Azure Search computes a *search score* for every item returned in search results. The score indicates an item's relevance in the context of the given search operation and determines the order of the item in the set of search results.

You can adjust the default scoring for a search index by adding a *scoring profile*. Scoring profiles allow you to increase the scoring weight of fields, or boost items based on specific field values and calculations.

For detailed information, refer to the Add scoring profiles to a search index article.

To add a scoring profile for an Azure Search index managed by Kentico, you need to customize the functionality that Kentico uses to build the indexes:

1. Open your Kentico solution in Visual Studio.
2. Create a custom module class.
3. Override the module's **OnInit** method and assign a handler to the **SearchServiceManager.CreatingOrUpdatingIndex.Execute** event.
4. Perform the following in the event's handler method:
    a. Access the **Microsoft.Azure.Search.Models.Index** object representing the processed index via the **Index** property of the handler's *CreateOrUpdateIndexEventArgs* parameter.
    b. Write conditions to assign different scoring profiles for specific indexes.
    c. Prepare a **Microsoft.Azure.Search.Models.ScoringProfile** object according to your requirements and add it to the **ScoringProfiles** list of the processed index.

    Important

    The system triggers the **CreatingOrUpdatingIndex** event both when building new indexes and when updating indexes that already exist under the specified Azure Search service. Depending on the number of indexed pages or objects and the used batch size, the event may occur multiple times when building a single search index (separately for each batch of processed search documents that include a new field not yet contained by the index).

    Your code needs to handle the following scenarios:

    - The index already contains a scoring profile with existing configuration.
    - The index does not yet contain all possible fields (for example in cases where the first processed batch of search documents does not include an object with the required fields).

    See the code of the example below.
5. Sign in to the Kentico administration interface.
6. Open the **Smart search** application and **Rebuild** any related Azure Search indexes.

The system creates the customized Azure Search indexes with the specified scoring profile. You can see the profile when viewing index details in the Microsoft Azure portal.

You can now use the scoring profile to adjust the relevance of search results – you need to specify the name of the scoring profile in the **SearchParameters** of your search requests within the implementation of your search components.

Example

The following example demonstrates how to create a basic scoring profile for an Azure Search index named *dg-store*. The sample scoring profile increases the weight of the *skuname* field and uses a freshness function to boost products that were created in Kentico within the last two days.

Start by preparing a separate project in your Kentico solution for the custom module class:

1. Open your Kentico solution in Visual Studio.
2. Create a new *Class Library* project in the Kentico solution named **SearchCustomization**.
3. Add references to the required Kentico libraries (DLLs) for the new project:

   a. Right-click the project and select **Add -> Reference**.
   b. Switch to the **Browse** tab, click **Browse**, and navigate to the ***Lib*** folder of your Kentico web project.
   c. Add references to the following libraries:

      - **CMS.Base.dll**
      - **CMS.Core.dll**
      - **CMS.DataEngine.dll**
      - **CMS.Search.Azure.dll**
4. Right-click the *SearchCustomization* project in the **Solution Explorer** and select **Manage NuGet Packages**.
5. Install the **Microsoft.Azure.Search** package.
6. Reference the *SearchCustomization* project from the Kentico web project *(CMSApp* or *CMS)*.
7. Edit the *SearchCustomization* project's **AssemblyInfo.cs** file (in the *Properties* folder).
8. Add the **AssemblyDiscoverable** assembly attribute:

```
using CMS;

[assembly:AssemblyDiscoverable]
```

Continue by implementing the custom module class:

1. Create a new class named **CustomAzureSearchModule** under the *SearchCustomization* project, with the following code:

```
using System;
using System.Collections.Generic;
using System.Linq;

using CMS;
using CMS.DataEngine;
using CMS.Search.Azure;

using Microsoft.Azure.Search.Models;

// Registers the custom module into the system
[assembly: RegisterModule(typeof(CustomAzureSearchModule))]

public class CustomAzureSearchModule : Module
{
    // Module class constructor, the system registers the module under the name
"CustomAzureSearch"
    public CustomAzureSearchModule()
        : base("CustomAzureSearch")
    {
    }


    // Contains initialization code that is executed when the application starts
    protected override void OnInit()
    {
        base.OnInit();

        // Assigns a handler to the CreatingOrUpdatingIndex event for Azure
Search indexes
        SearchServiceManager.CreatingOrUpdatingIndex.Execute += AddScoringProfile;
    }


    private void AddScoringProfile(object sender, CreateOrUpdateIndexEventArgs e)
    {
        Microsoft.Azure.Search.Models.Index index = e.Index;
```

```
        // Ends the handler method if the index name is not 'dg-store'
        if (!index.Name.Equals("dg-store", StringComparison.
InvariantCultureIgnoreCase))
        {
            return;
        }

        // Creates a dictionary containing the index's fields
        Dictionary<string, Field> indexFields = index.Fields.ToDictionary(f => f.
Name, StringComparer.InvariantCultureIgnoreCase);

        // Initializes the index's list of scoring profiles (if it does not exist)
        if (index.ScoringProfiles == null)
        {
            index.ScoringProfiles = new List<ScoringProfile>();
        }

        // Used to determine whether a new scoring profile was created and needs
to be added to the index
        bool newScoringProfile = false;

        // Checks whether the index already contains a scoring profile named
'productprofile'
        ScoringProfile scoringProfile = index.ScoringProfiles.FirstOrDefault(sp
=> sp.Name == "productprofile");

        // Creates a new scoring profile if it does not exist
        if (scoringProfile == null)
        {
            scoringProfile = new ScoringProfile
            {
                Name = "productprofile",
                FunctionAggregation = ScoringFunctionAggregation.Sum,
                TextWeights = new TextWeights(new Dictionary<string, double>())
            };
            // A new scoring profile was created, it needs to be added to the
index
            newScoringProfile = true;
        }

        // Confirms that the index contains the 'skuname' field and its weight is
not set yet in the scoring profile
        // Note: The 'skuname' field must be configured as 'searchable'
        if (indexFields.ContainsKey("skuname") && !scoringProfile.TextWeights.
Weights.ContainsKey("skuname"))
        {
            // Increases the scoring weight to '3' for search items with matches
in the 'skuname' field
            scoringProfile.TextWeights.Weights.Add("skuname", 3);
        }

        // Confirms that the index contains the 'skucreated' field
        // Note: The 'skucreated' field must be configured as 'filterable'
        if (indexFields.ContainsKey("skucreated"))
        {
            // Defines a freshness scoring function that boosts products created
within the last two days
            var freshnessFunction = new FreshnessScoringFunction
            {
```

```
                FieldName = "skucreated",
                Boost = 3,
                Parameters = new FreshnessScoringParameters(new TimeSpan(2, 0, 0,
0)),
                Interpolation = ScoringFunctionInterpolation.Logarithmic
            };
            // Assigns the freshness function to the scoring profile
            // Note: This sample code always overwrites any existing scoring
functions in the profile
            scoringProfile.Functions = new List<ScoringFunction>() {
freshnessFunction };
        }

        // If a new scoring profile was created and is not empty, adds it to the
index
        if (newScoringProfile && scoringProfile.TextWeights.Weights.Count > 0)
        {
            index.ScoringProfiles.Add(scoringProfile);
        }
    }
}
```

2. Save all changes and **Build** the *SearchCustomization* project.

Configure the required search settings for the fields used by the scoring profile:

1. Open the **Modules** application in the Kentico administration interface.
2. Edit the **E-commerce** module.
3. Select the **Classes** tab, edit the **SKU** class.
4. Open the **Search** tab (click **Customize** if you have not yet configured the search settings of product and page fields).
5. Enable the following search field options in the **Azure** section of the grid:
   - **SKUName** – Content, Retrievable, Searchable
   - **SKUCreated** – Filterable
6. Click **Save**.
7. Open the **Smart search** application and **Rebuild** the related index.

The *dg-store* Azure Search index now contains the *productprofile* scoring profile with the specified parameters. You can use the profile to adjust the relevance of search results – specify the profile name in the **SearchParameters** of your search requests (see Integrating Azure Search into pages to learn how to create search components).

```
using Microsoft.Azure.Search;
using Microsoft.Azure.Search.Models;

...

var searchParams = new Microsoft.Azure.Search.Models.SearchParameters
{
        ScoringProfile = "productprofile"
        ...
};

// Performs the search request with the specified parameters
DocumentSearchResult result = searchIndexClient.Documents.Search(searchString,
searchParams);
```