Estado del Proyecto: PizzeríaBot (24 de Mayo, 2025)

1. Descripción General del Proyecto

**PizzeríaBot** es un sistema de chatbot inteligente diseñado para automatizar y mejorar la experiencia de toma de pedidos para una pizzería. El proyecto se está desarrollando en Python utilizando el **Agent Development Kit (ADK) de Google**, con el objetivo de integrarse eventualmente con plataformas de mensajería como WhatsApp. La gestión de datos (menú, clientes, pedidos) se realiza a través de **Google Sheets**. El tono de comunicación del bot se define como amigable, proactivo, directo y formal.

El sistema se basa en una arquitectura multi-agente, donde cada agente se especializa en una parte del flujo conversacional:

* Gestión de clientes (registro, identificación).
* Toma de pedidos (navegación de menú, adición/modificación de ítems).
* Confirmación de detalles de entrega.
* (Futuro) Confirmación de pago y procesamiento final del pedido.
* Un agente raíz (RootAgent) orquesta el flujo entre los agentes especializados.

2. Logros y Avances Hasta la Fecha

Hemos alcanzado hitos significativos en el desarrollo de los componentes centrales del sistema:

2.1. Agentes Implementados y Funcionales:

* **CustomerManagementAgent\_v1**:
  + **Estado:** ✅ Altamente Funcional.
  + **Capacidades:**
    - Saluda a los usuarios nuevos y existentes.
    - Verifica la existencia de clientes usando la herramienta get\_customer\_data.
    - Registra nuevos clientes (obteniendo Nombre\_Completo) y actualiza datos de clientes existentes mediante register\_update\_customer.
    - Personaliza el saludo para clientes recurrentes.
    - Maneja el \_session\_user\_id de forma robusta a través de un callback.
* **OrderTakingAgent\_v1**:
  + **Estado:** 🟢 Muy Avanzado, con un punto crítico identificado recientemente.
  + **Capacidades Verificadas:**
    - Inicia la conversación ofreciendo el menú o la toma directa del pedido.
    - Utiliza manage\_order\_item para añadir ítems.
    - Maneja la ambigüedad de ítems (ej. diferentes tamaños de pizza) presentando opciones al usuario (obtenidas de get\_menu\_item\_details).
    - Responde a consultas sobre detalles de platos (get\_menu\_item\_details) y promociones (get\_active\_promotions).
    - Permite modificar la cantidad de un ítem (manage\_order\_item con action='update\_quantity').
    - Permite quitar un ítem del pedido (manage\_order\_item con action='remove').
    - Muestra el resumen del pedido actual (view\_current\_order).
    - Finaliza su parte de la selección de ítems mostrando un resumen y un mensaje de transición claro para el siguiente agente.
* **DeliveryConfirmationAgent\_v1**:
  + **Estado:** 🟢 Funcionalidad Base Implementada, con ajustes recientes en su lógica de manejo de direcciones.
  + **Capacidades Verificadas:**
    - Inicia su flujo llamando a view\_current\_order para obtener el pedido del agente anterior.
    - Presenta el resumen del pedido al cliente y solicita confirmación explícita del contenido.
    - Si el cliente no confirma el contenido, está instruido para devolver el control.
    - Si el contenido es confirmado, procede a gestionar la dirección de envío:
      * Llama a get\_saved\_addresses para buscar direcciones existentes.
      * Si no hay direcciones, solicita una nueva al cliente.
      * Confirma verbalmente la dirección proporcionada.
      * Intenta guardar/actualizar la dirección usando register\_update\_customer. (Manejo de error mejorado).
      * Llama a calculate\_delivery\_cost con la dirección confirmada.
      * Informa el costo y tiempo estimado de envío.
      * Finaliza su interacción con un mensaje de transición indicando que se realizará una "última verificación" antes de pasar a cocina.
* **RootAgent\_v1 (Orquestador Principal):**
  + **Estado:** 🟢 Lógica de Orquestación Básica y Verificación de Datos Implementada.
  + **Capacidades Verificadas:**
    - Transfiere correctamente el control al CustomerManagementAgent\_v1 al inicio de una nueva conversación.
    - Transfiere correctamente el control al OrderTakingAgent\_v1 después de que CustomerManagementAgent\_v1 completa el registro/identificación.
    - Transfiere correctamente el control al DeliveryConfirmationAgent\_v1 después de que OrderTakingAgent\_v1 finaliza la selección de ítems.
    - Llama a la herramienta get\_customer\_data (versión mejorada) después de que DeliveryConfirmationAgent\_v1 completa su parte.
    - Reacciona correctamente a los diferentes status devueltos por get\_customer\_data:
      * 'complete': Procede con un mensaje indicando que el pedido se procesará para cocina.
      * 'not\_found': Transfiere al CustomerManagementAgent\_v1 para un registro completo.
      * 'incomplete' (específicamente si falta Direccion\_Principal pero el nombre está): Transfiere al DeliveryConfirmationAgent\_v1 para que este solicite la dirección.

2.2. Herramientas (pizzeria\_tools.py):

* **get\_customer\_data**: ✅ Mejorada para devolver status, data y missing\_fields, funcionando correctamente en los escenarios probados.
* **register\_update\_customer**: ✅ Funcional para registrar nuevos clientes y actualizar existentes. Maneja correctamente el caso de cliente nuevo vs. existente.
* **get\_menu\_item\_details**: ✅ Funcional para buscar ítems y devolver detalles o opciones de clarificación (incluyendo id\_plato, nombre\_plato, precio para las opciones).
* **manage\_order\_item**: ✅ Funcional para las acciones add, update\_quantity, y remove.
* **view\_current\_order**: ✅ Funcional para mostrar el pedido actual o indicar si está vacío.
* **get\_active\_promotions**: ✅ Funcional.
* **get\_saved\_addresses**: ✅ Mejorada y depurada. Ahora devuelve status: "no\_addresses\_found" correctamente para clientes nuevos sin direcciones guardadas. Los prints de depuración fueron clave para esto.
* **calculate\_delivery\_cost**: ✅ Funcional con la lógica de ejemplo para costos y tiempos basados en zonas.

2.3. Infraestructura y Lógica General:

* **Manejo de \_session\_user\_id**: Implementado de forma robusta mediante el callback focused\_set\_user\_id\_callback, asegurando que las herramientas y agentes tengan el contexto correcto del usuario.
* **Estructura Multi-Agente con Transferencia:** El concepto de transferir el control entre agentes (manejado por el RootAgent o por los propios agentes al finalizar su tarea) está tomando forma.

3. Estado Actual y Dificultades (Basado en el Último Log)

A pesar de los grandes avances, el último flujo de prueba completo (run\_complete\_pizzeria\_flow\_via\_root\_agent()) reveló algunos puntos críticos:

3.1. Problema Principal Actual: OrderTakingAgent\_v1 - Adición de Ítems Post-Clarificación

* **Dificultad:** Cuando el OrderTakingAgent\_v1 presenta opciones de clarificación al usuario (ej., diferentes tamaños de pizza incluyendo el precio en el string que muestra, como "Pizza Americana - Grande (S/38.90)"), y el usuario selecciona una de estas opciones, el agente intenta añadir el ítem usando ese string completo (con el precio) en la llamada a manage\_order\_item. La herramienta get\_menu\_item\_details (usada internamente por manage\_order\_item) no encuentra una coincidencia exacta para "Pizza Americana - Grande (S/38.90)" en la hoja "Menú", ya que el nombre del ítem allí es probablemente solo "Pizza Americana - Grande".
* **Impacto:** Esto causa que el ítem no se añada al pedido. Si es el único ítem, el pedido queda vacío, y el OrderTakingAgent\_v1 no puede pasar un pedido válido al DeliveryConfirmationAgent\_v1, bloqueando el flujo completo.
* **Cómo se está afrontando:** La solución propuesta y pendiente de implementación/verificación final es ajustar la instruction del OrderTakingAgent\_v1 (específicamente el Paso 2.e) para que, después de la clarificación, utilice el nombre\_plato base (obtenido de las options devueltas por la herramienta) para la llamada a manage\_order\_item, en lugar del string completo que el LLM formuló para el usuario.

3.2. Manejo de Múltiples Ítems y Negativos en un Solo Input por OrderTakingAgent\_v1

* **Dificultad:** Cuando el usuario dice "Una pizza americana grande y dos gaseosas personales por favor", el OrderTakingAgent\_v1 (en el último log) se enfocó en la "Pizza Americana" y pidió clarificación, pero no abordó inmediatamente el hecho de que "gaseosas personales" no existen. Sería ideal un manejo más integral.
* **Cómo se está afrontando:** Esto es más un refinamiento de la instruction del OrderTakingAgent\_v1. Se podría instruir para que, ante un input múltiple, intente validar todos los ítems mencionados y reporte todos los problemas (no encontrados, ambiguos) de una vez o secuencialmente antes de proceder con los válidos. Por ahora, el foco está en el problema de adición post-clarificación.

3.3. "Alucinaciones" o Desviaciones del LLM de las Instrucciones

* **Dificultad:** Hemos observado casos donde los LLM de los agentes (ej. DeliveryConfirmationAgent\_v1 con la dirección fantasma, o el RootAgent omitiendo mensajes de transición antes de una transferencia) no siguen al pie de la letra instrucciones muy explícitas, especialmente si perciben un camino más directo al objetivo o si los ejemplos en la instruction son muy influyentes.
* **Cómo se está afrontando:**
  + Reforzando las instructions con lenguaje más imperativo ("ACCIÓN OBLIGATORIA", "DEBES informar EXACTAMENTE así", "NO INVENTES...").
  + Reordenando la lógica en las instructions para priorizar chequeos críticos (ej. el caso no\_addresses\_found para get\_saved\_addresses).
  + Aceptando pequeñas desviaciones si el flujo lógico principal no se rompe y el objetivo se cumple.
  + Asegurando que las herramientas devuelvan datos claros y estructurados para minimizar la ambigüedad para el LLM.

3.4. Flujo de Prueba y Sincronización de Inputs del Usuario

* **Dificultad:** En las pruebas de flujo completo, los inputs del "usuario simulado" en el script no siempre estaban perfectamente alineados con la respuesta anterior del bot, causando que algunos inputs llegaran al agente "equivocado" o en un contexto inesperado.
* **Cómo se está afrontando:** La necesidad de reestructurar las funciones de prueba (como run\_complete\_pizzeria\_flow\_via\_root\_agent()) para que cada input del usuario sea una respuesta lógica y secuencial al output previo del bot. Esto implica capturar la respuesta del agente y usarla para informar el siguiente input simulado.

4. Lo que Falta y Próximos Pasos

1. **Resolver el Problema Crítico en OrderTakingAgent\_v1 (Prioridad #1):**
   * Implementar y probar el ajuste en la instruction (Paso 2.e) para que use el nombre\_plato base después de la clarificación.
2. **Reestructurar y Completar la Prueba run\_complete\_pizzeria\_flow\_via\_root\_agent():**
   * Asegurar un diálogo secuencial y realista.
   * Probar el flujo donde el RootAgent detecta status: "incomplete" (ej. cliente con nombre pero sin dirección después de DeliveryConfirmationAgent\_v1) y ver si transfiere correctamente al agente adecuado para recolectar la información faltante, y si luego re-verifica.
3. **Desarrollo del RootAgent\_v1 (Continuación):**
   * Refinar su instruction para manejar todas las transiciones entre sub-agentes de manera fluida.
   * Implementar la lógica para cuando un sub-agente le devuelve el control (ej. después de que CustomerManagementAgent\_v1 registra un nombre, el RootAgent debe saber pasar a OrderTakingAgent\_v1).
4. **Implementar el Flujo de "Aprobación de Cocina":**
   * Diseñar el agente o herramienta (OrderProcessingAgent?) que:
     + Tome el pedido confirmado y los datos del cliente del state (después de la verificación del RootAgent).
     + Envíe el pedido a Telegram para aprobación del personal.
     + Actualice el estado del pedido en Google Sheets a "ESPERANDO APROBACIÓN COCINA".
   * Crear el mecanismo para recibir la respuesta de aprobación/rechazo desde Telegram.
   * Diseñar el agente o herramienta que:
     + Procese la respuesta del trabajador.
     + Actualice el estado del pedido en Sheets ("APROBADO, EN PREPARACIÓN" o "RECHAZADO").
     + Notifique proactivamente al cliente el nuevo estado y, si fue aprobado, pregunte "¿Te puedo ayudar en algo más?".
5. **Desarrollar Agentes Pendientes (Según documento de avance):**
   * PaymentConfirmationAgent (si se decide separar del flujo de entrega/aprobación).
   * DailyReportAgent.
6. **Integración con Plataforma de Mensajería (WhatsApp/Telegram):** Fase final de implementación.
7. **Mejoras Adicionales:**
   * Cacheo de Menú/Promociones.
   * Manejo de errores más robusto en todas las herramientas y agentes.
   * Refinamiento continuo de las instructions de los LLM Agents basado en pruebas.

5. Conclusión General

El proyecto PizzeríaBot ha avanzado considerablemente. Los agentes individuales para gestión de clientes, toma de pedidos y confirmación de entrega están en etapas avanzadas de funcionalidad. La herramienta get\_customer\_data ha sido mejorada exitosamente. El RootAgent está comenzando a orquestar el flujo y a tomar decisiones basadas en datos.

La principal dificultad actual radica en asegurar que el OrderTakingAgent\_v1 maneje correctamente la adición de ítems después de una clarificación de opciones presentadas con precio. Una vez resuelto esto, y con una reestructuración de la función de prueba principal, podremos validar el flujo orquestado por el RootAgent de manera más efectiva y proceder con la implementación del ciclo de aprobación de cocina.

El enfoque iterativo de desarrollo, prueba y refinamiento está demostrando ser efectivo para identificar y abordar los desafíos inherentes al trabajo con LLMs y sistemas multi-agente.  
  
Estado del Proyecto: PizzeríaBot (26 de Mayo, 2025)

**1. Descripción General del Proyecto** *(Sin cambios respecto a tu versión, sigue siendo la misma base: PizzeríaBot, Python, ADK de Google, Google Sheets, tono amigable, etc. Arquitectura multi-agente.)*

**2. Logros y Avances Hasta la Fecha (Actualizado)**

Hemos alcanzado hitos significativos en el desarrollo y depuración de los componentes centrales del sistema, logrando un flujo conversacional coherente para el "camino feliz" de un nuevo cliente.

**2.1. Agentes Implementados y Funcionales (Detalle Actualizado):**

* **CustomerManagementAgent\_v1 (Versión v2.4 - Máxima Restricción)**
  + **Estado:** ✅ **Altamente Funcional y Estable.**
  + **Capacidades:**
    - Saluda a los usuarios.
    - Verifica la existencia de clientes usando get\_customer\_data.
    - **Comportamiento Corregido:** Pide el nombre completo a nuevos clientes y **espera la respuesta del usuario** antes de intentar cualquier acción de registro.
    - **Comportamiento Corregido:** Registra nuevos clientes (obteniendo Nombre\_Completo) de forma correcta y en un solo intento (después de la respuesta del usuario) usando register\_update\_customer.
    - Identifica clientes existentes y los saluda adecuadamente.
    - Finaliza su turno correctamente, pasando el control para la toma del pedido.
  + **Mejoras Realizadas:** Se refinó exhaustivamente su instruction (hasta la v2.4) para asegurar un orden estricto de operaciones y evitar llamadas prematuras a herramientas, logrando un comportamiento predecible y correcto.
* **OrderTakingAgent\_v1 (Versión v7 - Parámetros Explícitos y Finalización Precisa)**
  + **Estado:** ✅ **Altamente Funcional y Estable.**
  + **Capacidades:**
    - **Comportamiento Corregido:** Procesa el pedido inicial del usuario inmediatamente después de la transición desde CustomerManagementAgent\_v1 (problema de "pérdida de mensaje" resuelto).
    - Utiliza manage\_order\_item con todos los parámetros requeridos (action, nombre\_plato\_o\_id, cantidad, instrucciones\_especiales), gracias a instructions explícitas y un parámetro por defecto en la herramienta.
    - Utiliza get\_menu\_item\_details (con búsqueda flexible por palabras clave) para buscar ítems y manejar ambigüedad, presentando opciones con precio si es necesario.
    - Confirma ítems añadidos y pregunta si el usuario desea algo más.
    - Maneja la finalización del pedido por parte del usuario (ej. "eso es todo").
    - Llama a view\_current\_order para obtener el resumen.
    - **Comportamiento Corregido:** Proporciona un mensaje de finalización exacto y conciso, indicando que se procederá con la confirmación y los detalles de entrega, y que su tarea termina ahí.
  + **Mejoras Realizadas:**
    - Se ajustó pizzeria\_tools.py para que manage\_order\_item tenga instrucciones\_especiales como opcional con valor por defecto, aumentando la robustez.
    - Se refinó su instruction (hasta la v7) para forzar la llamada inmediata a manage\_order\_item con todos los parámetros y para asegurar una respuesta conversacional después de la ejecución de la herramienta.
    - Se precisó su mensaje de finalización para facilitar una transición limpia al siguiente agente.
* **DeliveryConfirmationAgent\_v1 (Versión v2 - Finalización Explícita)**
  + **Estado:** ✅ **Funcional para el "Camino Feliz"; Próximo a Refinamiento.**
  + **Capacidades:**
    - Inicia correctamente después de la transición desde OrderTakingAgent\_v1.
    - Llama a view\_current\_order y presenta el resumen del pedido para confirmación.
    - Procesa la confirmación del pedido por parte del usuario.
    - Llama a get\_saved\_addresses y, si no hay direcciones, pide una nueva.
    - Recibe la dirección proporcionada por el usuario y la repite para confirmación.
    - **Comportamiento Corregido:** Después de que el usuario confirma la dirección, el agente ahora **correctamente utiliza las herramientas register\_update\_customer (para guardar la dirección) y calculate\_delivery\_cost**.
    - Informa el costo y el tiempo estimado de entrega.
    - **Comportamiento Corregido:** Da su mensaje final de cierre ("¡Excelente! Hemos registrado todos los detalles...") indicando que su tarea ha concluido.
  + **Mejoras Realizadas:**
    - Se ajustó su instruction (hasta la v2) para que, después de informar el costo de envío, proceda obligatoriamente a dar su mensaje final de "verificación/todo está en marcha".
  + **Pendiente de Refinamiento (Próximo Paso):** Aunque el flujo actual funciona bien con el script, se podría mejorar su robustez para manejar casos donde el usuario proporcione la dirección en el mismo mensaje que confirma el pedido (similar al problema de "pérdida de mensaje" que tenía OrderTakingAgent\_v1).
* **RootAgent\_v1 (Versión v1 - Original)**
  + **Estado:** ✅ **Funcional para Orquestación Básica.**
  + **Capacidades:**
    - Maneja el inicio de la conversación y transfiere correctamente a CustomerManagementAgent\_v1.
    - Transfiere correctamente de CustomerManagementAgent\_v1 a OrderTakingAgent\_v1.
    - Transfiere correctamente de OrderTakingAgent\_v1 a DeliveryConfirmationAgent\_v1.
  + **Pendiente de Implementación/Prueba (Próximo Paso):** El Paso 4 de su instruction (VERIFICACIÓN DE DATOS DEL CLIENTE después de DeliveryConfirmationAgent\_v1 y las acciones subsecuentes) aún no ha sido probado exhaustivamente ni implementado en detalle.

**2.2. Herramientas (pizzeria\_tools.py) (Actualizado):**

* get\_customer\_data: Funcional.
* register\_update\_customer: Funcional, maneja nuevos registros y actualizaciones.
* get\_menu\_item\_details: **Mejorada significativamente** con búsqueda flexible por palabras clave.
* manage\_order\_item: **Mejorada** para ser más robusta (parámetro instrucciones\_especiales con valor por defecto). Funcional.
* view\_current\_order: Funcional.
* get\_saved\_addresses: Funcional.
* calculate\_delivery\_cost: Funcional.

**2.3. Script de Prueba (run\_complete\_pizzeria\_flow\_LOGICO) (Actualizado):**

* **Estado:** ✅ **Altamente Funcional y Robusto para el "Camino Feliz".**
* Se ha transformado de un script rígido a uno lógico que reacciona a las respuestas del bot.
* Se han añadido múltiples DEBUG PRINTS y se han refinado las condiciones if para manejar las variaciones del lenguaje del LLM, lo que fue **clave para superar los bloqueos y entender el comportamiento real del sistema**.
* El script ahora navega con éxito todo el flujo desde el saludo inicial hasta la finalización por parte del DeliveryConfirmationAgent\_v1.

**3. Resumen de Desafíos Superados (Actualizado)**

* **Llamadas Prematuras a Herramientas (CustomerManagementAgent\_v1):** Solucionado con instructions v2.4 extremadamente estrictas sobre el orden de los turnos y acciones.
* **Falta de Parámetros en Llamadas a Herramientas (OrderTakingAgent\_v1):** Solucionado haciendo la herramienta (manage\_order\_item) más robusta (parámetros por defecto) y con instructions v7 muy explícitas para el agente.
* **Agente No Genera Respuesta Textual Después de Herramienta (OrderTakingAgent\_v1):** Solucionado con instructions v7 que fuerzan una respuesta conversacional post-herramienta.
* **"Pérdida de Mensaje" en Transición a OrderTakingAgent\_v1:** Solucionado con instruction v7 que le indica al agente procesar el último mensaje del usuario como un posible pedido.
* **Fragilidad del Script de Prueba:** Solucionado mediante la adición de DEBUG PRINTS detallados y la creación de condiciones if más flexibles y robustas para interpretar las respuestas del bot. Esto fue un proceso iterativo crucial.
* **DeliveryConfirmationAgent\_v1 No Daba Mensaje Final:** Solucionado con instruction v2 que lo obliga a dar su mensaje de cierre.

**4. Próximos Pasos y Enfoque (Actualizado)**

Con el "camino feliz" principal funcionando sólidamente, los próximos pasos son:

1. **Modo de Prueba Interactivo:**
   * **Acción:** Habilitar y realizar pruebas conversando directamente con el bot para evaluar la naturalidad y detectar nuevos comportamientos o fallos en escenarios no cubiertos por el script.
   * **Objetivo:** Recopilar logs de estas sesiones interactivas para un análisis conjunto.
2. **Probar e Implementar el Paso 4 del RootAgent\_v1:**
   * **Acción:** Extender el script de prueba lógico (o usar el modo interactivo) para verificar si el RootAgent, después de que DeliveryConfirmationAgent\_v1 finaliza, llama a get\_customer\_data y toma las decisiones correctas según si los datos del cliente están completos.
   * **Objetivo:** Asegurar que el ciclo de verificación de datos del RootAgent funcione y que pueda re-transferir a agentes específicos si falta información.
3. **Refinamiento Proactivo del DeliveryConfirmationAgent\_v1:**
   * **Acción (Opcional por ahora, pero recomendado):** Considerar actualizar la instruction del DeliveryConfirmationAgent\_v1 para que, al igual que el OrderTakingAgent\_v1, revise el mensaje del usuario en su primer turno en busca de una dirección (si el usuario la proporciona junto con la confirmación del pedido), para evitar la "pérdida de mensaje".
4. **Manejo de Casos de Uso Adicionales y Errores (Expansión):**
   * Cliente existente.
   * Usuario quiere modificar el pedido durante la confirmación de entrega.
   * OrderTakingAgent\_v1 informando sobre ítems no encontrados en un pedido inicial múltiple (ej. "pizza y gaseosa personal").
   * Manejo de errores de herramientas de forma más elegante por parte de los agentes.
5. **Desarrollar Agentes Pendientes:**
   * PaymentConfirmationAgent.
   * DailyReportAgent.
   * KitchenApprovalAgent (requiere definir claramente su interacción y herramientas).
6. **Integración con Plataforma de Mensajería.**
7. **Mejoras Adicionales (Continuo):**
   * Cacheo de Menú/Promociones.
   * Refinamiento continuo de todas las instructions basado en más pruebas.

**5. Conclusión General (Actualizada)**

El proyecto PizzeríaBot ha demostrado una **progresión excepcional**. Hemos superado los desafíos técnicos y de lógica de los LLM para establecer un flujo conversacional funcional y coherente a través de los tres agentes principales. La estrategia de depuración detallada del script de prueba y el refinamiento iterativo de las instructions de los agentes ha sido fundamental para este éxito.

El sistema está ahora en un punto donde el "camino feliz" está validado, y podemos proceder con confianza a probarlo interactivamente y luego expandir su funcionalidad y robustez para cubrir más escenarios y los agentes restantes. El CustomerManagementAgent\_v1 y el OrderTakingAgent\_v1 están en un estado muy maduro, y el DeliveryConfirmationAgent\_v1 ha demostrado su capacidad de seguir su flujo y usar herramientas correctamente.

¡Perfecto, amigo! Entiendo tu entusiasmo por esta etapa de planificación y diseño. Es el momento ideal para plasmar la visión y la estrategia. Actualizar el "Plan de Desarrollo" es una excelente idea para mantener todo organizado.

Basado en nuestras últimas conversaciones, aquí tienes un borrador de cómo podríamos estructurar esa actualización, incorporando el "Enfoque 2.1" y las nuevas ideas.

**Actualización del Plan de Desarrollo: PizzeríaBot (28 de Mayo, 2025)**

**1. Visión General y Enfoque Actual (Enfoque 2.1)**

* **Reafirmación del Objetivo**: PizzeríaBot sigue siendo un chatbot inteligente para automatizar pedidos, integrado con Telegram y usando Google Sheets como backend. El tono es "Angelo de Pizzería San Marzano", amigable, formal y eficiente.
* **Evolución de la Arquitectura de Agentes (Enfoque 2.1 - Orquestador Central Inteligente)**:
  + **RootAgent\_v1 (Orquestador Principal)**:
    - **Rol Centralizado**: Actúa como el principal oyente e intérprete de las intenciones del usuario en casi todos los turnos.
    - **Gestión de Estado Explícita**: Utiliza variables en session.state (como current\_main\_goal y processing\_order\_sub\_phase) para rastrear el objetivo general y la sub-fase actual de la conversación. (Se requiere implementar la herramienta update\_session\_flow\_state y que RootAgent\_v1 la use).
    - **Orquestación de Fases**: Delega fases completas de la interacción a agentes especializados.
    - **Manejo de Intenciones Diversas Post-CMA**: Después de que CustomerManagementAgent\_v1 identifica/saluda al cliente, RootAgent\_v1 analiza la respuesta del usuario para determinar si es un pedido, una queja, una pregunta general, etc., y actúa en consecuencia.
    - **Manejo de "Escape Hatch"**: Recibe el control de sub-agentes si el usuario introduce una intención fuera del alcance del sub-agente actual, re-evalúa y re-dirige.
    - **Responsabilidades Directas**: Finalización del pedido (llamando a get\_customer\_data, view\_current\_order, registrar\_pedido\_finalizado), manejo de preguntas generales básicas, y potencialmente cancelaciones de pedidos.
  + **CustomerManagementAgent\_v1 (CMA - Experto en Clientes y Saludo Único)**:
    - **Responsabilidad Única**: Siempre es el primer agente especializado en interactuar con el usuario (después de la derivación inicial de Root).
    - **Flujo**: Llama a get\_customer\_data. Saluda como "Angelo de Pizzería San Marzano" (diferenciando entre usuario nuevo/conocido). Si es nuevo, solicita nombre y llama a register\_update\_customer. Finaliza su interacción preguntando "¿En qué te puedo servir esta vez?" (o similar) y transfiere a RootAgent\_v1. Ya no se involucra en la toma de pedido.
  + **OrderTakingAgent\_v1 (OTA - Especialista en Pedidos con Menú PDF)**:
    - **Sin Saludo Inicial**: Asume que CMA ya saludó.
    - **Menú PDF como Flujo Primario**: Al activarse (o si el usuario pide el menú), llama a la herramienta solicitar\_envio\_menu\_pdf. El script de Telegram enviará el PDF. OTA guía al usuario a seleccionar ítems del PDF (por código o nombre).
    - **Procesamiento de Pedido**: Usa manage\_order\_item para añadir selecciones. Mantiene la lógica para promociones (get\_active\_promotions) y preguntas específicas sobre ítems/categorías (get\_item\_details\_by\_name, get\_items\_by\_category) como apoyo al PDF.
    - **Finalización**: Al terminar el usuario de añadir ítems, resume el pedido (sin subtotal detallado por ahora, solo ítems) y transfiere a RootAgent\_v1.
    - **"Escape Hatch"**: Implementado para transferir a RootAgent\_v1 si la intención del usuario se desvía.
  + **DeliveryConfirmationAgent\_v1 (DCA - Especialista en Entrega)**:
    - **Responsabilidad**: Confirmar los ítems del pedido (resumen de OTA), gestionar la dirección de envío (usando get\_saved\_addresses, register\_update\_customer para direcciones nuevas/actualizadas), calcular y presentar costos/tiempos de envío (usando calculate\_delivery\_cost).
    - **Llamadas a Herramientas Obligatorias**: Sus instructions se reforzarán para asegurar que llame a las herramientas mencionadas en los momentos precisos.
    - **Finalización**: Al confirmar todos los detalles de entrega, transfiere a RootAgent\_v1 para la verificación final.
    - **"Escape Hatch"**: Implementado.
* **Prompts (Instructions) Concisos y Directos**: Un objetivo continuo es refinar todas las instructions para que sean claras, específicas y minimicen la ambigüedad para los LLM.

**2. Avances Recientes y Estado Actual (Desde la última actualización proyecto adk google avance 3.0.docx)**

* **Entorno de Desarrollo en Mac mini**: Solucionados los problemas con la versión de Python y la instalación de pip en el entorno virtual (.venv), que ahora usa Python 3.11.
* **Integración con Telegram (telegram\_pizzeria\_bot.py)**:
  + El bot se conecta a Telegram y puede recibir mensajes y enviar respuestas.
  + Se ha implementado la estructura básica para manejar la interacción con el Runner de ADK.
  + Se ha añadido la lógica para que el script de Telegram envíe el menu\_pizzeria.pdf cuando el OrderTakingAgent\_v1 lo solicite.
* **Diseño Detallado de instructions (Enfoque 2.1)**:
  + Se han redactado versiones avanzadas de las instructions para:
    - CustomerManagementAgent\_v1 (v3.0 - Enfoque 2.1): Con el nuevo flujo de saludo y pregunta final.
    - OrderTakingAgent\_v1 (v8.0 - Enfoque PDF y sin saludo): Con la lógica del menú PDF.
    - RootAgent\_v1 (v2.0 - Enfoque 2.1 Orquestador Central): Con la lógica de estados (current\_main\_goal, processing\_order\_sub\_phase), detección de intención post-CMA, y orquestación de fases.
  + Está pendiente el refinamiento final de la instruction del DeliveryConfirmationAgent\_v1 y la adición formal de "escape hatches" a todos los sub-agentes.
* **Nuevas Herramientas en pizzeria\_tools.py**:
  + solicitar\_envio\_menu\_pdf(): Para que OTA indique al script de Telegram que envíe el PDF.
  + update\_session\_flow\_state(): Herramienta crucial para que RootAgent\_v1 pueda modificar explícitamente current\_main\_goal y processing\_order\_sub\_phase en session.state.
  + cancel\_pending\_order\_tool(): Herramienta conceptual para que RootAgent\_v1 maneje cancelaciones.
* **Pruebas Iniciales y Depuración (Conversación log3 de Telegram y log6/log7 de consola)**:
  + Se ha logrado un flujo de pedido completo en Telegram, desde el saludo hasta el registro del pedido.
  + Se identificó que RootAgent\_v1 no estaba actualizando el estado (current\_main\_goal, processing\_order\_sub\_phase) porque un LlmAgent no puede hacerlo solo con texto en su instruction -> Solución propuesta: usar la herramienta update\_session\_flow\_state.
  + Se observó que DeliveryConfirmationAgent\_v1 a veces parecía omitir llamadas explícitas a herramientas como get\_saved\_addresses o register\_update\_customer, aunque el flujo final sugirió que de alguna manera la información correcta llegaba (posiblemente por el LLM tomando atajos o el log de consola no capturando todas las llamadas internas del agente). Se necesita reforzar sus instructions.
  + El OrderTakingAgent\_v1 manejó bien la búsqueda por ingrediente después de la corrección de su PASO 7, pero mostró confusión con el "Pack Amigos" y un error genérico al intentar listar gaseosas (posiblemente por ReadTimeout o datos faltantes en Sheets).
  + Se detectó una advertencia en la herramienta registrar\_pedido\_finalizado sobre encabezados no coincidentes en la hoja "Pedidos\_Registrados" de Google Sheets.

**3. Tareas Pendientes de Desarrollo (Corto Plazo - Próxima Iteración)**

1. **Implementar la Herramienta update\_session\_flow\_state en pizzeria\_tools.py**: Ya te pasé el código para esto.
2. **Modificar la instruction del RootAgent\_v1 para que LLAME a update\_session\_flow\_state** en todos los puntos donde se necesite cambiar current\_main\_goal, processing\_order\_sub\_phase, y manejar pending\_initial\_query y \_current\_order\_items (para limpieza en cancelaciones). (Esta es la **prioridad #1** para estabilizar el flujo).
3. **Refinar y Finalizar la instruction del DeliveryConfirmationAgent\_v1 (v5.0 o superior)**:
   * Asegurar que **OBLIGATORIAMENTE** llame a get\_saved\_addresses antes de presentar direcciones.
   * Asegurar que **OBLIGATORIAMENTE** llame a register\_update\_customer después de que el usuario confirme una nueva dirección y **ANTES** de decir "dirección guardada".
   * Asegurar que **OBLIGATORIAMENTE** llame a calculate\_delivery\_cost después de tener la dirección confirmada y **ANTES** de dar el costo de envío.
   * Implementar su cláusula "Escape Hatch".
4. **Añadir/Refinar Cláusulas "Escape Hatch" en CustomerManagementAgent\_v1 y OrderTakingAgent\_v1**: Para que transfieran al RootAgent\_v1 si el input del usuario es inesperado o fuera de su alcance.
5. **Refinar instruction del OrderTakingAgent\_v1**:
   * Para evitar la confusión con promociones no seleccionadas explícitamente (ej. el "Pack Amigos").
   * Para mejorar el manejo de cuando el usuario pide listar ítems de una categoría (ej. "muestrame las gaseosas") y la herramienta falla o no encuentra nada.
6. **Corregir Encabezados en Google Sheets**: El usuario (tú) debe asegurar que la pestaña "Pedidos\_Registrados" tenga los encabezados exactos que espera la herramienta registrar\_pedido\_finalizado.
7. **Verificar Datos de Menú en Google Sheets**: Asegurar que haya ítems en categorías como "Bebidas", "Gaseosas", "Aguas" y que estén marcados como disponibles, para evitar los status='not\_found' de get\_items\_by\_category.
8. **Monitorear ReadTimeout de Google Sheets**: Si persiste, investigar posibles soluciones (reintentos en sheets\_client.py, verificar cuotas de API).
9. **Pruebas Exhaustivas y Logs Sincronizados**: Realizar pruebas completas en Telegram y analizar los logs de consola y transcripts de Telegram juntos para depurar y refinar.

**4. Tareas Pendientes de Desarrollo (Largo Plazo / Futuras Mejoras)**

* **Implementar Lógica de "Modificar Pedido" y "Cancelar Pedido Confirmado"** a través del RootAgent\_v1 y posiblemente un OrderModificationAgent.
* **Herramientas para RootAgent\_v1 para Preguntas Generales**: Como get\_store\_info\_tool (horarios, ubicación de la pizzería, etc.) o log\_complaint\_tool.
* **Manejo de Pago**: Diseñar e implementar PaymentProcessingAgent y herramientas asociadas.
* **Flujo de "Aprobación de Cocina" Completo**:
  + Integración con Telegram para enviar notificaciones a cocina.
  + Mecanismo para que cocina apruebe/rechace.
  + Agente para procesar esa respuesta y notificar al cliente.
* **Agentes Adicionales del Plan Original**: DailyReportAgent.
* **Persistencia de Sesión Robusta**: Cambiar de InMemorySessionService a DatabaseSessionService o VertexAiSessionService para que las conversaciones no se pierdan.
* **Optimización de Llamadas a LLM**: Una vez que el flujo sea estable, revisar si hay oportunidades para reducir llamadas innecesarias.
* **Mejoras en la Naturalidad del Lenguaje y Personalidad del Bot**.
* **Considerar Webhooks para Telegram**: Para una mayor eficiencia en producción en lugar de long polling.

**5. Conclusión de esta Actualización del Plan**

El proyecto ha entrado en una fase de refinamiento de la arquitectura de agentes hacia un modelo más robusto y centrado en un RootAgent\_v1 inteligente. Los problemas principales identificados (actualización de estado por RootAgent\_v1, y sub-agentes no llamando a herramientas consistentemente) son los bloqueadores actuales. La introducción del menú PDF y el manejo de intenciones más diversas por parte del RootAgent\_v1 son los siguientes grandes pasos funcionales.

La prioridad inmediata es estabilizar el flujo del "Enfoque 2.1" asegurando que el RootAgent\_v1 pueda gestionar el estado correctamente (mediante la nueva herramienta update\_session\_flow\_state) y que los sub-agentes sigan sus instructions de manera más precisa, especialmente en cuanto a las llamadas a herramientas y la gestión de los "escape hatches”.

**Actualización del Plan de Desarrollo: PizzeríaBot (30 de Mayo, 2025)**

**1. Visión General y Enfoque Actual (Enfoque 2.1 - Orquestador Central Inteligente)**

* **Reafirmación del Objetivo**: PizzeríaBot sigue siendo un chatbot inteligente para automatizar pedidos, integrado con Telegram (futuro) y usando Google Sheets como backend. El tono es "Angelo de Pizzería San Marzano", amigable, formal y eficiente.
* **Arquitectura de Agentes (Enfoque 2.1 - Orquestador Central Inteligente)**:
  + **RootAgent\_v1 (Orquestador Principal)**: Sigue actuando como el principal oyente e intérprete de las intenciones del usuario. Ahora gestiona el Flow State explícitamente y es responsable de la orquestación de fases, el manejo de intenciones post-CMA y la gestión de "Escape Hatch". Ha mejorado significativamente la emisión de respuestas textuales en sus transiciones.
  + **CustomerManagementAgent\_v1 (CMA - Experto en Clientes y Saludo Único)**: Responsabilidad única: siempre es el primer agente especializado en interactuar con el usuario para identificarlo/registrarlo y transferir a RootAgent\_v1.
  + **OrderTakingAgent\_v1 (OTA - Especialista en Pedidos con Menú PDF)**: Especialista en procesar pedidos, manejar ítems, categorías, promociones y ofrecer menú PDF. Se ha enfocado en mejorar la adición y modificación de ítems.
  + **DeliveryConfirmationAgent\_v1 (DCA - Especialista en Entrega)**: Responsable de confirmar el pedido, gestionar la dirección de envío (guardar/actualizar), calcular y presentar costos/tiempos.
* **Prompts (Instructions) Concisos y Directos**: Un objetivo continuo es refinar todas las instructions para que sean claras, específicas y minimicen la ambigüedad para los LLM.
* **Modelo LLM:** Se ha actualizado a **Gemini 2.0**, lo que ha mejorado la capacidad de los agentes para seguir instrucciones complejas y razonar sobre el uso de herramientas.

**2. Logros y Avances Desde la Última Actualización (¡Progreso Masivo!)**

Hemos alcanzado **hitos críticos** en la estabilización y funcionalidad del bot, superando problemas complejos de orquestación y manejo de datos:

* **pizzeria\_agents.py Actualizado:**
  + **RootAgent\_v1 (v2.3) - Orquestación Sólida y Respuestas Claras:**
    - **update\_session\_flow\_state Integrado y Funcional:** El RootAgent ahora utiliza esta herramienta de forma explícita para gestionar current\_main\_goal y processing\_order\_sub\_phase. Esto ha sido el **punto de inflexión** para la estabilidad del flujo.
    - **Verificación de Pedido Pendiente (check\_pending\_order):** El RootAgent ahora llama a esta nueva herramienta al inicio de una conversación y, si encuentra un pedido en curso del día, ofrece al cliente retomarlo o iniciar uno nuevo.
    - **Eliminación de Respuestas Silenciosas:** Las instructions del RootAgent han sido refinadas para asegurar que siempre haya una respuesta textual clara en las transiciones entre agentes o al finalizar una fase, mejorando la experiencia de usuario.
  + **CustomerManagementAgent\_v1 (v3.1) - Registro Robusto:**
    - Funcional y estable en el saludo, identificación y registro de clientes. Transfiere correctamente al RootAgent.
  + **OrderTakingAgent\_v1 (v8.3) - Gestión de Pedidos Avanzada:**
    - **Manejo de Clarificación por Ambüedad de Nombres (ej., "pizza americana"):** El bot ahora pide clarificación explícita si el nombre es ambiguo, listando opciones.
    - **Manejo de "Ítem No Encontrado":** Se han mejorado las sugerencias al usuario (revisar PDF, buscar por ingrediente).
    - **Flujo "Es Todo":** El mensaje de finalización ha sido refinado para una transición fluida al DeliveryConfirmationAgent\_v1.
  + **DeliveryConfirmationAgent\_v1 (v5.3) - Confirmación de Entrega Inteligente:**
    - **Ofrecimiento de Direcciones Guardadas:** El DCA ahora detecta y ofrece explícitamente las direcciones guardadas (principal y secundaria) al cliente.
    - **Muestra Resumen Completo del Pedido:** Antes de la confirmación final, el bot presenta un resumen detallado que incluye ítems, subtotal, dirección de envío, costo de envío y total a pagar.
* **pizzeria\_tools.py Actualizado:**
  + **Herramientas Asíncronas y No Bloqueantes:** Todas las herramientas que interactúan con Google Sheets (get\_customer\_data, register\_update\_customer, etc.) son ahora async def y utilizan asyncio.get\_event\_loop().run\_in\_executor() para no bloquear el bucle de eventos. **Esto resolvió el problema de "bot colgado".**
  + **Registro Robusto de ID de Cliente:** register\_update\_customer asegura que el ID\_Cliente se registre correctamente como texto plano (str()) y utiliza value\_input\_option='RAW' para evitar corrupción de datos en Google Sheets.
  + **Manejo Correcto de Estado None:** Las herramientas ahora manejan correctamente los valores None en el estado de la sesión, evitando TypeErrors.
  + **check\_pending\_order Implementado:** Nueva herramienta para verificar pedidos pendientes en Google Sheets.
  + **Coherencia de Encabezados:** La herramienta registrar\_pedido\_finalizado ahora está perfectamente alineada con los encabezados de la hoja "Pedidos\_Registrados" que acordamos, eliminando la advertencia de inconsistencia.
* **sheets\_client.py:** Mantiene su funcionalidad de acceso a Google Sheets.
* **telegram\_pizzeria\_bot.py:** Funciona como la capa de interfaz, orquestando las interacciones con el Runner de ADK.

**3. Estado Actual y Dificultades (¡Ya son Refinamientos Finos!)**

El proyecto ha alcanzado una **estabilidad y funcionalidad muy altas** para el flujo principal de toma de pedidos. Los problemas restantes son mayormente refinamientos para casos de uso específicos y la mejora de la experiencia de usuario:

* **1. OrderTakingAgent\_v1 (OTA) - Lógica de Reemplazo/Modificación de Ítems (Prioridad Alta):**
  + **Problema:** Cuando el usuario quiere "cambiar" o "reemplazar" un ítem ya en el pedido (ej., de "Grande" a "Familiar"), el OTA tiende a añadir el nuevo ítem y luego preguntar si se desea eliminar el anterior. No realiza un reemplazo directo.
  + **Cómo se está afrontando:** La instruction de OTA (PASO 5.b) ya incluye la lógica para detectar la intención de "cambiar" y realizar remove + add. Necesitamos probar esto y, si el LLM no lo sigue consistentemente, ajustar la instruction para que sea aún más imperativa o dar ejemplos más claros.
* **2. OrderTakingAgent\_v1 (OTA) - Manejo de "la más pequeña/grande" (Nombres Contradictorios) (Prioridad Media):**
  + **Problema:** Aunque la instruction de OTA pide clarificación explícita de tamaño ("¿Cuál prefieres?"), el LLM aún puede tener dificultades con nombres de ítems que son contradictorios (ej., "Pizza Grande" es más pequeña en precio que "Pizza Familiar"). Esto puede llevar a que el LLM elija el tamaño incorrecto si el usuario no especifica el nombre exacto.
  + **Cómo se está afrontando:** La instruction ya contiene la directriz de pedir el "nombre exacto del tamaño o ID". La prueba determinará si es suficiente.
* **3. RootAgent\_v1 - Lógica de Prioridad de check\_pending\_order y Limpieza de pending\_initial\_query (Prioridad Media):**
  + **Problema:** Cuando un usuario con un pedido pendiente inicia una *nueva* interacción ("hola"), el RootAgent detecta el pedido pendiente pero puede priorizar la lógica de "iniciar nuevo pedido" (pasando el "hola" a CMA) en lugar de ofrecer inmediatamente las opciones para el pedido pendiente, llevando a un inicio de conversación ligeramente confuso. Además, el pending\_initial\_query de la nueva frase no se limpia si el flujo se desvía a la gestión de pedido pendiente.
  + **Cómo se está afrontando:** Necesitamos ajustar la instruction del RootAgent para que la rama de "pedido pendiente" tenga la máxima prioridad si check\_pending\_order devuelve pending\_order, y para que limpie pending\_initial\_query si el flujo cambia de una "nueva consulta" a la gestión de un "pedido pendiente".
* **4. DeliveryConfirmationAgent\_v1 (DCA) - Manejo de Dirección Secundaria (Caso de Prueba Específico) (Prioridad Baja):**
  + **Problema:** La instruction de DCA ya tiene lógica para manejar direcciones secundarias, pero este flujo aún no ha sido probado interactivamente en el log.
  + **Cómo se está afrontando:** Necesitamos una prueba explícita para validar este escenario.

**4. Lo que Falta y Próximos Pasos (Hoja de Ruta)**

Con la estabilidad actual, podemos enfocarnos en:

* **Validar los refinamientos existentes con pruebas exhaustivas (los puntos 1, 2, 3 y 4 de "Problemas Pendientes" de arriba).**
* **Implementar el envío real del menú PDF en telegram\_pizzeria\_bot.py:** La herramienta solicitar\_envio\_menu\_pdf ya señaliza la acción, pero la parte de Telegram de enviar el archivo real aún está pendiente.
* **Refinar la Lógica de "Escape Hatch" de Agentes:** Aunque implementados, seguir monitoreando si los agentes transfieren correctamente al RootAgent cuando la intención del usuario es claramente fuera de su alcance.
* **Integración y Prueba con Telegram (Completa):** Migrar del chat de consola a pruebas en Telegram para un entorno más realista.
* **Persistencia de Sesión Real:** Migrar de InMemorySessionService a DatabaseSessionService o VertexAiSessionService para asegurar que las conversaciones no se pierdan al reiniciar el bot.
* **Desarrollo de Agentes Pendientes (Fase Futura):** PaymentConfirmationAgent, DailyReportAgent, KitchenApprovalAgent.