

# Blazor Simple AI Project

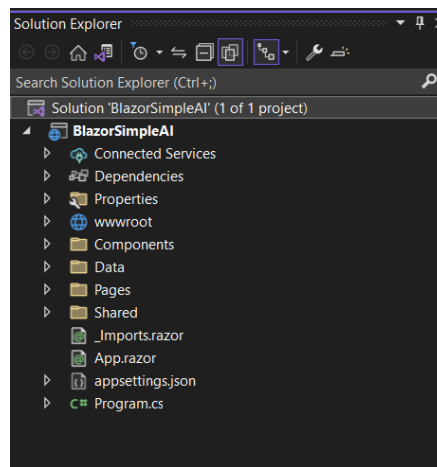
Welcome to the Blazor Simple AI Single Page App, the AI that responds to questions instantly using Microsoft Azure OpenAI Services. This document explains the project in my GitHub repository which is available here: <https://github.com/tejinderrai/public/tree/main/BlazorSimpleAI>.

## Technologies

Blazor Simple AI is made up of the following technologies:

- Microsoft .NET Blazor (.NET 6.0 LTS release)
- Microsoft Azure.AI.OpenAI .NET Library
- Microsoft Azure AI Services – OpenAI

It's that simple!



## Why Blazor?

Blazor is simply amazing, and I have been developing Blazor projects for over four years. There has been great demand for Blazor over the past few years and as a component framework and use of C# this is exactly what I need to develop solutions and concepts super-fast!

## What Blazor Simple AI Does?

Blazor Simple AI is a Blazor server-side single page app which has a single page and a single component. The razor page has two basic user interface controls, a textbox and a submit button for a user to enter the question for Azure OpenAI. The component "AzureOpenAIChat.razor", has a single parameter which receives the question from the main index page. When the parameter is received by the child component, the component has OnParametersSetAsync() method which then retrieves the appsettings.json values in relation to the Azure OpenAI service AI endpoint, Azure OpenAI key and the deployment name which has the associated model, which was deployed with Azure AI Studio, then send the text to the Azure OpenAI service and retrieves and displays the response.

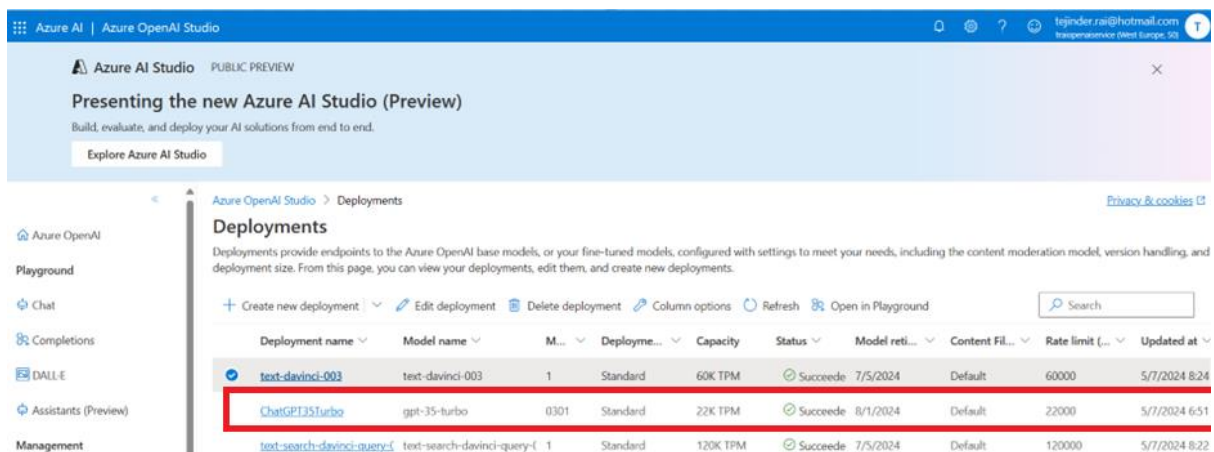
## Core Blazor Template Changes

There have been some basic changes to the basic Blazor layout to accommodate the project. These are as follows:

- 1) The sidebar has been removed from the MainLayout.razor page
- 2) A new Index.razor.css style sheet has been added to centre the UI components on the page
- 3) A new Components folder has been added to the project
- 4) A new component named AzureOpenAIChat.razor has been added into the Components folder
- 5) A new configuration section has been added to appsettings.json to include the configuration required for the project to interact with the Azure OpenAI service
- 6) The title and main element have had text changes to represent the project name and description

## Steps to Deploy Azure Open AI

- 1) Create an Azure Resource Group
- 2) Deploy the Azure OpenAI service in the resource group, see: [How-to: Create and deploy an Azure OpenAI Service resource - Azure OpenAI | Microsoft Learn](#)
- 3) Manage Deployments in Azure AI Studio and create a deployment using the gpt-35-turbo model



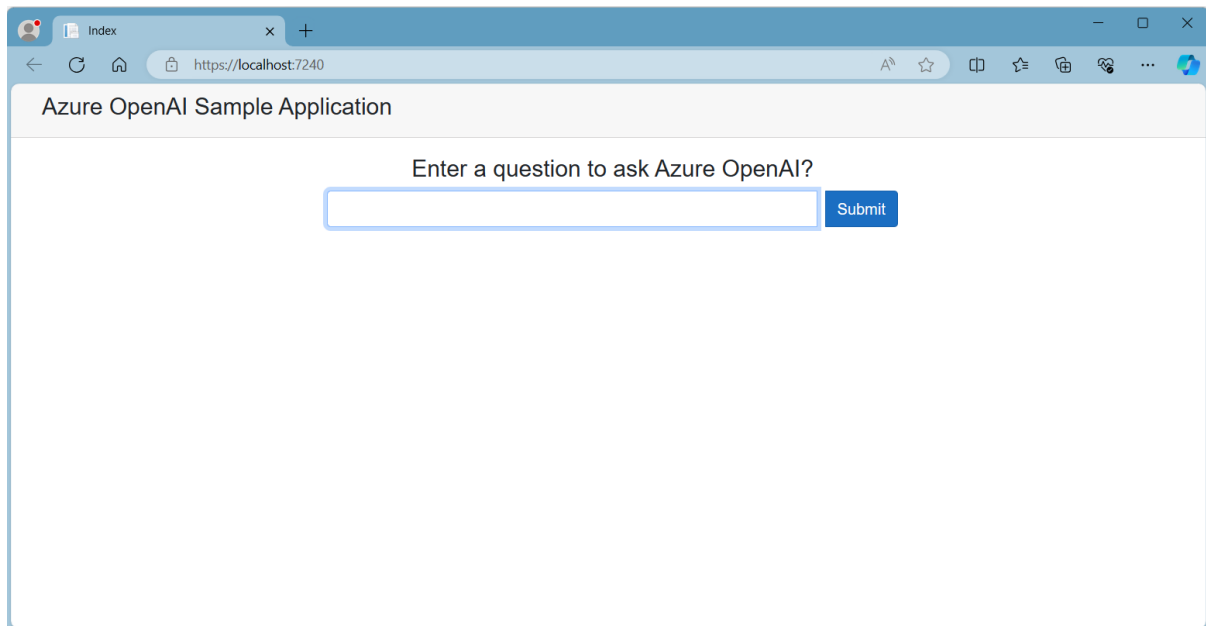
- 4) Update the appsettings.json with the settings

```
"AzureAIConfig": {
  "OpenAIEndpoint": "https://[You Azure OpenAI Service].openai.azure.com/",
  "OpenAIKeyCredential": "[Your Azure Open AI Key]",
  "OpenAIDeploymentName": "[Your Azure Open AI Deployment Name]"
  "RetroResponse": "true or false"
}
```

- 5) Build the project and ask Azure OpenAI anything you like.

## The UI

The landing page.

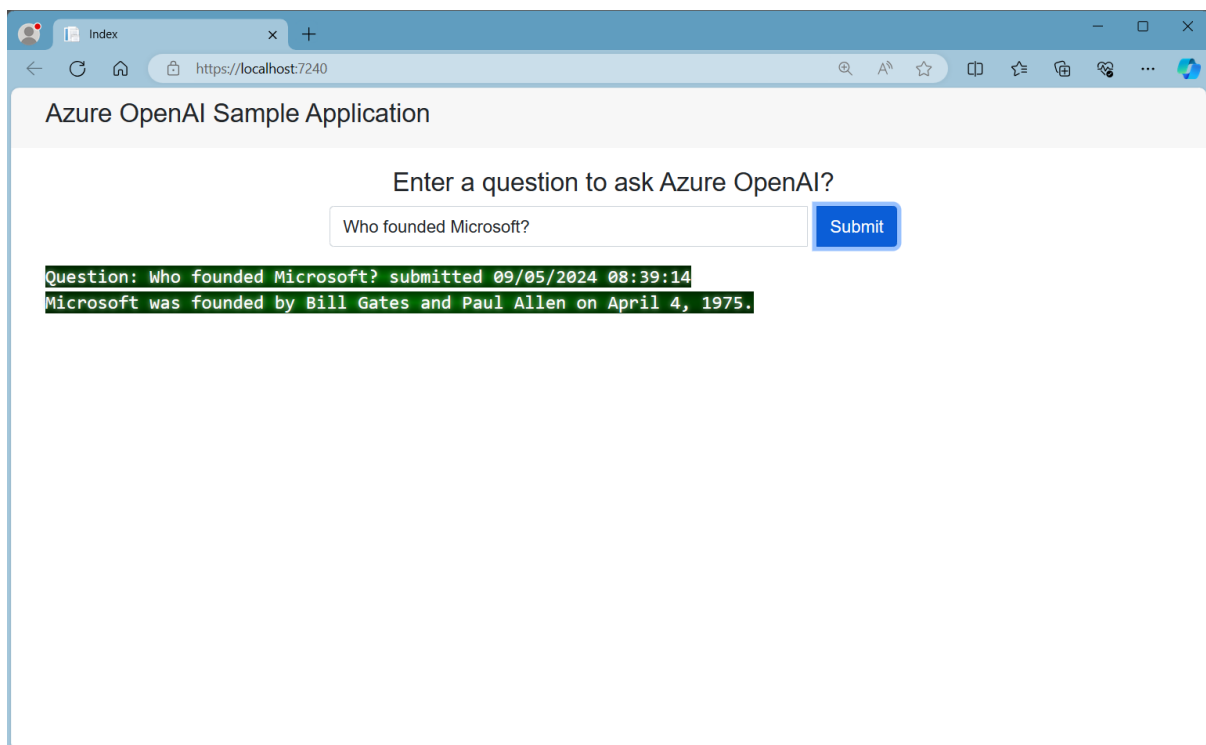


A screenshot of a web browser window showing the landing page of the 'Azure OpenAI Sample Application'. The browser's address bar displays 'https://localhost:7240'. The page has a light gray header with the title 'Azure OpenAI Sample Application'. Below the header, there is a large white area with the prompt 'Enter a question to ask Azure OpenAI?' centered at the top. Underneath this prompt is a text input field and a blue 'Submit' button to its right.

## Sample Questions and Responses

### Question 1

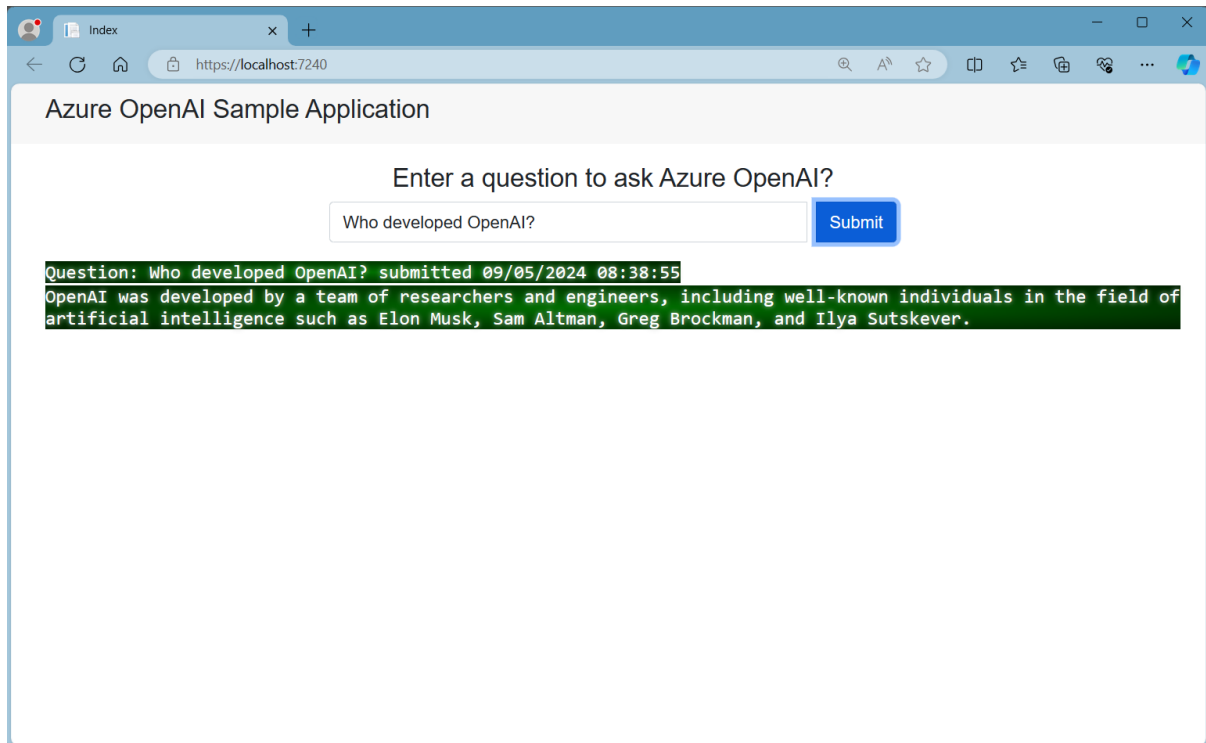
Who founded Microsoft?



A screenshot of the same web browser window, but now showing the application's response to the question 'Who founded Microsoft?'. The text 'Who founded Microsoft?' is entered into the input field. Below the input field, the application displays the following text in a green monospace font: 'Question: Who founded Microsoft? submitted 09/05/2024 08:39:14' followed by 'Microsoft was founded by Bill Gates and Paul Allen on April 4, 1975.' The blue 'Submit' button remains visible to the right of the input field.

## Question 2

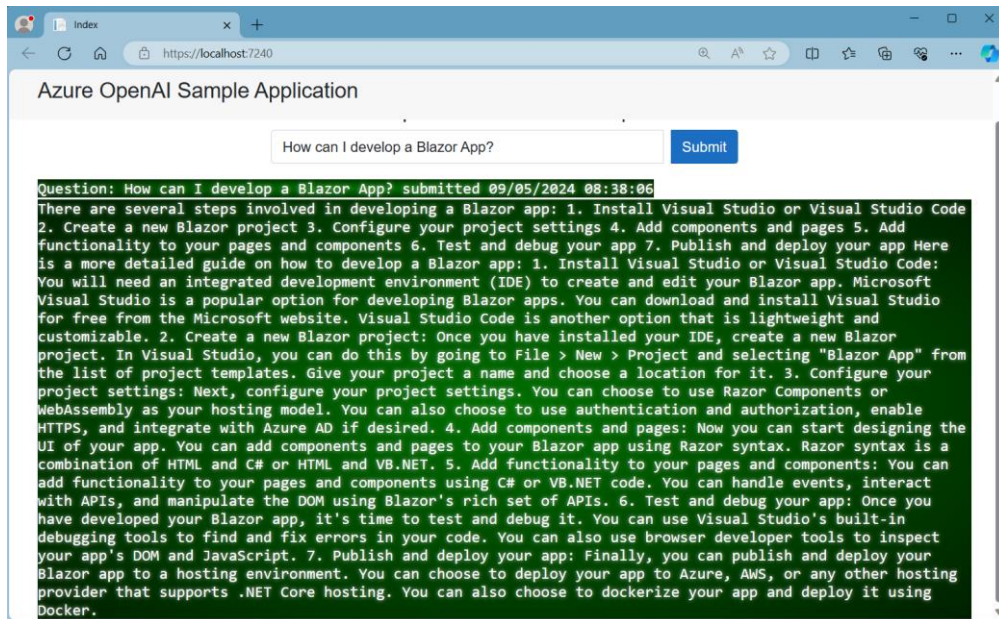
Who developed OpenAI?



The screenshot shows a web browser window with the title "Index" and the URL "https://localhost:7240". The page is titled "Azure OpenAI Sample Application". It features a text input field with the placeholder text "Enter a question to ask Azure OpenAI?". Below the input field, the text "Who developed OpenAI?" is entered. To the right of the input field is a blue "Submit" button. Below the input field, the text "Question: Who developed OpenAI? submitted 09/05/2024 08:38:55" is displayed. Below this, the text "OpenAI was developed by a team of researchers and engineers, including well-known individuals in the field of artificial intelligence such as Elon Musk, Sam Altman, Greg Brockman, and Ilya Sutskever." is displayed.

### Question 3

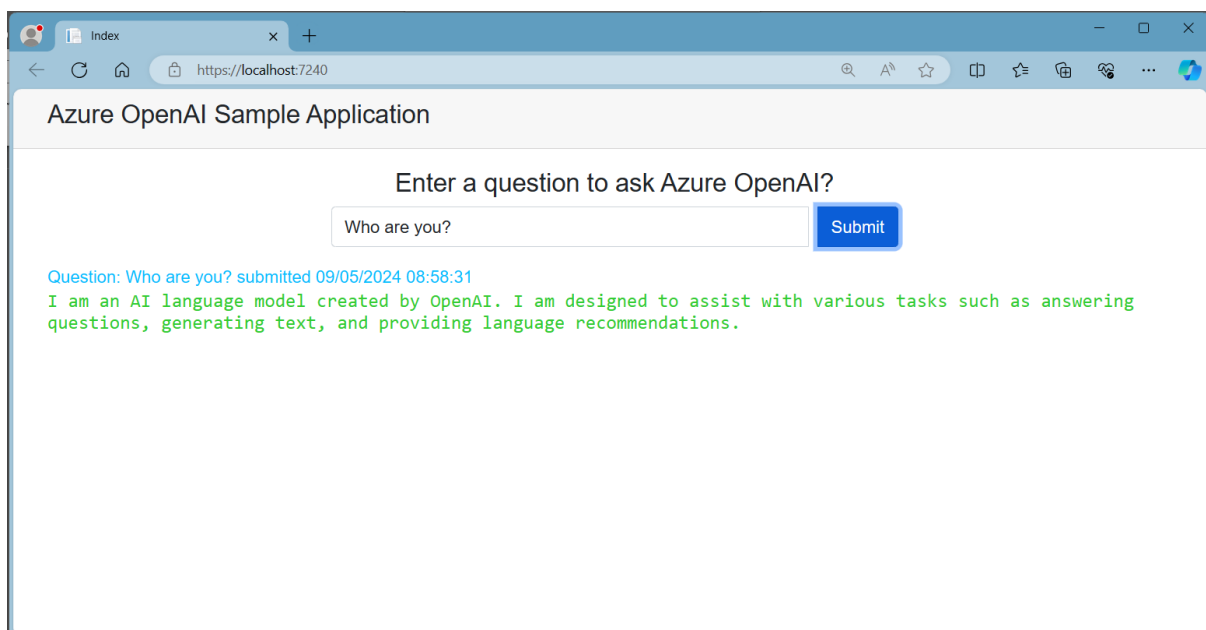
How can I develop a Blazor App?



### Basic CSS

The AzureOpenAIChat.razor component has a basic CSS style sheet which allows the deployment to have a retro style response or a basic response text visualization option. If the app setting below is set to true, you will get the retro response as per the sample above. For a standard non-retro style response, you can set the value to false, example below.

```
"AzureAIConfig": {  
  "RetroResponse": "false"  
}
```



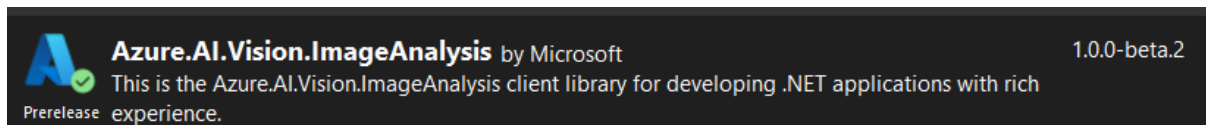
# Blazor Simple AI Project (Part 2)

## Image Analysis with Azure AI Vision

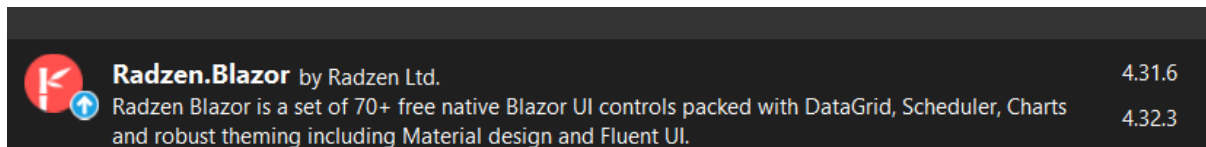
Welcome to the Blazor Simple AI Single Page App, Part 2 of the Microsoft AI services journey, which now includes image analysis utilising Microsoft Azure AI Vision. The Vision Read API is used to extract the text from an image. This document explains the project in my GitHub repository which is available here: <https://github.com/tejinderrai/public/tree/main/BlazorSimpleAI>.

Since part 1, the following nuget packages have been added to the project.

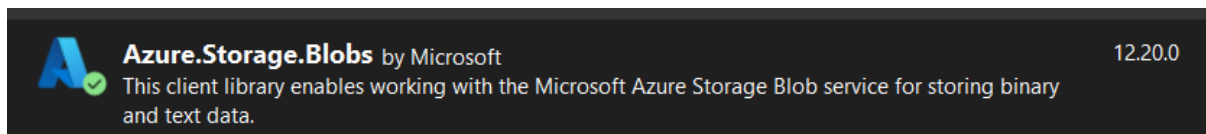
Azure AI Vision Image Analysis – for reading text and metadata from images.



Radzen Blazor – for providing an amazing UI experience.



Azure Storage Blob – for handling interactions with Azure Blob Storage.



### Visual Changes

I have made some appealing improvements from the basic Blazor template and styled the UI based on a project from Martin Mogusu available here: [GitHub - martinmogusu/blazor-top-navbar: A top navbar example created in blazor](#). This saved me a lot of time and all I had to do was apply my own visual styles after the top navigation was applied to the project in shared/NavMenu.razor. In addition, I had added a pre-built model for interactive Invoice Analysis and processing, which I will leave the full explanation until Part 3 of this post.

### Components

Three components have been developed for the image analysis. These are as follows:

- 1) Vision.razor – The Image Analysis page
- 2) VisionBlobLoader.razor – This includes the capability to upload files to Azure blob storage, which also sets the content type for the blob file.
- 3) VisionBlobFileList.razor – This is a child component embedded into the VisionBlobLoader component, which lists the image files that have been uploaded to Azure blob storage.

## Learn about Microsoft AI Vision

To learn more about the capabilities of Microsoft AI Vision, see [What is Azure AI Vision? - Azure AI services | Microsoft Learn](#). Azure AI Vision includes more analysis capabilities, not just specifically image files.

## Configuration Settings Changes

The following configuration settings were added to appsettings.json.

```
"AzureVsnConfig": {  
  "AzureAIVisionEndpoint": "https://[Your AI Vision  
Service].cognitiveservices.azure.com/",  
  "AzureAIVisionKeyCredential": "[AI Vision Service Key]"  
},  
"AzureStorageConfig": {  
  "AzureStorageConnectionString": "[Your Storage Account Connection String]",  
  "AzureStorageContainer": "[Your Storage Account Container]",  
  "AzureStorageAccountName": "[Your Storage Account Name]",  
  "AzureStorageAccountKey": "Your Storage Account Key"  
},
```

**Note:** Whilst this project utilises the service key, in an enterprise environment, you must consider using token based access to the service secured by Microsoft Entra ID, or if you wish to utilise the service key for any reason, utilise Azure Key Vault to protect the key used by the application with a managed identity for the application to access the service key stored in Azure Key Vault.

## Components

### File Upload Component (VisionBlobLoader)

The file upload component utilises Blazor InputFile for the user to select the file to upload in the application. The component reads the Azure Storage connection string from the configuration, including the container, then uploads the file to the container and also adds a blob http header for the file content type taken from the file properties. The Radzen notification service is used to notify the user of the application activities. I also included a basic spinner as part of the interaction for the upload process.

### Blob List Component (VisionBlobFileList.razor)

This component reads the Azure Storage connection string from the configuration, including the container, then displays the blob file names in a Radzen DataGrid. A button is added to Analyse the image, which then calls the Radzen notification service to display the activities being taken by the application.

## Data Classes

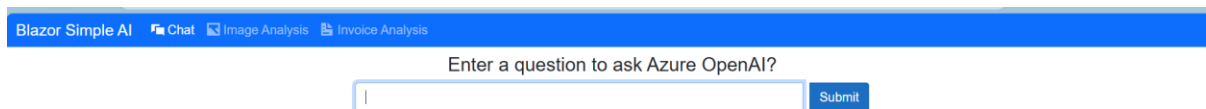
Two data classes have been created as follows:

- AzureBlobFile.cs – Azure blob file properties
- ImageDetails.cs – Image details for extraction from the AI Vision Analysis

## The UI

The UI is as follows. Notice the menu control has now changed since Part 1. Invoice Analysis will be formed in Part 3, at the time of writing this blog post, I had already uploaded the code to my GitHub repo.

## Home page (Chat)

The screenshot shows the top section of a web application. It features a blue header bar with the text "Blazor Simple AI" and three navigation links: "Chat", "Image Analysis", and "Invoice Analysis". Below the header, there is a light blue box containing the prompt "Enter a question to ask Azure OpenAI?". Underneath this prompt is a text input field with a vertical cursor and a blue "Submit" button to its right.

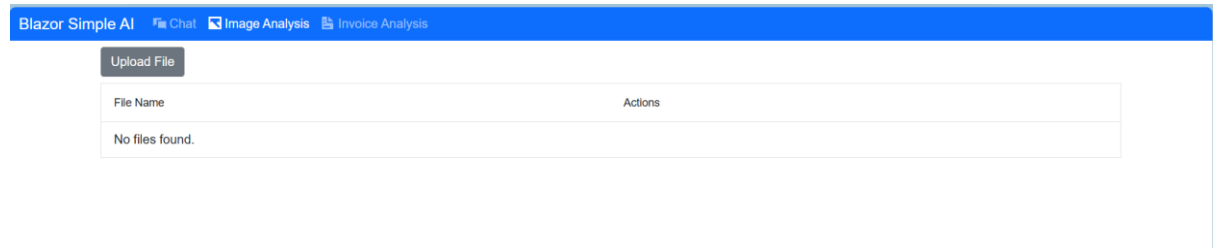
Blazor Simple AI Chat Image Analysis Invoice Analysis

Enter a question to ask Azure OpenAI?

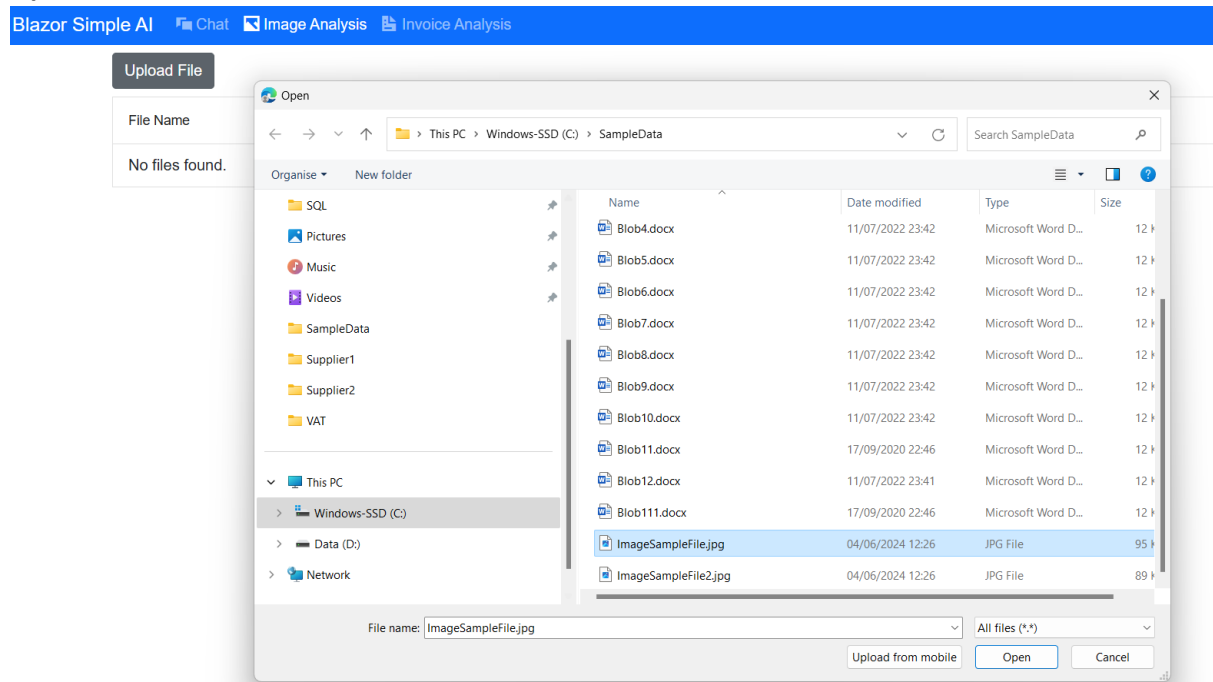
Submit



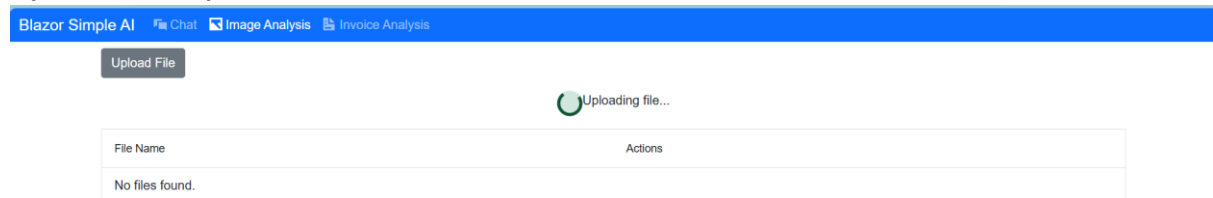
## Image Analysis



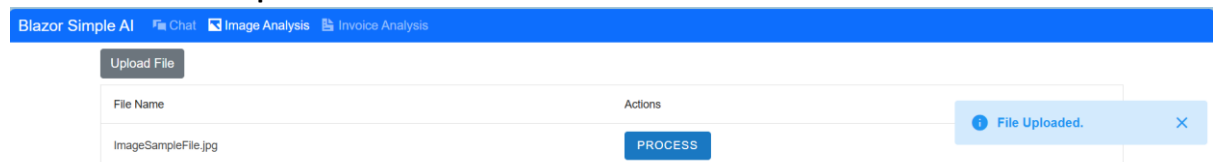
## Upload File Control



## Upload Action Spinner



## Radzen Blazor File Uploaded Notification

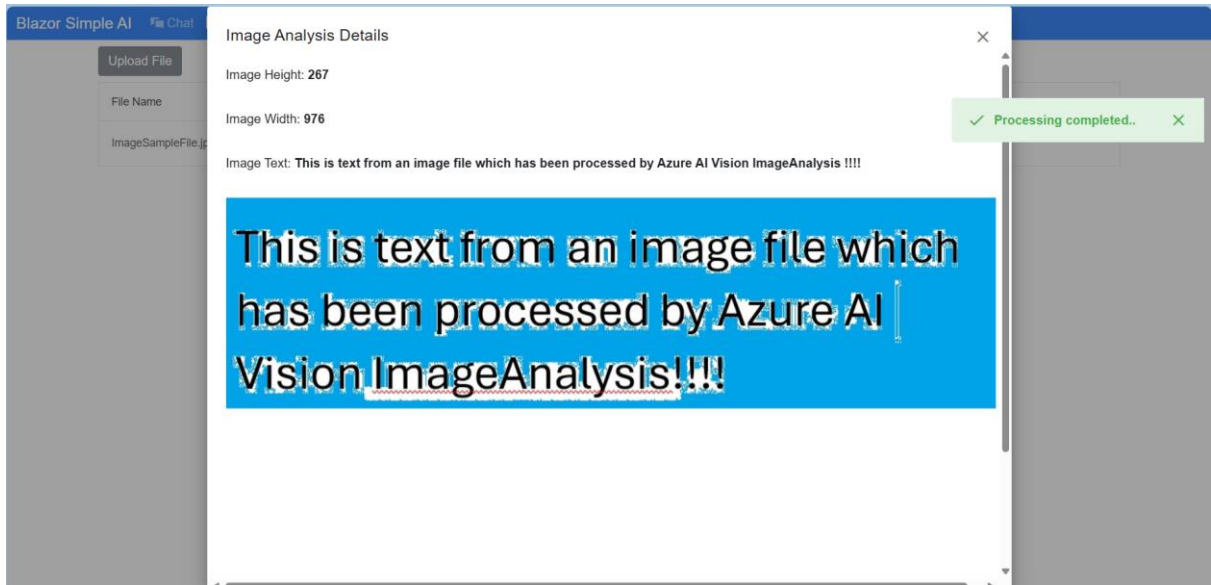


## Process Button

The process button reads the application configuration for the Azure AI Vision endpoint and service key, then retrieves a SAS token from Azure for the blob being processed and a URL is generated with the generated SAS token, then this is submitted to Azure AI Vision with the generated URL. The SAS token is generated by the async method `CreateServiceSASBlob(string BlobName)` in the component class. Whilst the method can be defined as a utility class, I have composed this for easier reading of code.

### Image Analysis Dialog

When the image processing has completed, a Radzen notification is displayed to the user, with a Radzen dialog popping up to show basic metadata (height and width) of the image, including the text the AI Vision service has extracted as well as the image itself.



That is AI Vision and Image Analysis wrapped up. Part 3 will focus on processing invoices using the pre-built AI model “prebuilt-invoice” part of Microsoft Azure AI Document Intelligence.

