Installation and configuration

Last updated by | Kalle Paulsson | Nov 22, 2024 at 2:44 PM GMT+1

Installing and configuring the add-on

The Epinova AI Assistant is easy to get started with. Most functionality is automatically set up after registering the module, but there are also custom attributes that can be used to further configure the module to work with each sites content modules.

1. Install the `Epinova.CMS.AiAssistant` package from the Optimizely nuget feed.

For Optimizely CMS 12

2. Add the following to Startup.cs

```
using Epinova.CMS.AiAssistant;
...
services.AddAiAssistant();
```

3. Configure the api key for the Ai Assistant add-on in app settings, for instance in appSettings.json

```
"Epinova": {
    "AiAssistantOptions": {
      "ApiKey": "your-api-key"
    }
}
```

4. Ensure that the file `Epinova.CMS.AiAssistant.zip` are created under `modules/_protected/Epinova.CMS.AiAssistant` after building the project.

```
modules
  - _protected
      - Epinova.CMS.AiAssistant
          Epinova.CMS.AiAssistant.zip
```

For Optimizely CMS 11

2. Configure the AI Assistant API routes in Global.asax.cs

```
using Epinova.CMS.AiAssistant;
....
public class EPiServerApplication : EPiServer.Global
{
    protected override void RegisterRoutes(RouteCollection routes)
    {
        ...
        routes.RegisterAiAssistantRoutes();
    }
}
```

3. Configure the api key for the AiAssistant add-on in web.config

```
<appSettings>
    ...
    <add key="Epinova-AiAssistantOptions-ApiKey" value="your-api-key" />
</appSettings>
```

4. Ensure that the file `Epinova.CMS.AiAssistant.zip` and the `Views` folder are created under `modules/_protected/Epinova.CMS.AiAssistant` after building the project. They should also be included in the project.

```
modules
  - _protected
      - Epinova.CMS.AiAssistant
          - Views
              - Default
                  Index.cshtml
              - Shared
                  _ShellLayout.cshtml
                _ViewStart.cshtml
                Web.config
            Epinova.CMS.AiAssistant.zip
```

[Optional] Configuration for meta title, meta description, image alt text

Configure property for meta title and meta description generation.

- You can configure fields that are used for meta title and meta description to be able to automatically suggest content based on the content types main content. This is done by adding the **AiMetaTitleGenerator** and **AiMetaDescriptionGenerator** attributes to the respective properties with a reference to the main content. Main content can be any type , like an `XHTMLString` or `ContentArea` property.
- If a `ContentArea` is used, the AI Assistant will extract string properties from its contained content items. To exclude specific properties within a content item or entire content item types from this extraction process for use in prompts, apply the **AiExcludeFromPrompt** attribute.

```
public virtual XhtmlString MainBody { get; set; }

[AiMetaTitleGenerator(nameof(MainBody))]
public virtual string MetaTitle { get; set }

[AiMetaDescriptionGenerator(nameof(MainBody))]
public virtual string MetaDescription { get; set }
```

_Note: Only one property can currently be configured as the source property.

Configure AiExcludeFromPrompt.

- For example, to exclude entire content of an `HeroBlock` content type from the extracted string, apply the **AiExcludeFromPrompt** attribute on the content type.

```
[AiExcludeFromPrompt]
public class HeroBlock : BlockData
{
    public virtual ImageBlock MainImage { get; set; }

    public virtual string Heading { get; set; }

    public virtual string Subheading { get; set; }
}
```

- For example, to exclude `MainImage` property content of an `HeroBlock` content type from the extracted string, apply the **AiExcludeFromPrompt** attribute on the property.

```
[AiExcludeFromPrompt]
public virtual ImageBlock MainImage { get; set; }

public virtual string Heading { get; set; }

public virtual string Subheading { get; set; }
```

Configure image alt text generation.

You can configure the AI Assistant to be able to automatically generate alt texts for an image. Since Optimizely does not contain a built in way to handle alt texts, you need to connect the alt text and image properties. This is done by connecting your properties using the **AiAltTextGenerator** attribute:

```
[UIHint(UIHint.Image)]
public virtual ContentReference Source { get; set; }

[AiAltTextGenerator(nameof(Source))]
public virtual string AltText { get; set; }
```

Configure image alt text generation on image data

```
public class ImageMediaData : ImageData
{
    [Display(Name = "Alternate text in English")]
    AiAltTextGenerator(Language = "en")]
    public virtual string EnglshAltText { get; set; }

    Display(Name = "Alternate text in Swedish")]
    AiAltTextGenerator(Language = "se")]
    public virtual string SwedishAltText { get; set; }
}
```

Adding custom AI options

Developers have the capability to create customized AI options and display them on the UI.

Create custom tone

1. Create a class that implements `IAiTone` interface.

```
public class SuccinctTone : IAiTone
{
    public string Name => "Succinct";

    public int Order => 60;
}
```

2. Register the new tone in Startup.cs

```
services
    .AddAiAssistant()
    .AddTone<SuccinctTone>();
```

Create custom format

1. Create a class that implements `IAiFormat` interface.

```
public class SalePitchFormat : IAiFormat
{
    public string Name => "Sale Pitch";

    public int Order => 70;
}
```

2. Register the new format in Startup.cs

```
services
   .AddAiAssistant()
   .AddFormat<SalePitchFormat>();
```

Create custom length

1. Create a class that implements `IAiLength` interface.

```
public class VeryShortLength : IAiLength
{
    public string Name => "Very Short";

    public int Order => 5;
}
```

2. Register the new length in Startup.cs

```
services
    .AddAiAssistant()
    .AddLength<VeryShortLength>();
```