Add search features onto your MVC site to allow visitors to search through pages, products or other data. The system provides an index-based search engine (smart search). See the Setting up search on your website chapter to learn more.

To set up smart search on your MVC site:

1. Configure automatic Web farm mode in Kentico.
2. Enable smart search indexing.
3. Create search indexes – you can use either Azure Search indexes or locally stored indexes. Assign the indexes to your site and define their exact content.

> If you want the system to process search indexing tasks at a regular interval, you need to use the Scheduler Windows service , since the standard scheduler does not run reliably when using a separate MVC application. See Processing scheduled tasks when developing MVC applications for more information.

With the **Web farm** service enabled and the related web farm servers showing the *Healthy* status, search indexes on MVC applications are updated automatically.

When your indexes are ready, you need to create a suitable search interface. The implementation depends on the type of search indexes that you use (Azure Search or locally stored). See the sections below for more information.

Using Azure Search

To build an MVC search interface for Azure Search indexes:

- Interact with your Azure Search indexes using the Azure Search .NET SDK (provided via the **Microsoft.Azure.Search** NuGet package).
- Create controllers and views according to your search requirements (to collect search input, display search results, provide filters, etc.).

For more detailed information, see Integrating Azure Search into pages. The example describes how to create a web part for a non-MVC Kentico site, but you can use the same general approach in your MVC application's controllers and views.

Using locally stored search indexes

To build an MVC search interface for locally stored search indexes:

1. Install the **Kentico.Search** integration package into your MVC project.
2. Initialize the **SearchService** from the integration package with the indexes you want to search.
3. Call the service's **Search** method to execute the search.

```
            ISearchService searchService = new SearchService();

            SearchResult searchResult = searchService.Search(new SearchOptions
(searchText, searchIndexes)
            {
                CultureName = "en-us",
                CombineWithDefaultCulture = true,
                PageSize = PAGE_SIZE
            });
```

The *Search* method returns a set of results of the **SearchResult** class.

In the search result's **Items** property, you can find the **Fields** and **Data** properties. The *Fields* property contains fields available for all full-text search result items (such as the title or image). The *Data* property is of an abstract *BaseInfo* class. Its run-time type contains all type specific properties. For example:

- If the search indexes users, the run-time type of a search result item's *Data* property is of the *UserInfo* class.

- If the search indexes pages, the run-time type of a search result item's *Data* property is of the *TreeNode* class. If you generated classes for your page types, the run-time type is of the generated class.
- If the search indexes pages representing products, the run-time type of a search result item's *Data* property is of the *SKU TreeNode* class.

With the *Data* property, you can access any object-specific property.

Currently, the *Kentico.Search* integration package supports all types of local search indexes except for custom indexes.

Example – Providing search on MVC sites

The following search example uses an index that indexes pages (*TreeNode* objects) and products (*SKUTreeNode* objects). The example does not use pagination when displaying search result items.

Tip: To view the full code of a functional example directly in Visual Studio, download the Kentico MVC solution from GitHub and inspect the **LearningKit** project. You can also run the Learning Kit website after connecting the project to a Kentico database.

1. Install the **Kentico.Search** integration package into your MVC project.
2. Create a new model for working with search queries and search result items.

```
    public class SearchResultModel
    {
        public string Query { get; set; }

        public IEnumerable<SearchResultItemModel> Items { get; set; }
    }
```

3. Create a new model for the search result item (*SearchResultItemModel* in the example).
4. Create a controller with a controller action for displaying the results.

```
using System;
using System.Collections.Generic;
using System.Web.Mvc;

using CMS.DataEngine;

using Kentico.Search;
```

```
        // Adds the smart search indexes that will be used when performing a
search and sets item count per page
        public static readonly string[] searchIndexes = new string[] { "MVCSite.
Index" };
        private const int PAGE_SIZE = 10;

        /// <summary>
        /// Constructor.
        /// Initializes the search service that takes care of performing
searches. You can use a dependency injection container to initialize the service.
        /// </summary>
        public SearchController()
        {
            searchService = new SearchService();
        }
```

```
        /// <summary>
        /// Performs a search and displays its result.
        /// </summary>
        /// <param name="searchText">Query for search.</param>
        [ValidateInput(false)]
        public ActionResult SearchIndex(string searchText)
        {
            // Displays the search page without any search results if the query
is empty
            if (String.IsNullOrWhiteSpace(searchText))
            {
                // Creates a model representing empty search results
                SearchResultModel emptyModel = new SearchResultModel
                {
                    Items = new List<SearchResultItemModel>(),
                    Query = String.Empty
                };

                return View(emptyModel);
            }

            // Searches with the smart search through Kentico and gets the search
result
            SearchResult searchResult = searchService.Search(new SearchOptions
(searchText, searchIndexes)
            {
                CultureName = "en-us",
                CombineWithDefaultCulture = true,
                PageSize = PAGE_SIZE
            });

            // Creates a list for search result item models
            List<SearchResultItemModel> itemModels = new
List<SearchResultItemModel>();

            // Loops through the search result items
            foreach (SearchResultItem<BaseInfo> searchResultItem in searchResult.
Items)
            {
                // Adds data to a view model. You can adjust the logic here to
get to the object specific fields
                itemModels.Add(new SearchResultItemModel(searchResultItem.
Fields));
            }

            // Creates a model with the search result items
            SearchResultModel model = new SearchResultModel
            {
                Items = itemModels,
                Query = searchText
            };

            return View(model);
        }
```

5. Provide a search field in one of your views. For example:

```
@using (Html.BeginForm("SearchIndex", "Search", FormMethod.Get))
{
    <input type="text" name="searchtext" placeholder="Search..." maxlength="1000">
    <input type="submit" value="Search">
}
```

6. Provide a view that displays search result items.

```
@model SearchResultModel

@if (!Model.Items.Any())
{
    if (!String.IsNullOrWhiteSpace(Model.Query))
    {
        <h3>No results found for "@Model.Query"</h3>
    }
}
else
{
    <h3>Results for "@Model.Query"</h3>
    foreach (SearchResultItemModel item in Model.Items)
    {
        <div>
            @item.Title
            @item.Date.ToLongDateString()
        </div>
    }
}
```

Your visitors can now search on your website with the created search field. The system displays the results based on the indexes specified in the MVC application.