AgentResult):
    # Using Django ORM synchronously (safe for simple inserts)
    log = AgentLog.objects.create(
        machine_id=data.machine_id,
        status=data.status,
        risk_score=data.risk_score,
        recommendation=data.recommendation
    )
    return {"msg": "Log Saved", "id": log.id}
```

2.

Phase 3: The CrewAI Integration (Hours 4-10)
This is where you build the agent that calls your new FastAPI endpoint.

1. **Create agents.py**:

2. **Create a Custom Tool** that sends data to your FastAPI endpoint (or just saves to DB directly since we are in the same repo!).
   Option A: The "Pure" Way (Agent calls API)
   - *Best if you want to show "Microservices" architecture to judges.*

Python
```
from crewai_tools import tool
import requests

@tool("Save to Database")
def save_to_db_tool(machine_id: str, status: str, risk_score: float, recommendation: str):
    """Saves the analysis result to the central database."""
    payload = {
        "machine_id": machine_id,
        "status": status,
        "risk_score": risk_score,
        "recommendation": recommendation
    }
    # Call local FastAPI endpoint
    response = requests.post("http://127.0.0.1:8000/api/save_log", json=payload)
    return f"Data saved successfully: {response.json()}"
```
3. **Option B: The "Winning" Way (Direct DB Access)**
   - *Faster, less code, less error-prone.*

Python
```
from crewai_tools import tool
from core_db.models import AgentLog # Import Django model directly!

@tool("Save to Database")
def save_to_db_tool(machine_id: str, status: str, risk_score: float, recommendation: str):
    """Saves the analysis result to the central database."""
    AgentLog.objects.create(
        machine_id=machine_id,
        status=status,
        risk_score=risk_score,
        recommendation=recommendation
    )
    return "Data saved to Django Database successfully."
```
4.

Phase 4: The Master Trigger (Hours 10-12)
You need an endpoint in FastAPI to **START** the Crew.

Update your api.py:

Python
```
from agents import praxis_crew # Import your Crew object
```

```
@app.post("/api/run_agent")
async def run_agent(background_tasks: BackgroundTasks):
    # Run the agent in the background so the API doesn't freeze
    background_tasks.add_task(praxis_crew.kickoff)
    return {"status": "Agent Started. Check Django Admin for results."}
```

Final Tech Stack Checklist

Layer	Technology	Why it Wins
Backend Core	**Django**	Gives you a ready-made **Admin Panel** (localhost:8000/admin). Judges love seeing data populate here in real-time.
API Layer	**FastAPI**	Handles the async nature of AI agents better than Django.
Database	**SQLite (Dev) / Postgres (Prod)**	Managed easily via Django Models.
AI Logic	**CrewAI**	Orchestrates the agents.
Agent "Hands"	**Django ORM**	Agents use the database directly via Python imports (Option B) for speed.
Frontend	**Streamlit**	Connects to the FastAPI endpoints to start agents and visualize data.

The "Secret Weapon" Demo Flow

1. Open **Django Admin** in one browser tab. Show it's empty.
2. Open **Streamlit** in another tab. Click "Start PraxisGuard."
3. Show the **Terminal** with CrewAI logs (Agents thinking...).
4. Switch back to **Django Admin** and hit refresh.
5. **BOOM:** The rows appear with "Critical Risk" and "Technician Scheduled."

6. This proves the full cycle: **UI -> FastAPI -> Agent -> DB -> Admin.**