**Creating First Application in Blazor from scratch**

Creeremo la nostra prima app ASP.NET Core Blazor in Visual Studio partendo dall'inizio. Creeremo un'app ASP.NET Core da una fase vuota e la configureremo per usare Blazor. Nota che Visual Studio ha un'opzione Blazor Template che configura automaticamente un nuovo progetto per Blazor. Dobbiamo presentarti tutte le strutture dei file di Blazor e il loro funzionamento, quindi creeremo questa applicazione da un modello ASP.NET Core vuoto e useremo l'ultima versione .NET 7.0. Amerai questo approccio poiché imparerai tutti i componenti ASP.NET Core Blazor necessari e il loro funzionamento, quindi segui questo tutorial fino alla fine.

Page Contents

* [Create a new Blazor Application](https://www.yogihosting.com/blazor-first-application/#create)
* [ASP.NET Core Blazor Configuration](https://www.yogihosting.com/blazor-first-application/#configure)
  + [Razor Components](https://www.yogihosting.com/blazor-first-application/#razor-components)
* [Blazor Imports File – “\_Imports.razor”](https://www.yogihosting.com/blazor-first-application/#blazor-imports)
* [Blazor Router component – “App.razor”](https://www.yogihosting.com/blazor-first-application/#blazor-router)
* [Endpoint Routing integration in Blazor](https://www.yogihosting.com/blazor-first-application/#routing)
* [Fallback Route “\_Host.cshtml”](https://www.yogihosting.com/blazor-first-application/#fallback-route)
* [Blazor Layouts](https://www.yogihosting.com/blazor-first-application/#layouts)
* [Create a new Razor Component](https://www.yogihosting.com/blazor-first-application/#razor-component)
* [Integrating Bootstrap in Blazor App](https://www.yogihosting.com/blazor-first-application/#bootstrap)
* [Creating an Advanced Feature in Blazor](https://www.yogihosting.com/blazor-first-application/#advanced)
* [NavLink in-built component for Navigation](https://www.yogihosting.com/blazor-first-application/#navlink)
* [Left Navigation using NavLink](https://www.yogihosting.com/blazor-first-application/#left-navigation)
* [NavigationManager class](https://www.yogihosting.com/blazor-first-application/#navigationmanager)
* [Download Source Code](https://www.yogihosting.com/blazor-first-application/#download)

**Create a new Blazor Application**

**In Visual Studio crea un nuovo Progetto e seleziona il modello denominato ASP.NET Core Vuoto come mostrato nell'immagine sottostante. Questo modello non ha alcun contenuto. Nella finestra successiva che dice – Configura il tuo nuovo progetto, assegna al progetto il nome BlazorFirstApp e seleziona la posizione sul disco per i file di progetto. Quindi fai clic sul pulsante Avanti. Guarda l'immagine sottostante:** **Successivamente, otterrai la finestra Additional information. Qui puoi selezionare la seguente versione DOT NET da utilizzare. Puoi vedere che ho selezionato la versione DOT NET 7.0. Infine, fai clic sul pulsante Create. Il progetto verrà creato in pochi secondi e sarai pronto per utilizzare e scatenare la potenza della tecnologia Blazor.**

**ASP.NET Core Blazor Configuration**

**L'app ASP.NET Core è già stata creata e se vedi la finestra Esplora soluzioni noterai che ci sono solo pochi file. Controlla lo screenshot riportato di seguito.** **Ora aggiungeremo le configurazioni Blazor su questa app ASP.NET Core. Per essere più specifici, faremo la configurazione "Blazor Server" all'app. Quindi il primo passaggio di configurazione è aggiungere i servizi e il middleware alla classe Program.cs che è disponibile nella radice dell'app. Inizia aggiungendo il supporto per Razor Pages nell'app poiché Blazor ha bisogno di Razor Pages per funzionare. Il codice che devi aggiungere è riportato di seguito.**

**builder.Services.AddRazorPages();**

**Quindi, aggiungi i servizi Blazor lato server alla raccolta di servizi. Il codice da aggiungere è riportato di seguito.**

**builder.Services.AddServerSideBlazor();**

**Ho mostrato tutte queste configurazioni ASP.NET Core Blazor in modo evidenziato di seguito. Gli altri codici sono configurazioni ASP.NET Core necessarie per l'app.**

**Infine dobbiamo integrare Blazor con ASP.NET Core Endpoint Routing. Ciò consentirà a SignalR, che è la parte di ASP.NET Core che gestisce la richiesta HTTP persistente, di funzionare correttamente. Il codice per questo è: app.MapBlazorHub(); Ho mostrato tutte queste configurazioni ASP.NET Core Blazor evidenziate di seguito. Gli altri codici sono configurazioni ASP.NET Core necessarie per l'app. Abbiamo eseguito con successo le configurazioni Blazor nell'app ASP.NET Core. Successivamente, tratteremo i singoli file Blazor come componenti Razor, router, layout, ecc.**

**Immagine che contiene testo, schermata, Carattere, software

Il contenuto generato dall'IA potrebbe non essere corretto.**

Componenti Razor Le app Blazor sono create utilizzando componenti chiamati Componenti Razor. Questi componenti sono implementati nei file dei Componenti Razor e questi file hanno un'estensione .razor. Nel componente Razor scrivi i tuoi codici Blazor. In genere contengono: Una miscela di HTML e C# e utilizza le note sintassi Razor. Codici C# definiti all'interno del blocco @code. Blazor ha molti Componenti Razor integrati, alcuni dei più comuni sono \_Imports.razor, \_App.razor, LayoutView.razor, MainLayout.razor, NavLink.razor, NavMenu.razor, Router.razor sono alcuni dei più comuni. Possiamo anche creare i nostri componenti Razor per eseguire attività specifiche.

**Blazor Imports File – “\_Imports.razor”**

Blazor Imports File ti consente di importare namespace di uso comune in un unico posto, in modo da non doverli importare separatamente su ogni componente Razor che ne ha bisogno. Fai clic con il pulsante destro del mouse sul nome del progetto in Solution Explorer e seleziona Aggiungi ➤ Nuovo elemento. Vedrai aprirsi una finestra Aggiungi nuovo elemento che contiene molti file diversi che possono essere selezionati. Qui seleziona Razor Component e assegnagli il nome \_Imports.razor. Fai clic sul pulsante Aggiungi per creare questo file nella cartella principale dell'applicazione.

Una volta creato il file Blazor Imports, rimuovi tutti i codici predefiniti e aggiungi solo le seguenti 3 righe di codice: @using Microsoft.AspNetCore.Components.Web @using Microsoft.AspNetCore.Components.Routing @using BlazorFirstApp.Shared Nella prima riga abbiamo importato lo spazio dei nomi Components.Web che contiene molti componenti necessari integrati in Blazor. La seconda riga importa la funzionalità Routing e viene utilizzata per il rendering e la mappatura dei percorsi ai componenti. Ciò è necessario affinché gli URL vengano mappati a diversi componenti Razor.

La terza riga importa il contenuto della cartella Shared. La cartella Shared è conservata all'interno della cartella radice dell'APP e contiene i file che vengono spesso condivisi tra i diversi componenti Razor. Si tratta di navigazioni, fogli di stile, layout e così via. Spiegherò di più sulla cartella "Shared" nella seconda metà di questo tutorial. Alcuni punti da ricordare: 1. Puoi aggiungere un file chiamato \_Imports.razor a ogni cartella dell'applicazione. Il .NET includerà le direttive specificate nel file Blazor Imports per tutti i modelli Razor che si trovano nella stessa cartella o in una qualsiasi delle sue sottocartelle. 2. Puoi anche specificare i layout nel file Blazor Imports per il componente Razor. Ad esempio: @layout MyLayout assicura che tutti i componenti utilizzino MyLayout.

**Blazor Router component – “App.razor”**

Componente Blazor Router – “App.razor” Blazor utilizza il sistema di routing ASP.NET Core per selezionare i componenti in base agli URL. Quindi le applicazioni Blazor rispondono alle modifiche nell'URL visualizzando diversi componenti Razor. Per iniziare, aggiungi un nuovo componente Razor nella cartella principale dell'applicazione e chiamalo App.razor. Aggiungi i seguenti codici forniti:

<Router AppAssembly="@typeof(App).Assembly">

<Found Context="routeData">

<RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />

<FocusOnNavigate RouteData="@routeData" Selector="h1" />

</Found>

<NotFound>

<PageTitle>Not found</PageTitle>

<LayoutView Layout="@typeof(MainLayout)">

<p role="alert">Sorry, there's nothing at this address.</p>

</LayoutView>

</NotFound>

</Router>

Lasciatemi spiegare in dettaglio tutto il codice di cui sopra. Innanzitutto, dobbiamo specificare il nome dell'AppAssembly .NET da utilizzare, che nella maggior parte dei casi è l'assembly corrente: <Router AppAssembly="@typeof(App).Assembly"> Quindi, definiamo le sezioni "Found" e "NotFound". Nella sezione Found fornisci i contenuti da visualizzare quando viene trovata la corrispondenza con la route richiesta (ad esempio URL). Analogamente, nella sezione NotFound fornisci i contenuti da visualizzare quando non viene trovata alcuna corrispondenza per la route richiesta.

In realtà Router, Found, NotFound, RouteView ecc. sono tutti componenti integrati di Blazor che eseguono attività specifiche fornite loro. Ho definito la sezione "Found" come mostrato di seguito: <Found Context="routeData"> <RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" /> <FocusOnNavigate RouteData="@routeData" Selector="h1" /> </Found> La proprietà context è impostata su routeData – Context="routeData". Al suo interno c'è la sezione RouteView che riceve i dati del percorso (impostati come RouteData="@routeData") e il layout predefinito che è impostato come DefaultLayout="@typeof(MainLayout)". Parleremo del layout predefinito nella prossima sezione. FocusOnNavigate imposta il focus dell'interfaccia utente sull'elemento h1 di livello superiore quando la navigazione avviene da una pagina all'altra.

RouteView popola i contenuti del componente corrispondente all'interno del layout (UI). In questo caso ho definito questo layout come MainLayout. Il layout contiene fondamentalmente le aree comuni dell'app come l'intestazione e il piè di pagina del sito web. Queste aree comuni possono essere mostrate da un file di layout anziché definirle più e più volte in diversi componenti dell'app Blazor. Questo approccio riduce notevolmente la ridondanza del codice. Venendo alla sezione "NotFound" che è data come:

<NotFound> <PageTitle>Non trovato</PageTitle> <LayoutView Layout="@typeof(MainLayout)"> <p role="alert">Spiacenti, non c'è nulla a questo indirizzo.</p> </LayoutView> </NotFound> Imposta il titolo della pagina su Non trovato e richiama MainLayout e mostra un messaggio di scuse all'utente. Questo caso si verifica quando non viene trovata alcuna corrispondenza per i percorsi. Puoi anche utilizzare un layout diverso per la sezione non trovata, se lo desideri. Integrazione di Endpoint Routing in Blazor

Blazor funziona con ASP.NET Core Endpoint Routing. Ricorda che nella sezione precedente lo abbiamo già integrato aggiungendo il codice – app.MapBlazorHub() all'interno della classe Program.cs. Consente all'app di chiamare qualsiasi componente Razor dall'interno delle viste del controller ASP.NET Core o delle pagine Razor. Fallback Route “\_Host.cshtml” In un'app Blazor autonoma (come nel nostro caso), dobbiamo usare una Fallback Route che chiamerà una pagina Razor con il nome di \_Host.cshtml. La fallback route ha una priorità molto bassa nella corrispondenza delle route e quindi questa route viene avviata quando altre route non corrispondono.

Per questo motivo ora possiamo anche aggiungere aree ASP.NET Core MVC e pagine Razor nella nostra app Blazor e non interferiranno con i componenti ASP.NET Core Blazor. Quindi vai avanti e aggiungi la riga di codice – app.MapFallbackToPage("/\_Host") nella classe “Program.cs” subito dopo aver aggiunto il metodo app.MapBlazorHub(). Guarda il codice evidenziato di program.cs di seguito.

Ho mostrato tutte queste configurazioni ASP.NET Core Blazor evidenziate di seguito. Gli altri codici sono configurazioni ASP.NET Core necessarie per l'app. var builder = WebApplication.CreateBuilder(args); builder.Services.AddRazorPages(); builder.Services.AddServerSideBlazor(); var app = builder.Build(); if (!app.Environment.IsDevelopment()) { app.UseExceptionHandler("/Error"); app.UseHsts(); } app.UseHttpsRedirection(); app.UseStaticFiles(); app.UseRouting(); app.MapBlazorHub(); app.Run();

Abbiamo eseguito con successo le configurazioni Blazor nell'app ASP.NET Core. Ora, parleremo dei singoli file Blazor come componenti Razor, router, layout, ecc. Componenti Razor Le app Blazor sono create utilizzando componenti chiamati Componenti Razor. Questi componenti sono implementati nei file dei componenti Razor e questi file hanno un'estensione .razor. Nel componente Razor scrivi i tuoi codici Blazor. In genere contengono:

* A mixture of HTML and C# and uses the familiar Razor syntaxes.
* C# codes defined inside the @code block.