Київський національний університет ім. Тараса Шевченка

Кафедра мережевих та інтернет технологій

Лабораторна робота № 7

Дисципліна: Хмарні технології

Тема: Створення застосунку виявлення особистої ідентифікаційної та медичної інформації із застосуванням Azure AI Language

Виконав: Студент групи МІТ-31

Пугач Назар

Мета: Створити застосунок для виявлення особистої ідентифікаційної та медичної інформації із застосуванням Azure AI Language

Хід роботи

Завдання 7.1: Створити Azure AI Language сервіс.

Ми створювали його в попередній лабораторні роботі.

Завдання 7.2: Доповнити створений на попередньому занятті Web-застосунок новими сторінками, на яких користувачу пропонується ввести текст та розпізнати особисту ідентифікаційну інформацію та медичну інформацію. Результати вивести у вигляді тексту із прихованою особистою ідентифікаційною інформацією та у формі таблиці із якомога повнішими данами про медичну інформацію.

Спочатку створимо модель.

```
public class HealthTextModel
{
    public string InputText { get; set; }
    public string OutputText { get; set; }
    public List<PII> PIIs { get; set; }
    public List<MedicalEntity> MedicalEntities { get; set; }
    public List<MedicalRelation> MedicalRelations { get; set; }
}

public class MedicalRelation
{
    public string RelationType { get; set; }
    public List<MedicalRole> Roles { get; set; } = new();
}
public class MedicalRole
{
    public string RoleName { get; set; }
    public string EntityText { get; set; }
    public string EntityCategory { get; set; }
}

public class MedicalEntity
{
    public string Text { get; set; }
    public string Category { get; set; }
    public double ConfidenceScore { get; set; }
}

public class PII
{
    public string Text { get; set; }
    public string Category { get; set; }
    public string SubCategory { get; set; }
    public double ConfidenceScore { get; set; }
}
```

Рисунок 7.1 – Модель.

Створимо контролер.

```
using Azure.AI.TextAnalytics;
using lab6.Models;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

```
namespace lab6.Controllers
{
    public class HealthTextController : Controller
    {
        private readonly TextAnalyticsClient _textAnalyticsClient;

        public HealthTextController(TextAnalyticsClient textAnalyticsClient)
        {
            _textAnalyticsClient = textAnalyticsClient;
        }

        public IActionResult Index()
        {
            return View(new HealthTextModel());
        }

        [HttpPost]
        public async Task<IActionResult> Analyze(HealthTextModel model)
        {
            if (string.IsNullOrWhiteSpace(model.InputText))
            {
                ModelState.AddModelError("InputText", "Please enter some
text.");
                return View("Index", model);
            }

            await ProcessText(model);
            return View("Index", model);
        }

        private async Task ProcessText(HealthTextModel model)
        {
            string processedText = model.InputText;
            var piiEntities = await ExtractPiiEntities(model.InputText);
            model.PIIs = piiEntities;
            processedText = ReplacePiiEntities(processedText, piiEntities);

            var (medicalEntities, medmedicalRelations) = await
ExtractMedicalEntities(model.InputText);
            model.MedicalEntities = medicalEntities;
            model.MedicalRelations = medmedicalRelations;
            processedText = HighlightMedicalEntities(processedText,
medicalEntities);

            model.OutputText = NormalizeText(processedText);
        }

        private async Task<List<PII>> ExtractPiiEntities(string inputText)
        {
            var piiEntities = new List<PII>();
            var piiResponse = await
_textAnalyticsClient.RecognizePiiEntitiesAsync(inputText);

            if (piiResponse.Value != null)
            {
                piiEntities = piiResponse.Value.Select(entity => new PII
                {
                    Text = entity.Text,
                    Category = entity.Category.ToString(),
                    SubCategory = entity.SubCategory?.ToString() ?? "N/A",
                    ConfidenceScore = entity.ConfidenceScore
                }).ToList();
            }

            return piiEntities;
```

```
        }

        private string ReplacePiiEntities(string inputText, List<PII>
piiEntities)
        {
            string processedText = inputText;

            foreach (var entity in piiEntities)
            {
                string entityWithSymbols = string.Join("%%",
entity.Text.ToCharArray());

                string hiddenText = $"<span class='bg-warning badge text-
dark' title='{entityWithSymbols}'>[{entity.Category}]</span>";
                processedText = processedText.Replace(entity.Text,
hiddenText);
            }
            return processedText;
        }


        private async Task<(List<MedicalEntity>, List<MedicalRelation>)>
ExtractMedicalEntities(string inputText)
        {
            var medicalEntities = new List<MedicalEntity>();
            var medicalRelations = new List<MedicalRelation>();

            List<string> batchInput = new List<string> { inputText };
            AnalyzeHealthcareEntitiesOperation healthOperation = await
_textAnalyticsClient.StartAnalyzeHealthcareEntitiesAsync(batchInput);
            await healthOperation.WaitForCompletionAsync();

            await foreach (var result in healthOperation.Value)
            {
                foreach (AnalyzeHealthcareEntitiesResult entitiesInDoc in
result)
                {
                    if (!entitiesInDoc.HasError)
                    {
                        foreach (var entity in entitiesInDoc.Entities)
                        {
                            medicalEntities.Add(new MedicalEntity
                            {
                                Text = entity.Text,
                                Category = entity.Category.ToString(),
                                ConfidenceScore = entity.ConfidenceScore
                            });
                        }
                        foreach (var relation in
entitiesInDoc.EntityRelations)
                        {
                            var medicalRelation = new MedicalRelation
                            {
                                RelationType =
relation.RelationType.ToString(),
                                Roles = relation.Roles.Select(role => new
MedicalRole
                                {
                                    RoleName = role.Name,
                                    EntityText = role.Entity.Text,
                                    EntityCategory =
role.Entity.Category.ToString()
                                }).ToList()
                            };
                            medicalRelations.Add(medicalRelation);
```

```
                        }
                    }
                }
            }
            return (medicalEntities, medicalRelations);
        }

        private string HighlightMedicalEntities(string inputText,
List<MedicalEntity> medicalEntities)
        {
            string processedText = inputText;
            foreach (var entity in medicalEntities)
            {
                string entityWithSymbols = string.Join("%%",
entity.Text.ToCharArray());

                string highlightedText = $"<span class='badge bg-success
text-light' title='{entity.Category}'>{entityWithSymbols}</span>";
                processedText = processedText.Replace(entity.Text,
highlightedText);
            }
            return processedText;
        }
        private string NormalizeText(string inputText)
        {
            return inputText.Replace("%%", "");
        }
    }
}
```

Рисунок 7.2 – Контролер.

Створимо відображення.

```
@model lab6.Models.HealthTextModel

@{
    ViewData["Title"] = "Health Text Analysis";
}

<h1>@ViewData["Title"]</h1>

<form asp-action="Analyze" method="post">
    <div class="form-group">
        <label for="InputText">Input Text</label>
        <textarea id="InputText" name="InputText" class="form-control"
rows="5">@Model.InputText</textarea>
        @if (ViewData.ModelState["InputText"]?.Errors.Count > 0)
        {
            <span class="text-
danger">@ViewData.ModelState["InputText"].Errors[0].ErrorMessage</span>
        }
    </div>

    <button type="submit" class="btn btn-primary">Analyze</button>
</form>

@if (!string.IsNullOrWhiteSpace(Model.OutputText))
{
    <hr />
    <h3>Processed Text</h3>
    <div>@Html.Raw(Model.OutputText)</div>

    @if (Model.MedicalEntities != null && Model.MedicalEntities.Any()){
        <hr />
```

```
        <h3>Extracted Medical Information</h3>
        <table class="table table-bordered">
            <thead>
                <tr>
                    <th>Entity</th>
                    <th>Category</th>
                    <th>Confidence Score</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var entity in Model.MedicalEntities)
                {
                    <tr>
                        <td>@entity.Text</td>
                        <td>@entity.Category</td>
                        <td>@entity.ConfidenceScore.ToString("P1")</td>
                    </tr>
                }
            </tbody>
        </table>
    }
    @if (Model.MedicalRelations != null && Model.MedicalRelations.Any())
    {
        <hr />
        <h3>Extracted Medical Relations</h3>
        <table class="table table-bordered">
            <thead>
                <tr>
                    <th>Relation Type</th>
                    <th>Role Name</th>
                    <th>Entity</th>
                    <th>Category</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var relation in Model.MedicalRelations)
                {
                    bool firstRow = true;
                    @foreach (var role in relation.Roles)
                    {
                        <tr>
                            @if (firstRow)
                            {
                                <td
rowspan="@relation.Roles.Count">@relation.RelationType</td>
                                firstRow = false;
                            }
                            <td>@role.RoleName</td>
                            <td>@role.EntityText</td>
                            <td>@role.EntityCategory</td>
                        </tr>
                    }
                }
            </tbody>
        </table>
    }
}
```

Рисунок 7.3 – Відобарження.

Додамо ключі в appsettings.json.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
```

```
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AzureAI": {
    "LanguageKey": "lanKey",
    "LanguageEndpoint": "lanEndpoint"
  },
  "AllowedHosts": "*"
}
```

Рисунок 7.4 – Ключі.

Завдання 7.3: Перевірка працездатності.

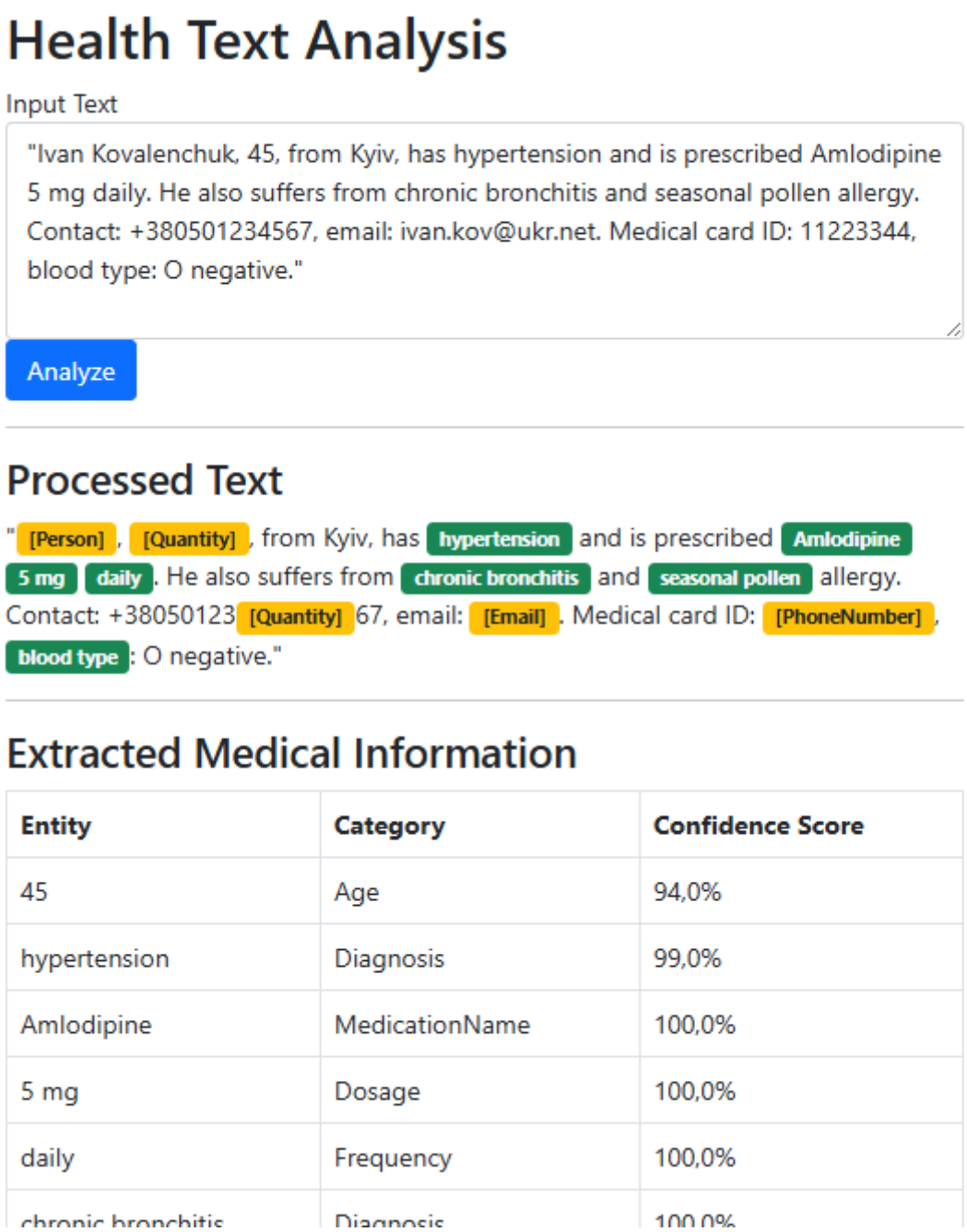

Рисунок 7.5 – Результат.

Висновок: Під час виконання лабораторної робити я додав до застосунку функціонал для виявлення особистої ідентифікаційної та медичної інформації із застосуванням Azure AI Language.