FastReport .NET for Blazor

Version 2021.2 Beta

Presenting FastReport.Web.Blazor package with the components to be embedded in your web application. This library is based on the Blazor Server technology (server-side), and means that all operations will be performed on the server-side, which will then transfer the application is ready to display.

The minimum target framework at the moment is .Net Core 3.1 to ensure the highest possible compatibility with the latest LTS (long-term support) version. Also, most users have this version and it is compatible with the latest .NET 5 framework (within this package).

Preflight Preparation

To use FastReport.Web.Blazor, you need to add a reference in your project file (csproj) PackageReference specifying the id of this package and the FastReport.Core package (versions may differ):

```
<ItemGroup>
  <PackageReference Include="FastReport.Core" Version="2021.2.11-demo"/>
  <PackageReference Include="FastReport.Web.Blazor" Version="2021.2.11-demo"/>
</ItemGroup>
```

Then, to simplify naming, we recommend adding the following namespaces to your project's imports (_Imports.razor file):

```
@using FastReport.Web
@using FastReport.Web.Blazor
@using FastReport.Web.Blazor.Components
@using FastReport.Web.Blazor.Components.Internal
```

In fact, just adding FastReport.Web.Blazor.Components may be enough, however, for some cases, you may need other namespaces as well. Also, some components are likely to move within these namespaces during the beta version.

In the configurator of your web application, you need to call the UseFastReport method with an optional lambda expression for setting FastReportOptions. Also, for some built-in common styles and SVG images of icons in Toolbar and Tab to work, you need to use the UseStaticFiles call (if you are not going to use Toolbar and Tabs, the UseStaticFiles call to use this package is optional):

```
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        // ...
        app.UseStaticFiles();
        // ...
```

```
        app.UseFastReport();
    }
```

Description of built-in components

The following is a description of the components included in FastReport.Web.Blazor. We do not recommend using components other than WebReportContainer, because they can be unstable outside of the standard component tree. However, to fine-tune the rendering, you may need to use other components.

WebReportContainer

The main and most versatile Blazor component that performs WebReport rendering is **<WebReportContainer/>**. It is located in the namespace FastReport.Web.Blazor.Components. The only parameter it takes is an object of the WebReport class. This means that to use this component, you must create an object of the WebReport class, assign it a Report, other necessary parameters, and pass this object to the WebReportContainer parameters.

Example:

```
<WebReportContainer WebReport="@UserWebReport" />

@code {
    public WebReport UserWebReport { get; set; }

    protected override void OnParametersSet()
    {
        var report = Report.FromFile(
        Path.Combine(
            directory,
            “My report.frx”));

        // Registers the application dataset
        Report.RegisterData(DataSet, "NorthWind");

        UserWebReport = new WebReport();
        UserWebReport.Report = Report;
    }
}
```

This component can define a different Mode (Designer, Dialog, and normal Preview) and can prepare a report, embed default styles and individual styles, display Toolbar, Outline and Tabs, work with interactive reports, etc.

WebReportPreview

It is similar to the previous component but does not take into account the Designer Mode. That is, it always tries to prepare and render a report.

ReportContainer

It is similar to the previous component but does not include loading WebReport styles (common and individual for Toolbar/Tabs/Outline).

It is engaged in the preparation of the report and subsequent display together with the Toolbar and Tabs (if necessary).

When working with any interactivity (clicks on the report / working with dialog forms), it is this component that is updated.

ReportBody / ExportComponent

The ReportBody calls the Outline rendering (if necessary) and "nests" a component that is the rendering of the report itself (ExportComponent), which the ReportContainer passes to it. Not recommended for use.

BlazorExport

The “lowest” level of a component is not a component at all, but BlazorExport itself - a tool for exporting a prepared report to the RenderTreeBuilder build format. Located in FastReport.Web.Blazor.Export namespace.

To build this export, you must:

1) Prepare the report;
2) Make sure that this report does not use dialog forms (they are rendered using the DialogPageComponent and are not covered in this tutorial);
3) Create your own component and explicitly define the construction method in it (call the override of the BuildRenderTree method)
4) In this build method, create a BlazorExport instance, set the properties it needs, and call Export passing the following parameters: a Report and a builder instance that is an argument to this overridden method.

```
        /// Main function
        protected override void BuildRenderTree(RenderTreeBuilder builder)
        {
            using (BlazorExport blazor = new BlazorExport())
            {
                blazor.StylePrefix = $"fr{WebReport.ID}";
                blazor.EmbedPictures = true;
                blazor.OnClick += ProcessClick;
                blazor.EnableMargins = WebReport.EnableMargins;
```

```
                blazor.SinglePage = true;
                blazor.CurPage = WebReport.CurrentPageIndex;

                blazor.Export(myReport, builder);
            }
        }
```

Online Designer

At the moment, Online Designer can work in the iframe element using javascript and it is fully compatible with the Online Designer assembly for Core, however, the Preview call does not work yet.

To use only the designer's capabilities, you can call the **<IFrameDesigner/>** component passing it the WebReport parameter with the configured Report property and the optional DesignerLocale and DesignerPath:

```
    <IFrameDesigner WebReport="CurrentWebReport" />
```

However, we remind you that the WebReportContainer component understands which Mode it is currently working with and it is not at all necessary to call IFrameDesigner in this form.

Setting up common styles and SVG

Unlike FastReport.Web for Core, SVG images for Toolbar and Tabs, as well as some general display styles of Tabs, Outline, etc. have been moved to staticWebAssets for possible customization in your web application (changing colors, sizes, replacing images).

These resources are located in your local storage. At the time of development/assembly of your application, they are located at:
{UserName}/.nuget/packages/fastreport.web.blazor/{version}/staticwebassets

At the time of publishing your web application (dotnet publish), these resources are copied to the directory:
wwwroot/_content/FastReport.Web.Blazor

Demo project

You can find a project for demonstrating work with the FastReport.Web.Blazor package on our GitHub. An example of using the **WebReportContainer** is in the custom component under Pages/Index.razor and Pages/Index.razor.cs.