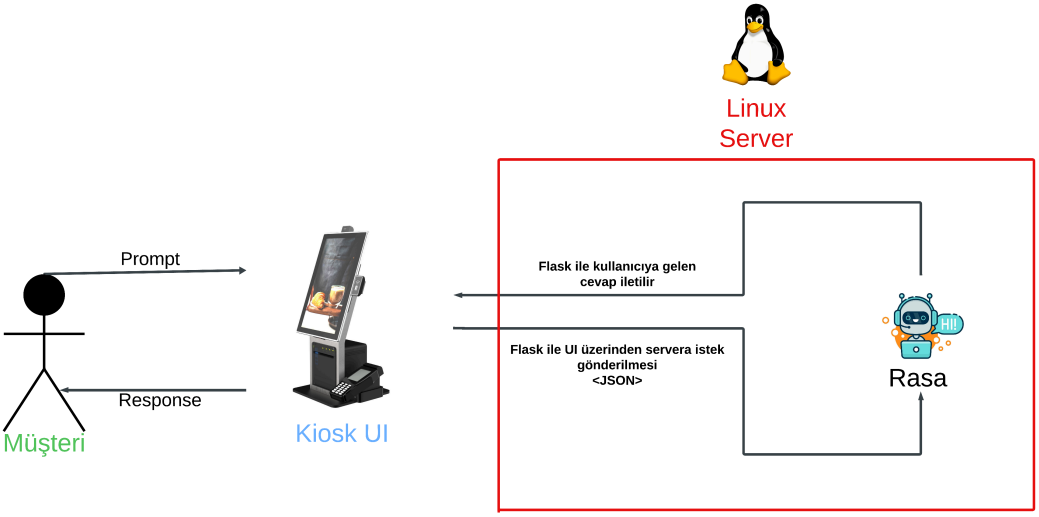
RASA - Kiosk Entegrasyonu ve Sistemin Linux Server Üzerinde Çalıştırılması

Toplantı raporlarında yazıldığı kadarıyla kiosk cihazlarında yer alacak sistem arayüz tasarımı firma tarafından yapılacak. Bizim için geriye kalan işlemler; UI üzerinden gelecek kullanıcı promptlarını RASA botumuzu ulaştırmak ve ardından üretilen cevabı yine aynı UI üzerinden kullanıcıya ulaştırmak olacak. Bu sistem mimarisi az çok aşağıdaki gibi olacaktır:



Flask Nedir?

Flask bir web framework'üdür ve Python programlama diliyle yazılmıştır. Flask, geliştiricilere basit ve hızlı bir şekilde web uygulamaları veya API'ler oluşturma imkanı tanır. Bu projede, Kiosk UI'den gelen kullanıcı mesajlarını RASA botuna iletmek ve botun cevabını geri göndermek için Flask kullanabilirizz. Burada JSON formatıyla kullanıcıdan gelen mesjlar servera gönderilir ve alınan cevap yine UI'ye verilir.

Tabi flask benim karşılaştığım bir seçenek, durumumuza göre buna alternatif olarak da kullanabileceğimiz web framework'ler şu şekilde mevcut:

Django: Daha büyük projeler ve daha fazla işlevsellik gerektiren projeler için uygundur. Ancak, daha karmaşıktır ve daha fazla konfigürasyon gerektirir.

FastAPI: Yüksek performanslı ve modern bir web framework'üdür. Asenkron yapıya sahiptir ve API geliştirme için idealdir.

Bottle: Flask'a benzer, minimalist bir web framework'üdür. Küçük ve bağımsız uygulamalar için uygundur.

- Flask ve RASA beraber kullanılarak geliştirilmiş bazı projeler:
  - RASA-Flask-Chatbot-v1

    - CovidBot-RASA
- Yardımcı olabilecek bir tutorial

Serverda Kullanıcıya Özel Oturum Açılması:

Her müşteri için benzersiz bir oturum açmak, mesajların doğru bir şekilde takip edilmesini ve yönetilmesini sağlar (bu sayede slotlarda kişilere özel bilgiler tutulacak ve sohbet ikili şekilde olacak). Bu oturum yönetimi için Redis ve MongoDB seçenekleri mevcut. Burada Redis ve MongoDB'nin birbirlerine göre avantaj ve dezavantajları şunlardır:

Redis:

- Avantajlar:
    - Hafıza tabanlı olması nedeniyle çok hızlıdır.
    - Oturum yönetimi için basittir.
    - Geçici veri saklama için idealdir.
- Dezavantajlar:
    - Veriler kalıcı değildir (her restart sonrası silinir, kalıcılık sağlamak için ek yapılandırmalar gerekebilir).
    - Büyük veri kümeleri için uygun değildir.

MongoDB:

- Avantajlar:
    - Kalıcı veri saklama sağlar.
    - Karmaşık sorgular ve büyük veri kümeleri ile daha iyi çalışır.
    - Daha fazla esneklik ve genişletilebilirlik sağlar.
- Dezavantajlar:
    - Redis kadar hızlı değildir.
    - Daha karmaşık bir kurulum ve bakım gerektirir.

Örnek Redis Kullanımı:

- **Oturum Başlatma**: Müşteri kiosk UI üzerinden ilk mesajını gönderdiğinde, Flask sunucusu Redis'te bir oturum anahtarı oluşturur ve bu anahtarı müşterinin mesajlarına bağlar.
- **Oturum Takibi**: Her müşteri mesajı, oturum anahtarı ile birlikte RASA botuna iletilir. Bu anahtar, mesajların doğru oturumda takip edilmesini sağlar.

```
import redis

# Redis bağlantısı kurullur
r = redis.Redis(host='redis_server', port=6379, db=0)

@app.route('/chat', methods=['POST'])
def chat():
    user_message = request.json.get("message")
    sender_id = request.json.get("sender_id")

    # Kullanıcı oturumu için benzersiz bir anahtar oluşturulur veya mevcut
anahtarı al
    session_key = f"session_{sender_id}"
    if not r.exists(session_key):
        r.set(session_key, '')

    response = requests.post(RASA_SERVER_URL, json={"sender": sender_id,
"message": user_message})
    bot_response = response.json()

    return jsonify(bot_response)
```

Mesajların İşlenmesi ve Geri İletilmesi

Flask sunucusu, kullanıcı mesajlarını RASA botuna iletir ve botun cevabını alarak kullanıcıya geri gönderir.

1. Mesaj Gönderimi:

- Kullanıcı mesajı JSON formatında Flask sunucusuna POST isteği olarak gönderilir.

  Örnek POST isteği:

  ```
  {
          "message": "Merhaba",
          "sender_id": "unique_user_id"
  }
  ```

2. Cevap Üretimi:

- Flask sunucusu, kullanıcıdan gelen mesajı alır ve RASA botuna bir POST isteği yapar (Buranın detayını tam bilmiyorum)

3.Cevap Gönderimi:

- RASA botu, gelen mesajı işleyerek bir cevap üretir ve bu cevabı Flask sunucusuna geri gönderir.
- Flask sunucusu, RASA botundan gelen cevabı JSON formatında Kiosk UI'ye geri gönderir.

Docker'ın Rolü ve Kullanımı

Docker, uygulamaların bağımsız ve izole bir şekilde çalıştırılmasını sağlayan bir konteyner platformudur. Bu projede Docker, Flask sunucusu ve RASA botunu kapsayan bir konteyner ortamında çalıştırılabilir.

- Kapsülleme: Docker, tüm bağımlılıkları ve konfigürasyonları içeren izole bir ortam sağlar. Bu, uygulamanın her yerde aynı şekilde çalışmasını garanti eder.
- Kolay Yönetim: Docker Compose ile birden fazla konteyner kolayca yönetilebilir ve başlatılabilir. Bu, uygulamanın farklı bileşenlerini tek bir komutla çalıştırmayı sağlar.
- RASA içinde `docker-compose.yml` şeklinde bir dosya da yazılır bu işlem için. (Örnek kullanım rasa-webchat)

Genel Akış

- Müşteri Mesaj Gönderir (Prompt):
  - Kiosk UI, kullanıcıdan mesaj alır.
  - Mesaj Flask sunucusuna JSON formatında POST isteği olarak gönderilir.
- Flask Sunucusu İsteği Alır ve RASA Botuna Gönderir:
  - Flask sunucusu, Kiosk UI'den gelen mesajı RASA botuna iletir.
- RASA Botu Mesajı İşler ve Cevap Üretir:
  - RASA botu, gelen mesajı işler ve bir cevap üretir.
  - Bu cevap Flask sunucusuna geri gönderilir.
- Flask Sunucusu Cevabı Alır ve Kiosk UI'ye Geri Gönderir:
  - Flask sunucusu, RASA botundan gelen cevabı Kiosk UI'ye iletir.
- Kiosk UI Cevabı Kullanıcıya Gösterir (Response):
  - Kiosk UI, Flask sunucusundan gelen cevabı kullanıcıya gösterir.

Minimum Sistem Gereksinimleri

1 - Minimum Donanım Gereksinimleri

- **CPU**:
  - En az 4 çekirdekli modern bir işlemci
  - Önerilen: 8 çekirdekli işlemci

- **RAM**:
  - En az 8 GB
  - Önerilen: 16 GB veya daha fazlası (Bu yoğun kullanıma göre belirlenecek bir şey aslında, mağazada yğun bir kullanıcı trafiği olacaksa buna göre bir RAM miktarı belirlenmelidir.)

- **Depolama**:
  - En az 256 GB SSD hafızası
  - Önerilen: 512 GB veya daha fazla (RASA modeli, Redis ve diğer log dosyaları için)
  - Şuan yazdığımız RASA modeli tüm eklentileriyle beraber 3.3 GB boyut kaplıyor.
  - Eğer yerelde bir database'de bilgi tutma gibi bazı işlemler yapılacaksa buna göre ihtiyaç yine artabilir.

- **Ağ Bağlantısı**:
  - Burada miktar hakkında çok açık bir bilgi bulamadım ancak bu yine kullanıcı trafiğine bağlı olarak değişebilecek bir durum.
  - Yoğun bir ağ trafiğinde müşteri isteklerini alabilecek bir sistem tasarımında, muhtemelen güçlü bir ağ bağlantısı da gerekecektir.

2 - Yazılım Gereksinimleri

- **İşletim Sistemi**:
  - Herhangi bir Linux dağıtımı olabilir (Önerilen Ubuntu 22.04 veya Debian dağıtımı)

- **Docker**:
  - Docker Engine ve Docker Compose kurulumu

- **Python**:

    - Kararlı bir Python sürümü (sürümler hakkında çok detaylı bir bilgim yok muhtemelen en son sürüm işimize yarayacaktır)
    - Flask ve Redis gibi kullanacağımız bazı paketlerin eklentileri.

- **Flask Sunucusu**:
    - Flask framework'ü ve gerekli diğer Flask eklentileri (örn: Flask-RESTful, Flask-CORS)

- **Redis Sunucusu**:
    - Redis server (oturum yönetimi için) ve gerekli yapılandırmalar

- **Ağ Trafiğini Yönetme**:
    - Bunun için konuşulmadı ancak eğer yoğun bir kullanım olursa ağ trafiğini yönetebilmek için ağ trafiği yöneticileri (load balancer) kullanılabilir.
    - Nginx veye Apache gibi

Bu gereksinimler, geniş bir mağaza için kapsamlı bir RASA tabanlı bot asistanı, Flask modülü, ve Redis oturum yönetimini destekleyecek yeterliliktedir. Daha yüksek trafik veya daha karmaşık işlemler için kaynakların arttırılması gerekebilir.