```
// ================================================================================
// 🎯 COMPLETE WEDDING AI APP - SINGLE FILE
// ================================================================================

// ================================================================================
// 📊 DATA SOURCES & APIs
// ================================================================================

// Data Source: Wedding_API
// Base URL: https://app.nocodb.com/api/v2/tables
// Headers:
// content-type: application/json
// xc-token: YCQ_vAuT1Zyvv6yClfXbZCmaSt_hVARWZN8bESYl

// ================================================================================
// 🔧 ESSENTIAL APIs (Create these in Appsmith)
// ================================================================================

// API: CreateWedding
// Method: POST
// Path: /mg0w6xb658pvkqb/records
// Body: {{WeddingFormHandler.getWeddingPayload()}}

// API: CreateCouple
// Method: POST
// Path: /m6v4on94fj53j9n/records
// Body: {{WeddingFormHandler.getCouplePayload()}}

// API: CreatePreferences
// Method: POST
// Path: /mhkpiytp9rqyd20/records
// Body: {{WeddingFormHandler.getPreferencesPayload()}}

// API: CreateVendor
// Method: POST
// Path: /ma3xsdfzx3fgxdf/records
// Body: {{WeddingFormHandler.getVendorPayload()}}

// API: GetWeddings
// Method: GET
// Path: /mg0w6xb658pvkqb/records

// API: GetVendors
// Method: GET
// Path: /ma3xsdfzx3fgxdf/records

// API: AIVendorDiscovery (Your local CrewAI)
// Method: POST
// URL: https://your-ngrok-url.ngrok.io/discover-vendors
// Body: {{WeddingFormHandler.getAISearchPayload()}}

// ================================================================================
// 🎨 ESSENTIAL WIDGETS LAYOUT
// ================================================================================

/*
PAGE LAYOUT:
┌─────────────────────────────────────────────────────────────┐
```

```
/*
┌─────────────────────────────────────────────────────────────┐
│ 📝 WEDDING PLANNING FORM                                    │
├─────────────────────────────────────────────────────────────┤
│ Your Name: [Input_YourName]     Partner: [Input_Partner]    │
│ City: [Select_City]             Date: [DatePicker_Wedding]  │
│ Budget: [Select_Budget]         Guests: [Input_Guests]      │
│ Type: [Select_WeddingType]      Duration: [Select_Duration] │
│ Events: [CheckboxGroup_Events]                              │
│ Priorities: [CheckboxGroup_Priorities]                      │
│ Special Requirements: [TextArea_Requirements]               │
│                                                             │
│ [Button_SubmitWedding] [Button_AIDiscoverVendors]           │
├─────────────────────────────────────────────────────────────┤
│ 🤖 AI VENDOR RECOMMENDATIONS                                │
│ [Table_Vendors]                                             │
├─────────────────────────────────────────────────────────────┤
│ 📊 WEDDING DASHBOARD                                        │
│ [Table_Weddings] [List_Tasks] [Chart_Budget]                │
└─────────────────────────────────────────────────────────────┘
*/

// ================================================================================
// 🎬 WEDDING FORM HANDLER (Main Logic)
// ================================================================================

export default {
  // 📝 FORM DATA MANAGEMENT
  formData: {
    yourName: () => Input_YourName.text || "",
    partnerName: () => Input_Partner.text || "",
    city: () => Select_City.selectedOptionValue || "Mumbai",
    weddingDate: () => DatePicker_Wedding.selectedDate || new Date(),
    budget: () => Select_Budget.selectedOptionValue || "2-5 lakhs",
    guestCount: () => parseInt(Input_Guests.text) || 100,
    weddingType: () => Select_WeddingType.selectedOptionValue || "Hindu",
    duration: () => Select_Duration.selectedOptionValue || "Multi-day",
    events: () => CheckboxGroup_Events.selectedValues || ["Sangeet", "Wedding"],
    priorities: () => CheckboxGroup_Priorities.selectedValues || ["Venue", "Food"],
    specialRequirements: () => TextArea_Requirements.text || ""
  },

  // 🎯 CORE WEDDING SUBMISSION
  async submitWedding() {
    try {
      showAlert("Creating your wedding plan...", "info");

      // 1. Create main wedding record
      const weddingResponse = await CreateWedding.run();
      const weddingId = weddingResponse.Id;

      await storeValue('currentWeddingId', weddingId);
      await storeValue('currentWeddingName', this.generateWeddingName());

      // 2. Create couple record
      await CreateCouple.run();

      // 3. Create preferences
      await CreatePreferences.run();
```

```
    // 4. Auto-trigger AI vendor discovery
    await this.discoverVendorsWithAI();

    showAlert("Wedding plan created successfully! 🎉", "success");

    // 5. Refresh data
    await GetWeddings.run();

  } catch (error) {
    showAlert(`Error: ${error.message}`, "error");
  }
},

// 🤖 AI VENDOR DISCOVERY
async discoverVendorsWithAI() {
  try {
    showAlert("AI is discovering vendors for you...", "info");

    const categories = ["Photography", "Catering", "Decoration", "Music"];

    for (const category of categories) {
      await storeValue('currentSearchCategory', category);

      // Call your local CrewAI service
      const aiResult = await AIVendorDiscovery.run();

      // Save discovered vendors to NocoDB
      if (aiResult && aiResult.vendors) {
        await this.saveAIVendors(aiResult.vendors, category);
      }
    }

    await GetVendors.run(); // Refresh vendor list
    showAlert("AI found amazing vendors for you! 🤖", "success");

  } catch (error) {
    console.log("AI discovery failed, continuing without AI enhancement");
  }
},

// 💾 SAVE AI DISCOVERED VENDORS
async saveAIVendors(vendors, category) {
  for (const vendor of vendors) {
    await storeValue('aiVendorName', vendor.name);
    await storeValue('aiVendorCategory', category);
    await storeValue('aiVendorLocation', this.formData.city());
    await storeValue('aiVendorServices', vendor.services);
    await storeValue('aiVendorPhone', vendor.phone || '');
    await storeValue('aiVendorWebsite', vendor.website || '');

    await CreateVendor.run();
  }
},

// 📦 PAYLOAD GENERATORS
getWeddingPayload() {
  return {
    "Name": this.generateWeddingName(),
```

```
      "Date": this.formatDate(),
      "Budget Total": this.convertBudgetRange(),
      "Guest Count": this.formData.guestCount(),
      "Status": "Planning",
      "Wedding Type": this.formData.weddingType(),
      "Region": this.getCulturalRegion(),
      "City": this.formData.city(),
      "Timeline Status": "On Track",
      "Planning Phase": "Initial Planning",
      "Season": moment(this.formData.weddingDate()).format('MMMM'),
      "Days Remaining": moment(this.formData.weddingDate()).diff(moment(), 'days')
    };
  },

  getCouplePayload() {
    return {
      "Primary Contact Name": this.formData.yourName(),
      "Secondary Contact Name": this.formData.partnerName(),
      "City": this.formData.city(),
      "Wedding": appsmith.store.currentWeddingId,
      "Account Status": "Active",
      "Relationship Status": "Engaged",
      "Communication Preference": "WhatsApp"
    };
  },

  getPreferencesPayload() {
    return {
      "Wedding": appsmith.store.currentWeddingId,
      "Style Tags": this.formData.weddingType(),
      "Form-collected Preferences": `Events: ${this.formData.events().join(', ')};
Duration: ${this.formData.duration()}; Priorities: ${this.formData.priorities().join(',
')}`,
      "Cultural Requirements": `${this.formData.weddingType()} wedding traditions and
customs`,
      "Must Have Elements": this.formData.specialRequirements(),
      "Location Preferences": this.formData.city()
    };
  },

  getVendorPayload() {
    return {
      "Name": appsmith.store.aiVendorName,
      "Category": appsmith.store.aiVendorCategory,
      "Location": appsmith.store.aiVendorLocation,
      "Services Offered": appsmith.store.aiVendorServices,
      "Phone": appsmith.store.aiVendorPhone,
      "Website": appsmith.store.aiVendorWebsite,
      "AI-generated": true,
      "Match Score": 85,
      "Availability Status": "Available"
    };
  },

  getAISearchPayload() {
    return {
      "city": this.formData.city(),
      "category": appsmith.store.currentSearchCategory,
```

```
      "wedding_type": this.formData.weddingType(),
      "guest_count": this.formData.guestCount(),
      "budget": this.convertBudgetRange()
    };
  },

  // 🛠️ UTILITY FUNCTIONS
  generateWeddingName() {
    const names = [this.formData.yourName(), this.formData.partnerName()].filter(n => n);
    return names.length === 2 ? `${names[0]} & ${names[1]} Wedding` : `${names[0] ||
'Your'} Wedding`;
  },

  formatDate() {
    return moment(this.formData.weddingDate()).format('YYYY-MM-DD');
  },

  convertBudgetRange() {
    const budget = this.formData.budget();
    const ranges = {
      "Under 2 lakhs": 150000,
      "2-5 lakhs": 350000,
      "5-10 lakhs": 750000,
      "10-20 lakhs": 1500000,
      "Above 20 lakhs": 2500000
    };
    return ranges[budget] || 350000;
  },

  getCulturalRegion() {
    const type = this.formData.weddingType();
    const regions = {
      "Hindu": "North Indian",
      "Muslim": "Islamic",
      "Christian": "Western",
      "Sikh": "Punjabi",
      "Buddhist": "Traditional"
    };
    return regions[type] || "Mixed";
  },

  // 🎯 QUICK ACTIONS
  async loadSampleData() {
    await storeValue('sampleLoaded', true);
    showAlert("Sample wedding data loaded!", "success");
  },

  async exportWeddingPlan() {
    const plan = {
      wedding: appsmith.store.currentWeddingName,
      budget: this.convertBudgetRange(),
      vendors: Table_Vendors.tableData,
      timeline: this.formData.weddingDate()
    };

    download(JSON.stringify(plan, null, 2),
`${appsmith.store.currentWeddingName}_plan.json`, 'application/json');
    showAlert("Wedding plan exported!", "success");
```

```
  }
};

// ================================================================================
// 🎨 WIDGET CONFIGURATIONS
// ================================================================================

// Input_YourName
// Widget: Input
// Placeholder: "Enter your name"
// Required: true

// Input_Partner
// Widget: Input
// Placeholder: "Enter partner's name"

// Select_City
// Widget: Select
// Options: ["Mumbai", "Delhi", "Bangalore", "Chennai", "Kolkata", "Pune", "Hyderabad"]
// Default: "Mumbai"

// DatePicker_Wedding
// Widget: DatePicker
// Default Date: 6 months from today
// Min Date: Today

// Select_Budget
// Widget: Select
// Options: ["Under 2 lakhs", "2-5 lakhs", "5-10 lakhs", "10-20 lakhs", "Above 20 lakhs"]
// Default: "2-5 lakhs"

// Input_Guests
// Widget: Number Input
// Default: 100
// Min: 50, Max: 1000

// Select_WeddingType
// Widget: Select
// Options: ["Hindu", "Muslim", "Christian", "Sikh", "Buddhist", "Mixed"]
// Default: "Hindu"

// Select_Duration
// Widget: Select
// Options: ["Single Day", "Multi-day", "Week-long"]
// Default: "Multi-day"

// CheckboxGroup_Events
// Widget: Checkbox Group
// Options: ["Engagement", "Sangeet", "Haldi", "Wedding", "Reception"]
// Default: ["Sangeet", "Wedding"]

// CheckboxGroup_Priorities
// Widget: Checkbox Group
// Options: ["Venue", "Food", "Photography", "Decoration", "Music", "Transportation"]
// Default: ["Venue", "Food"]

// TextArea_Requirements
// Widget: Text Area
```

```
// Placeholder: "Any special requirements or preferences..."

// Button_SubmitWedding
// Widget: Button
// Text: "Create Wedding Plan 🚀"
// onClick: {{WeddingFormHandler.submitWedding()}}
// Style: Primary

// Button_AIDiscoverVendors
// Widget: Button
// Text: "🤖 AI Discover Vendors"
// onClick: {{WeddingFormHandler.discoverVendorsWithAI()}}
// Style: Secondary

// Table_Vendors
// Widget: Table
// Data: {{GetVendors.data}}
// Columns: Name, Category, Location, Phone, Services, Rating

// Table_Weddings
// Widget: Table
// Data: {{GetWeddings.data}}
// Columns: Name, Date, City, Status, Guest Count

// ================================================================================
// 🚀 STARTUP ACTIONS
// ================================================================================

// Page Load Actions:
// 1. GetWeddings.run()
// 2. GetVendors.run()

// ================================================================================
// 🎯 CREWAI LOCAL SERVICE (wedding_api.py)
// ================================================================================

/*
from fastapi import FastAPI
from crewai import Agent, Task, Crew, LLM
from crewai_tools import DuckDuckGoSearchTool
import uvicorn

app = FastAPI()

llm = LLM(
    model="ollama/qwen2.5:7b",
    base_url="http://localhost:11434",
    api_key="NA"
)

search_tool = DuckDuckGoSearchTool()

@app.post("/discover-vendors")
async def discover_vendors(city: str, category: str, wedding_type: str, guest_count: int,
budget: int):
    agent = Agent(
        role="Wedding Vendor Expert",
        goal=f"Find top 5 {category} vendors in {city}",
```

```
        backstory="Expert at finding quality wedding vendors",
        tools=[search_tool],
        llm=llm
    )

    task = Task(
        description=f"""
        Search for {category} vendors in {city} for {wedding_type} wedding:
        - Guest count: {guest_count}
        - Budget range: {budget}
        - Find contact details, services, pricing
        - Rate each vendor 1-10
        """,
        agent=agent,
        expected_output="JSON list of vendors with name, phone, services, website"
    )

    crew = Crew(agents=[agent], tasks=[task])
    result = crew.kickoff()

    return {"vendors": result}

if __name__ == "__main__":
    uvicorn.run(app, host="0.0.0.0", port=8000)
*/

// ================================================================================
// 🎯 DEPLOYMENT COMMANDS
// ================================================================================

/*
# Terminal 1: Start AI
ollama serve

# Terminal 2: Start CrewAI API
python wedding_api.py

# Terminal 3: Expose to cloud
ngrok http 8000

# Update AIVendorDiscovery API URL in Appsmith with ngrok URL
*/
```

```
// ================================================================================
// 🚀 COMPLETE WEDDING AI SETUP - CLONE READY
// ================================================================================
// 🎉 Author: Wedding AI Project
// 📅 Version: 1.0
// 🎯 Purpose: Full-stack wedding planning app with AI vendor discovery
// 🔧 Stack: Appsmith + NocoDB + CrewAI + Ollama + Local LLM

// ================================================================================
// 📋 PREREQUISITES & SYSTEM REQUIREMENTS
// ================================================================================

/*
SYSTEM REQUIREMENTS:
```

```
- macOS/Linux (Windows with WSL2)
- 8GB+ RAM (16GB recommended for LLM)
- 10GB+ free disk space
- Internet connection for initial setup
- Python 3.8+
- Node.js 16+ (for additional tools)

ACCOUNTS NEEDED:
- Appsmith Cloud Account (free): https://app.appsmith.com
- NocoDB Cloud Account (free): https://app.nocodb.com
- Hugging Face Account (optional): https://huggingface.co
- Serper.dev Account (optional): https://serper.dev

BROWSERS SUPPORTED:
- Chrome 90+
- Firefox 88+
- Safari 14+
- Edge 90+
*/

// ================================================================================
// 🛠️ COMPLETE INSTALLATION GUIDE
// ================================================================================

/*
# 📦 STEP 1: INSTALL CORE DEPENDENCIES

# Install Homebrew (macOS)
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# Install Python & pip
brew install python

# Install Ollama
curl -fsSL https://ollama.ai/install.sh | sh

# Install ngrok for cloud tunneling
brew install ngrok

# Verify installations
python3 --version
ollama --version
ngrok version

# 📦 STEP 2: SETUP PYTHON ENVIRONMENT

# Create virtual environment
python3 -m venv wedding_ai_env
source wedding_ai_env/bin/activate  # On macOS/Linux
# wedding_ai_env\Scripts\activate  # On Windows

# Install Python packages
pip install crewai[tools] fastapi uvicorn python-dotenv requests schedule

# Verify CrewAI installation
crewai --version
```

```
# 🎬 STEP 3: DOWNLOAD AI MODEL

# Start Ollama server
ollama serve &

# Download recommended model (7GB download)
ollama pull qwen2.5:7b

# Alternative smaller model (1.5GB)
ollama pull qwen2.5:1.5b

# Verify model installation
ollama list

# Test model
ollama run qwen2.5:7b "Hello, are you working?"

# 🎬 STEP 4: SETUP NGROK (for cloud connection)

# Sign up at https://ngrok.com and get auth token
ngrok authtoken YOUR_NGROK_TOKEN_HERE

# Test ngrok
ngrok http 8000

# 🎬 STEP 5: VERIFY NOCODB ACCESS

# Test your NocoDB connection
curl -H "xc-token: YCQ_vAuT1Zyvv6yClfXbZCmaSt_hVARWZN8bESYl" \
     https://app.nocodb.com/api/v2/tables/mg0w6xb658pvkqb/records
*/

// ================================================================================
// 📁 PROJECT DIRECTORY STRUCTURE
// ================================================================================

/*
wedding_ai_project/
├── README.md                          # This file
├── .env                               # Environment variables
├── requirements.txt                   # Python dependencies
├── wedding_api.py                     # CrewAI FastAPI service
├── test_apis.py                       # Testing script
├── appsmith_config.json               # Appsmith export
├── nocodb_schema.sql                  # Database schema backup
├── sample_data.json                   # Test data
└── deployment/
    ├── docker-compose.yml             # Container deployment
    ├── nginx.conf                     # Reverse proxy config
    └── startup.sh                     # Auto-startup script
*/

// ================================================================================
// 🔑 ENVIRONMENT CONFIGURATION (.env file)
// ================================================================================

/*
# Create .env file with these exact values:
```

```
# ===== NOCODB CONFIGURATION =====
NOCODB_API_BASE=https://app.nocodb.com/api/v2/tables
NOCODB_TOKEN=YCQ_vAuT1Zyvv6yClfXbZCmaSt_hVARWZN8bESYl

# Table IDs (DO NOT CHANGE)
WEDDINGS_TABLE_ID=mg0w6xb658pvkqb
COUPLES_TABLE_ID=m6v4on94fj53j9n
VENUES_TABLE_ID=m5n3t4l0edym7w4
VENDORS_TABLE_ID=ma3xsdfzx3fgxdf
TASKS_TABLE_ID=m4v24wea9g5wino
PREFERENCES_TABLE_ID=mhkpiytp9rqyd20
COMMUNICATIONS_TABLE_ID=m72hw41xhkjuul3
CEREMONIES_TABLE_ID=mo8npbdp35aumyq
BUDGET_TABLE_ID=m9ba2y6vl9t005j
AI_ACTIVITIES_TABLE_ID=m4prxg2yuh75gtw

# ===== LOCAL AI CONFIGURATION =====
OLLAMA_API_BASE=http://localhost:11434
OLLAMA_MODEL=qwen2.5:7b
CREWAI_API_PORT=8000

# ===== OPTIONAL API KEYS =====
# Get from https://huggingface.co/settings/tokens
HF_TOKEN=hf_your_token_here

# Get from https://serper.dev (2500 free searches)
SERPER_API_KEY=your_serper_key_here

# Get from https://ngrok.com
NGROK_AUTH_TOKEN=your_ngrok_token_here

# ===== APPSMITH CONFIGURATION =====
APPSMITH_APP_URL=https://app.appsmith.com/app/your-app-id
CREWAI_WEBHOOK_URL=https://your-ngrok-url.ngrok.io

# ===== LOGGING & DEBUG =====
LOG_LEVEL=INFO
DEBUG_MODE=true
ENABLE_TELEMETRY=false
*/

// ================================================================================
// 🎯 COMPLETE CREWAI SERVICE (wedding_api.py)
// ================================================================================

/*
#!/usr/bin/env python3
"""
Wedding AI Service - CrewAI + FastAPI
Complete vendor discovery and wedding planning AI service
"""

import os
import json
import logging
from typing import List, Dict, Optional
from datetime import datetime
```

```
from fastapi import FastAPI, HTTPException, BackgroundTasks
from fastapi.middleware.cors import CORSMiddleware
from pydantic import BaseModel
import uvicorn
import requests

from crewai import Agent, Task, Crew, LLM
from crewai_tools import DuckDuckGoSearchTool, ScrapeWebsiteTool
from dotenv import load_dotenv

# Load environment variables
load_dotenv()

# Configure logging
logging.basicConfig(level=os.getenv('LOG_LEVEL', 'INFO'))
logger = logging.getLogger(__name__)

# Initialize FastAPI
app = FastAPI(
    title="Wedding AI Service",
    description="AI-powered wedding planning with vendor discovery",
    version="1.0.0"
)

# Enable CORS for Appsmith
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Initialize LLM
try:
    llm = LLM(
        model=f"ollama/{os.getenv('OLLAMA_MODEL', 'qwen2.5:7b')}",
        base_url=os.getenv('OLLAMA_API_BASE', 'http://localhost:11434'),
        api_key="NA",
        temperature=0.7
    )
    logger.info(f"✅ LLM initialized: {os.getenv('OLLAMA_MODEL', 'qwen2.5:7b')}")
except Exception as e:
    logger.error(f"❌ LLM initialization failed: {e}")
    llm = None

# Initialize tools
search_tool = DuckDuckGoSearchTool()
scraper_tool = ScrapeWebsiteTool()

# ===== PYDANTIC MODELS =====
class VendorSearchRequest(BaseModel):
    city: str
    category: str
    wedding_type: str = "Hindu"
    guest_count: int = 100
    budget: int = 500000
```

```
    max_results: int = 5

class BudgetPlanRequest(BaseModel):
    total_budget: int
    priorities: List[str]
    guest_count: int
    events: List[str]

class CulturalAdviceRequest(BaseModel):
    wedding_type: str
    region: str
    specific_questions: Optional[str] = None

# ===== AI AGENTS =====
def create_vendor_agent():
    return Agent(
        role="Wedding Vendor Discovery Specialist",
        goal="Find the best wedding vendors matching specific requirements",
        backstory="""You are an expert wedding planner with 15+ years of experience.
        You excel at finding quality vendors, comparing services, and matching
        vendors to couple preferences. You have deep knowledge of wedding
        industry pricing, quality indicators, and regional specialties.""",
        tools=[search_tool, scraper_tool],
        llm=llm,
        verbose=True
    )

def create_budget_agent():
    return Agent(
        role="Wedding Budget Planning Expert",
        goal="Create optimal budget allocations for weddings",
        backstory="""You are a financial planning expert specializing in weddings.
        You understand cost structures, can optimize budget allocations,
        and provide realistic pricing guidance for different wedding elements.""",
        llm=llm,
        verbose=True
    )

def create_cultural_agent():
    return Agent(
        role="Cultural Wedding Traditions Expert",
        goal="Provide authentic cultural guidance for weddings",
        backstory="""You are a cultural expert with deep knowledge of wedding
        traditions across India and the world. You understand religious customs,
        regional variations, and can provide authentic advice for traditional
        wedding ceremonies.""",
        tools=[search_tool],
        llm=llm,
        verbose=True
    )

# ===== API ENDPOINTS =====

@app.get("/")
async def root():
    return {
        "service": "Wedding AI Service",
        "version": "1.0.0",
```

```
        "status": "running",
        "llm_status": "available" if llm else "unavailable",
        "endpoints": [
            "/discover-vendors",
            "/plan-budget",
            "/cultural-advice",
            "/health"
        ]
    }

@app.get("/health")
async def health_check():
    return {
        "status": "healthy",
        "timestamp": datetime.now().isoformat(),
        "llm_available": llm is not None,
        "tools_available": True
    }

@app.post("/discover-vendors")
async def discover_vendors(request: VendorSearchRequest):
    if not llm:
        raise HTTPException(status_code=503, detail="LLM not available")

    try:
        logger.info(f"🔍 Discovering {request.category} vendors in {request.city}")

        agent = create_vendor_agent()

        task = Task(
            description=f"""
            Search for top {request.max_results} {request.category} vendors in
{request.city}:

            Requirements:
            - Wedding type: {request.wedding_type}
            - Guest count: {request.guest_count}
            - Budget range: ₹{request.budget:,}

            For each vendor, find:
            1. Business name and contact person
            2. Phone number and email
            3. Website and social media
            4. Services offered and specialties
            5. Pricing range and packages
            6. Customer reviews and ratings
            7. Portfolio/previous work examples
            8. Availability and booking process

            Return as structured JSON with vendor details.
            Focus on quality, reliability, and value for money.
            """,
            agent=agent,
            expected_output="JSON array of vendor objects with complete details"
        )

        crew = Crew(
            agents=[agent],
```

```
            tasks=[task],
            verbose=True
        )

        result = crew.kickoff()

        # Save AI activity to NocoDB
        await save_ai_activity(
            activity_type="Vendor Discovery",
            description=f"Discovered {request.category} vendors in {request.city}",
            result=str(result)
        )

        return {
            "success": True,
            "category": request.category,
            "city": request.city,
            "vendors": result,
            "timestamp": datetime.now().isoformat()
        }

    except Exception as e:
        logger.error(f"❌ Vendor discovery failed: {e}")
        raise HTTPException(status_code=500, detail=str(e))

@app.post("/plan-budget")
async def plan_budget(request: BudgetPlanRequest):
    if not llm:
        raise HTTPException(status_code=503, detail="LLM not available")

    try:
        logger.info(f"💰 Planning budget of ₹{request.total_budget:,}")

        agent = create_budget_agent()

        task = Task(
            description=f"""
            Create detailed budget plan for wedding:

            Total Budget: ₹{request.total_budget:,}
            Guest Count: {request.guest_count}
            Priority Areas: {', '.join(request.priorities)}
            Events: {', '.join(request.events)}

            Provide:
            1. Percentage allocation for each category
            2. Specific amount recommendations
            3. Must-have vs nice-to-have breakdowns
            4. Cost-saving tips for each category
            5. Buffer allocation for unexpected expenses

            Categories to cover:
            - Venue & Catering (40-50%)
            - Photography & Videography (8-12%)
            - Decoration & Flowers (8-15%)
            - Clothing & Jewelry (10-15%)
            - Music & Entertainment (5-8%)
            - Transportation (3-5%)
```

```
            - Miscellaneous & Buffer (5-10%)

            Return as structured JSON with detailed breakdown.
            """,
            agent=agent,
            expected_output="JSON budget plan with categories, amounts, and
recommendations"
        )

        crew = Crew(
            agents=[agent],
            tasks=[task],
            verbose=True
        )

        result = crew.kickoff()

        return {
            "success": True,
            "total_budget": request.total_budget,
            "budget_plan": result,
            "timestamp": datetime.now().isoformat()
        }

    except Exception as e:
        logger.error(f"❌ Budget planning failed: {e}")
        raise HTTPException(status_code=500, detail=str(e))

@app.post("/cultural-advice")
async def cultural_advice(request: CulturalAdviceRequest):
    if not llm:
        raise HTTPException(status_code=503, detail="LLM not available")

    try:
        logger.info(f"🎎 Providing cultural advice for {request.wedding_type} wedding")

        agent = create_cultural_agent()

        task = Task(
            description=f"""
            Provide comprehensive cultural guidance for {request.wedding_type} wedding in
{request.region}:

            Specific Questions: {request.specific_questions or "General guidance needed"}

            Cover these aspects:
            1. Essential ceremonies and their significance
            2. Traditional attire recommendations
            3. Auspicious dates and timing considerations
            4. Required rituals and their order
            5. Traditional foods and catering suggestions
            6. Music and entertainment traditions
            7. Decoration themes and auspicious symbols
            8. Gift-giving customs and etiquette
            9. Photography considerations for religious ceremonies
            10. Regional variations and modern adaptations

            Provide practical, respectful, and authentic guidance.
```

```
            Include both traditional requirements and modern flexibility options.
            """,
            agent=agent,
            expected_output="Comprehensive cultural wedding guide with practical
recommendations"
        )

        crew = Crew(
            agents=[agent],
            tasks=[task],
            verbose=True
        )

        result = crew.kickoff()

        return {
            "success": True,
            "wedding_type": request.wedding_type,
            "region": request.region,
            "cultural_guidance": result,
            "timestamp": datetime.now().isoformat()
        }

    except Exception as e:
        logger.error(f"❌ Cultural advice failed: {e}")
        raise HTTPException(status_code=500, detail=str(e))

# ===== HELPER FUNCTIONS =====

async def save_ai_activity(activity_type: str, description: str, result: str):
    """Save AI activity to NocoDB for tracking"""
    try:
        headers = {
            "content-type": "application/json",
            "xc-token": os.getenv('NOCODB_TOKEN')
        }

        payload = {
            "Activity Type": activity_type,
            "Description": description,
            "AI Response": result[:1000],  # Truncate long responses
            "Status": "Completed",
            "Model Used": os.getenv('OLLAMA_MODEL', 'qwen2.5:7b'),
            "Timestamp": datetime.now().isoformat()
        }

        url =
f"{os.getenv('NOCODB_API_BASE')}/{os.getenv('AI_ACTIVITIES_TABLE_ID')}/records"
        requests.post(url, json=payload, headers=headers, timeout=10)

    except Exception as e:
        logger.warning(f"Failed to save AI activity: {e}")

if __name__ == "__main__":
    uvicorn.run(
        app,
        host="0.0.0.0",
        port=int(os.getenv('CREWAI_API_PORT', 8000)),
```

```
        reload=os.getenv('DEBUG_MODE', 'false').lower() == 'true'
    )
*/

// ================================================================================
// 🧪 TESTING SCRIPT (test_apis.py)
// ================================================================================

/*
#!/usr/bin/env python3
"""
Test script for Wedding AI APIs
Run this to verify everything is working
"""

import requests
import json
import time

# Test configurations
BASE_URL = "http://localhost:8000"
NOCODB_BASE = "https://app.nocodb.com/api/v2/tables"
NOCODB_TOKEN = "YCQ_vAuT1Zyvv6yClfXbZCmaSt_hVARWZN8bESYl"

def test_health():
    """Test service health"""
    try:
        response = requests.get(f"{BASE_URL}/health")
        print(f"✅ Health Check: {response.status_code}")
        print(json.dumps(response.json(), indent=2))
        return True
    except Exception as e:
        print(f"❌ Health Check Failed: {e}")
        return False

def test_vendor_discovery():
    """Test vendor discovery"""
    try:
        payload = {
            "city": "Mumbai",
            "category": "Photography",
            "wedding_type": "Hindu",
            "guest_count": 200,
            "budget": 500000,
            "max_results": 3
        }

        print("🔍 Testing vendor discovery...")
        response = requests.post(f"{BASE_URL}/discover-vendors", json=payload)
        print(f"Status: {response.status_code}")

        if response.status_code == 200:
            result = response.json()
            print("✅ Vendor Discovery Success!")
            print(f"Found vendors: {len(result.get('vendors', []))}")
        else:
            print(f"❌ Vendor Discovery Failed: {response.text}")
```

```
    except Exception as e:
        print(f"❌ Vendor Discovery Error: {e}")

def test_nocodb_connection():
    """Test NocoDB API connection"""
    try:
        headers = {
            "xc-token": NOCODB_TOKEN,
            "content-type": "application/json"
        }

        # Test getting weddings
        response = requests.get(
            f"{NOCODB_BASE}/mg0w6xb658pvkqb/records?limit=5",
            headers=headers
        )

        if response.status_code == 200:
            print("✅ NocoDB Connection: Working")
            data = response.json()
            print(f"Weddings in database: {len(data.get('list', []))}")
        else:
            print(f"❌ NocoDB Connection Failed: {response.status_code}")

    except Exception as e:
        print(f"❌ NocoDB Error: {e}")

def run_all_tests():
    """Run complete test suite"""
    print("🧪 Starting Wedding AI Test Suite")
    print("=" * 50)

    # Test 1: Health check
    if not test_health():
        print("❌ Service not running. Start with: python wedding_api.py")
        return

    time.sleep(1)

    # Test 2: NocoDB
    test_nocodb_connection()
    time.sleep(1)

    # Test 3: AI functionality
    test_vendor_discovery()

    print("\n🎉 Test Suite Complete!")

if __name__ == "__main__":
    run_all_tests()
*/

// ================================================================================
// 📦 PYTHON REQUIREMENTS (requirements.txt)
// ================================================================================

/*
# Core CrewAI and AI
```

```
crewai[tools]==0.119.0
langchain==0.1.20
langchain-community==0.0.38
transformers==4.40.2

# API and Web Service
fastapi==0.111.0
uvicorn[standard]==0.30.1
pydantic==2.7.4
requests==2.31.0

# Tools and Utilities
python-dotenv==1.0.1
schedule==1.2.0
beautifulsoup4==4.12.3
selenium==4.21.0

# Optional: For advanced features
pandas==2.2.2
numpy==1.26.4
pillow==10.3.0

# Development and Testing
pytest==8.2.2
pytest-asyncio==0.23.7
black==24.4.2
flake8==7.0.0
*/

// ============================================================================
// 🐳 DOCKER DEPLOYMENT (docker-compose.yml)
// ============================================================================

/*
version: '3.8'

services:
  wedding-ai:
    build: .
    ports:
      - "8000:8000"
    environment:
      - OLLAMA_API_BASE=http://ollama:11434
      - NOCODB_TOKEN=${NOCODB_TOKEN}
      - DEBUG_MODE=false
    depends_on:
      - ollama
    volumes:
      - ./logs:/app/logs
    restart: unless-stopped

  ollama:
    image: ollama/ollama:latest
    ports:
      - "11434:11434"
    volumes:
      - ollama_data:/root/.ollama
    restart: unless-stopped
```

```
  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf
      - ./ssl:/etc/nginx/ssl
    depends_on:
      - wedding-ai
    restart: unless-stopped

volumes:
  ollama_data:
*/

// ================================================================================
// 🚀 STARTUP SCRIPT (startup.sh)
// ================================================================================

/*
#!/bin/bash
# Wedding AI Startup Script

echo "🎊 Starting Wedding AI Service..."

# Check dependencies
echo "📦 Checking dependencies..."
command -v python3 >/dev/null 2>&1 || { echo "❌ Python3 required"; exit 1; }
command -v ollama >/dev/null 2>&1 || { echo "❌ Ollama required"; exit 1; }

# Activate virtual environment
if [ -d "wedding_ai_env" ]; then
    source wedding_ai_env/bin/activate
    echo "✅ Virtual environment activated"
else
    echo "🏗️ Creating virtual environment..."
    python3 -m venv wedding_ai_env
    source wedding_ai_env/bin/activate
    pip install -r requirements.txt
fi

# Start Ollama in background
echo "🤖 Starting Ollama server..."
ollama serve &
OLLAMA_PID=$!

# Wait for Ollama to start
sleep 5

# Pull model if not exists
if ! ollama list | grep -q "qwen2.5:7b"; then
    echo "📥 Downloading AI model (this may take a few minutes)..."
    ollama pull qwen2.5:7b
fi

# Start FastAPI service
```

```
echo "🚀 Starting Wedding AI API..."
python wedding_api.py &
API_PID=$!

# Start ngrok tunnel
if command -v ngrok >/dev/null 2>&1; then
    echo "🌐 Starting ngrok tunnel..."
    ngrok http 8000 --log=stdout > ngrok.log &
    NGROK_PID=$!

    # Wait for ngrok and extract URL
    sleep 3
    NGROK_URL=$(curl -s http://localhost:4040/api/tunnels | grep -o
'https://[^"]*\.ngrok\.io')
    echo "✅ Public URL: $NGROK_URL"
    echo "📝 Update this URL in your Appsmith API configuration"
fi

# Save PIDs for cleanup
echo $OLLAMA_PID > .ollama.pid
echo $API_PID > .api.pid
echo $NGROK_PID > .ngrok.pid

echo "🎉 Wedding AI Service is running!"
echo "📊 Health check: http://localhost:8000/health"
echo "📖 API docs: http://localhost:8000/docs"
echo "🛑 To stop: ./stop.sh"

# Keep script running
wait
*/

// ============================================================================
// 🛑 STOP SCRIPT (stop.sh)
// ============================================================================

/*
#!/bin/bash
# Stop Wedding AI Service

echo "🛑 Stopping Wedding AI Service..."

# Kill processes
if [ -f .ollama.pid ]; then
    kill $(cat .ollama.pid) 2>/dev/null
    rm .ollama.pid
    echo "✅ Ollama stopped"
fi

if [ -f .api.pid ]; then
    kill $(cat .api.pid) 2>/dev/null
    rm .api.pid
    echo "✅ API service stopped"
fi

if [ -f .ngrok.pid ]; then
    kill $(cat .ngrok.pid) 2>/dev/null
    rm .ngrok.pid
```

```
    echo "✅ Ngrok tunnel stopped"
fi

# Clean up any remaining processes
pkill -f "ollama serve" 2>/dev/null
pkill -f "wedding_api.py" 2>/dev/null
pkill -f "ngrok http" 2>/dev/null

echo "🏁 All services stopped"
*/

// ================================================================================
// 📊 SAMPLE TEST DATA (sample_data.json)
// ================================================================================

/*
{
  "sample_weddings": [
    {
      "name": "Rajesh & Priya Wedding",
      "city": "Mumbai",
      "budget": 500000,
      "guests": 200,
      "wedding_type": "Hindu",
      "events": ["Sangeet", "Haldi", "Wedding", "Reception"]
    },
    {
      "name": "Ahmed & Fatima Wedding",
      "city": "Delhi",
      "budget": 800000,
      "guests": 300,
      "wedding_type": "Muslim",
      "events": ["Mehendi", "Nikah", "Walima"]
    }
  ],

  "test_searches": [
    {
      "city": "Mumbai",
      "category": "Photography",
      "expected_vendors": 5
    },
    {
      "city": "Bangalore",
      "category": "Catering",
      "expected_vendors": 5
    }
  ],

  "api_test_cases": [
    {
      "endpoint": "/discover-vendors",
      "payload": {
        "city": "Mumbai",
        "category": "Photography",
        "wedding_type": "Hindu",
        "guest_count": 200,
        "budget": 500000
```

```
      }
    }
  ]
}
*/

// ================================================================================
// 🔧 TROUBLESHOOTING GUIDE
// ================================================================================

/*
COMMON ISSUES & SOLUTIONS:

1. ❌ "Command 'ollama' not found"
   Solution: Install Ollama: curl -fsSL https://ollama.ai/install.sh | sh

2. ❌ "Port 11434 already in use"
   Solution: Kill existing: pkill ollama && ollama serve

3. ❌ "Model not found"
   Solution: Download model: ollama pull qwen2.5:7b

4. ❌ "CrewAI import error"
   Solution: Install in venv: pip install crewai[tools]

5. ❌ "NocoDB 401 Unauthorized"
   Solution: Check token in .env file

6. ❌ "ngrok command not found"
   Solution: Install: brew install ngrok

7. ❌ "FastAPI won't start"
   Solution: Check port 8000: lsof -i :8000

8. ❌ "AI responses are slow"
   Solution: Use smaller model: ollama pull qwen2.5:1.5b

9. ❌ "Appsmith can't connect"
   Solution: Update API URL with ngrok HTTPS URL

10. ❌ "Memory issues"
    Solution: Increase Docker memory or use smaller model
*/

// ================================================================================
// 🎉 DEPLOYMENT CHECKLIST
// ================================================================================

/*
PRE-DEPLOYMENT CHECKLIST:

□ System Requirements Met
  □ 8GB+ RAM available
  □ 10GB+ disk space
  □ Internet connection stable
  □ Python 3.8+ installed
  □ All dependencies installed
```

```
□ Service Configuration
  □ .env file created with all tokens
  □ NocoDB APIs tested
  □ Ollama server running
  □ AI model downloaded
  □ FastAPI service responding

□ Appsmith Setup
  □ Wedding_API data source created
  □ All 10 NocoDB APIs configured
  □ AIVendorDiscovery API added
  □ Widget configurations applied
  □ WeddingFormHandler code added

□ Testing Complete
  □ Health check passes
  □ Vendor discovery working
  □ NocoDB integration working
  □ Appsmith form submission working
  □ AI responses reasonable

□ Production Ready
  □ Ngrok tunnel stable
  □ Error handling tested
  □ Logs configured
  □ Backup procedures in place
  □ Performance acceptable

DEPLOYMENT COMMANDS:
1. chmod +x startup.sh
2. ./startup.sh
3. Test: curl http://localhost:8000/health
4. Update Appsmith with ngrok URL
5. Test full workflow
*/

// ================================================================================
// 🎯 APPSMITH WIDGET CONFIGURATION DETAILS
// ================================================================================

// Detailed widget configurations for exact replication:

// Input_YourName Configuration:
{
  widgetType: "INPUT_WIDGET_V2",
  displayName: "Your Name",
  searchTags: ["form", "text input", "name"],
  topRow: 1,
  bottomRow: 2,
  leftColumn: 0,
  rightColumn: 6,
  widgetName: "Input_YourName",
  version: 2,
  tags: ["Inputs"],
  props: {
    inputType: "TEXT",
    placeholder: "Enter your full name",
    defaultText: "",
```

```
    regex: "",
    errorMessage: "",
    tooltip: "Enter the primary contact name",
    isRequired: true,
    isDisabled: false,
    resetOnSubmit: false,
    shouldAllowAutofill: true,
    maxChars: 50,
    label: "Your Name *",
    labelPosition: "Left",
    labelAlignment: "left",
    labelWidth: 5,
    validation: true,
    isSpellCheck: false
  }
}

// Select_City Configuration:
{
  widgetType: "SELECT_WIDGET",
  displayName: "City Selection",
  topRow: 3,
  bottomRow: 4,
  leftColumn: 0,
  rightColumn: 6,
  widgetName: "Select_City",
  options: [
    { label: "Mumbai", value: "Mumbai" },
    { label: "Delhi", value: "Delhi" },
    { label: "Bangalore", value: "Bangalore" },
    { label: "Chennai", value: "Chennai" },
    { label: "Kolkata", value: "Kolkata" },
    { label: "Pune", value: "Pune" },
    { label: "Hyderabad", value: "Hyderabad" },
    { label: "Ahmedabad", value: "Ahmedabad" },
    { label: "Jaipur", value: "Jaipur" },
    { label: "Lucknow", value: "Lucknow" }
  ],
  defaultOptionValue: "Mumbai",
  placeholderText: "Select your city",
  label: "Wedding City *",
  isRequired: true,
  isFilterable: true,
  serverSideFiltering: false
}

// CheckboxGroup_Events Configuration:
{
  widgetType: "CHECKBOX_GROUP_WIDGET",
  displayName: "Wedding Events",
  topRow: 8,
  bottomRow: 10,
  leftColumn: 0,
  rightColumn: 12,
  widgetName: "CheckboxGroup_Events",
  options: [
    { label: "Engagement Ceremony", value: "Engagement" },
    { label: "Sangeet Night", value: "Sangeet" },
```

```
    { label: "Haldi Ceremony", value: "Haldi" },
    { label: "Wedding Ceremony", value: "Wedding" },
    { label: "Reception Party", value: "Reception" },
    { label: "Mehendi Ceremony", value: "Mehendi" },
    { label: "Tilak Ceremony", value: "Tilak" }
  ],
  defaultSelectedValues: ["Sangeet", "Wedding"],
  label: "Select Wedding Events *",
  isRequired: true,
  isInline: false,
  alignment: "LEFT"
}

// Button_SubmitWedding Configuration:
{
  widgetType: "BUTTON_WIDGET",
  displayName: "Submit Wedding",
  topRow: 12,
  bottomRow: 13,
  leftColumn: 0,
  rightColumn: 4,
  widgetName: "Button_SubmitWedding",
  text: "🚀 Create Wedding Plan",
  buttonStyle: "PRIMARY",
  buttonVariant: "PRIMARY",
  placement: "CENTER",
  onClick: "{{WeddingFormHandler.submitWedding()}}",
  isDisabled: false,
  isVisible: true,
  recaptchaType: "V3",
  disabledWhenInvalid: true,
  resetFormOnClick: false,
  tooltip: "Create your complete wedding plan with AI assistance"
}

// Table_Vendors Configuration:
{
  widgetType: "TABLE_WIDGET_V2",
  displayName: "Vendors Table",
  topRow: 15,
  bottomRow: 25,
  leftColumn: 0,
  rightColumn: 24,
  widgetName: "Table_Vendors",
  tableData: "{{GetVendors.data}}",
  columnOrder: ["Name", "Category", "Location", "Phone", "Services Offered", "Rating", "AI-
generated"],
  primaryColumns: {
    Name: {
      index: 0,
      width: 150,
      id: "Name",
      originalId: "Name",
      alias: "Name",
      horizontalAlignment: "LEFT",
      verticalAlignment: "CENTER",
      columnType: "text",
      textColor: "",
```

```
      textSize: "0.875rem",
      enableFilter: true,
      enableSort: true,
      isVisible: true,
      isCellVisible: true,
      isDerived: false,
      label: "Vendor Name",
      computedValue: "{{Table_Vendors.processedTableData.map((currentRow, currentIndex) =>
( currentRow['Name']))}}"
    },
    Category: {
      index: 1,
      width: 120,
      id: "Category",
      originalId: "Category",
      alias: "Category",
      horizontalAlignment: "LEFT",
      verticalAlignment: "CENTER",
      columnType: "text",
      enableFilter: true,
      enableSort: true,
      isVisible: true,
      label: "Category"
    }
    // ... additional columns configuration
  },
  enableClientSideSearch: true,
  isVisibleSearch: true,
  isVisibleFilters: true,
  isVisibleDownload: true,
  isVisiblePagination: true,
  isSortable: true,
  delimiter: ",",
  defaultPageSize: 10
}

// [REST OF WIDGET CONFIGURATIONS CONTINUE...]

// ================================================================================
// 🎊 FINAL SUCCESS VERIFICATION
// ================================================================================

/*
SUCCESS INDICATORS:

✅ Ollama server running (http://localhost:11434)
✅ CrewAI API responding (http://localhost:8000/health)
✅ Ngrok tunnel active (https://xxx.ngrok.io)
✅ NocoDB APIs working (test with curl)
✅ Appsmith form submitting successfully
✅ AI vendor discovery finding results
✅ Data saving to all 10 NocoDB tables
✅ Vendor recommendations appearing in UI
✅ Budget calculations working
✅ Cultural advice generating properly

FINAL TEST:
1. Fill wedding form in Appsmith
```

```
2. Click "Create Wedding Plan"
3. See "AI is discovering vendors..." message
4. Check vendors appear in table
5. Verify data in NocoDB
6. Confirm AI activity logged

IF ALL GREEN: 🎉 Your Wedding AI is LIVE! 🎉
*/

// ============================================================================
// 📞 SUPPORT & RESOURCES
// ============================================================================

/*
HELPFUL RESOURCES:

📚 Documentation:
- CrewAI: https://docs.crewai.com
- Ollama: https://ollama.ai/docs
- Appsmith: https://docs.appsmith.com
- NocoDB: https://docs.nocodb.com

🔧 Tools:
- ngrok Dashboard: https://dashboard.ngrok.com
- Ollama Models: https://ollama.ai/library
- HuggingFace: https://huggingface.co/models

💬 Community:
- CrewAI Discord: https://discord.gg/X4JWnZnxPb
- Appsmith Discord: https://discord.gg/rBTTVJp
- Ollama GitHub: https://github.com/ollama/ollama

🐛 Debug Commands:
- Check Ollama: curl http://localhost:11434/api/tags
- Check API: curl http://localhost:8000/health
- Check ngrok: curl http://localhost:4040/api/tunnels
- View logs: tail -f wedding_api.log

Version: 1.0.0
Last Updated: 2024
Author: Wedding AI Team
License: MIT
*/
```