ChatGPT

Instagram DM için ManyChat + OpenAI GPT-4 Entegrasyon Planı

Bu plan, bir Instagram eğitmeninin gelen DM mesajlarını ManyChat ve OpenAI GPT-4 kullanarak otomatik yanıtlaması için uçtan uca tüm adımları içeren teknik bir rehberdir. Backend sistemi Node.js/TypeScript ile sıfırdan kurulacak, Railway üzerinde çalışacak; opsiyonel olarak test amacıyla şifre korumalı bir React **Admin Paneli** de sunulacaktır. Aşağıda, geliştirme ortamından ManyChat ayarlarına, OpenAI entegrasyonundan deployment ve izlemeye kadar hiçbir adımı atlamadan detaylı bir implementasyon planı verilmiştir. **(Not: Planın çıktısı Türkçe hazırlanmıştır.)**

1. Geliştirme Ortamı ve Proje Yapısı (Google Antigravity IDE ile Uyumlu)

Proje Teknolojileri: Backend için Node.js + Express framework'ü TypeScript ile kullanılacaktır. Bu sayede güçlü tip desteğiyle daha hatasız bir kodlama sağlanır. İsteğe bağlı olarak bir React + TypeScript frontend paneli de hazırlanacaktır. Proje, Google'ın bulut tabanlı **Antigravity IDE**'si ile geliştirilmeye uygun şekilde yapılandırılacaktır.

- **Google Antigravity IDE Uygunluğu:** Antigravity IDE, çoklu proje ve AI destekli kod tamamlama özelliklerine sahip bir geliştirme ortamıdır. Proje kodunun GitHub üzerinden yönetilmesi ve Antigravity ile açılması önerilir. Bu amaçla yeni bir Google hesabı oluşturulup ilgili GitHub deposu bu hesapla ilişkilendirilebilir. Antigravity IDE içinde hem backend (Express projesi) hem de opsiyonel frontend (React projesi) paralel olarak açılıp çalıştırılabilir. Her iki kısım da aynı monorepo içinde veya ayrı repolar olarak yönetilebilir.

- **Proje Yapısı:** Backend klasörü içinde Express uygulaması (ör. `src/index.ts`), route'lar ve servis entegrasyon kodları bulunur. Frontend paneli ayrı bir dizinde (ör. `admin-panel/`) tutulabilir. Ortak ayarlar için `.env` dosyası veya benzeri yapı kullanılacak ancak bu dosya Git'e **işlenmeyecek** – gizli anahtarlar dağıtım platformunda ayarlanacak.

- **Bağımlılıklar:** Backend için `express`, `axios` veya OpenAI istemci kütüphanesi (`openai`), `dotenv` gibi paketler eklenecek. Frontend için React, form yönetimi için gerekirse ek kütüphaneler (basit bir form olduğundan belki gerekmez) ve istekler için `fetch` veya axios kullanılabilir. Bu teknolojiler Google Antigravity IDE'de sorunsuz desteklenmektedir.

  Not: Geliştirme sırasında kodu yerel olarak çalıştırmak isterseniz, ManyChat webhook'ini test etmek için `ngrok` gibi bir tünelleme aracı kullanabilirsiniz. Ancak Railway üzerinde çalışır hale getirdiğinizde ManyChat doğrudan o URL'yi kullanacaktır.

2. ManyChat Webhook Endpoint'inin Kurulumu (Instagram DM'den Sunucuya)

Öncelikle, ManyChat'in Instagram otomasyonunu kullanarak botumuzu kurmamız gerekiyor. ManyChat, Instagram iş hesabınıza bağlanarak gelen DM'leri yakalayabilir. Biz bu yapı içinde, her yeni mesaj

geldiğinde ManyChat'in **harici bir isteği (external request)** bizim sunucumuza yönlendirmesini sağlayacağız. Bu sayede kullanıcı mesajını backend alacak, işleyip yanıt üretecek ve tekrar ManyChat API'sini kullanarak kullanıcıya gönderecek.

Adım 2.1: ManyChat & Instagram Kurulumu – Eğer henüz yapılmadıysa, ManyChat hesabınızı oluşturun ve Instagram sayfanızı ManyChat'e bağlayın (Instagram bir **kanal** olarak eklenmeli). ManyChat panelinde **Settings > Instagram** bölümünden Instagram Business hesabınızı ManyChat'e entegre edin [1]. Ayrıca ManyChat'te Pro plan gerekliliklerine dikkat edin (harici istekler ve API erişimi Pro planda mümkündür).

Adım 2.2: ManyChat API Anahtarını Alma – ManyChat'in **Public API**sini kullanabilmek için bir API token'ına ihtiyacımız var. ManyChat panelinde **Settings > API** yolunu izleyip bir API anahtarı oluşturun ve kopyalayın. Bu anahtar, sunucumuzun ManyChat üzerinden mesaj göndermesi için gereklidir (aşağıda environment ayarlarında kullanacağız).

Adım 2.3: Backend'de Webhook Endpoint Tanımı – Express tabanlı sunucumuzda bir HTTP POST endpoint'i tanımlayacağız (örneğin `/webhook` yolu). Bu endpoint, ManyChat'ten gelen verileri alıp işleyecek. Aşağıda TypeScript ile basit bir Express uygulaması örneği verilmiştir:

```
import express from 'express';
import { json } from 'body-parser';

const app = express();
app.use(json());

app.post('/webhook', async (req, res) => {
    const body = req.body;
    console.log("ManyChat Webhook payload:", body);
    // ManyChat'ten gelen verileri al
    const subscriberId = body.subscriber_id;
    const userMessage = body.message || "";  // Kullanıcı mesaj metni
    let linkUrl: string | null = null;
    // Mesaj içindeki link'i tespit et (varsa)
    const urlMatch = userMessage.match(/https?:\/\/\S+/i);
    if (urlMatch) {
        linkUrl = urlMatch[0];
        // İsteğe bağlı: linki mesajdan çıkarabilirsiniz
        // userMessage = userMessage.replace(linkUrl, '');
    }

    // ... (OpenAI API çağrısı ile cevap üretilecek - bir sonraki adımda)

    res.sendStatus(200); // ManyChat'e isteğin alındığını belirt
});
```

Yukarıda: - `app.use(json())`, JSON gövdeli POST isteklerini otomatik parse eder. ManyChat webhook'i JSON formatında veri gönderecektir. - `body.subscriber_id` : ManyChat, kullanıcıyı tanımlamak için bir **contact id** (subscriber id) gönderecek. Bu, mesajı kimin gönderdiğini belirten benzersiz bir kimliktir. - `body.message` : Bu alana, ManyChat'e yapılandırırken kullanacağımız

değişken değeri gelecek (kullanıcının gönderdiği mesaj metni). Bir sonraki bölümde ManyChat'in harici isteğini ayarlarken, gönderilecek JSON içinde `message` alanına kullanıcının son mesajını koyacağız.

Adım 2.4: Kullanıcı Mesajı ve Link Ayrıştırma – Kod parçasında, `userMessage` değişkenine gelen metni aldık. Ardından basit bir RegEx ile mesaj içinde olası bir URL linki var mı kontrol ediyoruz. Eğer varsa, `linkUrl` değişkenine atıyoruz. İhtiyaca göre: - Link bulunduysa AI cevabına ek bir not düşülebilir veya sistem promptuna "Kullanıcı şu linki paylaştı: [URL]" şeklinde bir bilgi geçirilebilir. (GPT-4 modelinin internet erişimi olmadığı için link içeriğini otomatik okuması mümkün değildir. Yine de linkin varlığını bilmesi gerekebilir.) - Alternatif olarak, eğer AI'in linki görmezden gelmesini istiyorsanız, linki kullanıcı mesajından çıkarabilirsiniz (örnekte yorum olarak gösterilen satır).

Bu ayrıştırma sayesinde, kullanıcı mesajında **varsa link bilgisini kaybetmeden** AI'e göndermeye hazır hale getiriyoruz. (Örneğin, kullanıcı "Şu siteye bir bak: https://örnek.com" diye yazarsa, linkUrl = "https://örnek.com" olacak.)

Adım 2.5: ManyChat Webhook'inin Ayarlanması – Şimdi ManyChat panelinde, Instagram'dan gelen mesajları bu endpoint'e yönlendirecek bir otomasyon kuralı kuracağız. ManyChat'te **Varsayılan Yanıt (Default Reply)** özelliğini kullanacağız. Default Reply, botunuzun önceden tanımlanmamış herhangi bir kullanıcı mesajına verdiği cevaptır ve tüm anahtar kelime eşleşmelerine vs. yakalanmayan mesajları yakalar [2] [3].

- ManyChat panelinde **Settings > Instagram** kısmına gidin, "Default Reply" ayarını bulun. Default Reply'yi etkinleştirmek ve özelleştirmek için bir akış (flow) seçilebilir veya oluşturulabilir [3]. Yeni bir otomasyon oluşturup bunu Default Reply olarak atayacağız.
- Yeni bir **Flow** oluşturun ve adını örneğin "Incoming DM Webhook" koyun (sadece düzen içinde anlayabilmek için). Bu akışın ilk adımı olarak bir **"External Request" (Harici İstek)** eklenecek [4]. External Request, ManyChat'in mesaj akışını durdurup harici bir sunucuya istekte bulunmasını sağlar [5] [6].
- External Request bloğunu ekleyince, istek detaylarını dolduracağız:
- **Request URL:** Buraya biraz önce oluşturduğumuz Railway backend URL'sini ve `/webhook` yolunu gireceğiz. (Örneğin, daha sonra Railway'den alacağımız domain `https://mybot-production.up.railway.app/webhook` gibi olacaktır. Şimdilik placeholder koyabilirsiniz.)
- **Request Type:** `POST` seçin.
- **Headers:** Buraya ManyChat API anahtarı koymamıza **gerek yoktur** – çünkü bu istek ManyChat'ten bizim sunucumuza gidecek. (ManyChat'ten gelen istekleri doğrulamak için isterseniz basit bir sabit token kullanabilirsiniz, ancak proje gereği şart değil.) İsterseniz bir güvenlik önlemi olarak, ManyChat External Request gönderimine özel bir sabit token ekleyip sunucuda kontrol edebilirsiniz.

- **Body (JSON)**: ManyChat, isteğin gövdesini bizim belirlememize izin verir. Bu kısma, sunucumuzun ihtiyaç duyduğu verileri ManyChat değişkenleriyle yazacağız:

```
{
  "subscriber_id": "{{contact.id}}",
  "message": "{{last_text_input}}"
}
```

Burada `{{contact.id}}` ManyChat'te mevcut kullanıcının kimliğini veren dinamik değişkendir. `{{last_text_input}}` ise **kullanıcının son mesajının metnini** içeren hazır bir değişkendir [7]. Bu sayede kullanıcının DM içeriğini JSON içinde `message` alanına koyuyoruz. (ManyChat

dokümanlarında, Default Reply tetiklendiğinde kullanıcının girdiği metnin **Last Text Input** özel alanına otomatik olarak atandığı belirtilir [7]. Biz de bunu kullanıyoruz.)

> **Not:** Eğer kullanıcı yalnızca resim/emoji gönderdiyse, `last_text_input` boş olabilir. Bu durumda botun vereceği yanıtı tasarlamak zordur – basitçe resim hakkında genel bir yanıt verebilir veya "Görsel için teşekkürler!" gibi bir ifade kullanılabilir. Bu ayrık durumları isterseniz ileride ele alabilirsiniz.

- **JSON Encode Ayarı:** ManyChat arayüzünde JSON body girerken "Encode as JSON" gibi bir seçenek olabilir. Gönderdiğiniz gövde halihazırda JSON formatında olduğu için, ManyChat'te bu seçeneği **devre dışı bırakın** (aksi halde çift tırnaklar escape edilebilir). "Invalid JSON" hatası alırsanız, JSON formatını doğru yazdığınızdan ve encode kutusunun uygun şekilde ayarlandığından emin olun [8].

- **Response Mapping:** Bu kısımda herhangi bir mapping yapmayacağız. Normalde ManyChat, harici isteğe gelen cevabı bekleyip özel alanlara kaydedebiliyor. Ancak bizim tasarımımızda, backend hemen bir cevap döndürmeyecek (sadece `200 OK` ile alındı bilgisini verecek) ve yanıtı daha sonra ManyChat API ile göndereceğiz. Bu nedenle Response kısmını boş bırakıyoruz. ManyChat isteğimizi yaptıktan sonra akışı devam ettirmeyebilir veya biz buraya bir *"Geçici yanıt"* koymadıkça kullanıcı hemen bir mesaj görmez. (İsterseniz **Fallback** ayarlayabilirsiniz, aşağıya bakın.)

- **Fallback (Hata Durumu Mesajı):** ManyChat External Request bloğunda, sunucu hatası durumunda kullanıcıya gösterilecek bir yedek mesaj tanımlama imkanı vardır. Bunu kullanmanızı öneririm [9]. Örneğin, "⚠ Sistem şu an meşgul, lütfen daha sonra tekrar deneyin." gibi bir mesajı fallback olarak ekleyebilirsiniz. Bu, sunucu yanıt vermez veya hata dönerse, sohbetin akışının tamamen kesilmesini engeller [9].

- **Default Reply Aktivasyonu:** External Request bloğunu ekleyip ayarları yaptıktan sonra akışı **Yayınla (Publish)** edin. Ardından Default Reply ayarlarında bu oluşturduğunuz akışı Instagram kanalı için etkinleştirin [10]. ManyChat'in Default Reply ayar panelinde tetikleme sıklığını "Her mesajda" olarak bırakın (IG için 24 saatte bir opsiyonu da var, ancak burada her DM'de tetiklenmesini istiyoruz) [11]. Son olarak Default Reply özelliğini aktif hale getirin (toggle) ve **Set Live** ile yayına alın [10].

Bu noktada ManyChat tarafı, kullanıcıdan yeni bir DM mesajı geldiğinde otomatik olarak bizim belirlediğimiz webhook URL'sine JSON veriyi iletecek şekilde ayarlanmıştır. External Request adımından sonra biz ManyChat'te kullanıcıya anında bir şey göndermedik (örneğin "mesajınız alındı, yazıyorum..." gibi). Dilerseniz, kullanıcı deneyimi için, ManyChat'in "Typing..." (yazıyor) aksiyonunu kullandırabilirsiniz; ancak bu advanced bir detay ve zorunlu değil. Genellikle 1-3 saniye içinde yanıt geleceği için sorun olmaz.

3. OpenAI GPT-4 Entegrasyonu ile Cevap Üretimi

ManyChat'ten gelen kullanıcı mesajını alan sunucumuz, şimdi OpenAI'nin Chat Completion API'sini kullanarak uygun bir yanıt üretecek. GPT-4 modelini kullanacağız (daha iyi dil anlama ve uzun yanıt kabiliyeti için). Burada dikkat edilecek husus, **sistem mesajını (system prompt)** iyi tasarlamaktır. Bu prompt, AI asistanının kişiliğini, dilini ve bilgilerini şekillendirecek.

Adım 3.1: OpenAI API Anahtarı ve İstemcisi – OpenAI hesabınızdan bir API Key alın (OpenAI dashboard'daki **API Keys** bölümünden). Bunu Railway environment'ına koyacağız. Backend kodunda bu anahtarı kullanarak OpenAI'nin **ChatCompletion** API çağrılarını yapacağız. Node.js için resmi OpenAI NPM paketini kullanabiliriz veya `fetch/axios` ile kendimiz REST isteği de atabiliriz. Aşağıda resmi OpenAI Node.js kütüphanesi ile örnek kod verilmiştir:

```
import { Configuration, OpenAIApi } from 'openai';

// ... (Express app setup ve webhook route içinde)
const configuration = new Configuration({
    apiKey: process.env.OPENAI_API_KEY,  // .env içinde ayarlı olacak
});
const openai = new OpenAIApi(configuration);

// webhook route içinde, ManyChat verisini aldıktan sonra:
try {
    const systemPrompt =
generateSystemPrompt();  // Aşağıda tanımlayacağımız fonksiyon
    const openaiRes = await openai.createChatCompletion({
        model: "gpt-4",
        messages: [
            { role: "system", content: systemPrompt },
            { role: "user", content: userMessage }
        ],
        temperature: 0.7
    });
    const replyText = openaiRes.data.choices[0].message?.content;
    console.log("OpenAI yanıtı:", replyText);
    // (Sonraki adımda replyText ManyChat API ile gönderilecek)
} catch (err) {
    console.error("OpenAI API Hatası:", err);
    // Hata durumunda fallback olarak ManyChat'e bilgilendirme
gönderebilirsiniz (örn. sistem meşgul).
}
```

Bu kod parçası, webhook isteği içinden kullanıcı mesajını aldıktan sonra çalıştırılacaktır: - `generateSystemPrompt()` fonksiyonu, iş koluna özel bilgileri içeren bir sistem prompt metni döndürecek (aşağıda açıklanacak). - `openai.createChatCompletion({...})` ile GPT-4 modeline bir sohbet isteği gönderiyoruz. Mesaj listesinde ilk mesaj `role: "system"` olarak bizim yönergelerimizi içeriyor, ikinci mesaj `role: "user"` olarak kullanıcının DM içeriğini içeriyor. Bu şekilde model, bizim verdiğimiz kişilik/bağlam içinde, kullanıcının sorusuna cevap üretecek. - `temperature: 0.7` ile bir miktar yaratıcılık sağladık (0 ile 1 arası). Daha tutarlı ama monoton cevaplar istenirse 0.5 altına çekilebilir. - API yanıtından `replyText` içerik kısmını aldık. (GPT-4 genellikle tek bir sonuç döndürür. `choices[0]` bu nedenle yeterli.)

Adım 3.2: Sistem Promptunun Tasarlanması (İşletmeye Özel) – Sistem promptu, AI'ın nasıl davranacağını ve işletme hakkında hangi bilgilere sahip olduğunu tanımlar. Bu promptu *Azerice* yazmak mümkün (hedef kitle Azerice iletişim kuracaksa). **Her işletme için ayrı bir prompt hazırlanmalıdır.** Örneğin, Kung Fu eğitmeni için şöyle bir sistem mesajı yazabiliriz:

```
“Sən peşəkar bir Kung Fu təlimçisinin (Mehman müəllim kimi) süni intellekt köməkçisisən. İstifadəçilərə Instagram üzərindən onların suallarına nəzakətli və məlumatlı şəkildə, Azərbaycan dilində cavab verirsən.
- Cavablarında hər zaman dostcanlı bir üslub saxla, ama eyni zamanda peşəkar ol.
- İstifadəçinin sualına dəqiq cavab ver, bilmədiyin mövzu olarsa uydurma, bunun əvəzinə üzrxahlıq edib daha sonra kömək təklif et.
- Əgər sual təlim saatları, məkan və ya qiymətlərlə bağlıdırsa, [işletme adı]’ın mövcud məlumatlarına əsasən cavab ver. Məsələn, dərslər həftə içi hər gün 18:00-20:00 arası keçirilir, yer Gəncə Olimpiya İdman Kompleksidir.
- İstifadəçi link göndəribsə, qeyd et ki, mən linkləri açа bilmirəm, amma ümumi fikir vermək üçün mənə təsvir edə bilər.”
```

Yukarıdaki gibi kapsamlı bir sistem promptu, AI’ın rolünü ve sınırlarını netleştirir. İşletmenin çalışma saatleri, hizmetleri, üslubu gibi detaylar burada belirtilebilir. GPT-4 modeli bu talimatları dikkate alarak kullanıcının mesajını yanıtlayacaktır. Sistem mesajını dilersek İngilizce veya Türkçe de yazabiliriz; ancak son kullanıcı cevabı Azerice bekliyorsa, promptu Azerice vermek dilin tutarlılığı açısından faydalıdır.

> **Önemli:** Her işletmenin amacı ve verdiği hizmet farklı olacağından, sistem promptunu o işletmeye göre özelleştirmek gerekir. Bu rehberin ilerleyen kısmında, **Admin Paneli** üzerinden bir *Project Brief* doldurarak otomatik sistem promptu oluşturma yönteminden bahsedeceğiz. Oradaki sorular (işletme adı, hizmetler, hedef kitle, vs.) cevaplanınca, bu bilgilerle dolu bir prompt metni hazırlanacak. Bu sayede sistem her sektöre uygulanabilir olacak. Şu an Kung Fu eğitmeni örneği üzerinden gidiyoruz.

Adım 3.3: OpenAI Cevabının Hazırlanması – API’den gelen `replyText`, kullanıcının sorusuna verilmiş cevaptır. Bu metin bazen sonlandırma veya format sorunları içerebilir (cümlenin yarıda kalması gibi). Böyle durumlarda modelin token sınırına takıldığı anlaşılır; çözüm olarak `max_tokens` değerini artırmak gerekebilir. Biz GPT-4 ile genelde makul uzunlukta cevaplar alacağız.

- **Link Yanıtı:** Eğer kullanıcı bir link sormuş ya da paylaşmışsa ve model bunu algıladıysa, sistem promptunda belirttiğimiz gibi “linkleri görüntüleyemiyorum” gibi bir ifadeyle yanıt verecektir. Bu davranışı sistem promptunda yönlendirdik.
- **Çok Dilli Destek:** Kullanıcı Azerice yazdıysa, model Azerice cevap verecektir (promptu da Azerice verdiğimiz için). Farklı dillerde sorular gelirse, istenen bir dil politikası yoksa model aynı dilde cevaplayabilir. Dilerseniz sistem promptunda "İstifadəçi hansı dildə yazırsa, o dildə cavab ver" gibi bir kural ekleyebilirsiniz.

Şimdi elimizde `replyText` (ve belki `linkUrl` de tutuluyor ayrı değişkende ama genelde replyText içine dahil ettirebiliriz) var. Sırada bu cevabı ManyChat aracılığıyla ilgili Instagram kullanıcısına göndermek var.

4. ManyChat API kullanarak Kullanıcıya Yanıt Gönderme

ManyChat’ten mesaj alırken **webhook** kullandık; şimdi yanıtı iletmek için ManyChat’in **Public API** metodlarını kullanacağız. ManyChat Public API, harici uygulamaların ManyChat üzerinden mesaj göndermesine, etiket eklemesine, alan set etmesine vb. olanak tanır [12]. Biz burada iki adımlı bir yöntem kullanacağız:

Neden iki adım? ManyChat API'si doğrudan mesaj göndermeyi (`sendContent`) destekliyor olsa da, bunu kullanmak için mesaj içeriğini doğru formatta hazırlamak gerekiyor. Alternatif olarak, ManyChat'te önceden hazırladığımız bir akışı tetikleyebiliriz (`sendFlow`) veya bir özel alanı doldurup ManyChat tarafında otomasyonu bu alana bağlayabiliriz. İkinci yöntem daha derli toplu ve ManyChat içinde içerik düzenlemeye imkan verir. Ayrıca, Whatsapp gibi bazı kanallarda doğrudan `sendContent` yerine `sendFlow` kullanmak gerekebiliyor [13]. Bu nedenle biz şu yolu seçiyoruz:

- **Adım 4.1:** ManyChat'te bir **Custom User Field (Özel Kullanıcı Alanı)** oluşturacağız, örneğin adı `AI_Response` olsun. Bu alan, AI tarafından üretilen yanıt metnini tutacak.
- **Adım 4.2:** ManyChat'te bir **Flow (Akış)** oluşturacağız (örneğin "Send AI Response" adıyla). Bu akış, sadece bir mesaj bloğu içerecek. Mesaj içeriği olarak, az önce oluşturduğumuz `{{AI_Response}}` özel alanını koyacağız. Yani bu akış tetiklendiğinde, kullanıcıya bu custom field'de ne yazıyorsa gönderilecek.
- **Adım 4.3:** Backend sunucumuz, OpenAI cevabı geldiğinde:
- ManyChat **Set Custom Field** API'sini çağırarak ilgili kullanıcının `AI_Response` alanına cevap metnini yazacak.
- Ardından ManyChat **Send Flow** API'sini çağırarak, ilgili kullanıcı için "Send AI Response" akışını başlatacak. Bu akış ManyChat tarafında mesajı iletecek.

Bu yöntem, yanıtın içeriğini ManyChat içinde de görebilmemizi sağlar (örneğin field loglarında) ve eğer istenirse ManyChat'te bu mesaja buton, görsel vb. eklemeyi de ileride mümkün kılar.

Adım 4.1: ManyChat Custom Field Oluşturma – ManyChat panelinde **Settings > Fields > + New User Field** adımlarını izleyerek yeni bir **Kullanıcı Alanı** oluşturun [14]. Örneğin: - İsim: **AI_Response** (boşluksuz ve anlaşılır bir isim seçin, çünkü API çağrısında ismi kullanacağız) - Tür: Text (metin tutacağız) - Varsayılan Değer: Boş bırakılabilir.

Alanı oluşturduktan sonra, ManyChat arayüzü bu alanı bir ID ile de ilişkilendirir (her özel alanın arka planda bir ID'si vardır). Ancak ManyChat API, kolaylık olması için alan ismiyle de set edebilmemize izin verir [15]. Yine de, **Flow tetikleme** kısmında biz flow'u ID (aslında `flow_ns`) ile çağıracağımızdan, bu alan ismini kodda veya environment'ta saklamak iyi olur.

Adım 4.2: ManyChat Flow (Yanıt Akışı) Oluşturma – ManyChat'te bir yeni akış yaratın (Ana menüde **Flows** bölümünde +New Flow). Bu akışın amacını "AI cevabını kullanıcıya iletmek" olarak belirledik. - Akışa bir isim verin: örneğin **Send AI Response**. - İçerik: Bir **Message** bloğu ekleyin. Mesaj metni kısmına aynen `{{AI_Response}}` yazın. Bu şekilde, akış çalıştığında o anda kullanıcıya ait AI_Response alanında ne değer varsa onu mesaj olarak gönderecek. (Dilerseniz mesajın başına sabit bir ifade de koyabilirsiniz, örn: " ${AI_Response}" gibi, ama gerekmeyebilir.) - Bu akışta sadece bu mesaj bloğu olması yeterli (ek bir tetikleyiciye gerek yok, çünkü biz API ile başlatacağız). - Akışı **Publish** edin ve hazır hale getirin.

Artık ManyChat tarafında: - Kullanıcıya gönderilecek cevap için bir alan var (AI_Response). - Bu alanı mesaj olarak yollayan bir akış var (Send AI Response).

Adım 4.3: ManyChat API ile Alan Set ve Akış Başlatma – Backend sunucumuzda, OpenAI'den cevap alındıktan sonra aşağıdaki işlemleri yapacağız:

1. **Set Custom Field API çağrısı:** ManyChat API'sinde ilgili uç noktayı (endpoint) kullanarak `subscriber_id` ile belirlenen kullanıcıya ait `AI_Response` alanına cevabımızı yazacağız. ManyChat API dokümanına göre istek formatı şöyle olmalıdır:

2. URL: `https://api.manychat.com/fb/subscriber/setCustomField`
3. Metot: POST
4. Header: `Authorization: Bearer <ManyChat_API_Key>` (ManyChat API anahtarımızı Authorization header'ında taşımalıyız) [16].
5. Body (JSON):

```
{
  "subscriber_id": "1234567890",
  "field_name": "AI_Response",
  "field_value": "Merhaba! Bugün size nasıl yardımcı ola bilərəm?"
}
```

Burada `subscriber_id`, ManyChat'ten webhook ile aldığımız kullanıcı kimliğidir; `field_name` az önce oluşturduğumuz alanın adıdır; `field_value` ise AI'in ürettiği cevap metnidir. ManyChat API bu isteği aldığında ilgili kullanıcının alanını güncelleyecektir [15]. (Not: Alternatif olarak `setCustomFieldByName` veya ID ile `setCustomField` kullanımları da olabilir [17], ama en kolayı name kullanmaktır.)

Express ile bunu yapmak için ya `axios` gibi bir HTTP istemcisi kullanabiliriz ya da Node'un `fetch` fonksiyonu (Node 18+ sürümlerinde mevcut) kullanılabilir. Örnek bir axios çağrısı:

```
import axios from 'axios';
// ...
await axios.post("https://api.manychat.com/fb/subscriber/setCustomField", {
    subscriber_id: subscriberId,
    field_name: "AI_Response",
    field_value: replyText
}, {
    headers: {
        'Authorization': `Bearer ${process.env.MANYCHAT_API_KEY}`,
        'Content-Type': 'application/json'
    }
});
console.log("AI_Response field updated.");
```

ManyChat API anahtarımızı `.env` içinde `MANYCHAT_API_KEY` olarak sakladığımızı varsayıyoruz. Bu çağrı sonucunda başarılı bir `{ "status": "success" }` cevabı alınması beklenir.

1. **Send Flow API çağrısı:** Şimdi ManyChat'e, belirli bir kullanıcı için belirli bir akışı başlatmasını söyleyeceğiz. Bunun için ManyChat API'de `sendFlow` adı verilen uç nokta var:
2. URL: `https://api.manychat.com/fb/sending/sendFlow`
3. Metot: POST
4. Header: `Authorization: Bearer <ManyChat_API_Key>` (yine API key'imiz gerekli).
5. Body (JSON):

```
{
  "subscriber_id": "1234567890",
```

```
  "flow_ns": "content20231130123456_987654"
}
```

Bu yapı, ManyChat'e "şu ID'li kullanıcı için, flow_ns değeri şu olan akışı çalıştır" demektir. Peki `flow_ns` nedir? ManyChat'te her oluşturduğumuz flow'un bir kimliği (namespace) vardır. Bunu bulmak için ilgili akışı ManyChat Flow Builder'da açın ve tarayıcı adres çubuğuna bakın. URL'nin sonunda `content...` ile başlayan bir ibare göreceksiniz (örn: `content20231130123456_987654` gibi) [18]. Bu dize, o akışın benzersiz kimliğidir. Kopyalayıp kenara not edin. (Eğer URL'de görünmüyorsa, akışı paylaşma veya klonlama seçeneklerinde bu kod geçebilir. ManyChat community'de de bu kodun nasıl alınacağı sorulmuştu; en kolay yolu URL'den bakmaktır [19].)

`subscriber_id` yine ManyChat'ten gelen kullanıcı kimliği olacak. Bu verileri JSON'da belirttiğimiz gibi gönderiyoruz. Örnek axios ile gönderim:

```
await axios.post("https://api.manychat.com/fb/sending/sendFlow", {
    subscriber_id: subscriberId,
    flow_ns: process.env.MANYCHAT_FLOW_NS
}, {
    headers: { 'Authorization': `Bearer ${process.env.MANYCHAT_API_KEY}` }
});
console.log("SendFlow triggered.");
```

Burada akışın kimliğini `.env` içine önceden `MANYCHAT_FLOW_NS` olarak koyduğumuzu varsayıyoruz (Railway'de ekleyeceğiz). Bu çağrı başarılı olursa ManyChat API `{ "status": "success" }` ve belki bir `message_id` döndürür. Artık ManyChat tarafında, o kullanıcı için bizim "Send AI Response" flow'umuz tetiklenmiş olur – yani kullanıcı anında bot cevabını alır.

Bu iki adımı art arda yaparak cevabı iletmiş oluyoruz. Timing açısından: - `setCustomField` isteği tipik olarak çok hızlı (<200ms). - `sendFlow` akışı da hızlı tetiklenir. ManyChat genelde anında mesaj gönderir. Arada birkaç yüz ms gecikme olabilir.

API Çağrılarında Dikkat: ManyChat Public API'sinin hız limitleri vardır. Örneğin `setCustomField` ve `sendFlow` gibi çağrılar için saniyede 10 istek (RPS) sınırı olabilir [20]. Bizim kullanımımızda saniyede 10'dan fazla DM gelmesi pek muhtemel değil, ancak yüksek trafik bekleniyorsa istekleri **queued (sıralı)** yapmak gerekebilir. Şimdilik bu endişe verici değil.

Ayrıca ManyChat API çağrılarını ardışık yaptığımızdan emin olalım: Field set işlemi tamamlanmadan flow'u başlatırsak, flow mesajı gönderdiğinde alan henüz güncellenmemiş olabilir. Bu yüzden kodda önce `await setCustomField` sonra `await sendFlow` yapıyoruz ki sıralı ilerlesin.

Adım 4.4: ManyChat Alan ve Akış ID'lerinin Doğru Kullanımı – Summarize olarak, şunları environment'a koyacağız veya kodda sabit tanımlayacağız: - **MANYCHAT_API_KEY:** ManyChat panelinden aldığımız API token (başında **Bearer** kelimesini header'da kullanacağız). - **MANYCHAT_FLOW_NS:** "Send AI Response" akışının flow kimliği (content... ile başlayan kod). (Not: ManyChat bazı API'lerinde numeric flow id de kullanıyor olabilir ancak burada gördüğümüz string namespace'i kullanıyoruz [21].) - **AI_FIELD_NAME:** Özel alanın adı (`AI_Response`). Bunu isterseniz sabit de yazabilirsiniz, değişmeyecek bir string. Ama konfigürasyona koymak ileride farklı isim kullanmak

istersek esneklik sağlar. - **SUBSCRIBER_ID:** Bu environment'ta değil, her çağrıda ManyChat'ten geliyor. Sadece hatırlatma için belirttik.

Artık backend tarafında, ManyChat'ten mesaj alıp AI ile cevap üretip ManyChat üzerinden geri gönderecek tüm parçaları tamamladık. Şimdi bu sistemin kurulum ve yapılandırma detaylarını (Railway üzerinde çalıştırma, environment ayarları, vs.) ele alalım.

5. Railway Üzerinde Dağıtım ve Environment Değişkenlerinin Ayarlanması

Projemizi çalışır hale getirdikten sonra, bunu **Railway** platformuna deploy edeceğiz. Railway, tıpkı Heroku gibi, GitHub reposundan uygulamayı alıp konteyner içinde çalıştıran bir PaaS platformudur.

Adım 5.1: Railway Projesini Oluşturma – Öncelikle railway.app üzerinden bir hesap oluşturun ve yeni bir proje başlatın. Projeyi GitHub deposuna bağlayabilirsiniz. Railway, Node.js projesini algılayıp otomatik olarak uygun şekilde çalıştıracaktır (genellikle `package.json` içindeki `start` script'ine bakar). Eğer `start` script'iniz yoksa ekleyin (ör: `"start": "node dist/index.js"` ya da TS kullanıyorsanız build süreci de ekleyin).

Adım 5.2: Port Ayarı – Railway, container içinde genelde bir port atar (örn. 5163 gibi rastgele) ve bunu `PORT` environment değişkeniyle iletir. Express uygulamanızda `app.listen(process.env.PORT || 3000)` şeklinde dinlediğinizden emin olun ki Railway üzerinde doğru portta çalışsın. (Lokal geliştirirken 3000 portunu kullanır, production'da Railway'in verdiği PORT'u kullanır.)

Adım 5.3: Environment Değişkenlerini Tanımlama – Railway projesinde, ManyChat ve OpenAI anahtarları ile flow/field ID'lerini gizli olarak saklamalıyız. Railway arayüzünde projenizi seçip **Variables** sekmesine gidin. Burada tek tek değişken ekleyebilir veya toplu eklemek için "Raw Editor"u kullanabilirsiniz [22]. Eklememiz gereken değişkenler şunlardır: - `OPENAI_API_KEY` – (OpenAI hesabınızdan aldığınız secret key, `sk-...` ile başlayan) - `MANYCHAT_API_KEY` – (ManyChat panelinden aldığınız API token; bunu Railway'e girerken başına `Bearer` eklemenize gerek yok, kodda ekleyeceğiz) - `MANYCHAT_FLOW_NS` – (ManyChat "Send AI Response" flow'sunun kimliği, örn: `content20231130123456_987654`) - `AI_FIELD_NAME` – (Özel alan adı, örn: `AI_Response`. Bu isteğe bağlı, isterseniz kodda sabit de kullanabilirsiniz.) - (Opsiyonel) `ADMIN_PASSWORD` – (Admin paneli için bir şifre belirleyecekseniz, onu da burada tutabilirsiniz. Aşağıda Admin paneli bölümünde değineceğiz.)

Değişkenleri ekledikten sonra **Deploy/Apply** yapmayı unutmayın – Railway, env değişikliklerini uygulamak için genelde yeniden dağıtım yapar. Artık sunucumuzun ihtiyaç duyduğu gizli anahtarlar ve ID'ler tanımlandı.

Adım 5.4: Domain ve Webhook URL'si – Railway projeniz uygulamayı çalıştırdıktan sonra size otomatik bir alt alan adı verecektir (örneğin `mybot-production.up.railway.app`). Bunu proje panelindeki **Settings** altında veya direkt loglarda görebilirsiniz. Bu alan adını kullanarak ManyChat'teki External Request URL'sini güncelleyin. Örneğin, External Request URL'sini `https://mybot-production.up.railway.app/webhook` olarak ayarlamalısınız (HTTP yerine HTTPS kullanmayı unutmayın). ManyChat, güvenlik gereği HTTP yerine HTTPS isteyecektir.

Artık ManyChat, Instagram'dan bir mesaj geldiğinde bu Railway domainine isteği yapacak ve backend fonksiyonlarımız çalışacaktır.

Adım 5.5: Netlify (Opsiyonel Frontend) Deploy – Eğer Admin Paneli arayüzü hazırladıysanız, bunu ayrı olarak Netlify gibi bir platformda yayınlayabilirsiniz. Netlify hesabınızı oluşturup GitHub reposunuza bağlayın. Admin paneli React uygulaması için bir build komutu tanımlayın (`npm run build`) ve Netlify, build çıktısını otomatik host edecektir. Admin paneli yalnızca test ve konfigurasyon amaçlı olduğundan, URL'sini gizli tutmak isteyebilirsiniz (Netlify Pro planlarında parola koruması var, yoksa basit bir login zaten koyacağız). **Frontend ile backend'in haberleşmesi:** Admin panelini host ettiğiniz domainin, Railway backend API'sine CORS isteği yapabilmesi için backend kodunda CORS izni verin (ör. `app.use(cors({ origin: 'https://adminpanel.netlify.app' }))`). Ya da geliştirirken tüm origin'lere izin verip sonrasında kısıtlarsınız.

6. Backend'in Production-Ready Hale Getirilmesi (Hata Yönetimi, Loglama)

Sistemimizin omurgası kurulmuş durumda. Şimdi geliştirme bittikten sonra sorunsuz bir production ortamı için dikkat edilmesi gereken noktalara bakalım:

- **Hata Yönetimi:** Express sunucusunda, özellikle OpenAI API çağrısı uzun sürebilir veya hata dönebilir. Bu durumda try/catch ile hataları yakaladık (yukarıdaki kodda örnekledik). Yakalanan hatalarda `res.sendStatus(500)` gibi bir yanıt ManyChat'e dönersek, ManyChat bunu bir başarısızlık olarak görecek ve eğer fallback içeriğiniz varsa onu kullanıcıya gösterecektir [9]. Dolayısıyla fallback mesajı koymanız önemliydi. Ayrıca hatayı Railway loglarında görebiliriz. Mümkünse hatanın sebebini (OpenAI timeout mu, ManyChat API error mu vs.) loglamaya özen gösterin. ManyChat'in kendi Logs bölümünde de başarısız harici istekler görülebilir (ManyChat panelinde **Settings > Logs** kısmı).
- **Zaman Aşımı:** ManyChat External Request için belli bir timeout uygulayabilir (genellikle 5-10 saniye). GPT-4 cevabı bazen kompleks sorularda 5+ saniye sürebilir. Bizim yaklaşımımızda ManyChat bu cevabı beklemiyor, hemen 200 OK alıyor. Yani bu sorunu aştık. Ancak, biz OpenAI cevabını alana dek bekliyoruz ve belki 6-7 saniye sonra ManyChat'e cevap gönderiyoruz. Kullanıcı bu arada biraz beklemiş olacak. Bu bekleme normaldir, GPT-4'ün hızına bağlıdır. Aşırı gecikmeler yaşanmaması için gerekirse GPT-3.5 modelini kullanmayı da bir opsiyon olarak tutabilirsiniz (daha hızlı cevaplar için).
- **Loglama:** Railway loglarından tüm `console.log` çıktılarınızı görebilirsiniz. Prod ortamda debug için yeterli olacaktır. Daha profesyonel bir yaklaşım için `winston` gibi log kütüphaneleri entegre edilebilir ve hata seviyelerine göre loglama yapılabilir. Ancak ölçek küçükse console.log yeterlidir.
- **Güvenlik:** ManyChat webhook endpoint'imiz genel bir URL. Kötü niyetli kişiler bu URL'yi bilirlerse sahte istek atabilirler. Bunu engellemek için en basit yöntem, ManyChat External Request body'sine bir **secret token** eklemek ve sunucu tarafında bu token'ı doğrulamaktır. Örneğin ManyChat'te body'ye `"token": "<pre-shared-secret>"` ekler ve sunucu `if (body.token !== process.env.MY_SECRET) { return res.sendStatus(403) }` yapabilir. Bu secret'ı sadece ManyChat ayarına siz girdiğiniz için başkası bilmez. Bu yöntem opsiyoneldir ama önerilir.
- **Çoklu Kullanıcı ve Durum**: Bu sistem kullanıcı mesajlarını tek tek işler, hiçbir kullanıcı bilgisini veritabanında tutmaz. Yani tamamen stateless bir servistir. Bu, basitlik açısından iyidir; ancak **çok adımlı diyalog** gereken durumlarda (örneğin AI'nin bir soru sorup kullanıcıdan yanıt alması vb.) zorluk çıkarabilir. ManyChat aslında akışlar üzerinden çok adımlı süreçleri yönetebilir (ör. bir randevu takvimi sormak gibi). Bu entegrasyon, gelen soruya anlık yanıt vermek içindir. Daha karmaşık diyaloglar gerekecekse, belki bazı özel anahtar kelimeleri ManyChat kendi içinde yönlendirmeli ya da AI modeline tüm sohbet geçmişini de iletmeli (o durumda geçmişi bir

veritabanında tutmak gerekebilirdi). Projemizde böyle bir ihtiyaç olmadığı belirtilmiştir (her mesaj bağımsız cevaplanacak).

- **Ölçeklenebilirlik:** Kullanıcı sayısı veya mesaj trafiği artarsa, Railway içinde birden fazla instance çalıştırmak (horizontal scaling) gündeme gelebilir. Ancak ManyChat tarafı her bir mesajı bir kez gönderir, biz de tek bir instance'da işliyoruz. İşlem süresi büyük oranda OpenAI çağrısına bağlı. GPT-4 API'nin de çağrı limiti var (örn. dakikada X istek). Trafik beklentinize göre OpenAI API kullanım limitlerini gözden geçirin. Gerekirse önbellekleme, veya bazı sık sorulan sorulara statik yanıt verme gibi optimizasyonlar düşünebilirsiniz.
- **Test (Manuel):** Sistemi canlıya almadan önce Postman gibi bir araçla endpoint'i test edebilirsiniz. Örneğin ManyChat'in gönderdiğine benzer bir POST isteği atıp (`subscriber_id` : kendi test bir id, `message` : "Merhaba" gibi) sunucunuzun OpenAI'den mantıklı bir cevap dönüp ManyChat API'lerine başarılı istek yapıp yapmadığını görebilirsiniz. ManyChat API'sine istek yaptığımız kısımda gerçek bir subscriber_id kullanmak gerekiyor. Bunu test etmenin bir yolu, kendi Instagram hesabınızdan botunuza DM atıp ManyChat panelinde sizin kullanıcı ID'nizi almaktır (ManyChat > Contacts kısmında kullanıcıyı seçip **System Fields** içinde User ID'yi görebilirsiniz). Bu ID'yi test verisi olarak kullanıp, akışı gerçekten tetikleyip tetiklemediğini görebilirsiniz. (Dikkat: Test sırasında, sendFlow çağrısı tetiklenirse gerçek Instagram hesabınıza mesaj gelebilir, bu beklenen bir durum.)
- **Salesforce veya Database Gerekli mi?** Kullanıcı belirtmiş, "veri tabanı gerekmez, çok kullanıcı olsaydı belki Salesforce düşünürdük ama şu an lüzum yok" diye. Gerçekten de bu sistemde sürekli değişken veri saklanmadığı için (sadece environment konfigürasyonları var), klasik bir database'e ihtiyaç yok. ManyChat kendi içinde kullanıcı yönetimini yapıyor zaten (kullanıcı ID, ad, vs. orada mevcut). Eğer daha sonra sohbet geçmişi kaydı, analitik vs. istenirse, MongoDB gibi bir veritabanı eklenip her Q/A çifti kaydedilebilir, ama bu isteğe bağlı bir geliştirme olacaktır.

7. Admin Paneli (Opsiyonel) – Test ve Prompt Konfigürasyon Arayüzü

Projenin opsiyonel bir parçası, **yalnızca yöneticinin erişebileceği** bir web panelidir. Bu panelin iki temel amacı vardır: 1. **Project Brief / Sistem Promptu Hazırlama:** Her bir işletme için AI sistem promptunu tam isabetli oluşturabilmek adına, ilgili işletme hakkında soruların sorulup cevaplarının alınacağı bir arayüz. 2. **Mesaj Testi:** ManyChat/Instagram ortamını kullanmadan, doğrudan panel üzerinden bir kullanıcı mesajı simüle edip AI cevabını görme (ve gerekirse promptu iyileştirme) imkanı.

Bu panel güvenlik nedeniyle şifre korumalı olmalıdır (dışarıya açık bir URL'de host edeceğimiz için). Yönetici haricinde kimse erişememelidir.

Admin Panel Özellikleri:

- **Giriş (Login) Ekranı:** Paneli açan kullanıcıdan bir parola isteyen basit bir login sayfası olacak. Örneğin tek bir sabit parola belirleyip (environment değişkeni `ADMIN_PASSWORD` olarak tutuluyor), admin arayüzüne girmek isteyen kullanıcının bunu girmesi gerekir. Başarılı giriş sonrası bir **JSON Web Token (JWT)** oluşturup frontende kaydedebilir veya daha basitçe frontende bir flag atayabiliriz. Güvenlik süper kritik değil, temel amaç arayüzü meraklı gözlerden korumak. (Netlify gibi platformların kendi HTTP auth özellikleri de kullanılabilir.)
- **Project Brief Formu:** Giriş yapıldıktan sonra, ana ekranda işletmeye dair soruların yer aldığı bir form olacak. Bu form adeta bir anket gibi, işletmenin profilini çıkarmaya yönelik soruları içerir. Amaç, AI sistem promptunu bu yanıtlara göre otomatik oluşturmaktır. Sorular olabildiğince kapsamlı ve genel tutulmalıdır ki farklı sektörler için kullanılabilsin:

- **İşletmenin Adı ve Kısa Tanımı:** (Örn: “Mehman’s Kung Fu Academy – Gəncə’də kişisel Kung Fu dersleri veren bir eğitmen.”)
- **Hizmet veya Ürünler:** (Örn: “Bireysel Kung Fu dersleri, Grup seminerleri, Online danışmanlık.”) Mümkünse ayrı ayrı girebilecekleri bir liste olarak tasarlanabilir.
- **Hedef Kitle:** (Örn: “18-35 yaş arası, spora ilgi duyan genç yetişkinler.”)
- **İşletmenin Misyonu / Değerleri:** (Örn: “Kung Fu felsefesini yaymak ve öğrencilerin özgüvenini artırmak.”)
- **Çalışma Saatleri:** (Örn: “Hafta içi her gün 18:00-20:00 arası.”)
- **Lokasyon(lar):** (Örn: “Gəncə Olimpiya İdman Kompleksi” veya birden çok şubesi varsa liste.)
- **İletişim Bilgileri:** (Örn: Telefon, e-posta adresi, web sitesi. Bunlar, AI cevabında gerektiğinde kullanıcıya verilebilecek bilgileri içerir. Örn: “Detaylı bilgi için 0123456789 no’lu telefondan ulaşa bilərsiniz.” diye kullandırabiliriz.)
- **Sık Sorulan Sorular & Cevaplar:** Bu opsiyonel ama değerli olabilir. Örneğin işletmeye genelde sorulan 3-5 soruyu ve bunların cevaplarını admin buraya girebilir. AI modeline bu içerikler verilmese bile, belki sistem promptuna “Sık sorulan suallar və cavablar: 1)... 2)...” diye eklenebilir. Bu, GPT’nin bu konularda daha doğru cevaplar vermesini sağlar (RAG – retrieval augmented generation gibi).
- **Tercih Edilen Dil ve Üslup:** (Örn: “Resmi bir dil yerine samimi bir dil kullanılsın”, veya “Cevaplar kısa ve öz olsun” vs.)
- **Olmaması Gereken Yanıtlar:** (Örn: “Kesinlikle rakiplerden bahsetme, fiyat sorulursa yönlendirme yapma” gibi özel istekler.)

  Bu soruların sayısı işletmeye göre artabilir. Kullanıcı “40-50 sorudan bahsetmiş”. Aslında yukarıdaki maddeler alt kırılımlarıyla bu sayılara ulaşabilir. Örneğin Hizmetler bir soru, her hizmetin hedef kitlesi ayrı bir soru olabilir. Burada önemli olan, panelin esnek olması ve soruları genişletebilmeniz. Basit bir çözüm: Soruları sabit metinler olarak kodlamak yerine, bir JSON tanımı ile saklamak (soru listesi) ve formu buna göre oluşturmak. Ancak projenin kapsamı açısından sabit de bırakabilirsiniz ve gerektiğinde kodu güncelleyerek yeni sorular ekleyebilirsiniz.

- **Prompt Oluşturma:** Admin, formdaki tüm soruları doldurup “Oluştur” düğmesine bastığında, panel bu cevapları alıp arka planda bir fonksiyona gönderecek. Bu fonksiyon, önceden hazırlanmış bir **prompt şablonunu** bu cevaplarla dolduracak. Örneğin bir basit şablon:

```
Sən [İŞLETME_ADI] üçün süni intellekt köməkçisisən. [İŞLETME_TANIMI].
İstifadəçilərə [HEDEF_KITLE] üçün [HIZMETLER] barədə suallarında kömək
edirsən.
İş saatları: [SAATLER]. Məkan: [LOKASYON].
...
Üslub: [USLUP].
Sık sorulan suallar: [SSS_LISTESİ].
...
```

Bu şablonu kod içinde tanımlayabilir veya daha esnek hale getirmek için form sorularını bir objeye çevirip string interpolation yapabilirsiniz. Örneğin:

```
const brief = getFormAnswers(); // admin formundan alınan cevap objesi
const prompt =
  `Sən ${brief.businessName} üçün süni intellekt köməkçisisən. $
```

```
{brief.businessDescription} ... ` +
  `İş saatları: ${brief.hours}. Məkan: ${brief.location}. ...`;
```

Bu şekilde bir string üretilir. Ardından: - Bu prompt metnini panelde admin'e gösterin (bir textarea içinde veya ayrı bir bölümde). Admin isterse üzerinde düzeltme yapabilmeli belki. - **Promptu Onaylama/Kaydetme:** Admin memnun ise "Kaydet" butonu ile backend'e bu promptu gönderebilir. Backend bu promptu belki bir global değişkene tutabilir veya environment değişkenine yazması gerekir (ama running environment'ı dinamik değiştirmek zordur, kalıcı olmaz). En pratik çözüm: Promptu bir JSON dosyasına kaydedebiliriz sunucu içinde veya bellek içi bir değişkende tutarız. Database olmadığından, dosyaya yazmak bir seçenektir (Railway'de disk ephemeral olabilir, ancak `mnt/data` kalıcı alan sunuyor). Küçük bir JSON dosyada promptu saklayıp, yeni mesaj geldiğinde oradan okuyabilirsiniz. Yine de bu biraz gereksiz karmaşıklık olabilir.

Alternatif yaklaşım: Bu paneli sadece **geliştirme aşamasında** kullanıp çıkan prompt metnini manuel olarak `.env` veya kod içine koymak. Kullanıcı bunu da yapabilir. Yani panel bir araç görevi görür, esas üretim ortamında sabit prompt kullanılır. Karar sizin; burada tam otomasyonu isterseniz dosya yazma yoluna gidin.

- **Mesaj Testi (Chat arayüzü):** Panelin bir diğer önemli kısmı, oluşturduğumuz promptu test etmektir. Bunun için basit bir arayüz yapabiliriz:
- Bir metin input'u (textarea) ve "Gönder" butonu olsun. Admin buraya herhangi bir kullanıcı sorusu yazıp gönderdiğinde, frontend bu soruyu backend'e gönderir.
- Backend'te özel bir endpoint (ör. `/admin/test`) oluştururuz. Bu endpoint, ManyChat'i **devre dışı bırakıp** doğrudan OpenAI API'ye istek atar ve yanıtı döndürür. (ManyChat'e ihtiyaç yok, sadece GPT ile konuşuyoruz).
- Yani backend, gelen soruyu alır, `generateSystemPrompt()` fonksiyonundan **güncel promptu** çeker (mesela dosyadan veya bellekte tutuyorsanız oradan), OpenAI'den cevabı alır ve döndürür.
- Frontend de bu cevabı ekranda gösterir (basitçe altına bir `<div>` ile).
- Hatta küçük bir güzellik: Bu etkileşimi gerçek bir chat balonu gibi gösterip, çoklu mesaj göndermeyi de sağlayabilirsiniz (her seferinde API'ye gidip almak üzere). Ancak minimum gereklilik tek soru-cevap da olabilir.

Bu test arayüzü sayesinde, Instagram'dan bağımsız hızlı denemeler yapabilirsiniz. Örneğin bir kullanıcı garip bir soru sorduğunda AI'in cevabı çok mu uzun, uygun mu diye bakıp, gerekirse promptu panel üzerinden düzeltip tekrar test edersiniz. Prompt içeriğini mükemmelleştirdikten sonra sistem gerçekten kullanıma alınır.

- **Teknik Uygulama:** Admin panelini React ile yazabilirsiniz. Her soruyu bir `<input>` veya `<textarea>` olarak oluşturup state'de tutar, gönderirken bir JSON oluşturur. `fetch('/api/savePrompt', { method: 'POST', body: JSON.stringify(answers) } )` gibi bir çağrıyla backend'e iletebilirsiniz.

- **Stitch ile Tasarım:** Google'ın **Stitch** aracı, doğal dilde arayüz taslağı tarif ederek otomatik React kodu üretebilen deneysel bir araçtır. Eğer kullanmak isterseniz, şöyle bir prompt ile Stitch'te tasarımı oluşturabilirsiniz: > "Create an admin-only web app for configuring an AI chatbot. It should have a login page that asks for a password. After login, show a multi-section form titled 'Project Brief'. Include fields: Business Name (text), Business Description (multiline text), Services (multiple text entries), Target Audience (text), Business Hours (text), Locations (text), Contact Info (text), FAQs (perhaps a pair of question & answer inputs), Tone (text), etc. Also include a section to test the chatbot: a textarea for user message and a send button, and an area to display the AI's response. The design should be simple and clean."

Bu şekilde Stitch size bir HTML/CSS/React kodu önerebilir. Ancak Stitch henüz mükemmel olmayabilir, çıkan kodu yine elle düzeltmek gerekir. İsterseniz paneli tamamen elle de kodlayabilirsiniz (zaten çok kompleks değildir).

- **Backend Entegrasyonu:** Admin panelinden gelen istekleri karşılamak için Express app'imize birkaç route ekleyelim:

- `/admin/login` – POST, body'de gelen şifreyi kontrol eder. Doğruysa bir JWT token döner. (Veya basit bir boolean döndürüp frontend'de localStorage bir flag koyarsınız – ama JWT daha güvenli olur). JWT kullanırsanız, panelin diğer isteklerinde `Authorization: Bearer <token>` header'ı ile gönderip backend'de bir middleware ile doğrulayın.
- `/admin/savePrompt` – POST, body'de gelen prompt verilerini işler. Bu noktada iki yaklaşım var: (a) direk tam string prompt gönderip kaydetmek, (b) soru-cevap şeklinde gönderip backend'de birleştirmek. Kolaylık açısından backend'de de birleştirme yapabiliriz. Örneğin `generateSystemPrompt(answers)` fonksiyonunu burada çağırıp oluşan prompt metnini bir değişkende tutun (ya global değişken, ya disk).
  - Basit çözüm: Prompt'u global bir değişkende sakla. Örneğin `currentSystemPrompt` adında bir değişken, başta default bir değerle (hardcoded belki) başlar. Admin kaydedince onu günceller. Ve OpenAI çağrısı yaparken hep bu `currentSystemPrompt` kullanılır. Node.js tek process olduğundan bu global değişken paylaşımlıdır. Uygulama restart olana dek kalır. Restart olunca env'de bir default olabilir veya admin panelinden tekrar girilebilir. (Kalıcı yapmak isterseniz dosyaya yazın).
  - Bu yöntemle DB kullanmadan promptu runtime'da değiştirebilir hale geliyoruz.
- `/admin/testPrompt` – POST, body: `{ message: "deneme mesaj" }`. Bu, admin test mesajını içerir. Backend bu mesaja göre:

```
const prompt = currentSystemPrompt;
const openaiRes = await openai.createChatCompletion({
   model: "gpt-4",
   messages: [ {role: "system", content: prompt}, {role: "user",
content: req.body.message} ]
});
res.json({ reply: openaiRes.data.choices[0].message?.content });
```

gibi bir işlem yapıp cevabı dönebilir.

- Tüm bu `/admin/*` route'larına basit bir auth middleware eklemek şart. Örneğin:

```
app.use('/admin', (req, res, next) => {
  const auth = req.header('Authorization');
  if (!auth || auth !== `Bearer ${ADMIN_PASSWORD}`) {
    return res.sendStatus(403);
  }
  next();
});
```

(Çok basit bir yöntem: frontend login sonrası her isteğe header'da şifreyi Bearer token olarak göndersin. Bu tabii düz metin iletmiş oluyor JWT yoksa. Ama zaten HTTPS üzerinde olur ve tek kullanıcı bildiği için sorun çok kritik değil.)

Bu şekilde panelin fonksiyonları sunucuya bağlanmış olacak.

Admin Panelini Deploy Etme: Panel frontendi bir static site olarak Netlify/Vercel'da host edilebilir demiştik. Eğer backend API'sine farklı bir origin'den istek atacaksa, CORS ayarını yapmayı unutmayalım (Express'te `cors` paketini kullanarak). Netlify'de ortam değişkenlerine Admin Panelinin backend URL'sini koymak gerekebilir (örn. `REACT_APP_API_URL` olarak Railway domaini). Bunu `.env.production` dosyasında tutup Netlify'ye verebilirsiniz.

Paneli kullanırken şunu hedefliyoruz: İşletme sahibi veya geliştirici, paneldeki soruları doldurup promptu oluşturacak, birkaç test yapacak. Her şey yolunda ise **currentSystemPrompt** artık hazır. Ardından gerçek Instagram DM üzerinden test ettiğinde de aynı davranışı alacak çünkü backend artık bu prompt ile çalışıyor. Panel olmazsa ne olur? O zaman tüm bu bilgiler kod içine veya .env'ye önceden elle yazılmalıydı. Panel sadece bunu kolaylaştırıyor.

8. ManyChat Üzerinde Alan & Akış ID'lerini Bulma ve Otomasyon Kontrol Listesi

Bu bölümde ManyChat panelindeki kritik ayar adımlarını madde madde özetleyelim ki hiçbir adım atlanmasın:

- **Instagram Bağlantısı:** ManyChat hesabınıza Instagram Business sayfasını bağlayın [1] (Bunu yapmazsanız DM'ler ManyChat'e düşmez).
- **ManyChat API Anahtarı:** Settings > API'den token oluştur, kopyala (Railway `MANYCHAT_API_KEY` olarak ekle).
- **Custom Field Oluştur:** Settings > Fields altında `AI_Response` isimli Text tipi kullanıcı alanı yarat [1] . (ID'sine gerek yok, ismi not edin.)
- **Response Flow Oluştur:** Akışlar bölümünde yeni flow; içine `{{AI_Response}}` içeren mesaj bloğu koy, kaydet. (Opsiyonel: Görsel, buton eklemeyeceğiz, sadece metin.)
- **Flow ID (NS) Alma:** Oluşturduğunuz flow'u editörde açın, URL sonundaki `content...` kodunu kopyalayın. Railway `MANYCHAT_FLOW_NS` değişkenine değer olarak girin [18] .
- **Default Reply Akışı:** Instagram kanalınız için Default Reply'i etkinleştirin. Yeni bir akış oluştur diyerek Flow Builder'a girin. Bu akışa External Request bloğu ekleyin:
- URL: Railway webhook URL'niz (`.../webhook`).
- Method: POST, Body'de JSON (subscriber_id & message alanlarını yukarıda anlatıldığı gibi) [7] .
- Fallback mesajı tanımlayın (sunucu hatasında kullanıcıya gidecek uyarı) [9] .
- Bu akışta External Request'ten sonra başka bir mesaj blok KOYMAYIN (cevabı burada göndermiyoruz, ManyChat API ile ayrı göndereceğiz). Yani External Request tek adım olarak kalabilir.
- Akışı yayınlayın ve Default Reply olarak atayın [10] . Default Reply tetikleme ayarını "her zaman" yapın [11] .
- Default Reply'i aktif hale getirin (toggle on) ve Set Live yapın.
- **ManyChat Rule (Opsiyonel):** Default Reply yerine ManyChat Rules kullanarak da benzer bir şey yapabilirdik: "Incoming Message" tetiklendiğinde webhook çağır vs. Ancak Instagram DM'lerde en garanti yöntem Default Reply'dir çünkü her mesaja çalışır. Yine de, eğer belirli keyword'ler için farklı davranış istiyorsanız, o keyword'ler için ayrı akışlar tanımlayıp, genel fallback için bu entegrasyonu kullanabilirsiniz.

- **Test – ManyChat Paneli:** ManyChat'te **Live Chat** bölümünden veya direkt Instagram uygulamasından botunuza bir test mesajı gönderin. ManyChat'te Contact'lar altında sizin ID'niz görünecektir. Bir mesaj gönderdiğinizde:
- ManyChat Logs'da External Request'in başarılı olduğunu görmelisiniz (ayarlar->logs).
- Railway logs'da "ManyChat Webhook payload" logunu ve ardından "OpenAI yanıtı" ve "field updated / flow triggered" loglarını görmelisiniz (biz kodda console.log koymuştuk).
- Few seconds içinde Instagram'da bottan yanıt gelmiş olmalı.
- Tüm akış doğruysa bu noktada sistem çalışıyor demektir.

Bu kontrol listesini kullanarak, herhangi bir adımda takılırsanız eksik kalan yeri tespit edebilirsiniz. Özellikle ManyChat tarafında flow_ns veya benzeri küçük detaylar gözden kaçarsa entegrasyon olmaz. (Örneğin flow_ns yanlış girilirse sendFlow API çağrısı "flow not found" deyip başarısız olur ama bunu loglamanız gerek, ManyChat API böyle bir durumda hata döner).

9. Canlıya Alındıktan Sonra İzleme ve Destek

Sistem yayına alındığında, belirli aralıklarla düzgün çalıştığını teyit etmek gerekir:

- **Railway Monitoring:** Railway, projenizin temel istatistiklerini sunar (CPU, bellek). Ayrıca logları gerçek zamanlı takip edebilirsiniz. Hataları burada görmek mümkün. Railway'de uptime sorunu yaşamamak için ücretsiz plan sınırlarına dikkat edin (uyku moduna geçme vs. olabilir).
- **ManyChat Logs & Debug:** ManyChat panelinde **Logs** kısmı, tüm başarısız harici istekleri, API çağrılarını gösterir. Burada hata görürseniz üzerine tıklayıp detayını okuyun. Örneğin ManyChat API'den 400 hatası alındıysa, muhtemelen flow_ns veya field_name yanlış girilmiştir. "Invalid flow_ns" gibi bir mesaj olabilir.
- **Rate Limits:** Çok fazla istek gelirse (özellikle OpenAI tarafında) API limitlerine takılabilirsiniz. OpenAI, 429 Too Many Requests hatası dönerse loglarda bunu yakalayın. Gerekirse hızı düşürmek veya OpenAI API throughput'unuzu yükseltmek (paket limitlerini artırmak) gerekebilir.
- **AI Cevap Kalitesi:** Düzenli aralıklarla gelen yanıtları inceleyin. Özellikle ilk dönemlerde, AI'ın istenmeyen bir şey söylemediğinden emin olmak istersiniz. ManyChat'in kendi AI filtresi yok; ancak OpenAI'nin içerik politikaları var. Çok uç bir soru gelirse OpenAI model'i boş cevap verebilir veya bir uyarı iletebilir. Bu durumlarda kullanıcıya özel bir mesaj göstermek isteyebilirsiniz (OpenAI API, `error` döner bu durumda, fallback mesajınız devreye girer).
- **Güncelleme ve Bakım:** İşletmede değişiklik olursa (örneğin çalışma saatleri değişti), admin panelini kullanarak yeni promptu oluşturun ve kaydedin. Eğer Admin paneli kullanmıyorsanız, koddaki promptu güncellemeniz gerekir. Yani sistemi canlı tuttukça içerik güncelliğini sağlamak önemli.
- **Genişletme:** İleride bu sistemi başka platformlara (Messenger, WhatsApp) uyarlamak isterseniz, ManyChat'in aynı API'leri o platformlar için de kullanılır (sadece `subscriber_id` değerleri ve belki endpoint path'leri ufak değişiklik gösterir). Tasarımımız oldukça geneldir; örneğin ManyChat'te WhatsApp için de bir Default Reply benzeri akış tetiklenip webhooks + sendFlow ile çalıştırılabilir.

10. Sistemin Her Sektöre Uygulanabilir Hale Getirilmesi (Esneklik)

Kurulan altyapı, bir **Kung Fu eğitmeni** örneğiyle açıklansa da, aslında herhangi bir sektördeki Instagram işletme hesabı için kullanılabilecek kadar esnek tasarlanmıştır. Bunu sağlayan başlıca unsurlar:

- **Sistem Promptunun Özelleştirilebilir olması:** Admin Paneli aracılığıyla farklı işletmelerin bilgilerine göre yeni bir sistem promptu oluşturulabilir. Örneğin bir restoran için sorulara verilen cevaplar bambaşka olacaktır (çalışma saatleri, menü, rezervasyon politikası, vb.). Bu cevaplardan derlenen prompt, AI'ın o işletmeye özel bilgilerle donatılmasını sağlar. Böylece **aynı kod altyapısı**, sadece environment ayarları farklılaştırılarak başka bir sektörde yeniden kullanılabilir. (Örneğin bu sistemi kopyalayıp bir restoranın ManyChat hesabına entegre ederken, sadece ManyChat API anahtarını ve prompt verilerini değiştirmek yeterli olacaktır.)
- **Dil ve Üslup Ayarı:** Promptta hedef dili ve tonlamayı belirleyebildiğimiz için, sistemi farklı dillerde hizmet veren işletmelere uygulayabilirsiniz. Azerice örnek verdik; Türkçe, İngilizce veya başka dilde cevap vermesini promptta düzenlemek mümkün. OpenAI'nin modeli çoklu dil desteklediğinden, kullanıcı hangi dilde yazarsa yazsın cevaplayabilir. Ama tutarlı kurumsal bir deneyim için prompt içinde dil tercihini net belirtmek en iyisidir.
- **ManyChat Entegrasyonu Standartlığı:** ManyChat tarafındaki yapı (Default Reply + External Request, Custom Field, Flow, API çağrıları) bütün sektörler ve hesaplar için aynıdır. Sadece belki bazı işletmeler Default Reply yerine spesifik tetikleyici tercih edebilir (örneğin belirli bir anahtar kelime gelince AI devreye girsin, diğer durumda farklı cevap verilsin). Bu durumda ManyChat'te kural bazlı bir yönlendirme yapılabilir. Ancak özünde, entegrasyon mekanizması değişmez.
- **Kodun Yeniden Kullanımı:** Yazılan Express + TypeScript kodu modüler şekilde tasarlandıysa (örneğin ManyChat API çağrıları ayrı bir service fonksiyonunda, OpenAI çağrısı ayrı bir fonksiyonda), bunu başka projelere kopyalayıp kullanmak kolay olacaktır. Hatta bunu bir template repository haline getirip, yeni bir işletme için hızlıca fork edip sadece prompt ve anahtarları değiştirebilirsiniz. Bu anlamda rehberimizin de yeniden kullanılabilir olmasına özen gösterdik.
- **Geliştirmeye Açık Yapı:** Sistem henüz temel bir Q&A otomasyonu. İleride daha gelişmiş özellikler eklenebilir:
- Örneğin, kullanıcı randevu almak isterse AI bunu algılayıp ManyChat içinde önceden hazırlanmış randevu akışına yönlendirebilir. Bunu yapmak için AI cevabını üretirken belli trigger kelimeler belirlenebilir veya ManyChat'te AI cevabından sonra etiket eklenip (API ile) bir rule tetiklenebilir. Bu gibi genişletmeler, farklı sektörlerin ihtiyaçlarına göre eklenebilir.
- AI cevabına resim veya katalog göndermek: ManyChat API, eğer `sendContent` ile JSON formatta mesaj gönderirsek resim, kart, galeri gibi öğeleri de destekliyor. Bizim yaklaşımda bu yok, ancak istenirse `replyText` içeriğinde ManyChat formatına uygun JSON hazırlayıp `sendContent` API'sine yollayarak zengin medya mesajları göndermek mümkün. Bunu her sektör kendine göre uyarlayabilir (örneğin e-ticaret firması, AI ürün tavsiye ettiğinde o ürünün resmini de göndermek isteyebilir).
- **Çoklu Dil Desteği:** Bir işletme birden fazla dilde müşteriyle yazışıyorsa, AI bunu da yönetebilir. Promptu dinamik olarak kullanıcının diline göre seçmek gerekebilir (Örneğin ManyChat'te kullanıcı dilini bir alanda tutup, backend'e gönderirken dil bilgisini de iletebilirsiniz. Sonra uygun dilde system prompt seçersiniz). Bu da genel bir yaklaşımla eklenebilir.

Sonuç: Bu rehberde adım adım sıfırdan bir ManyChat + OpenAI GPT-4 entegrasyonunu, Instagram DM'lerine otomatik yanıt veren bir sistem şeklinde planladık. Tüm kritik yapı taşlarını – webhook kurulumu, prompt tasarımı, API entegrasyonları, panel ile özelleştirme ve dağıtım – detaylandırdık. Bu

planı izleyerek kendi projenizi kurabilir ve gerektiğinde diğer projelere de uygulayabilirsiniz. Artık hiçbir adım eksik kalmayacak şekilde süreci baştan sona ele aldığımıza göre, güvenle implementasyona geçebilirsiniz. Başarılar dileriz!

Kaynaklar:

- ManyChat Default Reply özelliği ve kurulumu [3] [7]
- ManyChat External Request ve API kullanım ipuçları [9] [23]
- ManyChat API ile alan setme ve akış tetikleme örnekleri [21] [18]
- Railway ortam değişkenleri yönetimi [22]
- n8n ManyChat-OpenAI entegrasyon şablonu (benzer adımlar içeriyor) [1] .

---

[1] AI agent for Instagram DM/inbox. Manychat + Open AI integration | n8n workflow template
https://n8n.io/workflows/2718-ai-agent-for-instagram-dminbox-manychat-open-ai-integration/

[2] [3] [7] [10] [11] Default Reply in Manychat – Manychat Help
https://help.manychat.com/hc/en-us/articles/14281159586588-Default-Reply-in-Manychat

[4] How-to-use-ChatGPT-in-Manychat.md · GitHub
https://gist.github.com/robrita/8ad8b1eb65725a6896cfcd1acceff88d

[5] [6] [9] [12] [15] Dev Tools: Basics – Manychat Help
https://help.manychat.com/hc/en-us/articles/14281252007580-Dev-Tools-Basics

[8] [16] [18] [19] [21] [23] Referral Tracking and Rewarding in ManyChat — Part 1 | by Chad Wyatt | Chatbots Life
https://blog.chatbotslife.com/referral-tracking-and-rewarding-in-manychat-6105eedda8c6?gi=887cd87ab0aa

[13] How to send WhatsApp messages via APIs - Manychat Community
https://community.manychat.com/general-q-a-43/how-to-send-whatsapp-messages-via-apis-1577

[14] How do I manually add a system field to a contact? | Manychat
https://community.manychat.com/general-q-a-43/how-do-i-manually-add-a-system-field-to-a-contact-7027

[17] error","message":"Api max rps reached"} while sending request to ...
https://stackoverflow.com/questions/59674706/429-statuserror-messageapi-max-rps-reached-while-sending-request-t

[20] Duda con API Swagger de Manychat
https://community.manychat.com/preguntas-y-respuestas-58/duda-con-api-swagger-de-manychat-1294

[22] Using Variables | Railway Docs
https://docs.railway.com/guides/variables