PRÁCTICA Agent Tools & U-QAM Score

Esta actividad evaluable consta de dos partes:

- En la primera repasarás los contenidos sobre Agentes Inteligentes y reutilizarás el código con el que se documentaron los LABs. Trabajarás sobre el **agente de cambio de moneda** desarrollado. Esta parte de la actividad se centrará en dos aspectos concretos:
    - Uso de **Agent Tools**
    - **Observabilidad**, mediante plugins de ADK
- En la segunda parte implementarás un agente inteligente de arquitectura paralela al anterior. El objetivo de aprendizaje está fijado en el desarrollo conceptual e implementación PoC de un agente con utilidad práctica, i.e. un **evaluador de calidad de textos escritos**.
    - Igualmente trabajarás los dos aspectos de agentes anteriores: tools & observabilidad
    - Sobre la base de unas métricas dadas, tendrás que diseñar un indicador de calidad de textos, implementándolo como *skill* de un Agente Inteligente

PARTE 1 Currency Exchange Agent

En esta parte vas a utilizar el código base del **Agente** de cambio de moneda (**currency exchange,** script **agent.py** que se adjunta), al que añadirás la funcionalidad de observabilidad en ejecución. Vas a desarrollar las siguientes tareas (*presta atención a la numeración, que es referencia en la tabla del apartado de entregables*):

1. Configurar el modelo LLM que utilizará el agente, i.e. *gemini-2.5-flash-lite* (recomendado)
2. Contrastar el log de ejecución con el de referencia **agent_OUTPUT.md** que se proporciona con la actividad

Basic Agent: Currency Exchange (Basic C.E.)

Trabajarás con el agente del LAB 3, que puedes localizar en Moodle:

Extracto del código del fichero LAB_3B_Agent_Tools.ipynb donde se define el agente.

```
currency_agent = LlmAgent(
    name="currency_agent",
    model=Gemini(model="gemini-2.5-flash-lite", retry_options=retry_config),
    instruction="""You are a smart currency conversion assistant.

    For currency conversion requests:
    1. Use `get_fee_for_payment_method()` to find transaction fees
    2. Use `get_exchange_rate()` to get currency conversion rates
    3. Check the "status" field in each tool's response for errors
    4. Calculate the final amount after fees based on the output from
`get_fee_for_payment_method` and `get_exchange_rate` methods and provide a clear breakdown.
    5. First, state the final converted amount.
        Then, explain how you got that result by showing the intermediate amounts. Your
explanation must include: the fee percentage and its
        value in the original currency, the amount remaining after the fee, and the exchange
rate used for the final conversion.

    If any tool returns status "error", explain the issue to the user clearly.
    """,
    tools=[get_fee_for_payment_method, get_exchange_rate],
)
```

Los pasos a seguir son:

3. Implementar en el código el plugin LoggingPlugin parte de la librería ADK. Nombra este nuevo script como **agent_obs.py**

   from google.adk.plugins.logging_plugin import LoggingPlugin

4. Ejecutar de nuevo el agente y revisar el log de ejecución
5. Implementar el plugin que incluye contadores de llamada a agente, modelo y tools CountInvocationPlugin (incluido en fichero **count_invocation_plugin.py**)

   from count_invocation_plugin import CountInvocationPlugin

6. Modifica el script **agent_obs.py** para que el log mostrado por consola sea el proporcionado por éste

Enhanced Agent: Currency Exchange (Enhanced C.E.)

Extracto del código del fichero LAB_3B_Agent_Tools.ipynb donde se define el agente.

```
enhanced_currency_agent = LlmAgent(
    name="enhanced_currency_agent",
    model=Gemini(model="gemini-2.5-flash-lite", retry_options=retry_config),
    # Updated instruction
    instruction="""You are a smart currency conversion assistant. You must strictly follow these
steps and use the available tools.

  For any currency conversion request:

   1. Get Transaction Fee: Use the get_fee_for_payment_method() tool to determine the transaction
fee.
   2. Get Exchange Rate: Use the get_exchange_rate() tool to get the currency conversion rate.
   3. Error Check: After each tool call, you must check the "status" field in the response. If the
status is "error", you must stop and clearly explain the issue to the user.
   4. Calculate Final Amount (CRITICAL): You are strictly prohibited from performing any arithmetic
calculations yourself. You must use the calculation_agent tool to generate Python code that
calculates the final converted amount. This
      code will use the fee information from step 1 and the exchange rate from step 2.
   5. Provide Detailed Breakdown: In your summary, you must:
       * State the final converted amount.
       * Explain how the result was calculated, including:
           * The fee percentage and the fee amount in the original currency.
           * The amount remaining after deducting the fee.
           * The exchange rate applied.
    """,
    tools=[
        get_fee_for_payment_method,
        get_exchange_rate,
        AgentTool(agent=calculation_agent),  # Using another agent as a tool!
    ],
)
```

(*continúa la lista numerada*)

7. Replicar las tareas 5 y 6 para esta versión mejorada del agente, que incluye generación de código *on-the-fly*

Entregables PARTE 1

Se entregará una tabla con el siguiente formato, así como los ficheros resaltados en VERDE:

Id	Agente	TAREA	SCRIPT	OUTPUT	COMENTARIOS
1,2	Basic C.E.	Comparar log	**agent.py**	agent_OUTPUT_.md	*¿Hay diferencias con la ejecución de referencia?*
3,4	Basic C.E.	Implementación *LoggingPlugin*	**agent_obs.py**	agent_obs_OUTPUT.md	*Tus comentarios*
5,6	Basic C.E.	Implementación *CountInvocationPlugin*	**agent_obs.py**	agent_obs_OUTPUT.md	*Tus comentarios*
7	Enhanced C.E.	Implementación *CountInvocationPlugin*	**agent_obs_v2.py**	agent_obs_v2_OUTPUT.md	*Tus comentarios*

NOTA: Para facilidad de documentación y claridad del código se recomienda utilizar ***git***. A modo de ejemplo:

- **Fila de tareas 3,4.** Fichero: **agent_obs.py** Commit: Implementación de *LoggingPlugin*
- **Fila de tareas 5,6.** Fichero: **agent_obs.py** Commit: Implementación de *CountInvocationPlugin*
- **Fila de tarea 7.** Fichero: **agent_obs_v2.py** Commit: Enhanced agent. Plugin *CountInvocationPlugin*
    - De esta forma las versiones del código quedan documentadas en el histórico de ***git*** y no necesitarás generar copias del fichero para versionarlo

Si no utilizas GIT tendrás que entregar una versión de cada fichero:

- Tareas 3, 4: **agent_obs.py** con implementación de *LoggingPlugin*
    - Salida de consola en **agent_obs_OUTPUT.md**
- Tareas 5, 6: **agent_obs.py** con implementación de *CountInvocationPlugin*
    - Salida de consola en **agent_obs_OUTPUT.md**
- Tarea 7: **agent_obs_v2.py** con implementación de *CountInvocationPlugin*
    - Salida de consola en **agent_obs_v2_OUTPUT.md**

PARTE 2. Agente de Calidad de Redacción

Reutilizando el esqueleto y base código de la PARTE 1, crear un agente que, recibiendo un texto de entre 800 y 1000 palabras, reporte un indicador de calidad de redacción que denominaremos **U-QAM.** Los textos elegidos como ejemplo deben proceder de la Wikipedia o fuente similar alternativa.

NOTA: *En el documento adjunto **U-QAM Score.pdf** tienes la definición del indicador y las métricas con las que se construye*

Las tareas a realizar son las siguientes:

1. Estudiar y entender las métricas y el código suministrado para su cálculo. Junto con la definición de cada métrica se aportan pautas de interpretabilidad útiles para la construcción del indicador U-QAM
2. Proponer 2 versiones alternativas de U-QAM. Seleccionar una de ellas justificando la decisión
3. Definir la arquitectura del sistema agéntico
4. Adaptar un plugin existente (de la PARTE 1) para hacer el *debug* del agente, poniendo el foco en la observabilidad del mismo durante la ejecución
5. Probar el código con 3 muestras de texto, de naturaleza diferente (**técnico**, literario, histórico)

El texto de tipo **técnico** se tomará como línea base para caracterizar los estilos de redacción de los otros tipos de texto.

Entregable PARTE 2

La extensión máxima de la PARTE 2 será de 1000 palabras, excluyendo el código, que se adjuntará por separado.

La parte escrita del entregable incluirá:

- Definición y Justificación del indicador U-QAM diseñado
- Indica brevemente las dificultades que has encontrado a la hora de depurar el código del agente frente a lo que sería un desarrollo tradicional de software
- Indica situaciones acontecidas durante el *debugging* que pueden atribuirse directamente al componente estotástico que introduce el LLM en la fiabilidad de ejecución
- Lecciones aprendidas

El entregable de código incluirá:

- El script del agente
- Los 3 textos de ejemplo elegidos
- La salida por consola de la ejecución del agente para cada uno (fichero *markdown*)

NO SE REQUIERE:

- Realizar ningún procesamiento de los textos adicional al que del código suministrado
- *Optimizar* el indicador U-QAM, dado que el objetivo de la actividad es el diseño y desarrollo del agente en sí mismo

EVALUACIÓN

El reparto de nota entre las partes de la actividad es el siguiente:

- PARTE 1 *Currency Exchange Agent* **30%**
- PARTE 2 *Agente de Calidad de Redacción U-QAM* **70%**

Evaluación PARTE 1

El objetivo de esta parte es el repaso y consolidación de los conocimientos adquiridos en los LABs. Se valorará:

- Concisión y claridad de las explicaciones
- Brevedad y funcionalidad del código. No se requieren *features* adicionales a las indicadas en la sección **Entregable PARTE 1**
- *Outputs* de las ejecuciones en bruto, tal como salen por consola (copiar y pegar en los ficheros markdown entregables)

Evaluación PARTE 2

El objetivo de esta parte es el diseño conceptual de un agente y la validación mediante una prueba rápida (**PoC**) en un entorno realista (textos elegidos). Se valorará:

- Definición y justificación del **indicador U-QAM** elegido para medir la calidad de redacción de un texto. Se valorará la simplicidad y utilidad de la propuesta
- Aplicación paralela de la arquitectura de código de la PARTE 1
- Concisión y claridad de las explicaciones
- Brevedad y funcionalidad del código