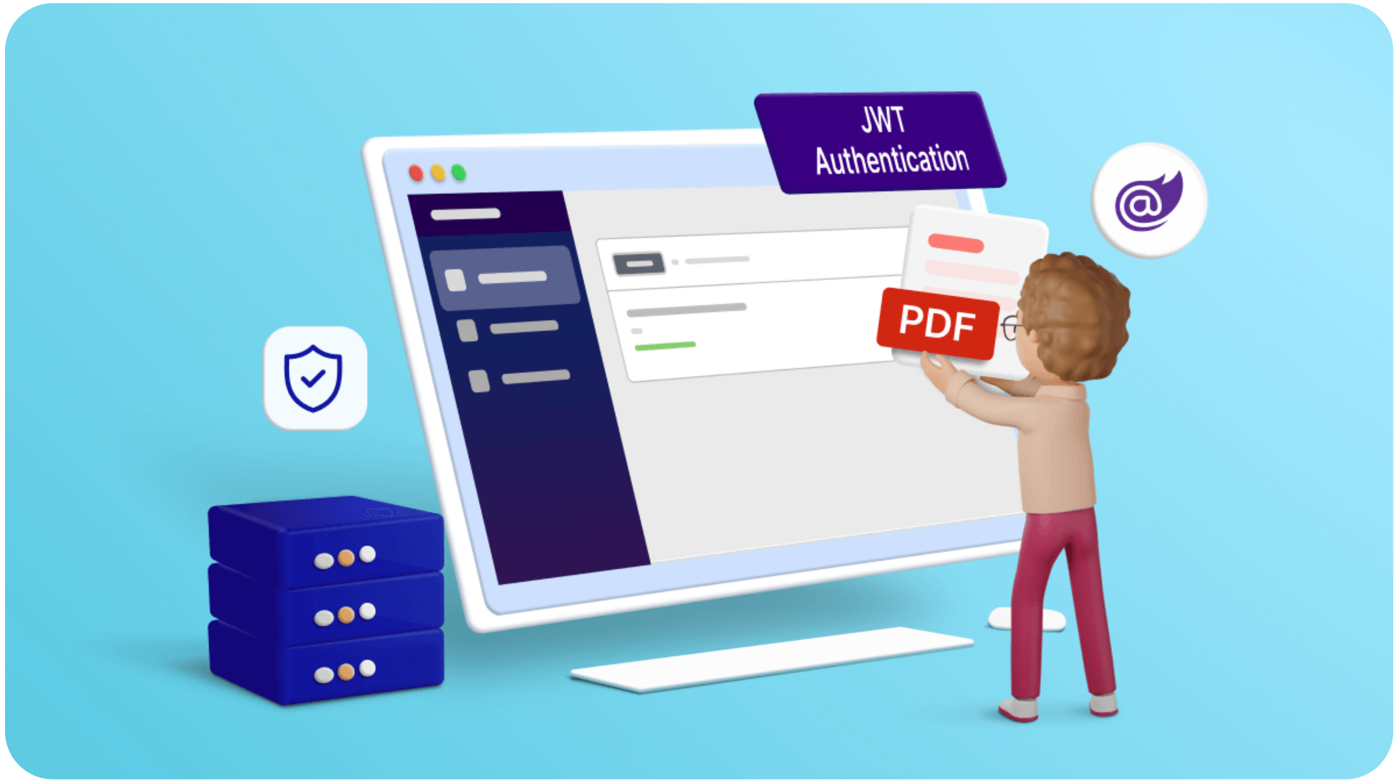
Secure File Handling in Blazor: Implement JWT Authentication

Saravanan G • 5 min read • Nov 19, 2024 • Updated



> **TL;DR:** *Are you worried about unauthorized access to your Blazor apps' file uploads? For enhanced security, learn to implement JWT authentication in the Syncfusion Blazor File Upload component. This guide covers adding the File Upload component, incorporating JWT headers for authentication, and handling file upload and removal on the server to restrict access to authenticated users only.*

File uploads are a common requirement in modern web applications.

Syncfusion Blazor File Upload is a component for uploading files, images, documents, and audio and video files to a server. It works in both WebAssembly and server-side Blazor apps. It also supports a rich set of features, including multiple file selection, progress bars, auto-uploading, drag and drop, folder (directory) uploading, file validation, and more.

In this blog, we'll see how to integrate the Syncfusion Blazor File Upload component with JWT (JSON Web Token) authentication in a Blazor app. This combination allows us to securely upload and remove files while ensuring that only authenticated users can perform the actions.

Let’s get started!

Prerequisites

- Visual Studio 2022
- .NET Core 6.0 and above

Step 1: Create a Blazor WebAssembly app

First, create a new Blazor WebAssembly app using Visual Studio. Then, install Syncfusion Blazor packages and configure the styles and script references using the getting started documentation.

Step 2: Add Blazor File Upload component

Now, integrate the Syncfusion Blazor File Upload component into your Blazor page. Refer to the following code example.

Copy

```
@using Syncfusion.Blazor.Inputs

<SfUploader ID="UploadFiles">
 <UploaderAsyncSettings SaveUrl="api/FileAction/Save" RemoveUrl="api/FileAction/
 </UploaderAsyncSettings>
</SfUploader>
```

Step 3: Add JWT authentication for file upload action

The Blazor File Upload component allows you to add an additional header to bind the authentication token during **file upload**, which can then be received on the server side. To configure the header as a key-value pair, you can achieve this behavior by using the FileSelected and BeforeRemove events and their CurrentRequest arguments.

Refer to the following code example.

Copy

```
<SfUploader ID="UploadFiles">
 <UploaderEvents FileSelected="onFileSelect" BeforeRemove="onRemove"></UploaderE
 <UploaderAsyncSettings SaveUrl="api/FileAction/Save" RemoveUrl="api/FileAction/
 </UploaderAsyncSettings>
</SfUploader>
```

```
@code {

    private void onFileSelect(SelectedEventArgs args)
    {
        args.CurrentRequest = new List<object> { new { Authorization = "test@123
    }
}
```

Step 4: Implement server-side handling for upload action

We should implement the API endpoints to handle file uploads and removals on the server side. These endpoints will validate JWT tokens to ensure authentication.

In the server-side control code, you can retrieve the authentication token key from the server project’s response header for file upload action, as demonstrated in the following code example.

Copy

```
[HttpPost("[action]")]
public async void Save(IList<IFormFile> UploadFiles)
{
     //To get the authorization header to handle save file action on the server
     var authorizationHeader = Request.Headers["Authorization"];
     if (authorizationHeader.Count == 0 || authorizationHeader[0] != "test123")
     {
        Response.Clear();
```

```
            Response.StatusCode = 401;
            Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
            return;
         }
         try
         {
            foreach (var file in UploadFiles)
            {
                if (UploadFiles != null)
                {
                    var filename = ContentDispositionHeaderValue.Parse(file.ContentD
                    filename = hostingEnv.WebRootPath + $@"\{filename}";
                    if (!System.IO.File.Exists(filename))
                    {
                        using (FileStream fs = System.IO.File.Create(filename))
                        {
                            file.CopyTo(fs);
                            fs.Flush();
                        }
                    }
                }
            }
        }
        catch (Exception e)
        {
            Response.Clear();
            Response.ContentType = "application/json; charset=utf-8";
            Response.StatusCode = 204;
            Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
            Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
        }
```

```
}
```

Step 5: Add JWT authentication for file removal action

In the same way, you can handle the **file removal** action by sending the JWT authentication header to the BeforeRemove event and its CurrentRequest argument to configure the header as a key-value pair.

Refer to the following code example.

```
private void onRemove(BeforeRemoveEventArgs args)
{
    args.CurrentRequest = new List<object> { new { Authorization = "test123" } };
}
```

Step 6: Implementing server-side file removal action

Within the server-side control code, you can retrieve the authentication token key to perform file removal action from the response header, mirroring the procedure in the file-saving action controller. Verify the authentication before executing the file removal process.

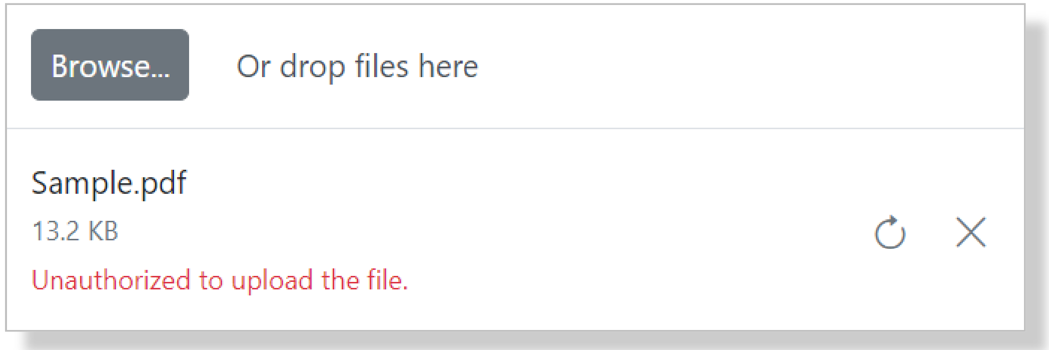
Refer to the following code example.

Copy

```
[HttpPost("[action]")]
public void Remove(IList<IFormFile> UploadFiles)
{
    // To get the authorization header to handle the file removal action on the
    var authorizationHeader = Request.Headers["Authorization"];
    if (authorizationHeader.Count == 0 || authorizationHeader[0] != "test123")
    {
        Response.Clear();
        Response.StatusCode = 401;
        Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
        return;
    }
    try
    {
        var filename = hostingEnv.ContentRootPath + $@"\{UploadFiles[0].FileName
        if (System.IO.File.Exists(filename))
        {
            System.IO.File.Delete(filename);
        }
    }
    catch (Exception e)
    {
        Response.Clear();
```

```
            Response.StatusCode = 200;
            Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
            Response.HttpContext.Features.Get<IHttpResponseFeature>().ReasonPhrase =
        }
    }
```

Refer to the following output image.



Unauthorized to upload the file.

Implementing JWT authentication in the Blazor File Upload component

GitHub reference

Also, check out the complete source code for integrating JWT authentication in Blazor File Upload component on GitHub.

Conclusion

Thanks for reading! This blog shows how to integrate JWT authentication in the Syncfusion Blazor File Upload component. This can enhance the security of our file upload functionality. Authenticated users can securely upload and manage files while unauthorized access is effectively prevented.

Experience our Blazor component firsthand by downloading a free 30-day trial or utilizing our NuGet package. Explore additional features through our Blazor online examples and documentation.

If you have any questions, please don't hesitate to let us know in the comments section given below. You can also contact us through our support forum, support portal, and feedback portal. We are always eager to assist you!

Related blogs

- Easily Perform CRUD Actions in Blazor Pivot Table with SQL Database & Entity Framework
- Seamlessly Load Data from Different Data Sources into Blazor Charts
- Advanced Query Building Techniques: Connecting Tables with Joins using Blazor Query Builder
- Creating Custom Forms and Validation in a Blazor Hybrid App

MEET THE AUTHOR

Saravanan G

Saravanan is a Technical Product Manager at Syncfusion for Web products. He is passionate about Web technology and has been active in development since 2010 and focuses mainly on delivering the products with perfection.