

Modular RAGs

Finn Vilsbæk

fvil@eadania.dk

Subjects for today

- Changes to the example code base
- Coding Challenge

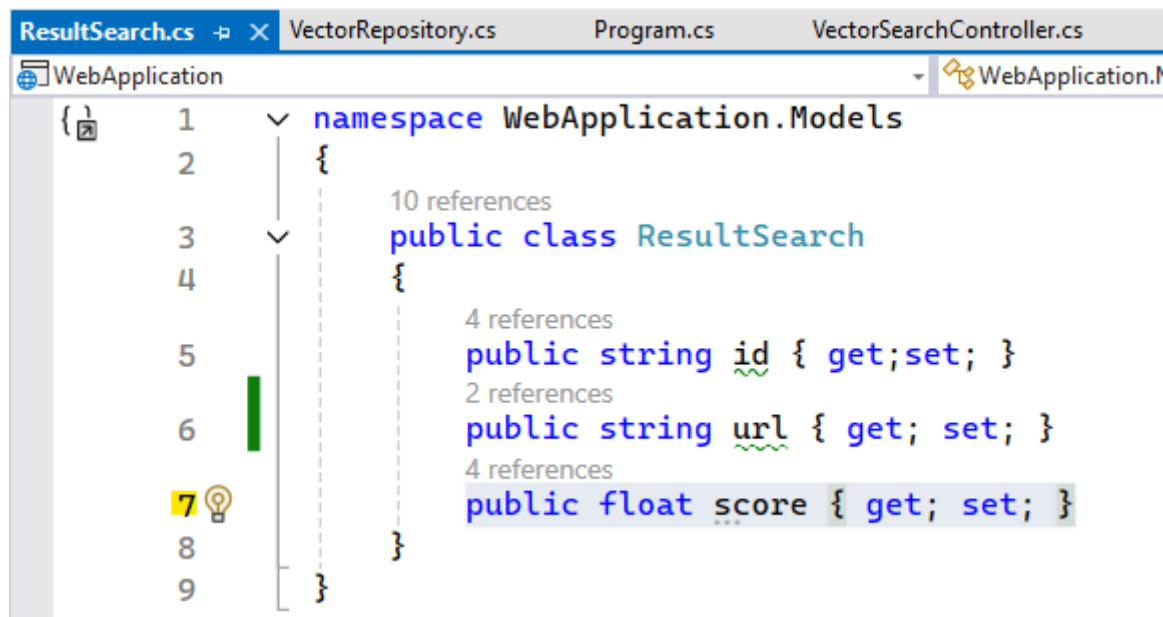
Modular RAGs

- We are going to make some changes to the code base, in order to enhance our web application with text output from two to three different LLM's.
- We will start with the VectorRepository class in the Models Folder. Comment out or remove line 45, and add the score line at line 57.

```
VectorRepository.cs | Program.cs | VectorSearchController.cs | AzureOpenAiService.cs | MistralService.cs | appsettings.json | version
WebApplication | WebApplication.Models.VectorRepository | version
41
42 message.Headers.Add("api-key", conf["AzureAiSearchKey"]);
43
44 var response = await client.SendAsync(message);
45 //response.EnsureSuccessStatusCode();
46 var responseJson = await response.Content.ReadAsStringAsync();
47 var o = JObject.Parse(responseJson);
48
49 var res = new List<ResultSearch>();
50
51 foreach (var result in o.GetValue("value").AsEnumerable())
52 {
53     res.Add(
54         new ResultSearch() {
55             id = ((JObject)result).GetValue("id").ToString(),
56             url = ((JObject)result).GetValue("url").ToString(),
57             score = float.Parse(((JObject)result).GetValue("@search.score").ToString())
58         }
59     );
60 }
61
62 return res;
63
64 ..
```

Modular RAGs

- Now, we'll add the score to the ResultSearch.cs file at line 7. ResultSearch.cs also lives in the models folder.



```
ResultSearch.cs  VectorRepository.cs  Program.cs  VectorSearchController.cs
WebApplication  WebApplication.I
{
1  namespace WebApplication.Models
2  {
3  10 references
4  public class ResultSearch
5  {
6  4 references
7  public string id { get; set; }
8  2 references
9  public string url { get; set; }
10 4 references
11 public float score { get; set; }
12 }
13 }
```

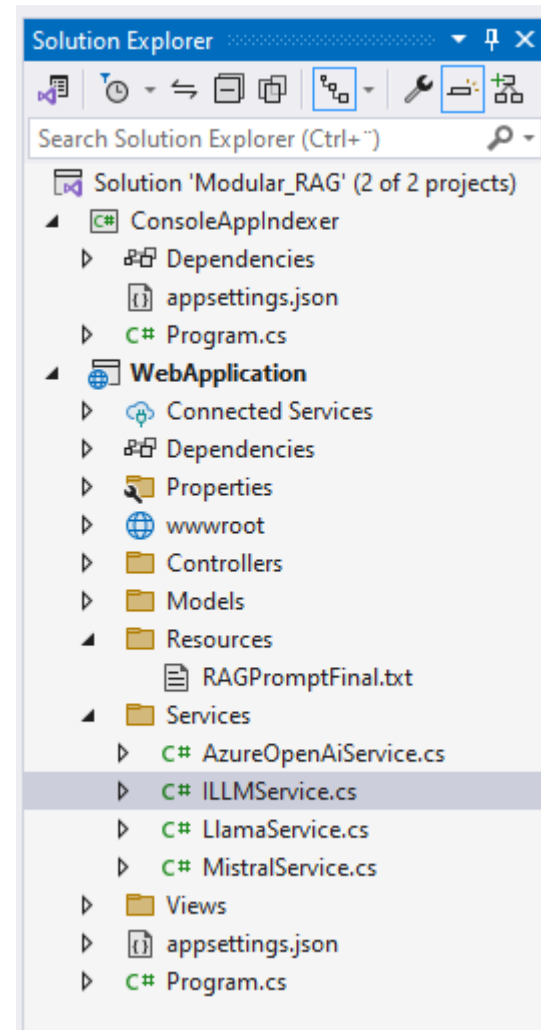
Modular RAGs

- We will now add two folders to the web application project: Services and Resources.

In the Resources folder, add an empty text file named 'RAGPromptFinal' – the text for this file will be supplied later.

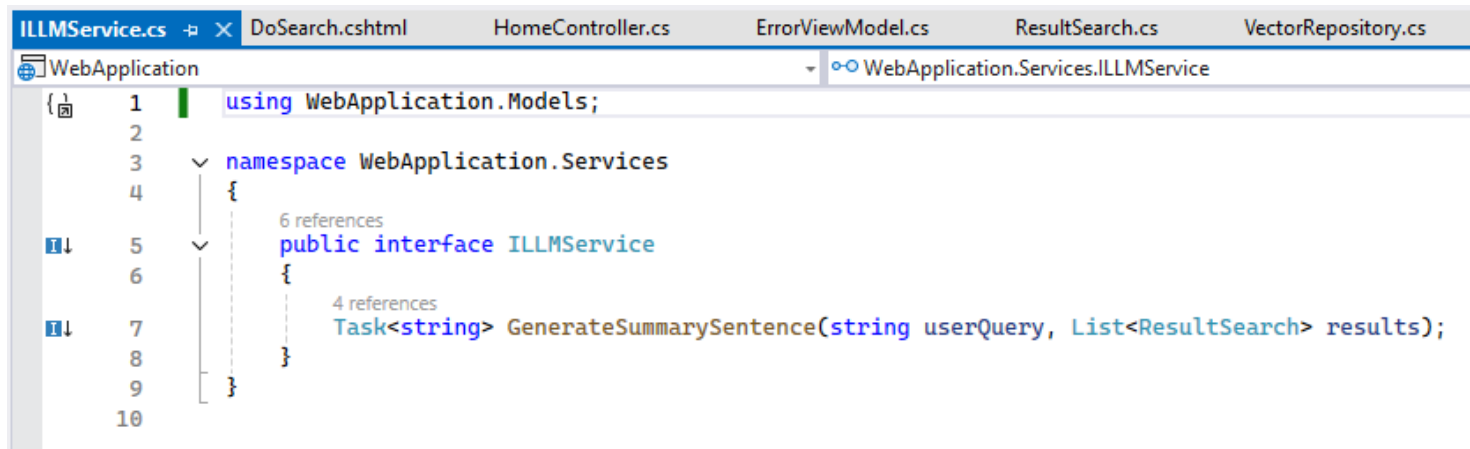
In Services, add four empty classes:

AzureOpenAIService.cs,
ILLMService.cs,
LLamaService.cs, and finally
MistralService.cs.



Modular RAGs

- Here is the code block for ILLMService.cs:



```
1  using WebApplication.Models;
2
3  namespace WebApplication.Services
4  {
5      public interface ILLMService
6      {
7          Task<string> GenerateSummarySentence(string userQuery, List<ResultSearch> results);
8      }
9  }
10
```

Modular RAGs

- Changes to VectorSearchController.cs in the Controllers folder:

```
search.cs  VectorRepository.cs  Program.cs  VectorSearchController.cs  AzureOpenAiService.cs  MistralService.cs  appsettings.json
Application  WebApplication.Controllers.VectorSearchController  DoSearch(string searchPrompt)

3 references
7 public class VectorSearchController : Controller
8 {
9     private readonly ILogger<VectorSearchController> _logger;
10    private readonly IConfiguration configuration;
11    private readonly ILLMService llmservice;
12
13    0 references
14    public VectorSearchController(ILogger<VectorSearchController> logger, IConfiguration configuration, ILLMService llmservice)
15    {
16        _logger = logger;
17        this.configuration = configuration;
18        this.llmservice = llmservice;
19    }
20
21    0 references
22    public IActionResult Index()
23    {
24        return View();
25    }
26
27    [HttpPost()]
28    0 references
29    public async Task<IActionResult> DoSearch(string searchPrompt)
30    {
31        var res = await VectorRepository.DoVectorSearch(searchPrompt, this.configuration);
32        // last part of the RAG - generate a user friendly message
33        var sentence = await this.llmservice.GenerateSummarySentence(searchPrompt, res);
34        this.ViewBag.Sentence = sentence;
35
36        return View(res);
37    }
38 }
39 }
```

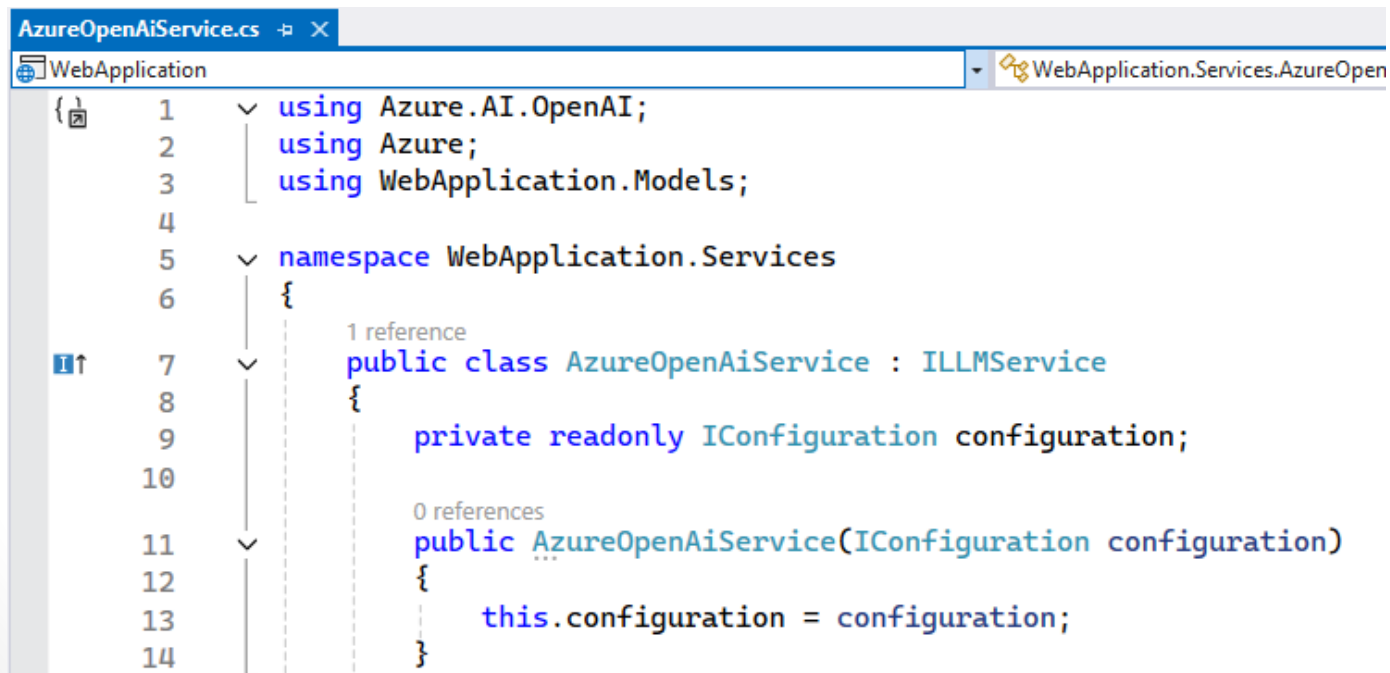
Modular RAGs

- Changes to DoSearch.cshtml in the Views >> VectorSearch folder:

```
DoSearch.cshtml  HomeController.cs  ErrorViewModel.cs  ResultSearch.cs  VectorRepositor
C# WebApplication
1  @model List<ResultSearch>
2  @{
3      ViewData["Title"] = "Vector Search";
4  }
5
6  @using (Html.BeginForm("DoSearch", "VectorSearch", FormMethod.Post))
7  {
8      @Html.TextBox("searchPrompt", "")
9      <input type="submit" value="Search" />
10 }
11
12 <div id="results">
13
14     @if(this.ViewData["sentence"] != null)
15     {
16         <div>
17             @this.ViewData["sentence"].ToString()
18         </div>
19     }
20
21     @if(this.Model != null)
22     {
23         @foreach(var r in this.Model)
24         {
25             @Html.Raw($"<img src='{r.url}' class='result'/>");
26         }
27     }
28 </div>
```


Modular RAGs

- Now we will build a central part of the Web Application: the class file `AzureOpenAiService.cs`, which we previously set up as an empty class in the folder 'Services'. Start by entering the following lines of code in the editor - and pay attention to the line numbers on the left side in the following slides, as a guide to where you are in the file when building out the class.



```
AzureOpenAiService.cs
WebApplication
WebApplication.Services.AzureOpen
1 using Azure.AI.OpenAI;
2 using Azure;
3 using WebApplication.Models;
4
5 namespace WebApplication.Services
6 {
7     public class AzureOpenAiService : ILLMService
8     {
9         private readonly IConfiguration configuration;
10
11         public AzureOpenAiService(IConfiguration configuration)
12         {
13             this.configuration = configuration;
14     }
```

Modular RAGs

- I am a lazy loader kind of guy, and I rather enjoy the concept of just-in-time information, so I thought I would ask ChatGPT to explain the three main parts of the class for us, as seen below. The constructor method for any C# class is distinguished by having the same name as the class, and **no return type** – not even void.
- In our specific case here, the constructor serves to ensure that we cannot instantiate the class without passing in the required parameter *configuration* of type *IConfiguration*. The configuration object passed in here contains the details of endpoint url and access key to the service, which we get from the file appsettings.json.

1. Constructor: `AzureOpenAiService(IConfiguration configuration)`

- **Purpose:** Initializes the class by accepting a configuration parameter (often holding settings like API keys or endpoints).
- **Expected Result:** Sets up the service with the necessary configuration details, preparing it to connect to Azure OpenAI.

Modular RAGs

- The following async method will generate a summary sentence from the data returned from the LLM in response to the entered search term.

```
15 public async Task<string> GenerateSummarySentence(string userQuery, List<ResultSearch> results)
16 {
17     var prompt = Properties.Resources.RAGPromptFinal;
18
19     prompt = prompt.Replace("<QUERY/>", userQuery);
20
21     foreach (var result in results)
22     {
23         prompt = prompt.Replace("<IMAGE/>", $"name: '{result.id}', cosineSimilarityScore: {result.score}\r\n<IMAGE/>");
24     }
25
26     prompt = prompt.Replace("<IMAGE/>", "");
27
28     return await generateAsync(prompt, this.configuration);
29 }
```

2. Method: `GenerateSummarySentence`

- **Purpose:** This method generates a prompt for Azure's OpenAI model to create a summary sentence based on a user query and a list of search results (`ResultSearch` objects). It replaces placeholders in a pre-defined prompt with the query and search result details.
- **Expected Result:** Returns a generated sentence summarizing the user query and the search results, after calling the `generateAsync` helper method to get the response from the AI model.

Modular RAGs

- Finally, the private generateAsync method, which is called in the method GenerateSummarySentence from the previous slide will ask the OpenAI LLM model for a response to the query sent in by the user. The access modifier *private* means 'only for internal use inside the class'. This completes the AzureOpenAIService class.

```
31 private static async Task<string> generateAsync(string userPrompt, IConfiguration conf)
32 {
33     OpenAIClient client = new OpenAIClient(new Uri(conf["AzureOpenAIEndpoint"]), new AzureKeyCredential(conf["AzureOpenAIKey"]));
34
35     Response<ChatCompletions> responseWithoutStream = await client.GetChatCompletionsAsync("Panmedia-GPT-4o-mini",
36     new ChatCompletionsOptions()
37     {
38         Messages =
39         {
40             new ChatMessage(ChatRole.System, @"You are an AI assistant that helps people find information."),
41             new ChatMessage(ChatRole.User, userPrompt),
42         },
43         Temperature = (float)0.7,
44         MaxTokens = 800,
45
46         NucleusSamplingFactor = (float)0.95,
47         FrequencyPenalty = 0,
48         PresencePenalty = 0,
49     });
50
51     ChatCompletions response = responseWithoutStream.Value;
52
53     return response.Choices[0].Message.Content;
54 }
55 }
56 }
57 }
```

Modular RAGs

- A bit of explanation courtesy of ChatGPT for the generateAsync method:

3. Private Method: `generateAsync`

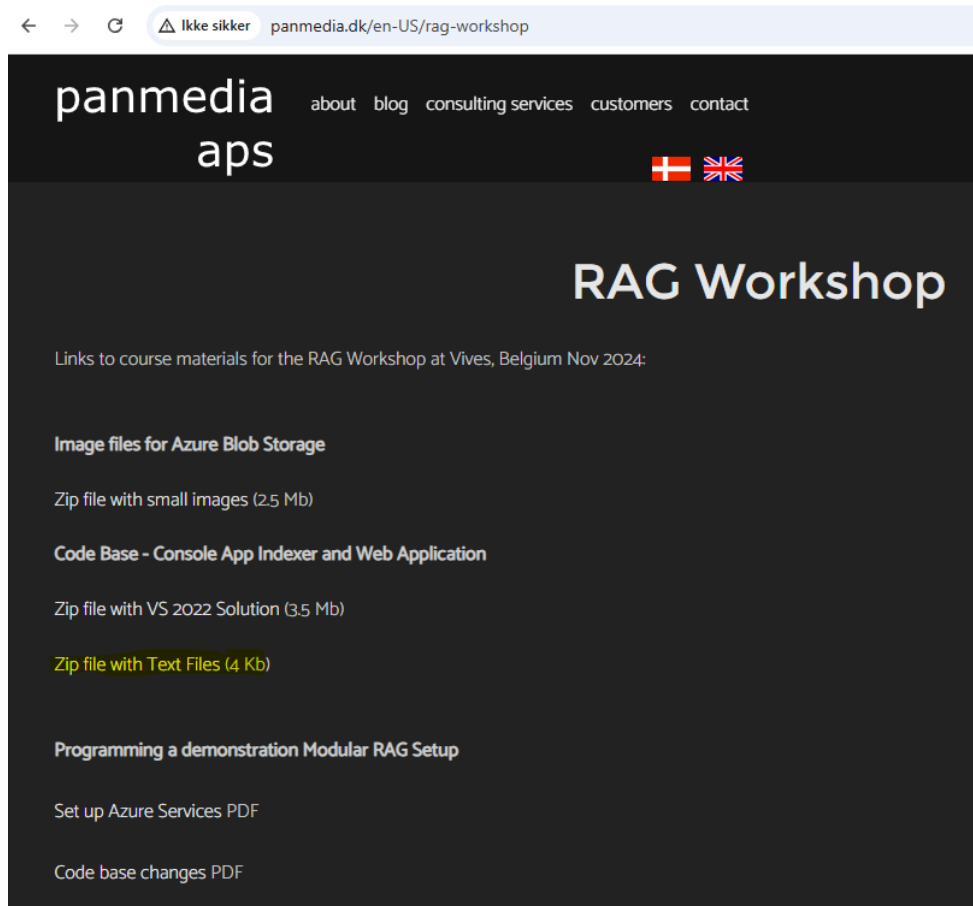
- **Purpose:** This helper method takes a prompt and configuration settings, initializes an OpenAI client, and sends the prompt to Azure OpenAI for processing. It creates a conversation context, sets parameters for AI response behavior (like `Temperature` and `MaxTokens`), and retrieves a response.
- **Expected Result:** Returns the AI-generated response content as a string, based on the prompt provided. This response will typically be a concise sentence or summary based on the user's query and search results.

This service class is designed to interact with the Azure OpenAI API, using a specific prompt format and configuration settings to generate AI-based summaries.

Modular RAGs

- <http://panmedia.dk/en-US/rag-workshop>

- The text for the remaining files RAGPromptFinal.txt as well as MistralService.cs can be found in the Text Files download zip at the above url.



Modular RAGs

- I have also supplied the keys for setting up endpoints in the bottom of the appsettings.json text file, as seen here:

```
appsettings.json  + X
Schema: https://json.schemastore.org/appsettings.json
15
16     "AzureOpenAIEndpoint": "https://panmediaservices.openai.azure.com",
17     "AzureOpenAIKey": "19reqLigD0QTd9ZSASk3JMKJzBrirhSQuBppDyipZN6dLqNBmdcNJQQJ99AKACfhMk5XJ3w3AAAAACOGRmoH",
18
19     "MistralEndpoint": "https://Mistral-Small-Panmedia.swedencentral.models.ai.azure.com/v1/chat/completions",
20     "MistralKey": "XDTsYkCprvS56rkRdONFunDqKy07jkaG",
21
22     "LlamaEndpoint": "https://Panmediallama.swedencentral.models.ai.azure.com/v1/chat/completions",
23     "LlamaKey": "gKpwILOHf5VGgRe0bd9Q9H7th7uzNDjr"
24
25     }
26
```

- Note that these three endpoints and keys are temporary Azure resources I have set up for this workshop. They will be deleted later on, so if you want to keep exploring how to set up and modify an example RAG system such as this one, I recommend that you research how to set up a hub and a project in Azure AI Services, where you can host your own models. You can follow the links at the end of this PDF to find out more about these subjects.

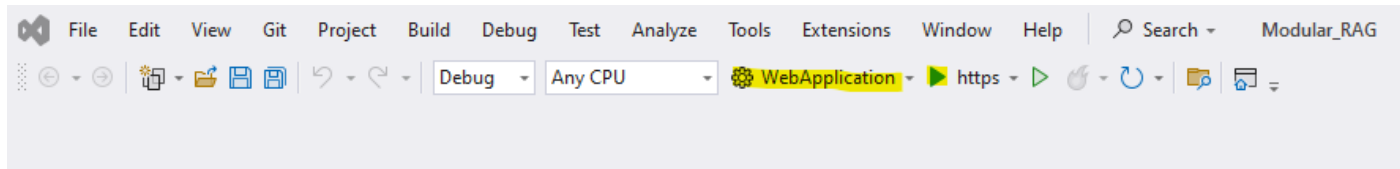
Modular RAGs

- Finally, we need to call one of the LLM Services as a singleton at the very start of the application. In Program.cs for the web application, add the lines 14, 15 and 16. Be sure to always comment out two of them at a time.

```
VectorRepository.cs Program.cs VectorSearchController.cs AzureOpenAiService.cs MistralService.cs appsettings.json
WebApplication WebApplication.Program
{
  1 using WebApplication.Services;
  2
  3 namespace WebApplication
  4 {
  5     0 references
  6     public class Program
  7     {
  8         0 references
  9         public static void Main(string[] args)
 10         {
 11             var builder = Microsoft.AspNetCore.Builder.WebApplication.CreateBuild
 12
 13             // Add services to the container.
 14             builder.Services.AddSingleton<ILLMService, AzureOpenAiService>();
 15             //builder.Services.AddSingleton<ILLMService, MistralService>();
 16             //builder.Services.AddSingleton<ILLMService, LlamaService>();
 17
 18             var app = builder.Build();
 19
```

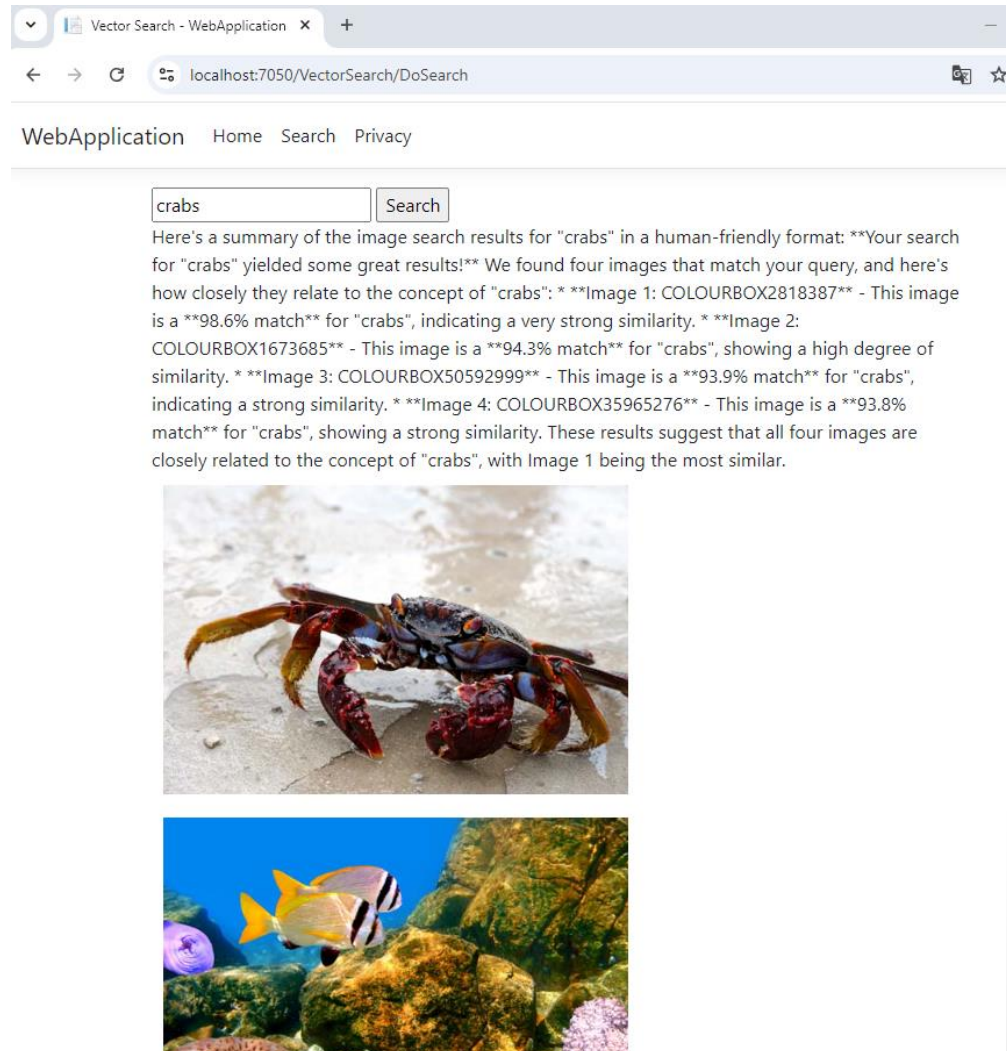

Modular RAGs

- Remember to point the solution default to the web app in the top dropdown before running it in debug mode in VS 2022:



Modular RAGs

- If everything is working as it should, you should now be able to see a text output from the model of choice along with the returned images from the previous version of the app:



The screenshot shows a web browser window with the title "Vector Search - WebApplication". The address bar displays "localhost:7050/VectorSearch/DoSearch". The page content includes a search bar with the text "crabs" and a "Search" button. Below the search bar, there is a text block providing a summary of image search results for "crabs". The text states: "Here's a summary of the image search results for 'crabs' in a human-friendly format: **Your search for 'crabs' yielded some great results!** We found four images that match your query, and here's how closely they relate to the concept of 'crabs': * **Image 1: COLOURBOX2818387** - This image is a **98.6% match** for 'crabs', indicating a very strong similarity. * **Image 2: COLOURBOX1673685** - This image is a **94.3% match** for 'crabs', showing a high degree of similarity. * **Image 3: COLOURBOX50592999** - This image is a **93.9% match** for 'crabs', indicating a strong similarity. * **Image 4: COLOURBOX35965276** - This image is a **93.8% match** for 'crabs', showing a strong similarity. These results suggest that all four images are closely related to the concept of 'crabs', with Image 1 being the most similar." Below the text, there are two images: the top one shows a crab on a sandy beach, and the bottom one shows a yellow and black striped fish swimming in an aquarium.

Modular RAGs

- Coding Challenge:
- Set up the LlamaService.cs class in the Services folder by yourself. Hint: It bears a remarkable resemblance to one of the other service classes..

Modular RAG's

- Learn more about Azure AI Services hubs and projects:
- Microsoft Learn AI Services landing page:
<https://learn.microsoft.com/en-us/azure/ai-services/>
- Azure AI Studio:
<https://azure.microsoft.com/en-us/products/ai-studio>
- Azure AI Hubs and projects:
<https://learn.microsoft.com/en-us/azure/ai-studio/how-to/create-azure-ai-resource?tabs=portal>