Proyecto Sistemas Operativos 3° Corte

Juan Manuel Beltran Mendez, Kevin Andres Leon Tarapues, Angie Tatiana Ruiz Ruiz

June 2025

El objetivo de este proyecto consiste en un sistema interactivo en el que el robot **Pepper** actúa como un asistente de ventas, guiando al usuario por una conversación mediante comandos de voz. Recopila sus preferencias sobre productos y, con esta información, consulta una API de lenguaje natural (**DeepSeek**) para generar una recomendación personalizada. El flujo general es:

Pepper (Cliente) ⇄ Servidor Flask (API Local) ⇄ DeepSeek (IA)

Para la creación de este inicia con los 3 componentes mencionados anteriormente, iniciamos con el cliente pepper (Pricesmap5.py) donde las funcionalidades principales, como en la configuración de hardware se realizó la **Conexión NAOqi**: Establecimiento de sesión con el robot Pepper y para los servicios que se utilizaron fueron:

- ALAnimatedSpeech: Síntesis de voz con gestos
- ALSpeechRecognition: Reconocimiento de voz
- ALMemory: Gestión de memoria compartida
- ALTabletService: Control de tablet (comentado)

En el reconocimiento de voz de Pepper se le agregó como vocabulario las siguientes palabras:

["audífonos", "relojería", "joyería", "casio", "apple", "pandora", "mercadolibre", "shopee", "amazon", "facebook", "gracias", "adiós", "uno", "dos", "tres", "cuatro", "cinco", "mil", "cien", "pesos", "COP", "ana", "carlos", "maria", "jose", "luis", "sofia", "diego", "laura", "miguel", "carmen", "pedro", "juan", "andrea", "david", "patricia", "Diego","Kevin","Angie","Manuel"]

Para la conversación con Pepper se da en las siguientes fases:

Presentación inicial

- Saludo de bienvenida a AK Watch
- Explicación del servicio

Captura de nombre

- Pregunta personalizada por el nombre
- Almacenamiento para personalización posterior

Proceso de consulta

- Iteración sobre preguntas dinámicas del servidor
- Personalización con nombre del usuario

- Envío de respuestas al servidor

Entrega de resultados

- Recepción de análisis de Deepseek
- Presentación personalizada del resultado
- Despedida personalizada

A Continuación se va a dar las funciones más importantes para el funcionamiento de este y tambien cual es su propósito o función dentro de este código:

- Función: escuchar_respuesta()
- pythondef escuchar_respuesta(asr, memory, max_palabras=2, tiempo_maximo=15):

Propósito: Captura de respuestas del usuario

Parámetros configurables:

- max_palabras: Límite de palabras por respuesta
- tiempo_maximo: Timeout en segundos

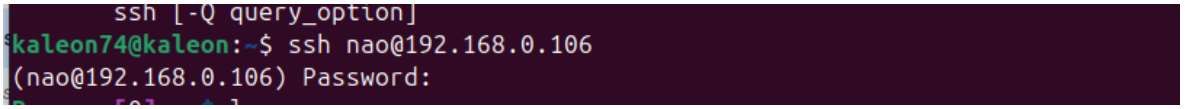
Manejo de errores: Limpieza de memoria y reintentos

Comunicación HTTP
IP del servidor: 192.168.0.107:9559
Endpoints utilizados:

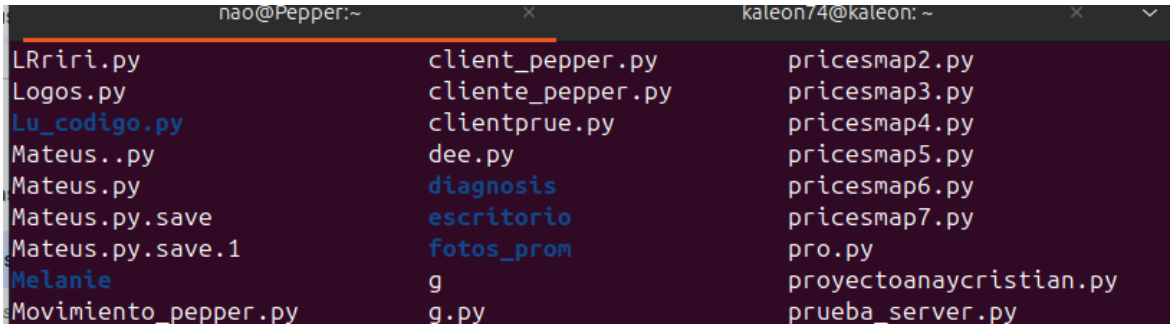
- /reiniciar: Reinicio de conversación
- /siguiente_pregunta: Obtención de preguntas
- /respuesta: Envío de respuestas
- /resultado_final: Recepción de análisis
- /obtener_nombre: Consulta de nombre almacenado

Es importante conocer la dirección IP para dos factores claves el primero es, establecer conexión con ssh e ingresa a su interfaz para este caso sabemos que es:
IP: 192.168.0.106
ssh nao@IP de Pepper

ssh nao@192.168.0.106



Nuestro cliente en Pepper recibe el nombre de pricesmap7.py



Es importante resaltar que el puerto por el cual se enviara la info al servidor de escucha sera el preestablecido y abierto por defecto, el puerto 9559

- Módulo ASR, debe limpiarse y apagarse para evitar que pepper entre en un bucle de escucha infinita. para ello se implemento una linea de comando fundamental en el desarrollo del mismo
  Agregué asr.pause(True) antes de asr.setVocabulary()

Para el servidor

- Este se alojara en el pc local por lo cual es importante conocer su IP, importante debe alojarse dentro de la misma red de Pepper 192.168.0.0/24, para nuestro caso se define la siguiente red.
  IP Local: 192.168.0.107
  Puerto de escucha: 9559
- Nuestro servidor escuchará a Pepper por el puerto 9559, asi se podra emular una integración de chatbot directamente.

Continuamos con el siguiente parámetro que es Servidor Flask (serverdef4.py), para las funcionalidades del servidor y para la gestión de la conversación con Pepper fue la siguiente para realizar preguntas:

("nombre", u"¿Cuál es tu nombre?"), ("producto", u"¿qué producto deseas consultar?"), ("plataformas", u"¿en qué plataformas deseas consultar?"), ("precio", u"¿cuál es tu rango de precio?"), ("marca", u"¿qué marca prefieres?"), ("cantidad", u"¿cuántos resultados quieres comparar?")

Para la ver el estado de la conversación se implementó en:

```
conversacion = {
  "estado": 0,  # índice actual de pregunta
  "respuestas": {}  # almacenamiento de respuestas
}
```

Para esta parte implementamos lo siguiente:
Endpoints REST API
GET /siguiente_pregunta

- Propósito: Envía la siguiente pregunta en secuencia
- Lógica: Salta automáticamente la pregunta del nombre

- Respuesta: JSON con pregunta o indicador de finalización

POST /respuesta
- Propósito: Recibe y almacena respuestas del usuario
- Procesamiento: Mapea respuestas a claves específicas
- Estado: Avanza el índice de conversación

GET /resultado_final
- Propósito: Genera análisis usando Deepseek
- Prerequisito: Todas las preguntas respondidas
- Procesamiento: Construcción de prompt personalizado

POST /reiniciar
- Propósito: Reinicia el estado de conversación
- Uso: Limpieza entre sesiones

Integración con DeepSeek
Para establecer comunicación entre el servidor Flask y el modelo de lenguaje DeepSeek, es necesario realizar una integración mediante una API RESTful segura. Esta configuración permite que el sistema envíe las respuestas recolectadas por Pepper como contexto dentro de un prompt, y reciba a cambio un análisis en lenguaje natural generado por IA. La integración se configura de la siguiente manera con los parámetros de conexión y demás:

Configuración API :

```
pythonAPI_KEY = 'sk-53751d5c6f344a5dbc0571de9f51313e'
API_URL = 'https://api.deepseek.com/v1/chat/completions'
construir_prompt()
```

Estructura del prompt:
- Personalización con nombre del cliente
- Parámetros de consulta estructurados
- Instrucciones específicas para el análisis
- Longitud objetivo: ~700 palabras
- Contexto de marca: AK Watch

Elementos del prompt:
- Características básicas del producto
- Análisis de precios
- Tabla comparativa
- Consejos personalizados para emprendedores
- Branding de AK Watch

consultar_deepseek()
- Modelo: deepseek-chat

- Temperatura: 0.85 (balance creatividad/precisión)
- Manejo de errores: Captura de excepciones HTTP
- Encoding: UTF-8 para caracteres especiales

Una de las características Técnicas:
se codifica con la siguiente funcion:
```
def safe_encode(texto):
  """Convierte texto a UTF-8 de forma segura"""
```

- Soporte para caracteres especiales en español
- Compatibilidad con Python 2.7 (NAOqi)

Personalización Avanzada

- Nombre dinámico: Captura y uso throughout la conversación
- Contexto de marca: Integración natural de AK Watch
- Fallbacks: Manejo de "no entendi" y respuestas vacías

Robustez del Sistema

- Timeouts configurables: Prevención de bloqueos
- Limpieza de memoria: Evita interferencias entre sesiones
- Reintentos: Manejo de errores de red
- Estado persistente: Mantiene conversación entre requests

Flujo de Datos Completo

El flujo de datos completo describe cómo la información viaja a lo largo del sistema, desde la interacción inicial del usuario con el robot Pepper hasta la generación y entrega de una respuesta personalizada mediante inteligencia artificial. Este flujo combina componentes físicos (Pepper), servicios internos (NAOqi), una arquitectura cliente-servidor basada en Flask y la integración con un modelo de lenguaje externo (DeepSeek).

A través de una serie de etapas bien definidas —inicialización, interacción, consulta, análisis y entrega— se asegura una experiencia conversacional fluida, robusta y adaptada al contexto comercial de AK Watch. Cada fase en este flujo cumple una función crítica para garantizar que la información sea recolectada, procesada y devuelta de manera precisa, contextualizada y natural para el usuario.

La siguiente sección detalla paso a paso cómo se realiza este recorrido de datos, resaltando la sincronización entre hardware, software, lógica de negocio y servicios externos de IA.

1. **Inicialización**:
    - Pepper se conecta a servicios NAOqi
    - Servidor Flask inicia en puerto 9559
    - Cliente reinicia conversación en servidor

2. **Interacción**:
    - Pepper presenta AK Watch y solicita nombre
    - Cliente envía nombre al servidor
    - Servidor personaliza experiencia con nombre
3. **Consulta estructurada**:
    - Servidor envía preguntas secuenciales
    - Pepper presenta preguntas personalizadas
    - Cliente captura y envía respuestas
4. **Procesamiento**:
    - Servidor construye prompt personalizado
    - API Deepseek genera análisis detallado
    - Servidor retorna resultado al cliente
5. **Entrega**:
    - Pepper presenta análisis personalizado
    - Despedida con nombre del usuario
    - Limpieza de recursos

Consideraciones de Implementación

La implementación del sistema Pepper Chatbot – AK Watch requiere la coordinación de múltiples componentes de hardware, software, red y servicios externos. Dado que la solución combina un robot humanoide interactivo, un backend de control lógico y una API de inteligencia artificial, es fundamental establecer correctamente las condiciones técnicas mínimas para su funcionamiento fluido y confiable. A continuación se dará a detalles de las consideraciones que se tuvieron en cuenta para poder realizar la práctica:

Hardware

- Robot: Pepper (NAOqi 2.x)
- Conectividad: WiFi local 192.168.0.x
- Recursos: Procesamiento distribuido (Pepper + PC)

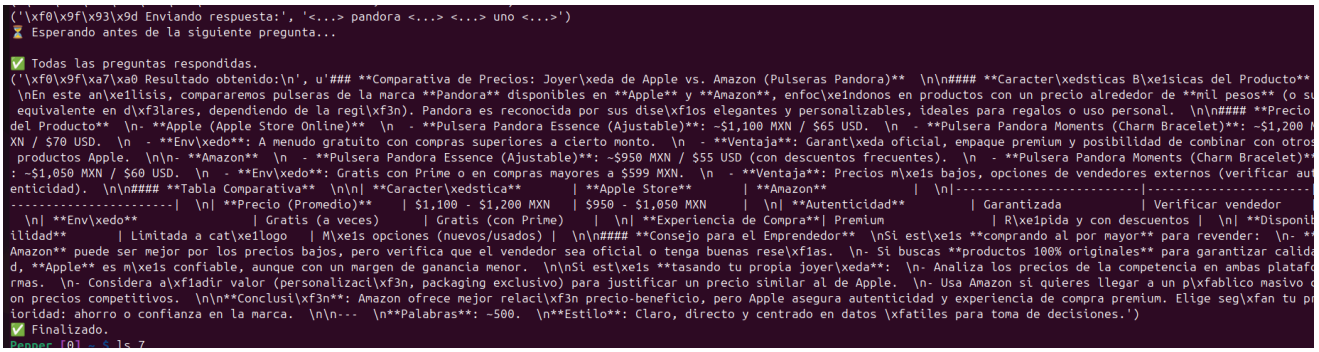
Software

- Pepper: Python 2.7, NAOqi SDK
- Servidor: Python 2.7, Flask
- Python 3.x para servidor Flask.
- API: Deepseek chat completion

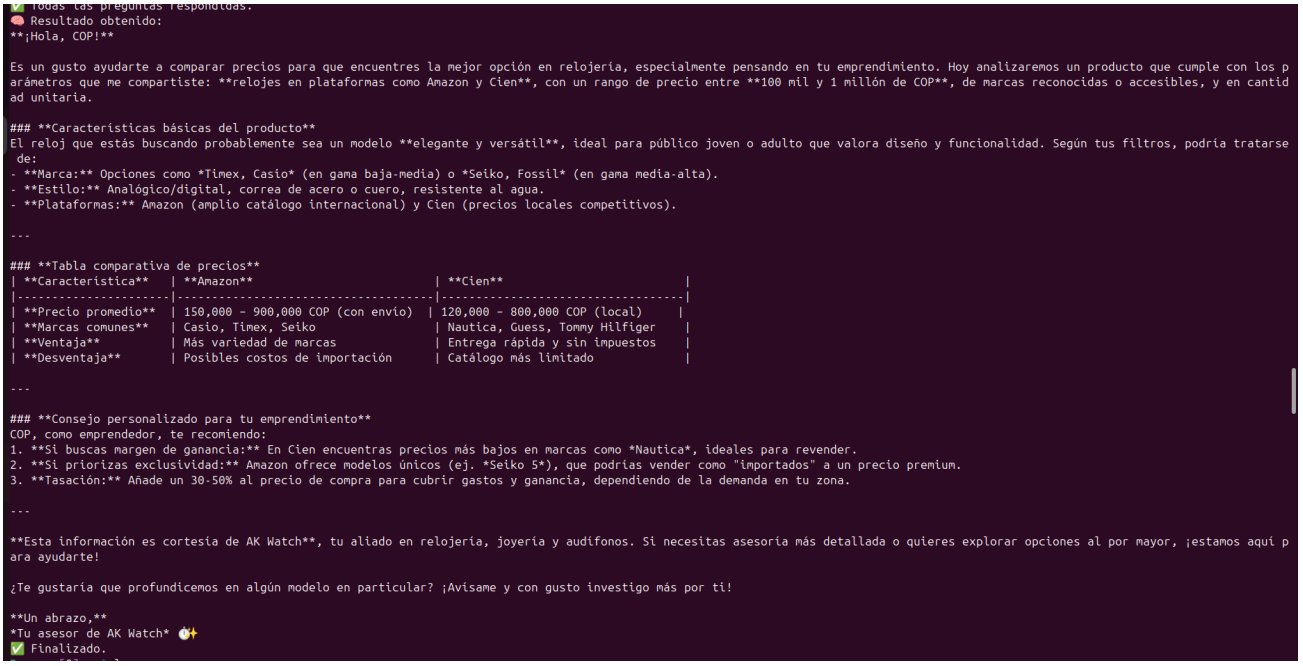
Configuración de Red

- Pepper y servidor en misma red local
- Puerto 9559 para comunicación HTTP
- Acceso a internet para API Deepseek

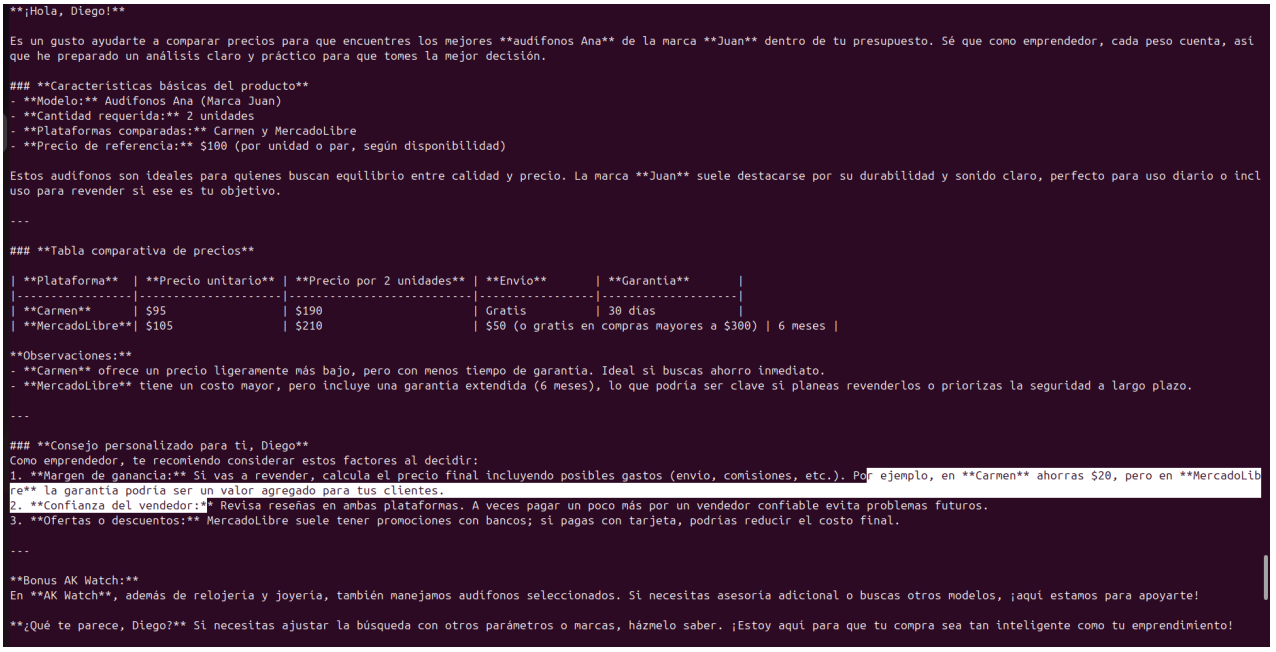
Funcionamiento y prueba de Pepper



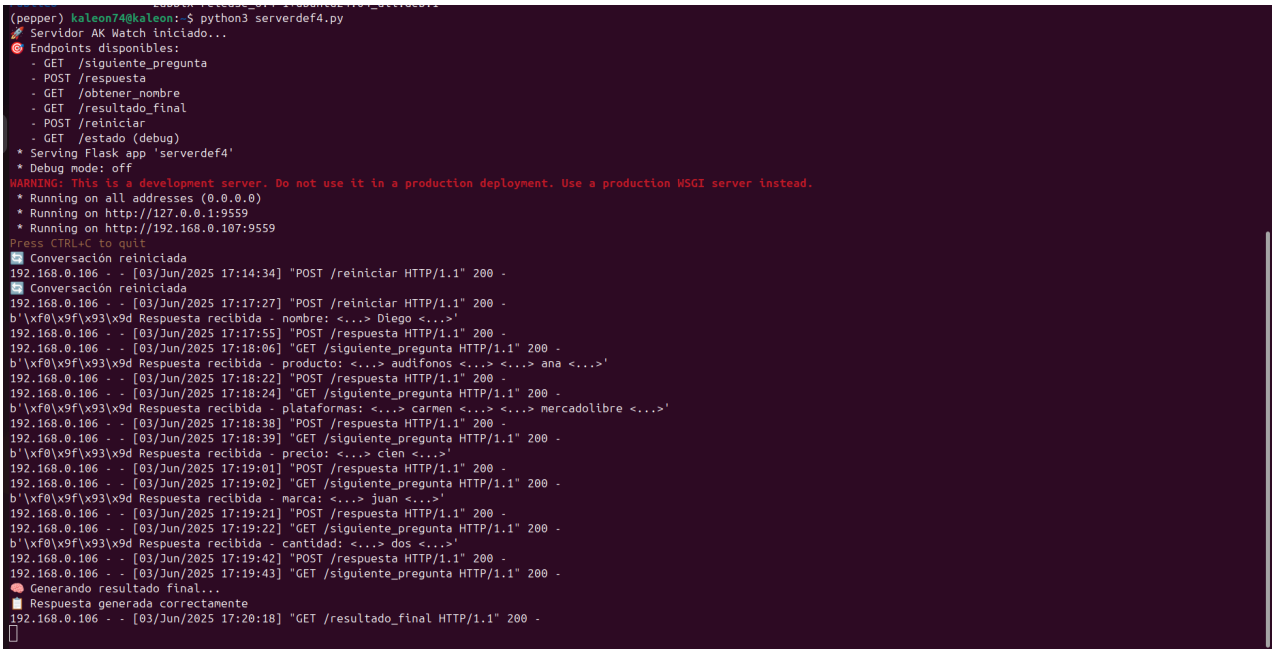
Prueba 2



Prueba Final



Servidor Local



Puntos importantes a tener en cuenta para el laboratorio con Pepper

1. **Vocabulario específico**: Adaptado al dominio de negocio
2. **Personalización**: Uso consistente del nombre del usuario
3. **Manejo de errores**: Fallbacks y recuperación de sesión
4. **Integración API**: Prompt engineering para resultados relevantes
5. **Experiencia de usuario**: Flujo conversacional natural
6. **Escalabilidad**: Arquitectura modular para futuras expansiones

Posibles Mejoras

- Implementación de logging detallado
- Manejo de múltiples usuarios simultáneos
- Integración con bases de datos de productos
- Métricas de uso y satisfacción
- Soporte para más idiomas
- Integración con sistemas de inventario

Para finalizar se añade el enlace del repositorio en github para que el docente pueda ver mas a profundidad, https://github.com/kaleon74/ProyectoCorte3_Pepper.