Daily Planner Agent - ReAct con OpenAI

Generado: 2025-10-21 18:10:41
Formato: ReAct Agent con LangChain y OpenAI

Contenido del Documento:

1. Plan Diario Generado

```
================================================================================
PLAN DIARIO GENERADO
================================================================================

Fecha: Tuesday, 21 de October de 2025
Objetivo: Estudiar microservicios

================================================================================

Plan diario para estudiar microservicios:
- 08:00 - 09:00: Estudio de Conceptos básicos de microservicios
- 09:00 - 10:00: Reunión con equipo AWS
- 10:00 - 11:00: Estudio de Arquitectura de microservicios
- 11:00 - 12:00: Estudio de Comunicación entre microservicios
- 12:00 - 13:00: Almuerzo
- 13:00 - 14:30: Estudio de Implementación de microservicios
- 14:30 - 15:30: Estudio de Pruebas y monitoreo de microservicios
- 15:30 - 17:00: Tiempo libre o revisión de proyectos activos
- 17:00 - 18:00: Bloque Gimnasio
- 18:00 - 19:00: Revisión de notas y preparación para el día siguiente

================================================================================
```

2. Código Python: daily_planner_agent.py

Implementación del Agente ReAct para planificación diaria

```
#!/usr/bin/env python3
"""Daily Planner Agent - ReAct Pattern con OpenAI"""

import json
import sys
import os
from datetime import datetime
from langchain.agents import AgentExecutor, create_react_agent
from langchain.tools import Tool
from langchain_openai import ChatOpenAI
from langchain_core.prompts import PromptTemplate


def get_calendar_events(_input: str = None) -> str:
    """Carga eventos del calendario desde archivo.
    """
    with open("calendar_data.json") as f:
        data = json.load(f)
    return json.dumps(data, ensure_ascii=False)


def list_projects(_input: str = None) -> str:
    """Carga proyectos activos desde archivo.
    """
    with open("projects_data.json") as f:
        data = json.load(f)
    return json.dumps(data, ensure_ascii=False)


def break_down_goal(goal: str) -> str:
    """Desglosa una meta en subtareas"""
    subtasks = []

    if "tesis" in goal.lower():
        subtasks.extend([
            {"task": "Implementar backoff exponencial", "minutes": 90, "priority": "high"},
            {"task": "Tests unitarios", "minutes": 60, "priority": "high"},
            {"task": "Documentar", "minutes": 30, "priority": "medium"}
        ])

    if "azure" in goal.lower() or "load balancer" in goal.lower():
        subtasks.extend([
            {"task": "Estudiar Azure Load Balancer", "minutes": 60, "priority": "high"},
            {"task": "Ejercicios prácticos", "minutes": 60, "priority": "high"}
        ])

    total_minutes = sum(t["minutes"] for t in subtasks) if subtasks else 0

    return json.dumps({
        "goal": goal,
        "subtasks": subtasks,
        "total_minutes": total_minutes,
        "total_hours": round(total_minutes / 60, 1) if total_minutes > 0 else 0
    }, ensure_ascii=False)
def create_agent():
    """Crea el agente ReAct con OpenAI"""
```

```
    tools = [
        Tool(
            name="GetCalendarEvents",
            func=get_calendar_events,
            description="Obtiene los eventos confirmados del calendario para el día (reuniones, bloques personales)"
        ),
        Tool(
            name="ListProjects",
            func=list_projects,
            description="Lista los proyectos técnicos activos con su estado y prioridad"
        ),
        Tool(
            name="BreakDownGoal",
            func=break_down_goal,
            description="Desglosa una meta en subtareas concretas con estimaciones de tiempo"
        )
    ]

    template = """Responde usando este formato exacto:
Thought: [tu razonamiento]
Action: [nombre de herramienta]
Action Input: [parámetros]
Observation: [resultado]
(repite Thought/Action/Observation si necesitas más info)
Thought: [razonamiento final]
Final Answer: [tu respuesta]

Herramientas disponibles:
{tools}

Nombres: {tool_names}

Tarea: {input}

{agent_scratchpad}"""

    prompt = PromptTemplate.from_template(template)

    # OpenAI API - Volver a gpt-4o-mini que entiende mejor ReAct
    llm = ChatOpenAI(
        openai_api_key=os.getenv("OPENAI_API_KEY"),
        model="gpt-4o-mini",
        temperature=0.2,
        max_tokens=1500
    )

    agent = create_react_agent(llm=llm, tools=tools, prompt=prompt)

    return AgentExecutor(
        agent=agent,
        tools=tools,
        verbose=False,
        max_iterations=6,
        handle_parsing_errors=True
    )


def plan_day(user_goal: str) -> str:
    """Genera plan diario usando el agente ReAct con OpenAI"""
    executor = create_agent()
```

```
    prompt = f"""Genera un plan diario para: {user_goal}

Usa las herramientas:
1. GetCalendarEvents - para eventos confirmados
2. ListProjects - para proyectos activos
3. BreakDownGoal - para desglosar el objetivo

Retorna: Plan con horarios (8am-7pm), respetando calendario, balanceando trabajo y descansos."""

    # Mostrar prompt final
    print("\n" + "="*70)
    print("■ PROMPT ENVIADO A LANGCHAIN:")
    print("="*70 + "\n")
    print(prompt)
    print("\n" + "="*70 + "\n")

    try:
        response = executor.invoke({"input": prompt})
        return response.get("output", "Error generando plan")
    except Exception as e:
        return f"■ Error: {str(e)}\n\n■ Verifica:\n- OPENAI_API_KEY está configurada\n- Tienes créditos en OpenAI\n


def main():
    """Entrada principal"""
    print("\n" + "="*70)
    print("■ DAILY PLANNER AGENT - ReAct con OpenAI")
    print("="*70 + "\n")

    # Mostrar ejemplos
    print("■ Ejemplos de objetivos:")
    print("  • Avanzar mi tesis y estudiar Azure Load Balancer")
    print("  • Debuggear cliente NATS y revisar seguridad en Azure")
    print("  • Estudiar microservicios y implementar tests unitarios")
    print("  • Trabajar en proyecto NATS y revisar documentación\n")

    # Tomar input del usuario
    user_goal = input("■ Ingresa tu objetivo del día: ").strip()

    if not user_goal:
        print("■ Error: Debes ingresar un objetivo")
        sys.exit(1)

    print("\n■ Generando plan con OpenAI...\n")

    # Generar plan
    plan = plan_day(user_goal)

    # Guardar en archivo
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    filename = f"plan_{timestamp}.txt"

    with open(filename, "w", encoding="utf-8") as f:
        f.write("="*70 + "\n")
        f.write("PLAN DIARIO GENERADO\n")
        f.write("="*70 + "\n\n")
        f.write(f"Fecha: {datetime.now().strftime('%A, %d de %B de %Y')}\n")
        f.write(f"Objetivo: {user_goal}\n\n")
        f.write("="*70 + "\n\n")
        f.write(plan)
```

```
        f.write("\n\n" + "="*70 + "\n")

    print(f"■ Plan guardado en: {filename}\n")
    print(f"Contenido:\n\n{plan}\n")


if __name__ == "__main__":
    main()
```

3. Datos: calendar_data.json

Eventos del calendario (reuniones, bloques personales)

```
{
  "events": [
    {
      "time": "09:00-10:00",
      "title": "Reunión con equipo AWS",
      "type": "meeting",
      "priority": "high"
    },
    {
      "time": "17:00-18:00",
      "title": "Bloque Gimnasio",
      "type": "personal",
      "priority": "medium"
    }
  ]
}
```

4. Datos: projects_data.json

Proyectos técnicos activos con estado y prioridad


```
{
  "projects": [
    {
      "name": "Tesis: Smart Retries Architecture",
      "status": "in-progress",
      "priority": "high",
      "progress": "40%"
    },
    {
      "name": "Azure Fundamentals (AZ-104)",
      "status": "in-progress",
      "priority": "high",
      "progress": "35%"
    },
    {
      "name": "NATS Message Queue SideProject",
      "status": "in-progress",
      "priority": "medium",
      "progress": "25%"
    }
  ]
}
```

5. Dependencias: requirements_planner.txt

Librerías Python necesarias

```
langchain==0.1.11
langchain-core==0.1.33
langchain-openai==0.1.3
python-dotenv==1.0.0
```

Información Técnica

Patrón: ReAct (Reasoning + Acting + Observation)

Framework: LangChain 0.1.11

LLM: OpenAI GPT-4o-mini

Herramientas: GetCalendarEvents, ListProjects, BreakDownGoal

Líneas de código: 182

Fecha de generación: 2025-10-21 18:10:41