

Laboratorio di Programmazione Web II

#WeAreSharp

Corso di Laurea “Programmazione e Gestione di Sistemi Informatici”
AA 2024-2025



Daniele Selvi

Head of Microsoft Strategy, MD Apsia Italia

email: daniele.selvi@apsia.eu



SHARP

Sharp in Europe

We've been operating for **over 100 years in Europe** with the head office based in **London**, operating across 30 Countries.

Apsia in Europe

150+ professionals **over 20 years of experience in Business Applications**. Head office in **Paris**, subs in Perugia, Barcelona.

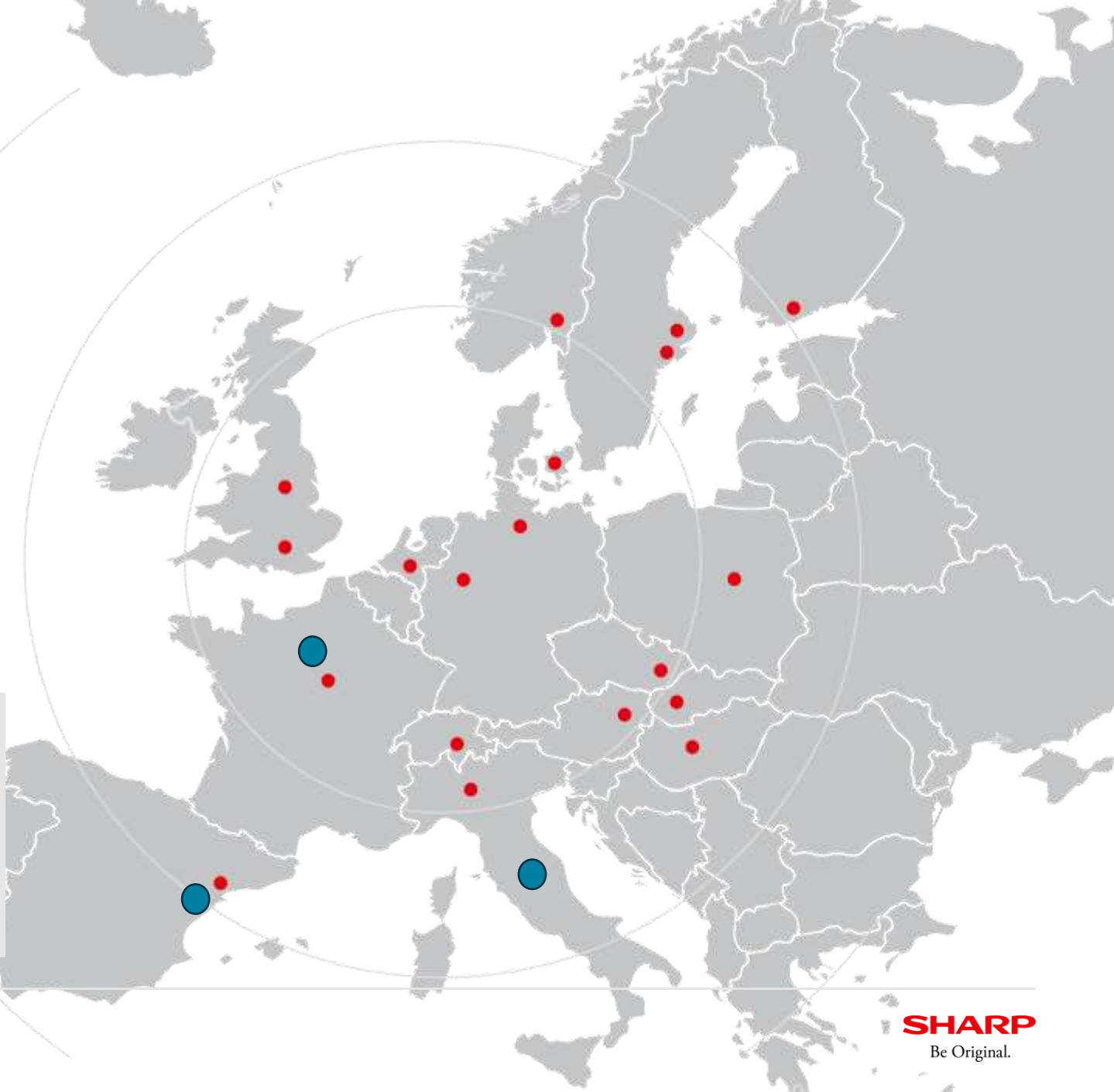
6.5%

of revenue to research
and development

Operating in

25+

countries



Foxconn Parentage

Foxconn is the **World's largest contracted electronics manufacturer**, building roughly two out of every three **iPhones**, along with devices including the **Google Pixel** and **Sony PlayStations**.

Foxconn acquired a majority stake in Sharp Corporation in 2016.

FOXCONN

More than

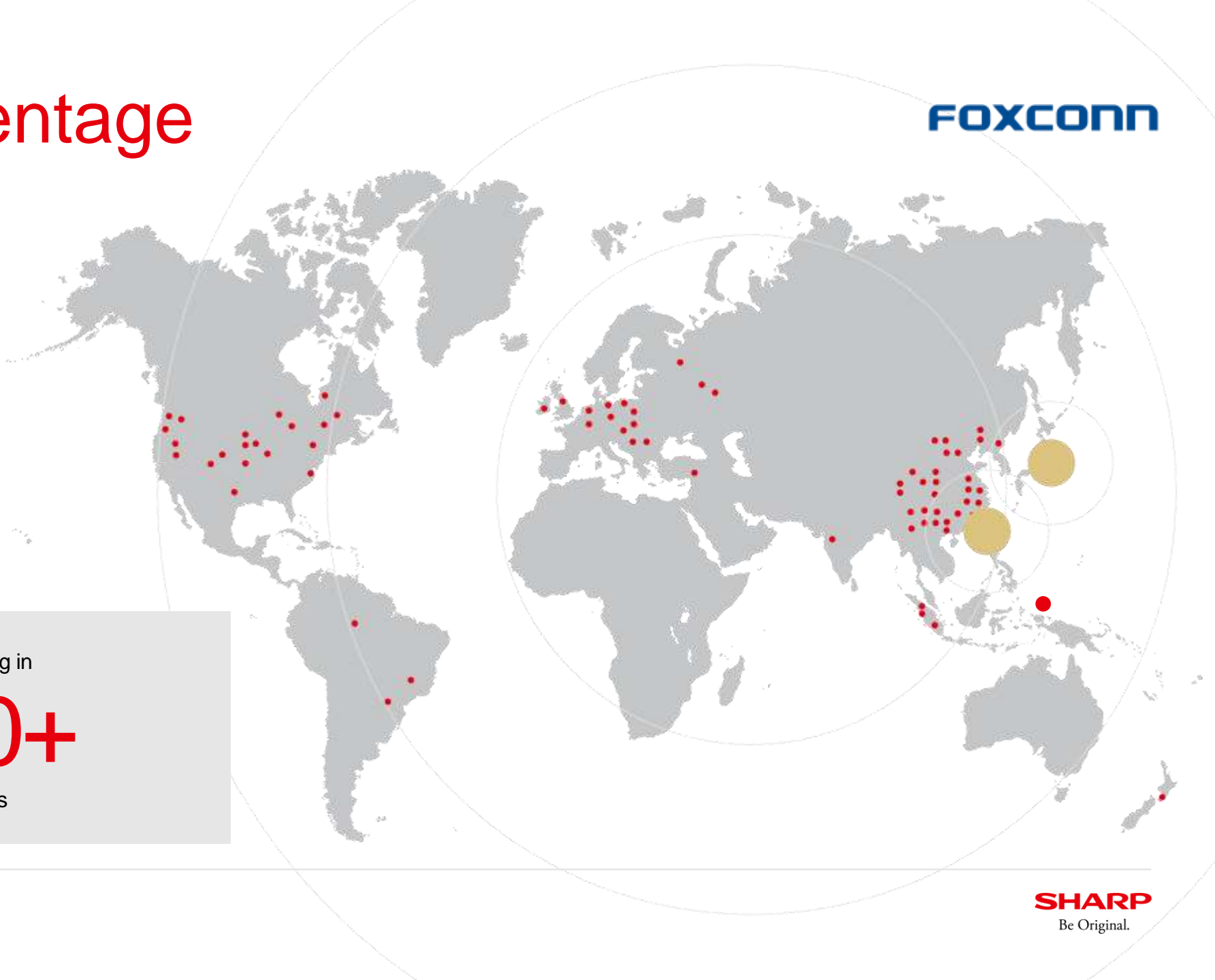
70,000

employees globally

Operating in

90+

countries



Past →

Our story begins way back in 1912 when our founder Tokuji Hayakawa had a rather smart idea. He wanted to start a company that would think of original ways to make life better for people. And so, from a small office in downtown Tokyo, he set to work. Dreaming. Creating. Being original.

One of the company's first success stories was the EverSharp pencil – which legend says inspired our name. More good ideas followed. And by 1969 Sharp UK had sprung up in sunny Manchester, offering a lot more than just pencils. A whole generation of kids grew up listening to the top 40 on our HiFis, doing their maths homework with our world first LCD calculator and watching the cup final on our TVs. You may even remember that Sharp sponsored a little-known football club named Manchester United who sported our logo for many a season.



1912

Belt Buckle

1915

Ever-Ready
Sharp pencil

1953

Sharp
TV s
Set

1962

Microwave
Oven

1963

Solar
Panels

1972

Photocopier

1973

LCD
Calculator

1982

Manchester
United
Sponsorship



Tech solutions built around your needs, not ours

From the big picture to the smallest detail, our consultative approach allows us to create the best combination of workplace technology products and services for your team.



IT Services



Managed Print Solutions



Audio Visual Solutions



Workplace Design

Sharp IT Services



SHARP
Be Original.

Seamless IT Services
that bring people and
technology together



IT Support



Cyber Security



Microsoft Copilot (AI)



Business Apps & ERP



Cloud Solutions



Unified Communications



Outlook



Word



Excel



PowerPoint



OneNote



OneDrive



Project

Productivity



Teams



SharePoint

Collaboration

And many more...



Integrate natively Dynamics 365 & the Power Platform with Microsoft 365



Microsoft Dynamics 365

Line of business applications



Marketing



Field Service



Commerce



Finance



Supply Chain



Sales



Customer Service



Human Resources



Project Operations

Customer Data Platform



Customer Voice



Customer Insights

Mixed Reality



Remote Assist



Guides

Small & Medium ERP



Business Central

Extend & customize with low code no code



Microsoft Power Platform

Low code App creation



Power Apps

Data analysis & insights



Power BI

Workflow Automation



Power Automate

Virtual Chat bots



Copilot

External facing website



Power Pages

Scalable, secure, and flexible data platform



AI Builder



Data connectors



Dataverse



Power FX



Managed Environments



Extend Dynamics 365 & the Power Platform with Azure

Cloud Infrastructure



Storage



Virtual Machines

Custom Development



App Services



Functions



DevOps

Data Analysis



Synapse Analytics

Artificial Intelligence



Machine Learning



Cognitive services



Open AI

Custom Integration



Logic Apps



Service Bus

Identity & Security



Active Directory



Active Directory B2C

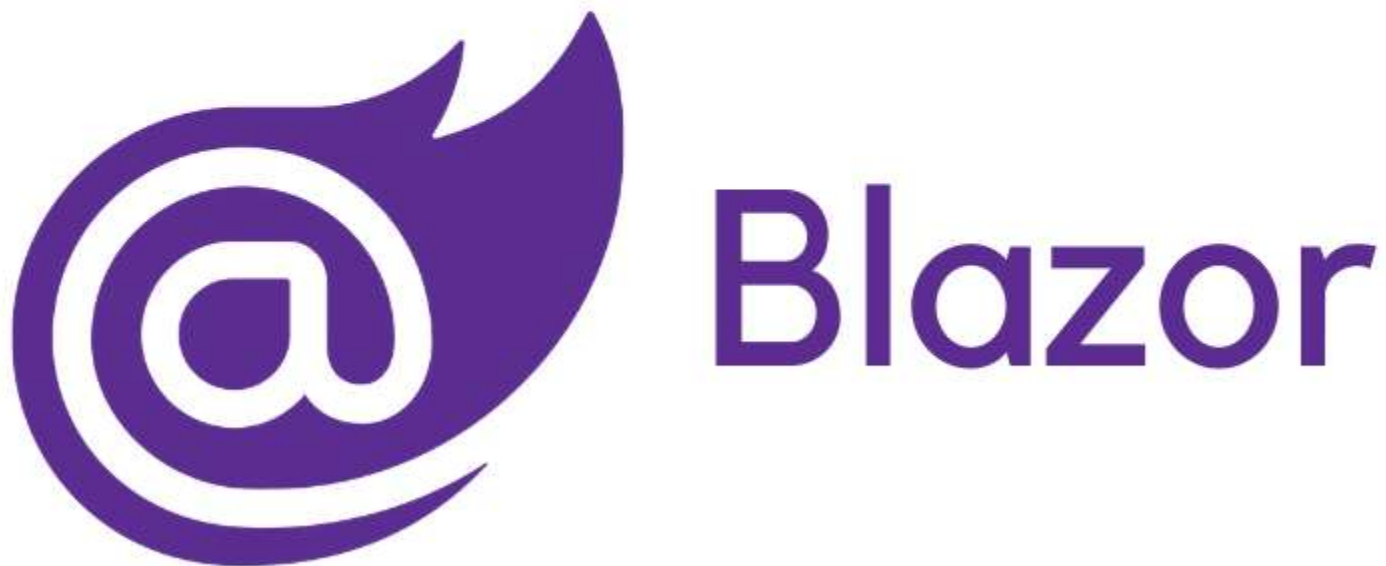
And many more...

Laboratorio di Programmazione Web II

#WeAreSharp

Corso di Laurea “**Programmazione e Gestione di Sistemi Informatici**”
AA 2024-2025





SHARP

Letizia Nasorri

Digital Transformation Analyst

email: letizia.nasorri@apsia.eu

- Sviluppo in Front End (**Power Pages**: Liquid, HTML, Javascript, CSS, **.NET: Blazor**)
- Sviluppo in Back End (**Power App**: C#, Python)
- Report SSRS e Power BI
- Rapporti e docenza **ITS Umbria Academy** e **Università degli Studi di Perugia**.



SHARP

Calendario e organizzazione

Febbraio 2025:

- 26/02

Marzo 2025:

- 05/03
- 12/03
- 19/03
- 26/03

Aprile 2025:

- 02/04
- 09/04
- 16/04
- 23/04(da posticipare)
- 30/04

Come struttureremo le lezioni?

- Lezioni da 4 ore.
- Faremo un'ora circa di lezione frontale in cui studieremo la teoria.
- Il tempo restante sarà dedicato alla pratica e alle esercitazioni.
- Troverete le slide alla fine di ogni lezione.

Materiale

Libro digitale «*Blazor for ASP.NET Web Forms Developers*», Daniel Roth, Jeff Fritz, Taylor Southwick. Potete scaricarlo gratuitamente da **Microsoft Learn**.
Documentazione ufficiale **Microsoft Learn**.

Troverete le fonti utilizzate in ogni blocco di slide.

Modalità d'esame

Modalità d'esame

- Progetto individuale.
- Inviare la tesina per e-mail a me e daniele.selvi@apsia.eu una settimana prima della data d'esame.
- L'esame prevederà la **presentazione del progetto e domande in merito alle scelte fatte**.
- L'ultimo giorno di lezione parleremo delle tesine e di spunti per i progetti.
- **Tesina**
 - Lunghezza massima: 5 pagine.
 - **Struttura:**
 - Descrizione del caso d'uso
 - Quale tipologia di applicazione è stato scelto di utilizzare e perché.
 - Schema dell'architettura (database, componenti).
 - Conclusioni e valutazioni: cosa è stato fatto? Cosa è migliorabile? Quali features potrebbero essere uno sviluppo futuro?
- **Il giorno dell'esame**
 - Portate il vostro laptop e presentate l'applicazione.
 - Qualche domanda in merito alle scelte fatte.

Blazor e .NET: una panoramica

1. Introduzione a Blazor e .NET
2. Architettura di Blazor
3. Blazor vs Angular
4. Blazor vs ASP .NET MVC
5. Ciclo di vita di un'applicazione Blazor

Fonti utilizzate per questa lezione

Esercitazione:

- <https://learn.microsoft.com/en-us/dotnet/architecture/blazor-for-web-forms-developers/>
- <https://learn.microsoft.com/it-it/training/modules/build-your-first-blazor-web-app/3-exercise-configure-environment?pivots=vstudio>

Teoria:

- <https://learn.microsoft.com/it-it/aspnet/core/blazor/?view=aspnetcore-9.0>
- <https://learn.microsoft.com/it-it/dotnet/core/introduction>
- <https://learn.microsoft.com/it-it/dotnet/csharp/tour-of-csharp/overview>
- <https://learn.microsoft.com/it-it/aspnet/mvc/overview/getting-started/introduction/getting-started>
- <https://angular.dev/overview>



Introduzione a Blazor e .NET

.NET

- Piattaforma per sviluppatori open-source gratuita e multiplatforma. Può eseguire programmi scritti in più linguaggi, anche se il più diffuso è C#.
- **Componenti:**
 - **Runtime:** esegue il codice dell'applicazione.
 - Librerie.
 - **Compilatore:** compila codice C# (e altri linguaggi) nel codice eseguibile.
 - **SDK e altri strumenti:** abilitano compilazione e monitoraggio dell'app mediante flussi di lavoro moderni.
 - **Stack di app.**

Come già accennato, il linguaggio principale per .NET è **C#**.



Introduzione a Blazor e .NET

C#

- C# è un linguaggio che si presta a svariati tipi di applicazioni: da **IoT a Cloud**, da applicazioni mobile, desktop, computer portatili e server.
- Offre un ampio supporto nell'ecosistema e in tutti i carichi di lavoro .NET.
- Si tratta di un linguaggio **orientato agli oggetti**.
- Fa parte della famiglia di linguaggi C, e la sua sintassi è familiare se si sono usati C, C++, JavaScript o Java.



Introduzione a Blazor e .NET

Feature principali di C#

- C# è un linguaggio **fortemente tipizzato**, il che significa che ogni variabile deve avere un tipo dichiarato esplicitamente, riducendo gli errori a runtime.
- C# supporta il paradigma **OOP** con classi, oggetti, ereditarietà, polimorfismo, incapsulamento e astrazione.
- Il **Garbage Collector** (GC) gestisce automaticamente la memoria, liberando quella non più utilizzata per evitare memory leaks.
- **LINQ** (Language Integrated Query) permette di eseguire query su collezioni, database e XML direttamente nel codice C#.
- C# supporta la programmazione asincrona con le **keyword async e await**, migliorando le performance delle applicazioni senza bloccare il thread principale.
- C# offre un modello robusto per la gestione delle eccezioni con **try-catch-finally** e supporta la scrittura di codice sicuro con controlli di accesso.



Introduzione a Blazor e .NET

ASP.NET Core Blazor

- Blazor è un **framework Web front-end .NET** che permette di sviluppare applicazioni web interattive utilizzando C# e .NET.
- Supporta due modalità di rendering:
 - **Blazor Server**: il codice viene eseguito sul server e le interazioni con il client avvengono tramite SignalR.
 - **Blazor WebAssembly**: l'applicazione viene eseguita direttamente nel browser grazie a WebAssembly, senza bisogno di un back-end attivo.
- Blazor semplifica lo sviluppo full-stack con C#, offrendo componenti riutilizzabili e un'integrazione diretta con **.NET**.
- Supporta componenti riutilizzabili, data binding, dependency injection e interoperabilità con JavaScript.



Introduzione a Blazor e .NET

Blazor Server vs Blazor WebAssembly: differenze principali

Blazor Server

- Il codice viene eseguito sul server, e le interazioni con il client avvengono tramite SignalR (una connessione in tempo reale).
- Le pagine si caricano più rapidamente perché il client scarica solo un'interfaccia leggera.
- Richiede una connessione costante con il server per funzionare correttamente.
- Adatto per applicazioni con accesso frequenti a dati server-side e dove la latenza non è critica.

Blazor WebAssembly

- L'intera applicazione viene scaricata ed eseguita direttamente nel **browser**, grazie a WebAssembly.
- Non necessita di una connessione continua con il server dopo il caricamento iniziale.
- Ha tempi di avvio più lunghi rispetto a Blazor Server, poiché deve scaricare il runtime .NET e l'app.
- Ideale per applicazioni con maggiore indipendenza dal server e per scenari offline.



Introduzione a Blazor e .NET

Componenti

- Le app Blazor si basano su **componenti**, ossia elementi dell'interfaccia utente.
- Sono riutilizzabili, modulari e definiti come fine con estensione **.razor**.
- Il rendering dei componenti viene eseguito in una rappresentazione in memoria del modello DOM del browser, l'albero di rendering, usato per aggiornare l'interfaccia utente in modo flessibile ed efficiente.

Caratteristiche principali

- **Markup e Logica Insieme:** Il codice C# e l'HTML sono combinati nello stesso file.
- **Data binding:** sincronizzazione automatica tra dati e UI con @bind.
- **Event Handling:** gestione degli eventi con @onclick, @onchange, ecc.
- **Componenti annidati:** un componente può contenere altri componenti.
- **Iniezione di dipendenze:** supporta dependency injection per servizi condivisi.
- **Ciclo di vita dei componenti:** metodi come OnInitialized e OnAfterRender permettono di gestire il comportamento del componente.



Introduzione a Blazor e .NET

Esempio di un componente semplice in Blazor

Vediamo come si presenta la struttura del componente Counter.razor, un componente di default creato nell'istanziare un'applicazione Blazor.

```
<PageTitle>Counter</PageTitle>

<h1>Counter</h1>

<p role="status">Current count: @currentCount</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
    }
}
```


Blazor e .NET: una panoramica

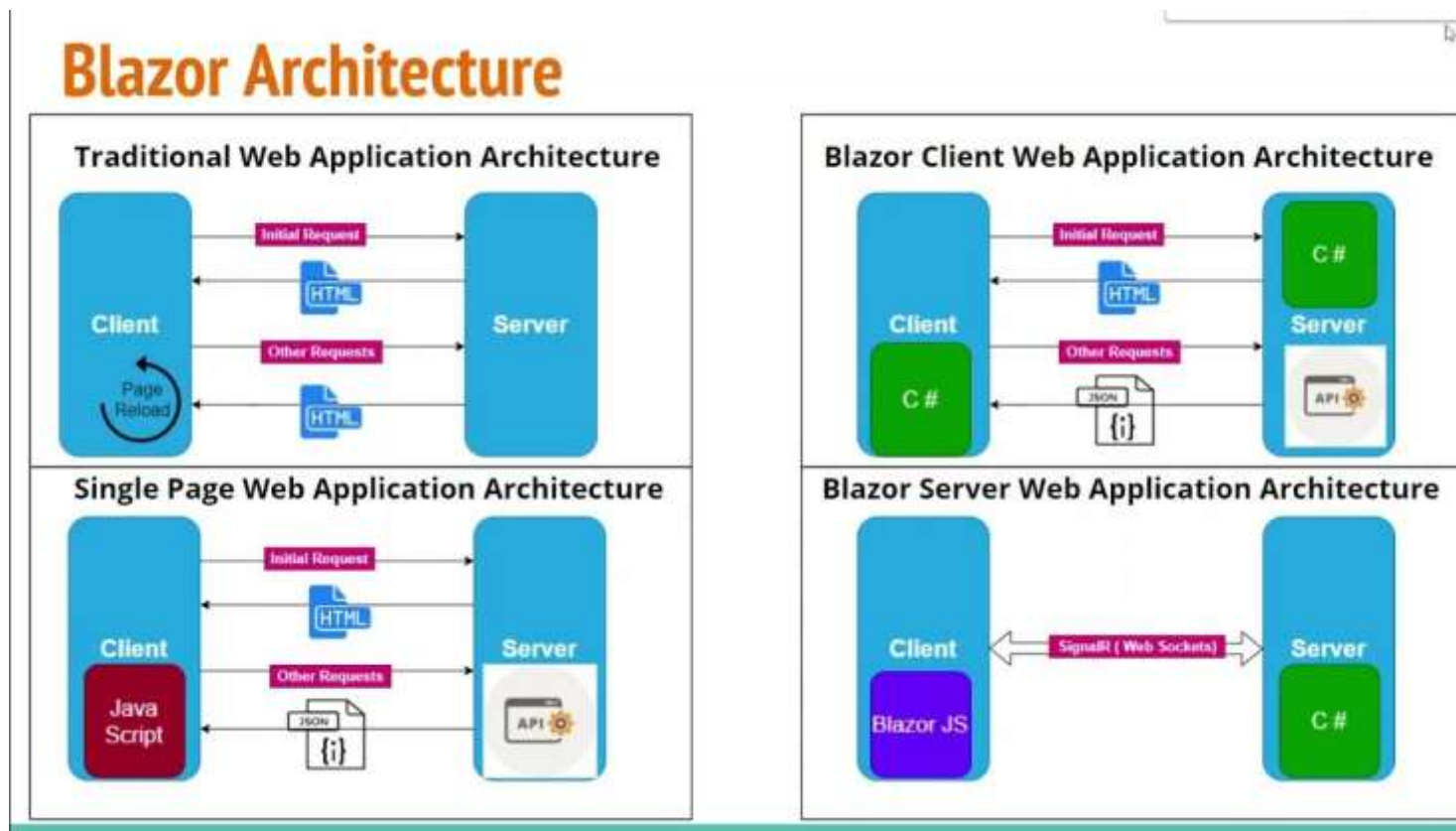
1. Introduzione a Blazor e .NET
2. Architettura di Blazor
3. Blazor vs Angular
4. Blazor vs ASP .NET MVC
5. Ciclo di vita di un'applicazione Blazor



Architettura di Blazor

Architettura di Blazor-Panoramica

Essendoci due diverse modalità di rendering, queste avranno anche architetture differenti. Vediamo quali sono le differenze anche in confronto ad applicazioni tradizionali.





Architettura di Blazor

Architettura di Blazor Server

Blazor Server utilizza un'architettura basata su:

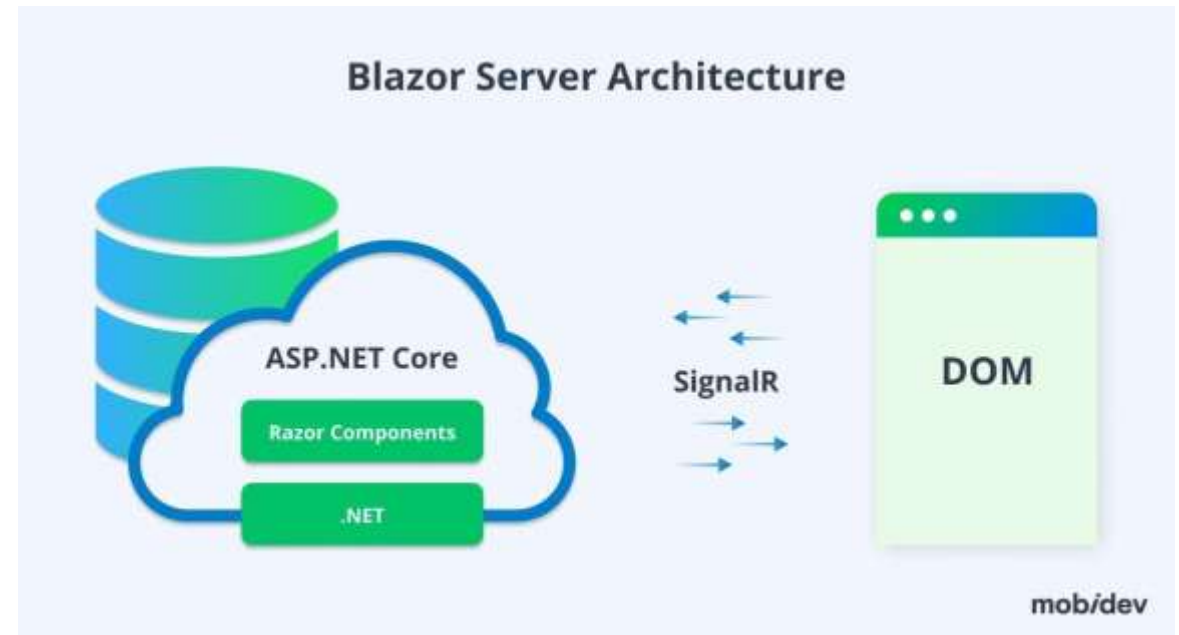
- **Rendering lato Server.**
- **Connessione costante.**
- **Aggiornamenti efficienti.**

Vantaggi

Caricamento veloce, ridotto uso di risorse client.

Svantaggi

Richiede connessione attiva. Latenza nelle interazioni.





Architettura di Blazor

Architettura di Blazor WebAssembly

Blazor WebAssembly permette di eseguire l'app direttamente nel browser.

- **Runtime .NET nel browser.**
- **Indipendenza dal server.**
- **Performance client-side.**

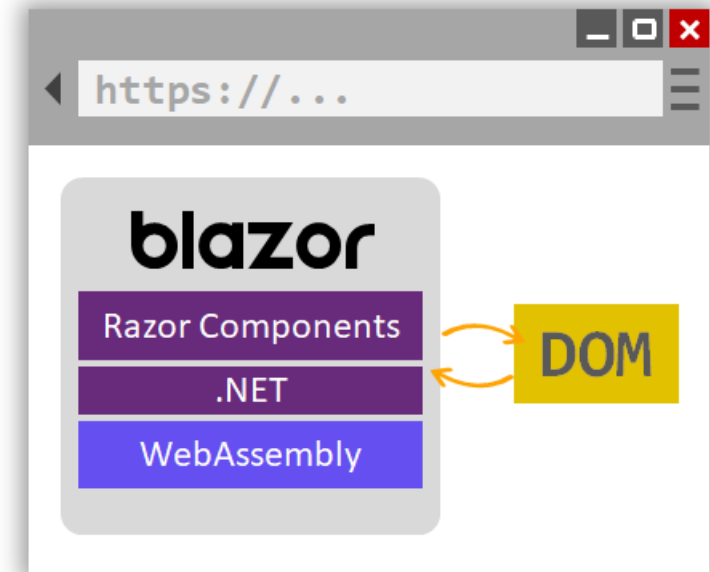
Vantaggi

Esperienza utente più fluida, indipendenza dalla connessione.

Svantaggi

Maggiori risorse necessarie lato client

Tempi di download iniziale più lunghi.



Blazor e .NET: una panoramica

1. Introduzione a Blazor e .NET
2. Architettura di Blazor
3. Blazor vs Angular
4. Blazor vs ASP .NET MVC
5. Ciclo di vita di un'applicazione Blazor



Blazor vs Angular

Blazor vs Angular

Per la nostra discussione, prendiamo in considerazione **Angular**, il framework di programmazione Web più famoso e consolidato.

Angular

Framework open source per sviluppo di **Single-Page application** sviluppato da Google.

Basato in **TypeScript**.

La sua architettura è basata su componenti e utilizza il pattern **MVC** (Model View Controller).

Gli elementi chiave sono:

- **Moduli.**
- **Componenti.**
- **Template e data binding.**
- **Service & Dependency Injection.**
- **Routing.**





Blazor vs Angular

Panoramica di Blazor vs Angular

Angular: Framework front-End open-source basato su TypeScript, usato principalmente per Single Page Applications.

Blazor: Framework sviluppato da Microsoft che consente di scrivere applicazioni Web interattive utilizzando C# anziché JavaScript, anche se vedremo che è possibile embeddarlo.

Differenze principali:

- **Linguaggio di programmazione:** Angular usa TypeScript (che è un superset di JavaScript), mentre Blazor usa C# con .NET.
- **Piattaforma:** Angular è focalizzato su JavaScript/TypeScript ed è indipendente dalla piattaforma, mentre Blazor è strettamente integrato nell'ecosistema .NET e può essere eseguito nel browser tramite WebAssembly o lato server.



Blazor vs Angular

Architettura di Angular vs Blazor

Angular:

- Architettura basata su **componenti**.
- Usa moduli (NgModules) per organizzare il codice.
- Data binding unidirezionale e due vie (two-way data binding).
- Gestione delle dipendenze tramite Dependency Injection.
- Routing per navigazione tra le pagine.

Blazor:

- Architettura basata anch'essa su componenti definibili tramite file .razor.
- Utilizza data binding per sincronizzare automaticamente la UI e il modello.
- Il codice può essere eseguito lato server o nel browser tramite WebAssembly.
- Supporta dependency injection per l'iniezione di servizi.
- Routing per la gestione della navigazione.

Differenze principali:

Angular ha una **gestione complessa** dei moduli e del routing, mentre Blazor ha una **gestione più semplice**, ma è legato all'ecosistema .NET.



Blazor vs Angular

Performance: Angular vs Blazor

Angular:

Usa DOM virtuale per ottimizzare le performance.

La gestione degli aggiornamenti del DOM è efficiente e minimizza il numero di modifiche.

Può avere tempi di caricamento maggiori per applicazioni molto complesse.

Blazor:

Blazor Server: Ogni interazione con la UI è gestita tramite una connessione SignalR al server, che può introdurre latenza nelle interazioni.

Blazor WebAssembly: L'applicazione viene eseguita direttamente nel browser, ma i tempi di avvio possono essere più lunghi a causa del caricamento del runtime .NET.

Differenze principali:

Angular ha un rendering più veloce nelle applicazioni client-side grazie al Virtual DOM, mentre **Blazor Server** può soffrire di latenze dovute alla connessione con il server.

Blazor WebAssembly offre un'ottima performance una volta caricato, ma ha tempi di avvio più lenti.



Blazor vs Angular

Ecosistema e Tooling: Angular vs Blazor

Angular:

Grande ecosistema con librerie e strumenti come Angular CLI.

RxJS per la gestione di eventi asincroni.

Ampio supporto della comunità e documentazione abbondante.

Integrazione con strumenti come Webpack per la gestione dei pacchetti.

Blazor:

Sfrutta l'ecosistema .NET, compreso ASP.NET e Entity Framework.

Ottima integrazione con strumenti di sviluppo come Visual Studio e Visual Studio Code.

Blazor WebAssembly ha un supporto in crescita, ma è ancora più giovane rispetto ad Angular.

Supporta anche PWA (Progressive Web Apps) tramite Blazor WebAssembly.

Differenze principali:

Angular ha un ecosistema molto maturo con un'ampia scelta di strumenti e librerie.

Blazor è ancora in fase di evoluzione ma beneficia delle potenzialità dell'ecosistema .NET e una forte integrazione con Visual Studio.



Blazor vs Angular

Linguaggio e Modello di Programmazione: Angular vs Blazor

Angular:

Basato su TypeScript, che è un superset di JavaScript, quindi ideale per sviluppatori con esperienza in JavaScript.

Usa RxJS per la programmazione reattiva e gestisce gli eventi asincroni tramite observables.

Blazor:

Utilizza C# e il modello di programmazione orientato agli oggetti (OOP).

La logica di business e l'interfaccia utente sono scritti nello stesso linguaggio (C#).

Integra facilmente i servizi di backend grazie alla stretta connessione con l'ecosistema .NET.

Differenze principali:

Angular si basa su TypeScript/JavaScript e la programmazione funzionale e reattiva, mentre Blazor usa C# e il modello OOP.



Blazor vs Angular

Facilità di Apprendimento: Angular vs Blazor

Angular:

Ha una curva di apprendimento più ripida rispetto a Blazor, soprattutto a causa della complessità del framework. Richiede una buona conoscenza di TypeScript, RxJS e dei concetti avanzati di programmazione reattiva.

Blazor:

Più semplice da imparare per gli sviluppatori C# e quelli già esperti nell'ecosistema .NET.

La curva di apprendimento è generalmente più bassa, soprattutto se si è già familiari con ASP.NET.

Differenze principali:

Blazor è generalmente più facile da imparare per chi ha esperienza con C# e .NET, mentre Angular ha una curva di apprendimento più ripida per via delle sue astrazioni e librerie.

Blazor e .NET: una panoramica

1. Introduzione a Blazor e .NET
2. Architettura di Blazor
3. Blazor vs Angular
4. Blazor vs ASP .NET MVC
5. Ciclo di vita di un'applicazione Blazor

Model View Controller

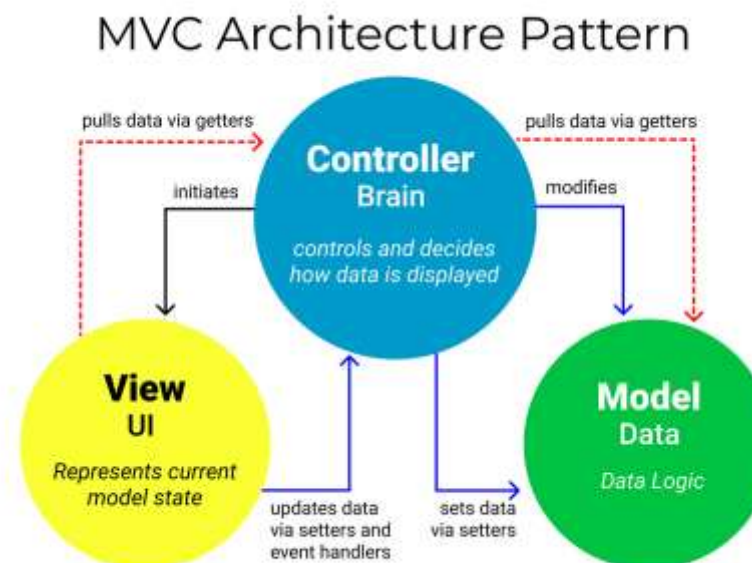
Pattern fondamentale nello sviluppo di sistemi software, questo modello deve essere alla base delle vostre conoscenze per quanto riguarda la programmazione orientata agli oggetti e in applicazioni web, dal momento che permette di **separare la logica di presentazione dei dati da quella di business**.

Comprende tre componenti principali:

- **Model:** rappresenta la logica di business e i dati.
- **View:** gestisce la visualizzazione dei dati e l'interfaccia utente.
- **Controller:** gestisce l'interazione dell'utente e coordina il flusso dati tra Model e View.

Flusso di lavoro

- L'utente interagisce con View.
- Il Controller riceve l'input e aggiorna il Model.
- Il Model modifica i dati, quindi aggiorna la View.





Blazor vs ASP .NET MVC

ASP .NET MVC

ASP.NET MVC è un framework web basato sul pattern Model-View-Controller (MVC), sviluppato da **Microsoft**. È parte dell'ecosistema ASP.NET e permette di costruire applicazioni web dinamiche, scalabili e facilmente manutenibili.

Caratteristiche principali:

La logica di business, la presentazione e l'interazione con l'utente sono separate.
Supporta la creazione di **Single Page Applications (SPA)**.
Estremamente flessibile per l'integrazione con altre tecnologie, come **Web API**.





Blazor vs ASP .NET MVC

Architettura di ASP.NET MVC

ASP.NET MVC è strutturato attorno al pattern Model-View-Controller:

Model: Rappresenta la logica di business e l'accesso ai dati (ad esempio, Entity Framework).

View: Gestisce la visualizzazione dei dati all'utente tramite **Razor Pages**.

Controller: Riceve le richieste dell'utente, interagisce con il Model e restituisce la View aggiornata.

Il framework offre **routing** flessibile, che mappa le URL alle azioni dei controller.





Blazor vs ASP .NET MVC

Architettura e Flusso dei Dati

Blazor Server:

L'applicazione viene eseguita sul server e la UI è sincronizzata con il client tramite SignalR (websocket-based). Tutti i dati e la logica di business sono gestiti sul server.

Blazor WebAssembly:

L'applicazione viene eseguita direttamente nel browser utilizzando WebAssembly. La logica di business è gestita nel browser senza necessità di connessione continua al server.

ASP.NET MVC:

Il flusso di dati segue una struttura Model-View-Controller, con il rendering della View eseguito sul lato server. I dati sono inviati e ricevuti tramite richieste HTTP standard (GET/POST).

Differenze principali:

Blazor Server richiede una connessione attiva al server, Blazor WebAssembly è completamente indipendente, mentre ASP.NET MVC dipende da richieste HTTP per ogni interazione.



Blazor vs ASP .NET MVC

Performance e Latenza

Blazor Server:

L'interfaccia utente viene aggiornata in tempo reale via SignalR.

Potenziale latenza nelle interazioni dovuta alla dipendenza dal server.

Caricamento iniziale più rapido, ma la performance può degradare con connessioni lente.

Blazor WebAssembly:

La logica viene eseguita nel browser, con WebAssembly.

Tempi di avvio più lunghi per il download del runtime e il caricamento iniziale.

Performance eccellente una volta caricato il runtime, poiché non dipende dalla connessione server.

ASP.NET MVC:

Le interazioni con il server avvengono tramite richieste HTTP.

Ogni azione dell'utente richiede un round-trip al server, il che può aumentare la latenza.

Differenze principali:

Blazor WebAssembly ha prestazioni superiori una volta caricata, ma con tempi di avvio maggiori. Blazor Server ha performance buone ma dipende dalla connessione, mentre ASP.NET MVC è meno interattivo per applicazioni moderne.



Blazor vs ASP .NET MVC

Facilità di Sviluppo e Manutenzione

Blazor Server:

Sviluppo centralizzato: La logica di business e la UI sono gestite nel server, riducendo la complessità lato client. Può risultare più facile da mantenere poiché tutto è centralizzato.

Blazor WebAssembly:

Sviluppo nel browser: La logica è gestita nel client, ma le applicazioni possono essere più complesse da debug e mantenere rispetto a quelle che girano sul server.

ASP.NET MVC:

Separazione chiara tra Model, View e Controller, che rende l'applicazione facile da gestire e mantenere. Richiede una buona organizzazione del codice per evitare complessità nella gestione dei dati.

Differenze principali:

Blazor Server centralizza la logica sul server, mentre Blazor WebAssembly offre maggiore flessibilità lato client. ASP.NET MVC ha una buona separazione dei compiti ma è meno interattivo rispetto a Blazor.



Blazor vs ASP .NET MVC

Conclusione

Blazor Server è ideale per applicazioni interattive con minor carico client-side, ma richiede una connessione attiva e può soffrire di latenza.

Blazor WebAssembly consente di costruire applicazioni indipendenti dal server, con un buon bilanciamento tra performance e interattività.

ASP.NET MVC è un'ottima scelta per applicazioni server-side tradizionali, ma non offre lo stesso livello di interattività e dinamismo rispetto a Blazor.

Scelta del Framework:

Blazor Server/WebAssembly per applicazioni moderne, interattive e scalabili.

ASP.NET MVC per applicazioni tradizionali che richiedono un approccio server-side.

Blazor e .NET: una panoramica

1. Introduzione a Blazor e .NET
2. Architettura di Blazor
3. Blazor vs Angular
4. Blazor vs ASP .NET MVC
5. Ciclo di vita di un'applicazione Blazor

Ciclo di vita di un'app Blazor

Ciclo di vita di un'applicazione Blazor

Il ciclo di vita di un'applicazione Blazor è il processo che gestisce la creazione, il rendering, l'aggiornamento e la distruzione dei componenti durante l'esecuzione. Ogni componente Blazor ha una serie di metodi del ciclo di vita che vengono invocati in sequenza, consentendo al framework di gestire l'interazione tra il client e il server (Blazor Server) o il client stesso (Blazor WebAssembly).

Obiettivo:

Comprendere i vari metodi e le fasi che definiscono il ciclo di vita dei componenti Blazor.

Ciclo di vita di un'app Blazor

Fasi principali del Ciclo di Vita

Creazione del Componente

Il componente viene istanziato dalla UI.

Settaggio e Inizializzazione

I parametri e le proprietà vengono iniettati nel componente.

Rendering

Il componente è renderizzato e la UI è aggiornata.

Aggiornamento

Quando i dati cambiano, il componente viene aggiornato.

Rimozione

Il componente viene distrutto quando non è più necessario.

Ciclo di vita di un'app Blazor

Metodi del Ciclo di Vita

I principali metodi che gestiscono il ciclo di vita di un componente Blazor sono:

OnInitialized():

Viene chiamato all'inizio del ciclo di vita del componente. Qui si esegue la logica di inizializzazione. Utilizzato per impostare variabili o eseguire chiamate asincrone (prima del rendering).

OnParametersSet():

Chiamato ogni volta che i parametri del componente vengono modificati. Utilizzato per gestire cambiamenti nei dati in input al componente.

OnAfterRender():

Viene chiamato subito dopo che il componente è stato renderizzato. Utilizzato per eseguire operazioni che devono avvenire dopo il rendering, come l'inizializzazione della libreria JavaScript.

OnAfterRenderAsync():

Variante asincrona di OnAfterRender(), che consente di eseguire operazioni asincrone dopo il rendering.

OnDestroy():

Chiamato prima che il componente venga distrutto. Utilizzato per la pulizia delle risorse o per fermare operazioni in corso, come timer o connessioni.

Ciclo di vita di un'app Blazor

Ciclo di Vita in Blazor Server vs WebAssembly

Blazor Server:

Il ciclo di vita del componente è gestito dal server.

SignalR è utilizzato per sincronizzare il rendering tra il server e il client.

Blazor WebAssembly:

Il ciclo di vita del componente è gestito completamente nel browser.

Ogni interazione con il componente avviene localmente nel client.

Differenze principali:

Blazor Server ha una comunicazione continua con il server, mentre Blazor WebAssembly è indipendente dal server dopo il caricamento iniziale.

Esercitazione 1

1. Creazione della prima applicazione Blazor Web App
2. Esploriamo i componenti e proviamo a customizzare le pagine.

Esercitazione 1

Cosa vi servirà

Visual Studio Community 


Visual Studio è un IDE completo per lo sviluppo .NET che semplifica la creazione di app Blazor.

Per usare Visual Studio per creare e usare app Web Blazor, assicurati di installare il carico di lavoro "ASP.NET e sviluppo web" usando il programma di installazione di Visual Studio.

Esercitazione 1


Workloads Individual components Language packs Installation locations

Web & Cloud (4)




ASP.NET and web development
Build web applications using ASP.NET Core, ASP.NET, HTML/JavaScript, and Containers including Docker supp...

☒




Python development
Editing, debugging, interactive development and source control for Python.

☐



Node.js development
Build scalable network applications using Node.js, an asynchronous event-driven JavaScript runtime.


☐



Azure development
Azure SDKs, tools, and projects for developing cloud apps and creating resources using .NET and .NET Framework....


☐

Desktop & Mobile (5)




.NET Multi-platform App UI development
Build Android, iOS, Windows, and Mac apps from a single codebase using C# with .NET MAUI.

☐




Desktop development with C++
Build modern C++ apps for Windows using tools of your

☐



.NET desktop development
Build WPF, Windows Forms, and console applications using C#, Visual Basic, and F# with .NET and .NET Frame...

☐



Universal Windows Platform development
Create applications for the Universal Windows Platform

☐

Location
C:\Program Files\Microsoft Visual Studio\2022\Preview [Change...](#)

By continuing, you agree to the [license](#) for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the [3rd Party Notices](#) or in its accompanying license. By continuing, you also agree to those licenses.

Remove out-of-support components

Total space required 6.84 GB

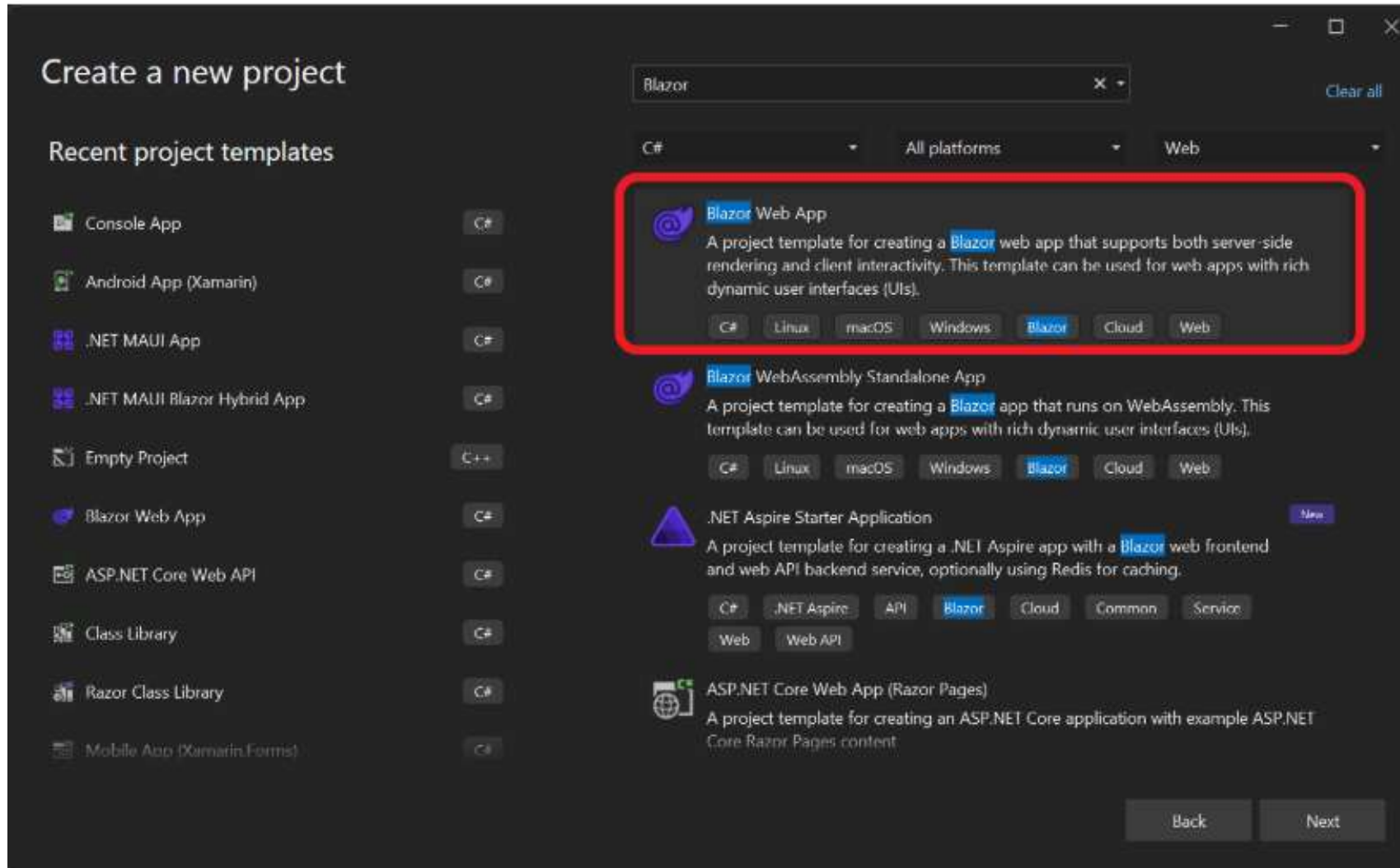
Install while downloading ▾ Install

Esercitazione 1

Per creare una nuova app Web Blazor con Visual Studio:

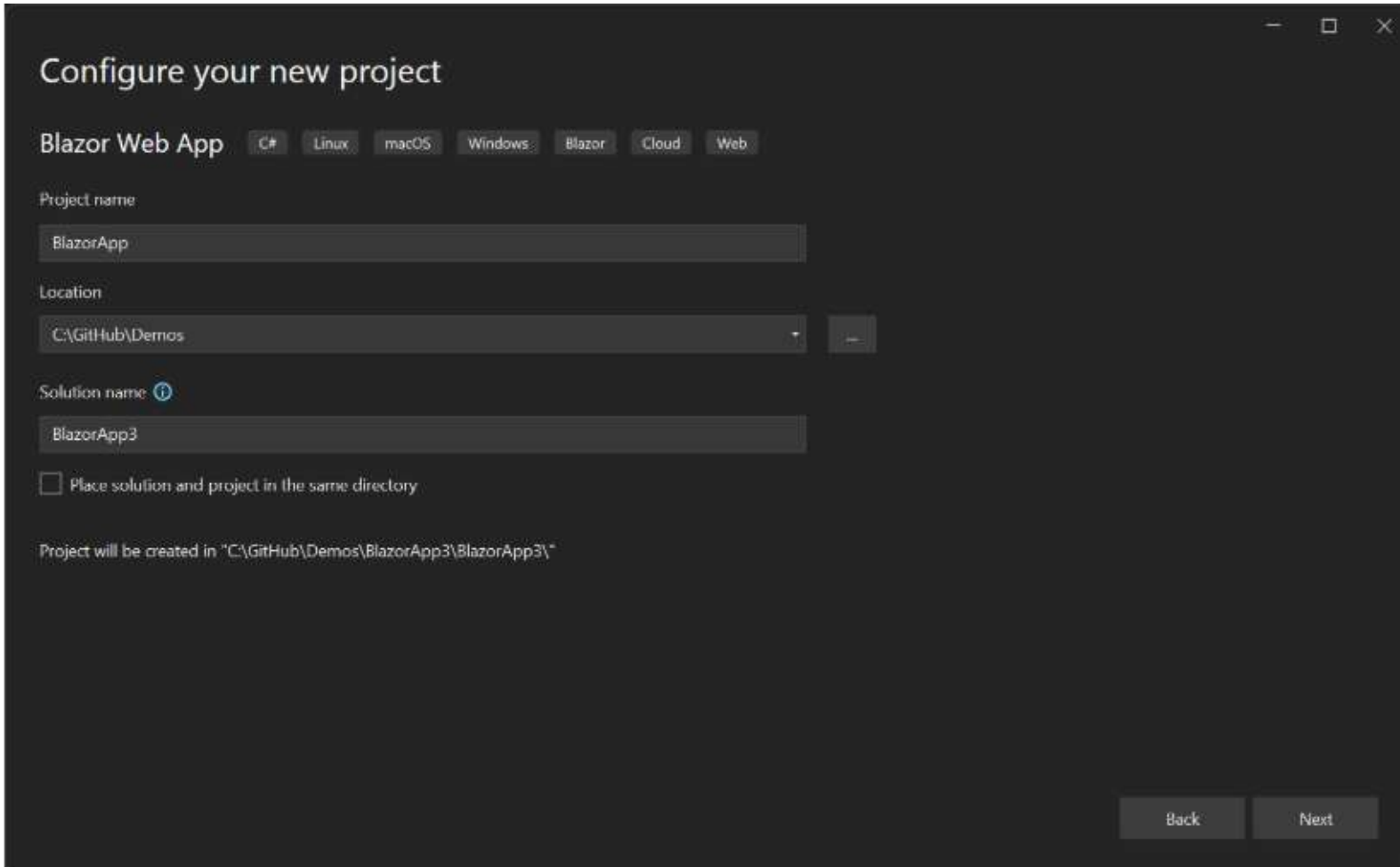
1. Avviare Visual Studio e selezionare **Crea un nuovo progetto**.
2. Nella finestra **Crea un nuovo progetto** digitare **Blazor** nella casella di ricerca e premere **INVIO**.
3. Seleziona il modello **Blazor Web App** e seleziona **Avanti**.

Esercitazione 1



Esercitazione 1

Nella finestra **Configura il nuovo progetto** immetti **BlazorApp** come nome del progetto e seleziona **Avanti**:



Configure your new project

Blazor Web App C# Linux macOS Windows Blazor Cloud Web

Project name
BlazorApp

Location
C:\GitHub\Demos

Solution name ⓘ
BlazorApp3

☐ Place solution and project in the same directory

Project will be created in "C:\GitHub\Demos\BlazorApp3\BlazorApp3\"

Back Next

Esercitazione 1

Nella finestra **Informazioni aggiuntive** seleziona **.NET 8.0 (supporto a lungo termine)** nell'elenco a discesa **Framework**, se non è già selezionato e fai clic sul pulsante **Crea**.

Quando crei un'app web Blazor, puoi selezionare tra varie opzioni, ad esempio se abilitare l'autenticazione, quali modalità di rendering interattive abilitare e in che misura vuoi che l'app sia interattiva. Per questa app, verifica che le impostazioni predefinite siano selezionate nel modo seguente:

- Tipo di autenticazione: Nessuna
- Modalità di rendering interattiva: Server
- Posizione di interattività: Per pagina/componente

Esercitazione 1

Additional information

Blazor Web App C# Linux macOS Windows Blazor Cloud Web

Framework ⓘ
_.NET 8.0 (Long Term Support)

Authentication type ⓘ
None

☒ Configure for HTTPS ⓘ

Interactive render mode ⓘ
Server

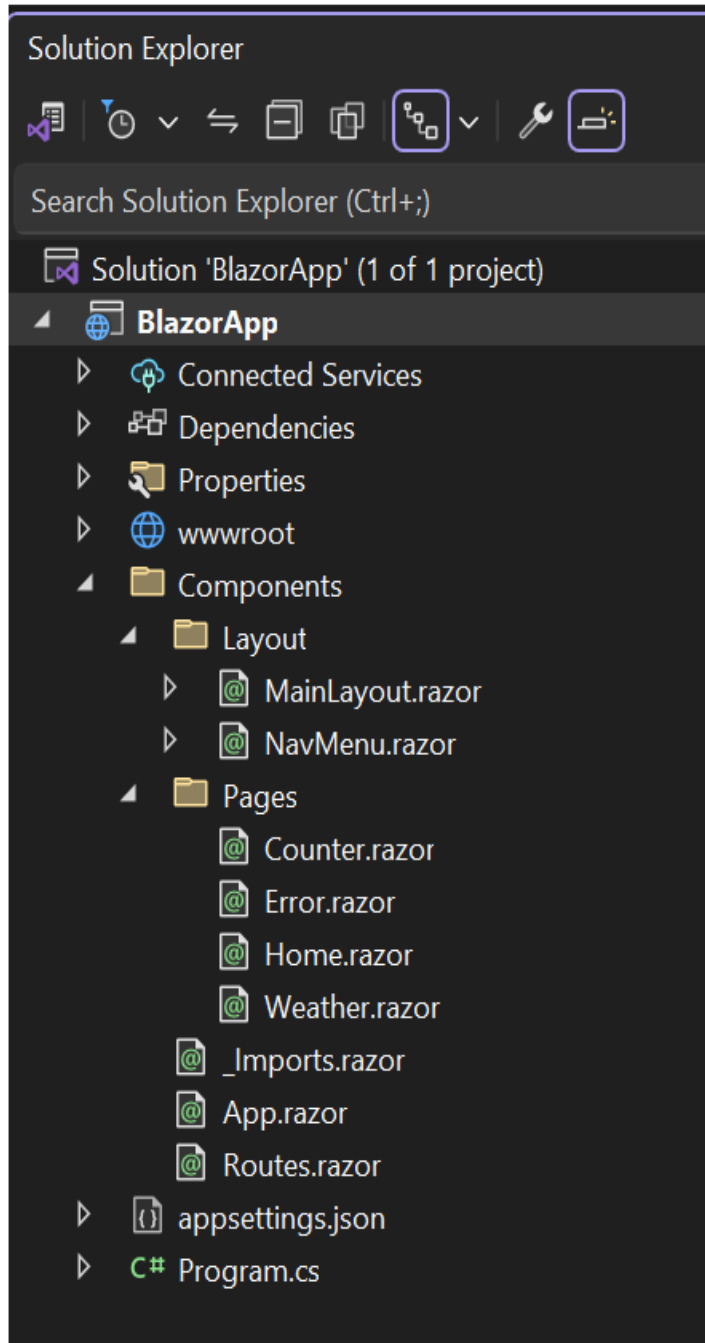
Interactivity location ⓘ
Per page/component

☒ Include sample pages ⓘ

☐ Do not use top-level statements ⓘ

Back Create

Esercitazione 1



Il tuo progetto viene creato e caricato in Visual Studio. Esamina il contenuto del progetto usando **Esplora soluzioni**.

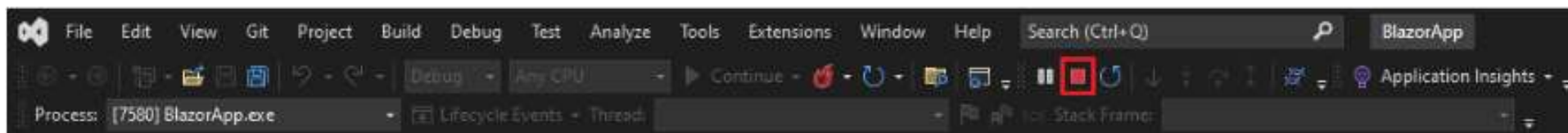
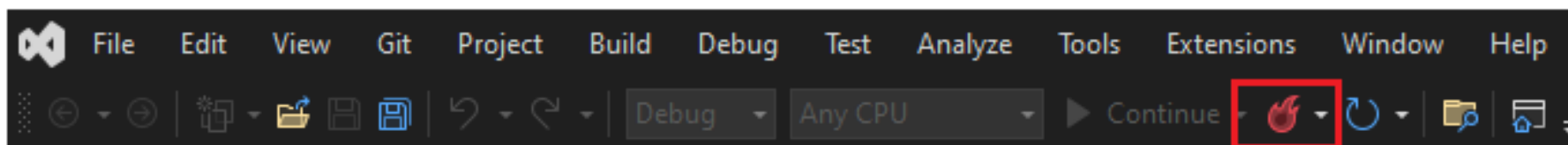
Esercitazione 1

Sono stati creati diversi file per offrire una semplice app Blazor pronta per l'esecuzione:

- ***Program.cs*** è il punto di ingresso per l'app che avvia il server e dove configuri i servizi app e il middleware.
- ***App.razor*** è il componente radice per l'app.
- ***Routes.razor*** configura il router Blazor.
- La directory ***Components/Pages*** contiene alcune pagine web di esempio per l'app.
- ***BlazorApp.csproj*** definisce il progetto dell'app e le relative dipendenze e può essere visualizzato facendo doppio clic sul nodo del progetto in Esplora soluzioni.
- Il file ***launchSettings.json*** all'interno della directory Proprietà definisce impostazioni di profilo diverse per l'ambiente di sviluppo locale. Al momento della creazione del progetto, viene assegnato automaticamente un numero di porta e salvato in questo file.

Esercitazione 1

Per avviare, modificare, arrestare la vostra app:



Esercitazione 1

La prima volta che si esegue un'app web in Visual Studio, verrà configurato un certificato di sviluppo per l'hosting dell'app tramite HTTPS e quindi verrà richiesto di considerare attendibile il certificato. È consigliabile accettare di considerare attendibile il certificato. Il certificato viene usato solo per lo sviluppo locale e senza che la maggior parte dei browser si lamenti della sicurezza del sito Web.

Attendi l'avvio dell'app nel browser. Una volta visualizzata la pagina seguente, hai eseguito correttamente la prima app Blazor.

Esercitazione 1

Componenti Razor

Dopo aver configurato l'ambiente di sviluppo, si esaminerà la struttura di un progetto Blazor e si apprenderà come funzionano i componenti Blazor.

Che cos'è Razor?

Razor è una sintassi di markup basata su HTML e C#. Un file Razor (con estensione razor) contiene codice HTML normale e quindi codice C# per definire qualsiasi logica di rendering, come istruzioni condizionali, flusso di controllo e valutazione delle espressioni. I file Razor vengono quindi compilati in classi C# che incapsulano la logica di rendering del componente.

Esercitazione 1

Componenti Razor

Utilizzo di componenti

Per usare un componente di un altro componente, aggiungere un tag di tipo HTML con un nome corrispondente al nome del componente. Ad esempio, se si dispone di un componente denominato `MyButton.razor`, è possibile aggiungere un componente `MyButton` a un altro componente aggiungendo un tag `<MyButton />`.

Parametri del componente

I componenti possono anche avere parametri, che consentono di passare i dati al componente quando viene usato. I parametri del componente vengono definiti aggiungendo una proprietà C# pubblica al componente che ha anche un attributo `[Parameter]`. È quindi possibile specificare un valore per un parametro del componente usando un attributo di tipo HTML che corrisponda al nome della proprietà. Il valore del parametro può essere qualsiasi espressione C#.

Esercitazione 1

Componenti Razor

Blocco @code

Il blocco @code in un file Razor viene usato per aggiungere membri di classe C# (campi, proprietà e metodi) a un componente. È possibile usare @code per tenere traccia dello stato del componente, aggiungere parametri del componente, implementare eventi del ciclo di vita del componente e definire gestori eventi.

Provate voi a capire la struttura del progetto e a effettuare modifiche sui vostri componenti.