Dezvoltarea Aplicațiilor Web utilizând ASP.NET Core MVC
Curs 12

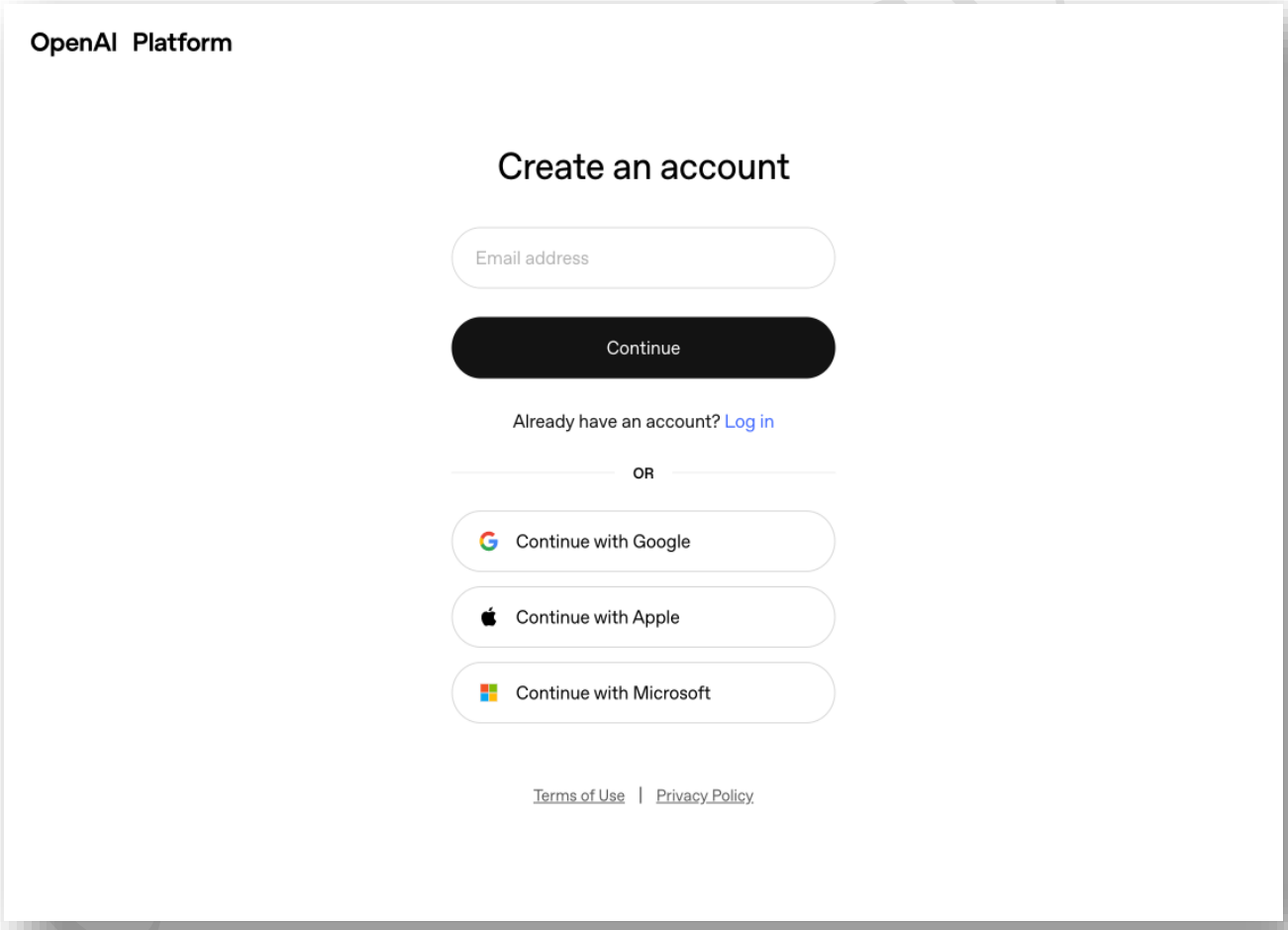
Cuprins

Crearea unui cont pe OpenAI Platform și obținerea unui API Key

Pasul 1: Se accesează pagina de înregistrare

Se deschide browserul și se navighează la adresa:

https://platform.openai.com/signup

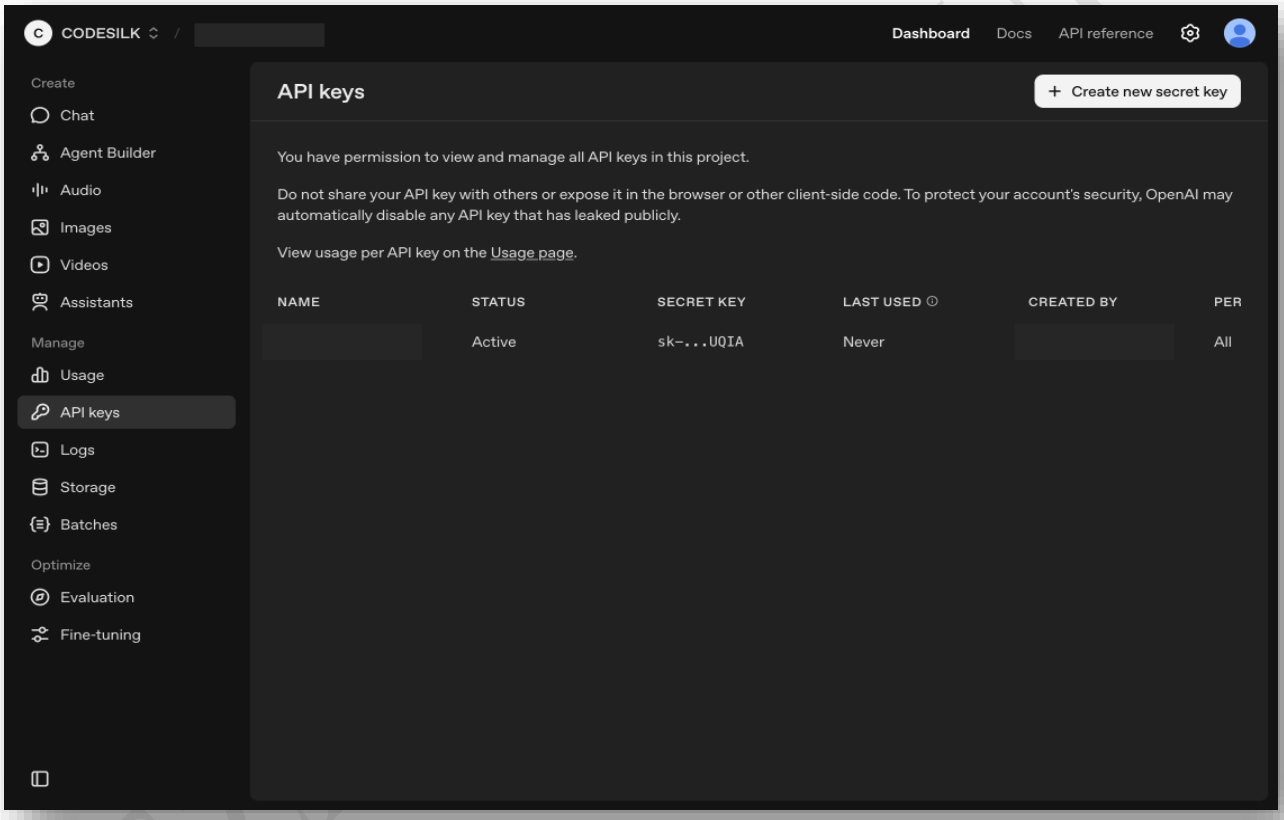


Pasul 2: Se realizează înregistrarea

Pasul 3: Se accesează secțiunea API Keys

După autentificare, se navighează la: https://platform.openai.com/api-keys

Din meniul din stânga, se apasă pe "**API keys**" (în secțiunea "**Manage**").



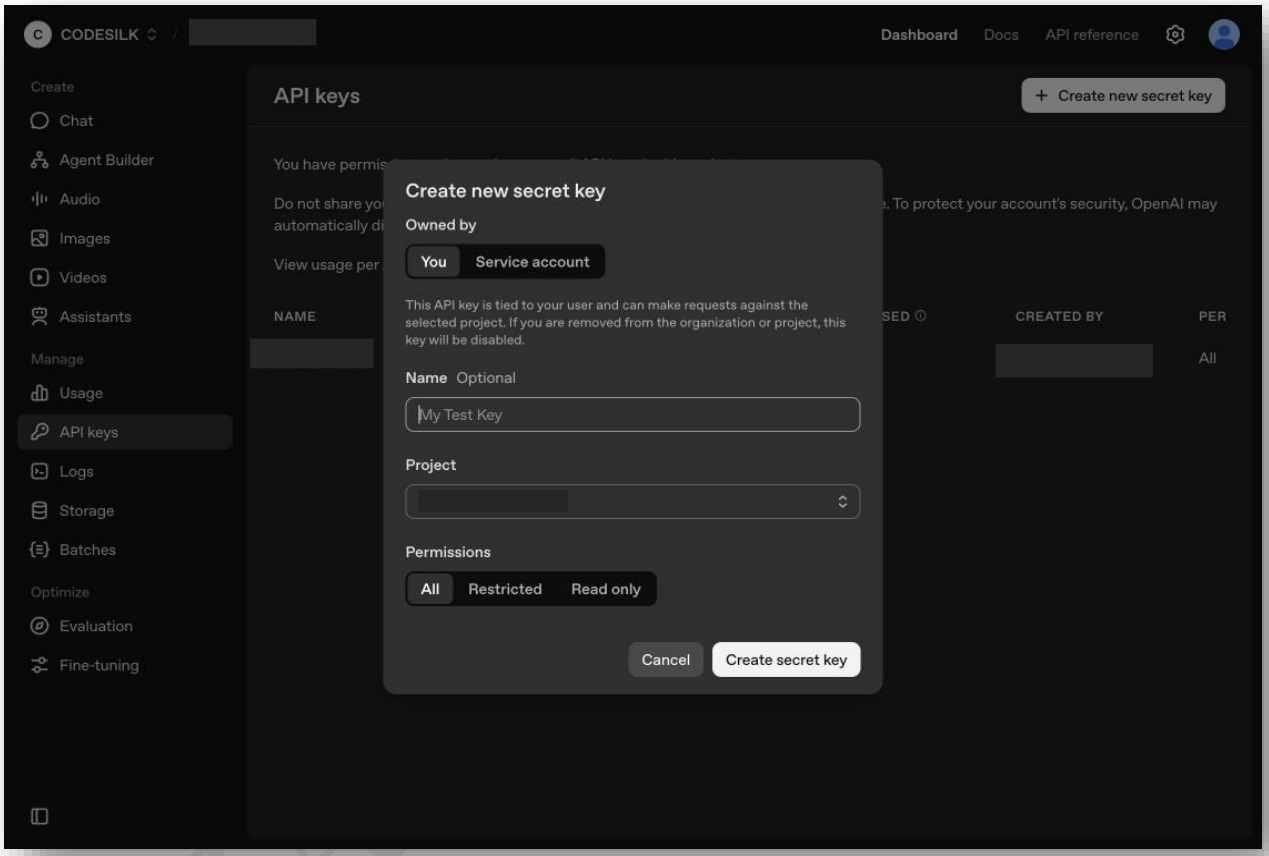
Pasul 4: Se creează un nou API Key

Se apasă pe butonul "**+ Create new secret key**". Se va deschide o fereastră unde se pot configura:

- Owned by: "**You**" (pentru contul tău) sau "**Service account**" (pentru aplicații);

- Name: un nume descriptiv care se dă cheii (de ex: “**App Web Production**”);
- Project: se selectează proiectul;
- Permissions: “**All**” pentru acces complet;

Se apasă “**Create secret key**” pentru a se genera cheia.



OBSERVAȚIE!

Nu partaja niciodată cheia API cu alte persoane și nu o expune în codul client-side!

Pasul 5: Se copiază și se salvează cheia API

OBSERVAȚIE!

După ce se apasă "**Create secret key**":

- Cheia se va afișa O SINGURĂ DATĂ
- Ea trebuie copiată imediat și salvată într-un loc sigur
- Nu o să mai fie vizibilă după închiderea ferestrei

Utilizarea cheii și serviciului AI în aplicația web

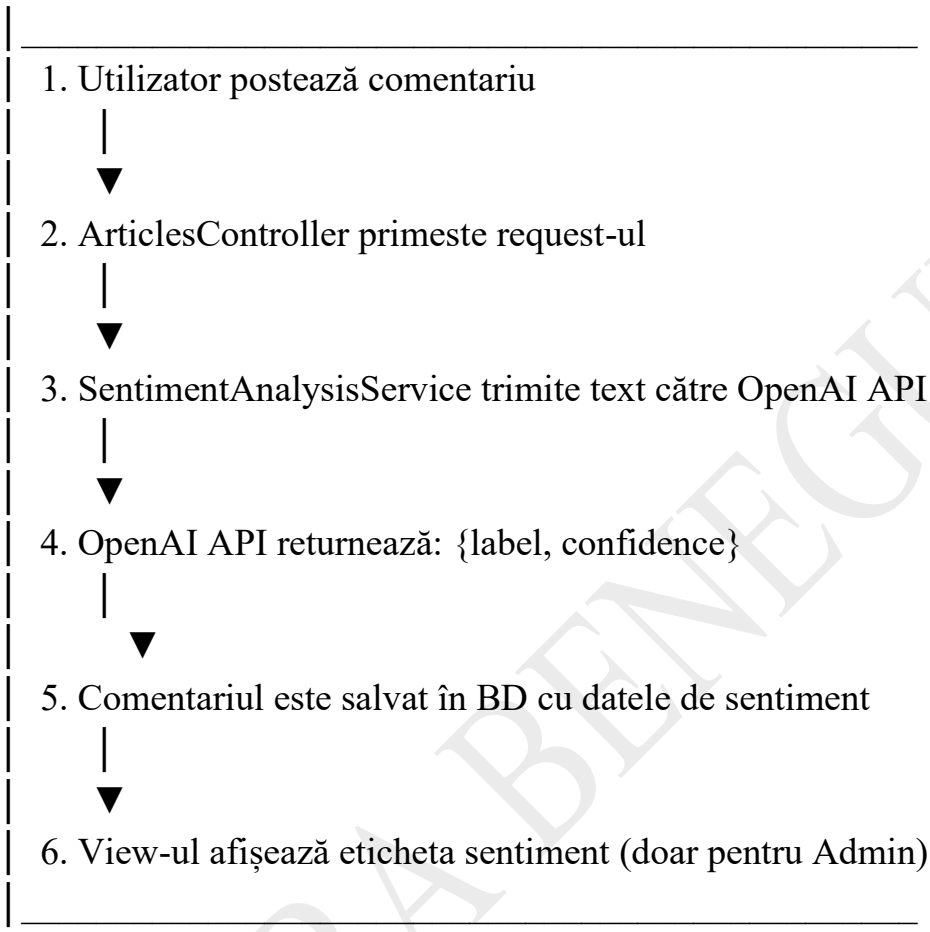
În continuare se va implementa un **sistem de analiză de sentiment** pentru comentariile postate pe platforma **ArticlesApp**. Sistemul utilizează API-ul **OpenAI** pentru a clasifica automat fiecare comentariu ca fiind **pozitiv**, **neutru** sau **negativ**, împreună cu un scor de încredere (confidence score).

Caracteristici principale:

- Analiza automată a sentimentului la postarea unui comentariu;
- Clasificare în trei categorii: positive, neutral, negative;
- Scor de încredere între 0% și 100%;
- Etichetele sunt vizibile **doar pentru utilizatorii cu rol de Admin**;

Arhitectura Soluției:

FLUX DE DATE



Pasul 1: Modificarea Modelului Comment

Primul pas este adăugarea câmpurilor necesare pentru stocarea rezultatelor analizei de sentiment în modelul ***Comment***.

Models/Comment.cs

```
using System;
using System.ComponentModel.DataAnnotations;

namespace ArticlesApp.Models
{
    public class Comment
    {
        [Key]
        public int Id { get; set; }

        [Required(ErrorMessage = "Continutul comentariului este
obligatoriu")]
        public string Content { get; set; }

        public DateTime Date { get; set; }

        public int ArticleId { get; set; }

        public string? UserId { get; set; }

        public virtual ApplicationUser? User { get; set; }

        public virtual Article? Article { get; set; }

        // CAMPURI NOI PENTRU ANALIZA DE SENTIMENT

        // Eticheta sentimentului: "positive", "neutral",
"negative"
        public string? SentimentLabel { get; set; }

        // Scorul de incredere: valoare intre 0.0 si 1.0
        public double? SentimentConfidence { get; set; }

        // Data si ora la care s-a efectuat analiza
        public DateTime? SentimentAnalyzedAt { get; set; }
    }
}
```

Explicație:

- **SentimentLabel**: Stochează clasificarea sentimentului (positive/neutral/negative)
- **SentimentConfidence**: Scorul de încredere al clasificării (0.0 - 1.0)
- **SentimentAnalyzedAt**: Timestamp-ul analizei pentru audit

Pasul 2: Crearea Serviciului de Analiză de Sentiment

Se creează un serviciu dedicat, care comunică cu API-ul OpenAI pentru analiza textului.

```
using System.Net.Http.Headers;
using System.Text;
using System.Text.Json;
using System.Text.Json.Serialization;

namespace ArticlesApp.Services
{
    // Clasa pentru rezultatul analizei de sentiment
    public class SentimentResult
    {
        public string Label { get; set; } = "neutral";  // positive, 
neutral, negative
        public double Confidence { get; set; } = 0.0;   // 0.0 - 1.0
        public bool Success { get; set; } = false;
        public string? ErrorMessage { get; set; }
    }

    // Interfata serviciului pentru dependency injection
    public interface ISentimentAnalysisService
    {
        Task<SentimentResult> AnalyzeSentimentAsync(string text);
    }


    // Implementarea serviciului de analiza de sentiment folosind 
OpenAI API
    public class SentimentAnalysisService : ISentimentAnalysisService
    {
        private readonly HttpClient _httpClient;
        private readonly string _apiKey;
        private readonly ILogger<SentimentAnalysisService> _logger;
```

```
        public SentimentAnalysisService(IConfiguration configuration,
ILogger<SentimentAnalysisService> logger)
        {
            _httpClient = new HttpClient();
            _apiKey = configuration["OpenAI:ApiKey"] ?? throw new
ArgumentNullException("OpenAI:ApiKey not configured");
            _logger = logger;

            // Configurare HttpClient pentru OpenAI API
            _httpClient.BaseAddress = new
Uri("https://api.openai.com/v1/");
            _httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", _apiKey);
            _httpClient.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
        }

        public async Task<SentimentResult>
AnalyzeSentimentAsync(string text)
        {
            try
            {
                // Construim prompt-ul pentru analiza de sentiment
                var systemPrompt = @"You are a sentiment analysis
assistant. Analyze the sentiment of the given text and respond ONLY
with a JSON object in this exact format:
{""label"": ""positive|neutral|negative"", ""confidence"": 0.0-1.0}

Rules:
- label must be exactly one of: positive, neutral, negative
- confidence must be a number between 0.0 and 1.0
- Do not include any other text, only the JSON object";

                var userPrompt = $"Analyze the sentiment of this
comment: \"{text}\"";

                // Construim request-ul pentru OpenAI API
                var requestBody = new
                {
                    model = "gpt-4o-mini",  // Using gpt-4o-mini as
gpt-5-nano doesn't exist
                    messages = new[]
                    {
                        new { role = "system", content = systemPrompt
},
                        new { role = "user", content = userPrompt }
                    },
                    temperature = 0.1,  // Low temperature for
consistent results
                    max_tokens = 50
                };
```

```
                var jsonContent =
JsonSerializer.Serialize(requestBody);
                var content = new StringContent(jsonContent,
Encoding.UTF8, "application/json");

                _logger.LogInformation("Sending sentiment analysis
request to OpenAI API");

                // Trimitem request-ul catre OpenAI API
                var response = await
_httpClient.PostAsync("chat/completions", content);
                var responseContent = await
response.Content.ReadAsStringAsync();

                if (!response.IsSuccessStatusCode)
                {
                    _logger.LogError("OpenAI API error: {StatusCode} -
{Content}", response.StatusCode, responseContent);
                    return new SentimentResult
                    {
                        Success = false,
                        ErrorMessage = $"API Error:
{response.StatusCode}"
                    };
                }

                // Parsam raspunsul de la OpenAI
                var openAiResponse =
JsonSerializer.Deserialize<OpenAiResponse>(responseContent);
                var assistantMessage =
openAiResponse?.Choices?.FirstOrDefault()?.Message?.Content;

                if (string.IsNullOrEmpty(assistantMessage))
                {
                    return new SentimentResult
                    {
                        Success = false,
                        ErrorMessage = "Empty response from API"
                    };
                }

                _logger.LogInformation("OpenAI response: {Response}",
assistantMessage);

                // Parsam JSON-ul din raspunsul asistentului
                var sentimentData =
JsonSerializer.Deserialize<SentimentResponse>(assistantMessage);

                if (sentimentData == null)
                {
                    return new SentimentResult
                    {
                        Success = false,
```

```
                    ErrorMessage = "Failed to parse sentiment
response"
                };
            }
            // Validam si normalizam label-ul
            var label = sentimentData.Label?.ToLower() switch
            {
                "positive" => "positive",
                "negative" => "negative",
                _ => "neutral"
            };

            // Validam confidence score
            var confidence = Math.Clamp(sentimentData.Confidence,
0.0, 1.0);

            return new SentimentResult
            {
                Label = label,
                Confidence = confidence,
                Success = true
            };
        }
        catch (Exception ex)
        {
            _logger.LogError(ex, "Error analyzing sentiment");
            return new SentimentResult
            {
                Success = false,
                ErrorMessage = ex.Message
            };
        }
    }
}

// Clase pentru deserializarea raspunsului OpenAI
public class OpenAiResponse
{
    [JsonPropertyName("choices")]
    public List<Choice>? Choices { get; set; }
}

public class Choice
{
    [JsonPropertyName("message")]
    public Message? Message { get; set; }
}

public class Message
{
    [JsonPropertyName("content")]
    public string? Content { get; set; }
}
```

```
    public class SentimentResponse
    {
        [JsonPropertyName("label")]
        public string? Label { get; set; }

        [JsonPropertyName("confidence")]
        public double Confidence { get; set; }
    }
}
```

Explicație Cod:

În acest exemplu construim un **serviciu** (un modul separat) care primește un text (comentariu) și întoarce un rezultat de tip **sentiment**: *positive / neutral / negative*, împreună cu un scor de încredere (*confidence*).

1) De ce se utilizează o interfață (ISentimentAnalysisService)

Interfața definește serviciul: metoda **AnalyzeSentimentAsync**(string text).

Avantajele sunt:

- putem injecta ușor serviciul cu **Dependency Injection**;
- putem schimba implementarea (ex: alt API) fără să modificăm restul aplicației;

2) Ce este SentimentResult și de ce e util

SentimentResult este obiectul pe care îl întoarce serviciul către aplicație. Conține:

- Label (positive/neutral/negative)
- Confidence (0.0–1.0)
- Success (true/false)
- ErrorMessage (dacă apare o problemă)

3) Cum se configurează cererea către OpenAI (HttpClient + headers)

Serviciul folosește HttpClient ca să trimită un request HTTP la OpenAI. În constructor:

- se setează BaseAddress = https://api.openai.com/v1/
- se setează header-ul Authorization: Bearer <ApiKey>
- se acceptă răspuns JSON (application/json)

Cheia API este citită din configurație: **OpenAI:ApiKey**. Dacă lipsește, serviciul aruncă o eroare.

4) Prompting: cum o să răspundă modelul

Serviciul trimite două mesaje:

- **system prompt**: reguli stricte (modelul trebuie să răspundă DOAR cu JSON în format fix)
- **user prompt**: textul comentariului care trebuie analizat

Regula "**respond ONLY with JSON**" este importantă, deoarece dorim să **parsam** răspunsul automat. Dacă modelul ar răspunde cu explicații complexe, nu ar mai funcționa deserializarea.

5) Request body – ce se trimite

În request body se trimite:

- model (ex: gpt-4o-mini)
- messages (system + user)
- temperature = 0.1 (rezultate consistente, mai puțină variație)
- max_tokens = 50 (răspuns scurt, fiind doar JSON)

Următorul pas îl constitutie serializarea obiectul în JSON (JsonSerializer.Serialize) și trimiterea prin POST către endpoint-ul **chat/completions**.

6) Tratarea erorilor (rețea / API)

După request:

- dacă status code nu este "success" (ex: 401, 429, 500), se întoarce Success=false și un mesaj de eroare
- se loghează detaliile

Acest pas este esențial în aplicații reale deoarece API-ul poate da rate-limit, cheie greșită, sau probleme temporare.

7) Parsarea răspunsului OpenAI și extragerea conținutului

Răspunsul OpenAI este un JSON mare, din care o să se preia doar: choices[0].message.content

Acest răspuns ar trebui să conțină JSON-ul mic cu sentimentul.

8) Deserializare + validare

După ce se preia conținutul, se deserializează în SentimentResponse (label + confidence), după care se realizează validări:

- label este normalizat la: positive / negative / neutral (orice altceva devine neutral)
- confidence este limitat cu Math.Clamp între 0 și 1

Aceste validări protejează aplicația de răspunsuri imperfecte sau neașteptate.

9) Rezultatul final

Dacă totul a mers bine, se returnează SentimentResult cu:

- Success=true
- Label și Confidence validate

Structura Request OpenAI:

```
{
 "model": "gpt-4o-mini",
 "messages": [
        {"role": "system", "content": "...prompt sistem..."},
        {"role": "user", "content": "Analyze the sentiment: \"text
comentariu\""}
 ],
 "temperature": 0.1,
 "max_tokens": 50
}
```

Structura Raspuns Asteptat:

```
{
 "label": "positive",
 "confidence": 0.85
}
```

Pasul 3: Configurarea API Key-ului OpenAI

API key-ul trebuie stocat în fișierul de configurare.

appsettings.json

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Data
Source=(localdb)\\mssqllocaldb;Initial
Catalog=EngineAppDB;Integrated Security=True;Multiple Active
Result Sets=True"
  },
  "OpenAI": {
    "ApiKey": "sk-proj-____"
  },
  "GoogleAI": {
    "ApiKey": "____"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"

}
```

OBSERVAȚIE!

- NU comiteți API key-uri în repository-uri Git publice
- Pentru producție, folosiți User Secrets sau Azure Key Vault
- API key-ul poate fi setat și prin variabila de mediu: OpenAI__ApiKey

Pasul 4: Inregistrarea Serviciului în DI Container

Serviciul trebuie înregistrat în container-ul de Dependency Injection.

Program.cs

```
using ArticlesApp.Services; // Adaugam using-ul


// ... cod existent ...

builder.Services.AddControllersWithViews();

// Inregistrare serviciu pentru analiza de sentiment
// Folosim Google AI (Gemini) in loc de OpenAI

builder.Services.AddScoped<ISentimentAnalysisService,
GoogleSentimentAnalysisService>();

var app = builder.Build();
```

Explicație:

- **AddScoped**: Creeaza o instanță nouă pentru fiecare request HTTP
- Alternativ, se poate folosi **AddSingleton** pentru o singură instanță globală

Pasul 5: Crearea Migratiei pentru Baza de Date

Trebuie actualizată schema bazei de date pentru a include noile câmpuri.

Comenzi Terminal:

 # Se creează migrația
dotnet ef migrations add AddSentimentToComments – MAC sau Linux

Add-Migration AddSentimentToComments – Windows

 # Se aplică migrația
dotnet ef database update – MAC sau Linux

Update-Database – Windows

Pasul 6: Integrarea în Controller

Se modifică **ArticlesController** pentru a analiza sentimentul la postarea unui comentariu.

Controllers/ArticlesController.cs

```
using ArticlesApp.Services;  // Adaugam using-ul

namespace ArticlesApp.Controllers
{

    public class ArticlesController(
        ApplicationDbContext context,
        UserManager<ApplicationUser> userManager,
        RoleManager<IdentityRole> roleManager,
        ISentimentAnalysisService sentimentService) : Controller
    {
        private readonly ApplicationDbContext db = context;
        private readonly UserManager<ApplicationUser>
_userManager = userManager;
```

```
        private readonly RoleManager<IdentityRole> _roleManager
= roleManager;
        private readonly ISentimentAnalysisService
_sentimentService = sentimentService;


     // ... alte metode ...

 // Adaugarea unui comentariu asociat unui articol in baza de
date
// Toate rolurile pot adauga comentarii in baza de date
// Se efectueaza analiza de sentiment folosind OpenAI API

[HttpPost]
[Authorize(Roles = "User,Editor,Admin")]

public async Task<IActionResult> Show([FromForm] Comment
comment)

{
    comment.Date = DateTime.Now;

    // preluam Id-ul utilizatorului care posteaza comentariul
    comment.UserId = _userManager.GetUserId(User);

    if (ModelState.IsValid)
    {
        // Analizam sentimentul comentariului folosind OpenAI
API
        var sentimentResult = await
_sentimentService.AnalyzeSentimentAsync(comment.Content);

        if (sentimentResult.Success)
        {
            comment.SentimentLabel = sentimentResult.Label;
            comment.SentimentConfidence =
sentimentResult.Confidence;
            comment.SentimentAnalyzedAt = DateTime.Now;
        }

        db.Comments.Add(comment);
        db.SaveChanges();
        return Redirect("/Articles/Show/" + comment.ArticleId);
    }
    else
    {
        Article? art = db.Articles
                         .Include(a => a.Category)
```

```
                        .Include(a => a.User)
                        .Include(a => a.Comments)
                            .ThenInclude(c => c.User)
                        .Where(art => art.Id ==
comment.ArticleId)
                        .FirstOrDefault();

        if (art is null)
        {
            return NotFound();
        }

        //return Redirect("/Articles/Show/" + comm.ArticleId);

        SetAccessRights();

        // Adaugam bookmark-urile utilizatorului pentru dropdown
        ViewBag.UserBookmarks = db.Bookmarks
                                  .Where(b => b.UserId ==
_userManager.GetUserId(User))
                                  .ToList();

        return View(art);
    }

}
```

Explicații!

- **Se injectează** *ISentimentAnalysisService* prin constructor
- **Metoda devine *async*** pentru a permite apelul asincron către API
- **Se apelează** *AnalyzeSentimentAsync* înainte de salvare
- **Se populează** câmpurile de sentiment dacă analiza a reușit

Pasul 7: Afișarea în View pentru Admini

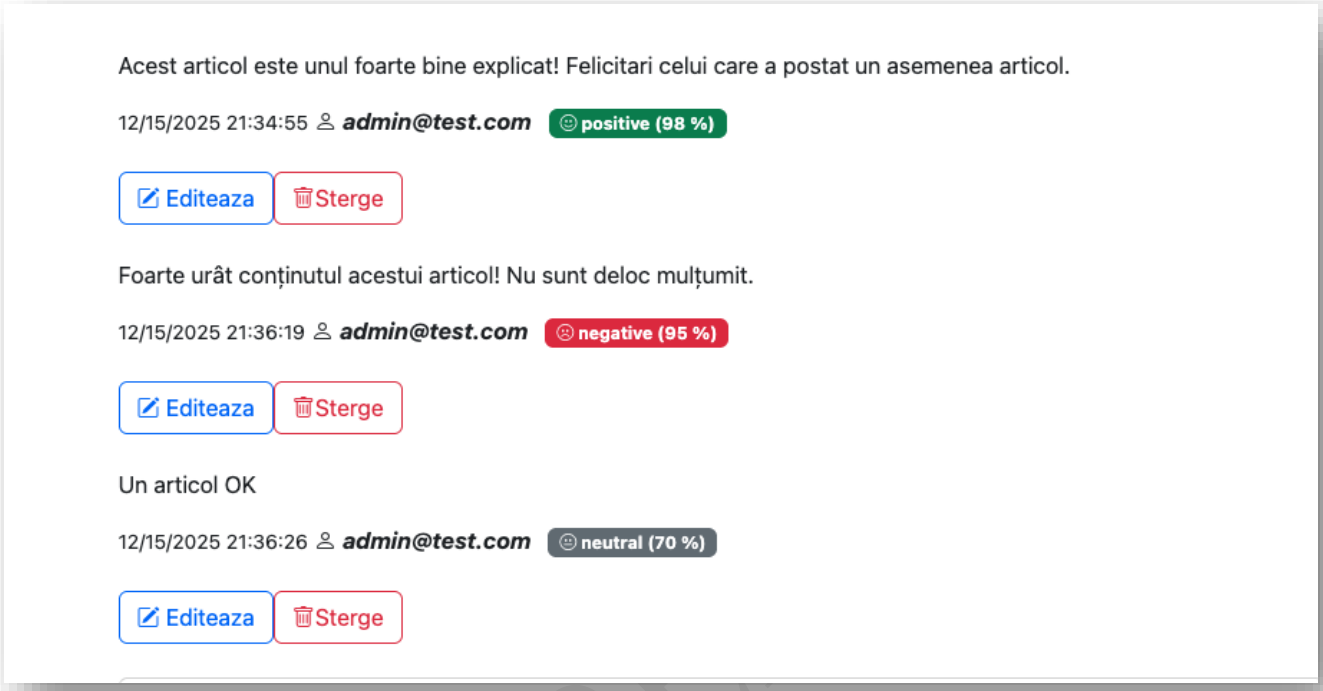
Se modifică view-ul pentru a afișa etichetele de sentiment, vizibile doar pentru administratori.

Views/Articles/Show.cshtml

```
@* Afisare comentarii impreuna cu butoanele de editare si
stergere *@

@foreach (var comm in Model.Comments)
{
    <div class="container">
        <div class="row">
            <div class="col-md-2"></div>

            <div class="col-md-8">

                <div>

                    <p>@comm.Content</p>
                    <small>@comm.Date</small>
                    <strong><i class="bi bi-person">
@comm.User.UserName</i></strong>

                    @* Afisare eticheta sentiment - vizibila
doar pentru Admin *@

                    @if (ViewBag.EsteAdmin == true &&
comm.SentimentLabel != null)
                    {
                        var badgeClass = comm.SentimentLabel
switch
                        {
                            "positive" => "bg-success",
                            "negative" => "bg-danger",
                            _ => "bg-secondary"
                        };

                        var sentimentIcon = comm.SentimentLabel
switch
                        {
                            "positive" => "bi-emoji-smile",
                            "negative" => "bi-emoji-frown",
                            _ => "bi-emoji-neutral"
                        };

                        <span class="badge @badgeClass ms-2">
                            <i class="bi @sentimentIcon"></i>
                            @comm.SentimentLabel

(@(comm.SentimentConfidence?.ToString("P0")))
                        </span>
```

```
                    }

                </div>
                <br>

                @if (comm.UserId == ViewBag.UserCurent ||
ViewBag.EsteAdmin == true)
                {
                    <div class="d-flex">
                        <div>
                            <a class="btn btn-outline-primary"
asp-controller="Comments" asp-action="Edit" asp-route-
id="@comm.Id">
                                <i class="bi bi-pencil-
square"></i> Editeaza
                            </a>

                        </div>
                        <div>
                            <form method="post" asp-
controller="Comments" asp-action="Delete" asp-route-
id="@comm.Id">

                                <button class="btn btn-outline-
danger" type="submit"><i class="bi bi-trash"></i>Sterge</button>

                            </form>
                        </div>
                    </div>
                }

                <br />

            </div>

            <div class="col-md-2"></div>

        </div>

    </div>
}
```

Exemplu Afișare:



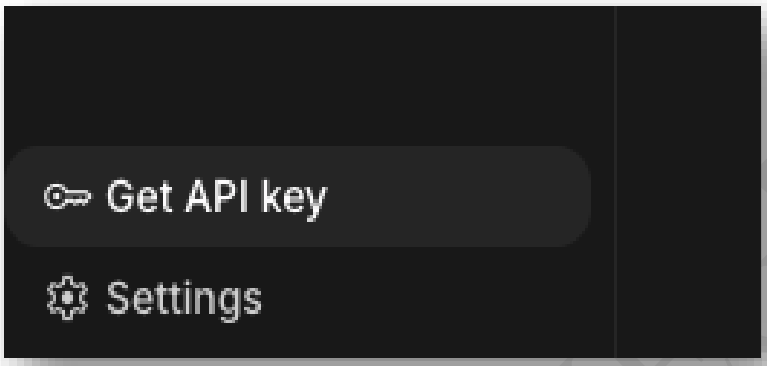
Resurse Suplimentare

- *Documentatie OpenAI API*
- *ASP.NET Core Dependency Injection*

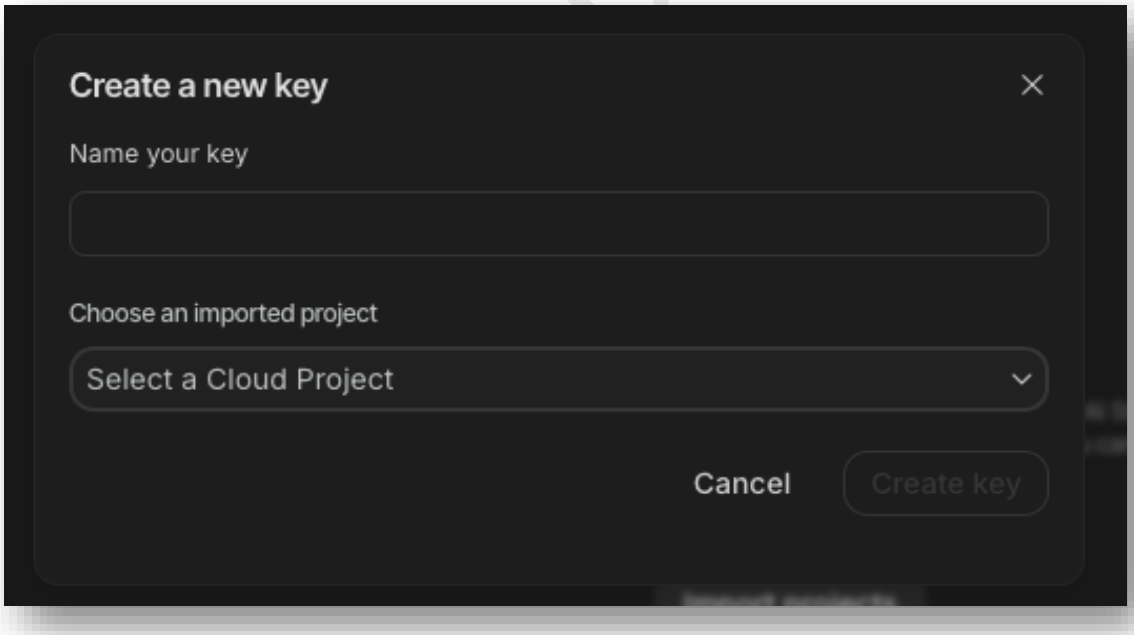
Schimbarea serviciului AI cu un alt serviciu

Google: https://aistudio.google.com/

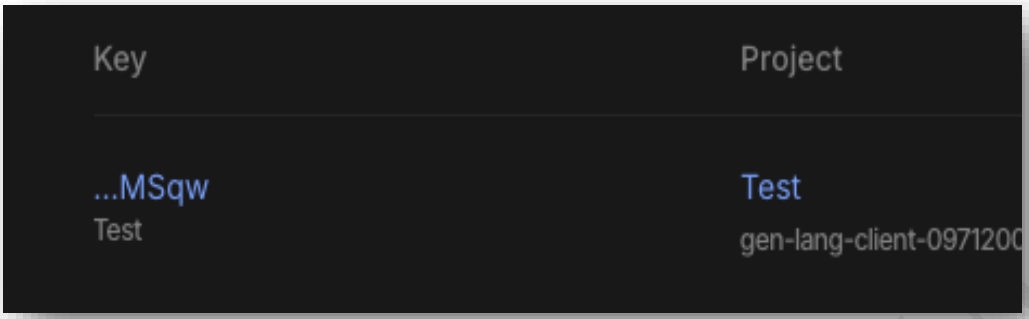
Se accesează meniul GET API KEY



Se adaugă un nume pentru API KEY și se creează un proiect



Se copiază API key



Exemplu de folosire cu Google AI:

Unul din modelele cu disponibilitate limitată și gratuită este: **gemini-2.5-flash-lite**

La acest URL se găsește o listă cu modelele disponibile gratuit și limitele lor: https://aistudio.google.com/usage?timeRange=last-28-days

Pentru a utiliza **Gemini 2.5 Flash Lite**, trebuie adăugat un serviciu nou pentru request-ul specific API-ului Google. Astfel, se adaugă în folderul **Services** fișierul **GoogleSentimentAnalysisService.cs** cu următorul conținut:

```
using System.Net.Http.Headers;
using System.Text;
using System.Text.Json;
using System.Text.Json.Serialization;

namespace ArticlesApp.Services
{

//
============================================================================
// SERVICIU DE ANALIZĂ SENTIMENT FOLOSIND GOOGLE AI (GEMINI)
//
============================================================================
//
// Acest fișier conține implementarea serviciului de analiză sentiment
// folosind Google Generative AI (Gemini) în loc de OpenAI.
//
```

```
// PAȘI PENTRU A SCHIMBA DE LA OPENAI LA GOOGLE AI:
//
================================================================
//
// 1. În fișierul appsettings.json, se adaugă configurația pentru
Google AI:
//
//    "GoogleAI": {
//        "ApiKey": "CHEIA_TA_API_GOOGLE"
//    }
//
// 2. În fișierul Program.cs, se schimbă înregistrarea serviciului:
//
//    // ÎNAINTE (OpenAI):
//    // builder.Services.AddScoped<ISentimentAnalysisService,
SentimentAnalysisService>();
//
//    // DUPĂ (Google AI):
//    builder.Services.AddScoped<ISentimentAnalysisService,
GoogleSentimentAnalysisService>();
//
// 3. Asigurați-vă că aveți o cheie API validă de la Google AI Studio:
//    https://aistudio.google.com/app/apikey
//
//
================================================================

/// <summary>
/// Implementarea serviciului de analiză sentiment folosind Google AI
(Gemini)
/// Această clasă implementează aceeași interfață
ISentimentAnalysisService
/// pentru a permite schimbarea ușoară între provideri (OpenAI/Google)
/// </summary>

    public class GoogleSentimentAnalysisService :
ISentimentAnalysisService
    {
        private readonly HttpClient _httpClient;
        private readonly string _apiKey;
        private readonly ILogger<GoogleSentimentAnalysisService>
_logger;

        // URL-ul de bază pentru API-ul Google Generative AI
        private const string BaseUrl =
"https://generativelanguage.googleapis.com/v1beta/models/";

        // Modelul folosit - gemini-2.5-flash-lite
        private const string ModelName = "gemini-2.5-flash-lite";
```

```
        public GoogleSentimentAnalysisService(IConfiguration
configuration, ILogger<GoogleSentimentAnalysisService> logger)
        {
            _httpClient = new HttpClient();

            // Citim cheia API din configurație
            // Asigurați-vă că ați adăugat "GoogleAI:ApiKey" în
appsettings.json
            _apiKey = configuration["GoogleAI:ApiKey"]
                ?? throw new ArgumentNullException("GoogleAI:ApiKey nu
este configurat în appsettings.json");

            _logger = logger;

            // Configurare HttpClient pentru Google AI API
            _httpClient.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
        }

        /// <summary>
        /// Analizează sentimentul unui text folosind Google AI
(Gemini)
        /// </summary>
        /// <param name="text">Textul de analizat</param>
        /// <returns>Rezultatul analizei de sentiment</returns>
        public async Task<SentimentResult>
AnalyzeSentimentAsync(string text)
        {
            try
            {
                // Construim prompt-ul pentru analiza de sentiment
                // Același format ca la OpenAI pentru consistență
                var prompt = $@"You are a sentiment analysis
assistant. Analyze the sentiment of the given text and respond ONLY
with a JSON object in this exact format:
{{""label"": ""positive|neutral|negative"", ""confidence"": 0.0-1.0}}

Rules:
- label must be exactly one of: positive, neutral, negative
- confidence must be a number between 0.0 and 1.0
- Do not include any other text, only the JSON object

Analyze the sentiment of this comment: ""{text}""";

                // Construim request-ul pentru Google AI API
                // Structura este diferită față de OpenAI - folosim
"contents" și "parts"
                var requestBody = new GoogleAiRequest
                {
                    Contents = new List<GoogleAiContent>
                    {
                        new GoogleAiContent
                        {
```

```
                        Parts = new List<GoogleAiPart>
                        {
                            new GoogleAiPart { Text = prompt }
                        }
                    }
                },
                // Configurări pentru generare - temperature
scăzută pentru rezultate consistente
                GenerationConfig = new GoogleAiGenerationConfig
                {
                    Temperature = 0.1,
                    MaxOutputTokens = 100
                }
            };

            var jsonContent =
JsonSerializer.Serialize(requestBody, new JsonSerializerOptions
            {
                PropertyNamingPolicy = JsonNamingPolicy.CamelCase
            });

            var content = new StringContent(jsonContent,
Encoding.UTF8, "application/json");

            // Construim URL-ul complet cu cheia API ca parametru
            // Google AI folosește X-goog-api-key sau parametru în
URL
            var requestUrl =
$"{BaseUrl}{ModelName}:generateContent?key={_apiKey}";

            _logger.LogInformation("Trimitem cererea de analiză
sentiment către Google AI API");

            // Trimitem request-ul către Google AI API
            var response = await _httpClient.PostAsync(requestUrl,
content);
            var responseContent = await
response.Content.ReadAsStringAsync();

            if (!response.IsSuccessStatusCode)
            {
                _logger.LogError("Eroare Google AI API:
{StatusCode} - {Content}", response.StatusCode, responseContent);
                return new SentimentResult
                {
                    Success = false,
                    ErrorMessage = $"Eroare API:
{response.StatusCode}"
                };
            }
```

```
                // Parsăm răspunsul de la Google AI
                var googleResponse =
JsonSerializer.Deserialize<GoogleAiResponse>(responseContent, new
JsonSerializerOptions
                {
                    PropertyNameCaseInsensitive = true
                });

                // Extragem textul din răspuns
                // Structura: candidates[0].content.parts[0].text
                var assistantMessage =
googleResponse?.Candidates?.FirstOrDefault()?.Content?.Parts?.FirstOrD
efault()?.Text;

                if (string.IsNullOrEmpty(assistantMessage))
                {
                    return new SentimentResult
                    {
                        Success = false,
                        ErrorMessage = "Răspuns gol de la API"
                    };
                }

                _logger.LogInformation("Răspuns Google AI:
{Response}", assistantMessage);

                // Curățăm răspunsul de eventuale caractere markdown
(```json ... ```)
                var cleanedResponse =
CleanJsonResponse(assistantMessage);

                // Parsăm JSON-ul din răspunsul asistentului
                var sentimentData =
JsonSerializer.Deserialize<SentimentResponse>(cleanedResponse);

                if (sentimentData == null)
                {
                    return new SentimentResult
                    {
                        Success = false,
                        ErrorMessage = "Nu s-a putut parsa răspunsul
sentiment"
                    };
                }

                // Validăm și normalizăm label-ul
                var label = sentimentData.Label?.ToLower() switch
                {
                    "positive" => "positive",
                    "negative" => "negative",
                    _ => "neutral"
                };
```

```
                // Validăm confidence score
                var confidence = Math.Clamp(sentimentData.Confidence,
0.0, 1.0);

                return new SentimentResult
                {
                    Label = label,
                    Confidence = confidence,
                    Success = true
                };
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Eroare la analiza
sentimentului");
                return new SentimentResult
                {
                    Success = false,
                    ErrorMessage = ex.Message
                };
            }
        }

        /// <summary>
        /// Curăță răspunsul JSON de eventuale caractere markdown
        /// Gemini poate returna răspunsul înconjurat de ```json ...
```
```
        /// </summary>
        private string CleanJsonResponse(string response)
        {
            var cleaned = response.Trim();

            // Eliminăm blocurile de cod markdown dacă există
            if (cleaned.StartsWith("```json"))
            {
                cleaned = cleaned.Substring(7);
            }
            else if (cleaned.StartsWith("```"))
            {
                cleaned = cleaned.Substring(3);
            }

            if (cleaned.EndsWith("```"))
            {
                cleaned = cleaned.Substring(0, cleaned.Length - 3);
            }

            return cleaned.Trim();
        }
    }
```

```
    //
================================================================
    // CLASE PENTRU SERIALIZAREA/DESERIALIZAREA RĂSPUNSURILOR GOOGLE
AI
    //
================================================================
    //
    // Structura request-ului Google AI:
    // {
    //   "contents": [
    //     {
    //       "parts": [
    //         { "text": "..." }
    //       ]
    //     }
    //   ]
    // }
    //
    // Structura răspunsului Google AI:
    // {
    //   "candidates": [
    //     {
    //       "content": {
    //         "parts": [
    //           { "text": "..." }
    //         ]
    //       }
    //     }
    //   ]
    // }
    //
================================================================

    /// <summary>
    /// Clasa pentru request-ul către Google AI
    /// </summary>
    public class GoogleAiRequest
    {
        [JsonPropertyName("contents")]
        public List<GoogleAiContent> Contents { get; set; } = new();

        [JsonPropertyName("generationConfig")]
        public GoogleAiGenerationConfig? GenerationConfig { get; set;
}
    }
```

```
    /// <summary>
    /// Conținutul mesajului pentru Google AI
    /// </summary>
    public class GoogleAiContent
    {
        [JsonPropertyName("parts")]
        public List<GoogleAiPart> Parts { get; set; } = new();
    }

    /// <summary>
    /// O parte din conținut (text, imagine, etc.)
    /// </summary>
    public class GoogleAiPart
    {
        [JsonPropertyName("text")]
        public string Text { get; set; } = string.Empty;
    }

    /// <summary>
    /// Configurări pentru generarea răspunsului
    /// </summary>
    public class GoogleAiGenerationConfig
    {
        [JsonPropertyName("temperature")]
        public double Temperature { get; set; } = 0.7;

        [JsonPropertyName("maxOutputTokens")]
        public int MaxOutputTokens { get; set; } = 1024;
    }

    /// <summary>
    /// Răspunsul de la Google AI API
    /// </summary>
    public class GoogleAiResponse
    {
        [JsonPropertyName("candidates")]
        public List<GoogleAiCandidate>? Candidates { get; set; }
    }

    /// <summary>
    /// Un candidat din răspuns (Google AI poate returna mai mulți
candidați)
    /// </summary>
    public class GoogleAiCandidate
    {
        [JsonPropertyName("content")]
        public GoogleAiContent? Content { get; set; }
    }
}
```

În appsettings.json, se adaugă:

```
"GoogleAI": {
   "ApiKey": "cheia ta api"
}
```

În Program.cs, se schimbă înregistrarea serviciului:

Se schimbă serviciul:
builder.Services.AddScoped<ISentimentAnalysisService, SentimentAnalysisService>();

Cu cel de la Google:
builder.Services.AddScoped<ISentimentAnalysisService, GoogleSentimentAnalysisService>();

Surse modele AI:

- https://huggingface.co/
- Anthropic
- **Open AI**
- **Google**
- DeepSeek
- Mistral