# Introduction to Blazor

## Introduction

Blazor is a Microsoft framework that enables the building of interactive web applications using C# and .NET without needing JavaScript. It operates through two primary hosting models that define how components run and interact within applications.

### Component Types

#### Blazor Server Components

- Hosting: Runs on the server, where application logic and components operate centrally. Any updates are sent to the client in real time using SignalR, enabling low client-side memory usage.
- Advantages: Provides a fast initial load since only minimal data is loaded in the client browser. It suits applications needing real-time data (like dashboards) or enhanced data security since data is processed server-side.
- Ideal Use Cases: Collaborative tools, dashboards, or systems that need constant updates and data protection, such as healthcare or finance applications.

#### Blazor WebAssembly Components

- Hosting: This model runs entirely on the client side, utilizing WebAssembly to execute .NET code in the browser. It supports offline capabilities, as the application code is downloaded and runs locally.
- Advantages: After initial loading, it offers high responsiveness with no need for continuous server interaction, reducing server dependency and potentially lowering hosting costs.
- Ideal Use Cases are applications needing offline support or those performing local data processing, such as task management apps or client-heavy tools for e-commerce.

## Conclusion

Choosing between Blazor Server and WebAssembly depends on project requirements. Blazor Server is ideal for real-time, data-sensitive applications, while Blazor WebAssembly is well-suited for standalone applications requiring offline functionality and reduced server reliance.