Google Gemini API'si ile akıllı chatbot geliştirme sürecini, tıpkı bir evin temelinden çatısına kadar inşa edilmesi gibi, adım adım ele alalım. Bu yaklaşım, mevcut sisteminizin üzerine güçlü bir yapay zeka katmanı ekleyerek, kullanıcılarınızın hem geleneksel arayüzü hem de doğal dil ile etkileşim kurabilmesini sağlayacaktır.

**Temel Mimari Anlayışı**

Öncelikle, yapacağımız entegrasyonun mantığını anlayalım. Şu anda elinizde iki ayrı katman var: Node.js/Express backend'iniz veri işleme ve API sunumu yaparken, Next.js frontend'iniz bu verileri kullanıcıya sunar. Şimdi bu ikisi arasına üçüncü bir katman ekleyeceğiz - bir "çevirmen" görevi görecek akıllı chatbot.

Bu chatbot, kullanıcının doğal dil ile ifade ettiği istekleri anlayacak, bunları mevcut API çağrılarınıza çevirecek ve dönen sonuçları yine doğal dil ile açıklayacaktır. Düşünün ki, kullanıcı "geçen ayın satış rakamlarını göster" dediğinde, chatbot bunu /api/sales?month=last şeklinde bir API çağrısına dönüştürüp sonuçları anlamlı bir şekilde sunacaktır.

**Gemini API'si Neden Tercih Edilmeli?**

Google Gemini API'si, bu proje için ideal bir seçimdir çünkü ücretsiz kotası oldukça cömerttir ve özellikle kod anlama ve API entegrasyonu konularında güçlü performans sergiler. Ayrıca, çok dilli desteği sayesinde Türkçe etkileşimler için de mükemmel sonuçlar verir. Gemini'nin context window'u geniş olduğu için, uzun API dokümantasyonlarını ve konuşma geçmişini rahatlıkla işleyebilir.

**Aşama 1: Backend Altyapısının Hazırlanması**

Backend entegrasyonu için öncelikle gerekli paketleri yüklememiz gerekiyor. Projenizin backend klasöründe şu komutu çalıştırın:

npm install @google/generative-ai dotenv

npm install -D @types/node

Bu paketleri yükledikten sonra, Gemini API'si ile iletişim kuracak servis sınıfımızı oluşturalım. src/services/geminiChatService.ts dosyasını oluşturun:

import { GoogleGenerativeAI } from '@google/generative-ai';

// Gemini API istemcisini başlat

const genAI = new GoogleGenerativeAI(process.env.GEMINI\_API\_KEY!);

export interface ChatMessage {

role: 'user' | 'model'; // Gemini'de 'model' kullanılır, 'assistant' yerine

parts: string;

}

export interface ApiEndpoint {

method: string;

path: string;

description: string;

parameters?: string[];

examples?: string[];

}

export class GeminiChatService {

private model: any;

private systemInstruction: string;

constructor(apiEndpoints: ApiEndpoint[]) {

// Gemini Pro modelini kullanacağız - ücretsiz ve güçlü

this.model = genAI.getGenerativeModel({

model: 'gemini-pro',

generationConfig: {

temperature: 0.1, // Tutarlı yanıtlar için düşük sıcaklık

topK: 1,

topP: 0.95,

maxOutputTokens: 1024,

}

});

this.systemInstruction = this.createSystemInstruction(apiEndpoints);

}

private createSystemInstruction(endpoints: ApiEndpoint[]): string {

const endpointDescriptions = endpoints.map(ep => {

const params = ep.parameters ? `Parametreler: ${ep.parameters.join(', ')}` : '';

const examples = ep.examples ? `Örnekler: ${ep.examples.join('; ')}` : '';

return `- ${ep.method} ${ep.path}: ${ep.description}${params ? '\n ' + params : ''}${examples ? '\n ' + examples : ''}`;

}).join('\n');

return `Sen bu web uygulamasının akıllı asistanısın. Kullanıcıların aşağıdaki API endpoint'leri ile etkileşim kurmasına yardımcı olacaksın:

${endpointDescriptions}

Kullanıcı bir şey istediğinde, hangi endpoint'i çağırmak gerektiğini belirle ve cevabını JSON formatında ver.

Eğer API çağrısı yapman gerekiyorsa:

{

"action": "api\_call",

"endpoint": "endpoint\_yolu",

"method": "GET|POST|PUT|DELETE",

"parameters": {...},

"explanation": "Bu işlemi neden yapıyorum açıklaması"

}

Eğer sadece sohbet ediyorsan:

{

"action": "conversation",

"message": "yanıtın buraya",

"suggestions": ["öneri 1", "öneri 2"] // kullanıcıya yapabileceği işlemler için öneriler

}

Türkçe konuş ve kullanıcı dostu ol. Teknik terimleri açıkla.`;

}

async processUserMessage(

userMessage: string,

conversationHistory: ChatMessage[] = []

): Promise<any> {

try {

// Konuşma geçmişini ve sistem talimatlarını birleştir

const fullPrompt = `${this.systemInstruction}

Konuşma Geçmişi:

${conversationHistory.map(msg => `${msg.role}: ${msg.parts}`).join('\n')}

Kullanıcı: ${userMessage}

Asistan:`;

const result = await this.model.generateContent(fullPrompt);

const response = result.response;

const responseText = response.text();

return this.parseGeminiResponse(responseText);

} catch (error) {

console.error('Gemini API hatası:', error);

throw new Error('Mesaj işlenemedi, lütfen tekrar deneyin');

}

}

private parseGeminiResponse(response: string): any {

try {

// Gemini bazen JSON'u kod bloğu içinde döndürür, temizleyelim

const cleanResponse = response.replace(/```json\n?/g, '').replace(/```\n?/g, '').trim();

return JSON.parse(cleanResponse);

} catch (error) {

// JSON parse edilemezse, düz metin olarak döndür

return {

action: 'conversation',

message: response,

suggestions: []

};

}

}

// API çağrısı sonuçlarını doğal dile çevir

async explainApiResult(

apiCall: any,

apiResult: any,

userQuestion: string

): Promise<string> {

const prompt = `Kullanıcı "${userQuestion}" diye sordu.

Bu soruya cevap olarak ${apiCall.method} ${apiCall.endpoint} API'sini çağırdım.

API'nin döndürdüğü sonuç:

${JSON.stringify(apiResult, null, 2)}

Bu sonucu kullanıcıya Türkçe olarak, anlaşılır bir şekilde açıkla. Teknik detayları basitleştir ve önemli bilgileri vurgula.`;

try {

const result = await this.model.generateContent(prompt);

return result.response.text();

} catch (error) {

return 'API çağrısı başarılı oldu ancak sonucu açıklayamadım. Ham veri: ' + JSON.stringify(apiResult);

}

}

}

Şimdi bu servisi kullanacak controller'ımızı oluşturalım. src/controllers/chatController.ts dosyasını oluşturun:

import { Request, Response } from 'express';

import { GeminiChatService } from '../services/geminiChatService';

// Mevcut API endpoint'lerinizi burada tanımlayın

const API\_ENDPOINTS = [

{

method: 'GET',

path: '/api/users',

description: 'Tüm kullanıcıları listele',

parameters: ['page', 'limit', 'search'],

examples: ['kullanıcıları göster', 'kullanıcı listesi']

},

{

method: 'POST',

path: '/api/users',

description: 'Yeni kullanıcı oluştur',

parameters: ['name', 'email', 'role'],

examples: ['yeni kullanıcı ekle', 'kullanıcı oluştur']

},

{

method: 'GET',

path: '/api/analytics/sales',

description: 'Satış analitik verilerini getir',

parameters: ['startDate', 'endDate', 'category'],

examples: ['satış rakamları', 'bu ayın satışları', 'geçen ayın performansı']

},

// Diğer endpoint'lerinizi buraya ekleyin

];

const chatService = new GeminiChatService(API\_ENDPOINTS);

export class ChatController {

async handleChatMessage(req: Request, res: Response) {

try {

const { message, conversationHistory, pageContext } = req.body;

// Sayfa bağlamını mesaja ekle (kullanıcı hangi sayfadaysa)

const contextualMessage = pageContext ?

`Şu anda ${pageContext} sayfasındayım. ${message}` :

message;

// Gemini'den yanıt al

const aiResponse = await chatService.processUserMessage(

contextualMessage,

conversationHistory || []

);

// Eğer API çağrısı yapmak istiyorsa

if (aiResponse.action === 'api\_call') {

const apiResult = await this.executeApiCall(aiResponse, req);

// Sonucu doğal dile çevir

const explanation = await chatService.explainApiResult(

aiResponse,

apiResult,

message

);

res.json({

type: 'api\_response',

message: explanation,

data: apiResult,

originalRequest: aiResponse,

suggestions: this.generateSuggestions(apiResult)

});

} else {

// Sohbet yanıtı

res.json({

type: 'conversation',

message: aiResponse.message,

suggestions: aiResponse.suggestions || []

});

}

} catch (error) {

console.error('Chat controller hatası:', error);

res.status(500).json({

type: 'error',

message: 'Üzgünüm, bir hata oluştu. Lütfen tekrar deneyin.',

error: error.message

});

}

}

private async executeApiCall(aiResponse: any, originalReq: Request): Promise<any> {

const { endpoint, method, parameters } = aiResponse;

// Burada mevcut API endpoint'lerinizi çağıracaksınız

// Her endpoint için ayrı bir handler yazabilirsiniz

try {

switch (endpoint) {

case '/api/users':

if (method === 'GET') {

return await this.handleGetUsers(parameters);

} else if (method === 'POST') {

return await this.handleCreateUser(parameters);

}

break;

case '/api/analytics/sales':

if (method === 'GET') {

return await this.handleGetSalesAnalytics(parameters);

}

break;

default:

throw new Error(`Desteklenmeyen endpoint: ${method} ${endpoint}`);

}

} catch (error) {

throw new Error(`API çağrısı başarısız: ${error.message}`);

}

}

// Mevcut servis fonksiyonlarınızı burada çağırın

private async handleGetUsers(params: any): Promise<any> {

// Mevcut user service'inizi çağırın

// Örnek: return await userService.getUsers(params);

return {

users: [

{ id: 1, name: 'Ahmet Yılmaz', email: 'ahmet@example.com' },

{ id: 2, name: 'Fatma Öz', email: 'fatma@example.com' }

],

total: 2,

page: 1

};

}

private async handleCreateUser(params: any): Promise<any> {

// Mevcut user service'inizi çağırın

return {

success: true,

user: { id: 3, ...params },

message: 'Kullanıcı başarıyla oluşturuldu'

};

}

private async handleGetSalesAnalytics(params: any): Promise<any> {

// Mevcut analytics service'inizi çağırın

return {

totalSales: 150000,

period: params.startDate + ' - ' + params.endDate,

growth: '+15%',

topProducts: ['Ürün A', 'Ürün B', 'Ürün C']

};

}

private generateSuggestions(apiResult: any): string[] {

// API sonucuna göre kullanıcıya öneriler üret

const suggestions = [];

if (apiResult.users) {

suggestions.push('Bu kullanıcılar hakkında daha fazla bilgi ver');

suggestions.push('Yeni kullanıcı ekle');

}

if (apiResult.totalSales) {

suggestions.push('Grafik olarak göster');

suggestions.push('Önceki dönemle karşılaştır');

}

return suggestions;

}

}

Express rotanızı ana uygulama dosyanıza ekleyin:

// app.ts veya server.ts dosyanızda

import { ChatController } from './controllers/chatController';

const chatController = new ChatController();

// Chat endpoint'i ekle

app.post('/api/chat', chatController.handleChatMessage.bind(chatController));

// Sağlık kontrolü endpoint'i (chatbot durumunu kontrol etmek için)

app.get('/api/chat/health', (req, res) => {

res.json({ status: 'OK', service: 'Gemini Chat Service' });

});

**Aşama 2: Frontend Chat Arayüzünün Geliştirilmesi**

Şimdi kullanıcı dostu bir chat arayüzü oluşturalım. Bu arayüz, modern web standartlarına uygun, responsive ve erişilebilir olacak. components/GeminiChatBot.tsx dosyasını oluşturun:

'use client';

import React, { useState, useRef, useEffect } from 'react';

import { usePathname } from 'next/navigation';

interface ChatMessage {

id: string;

role: 'user' | 'assistant';

content: string;

timestamp: Date;

data?: any; // API yanıt verisi

suggestions?: string[]; // Kullanıcı önerileri

}

interface ChatResponse {

type: 'conversation' | 'api\_response' | 'error';

message: string;

data?: any;

suggestions?: string[];

}

export default function GeminiChatBot() {

const [messages, setMessages] = useState<ChatMessage[]>([

{

id: '1',

role: 'assistant',

content: 'Merhaba! Ben bu uygulamanın akıllı asistanıyım. Size nasıl yardımcı olabilirim? Veri sorgulama, kullanıcı işlemleri veya analitik raporlar hakkında sorularınızı yanıtlayabilirim.',

timestamp: new Date(),

suggestions: [

'Kullanıcı listesini göster',

'Bu ayın satış rakamlarını getir',

'Yeni kullanıcı nasıl eklerim?'

]

}

]);

const [inputValue, setInputValue] = useState('');

const [isLoading, setIsLoading] = useState(false);

const [isTyping, setIsTyping] = useState(false);

const messagesEndRef = useRef<HTMLDivElement>(null);

const pathname = usePathname(); // Mevcut sayfa bilgisi

// Mesajlar güncellendiğinde otomatik kaydırma

useEffect(() => {

messagesEndRef.current?.scrollIntoView({ behavior: 'smooth' });

}, [messages]);

const sendMessage = async (messageText?: string) => {

const textToSend = messageText || inputValue;

if (!textToSend.trim() || isLoading) return;

// Kullanıcı mesajını ekle

const userMessage: ChatMessage = {

id: Date.now().toString(),

role: 'user',

content: textToSend,

timestamp: new Date()

};

setMessages(prev => [...prev, userMessage]);

setInputValue('');

setIsLoading(true);

setIsTyping(true);

try {

// Backend'e mesaj gönder

const response = await fetch('/api/chat', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

message: textToSend,

conversationHistory: messages.map(m => ({

role: m.role === 'user' ? 'user' : 'model',

parts: m.content

})),

pageContext: pathname // Mevcut sayfa bilgisi

})

});

if (!response.ok) {

throw new Error('Sunucu hatası');

}

const result: ChatResponse = await response.json();

// Typing animasyonunu durdur

setIsTyping(false);

// Asistan mesajını ekle

const assistantMessage: ChatMessage = {

id: (Date.now() + 1).toString(),

role: 'assistant',

content: result.message,

timestamp: new Date(),

data: result.data,

suggestions: result.suggestions

};

setMessages(prev => [...prev, assistantMessage]);

} catch (error) {

setIsTyping(false);

console.error('Chat hatası:', error);

const errorMessage: ChatMessage = {

id: (Date.now() + 1).toString(),

role: 'assistant',

content: 'Üzgünüm, bir hata oluştu. Lütfen tekrar deneyin veya farklı bir şekilde sorun.',

timestamp: new Date(),

suggestions: ['Yeniden dene', 'Basit bir soru sor']

};

setMessages(prev => [...prev, errorMessage]);

} finally {

setIsLoading(false);

}

};

const handleKeyPress = (e: React.KeyboardEvent) => {

if (e.key === 'Enter' && !e.shiftKey) {

e.preventDefault();

sendMessage();

}

};

const handleSuggestionClick = (suggestion: string) => {

setInputValue(suggestion);

sendMessage(suggestion);

};

const formatApiData = (data: any): string => {

if (!data) return '';

try {

return JSON.stringify(data, null, 2);

} catch {

return String(data);

}

};

return (

<div className="flex flex-col h-[600px] bg-white rounded-lg shadow-lg border border-gray-200 overflow-hidden">

{/\* Chat Başlığı \*/}

<div className="bg-gradient-to-r from-blue-500 to-purple-600 text-white p-4">

<div className="flex items-center justify-between">

<div>

<h3 className="font-bold text-lg">Akıllı Asistan</h3>

<p className="text-blue-100 text-sm">Gemini ile destekleniyor</p>

</div>

<div className="w-3 h-3 bg-green-400 rounded-full animate-pulse"></div>

</div>

</div>

{/\* Mesajlar Alanı \*/}

<div className="flex-1 overflow-y-auto p-4 space-y-4 bg-gray-50">

{messages.map((message) => (

<div key={message.id} className="space-y-2">

<div

className={`flex ${message.role === 'user' ? 'justify-end' : 'justify-start'}`}

>

<div

className={`max-w-xs lg:max-w-md px-4 py-3 rounded-lg ${

message.role === 'user'

? 'bg-blue-500 text-white rounded-br-none'

: 'bg-white text-gray-800 shadow-md rounded-bl-none border'

}`}

>

<div className="whitespace-pre-wrap text-sm leading-relaxed">

{message.content}

</div>

<div className="text-xs opacity-70 mt-2">

{message.timestamp.toLocaleTimeString('tr-TR')}

</div>

</div>

</div>

{/\* API Verisi Gösterimi \*/}

{message.data && (

<div className="ml-4 p-3 bg-gray-100 rounded-lg border-l-4 border-blue-500">

<div className="text-xs text-gray-600 mb-2 font-medium">

📊 API Yanıt Verisi:

</div>

<pre className="text-xs bg-white p-2 rounded overflow-x-auto">

{formatApiData(message.data)}

</pre>

</div>

)}

{/\* Öneriler \*/}

{message.suggestions && message.suggestions.length > 0 && (

<div className="ml-4 space-y-2">

<div className="text-xs text-gray-600 font-medium">

💡 Bunları da deneyebilirsiniz:

</div>

<div className="flex flex-wrap gap-2">

{message.suggestions.map((suggestion, index) => (

<button

key={index}

onClick={() => handleSuggestionClick(suggestion)}

className="px-3 py-1 bg-blue-50 text-blue-700 rounded-full text-xs hover:bg-blue-100 transition-colors border border-blue-200"

>

{suggestion}

</button>

))}

</div>

</div>

)}

</div>

))}

{/\* Typing Göstergesi \*/}

{isTyping && (

<div className="flex justify-start">

<div className="bg-white px-4 py-3 rounded-lg shadow-md border rounded-bl-none">

<div className="flex items-center space-x-2">

<div className="flex space-x-1">

<div className="w-2 h-2 bg-blue-500 rounded-full animate-bounce"></div>

<div className="w-2 h-2 bg-blue-500 rounded-full animate-bounce" style={{animationDelay: '0.1s'}}></div>

<div className="w-2 h-2 bg-blue-500 rounded-full animate-bounce" style={{animationDelay: '0.2s'}}></div>

</div>

<span className="text-sm text-gray-500">Asistan yazıyor...</span>

</div>

</div>

</div>

)}

<div ref={messagesEndRef} />

</div>

{/\* Giriş Alanı \*/}

<div className="p-4 bg-white border-t border-gray-200">

<div className="flex items-end space-x-3">

<div className="flex-1">

<textarea

value={inputValue}

onChange={(e) => setInputValue(e.target.value)}

onKeyPress={handleKeyPress}

placeholder="Mesajınızı yazın... (Enter ile gönder)"

className="w-full resize-none border border-gray-300 rounded-lg px-4 py-3 focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent"

rows={1}

style={{ minHeight: '44px', maxHeight: '120px' }}

/>

</div>

<button

onClick={() => sendMessage()}

disabled={isLoading || !inputValue.trim()}

className="px-6 py-3 bg-blue-500 text-white rounded-lg hover:bg-blue-600 disabled:opacity-50 disabled:cursor-not-allowed transition-colors flex items-center space-x-2"

>

{isLoading ? (

<div className="w-4 h-4 border-2 border-white border-t-transparent rounded-full animate-spin"></div>

) : (

<>

<span>Gönder</span>

<span>📤</span>

</>

)}

</button>

</div>

</div>

</div>

);

}

**Aşama 3: Entegrasyon Stratejileri**

Chatbot'u mevcut uygulamanıza entegre etmek için üç farklı yaklaşım öneriyorum. Her birinin kendine özgü avantajları vardır:

**Yaklaşım 1: Yüzen Chat Widget'ı** Bu yaklaşım, kullanıcıların herhangi bir sayfada chatbot'a erişebilmesini sağlar. components/FloatingChatWidget.tsx dosyasını oluşturun:

'use client';

import React, { useState, useEffect } from 'react';

import GeminiChatBot from './GeminiChatBot';

export default function FloatingChatWidget() {

const [isOpen, setIsOpen] = useState(false);

const [hasUnreadMessage, setHasUnreadMessage] = useState(false);

// Yeni mesaj geldiğinde bildirimi göster

useEffect(() => {

if (!isOpen) {

setHasUnreadMessage(true);

}

}, [isOpen]);

const toggleChat = () => {

setIsOpen(!isOpen);

if (!isOpen) {

setHasUnreadMessage(false);

}

};

return (

<>

{/\* Chat Butonu \*/}

<button

onClick={toggleChat}

className="fixed bottom-6 right-6 w-16 h-16 bg-gradient-to-r from-blue-500 to-purple-600 text-white rounded-full shadow-lg hover:shadow-xl transition-all duration-300 flex items-center justify-center z-50 group"

aria-label="Chat Asistanı"

>

{isOpen ? (

<span className="text-xl">✕</span>

) : (

<div className="relative">

<span className="text-2xl">🤖</span>

{hasUnreadMessage && (

<div className="absolute -top-1 -right-1 w-4 h-4 bg-red-500 rounded-full animate-pulse"></div>

)}

</div>

)}

{/\* Hover tooltip \*/}

<div className="absolute right-full mr-3 px-3 py-1 bg-gray-800 text-white text-sm rounded-lg opacity-0 group-hover:opacity-100 transition-opacity whitespace-nowrap">

{isOpen ? 'Kapat' : 'Asistan ile konuş'}

</div>

</button>

{/\* Chat Penceresi \*/}

{isOpen && (

<div className="fixed bottom-24 right-6 w-96 h-[600px] z-50 animate-in slide-in-from-bottom-5 duration-300">

<GeminiChatBot />

</div>

)}

</>

);

}

**Yaklaşım 2: Bağlamsal Chat Entegrasyonu** Bu yaklaşım, belirli sayfalarda özel chat bileşenleri gösterir. Örneğin, kullanıcı yönetimi sayfasında kullanıcılarla ilgili soruları cevaplayacak özel bir chat:

'use client';

import React from 'react';

import GeminiChatBot from './GeminiChatBot';

interface ContextualChatProps {

pageContext: string;

initialMessage?: string;

suggestedActions?: string[];

}

export default function ContextualChat({

pageContext,

initialMessage,

suggestedActions = []

}: ContextualChatProps) {

return (

<div className="bg-white rounded-lg shadow-lg border border-gray-200 p-6">

<div className="mb-4">

<h3 className="text-lg font-semibold text-gray-800 mb-2">

{pageContext} Asistanı

</h3>

<p className="text-gray-600 text-sm">

{initialMessage || `${pageContext} ile ilgili sorularınızı yanıtlayabilirim.`}

</p>

</div>

{suggestedActions.length > 0 && (

<div className="mb-4">

<p className="text-sm text-gray-600 mb-2">Hızlı işlemler:</p>

<div className="flex flex-wrap gap-2">

{suggestedActions.map((action, index) => (

<button

key={index}

className="px-3 py-1 bg-blue-50 text-blue-700 rounded-full text-sm hover:bg-blue-100 transition-colors border border-blue-200"

>

{action}

</button>

))}

</div>

</div>

)}

<div className="h-96">

<GeminiChatBot />

</div>

</div>

);

}

**Yaklaşım 3: Ayrı Chat Sayfası** Daha kapsamlı sohbetler için ayrı bir sayfa oluşturabilirsiniz. pages/chat.tsx veya app/chat/page.tsx dosyasını oluşturun:

'use client';

import React from 'react';

import GeminiChatBot from '@/components/GeminiChatBot';

export default function ChatPage() {

return (

<div className="min-h-screen bg-gray-50 py-8">

<div className="max-w-4xl mx-auto px-4">

<div className="text-center mb-8">

<h1 className="text-3xl font-bold text-gray-800 mb-2">

Akıllı Asistan

</h1>

<p className="text-gray-600">

Uygulamanızla ilgili her türlü sorunuzu yanıtlayabilirim

</p>

</div>

<div className="bg-white rounded-lg shadow-lg overflow-hidden">

<GeminiChatBot />

</div>

<div className="mt-8 text-center">

<div className="grid grid-cols-1 md:grid-cols-3 gap-4 text-sm text-gray-600">

<div className="p-4 bg-white rounded-lg shadow">

<h3 className="font-semibold mb-2">🔍 Veri Sorgulama</h3>

<p>Kullanıcı listesi, satış raporları ve analitik veriler</p>

</div>

<div className="p-4 bg-white rounded-lg shadow">

<h3 className="font-semibold mb-2">⚙️ İşlem Yönetimi</h3>

<p>Yeni kayıt oluşturma, güncelleme ve silme işlemleri</p>

</div>

<div className="p-4 bg-white rounded-lg shadow">

<h3 className="font-semibold mb-2">📊 Analiz ve Rapor</h3>

<p>Performans metrikleri ve detaylı analiz raporları</p>

</div>

</div>

</div>

</div>

</div>

);

}

## Aşama 4: Gelişmiş Özellikler ve Optimizasyonlar

Chatbot'unuzun performansını artırmak ve kullanıcı deneyimini iyileştirmek için aşağıdaki gelişmiş özellikleri ekleyebilirsiniz:

**Konuşma Geçmişi Yönetimi** services/conversationManager.ts dosyasını oluşturun:

interface ConversationSession {

id: string;

userId?: string;

messages: ChatMessage[];

context: any;

createdAt: Date;

lastActivity: Date;

}

export class ConversationManager {

private sessions: Map<string, ConversationSession> = new Map();

private readonly MAX\_MESSAGES = 50; // Bellek kullanımını sınırla

private readonly SESSION\_TIMEOUT = 30 \* 60 \* 1000; // 30 dakika

createSession(userId?: string): string {

const sessionId = `chat\_${Date.now()}\_${Math.random().toString(36).substr(2, 9)}`;

this.sessions.set(sessionId, {

id: sessionId,

userId,

messages: [],

context: {},

createdAt: new Date(),

lastActivity: new Date()

});

return sessionId;

}

addMessage(sessionId: string, message: ChatMessage): void {

const session = this.sessions.get(sessionId);

if (!session) return;

session.messages.push(message);

session.lastActivity = new Date();

// Mesaj sayısını sınırla

if (session.messages.length > this.MAX\_MESSAGES) {

session.messages = session.messages.slice(-this.MAX\_MESSAGES);

}

}

getSession(sessionId: string): ConversationSession | undefined {

return this.sessions.get(sessionId);

}

updateContext(sessionId: string, context: any): void {

const session = this.sessions.get(sessionId);

if (session) {

session.context = { ...session.context, ...context };

}

}

// Eski oturumları temizle

cleanupExpiredSessions(): void {

const now = new Date();

for (const [sessionId, session] of this.sessions.entries()) {

if (now.getTime() - session.lastActivity.getTime() > this.SESSION\_TIMEOUT) {

this.sessions.delete(sessionId);

}

}

}

}

**Hata Yönetimi ve Yeniden Deneme Mekanizması** services/errorHandler.ts dosyasını oluşturun:

export class ChatErrorHandler {

private static readonly MAX\_RETRIES = 3;

private static readonly RETRY\_DELAY = 1000; // 1 saniye

static async withRetry<T>(

operation: () => Promise<T>,

context: string = 'Operation'

): Promise<T> {

let lastError: Error;

for (let attempt = 1; attempt <= this.MAX\_RETRIES; attempt++) {

try {

return await operation();

} catch (error) {

lastError = error as Error;

console.warn(`${context} failed on attempt ${attempt}:`, error);

if (attempt < this.MAX\_RETRIES) {

await this.delay(this.RETRY\_DELAY \* attempt);

}

}

}

throw new Error(`${context} failed after ${this.MAX\_RETRIES} attempts: ${lastError.message}`);

}

private static delay(ms: number): Promise<void> {

return new Promise(resolve => setTimeout(resolve, ms));

}

static handleGeminiError(error: any): string {

if (error.message?.includes('quota')) {

return 'API kotası doldu. Lütfen biraz sonra tekrar deneyin.';

}

if (error.message?.includes('rate limit')) {

return 'Çok hızlı mesaj gönderiyorsunuz. Lütfen bekleyin.';

}

if (error.message?.includes('safety')) {

return 'Güvenlik filtresi devreye girdi. Lütfen daha uygun bir soru sorun.';

}

return 'Bir hata oluştu. Lütfen tekrar deneyin.';

}

}

**Performans İzleme ve Analitik** services/chatAnalytics.ts dosyasını oluşturun:

interface ChatMetrics {

messageCount: number;

averageResponseTime: number;

apiCallCount: number;

errorRate: number;

userSatisfaction: number;

popularQueries: string[];

}

export class ChatAnalytics {

private metrics: ChatMetrics = {

messageCount: 0,

averageResponseTime: 0,

apiCallCount: 0,

errorRate: 0,

userSatisfaction: 0,

popularQueries: []

};

logInteraction(data: {

userMessage: string;

responseTime: number;

success: boolean;

apiCalled: boolean;

}): void {

this.metrics.messageCount++;

// Ortalama yanıt süresini güncelle

this.metrics.averageResponseTime = (

(this.metrics.averageResponseTime \* (this.metrics.messageCount - 1)) +

data.responseTime

) / this.metrics.messageCount;

if (data.apiCalled) {

this.metrics.apiCallCount++;

}

if (!data.success) {

this.metrics.errorRate = (this.metrics.errorRate \* (this.metrics.messageCount - 1) + 1) / this.metrics.messageCount;

}

// Popüler sorguları takip et

this.trackPopularQuery(data.userMessage);

}

private trackPopularQuery(query: string): void {

// Basit bir popüler sorgular takibi

const cleanQuery = query.toLowerCase().trim();

if (cleanQuery.length > 5) {

this.metrics.popularQueries.push(cleanQuery);

// Son 100 sorguyu tut

if (this.metrics.popularQueries.length > 100) {

this.metrics.popularQueries.shift();

}

}

}

getMetrics(): ChatMetrics {

return { ...this.metrics };

}

logUserFeedback(rating: number): void {

this.metrics.userSatisfaction = (this.metrics.userSatisfaction + rating) / 2;

}

}

## Aşama 5: Güvenlik ve Sınırlamalar

Chatbot'unuzun güvenli çalışması için aşağıdaki önlemleri alın:

**Rate Limiting ve Güvenlik** middleware/rateLimiter.ts dosyasını oluşturun:

import { Request, Response, NextFunction } from 'express';

interface RateLimitInfo {

count: number;

resetTime: number;

}

export class ChatRateLimiter {

private static requests: Map<string, RateLimitInfo> = new Map();

private static readonly WINDOW\_MS = 60 \* 1000; // 1 dakika

private static readonly MAX\_REQUESTS = 20; // Dakikada 20 mesaj

static middleware(req: Request, res: Response, next: NextFunction): void {

const clientId = req.ip || 'anonymous';

const now = Date.now();

// Eski kayıtları temizle

this.cleanupOldRequests(now);

const requestInfo = this.requests.get(clientId);

if (!requestInfo) {

this.requests.set(clientId, { count: 1, resetTime: now + this.WINDOW\_MS });

return next();

}

if (now > requestInfo.resetTime) {

// Pencere sıfırlandı

this.requests.set(clientId, { count: 1, resetTime: now + this.WINDOW\_MS });

return next();

}

if (requestInfo.count >= this.MAX\_REQUESTS) {

return res.status(429).json({

error: 'Çok fazla mesaj gönderiyorsunuz. Lütfen biraz bekleyin.',

retryAfter: Math.ceil((requestInfo.resetTime - now) / 1000)

});

}

requestInfo.count++;

next();

}

private static cleanupOldRequests(now: number): void {

for (const [clientId, info] of this.requests.entries()) {

if (now > info.resetTime) {

this.requests.delete(clientId);

}

}

}

}

**Girdi Doğrulama ve Temizleme** utils/inputValidator.ts dosyasını oluşturun:

export class InputValidator {

private static readonly MAX\_MESSAGE\_LENGTH = 2000;

private static readonly FORBIDDEN\_PATTERNS = [

/(<script[\s\S]\*?>[\s\S]\*?<\/script>)/gi,

/(<iframe[\s\S]\*?>[\s\S]\*?<\/iframe>)/gi,

/(javascript:)/gi,

/(on\w+\s\*=)/gi

];

static validateMessage(message: string): { valid: boolean; error?: string } {

if (!message || typeof message !== 'string') {

return { valid: false, error: 'Mesaj boş olamaz' };

}

if (message.length > this.MAX\_MESSAGE\_LENGTH) {

return { valid: false, error: 'Mesaj çok uzun' };

}

// Zararlı içerik kontrolü

for (const pattern of this.FORBIDDEN\_PATTERNS) {

if (pattern.test(message)) {

return { valid: false, error: 'Güvenlik nedeniyle mesaj reddedildi' };

}

}

return { valid: true };

}

static sanitizeMessage(message: string): string {

return message

.replace(/[<>]/g, '') // HTML taglarını temizle

.replace(/javascript:/gi, '') // JavaScript URL'lerini temizle

.trim();

}

}

## Aşama 6: Test ve Dağıtım

Chatbot'unuzu test etmek için kapsamlı bir test paketi oluşturun:

**Unit Test Örneği** tests/chatService.test.ts dosyasını oluşturun:

import { GeminiChatService } from '../src/services/geminiChatService';

describe('GeminiChatService', () => {

let chatService: GeminiChatService;

beforeEach(() => {

const mockEndpoints = [

{

method: 'GET',

path: '/api/users',

description: 'Kullanıcıları listele',

examples: ['kullanıcıları göster']

}

];

chatService = new GeminiChatService(mockEndpoints);

});

test('should identify user intent correctly', async () => {

// Mock Gemini API response

const mockResponse = {

action: 'api\_call',

endpoint: '/api/users',

method: 'GET',

explanation: 'Kullanıcı listesini getiriyorum'

};

// Test implementation

expect(mockResponse.action).toBe('api\_call');

expect(mockResponse.endpoint).toBe('/api/users');

});

test('should handle conversation gracefully', async () => {

const mockResponse = {

action: 'conversation',

message: 'Size nasıl yardımcı olabilirim?',

suggestions: ['Kullanıcı listesi', 'Satış raporu']

};

expect(mockResponse.action).toBe('conversation');

expect(mockResponse.suggestions).toHaveLength(2);

});

});

**Entegrasyon Testi** tests/integration/chat.test.ts dosyasını oluşturun:

import request from 'supertest';

import app from '../../src/app';

describe('Chat API Integration', () => {

test('POST /api/chat should respond successfully', async () => {

const response = await request(app)

.post('/api/chat')

.send({

message: 'Merhaba',

conversationHistory: []

});

expect(response.status).toBe(200);

expect(response.body).toHaveProperty('type');

expect(response.body).toHaveProperty('message');

});

test('POST /api/chat should handle rate limiting', async () => {

// Hızlı ardışık istekler gönder

const requests = Array(25).fill(null).map(() =>

request(app)

.post('/api/chat')

.send({ message: 'Test' })

);

const responses = await Promise.all(requests);

const rateLimitedResponses = responses.filter(res => res.status === 429);

expect(rateLimitedResponses.length).toBeGreaterThan(0);

});

});

## Aşama 7: Ortam Değişkenleri ve Yapılandırma

.env dosyanızı oluşturun:

# Gemini API Anahtarı

GEMINI\_API\_KEY=your\_gemini\_api\_key\_here

# Uygulama Ayarları

NODE\_ENV=development

PORT=3000

# Chat Ayarları

CHAT\_RATE\_LIMIT\_WINDOW\_MS=60000

CHAT\_RATE\_LIMIT\_MAX\_REQUESTS=20

CHAT\_MAX\_MESSAGE\_LENGTH=2000

CHAT\_SESSION\_TIMEOUT\_MS=1800000

# Logging

LOG\_LEVEL=info

next.config.js dosyanızı güncelleyin:

/\*\* @type {import('next').NextConfig} \*/

const nextConfig = {

env: {

GEMINI\_API\_KEY: process.env.GEMINI\_API\_KEY,

},

// API rotalarını proxy olarak yönlendir

async rewrites() {

return [

{

source: '/api/:path\*',

destination: 'http://localhost:3001/api/:path\*',

},

];

},

}

module.exports = nextConfig;

## Aşama 8: Performans Optimizasyonu

**Yanıt Önbelleği** services/responseCache.ts dosyasını oluşturun:

interface CacheEntry {

response: any;

timestamp: number;

ttl: number;

}

export class ResponseCache {

private cache: Map<string, CacheEntry> = new Map();

private readonly DEFAULT\_TTL = 5 \* 60 \* 1000; // 5 dakika

generateKey(message: string, context: any): string {

return `${message.toLowerCase().trim()}\_${JSON.stringify(context)}`;

}

set(key: string, response: any, ttl: number = this.DEFAULT\_TTL): void {

this.cache.set(key, {

response,

timestamp: Date.now(),

ttl

});

}

get(key: string): any | null {

const entry = this.cache.get(key);

if (!entry) return null;

if (Date.now() - entry.timestamp > entry.ttl) {

this.cache.delete(key);

return null;

}

return entry.response;

}

clear(): void {

this.cache.clear();

}

// Periyodik temizlik

cleanup(): void {

const now = Date.now();

for (const [key, entry] of this.cache.entries()) {

if (now - entry.timestamp > entry.ttl) {

this.cache.delete(key);

}

}

}

}

## Kullanım Örnekleri ve Best Practices

**Chatbot'u Ana Sayfanıza Entegre Etme** app/layout.tsx dosyanızı güncelleyin:

import FloatingChatWidget from '@/components/FloatingChatWidget';

export default function RootLayout({

children,

}: {

children: React.ReactNode;

}) {

return (

<html lang="tr">

<body>

{children}

<FloatingChatWidget />

</body>

</html>

);

}

**Sayfa Bazlı Bağlamsal Chat** Kullanıcı yönetimi sayfanızda:

// pages/users/index.tsx

import ContextualChat from '@/components/ContextualChat';

export default function UsersPage() {

return (

<div className="container mx-auto px-4 py-8">

<h1 className="text-2xl font-bold mb-6">Kullanıcı Yönetimi</h1>

<div className="grid grid-cols-1 lg:grid-cols-3 gap-6">

<div className="lg:col-span-2">

{/\* Mevcut kullanıcı listesi \*/}

<UsersList />

</div>

<div className="lg:col-span-1">

<ContextualChat

pageContext="Kullanıcı Yönetimi"

initialMessage="Kullanıcılarla ilgili işlemlerinizde size yardımcı olabilirim."

suggestedActions={[

"Tüm kullanıcıları listele",

"Yeni kullanıcı ekle",

"Kullanıcı ara"

]}

/>

</div>

</div>

</div>

);

}

Bu kapsamlı kılavuz, Gemini LLM API'si ile güçlü bir chatbot sistemi kurmanızı sağlar. Sistem, kullanıcılarınızın hem geleneksel GUI'yi hem de doğal dil etkileşimini kullanarak aynı işlevlere erişebilmesini mümkün kılar. Kademeli olarak uygulayarak, her aşamada test ederek ilerleyebilir ve ihtiyaçlarınıza göre özelleştirmeler yapabilirsiniz.