Universidad Rey Juan Carlos

TEMA 6 – HOJA DE EJERCICIOS III

Se plantean diferentes ejercicios basados en agentes LLM + Tools + Skills (framework práctico: LangGraph).

Objetivo: diseñar flujos agénticos controlados, integrar herramientas externas y modularizar capacidades mediante skills.

1) Bloque teórico (sin código)

Ejercicio T1 — Tool vs Skill vs Agent (10–12 líneas)

Define con tus palabras (sin copiar definiciones) las diferencias entre tool, skill y agente LLM. Incluye un ejemplo de cada uno en el contexto de un asistente universitario (normativa, tutorías, evaluación, etc.).

Ejercicio T2 — Diseño de control: evitar loops y alucinaciones (12–15 líneas)

Explica por qué los agentes pueden entrar en bucles o tomar acciones incorrectas. Propón 4 mecanismos concretos de control (p. ej., límite de pasos, validación de tool output, política de abstención, checks de groundedness) y di dónde se aplicarían en el pipeline.

Ejercicio T3 — RAG como tool dentro de un agente (8–10 líneas)

Describe cómo integrarías un sistema RAG como tool dentro de un agente. ¿Qué decide el agente? ¿Qué devuelve la tool? ¿Qué riesgos aparecen (p. ej., retrieval irrelevante) y cómo los mitigarías?

2) Bloque práctico con LangGraph

Se recomienda crear un repositorio con scripts o notebooks por ejercicio, y un README con instrucciones. En cada ejercicio, incluye un ejemplo de ejecución y una breve discusión (5–10 líneas) de lo observado.

Ejercicio P1 — Flujo sencillo (grafo lineal con estado)

Implementa un flujo en LangGraph con 3 nodos y estado compartido:
1) Nodo "preprocess": normaliza la pregunta (p. ej., minúsculas, recorta espacios, detecta idioma simple).

2) Nodo "llm_answer": el LLM responde con un formato fijo (título + 3 bullets).

3) Nodo "postprocess": añade una sección final "Calidad" con dos checks: "¿hay respuesta?" y "¿es clara?".

Requisitos:

- Define un estado TypedDict con campos: question (str), normalized_question (str), answer (str).
- El grafo debe compilar y ejecutarse con app.invoke({...}).
- Añade logging simple para ver el paso por nodos.

Pista: este ejercicio busca que entiendas el concepto de state y cómo LangGraph estructura un pipeline, sin herramientas todavía.

Ejercicio P2 — Agente con tools (router + tool nodes)

Construye un agente con decisión de herramienta (router) en LangGraph. El agente debe elegir entre:

- Tool 1: "calc" (calculadora segura para operaciones básicas)
- Tool 2: "search_policy" (mock de consulta a normativa: un diccionario en memoria)
- Tool 3: "final" (responder sin tool si no hace falta)

Especificación:

1) Nodo "router": el LLM clasifica la intención en una etiqueta {calc | policy | final}.
2) Nodo "calc_tool": ejecuta el cálculo y añade el resultado al estado.
3) Nodo "policy_tool": busca en un diccionario (p. ej., claves: "evaluación continua", "convocatoria extraordinaria") y devuelve un texto breve.
4) Nodo "final_answer": el LLM genera la respuesta final usando los resultados de la tool.

Requisitos técnicos:

- Usa aristas condicionales desde "router" en función de la etiqueta.
- Añade un campo "tool_result" al estado.
- Implementa un contador de pasos o una protección contra loops (max_steps).
- Muestra por pantalla qué tool se eligió en cada consulta.

Pista: la herramienta "policy_tool" es intencionalmente simple; la mejora vendrá cuando sustituyamos este mock por un RAG real en ejercicios posteriores.

Ejercicio P3 — Agente con tool RAG (conocimiento externo)

Integra una tool tipo RAG como nodo de LangGraph (puedes reutilizar tu vector store de ejercicios de RAG). El agente debe:

1) Detectar si la pregunta requiere conocimiento documental.
2) Si sí, llamar a "rag_tool" (retrieval) y luego responder con groundedness.
3) Si no, responder directamente.

Requisitos:

- El estado debe incluir: question, decision, retrieved_context, sources, answer.
- La respuesta debe terminar con "Fuentes:" listando metadatos.
- Debe existir una política de abstención: si no hay contexto suficiente, decir "No puedo responder con la información disponible."

Ejercicio P4 — Skills + tools (modularización avanzada)

Define dos skills como sub-grafos o nodos compuestos:

Skill A: "Explainer" → genera una explicación pedagógica (definición + ejemplo + advertencia).

Skill B: "Checklist" → genera una lista de verificación (pasos) para ejecutar una tarea.

El agente principal debe:

- Enrutar a Skill A si la intención es "explicar un concepto".
- Enrutar a Skill B si la intención es "procedimiento" (p. ej., "cómo entregar prácticas").
- Usar tools cuando sea necesario (RAG para normativa, calc si hay números).

Requisitos:

- Implementa un nodo "intent_router" que devuelva {explain | procedure | other}.
- Demuestra al menos 3 conversaciones donde el agente active skills distintas.
- Incluye un control de coste/pasos: max_steps y un mensaje de salida si se supera.

Objetivo didáctico: entender que las skills son macro-capacidades reutilizables y que LangGraph permite componer comportamientos complejos de forma controlada.

Anexo — Esqueleto mínimo de LangGraph (referencia)

```
from typing import TypedDict
from langgraph.graph import StateGraph, END

class State(TypedDict):
    question: str
    answer: str

def node_a(state: State) -> State:
    # ...
    return state

graph = StateGraph(State)
graph.add_node("a", node_a)
graph.set_entry_point("a")
graph.add_edge("a", END)

app = graph.compile()
print(app.invoke({"question": "Hola", "answer": ""}))
```