Dr. Holger Schwichtenberg

ASP.NET Core Blazor 3.1/3.2: Blazor Server und Blazor WebAssembly

Moderne Single-Page-Web-Applications mit .NET, C# und Visual Studio

www.IT-Visions.de®
Dr. Holger Schwichtenberg

Buchversion:	2.25.0 (05.09.2020)
Verlag:	www.IT-Visions.de, Fahrenberg 40b, D-45257 Essen
Sprachkorrektur:	Matthias Bloch, M.A.
ISBN:	978-3-934279-34-6
Bezugsquelle gedruckt:	*www.amazon.de/exec/obidos/ASIN/3934279341/itvisions-21*
Bezugsquelle Kindle:	*www.amazon.de/exec/obidos/ASIN/B084FFQGJM/itvisions-21*
Bezugsquelle PDF:	*www.leanpub.com/Blazor*

Für Heidi, Felix und Maja

1 Inhaltsverzeichnis

2 Vorwort

Liebe Leserinnen und Leser,

ich entwickle Webanwendungen seit Mitte der 1990er Jahre. Zunächst habe ich damals mit dem Common Gateway Interface (CGI) und Perl, danach mit dem Internet Database Connector (IDC) und Active Server Pages (ASP) sowie Visual Basic Webclasses programmiert. Nach einer Zwischenetappe in der Java-Welt mit Java Server Pages (JSP), Servlets und Applets folgte dann für mich ASP.NET in diversen Ausprägungen (Webforms, AJAX, Dynamic Data und MVC), schließlich Silverlight, danach ganz viel JavaScript (mit jQuery, Angular und diversen anderen Bibliotheken). Seit dem Jahr 2016 verwende ich in unseren Softwareentwicklungsprojekten ASP.NET Core und seit 2019 auch Blazor.

Der Grund dieser Vorrede: Ich habe schon eine Menge Webframeworks gesehen und verspüre dennoch (oder auch gerade deswegen) einige Begeisterung für die "neuste Sau", die durch das Webdorf getrieben wird; diese "Sau" namens Blazor.

Seit einiger Zeit sind Blazor-Varianten (Blazor Server und Blazor WebAssembly) in einer stabilen Version verfügbar. Die nächste Version (5.0) wird im November 2020 erscheinen und ist in diesem Buch in der aktuellen Preview 8-Version auch bereits behandelt.

Alle Ausführungen im Buch und abgedruckten Beispiele gelten – sofern nicht ausdrücklich anders erwähnt – sowohl für Blazor Server als auch Blazor WebAssembly. Es gibt nur wenige Unterschiede zwischen beiden Architekturen und auf die wird an entsprechender Stelle im Buch hingewiesen.

Mit Blazor konnte ich schon einige SPA-Webanwendungen in Rekordzeit entwickelt. Wenn man lange Erfahrung mit den verschiedenen Microsoft-Webframeworks einerseits und JavaScript-basierten Frameworks andererseits hat, ist man mit Blazor wirklich sehr produktiv. Ich denke aber auch, dass Entwickler mit weniger Vorerfahrungen in Blazor eine gute Technik finden werden.

In den letzten Jahren haben wir bei www.IT-Visions.de immer mehr klassische Windows-Desktop-Entwickler bei unseren Kunden im deutschsprachigen Raum, die bisher mit Windows Forms oder WPF gearbeitet haben, auf Webtechniken umgeschult; getrieben von Chefs oder externen Beratern, die eine "Alles ins Web"- und "Cross-Platform"-Strategie verkündet haben. Mangels technischer Alternativen zur Entwicklung von Single-Page-Web-Applications (SPAs) war die Technikentscheidung für den Browser immer JavaScript bzw. TypeScript, mit Angular, React oder anderen Bibliotheken. Auf dem Server konnten die Entwickler oft in der .NET-Welt bleiben mit ASP.NET Core WebAPI. Manchmal stand aber auch hier JavaScript via node.js auf dem Plan. Es waren nicht wenige .NET-Entwickler, die auch nach intensiver Motivation und Schulung keinerlei Gefallen an der JavaScript-Welt finden konnten.

Blazor ist die Hoffnung für alle JavaScript-Hasser - wenngleich man hier zumindest in der ersten Zeit noch nicht ohne etwas JavaScript auskommen wird. Aber JavaScript ist wieder – wie früher – nur gelegentlich eingestreut und nicht mehr das Zentrum der Webanwendung.

Natürlich gibt es auch bei Blazor, auch bei Blazor Server, noch einige Hürden zu überwinden, wie immer bei den ersten Versionen einer neuen Technik.

Auch wenn die Zeit bei mir knapp ist, habe ich mich dennoch entschlossen, nochmal ein neues Fachbuch zu beginnen. Ich werde dieses Buch aber nicht mehr wie früher einige Monate im Kämmerlein schreiben, sondern genauso iterativ und agil weiterentwickeln, wie heutzutage auch die Webframeworks entstehen. Vor Ihnen liegt eine erste frühe Version dieses Buchs, die nur die wesentlichen Aspekte thematisiert und an vielen Stellen noch auf externe Quellen verweist.

Ich plane, in Zukunft weitere Versionen dieses Buchs zu veröffentlichen mit weiteren Aspekten zum Thema Blazor und auch zu den Basistechniken aus ASP.NET Core. Auch die Weiterentwicklungen von Blazor werde ich mit diesem Buch dokumentieren.

Dieses Fachbuch wird vertrieben auf folgenden Wegen (Ich nenne neben dem Verkaufspreis auch, wie viel – bzw. wenig – ich als Autor von den jeweiligen Händlern erhalte. Der Rest ist Gewinn der Händler):

- Gedruckt bei Amazon.de für 44,99 Euro (der Autor erhält 19,34 Euro): *www.amazon.de/exec/obidos/ASIN/3934279341/itvisions-21*
- Kindle-E-Book bei Amazon.de für 39,99 Euro (der Autor erhält 13,99 Euro): *www.amazon.de/exec/obidos/ASIN/B084FFQGJM/itvisions-21*
- PDF **inkl. Updates** bei Leanpub.com ab 39,99 Dollar (der Autor erhält ca. 28,75 Euro): *www.leanpub.com/Blazor*

Tipp: Sie können jederzeit Aktualisierungen des PDF-Buchs (gleiche Hauptversion) kostenfrei bei Leanpub.com beziehen. Amazon erlaubt dies leider nicht! Käufer bei Amazon können die PDF-Version zum Preis von 15,00 Dollar beziehen: *leanpub.com/Blazor/c/update2*

Ich habe mich für den Vertriebsweg des gedruckten Buchs über Amazon entschieden, weil ich dort ständig Updates zu dem Buch einreichen kann. Per Print-on-Demand erhalten Leser dann immer das topaktuelle Buch. Oft liefert Amazon dennoch am Tag nach der Bestellung das Buch schon aus. Der Vertrieb dieses Buch über klassische IT-Verlage, die leider heutzutage immer noch größere Auflagen vorproduzieren, sind für ein sehr agiles Softwareprodukt wie Blazor keine Alternative mehr.

Da diese Preise in Anbetracht der vielen Stunden Arbeit an diesem Werk leider nicht nennenswert dazu beitragen können, den Lebensunterhalt meiner Familie zu bestreiten, ist dieses Projekt ein Hobby. Dementsprechend kann ich nicht garantieren, wann es Updates zu diesem Buch geben wird. Ich werde dann an diesem Buch arbeiten, wenn ich neben meinem Beruf als Softwarearchitekt, Berater und Dozent und meinen sportlichen Betätigungen noch etwas Zeit für das Fachbuchautorenhobby übrig habe.

Falls mir in diesem Buch oder den zugehörigen Downloads menschliche Fehler passiert sind, möchte ich mich dafür schon jetzt in aller Form bei Ihnen entschuldigen. Bitte geben Sie mir einen freundlichen, genau beschriebenen Hinweis auf meine Fehler. Ich freue mich immer über konstruktives Feedback und Verbesserungsvorschläge. Bitte verwenden Sie dazu das Kontaktformular auf *www.dotnet-doktor.de*.

Ich helfe Ihnen gerne, Ihren eigenen Programmcode zu schreiben, aber ich hoffe, Sie verstehen, dass ich dies nicht ehrenamtlich tun kann. Wenn Sie **technische Hilfe, Beratung oder Schulung** zu ASP.NET Core Blazor oder anderen Themen rund um .NET, Visual Studio, Windows oder andere Microsoft-Produkte benötigen, stehe ich Ihnen im Rahmen meiner beruflichen Tätigkeit für die Firma *www.IT-Visions.de* gerne zur Verfügung. Bitte wenden Sie sich für ein Angebot an *Kundenteam@IT-Visions.de*. Bitte kontaktieren Sie die Firma aber nicht für Feedback und Verbesserungsvorschläge zu diesem Buch, da dieses Werk Privatsache ist.

Die Beispiele zu diesem Buch können Sie herunterladen auf der von mir ehrenamtlich betriebenen **Leser-Website** unter *www.IT-Visions.de/Leser*. Dort müssen Sie sich registrieren. Bei der Registrierung wird ein Losungswort abgefragt. Bitte geben Sie dort **Nightflyers** ein (siehe auch Kapitel "Programmcodebeispiele zum Download").

Herzliche Grüße aus Essen, dem Herzen der Metropole Ruhrgebiet

Holger Schwichtenberg

3 Über den Autor

- Studienabschluss Diplom-Wirtschaftsinformatik an der Universität Essen
- Promotion an der Universität Essen im Gebiet komponentenbasierter Softwareentwicklung
- Seit 1996 selbstständig als unabhängiger Berater, Dozent, Softwarearchitekt und Fachjournalist
- Fachlicher Leiter des Expertenteams bei *www.IT-Visions.de*
- Chief Technology Expert (CTE) der Softwareentwicklung bei der MAXIMAGO GmbH in Dortmund (*www.MAXIMAGO.de*)
- Über 70 Fachbücher beim Carl Hanser Verlag, bei O'Reilly, Microsoft Press, APress und Addison-Wesley sowie mehr als 1300 Beiträge in Fachzeitschriften
- Gutachter in den Wettbewerbsverfahren der EU gegen Microsoft (2006-2009)
- Ständiger Mitarbeiter der Zeitschriften iX (seit 1999), dotnetpro (seit 2000) und Windows Developer (seit 2010) sowie beim Online-Portal heise.de (seit 2008)
- Regelmäßiger Sprecher auf nationalen und internationalen Fachkonferenzen (z.B. Microsoft TechEd, Microsoft Summit, Microsoft IT Forum, BASTA, BASTA-on-Tour, enterJS, .NET Architecture Camp, Advanced Developers Conference, Developer Week, OOP, DOTNET Cologne, MD DevDays, Community in Motion, DOTNET-Konferenz, VS One, NRW.Conf, Net.Object Days, Windows Forum, Container Conf)
- Zertifikate und Auszeichnungen von Microsoft:
  - Microsoft Most Valuable Professional (MVP)
  - Microsoft Certified Solution Developer (MCSD)
- Thematische Schwerpunkte:
  - Softwarearchitektur, mehrschichtige Softwareentwicklung, Softwarekomponenten, SOA
  - Visual Studio, Continous Integration, Continous Delivery, Azure DevOps
  - Microsoft .NET Framework, .NET Core, C#, Visual Basic
  - .NET-Architektur/Auswahl von .NET-Technologien
  - Einführung von .NET Framework/.NET Core und Visual Studio/Migration auf .NET
  - Webanwendungsentwicklung und Cross-Plattform-Anwendungen mit HTML, ASP.NET, JavaScript/TypeScript und Webframeworks wie Angular und Blazor
  - Enterprise .NET, verteilte Systeme/Webservices mit .NET, insbesondere Windows Communication Foundation, WebAPI und gRPC
  - Relationale Datenbanken, XML, Datenzugriffsstrategien
  - Objektrelationales Mapping (ORM), insbesondere ADO.NET Entity Framework und EF Core
  - Windows PowerShell, PowerShell Core und Windows Management Instrumentation (WMI)
- Ehrenamtliche Community-Tätigkeiten:
  - Vortragender für die International .NET Association (INETA)
  - Betrieb diverser Community-Websites: *www.dotnet-lexikon.de*, *www.dotnetframework.de*, *www.windows-scripting.de*, *www.aspnetdev.de* u. a.
- Firmenwebsites: *www.IT-Visions.de* und *www.MAXIMAGO.de*
- Weblog: *www.dotnet-doktor.de*

- Kontakt für Anfragen zu Schulung und Beratung: **kundenteam@IT-Visions.de** Telefon **0201 / 64 95 90 - 50**
- Kontakt für Anfragen für Softwareentwicklungsprojekte: **hsc@MAXIMAGO.de** Telefon **0231 / 58 69 67 - 12**
- Kontakt für Feedback zu diesem Buch: *www.dotnet-doktor.de/Leserfeedback*

4 Über dieses Buch

4.1 Versionsgeschichte dieses Buchs

Die folgende Tabelle zeigt die Versionen, die von diesem Fachbuch erschienen sind, sowie die darin besprochenen Blazor-Versionen.

> **Hinweis:** Diese Tabelle ist eine wichtige Referenz für die Leser, die sich aktuelle Versionen des Buchs beschaffen (z.B. über Leanpub.com). Wenn Sie das Buch erstmalig lesen, können Sie dieses Kapitel überspringen.

Die Behandlung einer neuen Versionsnummer des Produkts und die daraus resultierende Änderung des Buchtitels erfordert eine gemäß Amazon-Richtlinien ein neues Buchprojekt. In diesem Fall wird die Versionsnummer des Buchs an der ersten Stelle hochgezählt (z.B. 1.4 auf 2.0)

Eine Änderung der Versionsnummer an der zweiten Stelle (z.B. 1.3 auf 1.4) sind Aktualisierungen oder Erweiterungen, die keine Titeländerung erfordern.

Ergänzungen der Versionsnummer an der dritten Stelle (z.B. 1.2.2 auf 1.2.3) sind kleine Korrekturen im Buch, die nicht explizit in dieser Versionstabelle erscheinen. Das Erscheinungsdatum auf der Titelseite entspricht dem Erscheinungsdatum der Unterversion, kann also von dem in der Tabelle genannten Erscheinungsdatum der übergeordneten Version abweichen.

Buchversion Datum Umfang	**Amazon.de-Preis für gedruckte Ausgabe**	**Amazon.de-Preis für Kindle-Ausgabe**	**Leanpub.com-Preis für PDF**	**Blazor-Version**	**Bemerkung zum Inhalt**
0.9 19.11.2019 134 Seiten	19,99 Euro	9,99 Euro	19,99 Dollar	3.0 RTM und 3.1 Preview 3	Grundversion des Buchs
0.10 21.11.2019 137 Seiten	19,99 Euro	9,99 Euro	19,99 Dollar	3.0 RTM und 3.1 Preview 3	Unterkapitel "Authentifizierung und Benutzerverwaltung/Autorisierung" ergänzt Kapitel "Routing und Navigation" ergänzt
0.11 25.11.2019 171 Seiten	19,99 Euro	9,99 Euro	19,99 Dollar	3.0 RTM und 3.1 Preview 3	Kapitel "Projektaufbau" um den Startcode einer Blazor Server- bzw. Blazor WebAssembly-Anwendung ergänzt Kapitel "Ereignisse" stark erweitert Unterkapitel "Authentifizierung und Benutzerverwaltung/ Eigene Authentifizierungs-provider" ergänzt Kapitel "Zustandsverwaltung" stark erweitert

					Kapitel "Komponenteneinbettung" erweitert Kapitel "Fallbeispiel MiracleList" ergänzt um Anmeldedialoge
1.0 05.12.2019 172 Seiten	19,99 Euro	9,99 Euro	19,99 Dollar	3.0 RTM und 3.1 RTM	Sprachliche Kontrolle ist erfolgt durch Korrektor Aktualisierungen auf 3.1 RTM Kapitel "Support für Blazor" ergänzt
1.1 17.12.2019 172 Seiten	19,99 Euro	9,99 Euro	19,99 Dollar	3.0 RTM und 3.1 RTM	Kapitel "Komponenten mit Code-Behind-Datei" erweitert
1.2 10.01.2020 202 Seiten	24,99 Euro	9,99 Euro	24,99 Dollar	3.0 RTM und 3.1 RTM	Neues Kapitel "Komponentendateien/Layoutseiten" Kapitel "Komponentendateien/ Komponenten mit Code-Behind-Datei" erweitert Neues Hauptkapitel "Installieren von Blazor-Anwendungen Neues Hauptkapitel "Serversystem- und Browserinformationen" Neues Hauptkapitel "Umstieg von ASP.NET Webforms zu ASP.NET Core Blazor"
1.3 15.01.2020 211 Seiten	27,50 Euro	9,99 Euro	24,99 Dollar	3.0 RTM und 3.1 RTM	Kapitel "Was ist ASP.NET Core Blazor?" erweitert Neues Hauptkapitel "Mobile Blazor Bindings"
1.4 01.02.2020 214 Seiten	27,50 Euro	9,99 Euro	24,99 Dollar	3.0 RTM und 3.1.1 RTM	Kapitel "Fallbeispiel MiracleList" erweitert
2.0 03.02.2020 225 Seiten	29,99 Euro	9,99 Euro	29,99 Dollar	3.1.1 RTM und 3.2 Preview 1	Aktualisiert auf 3.2 Preview 1 Kapitel "Was ist Blazor? Blazor WebAssembly" erweitert Kapitel "Projektaufbau" erweitert Kapitel "Fallbeispiel MiracleList" erweitert
2.1 04.02.2020 237 Seiten	29,99 Euro	9,99 Euro	29,99 Dollar	3.1.1 RTM und 3.2 Preview 1	Kapitel "Übersicht über die Beispiele" erweitert Kapitel "Razor Class Libraries (RCL)" erweitert

					Kapitel "Übersetzung und Debugging" erweitert Neues Kapitel "Tipps, Tricks und Tools"
2.2 05.02.2020 243 Seiten	29,99 Euro	9,99 Euro	29,99 Dollar	3.1.1 RTM und 3.2 Preview 1	Kapitel "Projektaufbau" erweitert Kapitel "Komponenten" erweitert Kapitel "Ereignisse" erweitert
2.3 12.02.2020 258 Seiten	29,99 Euro	9,99 Euro	29,99 Dollar	3.1.1 RTM und 3.2 Preview 1	Kapitel "Unterschiede zwischen Blazor WebAssembly und Blazor Server" erweitert Neues Kapitel "Performance-Vergleich" Kapitel "Formulare/ Eingabemasken" erweitert Kapitel "Authentifizierung und Benutzerverwaltung" erweitert Kapitel "Fallbeispiel MiracleList" erweitert Kapitel "Tipps, Tricks und Tools" erweitert
2.4 16.02.2020 282 Seiten	29,99 Euro	9,99 Euro	29,99 Dollar	3.1.1 RTM und 3.2 Preview 1	Kapitel "Routing und Navigation" erweitert Neues Unterkapitel "Genauere Fehlermeldungen aktivieren (Detailed Errors)" Unterkapitel "Einsatz von Telerik UI for Blazor" im Kapitel "Umstieg von ASP.NET Webforms zu ASP.NET Core Blazor" ergänzt
2.5 05.03.2020 290 Seiten	29,99 Euro	9,99 Euro	29,99 Dollar	3.1.2 RTM und 3.2 Preview 1	Unterkapitel "Komponenten ohne Razor nur aus Programmcode" ergänzt Neues Hauptkapitel "Benachrichtigungen mit ASP.NET Core SignalR in Blazor"
2.6 07.03.2020 295 Seiten	29,99 Euro	9,99 Euro	29,99 Dollar	3.1.2 RTM und 3.2 Preview 1	Kapitel "Benachrichtigungen mit ASP.NET Core SignalR in Blazor" erweitert Neues Unterkapitel "Tipps, Tricks & Tools/ Statusabfragen im Hintergrund"

2.7 12.03.2020 316 Seiten	29,99 Euro	9,99 Euro	29,99 Dollar	3.1.2 RTM und 3.2 Preview 2	Aktualisiert auf 3.2 Preview 2 Kapitel "Programmcodebeispiel zum Download" erweitert Kapitel "Authentifizierung und Benutzerverwaltung" erweitert
2.8 29.03.2020 322 Seiten	29,99 Euro	9,99 Euro	29,99 Dollar	3.1.3 RTM und 3.2 Preview 3	Aktualisiert auf 3.2 Preview 3 Neues Kapitel "Projektaufbau/Anlegen einer kompletten mehrschichtigen Projektstruktur mit der .NET CLI" Neues Kapitel "Zustandsverwaltung/Cookies" Neues Kapitel "Tipps & Tricks/Zugriff auf die lokale Zeit des Webbrowsers"
2.9 15.4.2020 334 Seiten	29,99 Euro	9,99 Euro	29,99 Dollar	3.1.3 RTM und 3.2 Preview 3	Unterkapitel "Einsatz von DevExpress UI for Blazor" im Kapitel "Umstieg von ASP.NET Webforms zu ASP.NET Core Blazor" ergänzt
2.10 24.04.2020 336 Seiten	29,99 Euro	9,99 Euro	29,99 Dollar	3.1.3 RTM und 3.2 Preview 5	Kapitel "Fallbeispiel MiracleList" erweitert
2.11 02.05.2020 338 Seiten	29,99 Euro	24,99 Euro (*)	29,99 Dollar	3.1.3 RTM und 3.2 RC	Aktualisiert auf RC-Version Neues Kapitel "Tipps & Tricks/Custom Boot Resource Loading"
2.12 22.05.2020 338 Seiten	29,99 Euro	24,99 Euro	29,99 Dollar	3.1.4 RTM und 3.2 RTM	Aktualisiert auf RTM-Version Neues Kapitel "Programmcodebeispiele zum Download/ Notwendige .NET SDK-Version"
2.13 01.06.2020 351 Seiten	29,99 Euro	24,99 Euro	29,99 Dollar	3.1.4 RTM und 3.2 RTM	Neues Kapitel "Tipps & Tricks/Progressive Web Applications (PWA) mit Blazor WebAssembly" Kapitel "Installation von Blazor-Anwendungen" erweitert Kapitel "Debugging in Blazor WebAssembly-Projekten" erweitert Kapitel "Mobile Apps mit Mobile Blazor Bindings" erweitert

2.14 10.06.2020 364 Seiten	29,99 Euro	24,99 Euro	29,99 Dollar	3.1.5 RTM und 3.2 RTM	Neues Unterkapitel "Anwendungsgröße und Startzeiten" Neues Unterkapitel "Fehlende Funktionen" Unterkapitel "Performance-Vergleich" aktualisiert Kapitel "Übersetzen und Debugging" erweitert Neues Unterkapitel "Klassenbibliotheken/ Code-Sharing zwischen Blazor Server und Blazor WebAssembly" Neues Unterkapitel "Installation/Self-Hosting"
2.15 17.06.2020 379 Seiten	29,99 Euro	24,99 Euro	29,99 Dollar	3.1.5 RTM und 3.2 RTM	Unterkapitel "Aufruf von JavaScript zu .NET" erweitert Neues Unterkapitel "Praxisbeispiel: Nutzung einer vorhandenen JavaScript-Bibliothek" Neues Unterkapitel "Tipps & Tricks/Micro-Apps mit Blazor"
2.16 18.06.2020 381 Seiten	39,99 Euro	29,99 Euro	29,99 Dollar	3.1.5 RTM und 3.2 RTM	Kapitel "Interoperabilität mit JavaScript" erweitert
2.17 19.06.2020 384 Seiten	39,99 Euro	29,99 Euro	29,99 Dollar	3.1.5 RTM und 3.2 RTM	Unterkapitel "Tipps & Tricks/Verzögerter Start einer Blazor WebAssembly-Anwendung" ergänzt Unterkapitel "Tipps & Tricks/Integration von Blazor-Anwendungen in andere Webanwendungen" ergänzt
2.18 05.07.2020 401 Seiten	39,99 Euro	29,99 Euro	29,99 Dollar	3.1.5 RTM und 3.2 RTM	Kapitel "Zustandsverwaltung/ Nutzung des Webbrowserspeichers" erweitert
2.19 25.07.2020 416 Seiten	39,99 Euro	29,99 Euro	29,99 Dollar	3.1.6 RTM und 3.2.1 RTM sowie .NET 5.0 Preview 7	Neues Kapitel "Komponenteneinbettung/Kaskadierende Parameter" Neues Kapitel "Komponenteneinbettung/Vorlagenbasierte Komponenten (Templated Components)" Kapitel "Dependency Injection" erweitert

2.20 27.07.2020 425 Seiten	39,99 Euro	29,99 Euro	29,99 Dollar	3.1.6 RTM und 3.2.1 RTM sowie .NET 5.0 Preview 7	Kapitel "Komponenteneinbettung" erweitert Kapitel "Formulare/ Eingabemasken" erweitert
2.21 29.07.2020 426 Seiten	39,99 Euro	29,99 Euro	29,99 Dollar	3.1.6 RTM und 3.2.1 RTM sowie .NET 5.0 Preview 7	Neues Kapitel "Was ist Blazor?/Ausblick auf Blazor 5.0"
2.22 10.08.2020 443 Seiten	44,99 Euro	39,99 Euro	39,99 Dollar	3.1.6 RTM und 3.2.1 RTM sowie .NET 5.0 Preview 7	Kapitel "Unterschiede zwischen Blazor WebAssembly und Blazor Server" erweitert Neues Kapitel "Komponenten/Nicht-visuelle Komponenten" Neues Kapitel "Tipps & Tricks/Eigene Timer-Komponente" Kapitel "Razor Class Library" erweitert Kapitel "Tipps, Tricks und Tools" erweitert
2.23 11.08.2020 456 Seiten	44,99 Euro	39,99 Euro	39,99 Dollar	3.1.7 RTM und 3.2.1 RTM sowie .NET 5.0 Preview 7	Neues Kapitel "Programmcodebeispiel zum Download/Visual Studio 2019" Kapitel "Komponenten/ Ein-Datei-Komponenten" erweitert Kapitel "Dependency Injection (DI)" erweitert Neues Kapitel "Formulare/Datenbindung mit Datumsangaben" Neues Kapitel "Tipps, Tricks und Tools/ Meldungskomponente"
2.24 27.08.2020 471 Seiten	44,99 Euro	39,99 Euro	39,99 Dollar	3.1.7 RTM und 3.2.1 RTM sowie .NET 5.0 Preview 8	Kapitel "Projektaufbau" erweitert Neues Kapitel "Komponenten/CSS-Isolation" Kapitel "Interoperabilität mit JavaScript" erweitert Neues Kapitel "Klassenbibliotheken und Razor Class Libraries (RCL)/Lazy Loading"

					Neues Kapitel "Formulare/ Optionsfelder" Neues Kapitel "Tipps & Tricks/Toast-Benachrichtigungen"
2.25 05.09.2020 486 Seiten	44,99 Euro	39,99 Euro	39,99 Dollar	3.1.7 RTM und 3.2.1 RTM sowie .NET 5.0 Preview 8	Neue Kapitel "Authentifizierung/Umleitung für nicht-autorisierte Benutzer" Kapitel "Zugriff auf Webservices/WebAPIs" stark erweitert, insbesondere um gRPC-Dienste

(*) Leider war dies eine notwendige Preiserhöhung, da der Arbeitsaufwand der ständigen Aktualisierungen dieses Buchs sehr hoch ist. Die Preiserhöhung musste dabei leider so groß ausfallen, da Amazon bei einem Verkaufspreis bis 9,99 Euro 70% Tantiemen zahlt, bei einem höheren Verkaufspreis aber nur noch 35% ☹

4.2 Bezugsquelle für Aktualisierungen

Sie können jederzeit Aktualisierungen des PDF-Buchs (gleiche Hauptversion!) kostenfrei bei Leanpub.com beziehen.

Käufer der Kindle- oder Druck-Version können die aktuelle PDF-Version zum Preis von 15,00 Dollar (zzgl. 5% Mehrwertsteuer, ab 1.1.2021: 7%) unter folgender Webadresse beziehen:

https://leanpub.com/ASPNETBlazor/c/update2

Hinweise: Leider erlauben Amazon u.a. Buchhändler aufgrund der Buchpreisbindungsgesetze in Deutschland den Autoren grundsätzlich nicht, dass Leser eine Aktualisierung im Kindle-Format oder in gedruckter Form vergünstigt erhalten.

Bitte beachten Sie auch, dass die ISBN-Regularien erfordern, dass bei einer Titeländerung bei neuer Produktversion eine neue ISBN vergeben werden muss und damit auch ein neues Buchprojekt bei Amazon und Leanpub erstellt werden muss.

4.3 Geplante Kapitel

Die Reihenfolge der für folgende Buchversionen geplanten Kapitel ist hier zunächst alphabetisch angeordnet und entspricht nicht der Reihenfolge, in der die Kapitel erscheinen werden.

- (kommerzielle) Blazor-Erweiterungen
- Anwendungskonfigurationsdatei AppSettings.json
- Autorisierung mit Rollen
- Cross-Platform-HTML-Apps in Electron mit Blazor (Blazor Electron)
- Dateiupload
- Eigene Basisklassen für Razor Components
- Fehlerbehandlung / blazor-error-ui

- Fluent Validator (*https://github.com/Blazored/FluentValidation*)
- Host Environments verwenden
- IComponentActivator zur Kontrolle der Komponenteninstanziierung (ab Blazor 5.0)
- Installation von Blazor-Anwendungen auf Linux-Servern
- Integration mit komplexen JavaScript-Web-Frameworks wie Angular
- IL-Linker-Konfigurationsdatei (LinkerConfig.xml)
- Lokalisierung/Mehrsprachigkeit
- Leistungsoptimierung mit @key
- Microsoft.AspNetCore.Components.Web.Extensions (ab Blazor 5.0)
- Notifications im Browser
- Render Modus bei Blazor Server (Static, Server, ServerPrerendered)
- Token-basierte Authentifizierung für Blazor WebAssembly mit Microsoft Authentication Library (MSAL) in den NuGet-Paketen Microsoft.Authentication.WebAssembly.Msal und Microsoft.AspNetCore.Authentication.AzureAD.UI gegen Azure Active Directory (AAD) (seit 3.2 Preview 2)
- Testen von Blazor-Anwendungen (Microsoft.AspNetCore.Components.Testing und Razor.Components.Testing.Library)
- TypeScript in Blazor Apps
- Visual Studio Code als alternatives Werkzeug zu Visual Studio

Hinweis: Mit dieser Liste kommender Themen möchte ich gleichzeitig klarstellen, welche Themen Sie derzeit nicht im Buch finden, damit Sie als Leser keine falschen Erwartungen haben.

4.4 Programmiersprache in diesem Buch

Als Programmiersprache kommt in diesem Buch C# zum Einsatz, weil C# die einzige offiziell von Microsoft unterstützte Programmiersprache für Blazor ist.

Als zweite Programmiersprache für Blazor gibt es nur F# über eine Erweiterung [*https://github.com/fsbolero/Bolero*], die außerhalb von Microsoft entwickelt wird und daher kein offizielles Produkt von Microsoft ist.

4.5 Notwendige Vorkenntnisse

Webprogrammierung ist ein großes, komplexes Gebiet. In diesem Buch konzentriere ich mich auf ASP.NET Blazor im engeren Sinne.

Als Vorkenntnisse sollten Sie zumindest ein gutes Grundlagenwissen in folgenden Gebieten mitbringen:

- HTTP
- HTML
- CSS

- JavaScript
- .NET Core
- ASP.NET Core
- C#
- Entity Framework Core
- Visual Studio

Ich habe drei Bücher geschrieben, durch die Sie dieses Grundlagenwissen erlangen können.

Für die ersten fünf Punkte ist dies:

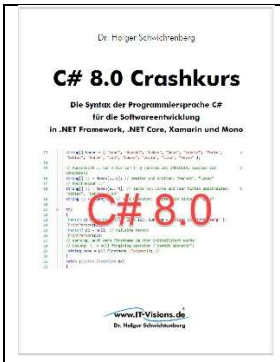
Moderne Webanwendungen für .NET-Entwickler: Server-Anwendungen, Web APIs, SPAs & HTML-Cross-Platform-Anwendungen mit ASP.NET, ASP.NET Core, JavaScript/TypeScript und Angular

ISBN 3960090153

https://www.amazon.de/exec/obidos/ASIN/3960090153/itvisions21

- Gedruckt: 49,90 Euro
- Kindle: 39,99 Euro

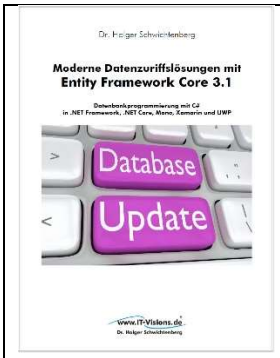
Wenn Sie C# lernen wollen, möchte ich Ihnen mein Buch "C# 8.0 Crashkurs" empfehlen:



C# 8.0 Crashkurs

- Gedruckt (Print-on-Demand) bei Amazon.de für 19,99 Euro (der Autor erhält 7,96 Euro): *www.amazon.de/exec/obidos/ASIN/3934279325/itvisions-21*
- Kindle-E-Book bei Amazon.de für 9,99 Euro (der Autor erhält 5,56 Euro): *www.amazon.de/exec/obidos/ASIN/B07XSLF1HG/itvisions-21*
- PDF-E-Book bei Leanpub.com ab 11,99 Dollar (der Autor erhält ca. 8,74 Euro): *www.leanpub.com/CSharp8*

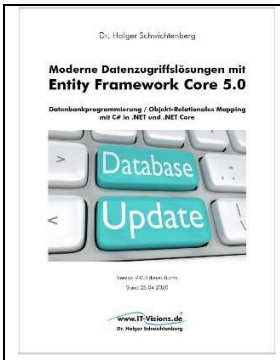
Um Entity Framework Core zu lernen, möchte ich Ihnen mein Buch "Moderne Datenzugriffslösungen mit Entity Framework Core 3.1" empfehlen:



Moderne Datenzugriffslösungen mit Entity Framework Core 3.1

- Gedruckt bei Amazon.de für 64,99 Euro (der Autor erhält 27,46 Euro):
  www.amazon.de/exec/obidos/ASIN/3934279244/itvisions-21
- Kindle-E-Book bei Amazon.de für 54,99 Euro (der Autor erhält 17,99 Euro):
  www.amazon.de/exec/obidos/ASIN/B081ZCJVH7/itvisions-21
- PDF-E-Book bei Leanpub.com ab 44,99 Dollar (der Autor erhält ca. 30,00 Euro):
  www.leanpub.com/EntityFrameworkCore31

Alternativ dazu gibt es auch bereits eine Ausgabe zu Entity Framework Core 5.0:



Moderne Datenzugriffslösungen mit Entity Framework Core 5.0

- Gedruckt bei Amazon.de für 69,99 Euro (der Autor erhält 29,90 Euro):
  www.amazon.de/exec/obidos/ASIN/3934279252/itvisions-21
- Kindle-E-Book bei Amazon.de für 59,99 Euro (der Autor erhält 18,52 Euro):
  www.amazon.de/exec/obidos/ASIN/B087QSSKW1/itvisions-21
- PDF-E-Book bei Leanpub.com ab 44,99 Dollar (der Autor erhält 35,99 Dollar):
  www.leanpub.com/EntityFrameworkCore5

4.6 Aktion "Buch-Abo"

Sie zahlen einmalig 99 Euro (zzgl. 5% Mehrwertsteuer bzw. 7% ab 1.1.2021) und erhalten

- alle meine aktuellen .NET-Bücher
- als PDF zum Download
- in der jeweils aktuellen Version
- UND: für ein Jahr lang alle Updates dieser Bücher
- ohne automatische Verlängerung! (sie entscheiden selbst nach einem Jahr, ob sie das Abo fortsetzen wollen)

Enthalten sind folgende Bücher:

- C# 8.0 Crashkurs (226 Seiten, Wert: ~14 €) [*]
- C# 9.0 Crashkurs (erscheint ca. September 2020) [*]
- ASP.NET Core Blazor 3.1/3.2: Blazor Server und Blazor Webassembly: Moderne Single-Page-Web-Applications mit .NET, C# und Visual Studio (380 Seiten, Wert: ~35 €) [*]
- ASP.NET Core Blazor 5.0 (erscheint Ende 2020) [*]
- Moderne Datenzugriffslösungen mit Entity Framework Core 5.0 (726 Seiten, Wert: ~44 €) [*]
- Moderne Datenzugriffslösungen mit Entity Framework Core 3.1 (723 Seiten, Wert: ~44 €)

- Moderne Datenzugriffslösungen mit Entity Framework 6.x (287 Seiten, Wert: ~25 €)

[*] Diese Bücher werden im Abstand von einigen Wochen aktualisiert jeweils bis zum Erscheinen des Nachfolgers. Der Nachfolger ist ebenfalls enthalten, sofern er innerhalb des Abo-Zeitraums erscheint.

www.IT-Visions.de/Buecher/Abo

4.7 Aktion "Buch für Buchrezension"

Ich möchte Sie animieren, eine Rezension dieses Fachbuchs bei Amazon.de zu schreiben. Als Dank dafür erhalten Sie kostenlos ein weiteres E-Book aus meiner Buchreihe.

So geht es:

- Sie schreiben bei Amazon.de eine Rezension zu diesem Fachbuch.
- Nach dem Erscheinen der Rezension besuchen die Webadresse *https://www.it-visions.de/buchrezension*
- Füllen Sie bitte das Formular aus. Geben Sie dabei in den Details insbesondere ihren Rezensionstext an, damit wir dies auf Amazon.de überprüfen können. **Sie müssen nicht Ihr Amazon-Konto angeben!**
- Das www.IT-Visions.de-Kundenteam sendet Ihnen nach der Überprüfung das E-Book (PDF-Format) des gewünschten Buchs per E-Mail.

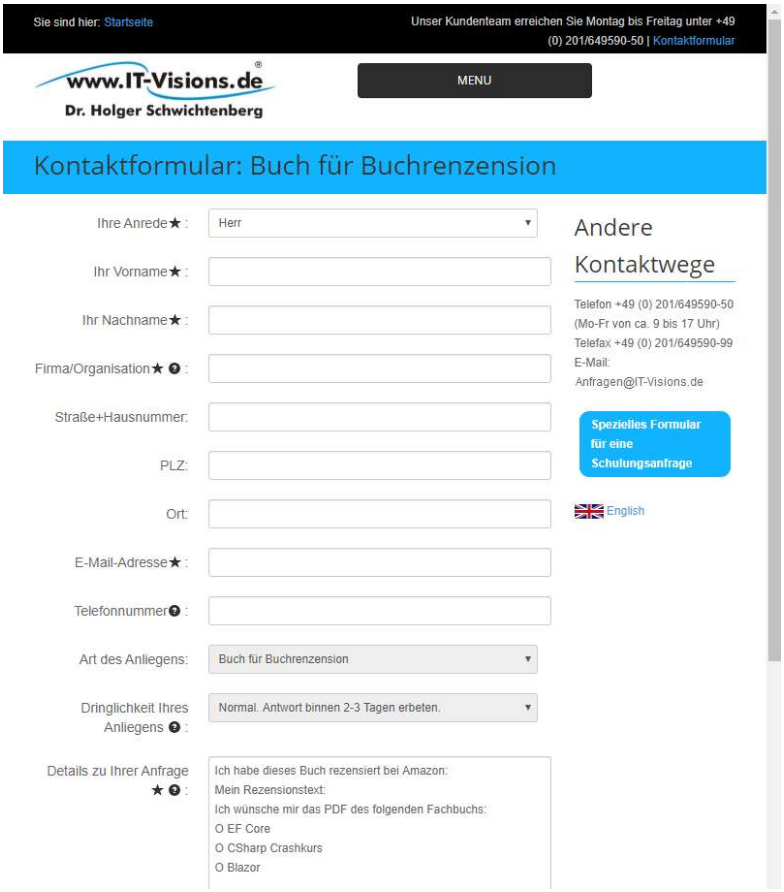


Abbildung: Webformular für die Aktion "Buch für Buchrezension"

5 Programmcodebeispiel zum Download

Die Beispiele zu diesem Buch können Sie als Visual Studio-Projekte herunterladen.

Hinweis: Bitte beachten Sie, dass nicht jede einzelne Zeile Programmcode, die Sie in diesem Buch finden, in den herunterladbaren Projekten enthalten sein kann. Die Projekte bilden funktionierende Lösungen. In diesem Buch werden auch alternative Lösungen für Einzelfälle diskutiert, die nicht unbedingt zu einer Gesamtlösung passen.

5.1 Webadresse für Downloads

Den Download der Beispiele zu diesem Buch finden Sie auf der Leser-Website unter der Webadresse:

www.dotnet-doktor.de/Leser

Dort müssen Sie sich bitte einmalig registrieren. Bei der Registrierung wird ein **Losungswort** abgefragt, das Sie als Käufer dieses Buchs ausweist. Bitte geben Sie dort **Nightflyers** ein.

Durch die Registrierung erhalten Sie dann ein persönliches **Kennwort** per E-Mail zugesendet, das Sie danach immer wieder für die Anmeldung zum Leserportal nutzen können.

Bei Problemen mit der Registrierung nutzen Sie bitte das auf der o.g. Webseite verlinkte FAQ.

5.2 Übersicht über die Beispiele

Die folgende Tabelle zeigt, wo Sie in dem herunterladbaren ZIP-Paket die Beispiele der einzelnen Kapitel des Fachbuchs finden.

Abbildung: Inhalt des Download-Pakets zu diesem Buch

Es war eine bewusste Entscheidung, in diesem Buch nicht für jedes einzelne Beispiel ein eigenes Blazor-Projekt zu schaffen, sondern alle Beispiele in zwei Blazor-Projekte (eins für Blazor WebAssembly und eines für Blazor Server) zu integrieren. Dies hat folgende Vorteile:

- Auf diese Weise sind komplexere Beispiele möglich.
- Die Aktualisierung der Beispiele auf neue NuGet-Paket-Versionen und API-Änderungen geht schneller (dies ist wichtig in der heutigen Zeit, in der es ständig neue Versionen, auch mit Breaking Changes, gibt).

Ordner	Programmcode aus Kapitel(n)	Hinweise

/src/MiracleList_SSB	Alle	Blazor Server-Beispiele einschließlich der Blazor Server-Implementierung des Fallbeispiel "MiracleList"
/src/MiracleList_CSB	Alle	Blazor WebAssembly-Beispiele einschließlich der Blazor WebAssembly-Implementierung des Fallbeispiel "MiracleList"
/src/MiracleListAPI_Proxy	Fallbeispiel	Generierte Proxy-Klasse für MiracleList-Backend-WebAPI
/src/ITVBlazorRCL	JavaScript-Interoperabilität Razor Class Library Fallbeispiel	Razor Class Library mit Code für JavaScript-Interoperabilität; wird im Fallbeispiel von Blazor Server und Blazor WebAssembly verwendet. Realisiert die vielfach in diesem Buch verwendete Hilfsklasse BlazorUtil.cs mit zugehöriger JavaScript-Datei BlazorUtil.js
/src/MLBlazorRCL	Alle	Diese Razor Class Library stellt gemeinsam genutzte Inhalte (Klassen, Razor Components sowie statische Webelemente wie Grafiken und eine CSS-Datei) für das MiracleList-Fallbeispiel und ergänzende Beispiele aus diesem Buch bereit. Dieses Projekt wird im Fallbeispiel von Blazor Server und Blazor WebAssembly gemeinsam verwendet.
/src/MiracleList_Backend	Fallbeispiel	Backend (ASP.NET Core WebAPI und ASP.NET Razor Pages)
/Blazor50_Neuheiten	Alle	Für Blazor 5.0 gibt es derzeit ein eigenes Beispielprojekt BW50_Demos.csproj mit neuen Funktionen in Blazor 5.0 (z.B. Lazy Loading, InputRadioGroup/InputRadio). Das MiracleList-Beispiel kann noch nicht auf Version 5.0 aktualisieren lässt, weil einige der dort verwendeten NuGet-Komponenten leider noch nicht in Blazor 5.0 funktionieren.
/BlazorWebservices	Zugriff auf Webservices/Web APIs	Die Projektmappe "BlazorWebservices" umfasst die Implementierung von WebAPIs und Google RPC-Diensten und den Aufruf aus Blazor-WebAssembly. Eigenständiges Beispiel, das die Datenbank der fiktiven Fluggesellschaft World Wide Wings (*www.world-wide-wings.de)* verwendet.

/WebformsToBlazor	Umstieg von Webforms auf Blazor	Eigenständiges Beispiel für den Umstieg von ASP.NET Webforms zu ASP.NET Core Blazor, das die Datenbank der fiktiven Fluggesellschaft World Wide Wings (*www.world-wide-wings.de*) verwendet.
/BlazorMicroApps	Integration von Blazor-Apps in eine Webforms-Anwendung	Eigenständiges Beispiel, das keine Datenbank verwendet, sondern auf den Standardprojektvorlagen mit Daten im RAM basiert.

5.3 Eingesetztes CSS-Framework

Alle Beispiele verwendet die CSS-Klassen des CSS-Frameworks Bootstrap [*https://getbootstrap.com*] zur Formatierung. Dies ist natürlich nur eine von vielen Getslatungsoptionen in Blazor.

5.4 Das Fallbeispiel "MiracleList"

Dieses Kapitel führt in das Blazor-Fallbeispiel "MiracleList" ein, das an vielen Stellen in diesem Buch verwendet wird.

5.4.1 Szenario

Das Fallbeispiel ist sehr praxisnah, weil das Anwendungsszenario ein Nachbau einer existierenden, sehr erfolgreichen Anwendung ist. Es gibt zwei Implementierungen: eine für Blazor Server und eine für Blazor WebAssembly.

Im Jahr 2015 zahlte Microsoft für die Übernahme des Berliner App-Herstellers Wunderlist mehr als 100 Millionen US-Dollar [*www.heise.de/newsticker/meldung/Microsoft-uebernimmt-Berliner-Startup-6Wunderkinder-2678017.html*].

"MiracleList" ist eine Nachprogrammierung dieser Aufgabenverwaltung als Webanwendung und Cross-Platform-Anwendung.

Hinweis: Microsoft hat mittlerweile Wunderlist neu implementiert als "Microsoft To-Do". Wunderlist wird am 6. Mai 2020 eingestellt. Das Fallbeispiel MiracleList existiert dessen ungeachtet weiterhin.

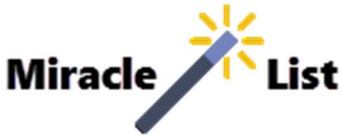


Abbildung: Das MiracleList-Logo

5.4.2 Technische Basis für MiracleList

Das MiracleList-Backend ist mit ASP.NET Core realisiert.

Die ursprüngliche Version des MiracleList-Frontends ist mit TypeScript und Angular realisiert. Die zwei in diesem Buch beschriebenen Frontend-Versionen basieren auf Blazor Server und Blazor WebAssembly.

5.4.3 Webadressen

Die fertige Webanwendung läuft öffentlich im Internet in der Microsoft-Cloud "Azure":

- Webanwendung "MiracleList"-Frontend mit Blazor Server und C#:
  https://miraclelist-bs.azurewebsites.net
- Webanwendung "MiracleList"-Frontend mit Blazor WebAssembly und C#:
  https://miraclelist-bw.azurewebsites.net
- Webanwendung mit Angular und TypeScript:
  https://miraclelist.azurewebsites.net
- Backend mit ASP.NET Core für Blazor WebAssembly- und Angular-Clients:
  https://miraclelistbackend.azurewebsites.net

5.4.4 Projektmappenaufbau

Die MiracleList-Projektmappe (MiracleList.sln) besteht aus zahlreichen Projekten, die logisch in fünf Ordnern sortiert sind:

- **Backend**: Alle Projekte für das MiracleList-Backend
- **Blazor-Frontends**: Die Blazor WebAssembly- und Blazor Server-Implementierung des Frontends mit Hilfsbibliotheken
- **Solution Items**: Hilfsskript für DevOps-Prozesse
- **Tests**: Unit Tests und Integrationstests
- **Tools**: Hilfsprojekte für das Anlegen der Datenbank im DevOps-Prozess und die ein Diagramm der Geschäftsobjektklassen

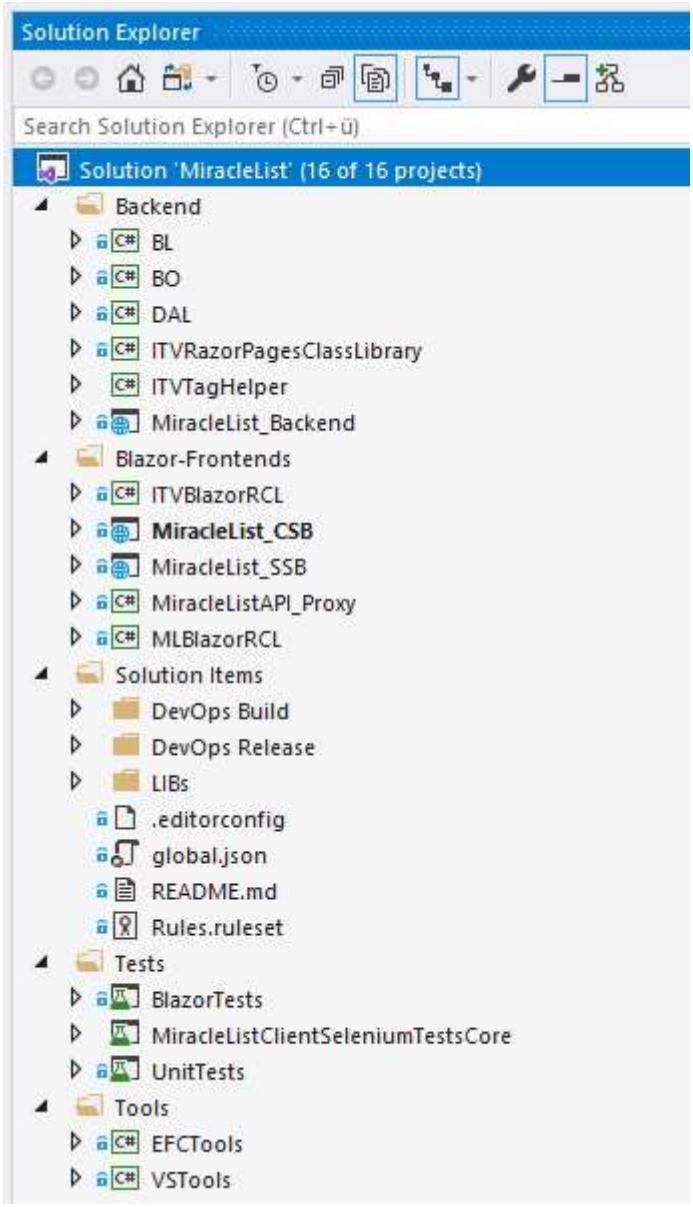


Abbildung: Aufbau der Projektmappe

5.4.5 MiracleList-Bildschirmmasken

Die Anmeldeseite fragt E-Mail-Adresse und Kennwort. Damit es leicht ist, MiracleList als Demonstrations-Anwendung zu nutzen, ist eine explizite Benutzerregistrierung ausdrücklich nicht vorgesehen. **Wenn es für diese E-Mail-Adresse noch kein Benutzerkonto gibt, wird automatisch ein neues Benutzerkonto angelegt mit einigen Beispielaufgaben.**

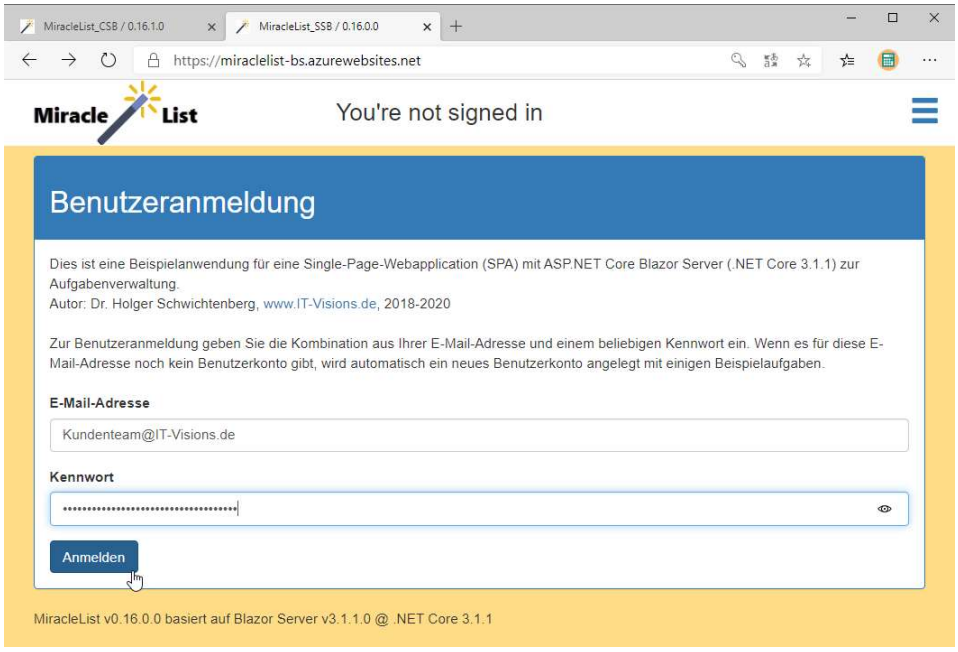


Abbildung: Anmeldeseite bei der Blazor Server-Implementierung des MiracleList-Frontends

Bei der Blazor WebAssembly-Implementierung wird zusätzlich eine Auswahl für den Backend-Server gegeben. Wenn die Blazor WebAssembly-Anwendung in Visual Studio gestartet wird, wird als Backend zusätzlich "localhost" zur Auswahl gestellt, um sich mit einer lokalen Version des Backends statt dem in Azure gehosteten Backend zu verbinden. Unter der Backend-Server sieht man eine Zeile mit Zustands-Abzeichen ("Badges") für die Server. Ein grünes Abzeichen bedeutet, dass der Server verfügbar ist. Grau bedeutet, der Zustand wird gerade ermittelt. Bei einem roten Abzeichen ist der Server nicht verfügbar.

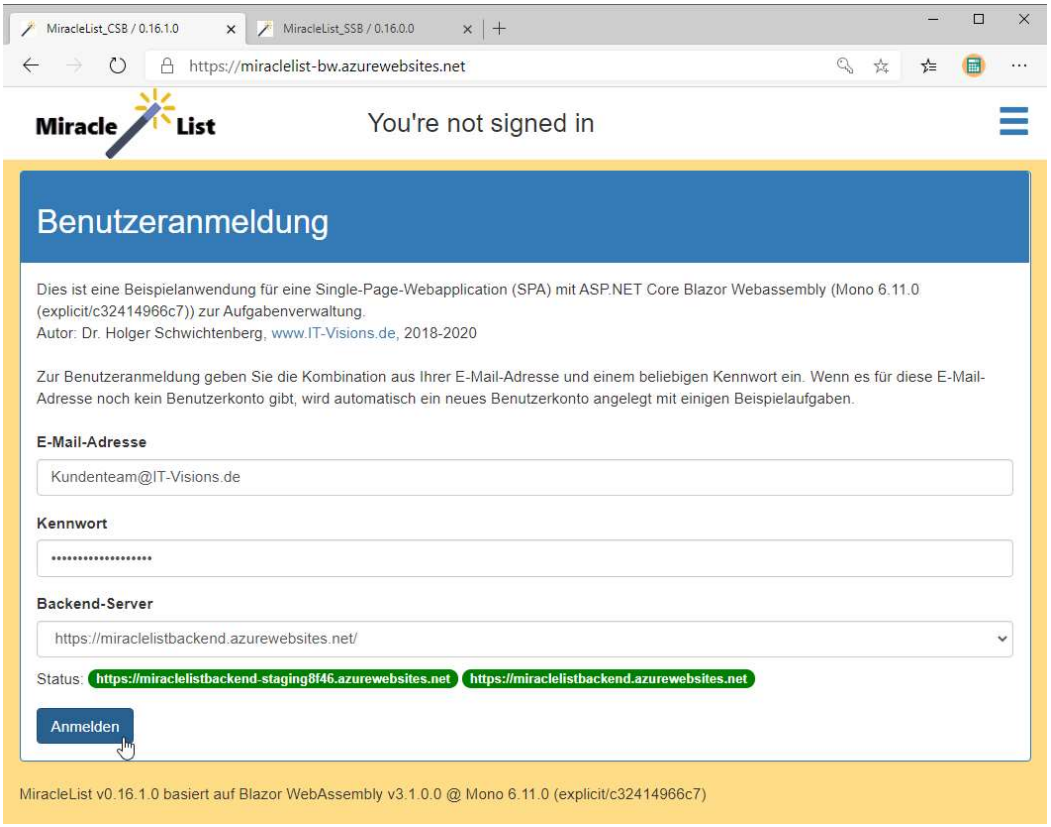


Abbildung: Anmeldeseite bei der Blazor WebAssembly-Implementierung des MiracleList-Frontends

Hinweis: Wenn Sie localhost als Backend-Server nutzen wollen, müssen Sie auf Ihrem Rechner das Backend (Projekt MiracleList_Backend.csproj) zuvor gestartet und auch die Datenbank angelegt haben (siehe dazu Readme.md)! Wenn Sie das Backend auf einem anderen Port laufen lassen, müssen Sie in AppSettings.cs den Port ändern!

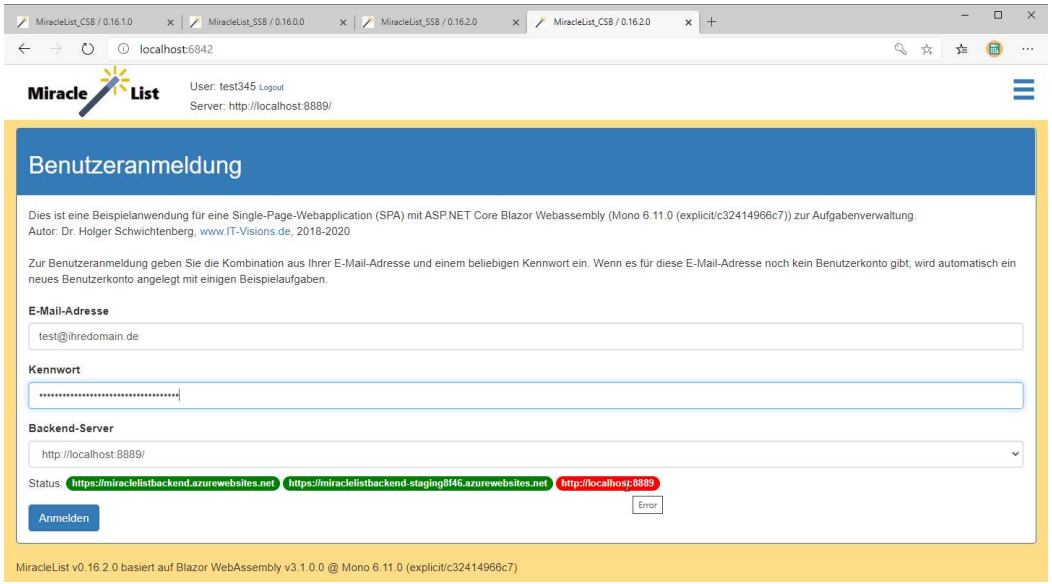


Abbildung: Beim Start der Blazor WebAssembly-Anwendung auf "localhost" wird auch das lokale Backend angeboten

Der angemeldete Benutzer kann eine Liste von Aufgabenkategorien erstellen und in jeder Kategorie eine Liste von Aufgaben anlegen.

Eine Aufgabe besteht aus einem Titel, einer Notiz, einem Eintragsdatum, einem Fälligkeitsdatum und kann als erledigt markiert werden. Über die Funktionen von Wunderlist hinaus kann in MiracleList eine Aufgabe drei (A, B oder C), statt nur zwei Wichtigkeitsgrade (Wichtig ja/nein) sowie einen Aufwand (Zahl) besitzen. Bewusst besitzt der Aufwand keine Maßeinheit; der Benutzer kann selbst entscheiden, ob er den Aufwand in Stunden, Tagen oder nur in relativen Werten, wie z.B. "1" (für niedrig) bis "10" (für hoch), vergeben will.

Wie bei Wunderlist kann eine Aufgabe Teilaufgaben besitzen, wobei eine Teilaufgabe nur einen Titel und einen Status besitzt. Einige Details aus dem Original fehlen aber in MiracleList, z.B. das Hochladen von Dateien zu Aufgaben, das Verschieben von Aufgaben zwischen Kategorien, die Suche nach Hashtags, das Duplizieren und Drucken von Listen sowie der Aufgabenaustausch zwischen Benutzern. Einige Funktionen wie anklickbare Hyperlinks in Aufgabentexten sind nicht realisiert, um einen Missbrauch der für alle Nutzer offenen Website zu vermeiden.

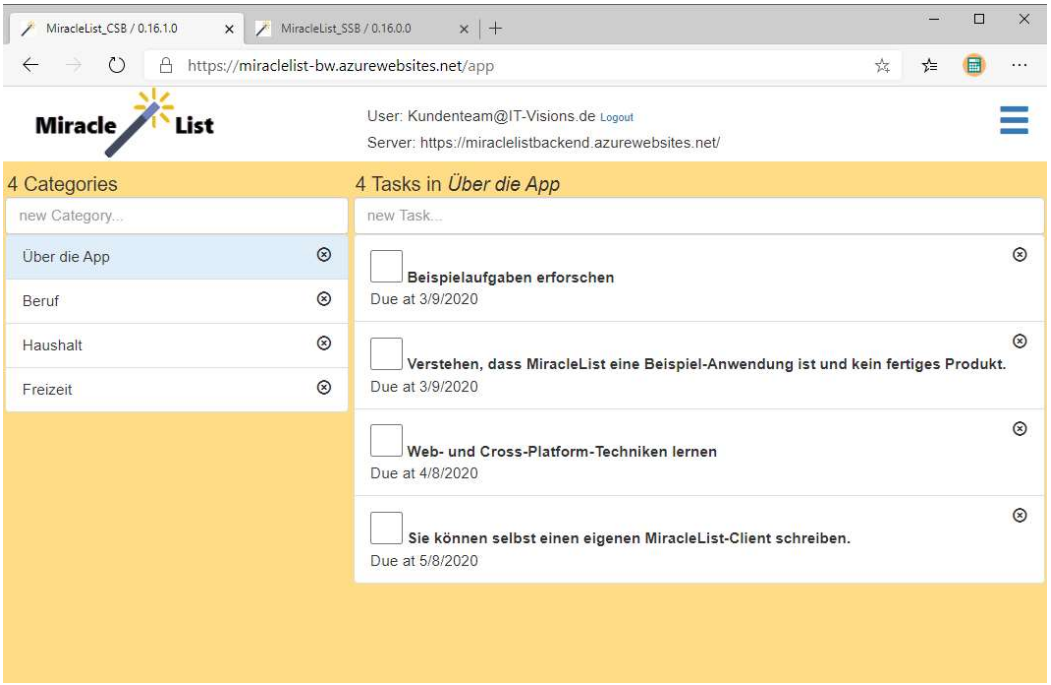


Abbildung: Hauptansicht der MiracleList-Webanwendung

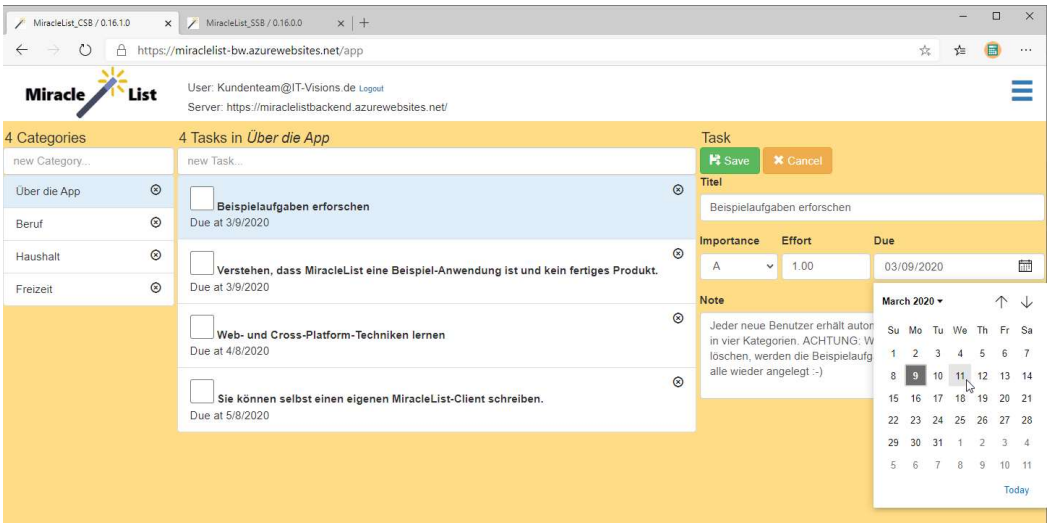


Abbildung: Bearbeitungsansicht der MiracleList-Webanwendung

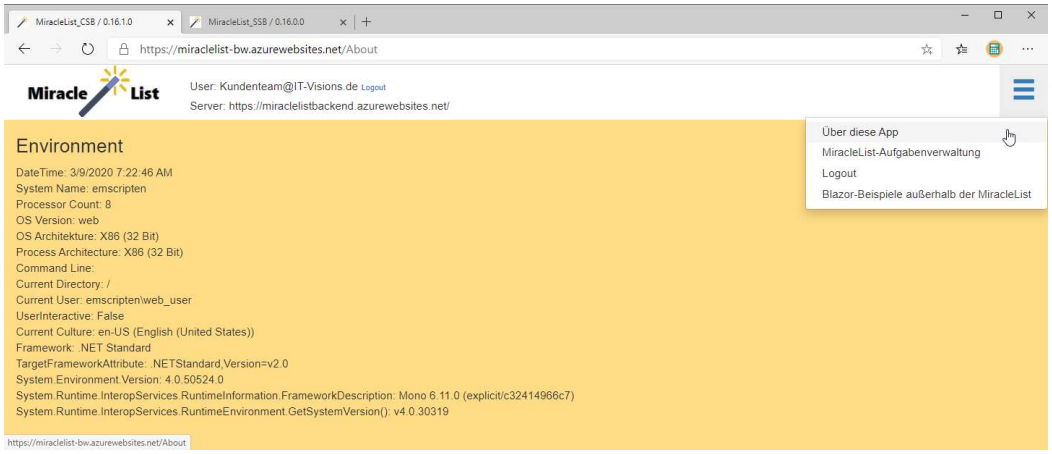


Abbildung: Technische Daten sieht man im Menü "Über diese App"

5.5 Beispiele außerhalb des MiracleList-Szenarios

Viele Beispiele in diesem Buch stammen direkt aus dem Anwendungsszenario der Aufgabenverwaltung "MiracleList". Allerdings stammen nicht alle Beispiele dieses Buchs aus dem Szenario der Aufgabenverwaltung selbst. Einige Beispiele greifen nicht auf die Geschäftslogik und die Datenbank von MiracleList zurück. Dies hat zwei Gründe:

- Einige Beispiele passen inhaltlich nicht das Szenario "MiracleList".
- Aus didaktischen Gründen ist es an einigen Stellen geboten, einfachere Beispiele zu machen.

Alle Beispiele, die nicht Teil der Aufgabenverwaltung sind, finden Sie dennoch in den beiden Blazor-Projekten der MiracleList. Sowohl in der Blazor WebAssembly- als auch der Blazor Server-Variante rufen Sie diese unter der relativen URL /Demos oder den Menüpunkt "Blazor-Beispiele außerhalb der MiracleList" auf.

Abbildung: Die Blazor-Beispiele außerhalb der MiracleList sind in das Menü der MiracleList integriert.

Hinweis: Den Quellcode der meisten Beispiele in diesem Buch, deren URL mit "/Demos" beginnt, finden Sie in dem Projekt /src/MLBlazorRCL. Nur einige wenige Beispiele, die spezifisch sind für eine bestimmte Blazor-Variante, finden Sie in den spezifischen Projekten, also /src/MiracleList_CSB/Demos und /src/MiracleList_SSB/Demos.

5.6 Visual Studio 2019

Für die Entwicklung von Blazor-Anwendung sollten Sie die Microsoft-Entwicklungsumgebung Visual Studio 2019 verwenden in der jeweils aktuellen Update-Version.

Bezugsquelle: *https://visualstudio.microsoft.com/de/vs*

Es gibt Visual Studio 2019 in drei Varianten: Community, Professional und Enterprise. Für die Entwicklung von Blazor-Anwendung ist (für Einzelpersonen und Organisation bis fünf Entwickler) kostenfreie Community-Variante ausreichend. Einige fortgeschrittene Werkzeugfunktionen (z.B. einige Analyse- und Testfunktionen wie IntelliTrace, Live Unit Testing und Code Coverage) erhalten Sie nur mit der Enterprise-Version. Einen Vergleich der Fähigkeiten der Versionen finden Sie unter:

https://visualstudio.microsoft.com/de/vs/compare

Unterstützte Features	Visual Studio Community Kostenloser Download	Visual Studio Professional Kaufen	Visual Studio Enterprise Kaufen
⊕ Unterstützte Nutzungsszenarien	●●●○	●●●●	●●●●
Unterstützung für Entwicklungsplattformen [2]	●●●●	●●●●	●●●●
⊕ Integrierte Entwicklungsumgebung	●●●○	●●●○	●●●●
⊕ Erweitertes Debuggen und Diagnose	●●○○	●●○○	●●●●
⊖ Testtools	●○○○	●○○○	●●●●
Live Unit Testing			●
IntelliTest			●
Microsoft Fakes (Isolation von Komponententests)			●
Code Coverage			●
Komponententests	●	●	●

Abbildung: Ausschnitt aus dem Vergleich der Visual Studio-Varianten

Die aktuellen Preise finden Sie unter *https://visualstudio.microsoft.com/de/vs/pricing*.

Tipp: Kostenfrei können Sie die jeweils aktuelle Preview-Version verwenden, die es hier gibt: *https://visualstudio.microsoft.com/de/vs/preview*. Diese ist aber jeweils in einigen Punkten noch nicht ausgereift und der Einsatz für die professionelle Softwareentwicklung daher nicht empfohlen.

Man kann eine stabile Version und eine Preview-Version parallel auf einem Windows-System installieren.

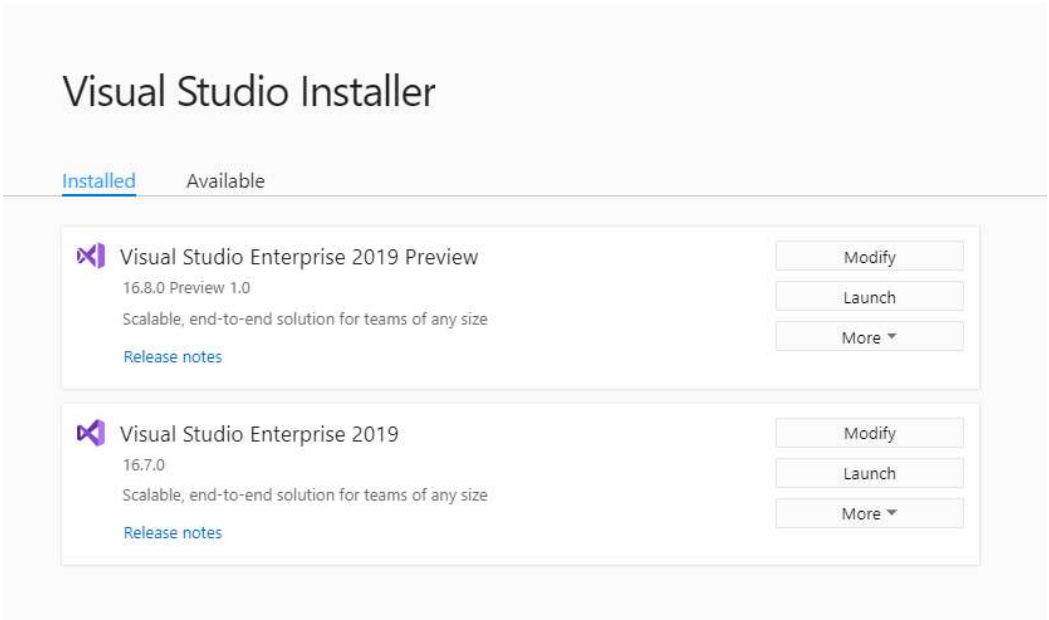


Abbildung: Parallelinstallation der stabilen Version 16.7 und der Preview 1 von 16.8

Hinweis: Visual Studio 2019 entspricht der Version 16. Seit einigen Jahren aktualisiert Microsoft die Entwicklungsumgebung im Abstand von wenigen Wochen und vergibt dafür neue Nummern an der zweiten Stelle: 16.1, 16.2, 16.3 usw.

Für die Softwareentwicklung mit Blazor müssen Sie den Workload "ASP.NET and web development" installieren.

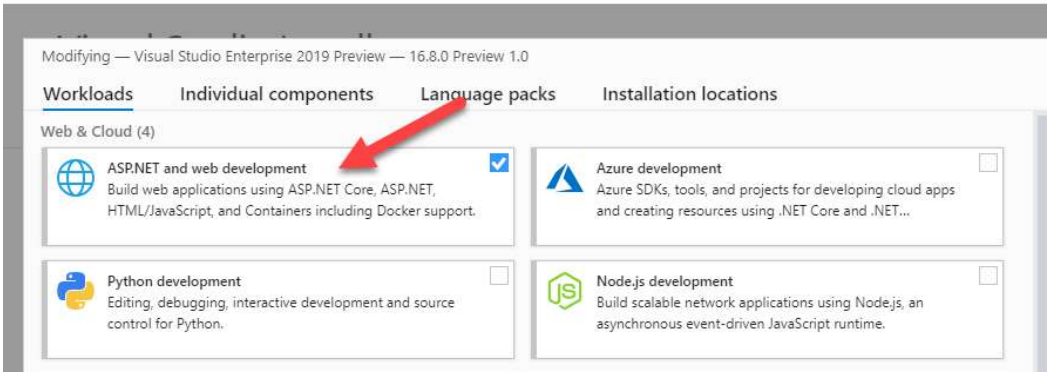


Abbildung: Notwendige Workloads für ASP.NET Core Blazor

Praxistipp: Verwenden Sie die englische Version von Visual Studio, nicht die deutsche Übersetzung. Diese hat einige Übersetzungsschwächen. Zudem ist das "googeln" nach Hilfe im Internet eingeschränkt, wenn Sie nach deutschen Begriffen suchen, weil sie die genaue englische Bezeichnung nicht kennen.

5.7 .NET SDK

Ein typisches Problem, dass Sie nicht nur mit den Programmcodebeispielen in diesem Buch, sondern auch mit Ihren Projekten in der Praxis haben können, ist das Fehlen der notwendigen Installation des .NET (Core) Software Development Kits auf Ihrem Entwicklungssystem. Dann kann Visual Studio die Projekte nicht laden, siehe folgende Abbildung.

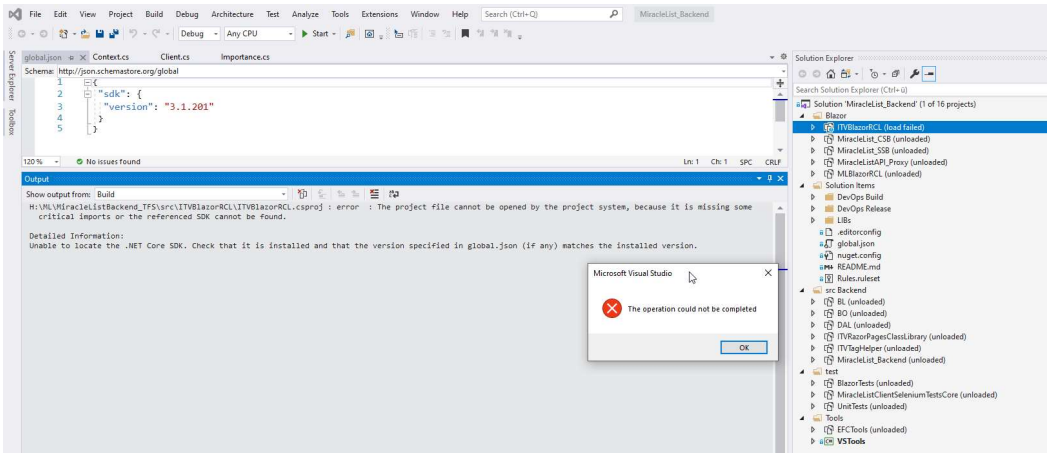


Abbildung: Die notwendige SDK-Version fehlt

Es gibt zwei grundsätzlich zwei Verfahren, wie die .NET SDK-Version gewählt wird:

- Im Standard wird die aktuellste .NET SDK-Version verwenden (einschließlich Preview-Versionen). Dies kann zu unerwünschten Effekten beim Kompilieren führen, z.B. aufgrund Veränderungen (Breaking Changes) in neueren SDK-Versionen.
- Wenn es eine Datei global.json im Projektunterordner gibt, wird die dort festgelegte SDK-Version für diesen Ordner und alle Unterordner verwendet.

Hinweis: Der Einsatz einer global.json-Datei ist empfohlen. Sie erhalten z.B. in Azure DevOps folgenden Hinweis: "Unless you have locked down a SDK version for your project(s), a newer SDK might be picked up which might have breaking behavior as compared to previous versions."

Wenn Visual Studio die in der global.json festgelegte .NET SDK-Version nicht finden kann, haben Sie zwei Optionen:

- Installieren Sie die in der global.json-Datei verlangte .NET SDK-Version. Sie bekommen diese kostenfrei unter der Webadresse *https://dotnet.microsoft.com/download*.
- Sie ändern die SDK-Version in der global.json auf eins der bei Ihnen installierten .NET SDKs. Welche SDKs Sie installiert haben, erfahren Sie mit dem Kommandozeilenbefehl *dotnet --list-sdks*.

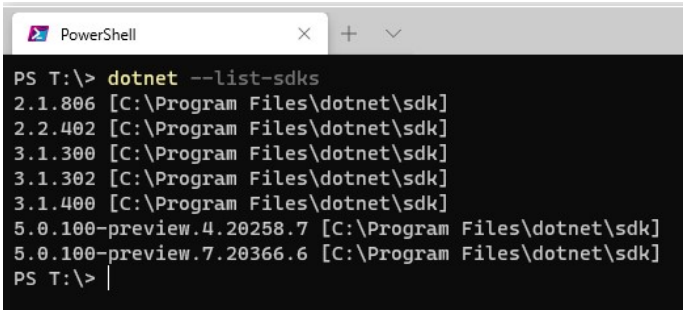


Abbildung: Liste der auf diesem System installierten .NET SDKs

Hinweise: Wenn die Eingabeaufforderung den Befehl "dotnet" nicht finden kann, haben Sie gar kein .NET SDK installiert.

Nicht jede .NET SDK-Version ist mit jeder Version von Visual Studio kompatibel.

5.8 Testen Ihrer PC-Konfiguration

In diesem Kapitel wird ein kurzer Test Ihrer Systemkonfiguration durchgeführt mit dem Ziel, festzustellen, ob Ihr System korrekt funktioniert. Es wird ein Blazor WebAssembly-Projekt erstellt.

Ein Projekt startet man mit Verwendung der Projektvorlage "Blazor App" im Dialog "File/New Project" in Visual Studio. Nach der Auswahl des Namens (geben Sie ein "MiracleListBW" für das Projekt und "MiracleList" für die Projektmappe ein) und des von Ihnen wählbaren Zielpfades (wie immer sollte man einen Dateisystempfad ohne Leerzeichen wählen, das macht die Verwendung von Kommandozeilenwerkzeugen einfacher), kommt die Auswahl zwischen "Blazor Server App" und "Blazor WebAssembly App". Hier ist ".NET Core 3.1" oder ".NET 5.0", "No Authentication", "Configure for HTTPS" und "Progressive Web App" zu wählen (siehe Abbildung).

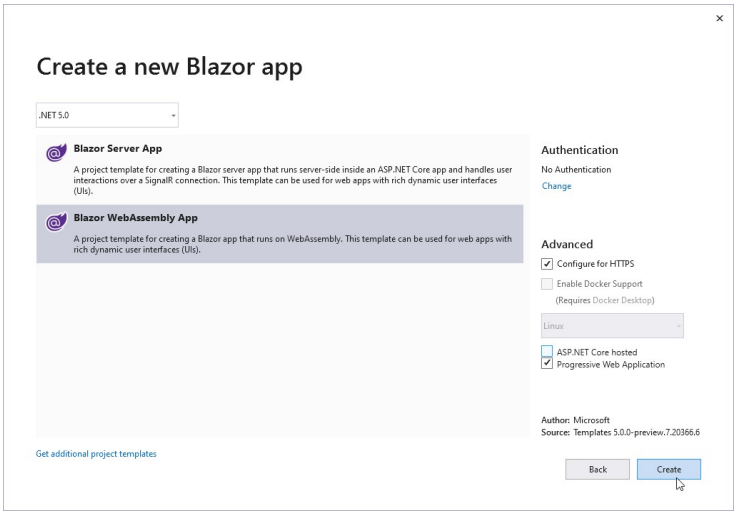


Abbildung: Wichtige Einstellungen für die neue Blazor WebAssembly-Anwendung

Nach dem Anlegen des Projekts sehen Sie in Visual Studio eine Projektmappe mit einem Projekt (siehe rechts in der nächsten Abbildung). Darin gibt es einen Ordner "wwwroot" mit CSS- und Grafik-Dateien (u.a. Twitter Bootstrap und Open Iconic) sowie einer statischen index.html. Die

Razor Components (.razor-Dateien) der einfachen Beispielanwendung von Microsoft finden Sie unter /Pages und /Shared. Der Startcode der Anwendung liegt in Program.cs und App.Razor. _Imports.Razor enthält die Einbindungen von Namensräumen mit @using, die für alle Razor Component-Dateien gelten.

Nun sollten Sie Anwendung übersetzen ("Build/Build Solution") und dann starten, entweder mit "Debug/Start Debugging" (Taste F5) oder "View in Browser" im Kontextmenü des Projekts. Wenn alles auf Ihrem PC korrekt arbeitet, sehen Sie die Webanwendung mit drei Menüpunkten (links in der Abbildung).

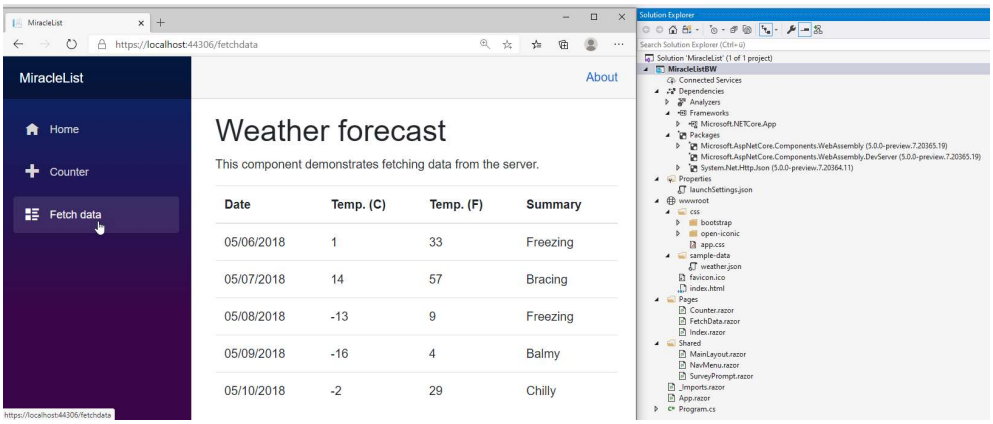


Abbildung: Beispielanwendung von Microsoft

Hinweis: Es gibt leider mannigfaltige Gründe, warum auf Ihrem System es zu einem Fehler kommt. Microsoft hat leider in jeder Version von Visual Studio immer neue Fehler eingebaut. Falls Sie einen Fehler bekommen, recherchieren Sie im Internet oder wenden Sie sich an den Support von Microsoft. Falls Sie Hilfe durch mich als Autor wünschen, nutzen Sie bitte den Support von www.IT-Visions.de [*www.IT-Visions.de/Support*].

5.9 Util.Log()

In den Beispielen in diesem Buch kommt immer wieder der Aufruf Util.Log() zur Ausgabe in das Konsolenfenster des Webbrowsers vor. Dies ist keine vorgefertigte Blazor-Funktion, sondern eine vom Autor dieses Buchs selbstdefinierte Hilfsroutine, die anders als das eingebaute Console.WriteLine() sowohl in Blazor WebAssembly als auch Blazor Server funktioniert. Die Implementierung finden Sie im Kapitel "Tipps & Tricks".

5.10 Qualitätssicherung der Programmcodebeispiele

Ich versichere Ihnen, dass die Programmcodebeispiele auf zwei meiner Entwicklungssysteme kompilierten und liefen, bevor ich sie per Kopieren & Einfügen in das Manuskript zu diesem Buch übernommen habe und auf der Leser-Website zum Download veröffentlicht habe.

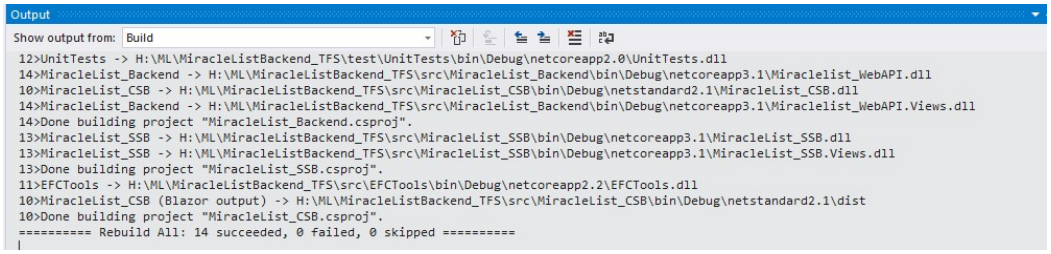


Abbildung: Beweis, dass alle Projekte in Visual Studio fehlerfrei übersetzen

Dennoch gibt es leider Gründe, warum die Beispiele bei Ihnen als Leser dieses Fachbuchs nicht laufen:

- Eine abweichende Systemkonfiguration (in der heutigen komplexen Welt der vielen Varianten und Versionen von Betriebssystemen und Anwendungen nicht unwahrscheinlich). Es ist einem Autor nicht möglich, alle Konfigurationen durchzutesten.
- Änderungen, die sich seit der Erstellung der Beispiele ergeben haben (von den vielen Breaking Changes, die ASP.NET Core immer wieder durch Microsoft erhält, können auch Beispiele betroffen sein, was nicht immer leicht zu entdecken ist)
- Schließlich sind auch menschliche Fehler des Autors möglich. Bitte bedenken Sie, dass das Fachbuchschreiben – wie im Vorwort erwähnt – nur ein Hobby ist. Es gibt nur sehr wenige Menschen in Deutschland, die hauptberuflich als Fachbuchautor arbeiten und so professionell Programmcodebeispiele erstellen und testen können wie kommerziellen (bezahlten) Programmcode.

Wenn Beispiele bei Ihnen nicht laufen, kontaktieren Sie mich bitte mit einer sehr genauen Fehlerbeschreibung (Kontaktdaten siehe Vorwort). Ich bemühe mich, Ihnen binnen zwei Wochen zu antworten. Im Einzelfall kann es wegen dienstlicher oder privater Abwesenheit aber auch länger dauern.

6 Was ist ASP.NET Core Blazor?

ASP.NET Core Blazor (kurz: Blazor) ist ein Webentwicklungsframework von Microsoft zur Entwicklung von Single-Page-Web-Applications (SPAs).

6.1 Blazor-Arten

Es gibt derzeit zwei Arten von Blazor mit verschiedenen Softwarearchitekturmodellen:

- **Blazor WebAssembly** (alias: Client-Side Blazor): Hier läuft der .NET-Programmcode und das HTML-Rendering im Webbrowser in dessen WebAssembly-Laufzeitumgebung. Eine von Microsoft entwickelte JavaScript-Bibliothek kommt hier dennoch auch zum Einsatz, um zwischen der WebAssembly-VM des Browsers und dem Document Object Model des Webbrowsers (DOM) zu synchronisieren, denn die WebAssembly-VM des Browsers hat laut W3C-Standard keinen Zugriff auf dessen DOM. Konkret bedeutet dies: Jedes Ereignis im Browser sendet die JavaScript-Bibliothek an die Blazor WebAssembly-Anwendung. Diese kann auf Ereignisse mit der Ausführung von C#-Code oder sogenannten Razor-Templates (mit Platzhaltern in der Syntax @xy) reagieren. C#-Code und Templates manipulieren aber mangels Zugang nicht das echte DOM, sondern eine Kopie davon innerhalb der WebAssembly-VM, die "Virtual DOM" genannt wird. Der vordefinierte Code von Microsoft synchronisiert die Änderungen auf das echte DOM und ändert so die sichtbare Oberfläche im Webbrowser.
- **Blazor Server** (alias: Server-Side Blazor): Hier läuft der .NET-Programmcode auf dem Webserver. Eine JavaScript-Bibliothek agiert im Browser als Gegenpart. Auch hier gibt es ein Virtual DOM, das aber auf dem Webserver liegt. Die Synchronisierung zwischen DOM im Webbrowser und Virtual DOM auf dem Webserver erfolgt hier in einzelnen Datenpaketen mit ASP.NET Core SignalR via Netzwerk über eine kontinuierliche Websocket-Verbindung. Dafür kommt ein kompaktes Synchronisierungsverfahren zum Einsatz. Wie bei Blazor WebAssembly aktualisiert auch bei Blazor Server die JavaScript-Bibliothek die Oberfläche und sendet Ereignisse zurück an Blazor (in diesem Fall über das Netzwerk).

Hinweis: Manche Entwickler denken, "Blazor Server" wäre der Server zu einer Blazor WebAssembly-Anwendung. Dies ist falsch. Blazor Server ist ein eigenes Softwarearchitekturmodell für die Entwicklung von Single-Page-Web-Applications.

Den wesentlichen Unterschied zwischen beiden Architekturmodellen veranschaulicht die nachstehende Abbildung von Microsoft: Während bei Blazor Server der .NET-Programmcode auf dem Server läuft und eine kontinuierliche Netzwerkübertragung aller Ereignisse und DOM-Änderungen zwischen Webserver und Webbrowser stattfindet, läuft bei Blazor WebAssembly der .NET-Code im Webbrowser. Details zu den Unterschiedenen beider Modelle werden Sie in weiteren Unterkapiteln erfahren.

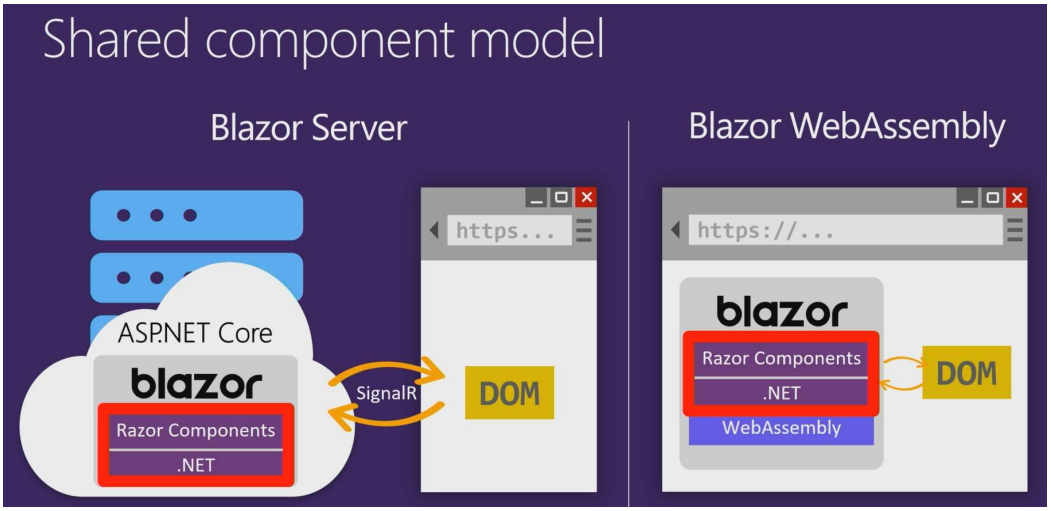


Abbildung: Blazor Server vs. Blazor WebAssembly. Quelle: Microsoft .NET Conf 14.01.2020 [https://focus.dotnetconf.net]

Blazor Server und Blazor WebAssembly teilen sehr viele Gemeinsamkeiten, insbesondere das gleiche Komponentenmodell (Razor Components), sodass die Benutzerschnittbeschreibung und Benutzerschnittstellensteuerungen gleich und damit zwischen beiden Architekturmodellen austauschbar sind. Unterschiede gibt es beim Startcode und der Authentifizierung.

Endbenutzer sehen auf den ersten Blick keinen Unterschied zwischen einer Blazor Server- und einer Blazor WebAssembly-Anwendung, d.h. auch eine Blazor Server-Anwendung wirkt NICHT wie eine klassische Multi-Page-Application (MPA), sondern wie einer moderne Single-Page-Application.

Blazor Server und Blazor WebAssembly sind zu verschiedenen Zeitpunkten erschienen:

- Blazor Server mit Rendering der HTML-Ausgabe auf dem Web. Diese Blazor-Art ist am 23.9.2019 erstmals erschienen in ASP.NET Core 3.0. Eine Aktualisierung mit Fehlerbehebungen gab es in Version 3.1 am 3.12.2019. Seitdem gab es viele kleinere Updates
- Blazor WebAssembly ist am 19.5.2020 unter der Versionsnummer 3.2 erstmals als einsatzreife Version erscheinen.

In Zukunft werden Sie zusammen als Teil von .NET 5.0 (10. November 2020), .NET 6.0 (November 2021), .NET 7.0 (November 2022) usw. erscheinen.

Für die Zukunft plant Microsoft weitere Varianten von Blazor:

- Blazor Hybrid: Hybride Cross-Platform-Anwendungen auf Basis von Electron [*https://aka.ms/blazorelectron*] oder WebWindow [*https://aka.ms/webwindow*]. Während bei Blazor Electron der Hauptprozess auf node.js basiert und das Rendering in Google Chromium stattfindet, nutzt WebWindow im Hauptprozess .NET und greift zum Rendern auf die auf den jeweiligen Betriebssystemen vorhandenen Browserfunktionen (Edge unter Windows 10, WKWebView unter macOS und WebKitGTK+2 unter Linux) zurück.
- Blazor Native: Dies ist die weitreichendste und fernste Idee, bei der Blazor auch verwendet werden soll, um nicht-HTML-basierte Oberflächen zur rendern. Ein erstes Projekt in diese Richtung ist Mobile Blazor Bindings zur Entwicklung von iOS- und Android-Apps [*https://github.com/xamarin/MobileBlazorBindings*], das auf Xamarin Forms basiert.

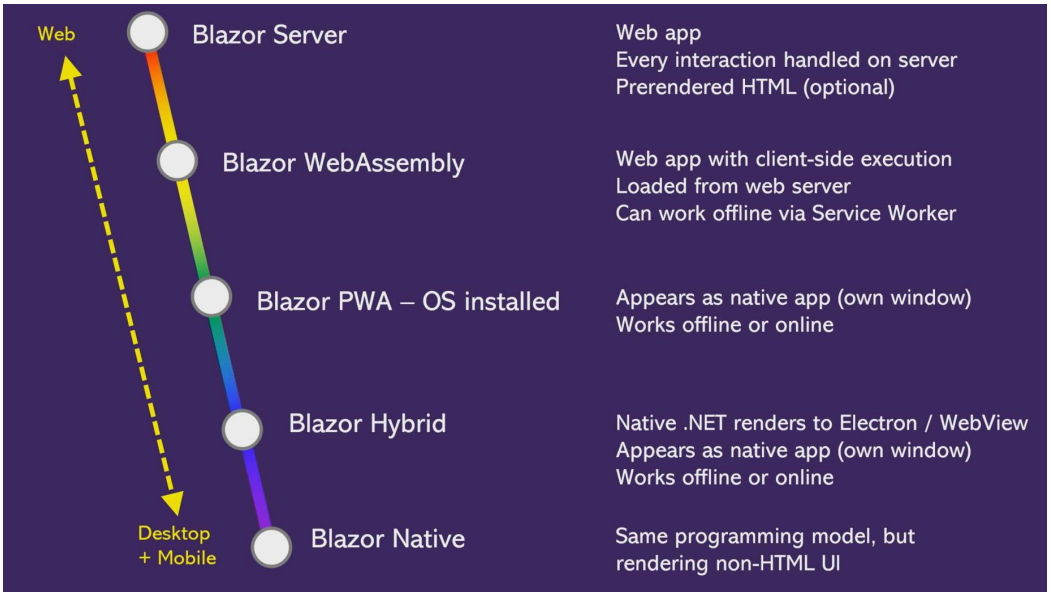


Abbildung: Pläne für Blazor. Quelle: Microsoft .NET Conf 14.01.2010 [https://focus.dotnetconf.net]

6.2 Geschichte von Blazor

Microsoft entwickelte in den Jahren 2007 bis 2012 sehr aktiv das Browser-Plug-In Silverlight, das sehr beliebt war für die Entwicklung interaktiver Anwendungen im Webbrowser. Dann aber stellte Microsoft die Weiterentwicklung ein (u.a. weil die Firma Apple keine Browser-Plug-Ins auf Mobilgeräten zulassen wollte) und setzte auf JavaScript-basierte Web-Frameworks von Drittanbietern. Von Microsoft selbst gab es nur zaghafte Versuche in dem Markt der JavaScript-basierten Web-Frameworks. Die für die Windows-App-Programmierung erschienene WinJS-Bibliothek von Microsoft [*https://github.com/winjs/winjs*] ist inzwischen auch eingefroren.

Bei der Erstankündigung von Blazor im Februar 2018 [*https://devblogs.microsoft.com/aspnet/blazor-experimental-project*] hatte Microsoft zunächst nur ein "Blazor" auf Basis von WebAssembly im Webbrowser vorgestellt. Hier laufen C#-Code und .NET-Assemblies direkt innerhalb des Browsers. Man denkt sofort an Silverlight, aber Blazor ist besser als Silverlight, denn es erfordert keine Browser-Plug-Ins, sondern läuft in allen modernen Webbrowsern, die alle WebAssembly unterstützen.

Am 10. August 2018 in Preview 0.5 folgte dann Server-Side-Blazor, bei der der gleiche Programmcode auch auf dem Webserver laufen konnte. Dieses Feature wurde unter dem Stichwort "Out of Process Execution" entwickelt (vgl. [*https://github.com/aspnet/AspNetCore/issues/15934*]). Danach bezeichnete Microsoft zur Abgrenzung das ursprüngliche WebAssembly-basierte Architekturmodell nun als "Client-Side-Blazor". Seit dem 28. April 2019 war Blazor Server ein offizielles Microsoft-Projekt; zuvor war der Status lediglich "experimentelles Projekt".

In der Folgezeit hat Microsoft nicht nur mit der Technik, sondern auch mit dem Namen der beiden Blazor-Varianten experimentiert (siehe Tabelle). Zunächst sollte die serverseitige Ausführung mit "ASP.NET Core Razor Components" einen komplett anderen Namen erhalten. Im April 2019 revidierte Microsoft dies aber, was auch weise war, denn Razor Components war den vorhandenen

Konzepten wie Razor Views (in ASP.NET Core MVC) und Razor Pages (eine Alternative zu ASP.NET Core MVC) und Razor Class Libraries (zur Kapselung von Razor Views und Razor Pages) namentlich zu ähnlich. Der letzte Namensstand aus dem Juli 2019, der auch für die am 23. September 2019 erschienene RTM-Version der Serverausführung gilt, ist: Die Serverausführung heißt Blazor Server, die Clientausführung Blazor WebAssembly.

	Clientausführung (mit WebAssembly)	**Serverausführung (mit ASP.NET SignalR)**
22.3.2018 (0.1)	Blazor	--
10.8.2018 (0.5)	Client-Side-Blazor	Server-Side-Blazor
Angekündigt Nov 2018, enthalten in ASP.NET Core 3.0 Preview 2 (Jan 2019)	Blazor	ASP.NET Core Razor Components
ASP.NET Core 3.0 Preview 4 (April 2019)	Client-Side-Blazor	Server-Side-Blazor
ASP.NET Core 3.0 Preview 7 (Juli 2019)	Blazor WebAssembly (Blazor WASM)	Blazor Server

Tabelle: Mehrfacher Namenswechsel bei beiden Blazor-Arten

	Blazor Server	Blazor WebAssembly
23.9.2019	Erste stabile Version erschienen in ASP.NET Core 3.0	3.0-Preview9
3.12.2019	ASP.NET Core 3.1 ist erscheinen	3.1-Preview4 Verbleibt im Preview, erste stabile Version für Mai 2020 geplant
17.1.2020	ASP.NET Core 3.1.1 ist erscheinen	
28.1.2020		3.2-Preview 1, basiert auf Mono 6.11 Microsoft hat die Versionszählung umgestellt.
10.3.2020		3.2-Preview 2, basiert auf Mono 6.13
26.3.2020		3.2-Preview 3, basiert auf Mono 6.13

16.4.2020		3.2-Preview 4, basiert auf Mono 6.13
22.4.2020		3.2-Preview 5, basiert auf Mono 6.13
30.4.2020		3.2-RC (3.2.0-rc1.20223.4), basiert auf Mono 6.13
19.5.2020		3.2-RTM basiert auf Mono 6.13

Tabelle: Erschienene (stabile) Blazor-Versionen

6.3 Technischer Support für Blazor

Blazor folgt der Support-Richtlinie von Microsoft für ASP.NET Core, Entity Framework Core und .NET Core.

6.3.1 Support für Blazor Server

Die .NET Core-Supportrichtlinie bedeutet für Blazor Server:

- Blazor in ASP.NET Core 3.1 ist eine Long-Term-Support-Version (LTS), die Microsoft drei Jahre nach dem Erscheinen mit Bug- und Sicherheitsupdates sowie kommerziellem Support unterstützt. Die Version 3.1 ist am 3. Dezember 2019 erschienen, wird also drei Jahre bis zum 3. Dezember 2022 unterstützt.
- Blazor in ASP.NET Core 3.0 war hingegen eine sogenannte "Current"-Version, die nur drei Monate nach dem Erscheinen der LTS-Version noch unterstützt hat. Da am 3.12.2019 eine LTS-Version erschienen ist, wurde Blazor Server in ASP.NET Core 3.0 folglich nur bis zum 3.3.2020 unterstützt.

Praxistipp: Sie sollten neue Projekte nicht mehr mit Version 3.0 starten und bestehende Projekt so bald wie möglich auf Version 3.1 umstellen. Dies gilt nicht nur für Blazor-Projekte, sondern für alle .NET Core- und ASP.NET Core-Projekte.

Die aktuellen Support-Richtlinien für .NET Core, Entity Framework Core und ASP.NET Core inklusive Blazor können Sie unter *https://dotnet.microsoft.com/platform/support/policy/dotnet-core* nachlesen.

Version	Original Release Date	Latest Patch Version	Patch Release Date	Support Level	End of Support
.NET Core 3.1	December 3, 2019	3.1.4	May 12, 2020	LTS	December 3, 2022
.NET Core 3.0	September 23, 2019	3.0.3	February 18, 2020	EOL	March 3, 2020
.NET Core 2.2	December 4, 2018	2.2.8	November 19, 2019	EOL	December 23, 2019
.NET Core 2.1	May 30, 2018	2.1.18	May 12, 2020	LTS	August 21, 2021
.NET Core 2.0	August 14, 2017	2.0.9	July 10, 2018	EOL	October 1, 2018
.NET Core 1.1	November 16, 2016	1.1.13	May 14, 2019	EOL	June 27 2019
.NET Core 1.0	June 27, 2016	1.0.16	May 14, 2019	EOL	June 27 2019

Abbildung: Support-Richtlinie für .NET Core, Entity Framework Core und ASP.NET Core inklusive Blazor (Quelle: [https://dotnet.microsoft.com/platform/support/policy/dotnet-core]).

Tipp: Wenn Sie keinen Support-Vertrag mit Microsoft haben oder einen herstellerunabhängigen Support wünschen, können Sie Unterstützung zu allen Produkten im Bereich .NET und .NET Core auch fallweise unter *www.IT-Visions.de/support* bekommen.

6.3.2 Support für Blazor WebAssembly

Blazor WebAssembly 3.2 hat den Status "Current Version", erhält also nicht den Long-Term-Support der .NET Core-Version 3.1 aus dem Dezember 2019. Dies bedeutet, dass Anwender nach dem Erscheinen der nächsten Version am 10. November 2020 nur drei Monate Zeit haben werden, die Blazor WebAssembly-Version zu wechseln.

Die ersten Long-Term-Support-Version von Blazor WebAssembly wird nach aktuellem Stand erst die Version 6.0 in .NET 6.0 im November 2021 sein.

6.4 Blazor Server

Blazor Server ist stark erklärungsbedürftig, denn obwohl es auf dem Webserver läuft (wie der Name besagt), macht es dennoch keine vollständigen Seiten-Rundgänge wie die klassischen Server-Side-Rendering-Frameworks (SSR) alias Multi-Page-Application-Frameworks (MPA) wie z.B. ASP.NET MVC, ASP.NET Razor Pages, JSP oder PHP. Mit Blazor Server entsteht vielmehr eine Single-Page-Web-Application (SPA): Der Benutzer sieht im Browser nur eine URL, es flackert auch nicht zwischendurch durch den Wechsel ganzer HTML-Seiten, denn es ändern sich bei der SPA-Interaktion nur immer einzelne Seitenteile.

Wer nun an das Feature "Server Side (Pre-)Rendering" denkt, das in einigen JavaScript-basierten SPA-Webframeworks gibt (vgl. Angular Universal [*https://angular.io/guide/universal*], React [*https://www.freecodecamp.org/news/server-side-rendering-your-react-app-in-three-simple-steps-7a82b95db82e/*], Vue.js [*https://vuejs.org/v2/guide/ssr.html*], NextJS [*https://nextjs.org/features/server-side-rendering*] und Aurelia [*https://aurelia.io/docs/ssr/introduction*]) erfasst damit die Fähigkeiten von Blazor Server nicht ganz. Blazor Server dient nicht nur dem Vorrendern einiger statischer Seiten auf dem Webserver, um dem Benutzer das erste Seitenerlebnis schneller zu präsentieren und währenddessen im SPA-Framework in den Browser zu laden, sondern mit Blazor Server kann ein Entwickler tatsächlich eine komplette SPA-Webanwendung erschaffen und hat bei Bedarf (mit etwas eingestreutem JavaScript/TypeScript) auch vollen Zugriff auf alle Browser-APIs.

6.4.1 Rendering und Interaktion bei Blazor Server

Es stellt sich die Frage, wie dies möglich ist. Die nächste Abbildung veranschaulicht die Architektur und Funktionsweise von Blazor Server. Auf der Server-Seite läuft bei Blazor Server eine Razor Component, die neben einen Razor-Template-Teil (HTML-Tags mit eingestreutem C#) und/oder noch eine Code(-Behind)-Datei (in C#) umfassen kann. Mit dem Razor Template erzeugt der Softwareentwickler zwar eine komplette Webseite, diese wird aber nur beim ersten Aufruf einmalig als Ganzes zum Browser gesendet. Zusätzlich wird beim ersten Laden eine JavaScript-Bibliothek von Microsoft (blazor.server.js) zum Browser gesendet.

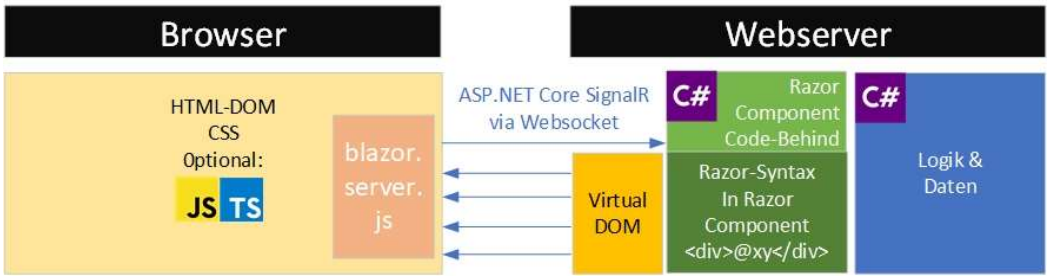


Abbildung: So funktioniert Blazor Server.

Blazor merkt sich auf dem Webserver das initial zum Browser gesendete HTML in Form eines virtuellen Abbilds des Browserinhalts, also des Document Object Models (DOM). Dieses Virtual DOM wird im Folgenden dazu verwendet, jegliche DOM-Änderungen über einen Differenzmechanismus festzustellen und nur noch diese zum Webbrowser zu übertragen. Die Bibliothek blazor.server.js ist clientseitig dafür zuständig, diese Änderungsmitteilungen in das echte DOM des Webbrowsers einzubauen.

Ebenso ist es Aufgabe von blazor.server.js, die Interaktionen des Benutzers mit dem Browser (Tastatureingaben und Mausklicks) zum Webserver zu übertragen und dort für Ereignisse in der Razor Component zu sorgen. Nach Abarbeitung des Ereignisses erfolgt eine Übertragung der resultierenden DOM-Änderungen zum Client.

Wichtig: Blazor synchronisiert das Virtual DOM erst mit dem echten DOM, sobald der Hauptthread nicht mehr blockiert ist!

6.4.2 Netzwerkverkehr bei Blazor Server

Eine Blazor Server-basierte Webanwendung ist nicht offline-fähig und kann auch niemals offline-fähig werden.

Die Übertragungen zwischen Webbrowser und Webserver erfolgt bei Blazor Server in einem von Microsoft definierten Format via ASP.NET Core SignalR über eine Websocket-Verbindung, siehe nächste Abbildung. Microsoft nennt das neue Protokoll BlazorPack (vgl. [*https://github.com/aspnet/AspNetCore/blob/master/src/Components/Server/src/BlazorPack/BlazorPackHubProtocol.cs*]).

Eine Blazor Server-Anwendung ist folglich nicht offline-fähig; wenn der Server nicht mehr erreichbar ist, funktioniert die Anwendung nicht mehr. Die blazor.server.js prüft regelmäßig die Erreichbarkeit des Servers (siehe Nachrichten mit größer "3 B" (Byte) in der folgenden Abbildung).

Kurze Verbindungsabbrüche kann Blazor Server jedoch überbrücken, bei längeren Leistungsproblemen erhält der Benutzer eine Aufforderung, die Seite neu zu laden. Wenn der Server nicht mehr erreichbar ist, wird die aktuelle Ansicht im Browser ausgegraut und der Benutzer sieht oben die Warnmeldung "Attempting to reconnect to the server...". Sollte dies nicht gelingen, erfolgt dann die Meldung "Could not reconnect to the server. Reload the page to restore functionality."

Praxishinweis: Damit der Benutzer ein gutes Erlebnis mit einer Blazor Server-basierten Webanwendung hat, muss laut Microsoft die Netzwerklatenz unter 250 Millisekunden liegen [*https://docs.microsoft.com/de-de/aspnet/core/host-and-deploy/blazor/server?view=aspnetcore-3.0*].

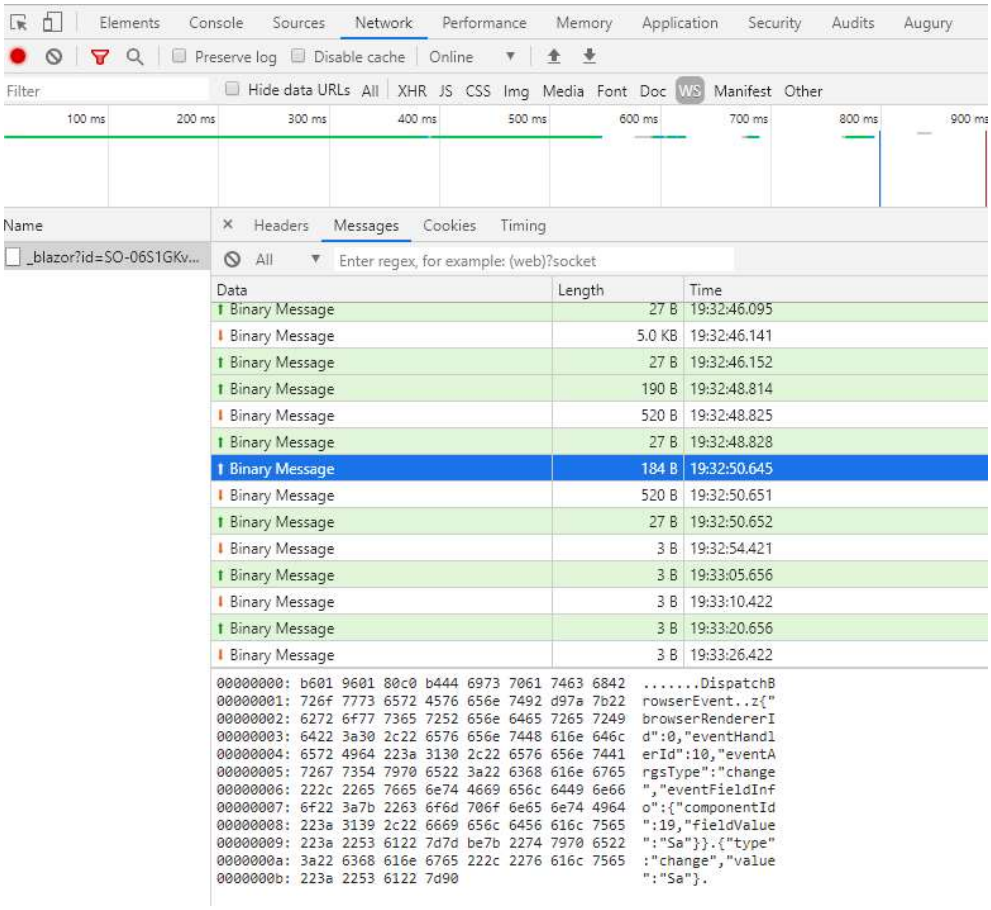


Abbildung: Die Chrome Developer Tools offenbaren die Übertragung der DOM-Änderungen bei Blazor Server via Websockets.

6.4.3 Ressourcenbedarf und Skalierbarkeit

Eine Blazor Server-Anwendung ist aufgrund des serverseitigen Virtual DOM, das es für jeden angeschlossenen Browser gibt, nicht beliebig skalierbar.

Microsoft hat dokumentiert, dass für jeden angeschlossenen Browser schon bei einer Hello World-Anwendung 250 KB RAM im Server gebraucht werden [*https://docs.microsoft.com/de-de/aspnet/core/host-and-deploy/blazor/server?view=aspnetcore-3.0*]. Für echte Anwendungen sei der RAM-Bedarf jeweils selbst zu messen, weil auch Microsoft keine Formel besitzt, um den RAM-Bedarf vorherzusagen. Dass Microsoft dies nicht für eine beliebige Anwendung vorhersagen kann, ist nachvollziehbar, denn der RAM-Bedarf ist von vielen Faktoren abhängig, insbesondere natürlich von der Komplexität der Benutzeroberflächen und der Menge der im Zustand der Komponente gehaltenen Daten.

Der RAM-Bedarf ist aber kein grundsätzliches KO-Kriterium für Blazor Server. Es gibt genug Webwendungen auf dieser Welt, die eine begrenzte Benutzeranzahl haben und für diese so das "SPA light"-Konzept der Blazor Server-Apps geeignet ist. Zudem hat der Softwareentwickler bei Blazor Server mit seiner Programmiertechnik erheblichen Einfluss auf die Höhe des RAM-Bedarfs.

Im Rahmen der .NET Conf am 14.01.2020 hat Microsoft eigene Messergebnisse veröffentlicht. So kann auf einer virtuellen Maschine in der Azure-Cloud mit 4 virtuellen CPUs und 14 GB RAM bei 20.000 gleichzeitigen Benutzern immer noch eine Latenz (Verzögerung bei der Darstellung von Änderungen) von unter 200 Millisekunden erreicht werden.

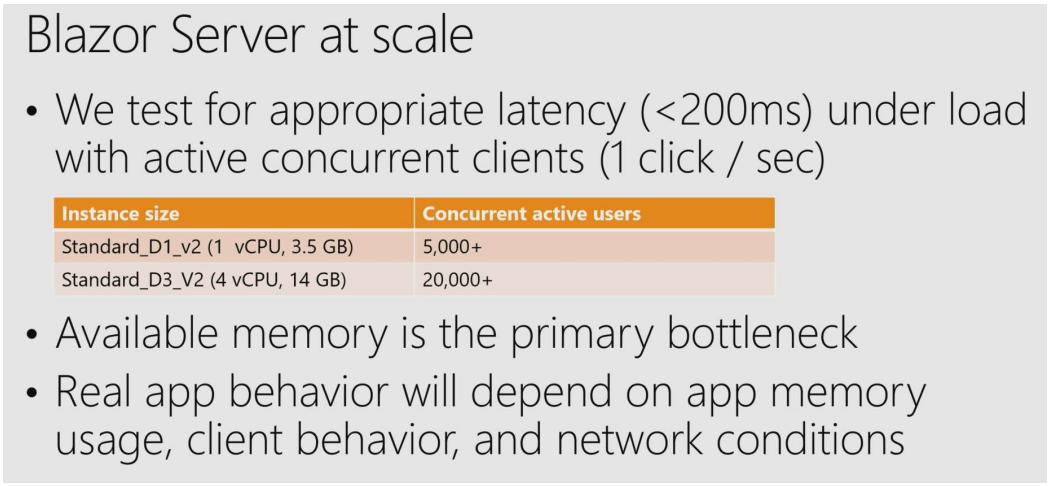


Abbildung: Skalierbarkeit von Blazor Server (Quelle: Microsoft .NET Conf 14.01.2020)

6.4.4 Vor- und Nachteile von Blazor Server

Blazor Server ist eine neue, innovative Architektur für moderne Webanwendungen, die sowohl Vorteile als auch Nachteile mit sich bringt.

Die Vorteile sind:

- Eine Blazor Server-Anwendung hat aus der Sicht des Benutzers das "Look and Feel" einer Single-Page-Web-Application.
- Es gibt in den Razor Components eine Zustandsbehaftung auch ohne Workarounds wie Sessions und Cookies.
- Die Kapselung der Geschäftslogik in eine REST-WebAPI-Schicht ist nicht notwendig (gleichwohl dennoch möglich).
- Es gibt keine Restriktionen durch die Browser-Sandbox, d.h. alle Ressourcen wie Datenbanken und Hardwarekomponenten sind direkt nutzbar.
- Man kann auf dem Server alle Klassen nutzen, die kompatibel zu .NET Core 3.x und .NET Standard bis einschließlich Version 2.1 sind.
- Blazor Server kann alle in .NET Core erreichbaren Programmierschnittstellen verwenden, also sowohl .NET-Assemblies und COM-Komponenten als auch direkt die Windows-Betriebssystem-APIs Windows 32 (Win32) und Windows Runtime (WinRT) nutzen. So aus sind einer Blazor Server-Anwendung heraus zum Beispiel direkte Zugriffe auf Datenbankmanagementsysteme und Verzeichnisdienste sowie an den Server angeschlossene oder im Netzwerk erreichbare Hardware möglich.
- Die Anforderungen an den Browser sind gering: Blazor Server Apps funktionieren auch mit alten Browsern, die zwar JavaScript, aber kein WebAssembly können.

- Die Anwendung startet schnell: nur ca. ~265 KB JavaScript müssen in den Browser geladen werden.
- Später ist eine einfache Migrierbarkeit auf Blazor WebAssembly möglich.

Es gibt aber auch signifikante Nachteile von Blazor Server im Vergleich zu klassischen SPAs:

- Eine Blazor Server-basierte Webanwendung kann nicht offline-fähig werden.
- Wenn die Verbindung (temporär) abreißt, besteht die Gefahr, dass die Anwendung sich davon nicht mehr erholt und neu geladen werden muss.
- Es entsteht wesentlich mehr Datenverkehr zwischen Client und Server.
- Die Anwendung ist bei Netzwerklatenzen größer 250 Millisekunden nicht gut bedienbar.
- Die Verarbeitungsdauer ist bei Änderungen größer, die eigentlich rein im Browser stattfinden könnten (außer man schreibt doch dafür JavaScript oder TypeScript).
- Die Skalierbarkeit ist schlechter, da alle Rechenlast auf dem Server liegt und dieser pro Client ein Virtual DOM und den Komponentenzustand im RAM hält. Jeder angeschlossene Browser erhöht den RAM- und CPU-Bedarf des Webservers.

Praxishinweis: Gerade der letzte Punkt ist bedeutend: Blazor Server ist somit keine Lösung für hoch-skalierbare Webanwendungen mit sehr hohen Benutzerzahlen, sondern für Anwendungen von kleinen bis mittleren Nutzerzahlen. Eine Blazor Server-Anwendung kann auch die Vorstufe zu einer späteren, dann sehr einfachen Umstellung auf Blazor WebAssembly sein.

6.5 Blazor WebAssembly

Blazor WebAssembly ist die Variante von Blazor, die auf Basis von WebAssembly läuft.

6.5.1 Konzept von Blazor WebAssembly

Die Grundmechanismen von Blazor WebAssembly entspricht denen von Blazor Server: Die Benutzeroberfläche wird in Razor Components ereignisgesteuerten gerendert. Die Razor Components manipulieren ein Virtual DOM.

Auch bei Blazor WebAssembly gibt es das Virtual DOM, weil man in der WebAssembly Virtual Maschine keinen Zugriff auf das DOM des Browsers hat. Eine JavaScript-Bibliothek (blazor.WebAssembly.js) vermittelt zwischen dem echten DOM und dem virtuellen DOM. Dies spielt sich jedoch komplett im Webbrowser ab.

Die .NET-DLLs werden bei Blazor WebAssembly in ihrer üblichen Form in Microsoft Intermediate Language (MSIL) in den Webbrowser geladen. Die Mono-basierte Implementierung basiert nicht auf einem Just-In-Timer-Compiler, sondern einem Interpreter, der die Microsoft Intermediate Language (MSIL) als WebAssembly-Code (WASM) ausführt. Der Interpreter von Mono übersetzt MSIL in WASM. Dies bezeichnet man als den Interpreted Mode.

Für die Zukunft ist auch eine Ahead-of-Time-Kompilierung geplant (Ahead-of-Time (AOT) Compiled Mode). Die zugehörige Aufgabe auf Github ist aber seit längerem offen (siehe [*https://github.com/mono/mono/issues/10222*]). Der AOT-Compiler wird nicht in der ersten Version (3.2 im Mai 2020) enthalten sein. Er sollte in .NET 5.0 im November 2020 erscheinen, wurde dann aber erneut vertagt.

Hinweis: Microsoft hat in Blazor WebAssembly 3.2 Preview 1 umbenannt: "mono.wasm" in "dotnet.wasm" und "mono.js" in "dotnet.js". In Blazor WebAssembly 5.0 kommt dann dotnet.5.0.0.js zum Einsatz.

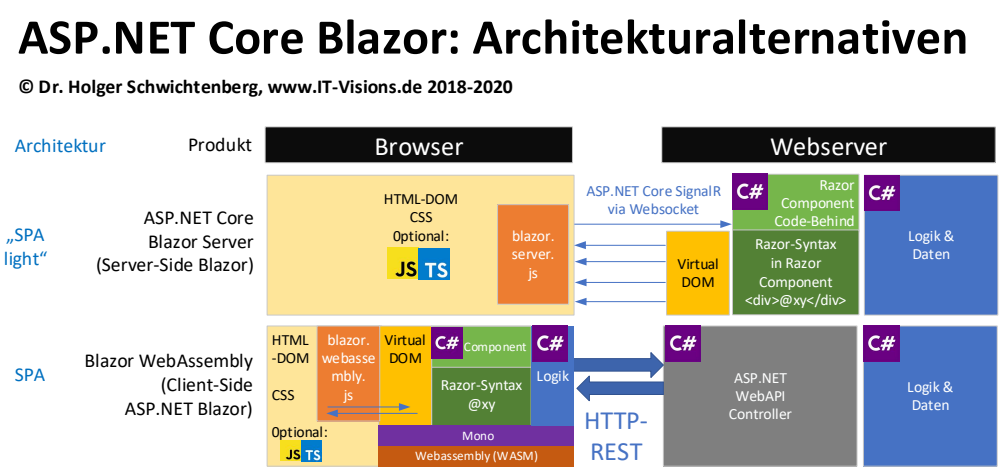


Abbildung: Vergleich Blazor Server mit Blazor WebAssembly aus der Vogelperspektive

6.5.2 WebAssembly (WASM)

Blazor WebAssembly basiert auf dem WebAssembly-Standard. Dieses Unterkapitel liefert einige hilfreiche Erläuterungen zu diesem Standard.

WebAssembly (WASM) ist eine Laufzeitumgebung, implementiert als stack-basierte virtuelle Maschine (VM) zur Ausführung von WebAssembly-Bytecode. Ursprünglich war WebAssembly nur für die Ausführung in Webbrowsern vorgesehen. Mit dem WebAssembly System Interface (WASI) gibt es aber inzwischen auch ein API für die direkte Interaktion mit einem Betriebssystem. Neben dem kompakten Bytecode gibt es auch eine menschlesbare, textliche Form, das WebAssembly Text Format (WAT).

C (input source)

```
int factorial(int n) {
  if (n == 0)
    return 1;
  else
    return n * factorial(n-1);
}
```

text "linear assembly bytecode" (intermediate representation)

```
get_local 0
i64.eqz
if (result i64)
    i64.const 1
else
    get_local 0
    get_local 0
    i64.const 1
    i64.sub
    call 0
    i64.mul
end
```

WASM binary encoding (binary shown below in hexadecimal)

```
20 00
50
04 7E
42 01
05
20 00
20 00
42 01
7D
10 00
7E
0B
```

Abbildung: WebAssembly-Beispiel (Quelle: Wikipedia)

WASM ist eine Low-Level-Bytecode-Sprache (deutlich weniger mächtig als die Microsoft Intermediate Language (MSIL) in .NET), in der es bisher keinen Garbage Collector (GC) gibt. Damit ist WASM stark auf C, C++ und Rust ausgerichtet. Gleichwohl können anderen Programmiersprachen wie C#, Python, Go, Java und PHP nach WASM übersetzt werden, sie

müssen dann aber eine entsprechende Laufzeitumgebung mit höheren Diensten wie dem Garbage Collector selbst mitbringen.

WASM wurde beim World Wide Web Consortiuum (W3C) standardisiert [www.w3.org/TR/wasm-core] und hat seit dem 5. Dezember 2019 den Status "Recommendation" [*https://www.w3.org/2019/12/pressrelease-wasm-rec.html.en*].

Abbildung: Logo des WebAssembly-Standards

WASM ist nicht als Ersatz, sondern als Ergänzung zu JavaScript entwickelt worden, um komplexere Berechnungen, performanter mit einer Low-Level-Sprache ausführen zu können. Zum Start eines WebAssembly-Moduls ist JavaScript im Browser erforderlich.

In WASM ist eine Interoperabilität zu JavaScript vorgesehen, d.h. JavaScript kann WASM-Code aufrufen und umgekehrt, aber WASM hat im Standard keinen Zugriff auf das Document Object Model (DOM) des Browsers. Diese Beschränkung hat Microsoft in Blazor auf Basis der Interoperabilität zwischen WASM und JavaScript überwunden. Dafür gibt es genau wie bei Blazor Server auch in Blazor WebAssembly ein Virtual DOM. Das Virtual DOM läuft jedoch nicht wie bei Blazor Server auf dem Webserver, sondern in der WebAssembly-Umgebung des Webbrowsers. Per JavaScript wird das Virtual DOM mit dem echten DOM synchronisiert.

Die WebAssembly-VM läuft genau wie JavaScript in der Sandbox des Webbrowsers. Daher kann der Entwickler nicht alle Programmierschnittstellen nutzen. So ist ein direkter Datenbankzugriff in Blazor WebAssembly nicht möglich. Der Entwickler kann zwar einen Datenbankprovider wie Microsoft.Data.SqlClient via NuGet.org-Paket in den Browser laden, ein Datenbankverbindungsaufbau scheitert dann aber mit dem Laufzeitfehler "Microsoft.Data.SqlClient is not supported on this platform.". Auch andere Netzwerkprotokolle wie LDAP sind damit unterbunden. Der Entwickler muss alle Daten über HTTP-basierte Webservices beziehen und senden.

Auch Zugriffe auf viele lokale Ressourcen werden abgeblockt, so kann Blazor WebAssembly nicht Informationen über den eigenen Prozess beziehen via System.Diagnostics.Process.GetCurrentProcess(). Dies führt zum Laufzeitfehler " Process has exited or is inaccessible, so the requested information is not available." Unterbunden in WebAssembly ist auch ein voller Dateisystemzugriff sowie der Zugriffe auf beliebige Speicherbereiche. Die Einschränkungen durch die Sandbox sind bei WebAssembly die gleichen wie bei JavaScript.

Die Bytecode Alliance [*https://bytecodealliance.org*] mit Mozilla, Fastly, Intel und RedHat ist angetreten, WebAssembly auch außerhalb des Webbrowsers auf allen Plattformen und Geräten anzubieten. WebAssembly-basierte Anwendungen könnten also eines Tages direkt auf kleinen und großen Geräten laufen.

6.6 Fehlende Funktionen

Leider fehlen noch einige Funktionen in den aktuellen Versionen von Blazor. Die mit Stern (*) markierten Punkte betrifft nur Blazor WebAssembly:

- Debugging aus Visual Studio noch nicht vollständig (*)
  docs.microsoft.com/de-de/aspnet/core/blazor/debug
- Kein direkter Zugriff auf den Inhalt des HTML-Tags <head> → Blazor 5.0
  github.com/dotnet/aspnetcore/issues/10450
- Keine Restriktionen für die Einbettung von Komponenten
  github.com/dotnet/aspnetcore/issues/12302
- Keine Pflichtparameter für Komponenten
  github.com/dotnet/aspnetcore/issues/11815
- Keine Weitergabe von Typparametern an untergeordnete Komponenten
  https://github.com/dotnet/aspnetcore/issues/7268
- Keine integrierte Datei-Upload-Komponente
  https://github.com/dotnet/aspnetcore/issues/12205
- Das Setzen des Fokus auf ein Element erfordert JavaScript → Blazor 5.0
  https://github.com/dotnet/aspnetcore/issues/17472
- AOT-Übersetzung von MSIL zu WASM (*)
  github.com/dotnet/aspnetcore/issues/5466
- Kein Multi-Threading (fehlt in WASM) (*)
  github.com/dotnet/aspnetcore/issues/17730
- Keine Modulbildung mit Lazy Loading → Blazor 5.0
  github.com/dotnet/aspnetcore/issues/5465
- Kein Hot Module Replacement (HMR) / Hot Reload
  github.com/dotnet/aspnetcore/issues/5456
- Keine CSS-Kapselung *github.com/dotnet/aspnetcore/issues/10170* → Blazor 5.0
  (Workaround: *github.com/alexandrereyes/BlazorScopedCss*)
- Keine einfache Unterstützung für Drag&Drop
  https://github.com/dotnet/aspnetcore/issues/18754
- Keine Standard Web Components (aber solche sind nutzbar)
- Begrenzter Hauptspeicher für WebAssembly-VM (*)

Ausblick: Auf GitHub findet man eine Roadmap für geplante Verbesserungen und weitere Funktionen in Blazor [*https://github.com/dotnet/aspnetcore/issues/21514*]. Die kommende Versionsnummer 5.0 wird für Blazor WebAssembly und Blazor Server gelten und im Rahmen von .NET 5.0 am 10. November 2020 erscheinen.

6.7 Ausblick auf Blazor 5.0

Ab .NET 5.0 sollen Blazor WebAssembly und Blazor Server, die bisher getrennt veröffentlicht wurden. jeweils zusammen mit einer Veröffentlichung einer neuen .NET-Version erscheinen und dann beide jeweils die entsprechende .NET-Versionsnummer tragen.

Blazor WebAssembly-basierte Anwendungen melden sich in .NET 5.0 (seit Preview 7) nun nicht mehr mit Mono 6.13.0 (3.2-wasm/1a6e64a9381), sondern mit ".NET 5.0.0-preview.x", wenn man System.Runtime.InteropServices.RuntimeInformation.FrameworkDescription abfragt. Zur Leistungssteigerung von Blazor WebAssembly hat Microsoft das Komponentenrendering, die JSON-Serialisierung und die Interoperabilität mit JavaScript in Preview 7 beschleunigt.

Neue Funktionen gibt es erst seit Preview 8. Dort ist die Unterstützung für die Isolation von CSS-Elementen pro Razor Component (CSS Isolation) sowie die wirklich schmerzlich vermisste Möglichkeit, Anwendungsteile nachzuladen (Lazy Loading), enthalten. Es gibt neue Steuerelemente: <InputRadio> und <InputRadioGroup>. Auch kann man nun den Fokus auf ein HTML-Element über ElementRef.FocusAsync() setzen; bisher brauchte man dazu JavaScript. Mit dem Zusatzpaket Microsoft.AspNetCore.Components.Web.Extensions kann man auch die Inhalte des <head>-Tags (z.B. Title, Link und Meta) beeinflussen.

Die Schnittstelle IAsyncDisposable wird unterstützt. Neu ist auch die Schnittstelle IComponentActivator.

Im Release Candidate soll es dann nun Unterstützung für Pflichtparameter in Komponenten (bisher sind alle optional), Datei-Uploads und Drag & Drop geben. und das Paket Microsoft.AspNetCore.ProtectedBrowserStorage, das es schon seit August 2019 als "experimentelles" Paket gibt, soll offiziell einsatzreif werden (aber nur für Blazor Server).

Hinweis: Für Blazor 5.0 brauchen Sie Visual Studio Version 2019 / 16.8 oder höher.

6.8 Vergleich zwischen Blazor WebAssembly und Blazor Server

Dieses Kapitel gibt einen Überblick über alle wesentlichen Unterschiede zwischen Blazor WebAssembly und Blazor Server. Details werden zum Teil erst in Kapiteln thematisiert. Diese Tabelle soll aber einen zusammenfassenden Überblick der Unterschiede bieten.

6.8.1 Die wichtigsten Unterschiede

Die wesentlichen Unterschiede zwischen Blazor WebAssembly und Blazor Server sind:

- Die erste Version von Blazor Server ist am 23. September 2019 erschienen mit der Versionsnummer 3.0, die zweite am 10. Dezember 2020 mit Versionsnummer 3.1. Die aktuelle Version ist 3.1.6. Die erste Version von Blazor WebAssembly ist unter der Nummer 3.2 am 19.5.2020 erschienen. Aktuell ist die Version 3.2.1. Eine Versionsnummernangleichung erfolgt erst mit .NET 5.0 (geplant für den 10. November 2020). Dann erhalten Blazor Server und Blazor WebAssembly beide die Nummer 5.0.
- Blazor Server 3.1 ist eine "Long-Term-Support"-Version (LTS), die Microsoft drei Jahre nach dem Erscheinen mit Bugfixes und Sicherheitsupdates sowie kommerziellem Support unterstützt. Blazor WebAssembly 3.2 hat allerdings lediglich den Status "Current Version", die schon drei Monate nach dem Erscheinen der nächsten Version aus dem Support läuft. Version 5.0 wird für beide Architekturen eine "Current Version" sein. Langfristigen Support (d.h. drei Jahre) gibt es für Blazor WebAssembly erst in .NET 6.0 im November 2021.
- Bei Blazor Server läuft der C#-Programmcode und das Virtual DOM auf dem Webserver auf Basis der .NET Core-Laufzeitumgebung. Bei Blazor WebAssembly läuft der C#-Programmcode und das Virtual DOM im Webbrowser im Rahmen von dessen Sandbox.

- Blazor Server läuft auf Basis von .NET Core (aktuell 3.1), Blazor WebAssembly auf Basis von Mono (aktuell 6.13). Das liegt einfach daran, dass das Mono-Entwicklungsteam die Idee hatte, wie man .NET mit WebAssembly kompatibel macht. Erst in .NET 5.0 werden beide die gleiche Laufzeitumgebung verwenden.
- Blazor Server funktioniert mit einem Polyfill [ttps://github.com/Daddoon/Blazor.Polyfill] auch im Internet Explorer, Blazor WebAssembly läuft dort gar nicht.
- Während eine Blazor WebAssembly-Anwendung auch offline arbeiten kann und als Progressive Web App (PWA) lokal installierbar ist, kann eine Blazor Server-Anwendung niemals offline-fähig werden. Wenn bei einer Blazor Server-Anwendung der Webserver nicht mehr erreichbar ist, wird die aktuelle Ansicht im Browser ausgegraut und der Benutzer sieht oben die Warnmeldung. Sollte auch nach einigen Sekunden die Verbindung nicht wiederhergestellt werden, wird der Benutzer aufgefordert, die Anwendung neu zu laden. Wenn der Benutzer dann zur alten Position in der Anwendung zurückkehren soll, muss die Anwendung dies über eine eigene Zustandsverwaltung im Webbrowser oder auf dem Webserver regeln.
- Während bei Blazor WebAssembly die Rechenzeit und das RAM des Clients-Systems mit dem Webbrowser verwendet wird, teilen sich bei Blazor Server alle Nutzer die Prozessoren und den Speicher des Webservers. Blazor Server ist folglich schlechter skalierbar.
- Während eine Blazor Server naturgemäß immer auf einem ASP.NET Core-fähigen Webserver laufen muss, kann eine Blazor WebAssembly von einem beliebigen Webserver gestartet werden.
- Bei der Erstellung einer verteilbaren Version (dotnet publish) entsteht bei Blazor Server eine .exe-Datei, bei Blazor WebAssembly eine .dll. Bei Blazor Server ist also ein Self-Hosting außerhalb eines Webservers möglich, wie bei ASP.NET Core üblich, indem man die entstandene .exe-Datei einfach startet. Hier startet "Kestrel", der in ASP.NET Core integrierte Webserver.
- Während bei Blazor WebAssembly der Startcode der Anwendung inklusive der Konfiguration des Dependency Injection-Containers in der Datei Program.cs erfolgt, verwendet Blazor Server den in ASP.NET Core üblichen Host Builder mit einer Klasse Startup.cs.
- Bei Blazor Server sind alle Debugging-Funktionen von Visual Studio uneingeschränkt verfügbar, bei Blazor WebAssembly gibt es noch Restriktionen (z.B. keinen Halt bei unbehandelten Ausnahmen, Inhalte von Objektmengen werden nicht korrekt angezeigt, Debugging nur mit Chrome und Edge Chromium).
- Blazor WebAssembly lädt beim Start eine Vielzahl von Dateien (35 Dateien mit 5.7 MB schon für "Hello World"). Für eine öffentliche Internet-Anwendung ist der Anwendungsstart von Blazor WebAssembly daher zu langsam. Bei Blazor Server sieht der Benutzer hingegen sehr schnell die erste Ansicht.
- Bei Blazor Server findet die in .NET übliche Übersetzung von Intermediate Language (MSIL) in Maschinencode per Just-in-Time-Compiler (JIT) statt. Bei Blazor WebAssembly wird ebenfalls MSIL in den Webbrowser geladen, dort aber interpretiert von einem MSIL-Interpreter innerhalb der dortigen .NET-Laufzeitumgebung, die eine WebAssembly-basierte Variante von Mono ist. Diese Architektur macht Blazor WebAssembly vergleichsweise langsam. Die für Blazor WebAssembly geplante Ahead-of-Time-Kompilierung (AOT) in WebAssembly-Bytecode, die die Ausführung beschleunigen könnte, hat Microsoft leider vertagt (sie wird auch nicht in .NET 5.0 erscheinen).

- Die Interoperabilität zwischen dem C#-Programmcode und JavaScript ist in beiden Blazor-Varianten vorhanden. In Blazor Server ist die JavaScript-Interoperabilität allerdings nicht im Rahmen der sogenannten Pre-Rendering-Phase einer Razor Component möglich, d.h. nicht in den Lebenszyklusereignisbehandlungsroutinen OnInitialized() und OnInitializedAsync(), sondern erst in OnAfterRenderAsync().
- Ein Zugriff auf den Titel des Browserfensters oder andere Daten im <head>-Element der Webseite ist in Blazor WebAssembly nur über JavaScript möglich. Unter Blazor Server kann der Entwickler über die Razor Page _host.cshtml darauf zugreifen, die im gleichen Prozess läuft wie die Razor Component von Blazor Server.
- Unter Blazor WebAssembly erzeugt der Aufruf der Methode Console.WriteLine() eine Ausgabe in der Entwicklerkonsole des Webbrowsers. Unter Blazor Server landet Console.WriteLine() auf dem Webserver; für die Ausgabe in der Entwicklerkonsole des Webbrowsers muss man per JavaScript-Interoperabilität die JavaScript-Funktion console.log() aufrufen.
- Blazor Server kann ohne die Einschränkung einer Sandbox alle in .NET Core erreichbaren Programmierschnittstellen verwenden, also sowohl .NET-Assemblies und COM-Komponenten als auch direkt die Windows-Betriebssystem-APIs Windows 32 (Win32) und Windows Runtime (WinRT) nutzen. Die WebAssembly-VM, auf der Blazor WebAssembly aufsetzt, läuft hingegen genau wie JavaScript in der Sandbox des Webbrowsers. Dies schränkt die Nutzung von APIs und Ressourcen stark ein.
- Blazor Server kann alle .NET Core 3.x-fähigen Assemblies (also alle Assemblies, die .NET Standard bis Version 2.1 unterstützen) laden und nutzen. Blazor WebAssembly 3.2 kann nur .NET Standard 2.1-Assemblies nutzen.
- In Blazor Server ist ein direkter Datenbankzugriff möglich via ADO.NET, Entity Framework Core oder anderen Datenzugriffstechniken. Auch Dateisystem und beliebige andere Ressourcen (z.B. Verzeichnisdienste) sind erreichbar. Bei Blazor WebAssembly ist ein Zugriff auf Datenbanken, Dateisystem und andere Ressourcen nur über einen HTTP-/HTTPS-basierten Webservice (WebAPI/REST-Dienst sowie Google RPC - gRPC) möglich. Lediglich die lokale Browserdatenbank kann in Blazor WebAssembly verwendet werden.

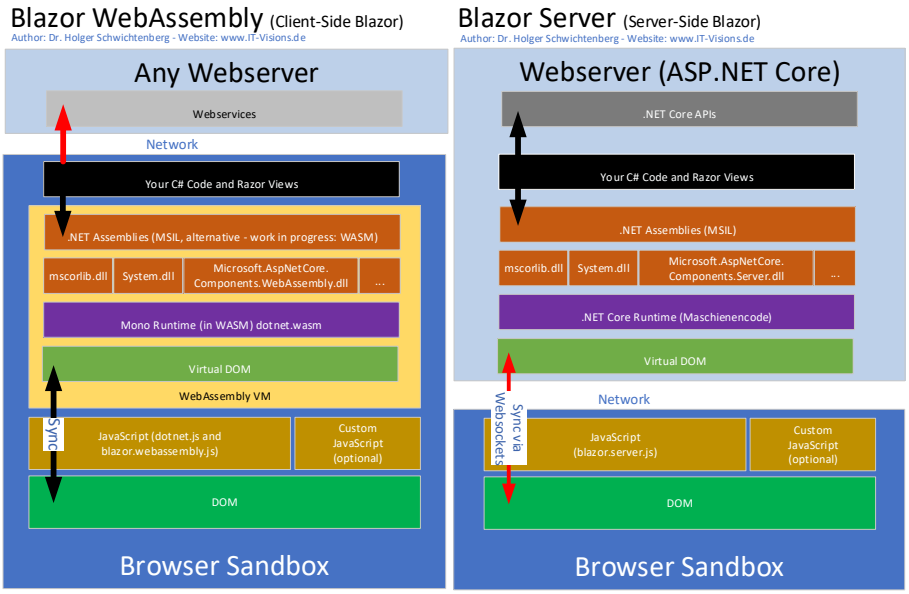


Abbildung: Verwendung von APIs bei Blazor Server und Blazor WebAssembly

Diese Tabelle soll einen zusammenfassenden Überblick der Unterschiede bieten.

	Blazor Server (Server Side Blazor)	**Blazor WebAssembly (Client Side Blazor)**
Anwendungsarchitektur	SPA "Light" aufgrund fehlender Offline-Fähigkeit	SPA
Status	Einsatzreif (RTM) seit Version 3.0 (23.9.2019)	Einsatzreif (RTM) seit Version 3.2 (19.5.2020)
Erste Version	3.0 (23.9.2020)	3.2 (19.5.2020)
Aktuelle Version	3.1.4 (5.0 Preview 7)	3.2 (5.0 Preview 7)
Runtime	.NET Core 3.1.4 (.NET 5.0)	Mono 6.13 (.NET 5.0)
Support	Long-Term-Support (bis 3.12.2022)	Current-Version (also nächste Version 10. November 2020 + 3 Monate)

	Blazor Server (Server Side Blazor)	Blazor WebAssembly (Client Side Blazor)
Target Framework Moniker	netcoreapp3.1 (net5.0)	netstandard2.1 (net5.0)
C# läuft im	Webserver	Webbrowser
Virtual DOM im	Webserver, zum Browser synchronisiert mit SignalR	Webbrowser
Browserunterstützung	Chrome, Firefox, Safari, Edge, Internet Explorer mit Polyfill	Chrome, Firefox, Safari, Edge (kein Internet Explorer!)
Webserver	Webserver mit ASP.NET Core	Webserver mit ASP.NET Core oder beliebig (Statischer Webserver reicht)
Skalierbarkeit	O	++
Startgeschwindigkeit	++	O
Offline-Fähigkeiten	-- (nicht möglich)	++
Grundgerüst der Seite	pages/_host.cshtml (Razor Page)	wwwroot/index.html (statisches HTML-Seite)
Nutzbare APIs	Beliebige .NET Core-APIs, inkl. ADO.NET / EF(C)	Web-APIs, gRPC-Dienste, SignalR
NuGet-Referenzen	Microsoft.NET.Sdk.Web	Microsoft.NET.Sdk.Web, Microsoft.AspNetCore.Components.WebAssembly.*, System.Net.Http.Json
Debugging	Möglich in Razor Template und Code-Behind	Möglich in Razor Template und Code-Behind, aber noch nicht alle Debugging-Funktionen verfügbar (z.B. kein Halt bei Exceptions ☹)

	Blazor Server (Server Side Blazor)	Blazor WebAssembly (Client Side Blazor)
Protokollierung in Browser-Konsole	Dafür muss man in JavaScript console.log("…") aufrufen	Console.WriteLine("…"), allerdings kein automatisches Objekt-Dump wie bei console.log()
Authentifizierung	AuthenticationStateProvider, optional Integration mit ASP.NET Core Identity (Razor Pages)	AuthenticationStateProvider, optional Integration mit ASP.NET Core Identity (Razor Pages) oder OIDC
Title und Metatags setzen	per Razor Syntax	per JS-Interop
Nutzbare APIs	Alle .NET-APIs, COM, Win32, WinRT	Alle von der Browser-Sandbox erlaubten .NET-APIs (z.B. System.Data ist ausgeschlossen)
Unterstützung für Progressive Web Apps (PWA)	Nein	Ja
Kompilat des eigenen Projekts	.exe	.dll

Tabelle: Unterschiede zwischen Blazor Server und Blazor WebAssembly

6.8.2 Browserunterstützung für Blazor

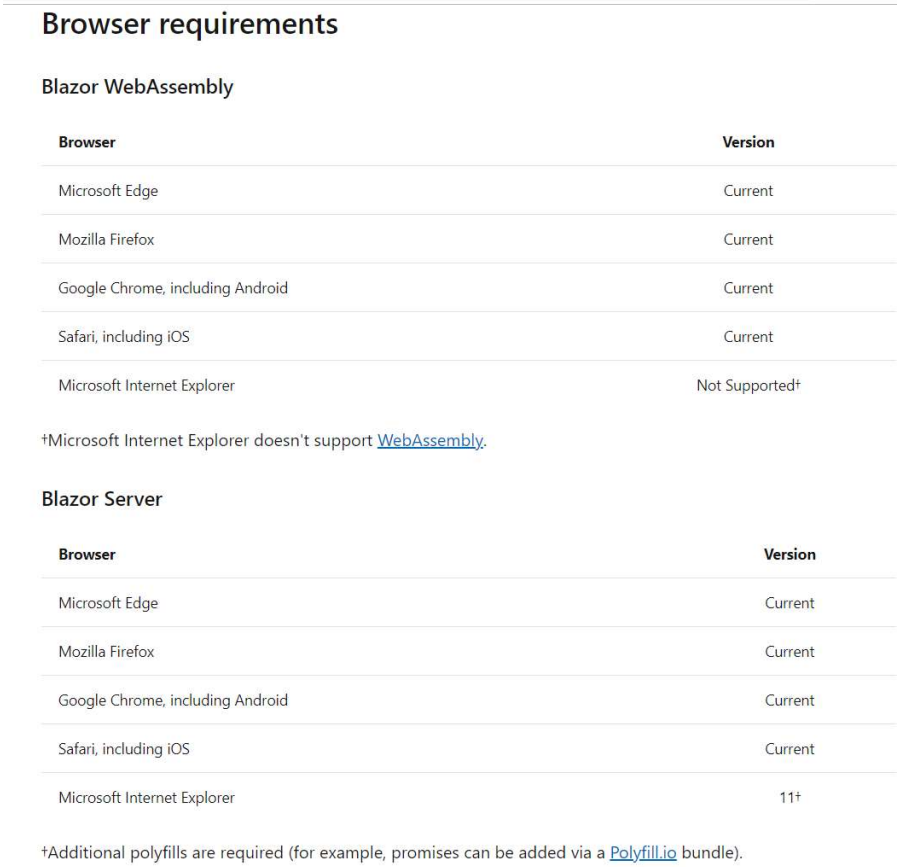
Viele moderne Webbrowser haben WebAssembly schon seit längerem implementiert: Firefox (seit Version 58), Chrome (seit Version 63), Edge (seit Version 16), Opera (seit Version 57) und Safari (seit Version 11.2), vgl. *https://caniuse.com/#feat=wasm*.

Nicht verfügbar ist WASM für den Internet Explorer. Ursprünglich war geplant, im Internet Explorer über asm.js [*http://asmjs.org/*] WebAssembly zu emulieren. Dieses Features wurde jedoch wieder entfernt [*https://github.com/aspnet/Blazor/commit/4006cd543900fcc1cf76cd75a1b24007e60c8a67*].

Auch Blazor Server läuft im Standard nicht im Internet Explorer (siehe folgende Abbildung). Hier kann der Softwareentwickler jedoch mit Hilfe eines Polyfills mit Namen "Blazor.Polyfill" [*https://github.com/Daddoon/Blazor.Polyfill*] die notwendigen Programmierschnittstellen nachrüsten. Der Polyfill besteht aus einer Datei blazor.polyfill.min.js, die man im Verzeichnis /wwwroot des Blazor Server-Projekts ablegt und dann in der Datei Host.cshtml vor dem <script>-Tag für die blazor.server.js einfügt:

```
<script src="~/blazor.polyfill.min.js"></script>
```

```
<script src="_framework/blazor.server.js"></script>
```



Browser requirements

Blazor WebAssembly

Browser	Version
Microsoft Edge	Current
Mozilla Firefox	Current
Google Chrome, including Android	Current
Safari, including iOS	Current
Microsoft Internet Explorer	Not Supported†

†Microsoft Internet Explorer doesn't support WebAssembly.

Blazor Server

Browser	Version
Microsoft Edge	Current
Mozilla Firefox	Current
Google Chrome, including Android	Current
Safari, including iOS	Current
Microsoft Internet Explorer	11†

†Additional polyfills are required (for example, promises can be added via a Polyfill.io bundle).

Abbildung: Unterstützte Webbrowser bei Blazor Server und Blazor WebAssembly

6.8.3 Performance-Vergleich

Bei Blazor Server findet die in .NET übliche Übersetzung von Intermediate Language (MSIL) in Maschinencode per Just-in-Time-Compiler (JIT) statt.

Bei Blazor WebAssembly wird ebenfalls MSIL in den Browser geladen, dort aber interpretiert von einem MSIL-Interpreter innerhalb der dortigen .NET-Laufzeitumgebung, die eine WebAssembly-basierte Variante von Mono ist. Diese Interpreter-Architektur macht schon deutlich, dass die Codeausführung in Blazor WebAssembly deutlich langsamer als bei Blazor Server ist.

Daniel Roth, Program Manager im ASP.NET-Entwicklungsteam bei Microsoft, bestätigt dies in der Antwort auf eine Frage in einem Blogeintrag: "Blazor WebAssembly runs on a .NET IL interpreter based runtime – there's no JIT compilation to WebAssembly happening, so execution performance is much slower than normal .NET code execution." [*https://devblogs.microsoft.com/aspnet/blazor-webassembly-3-2-0-now-available/#comment-2276*]. In seiner Antwort auf die Performance-Frage eines Kunden geht er auch offen damit um, dass Blazor WebAssembly derzeit hinsichtlich der Ausführungsgeschwindigkeit nicht konkurrenzfähig zu aktuellen JavaScript-Frameworks ist: "Blazor WebAssembly isn't going to win in any performance comparisons with JavaScript based frameworks like Angular or React. ".

Hinweis: Schon bei der Erstankündigung von Blazor im März 2018 brachte Microsoft auch eine Ahead-of-Time-Kompilierung (AOT) von MSIL in WebAssembly-Bytecode ins Spiel, die die Ausführung beschleunigen könnte. AOT ist aber nicht in der Version 3.2 enthalten und auch aus der Roadmap für .NET 5.0 gestrichen worden [*https://github.com/dotnet/aspnetcore/issues/21514*]. Microsoft tut sich seit Jahren schwer mit AOT für .NET; man hat schon diverse Ansätze versucht und wieder beerdigt.

Die folgenden Tabellen zeigen zwei Performance-Vergleichsszenarien zwischen Blazor Server und Blazor WebAssembly:

- Szenario 1: Eine reine Berechnung ohne UI-Ausgabe (Berechnung der ersten 45 Fibonacci-Zahlen). Die Zahl in der ersten Spalte der folgenden Tabellen ist eine Anzahl der Berechnung der ersten 45 Fibonacci-Zahlen.
- Szenario 2: Das Rendern einer Liste von -Elementen. Die Zahl in der ersten Spalte der folgenden Tabellen ist die Anzahl der gerenderten -Elemente.

In beiden Szenarien gibt es keinen Datenbank-, WebAPI- oder sonstigen Ressourcenzugriff, der das Ergebnis verfälschen könnte

Die beiden Szenarien sind in den MiracleList-Projekten (\MLBlazorRCL\Performance\Performance.razor) enthalten.

Die Leistungsdaten in der folgenden Abbildung wurden jeweils mit den Webbrowsern Chrome und Firefox erhoben auf drei Servertypen:

a) auf einem lokalen System (Webbrowser und Webserver auf dem gleichen Rechner, ein aktueller Entwickler-PC mit AMD Ryzen 9 3950X, 16 Kerne und 128 GB RAM)

b) in der Microsoft Azure-Cloud als "Web App Service" gehosteten Webanwendungen (Tarif "S1", ca. 65 Euro/Monat)

c) in der Microsoft Azure-Cloud als "Web App Service" gehosteten Webanwendungen (Tarif "P3V2", ca. 500 Euro/Monat)

Die Messdaten führen zu folgenden Erkenntnissen:

- Der Vergleich von Blazor Server und Blazor WebAssembly auf dem lokalen System zeigt, dass Blazor WebAssembly – erwartungsgemäß – grundsätzlich deutlich langsamer ist als Blazor Server.
- Es ist aber zu bedenken, dass sich bei Blazor Server alle Nutzer die Rechenleistung des Servers teilen, während bei Blazor WebAssembly die Leistung auf jedem Client getrennt erbracht wird.
- Sowohl Blazor Server als auch Blazor WebAssembly stürzen bei vielen Rechenoperationen ab. Bei Blazor WebAssembly kommt es zum Laufzeitfehler "System.OutOfMemoryException: Out of memory", weil die Berechnung die Ergebnisse im RAM speichert und der die WebAssembly-VM nutzbare und bisher nicht erweiterbare Speicher begrenzt ist.
- Der Absturz von Blazor Server bei umfangreichen Operationen auf Azure liegt an der begrenzten Rechenleistung. Der Client bekommt keine Reaktion mehr von dem beschäftigten Server und geht davon aus, dass diese nicht mehr antworten wird.
- Bei Blazor Server kann man – wie zu erwarten – durch einen besser ausgestatteten Webserver eine höhere Rechenleistung erreichen.

Praxishinweis: Bei Szenario 2 geht Blazor WebAssembly schon bei 9500 Elementen der Speicherplatz aus. Hier wurde aber absichtlich ungünstig programmiert mit Zeichenkettenverknüpfungen (s += "Hello World"). Wer sich an die Best Practices von .NET hält und dafür die Klasse System.Text.StringBuilder einsetzt, schafft allerdings über 10.000.000 dieser Konkatenationen vor Erreichen der (bisher nicht erweiterbaren) Speichergrenze.

Blazor-Performance-Tests

Fibonacci-Berechnungen mit Speicherung der Ergebnisse im RAM (List<int>)

© Dr. Holger Schwichtenberg, www.IT-Visions.de 2020

Firefox	Lokal	Lokal	
Berechnungen	Blazor Server	Blazor WebAsssembly	Faktor
1.000	2	69	34,5
10.000	25	698	27,9
100.000	260	7.027	27,0
1.000.000	2.611	67.951	26,0
10.000.000	26.666	Out-of-Memory	

Chrome	Lokal	Lokal	
Berechnungen	Blazor Server	Blazor WebAsssembly	Faktor
1.000	2	105	52,5
10.000	26	1.019	39,2
100.000	273	10.035	36,8
1.000.000	2.693	101.350	37,6
10.000.000	26.385	Out-of-Memory	

Firefox	Azure S1	Azure P3V2	
Berechnungen	Blazor Server	Blazor Server	Faktor
1.000	1	1	1,0
10.000	40	18	2,2
100.000	980	186	5,3
1.000.000	10.521	1.861	5,7
10.000.000	Absturz	Absturz	

Chrome	Azure S1	Azure P3V2	
Berechnungen	Blazor Server	Blazor Server	Faktor
1.000	1	1	1,0
10.000	44	16	2,8
100.000	575	235	2,4
1.000.000	10.294	1.828	5,6
10.000.000	Absturz	Absturz	

Abbildung: Leistungsmessdaten für Szenario 1

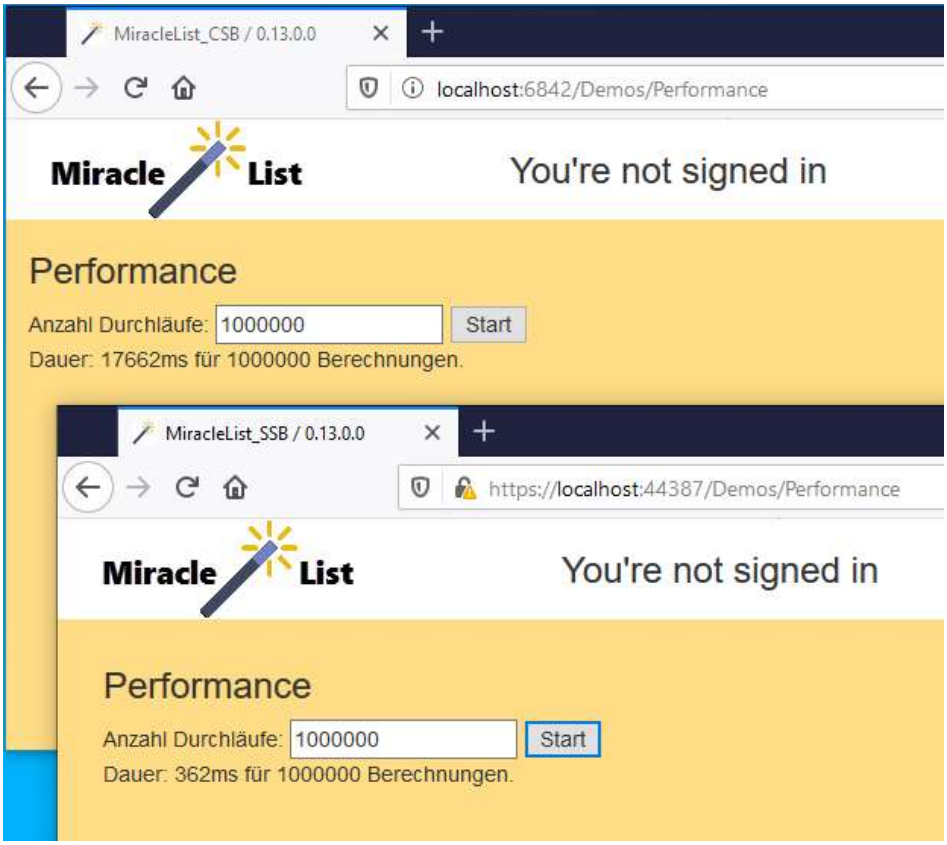


Abbildung: Bildschirmausgabe für Szenario 1

Blazor-Performance-Tests
Rendern einer Liste von -Elementen (mit String-Konkatenation)
© Dr. Holger Schwichtenberg, www.IT-Visions.de 2020

Firefox	Lokal	Lokal
-Rendering	Blazor Server	Blazor WebAsssembly
1.000	6	26
5.000	802	613
7.500	1.439	Out-of-Memory
10.000	1.807	Out-of-Memory
100.000	17.732	Out-of-Memory
1.000.000	154.144	Out-of-Memory

Chrome	Lokal	Lokal
-Rendering	Blazor Server	Blazor WebAsssembly
1.000	23	37
5.000	557	900
7.500	1.439	Out-of-Memory
10.000	2.384	Out-of-Memory
100.000	19.000	Out-of-Memory
1.000.000	210.342	Out-of-Memory

Chrome	Azure S1	Azure P3V2
-Rendering	Blazor Server	Blazor Server
1.000	112	30
5.000	5.806	4.224
7.500	8.210	6.345
10.000	14.320	9.281
100.000	Absturz	Absturz
1.000.000	Absturz	Absturz

Abbildung: Leistungsmessdaten für Szenario 2

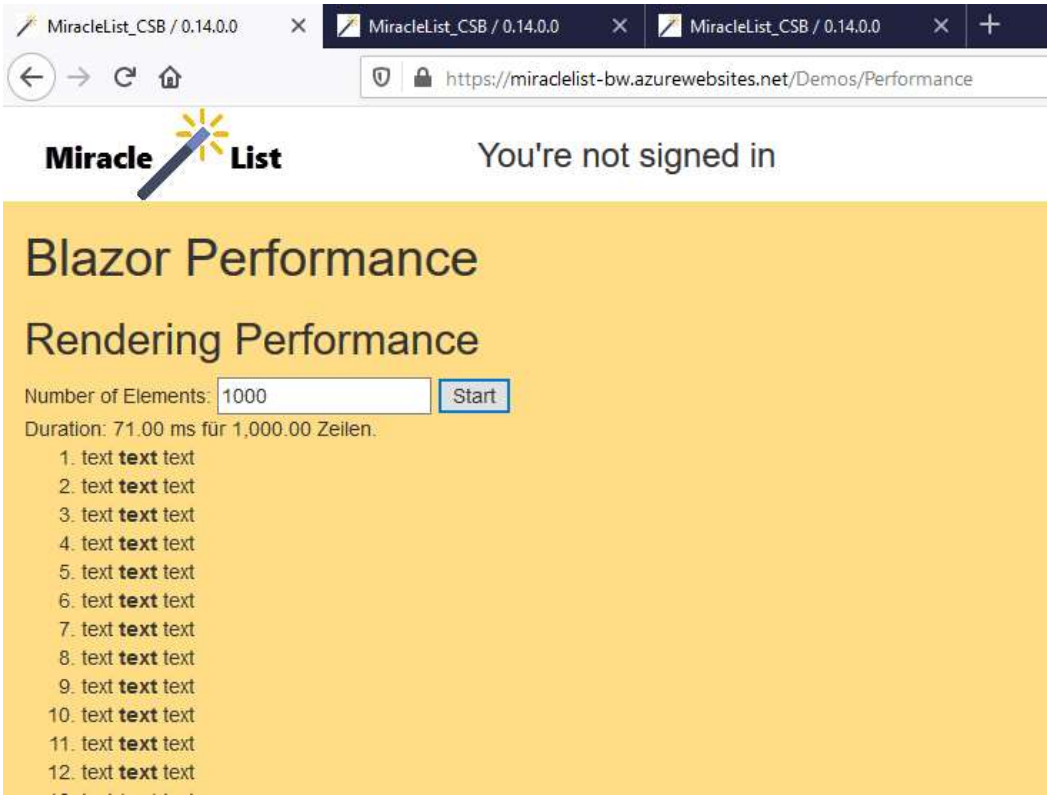


Abbildung: Bildschirmausgabe für Szenario 2

6.8.4 Anwendungsgröße und Startzeiten

Blazor WebAssembly lädt beim Start eine Vielzahl von Dateien, der Anwendungsstart ist entsprechend langsam. Selbst eine minimale Blazor WebAssembly-Anwendung mit einer einzigen statischen Razor Component benötigt beim Start 35 HTTP-Anfragen mit insgesamt 5,5 MB Netzwerkdatenverkehr (siehe Abbildung 3), trotz Einsatz des Mono IL Linkers [*https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/blazor/configure-linker?view=aspnetcore-3.1*] und durch Brotli-Komprimierung [*https://docs.microsoft.com/en-us/aspnet/core/host-and-deploy/blazor/webassembly?view=aspnetcore-3.1#brotli-precompression*]. Wenn man in den Entwicklerwerkzeugen im Chrome-Browser eine "Fast 3G"-Verbindung simulieren lässt, dann sieht der Benutzer rund 32 Sekunden lang "Loading…". Das ist inakzeptabel für Internet-Anwendungen. Das kann man allenfalls für Intranet- und Extranet-Anwendungen vertreten – für öffentlichen Websites ist dies zu lang. Bei aktiviertem Cache dauert das zweite Laden rund 6,5 Sekunden.

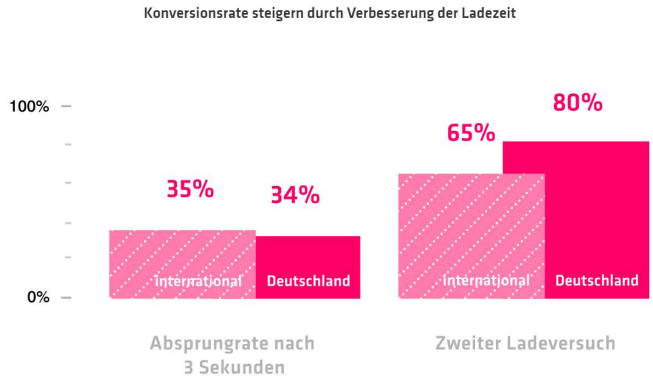


Abbildung: Einfluss von Ladezeiten von mehr als 3 Sekunden (Quelle: [https://www.kostaldesign.com/themen/digitalisierung/webseiten-ladezeit-auswirkung-konversion-ranking/]

Immerhin wächst die Anwendungsgröße nicht linear. Eine Blazor WebAssembly-Anwendung mit 200 Eingabemasken auf denen sich jeweils 25 Steuerelemente befinden umfasst "nur" rund 8,7 MB mehr. Die Anzahl der HTTP-Anfragen ändert sich nicht bei steigender Komponentenanzahl, da alle Komponenten in eine einzige DLL hineinkompiliert werden. In den Zahlen des MiracleList-WebAssembly-Clients sieht man aber mehr HTTP-Anfragen, da hier viele weitere Ressourcen (Style Sheets, Grafiken, JavaScript-Dateien etc) verwendet werden.

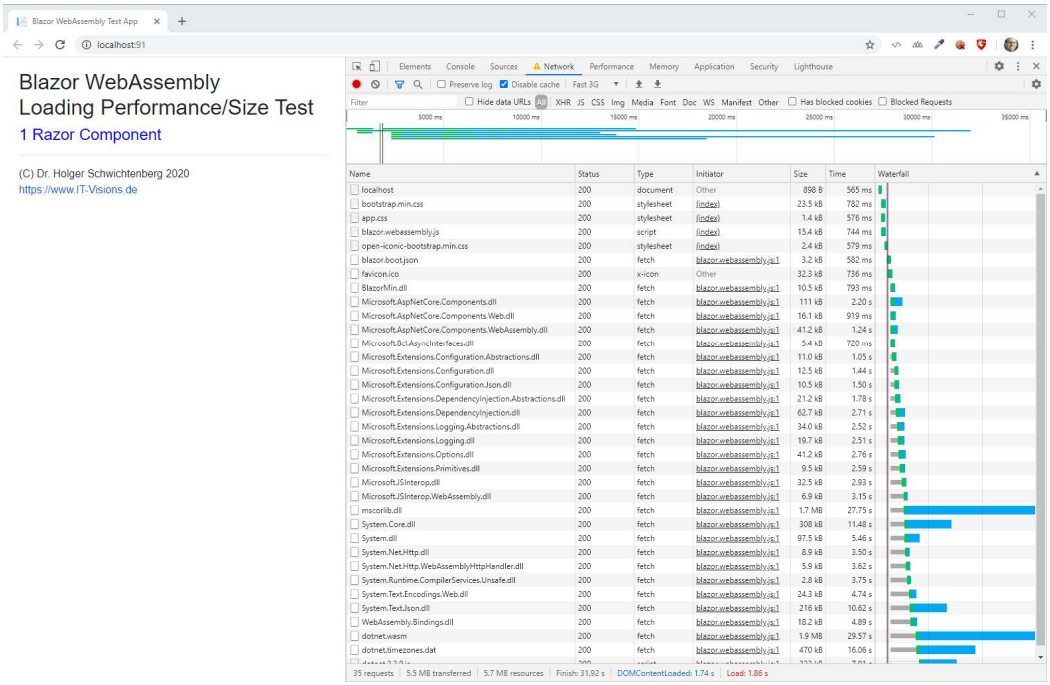


Abbildung: Das "Hello World"-Kompilat umfasst in Blazor WebAssembly schon 18,3 MB, die in 62 HTTP-Anfragen geladen werden (hier: geladen mit "Fast 3G"-Simulation)

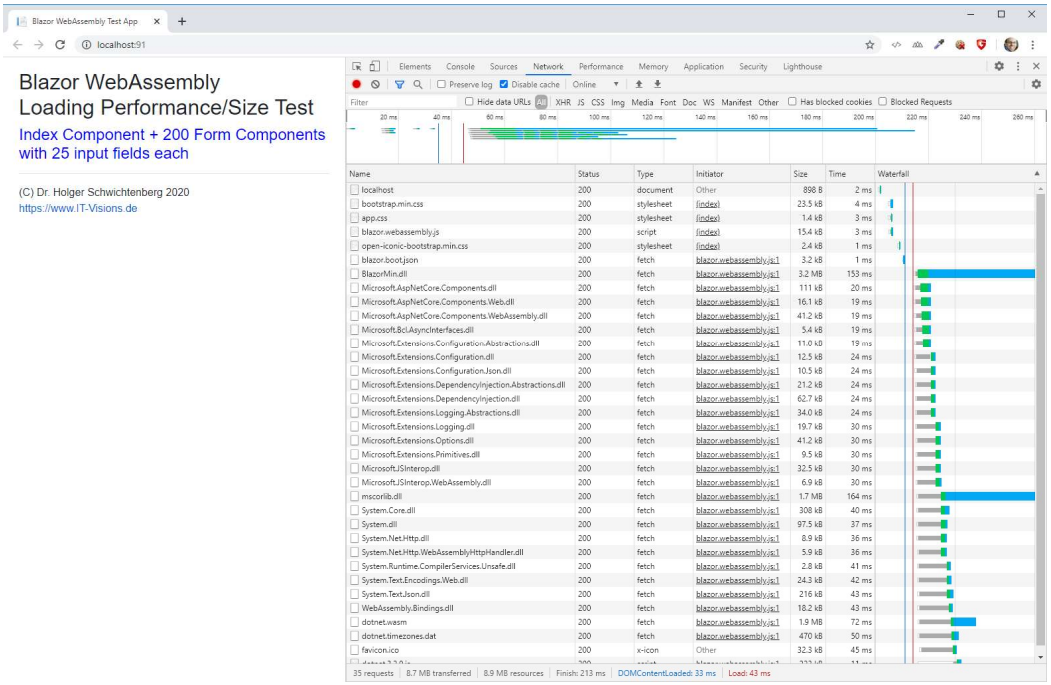


Abbildung: Eine größere Blazor WebAssembly-Anwendung mit 200 Eingabemasken

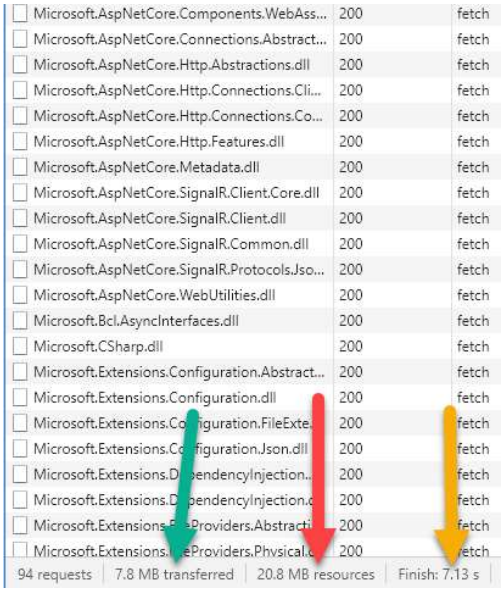


Abbildung: Erstes Laden der MiracleList-Implementierung mit Blazor WebAssembly

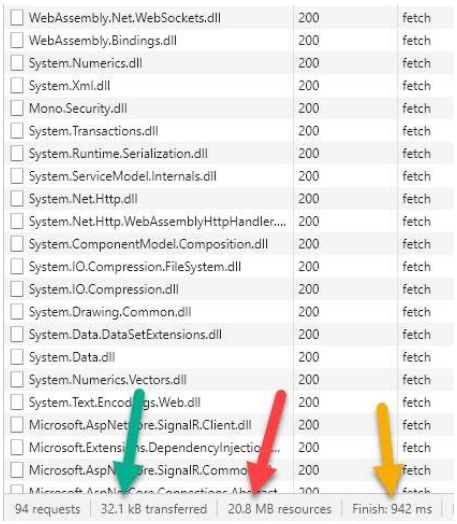


Abbildung: Zweites Laden der MiracleList-Implementierung mit Blazor WebAssembly bei aktiviertem Cache

Erschwerend kommt hinzu, dass es in Blazor bisher kein Nachladen von Modulen bei Bedarf gibt (Lazy Loading). Moderne JavaScript-Frameworks wie React, Angular und vueJS können dies, um die Ladelast zu Beginn zu verringern und das "Teile und Herrsche"-Prinzip zu realisieren.

Der Entwickler kann in Blazor zwar C#- und ggf. notwendigen JavaScript-Programmcode, Razor Components und auch statische Inhalte in eine sogenannte Razor Class Library auslagern, aber die wird dann als DLL bei Anwendungsstart automatisch mitgeladen. Ebenso fehlt in Blazor eine komponentenweise Isolation von JavaScript und Style Sheets. Beide gelten immer global für die ganze Anwendung.

6.8.5 Speicherlimit

WebAssembly ist derzeit eine 32-Bit-Architektur. Der nutzbare Speicher ist daher auf 4 GB RAM begrenzt [*https://v8.dev/blog/4gb-wasm-memory*]. Eine 64Bit-Variante der WebAssembly-VM (wasm64) ist in Arbeit.

In Blazor WebAssembly kommt man schon deutlich schneller an die Speichergrenze, wie dieser Testcode beweist. Offensichtlich steht etwas weniger 1 GB RAM zur freien Verfügung.

Listing: Testen der Speichergrenze von Blazor WebAssembly

```
public void MemoryTest()
 {
  Console.WriteLine("=== Blazor Memory Test by Dr. Holger Schwichtenberg, www.IT-Visions.de,
2020 ===");

  Stopwatch sw = new Stopwatch();
  int iterations = 100000000;
  int i = 0;

  System.Text.StringBuilder sb = new System.Text.StringBuilder();
  Console.WriteLine("StringBuilder MaxCapacity: " + MB(sb.MaxCapacity));
  // geht nicht:
"Microsoft.AspNetCore.Components.WebAssembly.Rendering.WebAssemblyRenderer[100]"
```

```
  //Console.WriteLine(System.Diagnostics.Process.GetCurrentProcess().ProcessName);
  //Console.WriteLine("System.Diagnostics.Process.GetCurrentProcess().PrivateMemorySize64: "
+ System.Diagnostics.Process.GetCurrentProcess().PrivateMemorySize64);
  Console.WriteLine("GC.GetTotalMemory(false): " + MB(GC.GetTotalMemory(false)));

  string MB(long bytes)
  {
   return Math.Round(bytes / 1024m / 1024m,2) + " MB";
  }

  void PrintProgress()
  {
   Console.WriteLine($"{sb.Length} characters / {sw.ElapsedMilliseconds:n} ms / {i:n}
iterations / {MB((GC.GetTotalMemory(false)))} used");
  }

  try
  {
   Console.WriteLine("Start...");

   sw.Start();

   for (i = 0; i < iterations; i++)
   {
    if ((i % (iterations / 10)) == 0) PrintProgress();
    sb.Append("Hello World!");
   }
   Console.WriteLine();
   sw.Stop();
   Console.WriteLine("DONE!");
   PrintProgress();
  }
  catch (Exception ex)
  {
   Console.WriteLine("ERROR: " + ex.ToString());
   PrintProgress();
  }
 }
```

```
=== Blazor Memory Test by Dr. Holger Schwichtenberg, www.IT-Visions.de, 2020 ===
StringBuilder MaxCapacity: 2048,00 MB
GC.GetTotalMemory(false): 4,14 MB
Start...
0 characters / 0,00 ms / 0,00 iterations / 4,14 MB used
120000000 characters / 17.042,00 ms / 10.000.000,00 iterations / 233,24 MB used
240000000 characters / 34.472,00 ms / 20.000.000,00 iterations / 462,35 MB used
360000000 characters / 52.061,00 ms / 30.000.000,00 iterations / 691,46 MB used
480000000 characters / 69.737,00 ms / 40.000.000,00 iterations / 920,58 MB used
ERROR: System.OutOfMemoryException: Out of memory
  at System.Text.StringBuilder.ExpandByABlock (System.Int32 minBlockCharCount) <0x2b61c38 + 0x000da> in <filename unknown>:0
  at System.Text.StringBuilder.Append (System.Char* value, System.Int32 valueCount) <0x2b61938 + 0x000fc> in <filename unknown>:0
  at System.Text.StringBuilder.AppendHelper (System.String value) <0x2b617d0 + 0x0002e> in <filename unknown>:0
  at System.Text.StringBuilder.Append (System.String value) <0x295a608 + 0x000e0> in <filename unknown>:0
  at MLBlazorRCL.Performance.Performance.MemoryTest () <0x31cd080 + 0x00112> in <filename unknown>:0
499960192 characters / 72.655,00 ms / 41.663.349,00 iterations / 958,68 MB used
```

Abbildung: Speichermangel in Google Chrome 83

```
=== Blazor Memory Test by Dr. Holger Schwichtenberg, www.IT-Visions.de, 2020 ===
StringBuilder MaxCapacity: 2048.00 MB
GC.GetTotalMemory(false): 4.14 MB
Start...
0 characters / 0.00 ms / 0.00 iterations / 4.14 MB used
120000000 characters / 12,195.00 ms / 10,000,000.00 iterations / 233.24 MB used
240000000 characters / 24,363.00 ms / 20,000,000.00 iterations / 462.35 MB used
360000000 characters / 36,506.00 ms / 30,000,000.00 iterations / 691.46 MB used
480000000 characters / 48,614.00 ms / 40,000,000.00 iterations / 920.58 MB used
ERROR: System.OutOfMemoryException: Out of memory
  at System.Text.StringBuilder.ExpandByABlock (System.Int32 minBlockCharCount) <0x2b62430 + 0x000da> in <filename unknown>:0
  at System.Text.StringBuilder.Append (System.Char* value, System.Int32 valueCount) <0x2b62130 + 0x000fc> in <filename unknown>:0
  at System.Text.StringBuilder.AppendHelper (System.String value) <0x2b61fc8 + 0x0002e> in <filename unknown>:0
  at System.Text.StringBuilder.Append (System.String value) <0x295a608 + 0x000e0> in <filename unknown>:0
  at MLBlazorRCL.Performance.Performance.MemoryTest () <0x31ca4d0 + 0x00112> in <filename unknown>:0
499960192 characters / 50,625.00 ms / 41,663,349.00 iterations / 958.68 MB used
```

Abbildung: Speichermangel in Firefox 77

6.8.6 Fazit

An der Frage, ob man Blazor WebAssembly oder Blazor Server einsetzen sollte, scheiden sich die Geister unter den .NET-Entwicklern, wie man an Diskussionen im Internet (vgl. z.B. https://twitter.com/quorralyne/status/1288121722336550913) sieht.

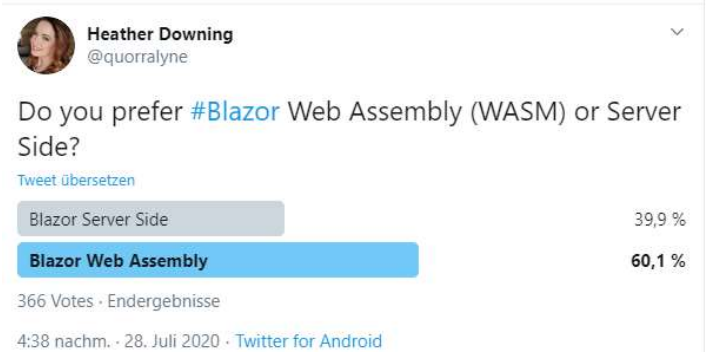


Abbildung 4: Nicht-Repräsentative Twitter-Umfrage [https://twitter.com/quorralyne/status/1288121722336550913]

Die bestehenden Unzulänglichkeiten von Blazor WebAssembly schränken den Einsatz auf Intra- und Extranets ein, denn durch die langen Ladezeiten von Blazor WebAssembly verliert man sehr viele Nutzer. Für öffentliche Websites wird man Blazor Server mit dem schnellen Anwendungsstart bevorzugen wollen. Hierbei hat man dann aber Herausforderungen bei hohen Nutzerzahlen und abreißenden Internetverbindungen.

Ein Vorteil von Blazor Server ist auch die höhere Produktivität, da man keine Serviceschicht für den Zugang zur Datenbank und anderen Ressourcen braucht. Wenn nicht sowieso schon eine WebAPI-Schicht vorhanden ist oder für die Kommunikation mit anderen Anwendungen eingepreist ist, spart man hier viel Budget.

Durch die mögliche Mehrfachverwendung von in Razor Class Library gekapselten Razor Components (sowohl in Blazor WebAssembly als auch in Blazor Server) ist es einfach möglich, eine Anwendung von einer Architektur in die andere zu überführen oder aber in beiden Architekturen gleichzeitig zu nutzen.

6.9 Blazor im Vergleich zu anderen Spielarten von ASP.NET Core

Die nächste große Abbildung zeigt den Vergleich aller fünf nun in ASP.NET Core verfügbaren Architekturalternativen:

- Ganz oben sieht man die Model-View-Controller-Architektur (MVC), welche es seit Version 1.0 (Jahr 2016) in ASP.NET Core gibt und die es zuvor schon im klassischen ASP.NET gab. Der Controller auf dem Server füttert den MVC-View mit Daten. Der View erzeugt mit der Razor Template-Syntax eine HTML-Seite, die als Ganzes zum Webbrowser übertragen wird. Der Browser schickt Daten per URL oder HTTP-Request-Body zurück an den Server, der dann eine neue, ganze Seite rendert und diese wieder zum Client sendet. Der Client tauscht die Seite aus, der Benutzer nimmt ein Flackern war. JavaScript zur Ausführung im Browser ist optional einstreubar in vom Webserver gerenderten HTML-Dokumenten.
- Seit 2017 (ASP.NET Core 2.0) gibt es die Razor Pages, die ein Page Model statt eines Controllers besitzen, aber die gleichen Interaktionen mit dem Webbrowser ermöglichen. Lediglich die Architektur auf dem Webserver, insbesondere die Zusammenarbeit zwischen Page Model und View, ist anders.
- Ebenfalls im klassischen ASP.NET als auch seit ASP.NET Core 1.0 gibt es das WebAPI-Modell, bei dem der Webserver lediglich Daten per REST-Dienst im JSON-Format an den Webbrowser liefert und das HTML-Rendering und die Benutzerinteraktionsereignisbehandlung komplett im Client per JavaScript/TypeScript stattfindet, in der Regel unterstützt durch SPA-Webframeworks wie Angular, React, Vue.js oder Aurelia. Dieses Modell ist heute die vorherrschende Architektur bei modernen Webanwendungen. Das Modell wird aber von vielen .NET-Entwicklern nicht geliebt, weil die Client-Programmierung mit einer anderen Programmiersprache und andersartigen Frameworks erfolgen muss, die nicht die in .NET geschätzten Eigenschaften hinsichtlich Komfort, Produktivität und Stabilität bieten.
- An vierter Stelle sieht man nun den neuen Blazor Server mit dem bereits oben geschriebenen Zyklus der Übertragung von Interaktionen und DOM-Änderungen via ASP.NET SignalR.
- Das fünfte Architekturmodell ist Blazor WebAssembly. Die Serverseite bietet hier wieder nur noch WebAPIs an. C# läuft nun auf Basis der Mono-Laufzeitumgebung in WebAssembly auf Browserseite.

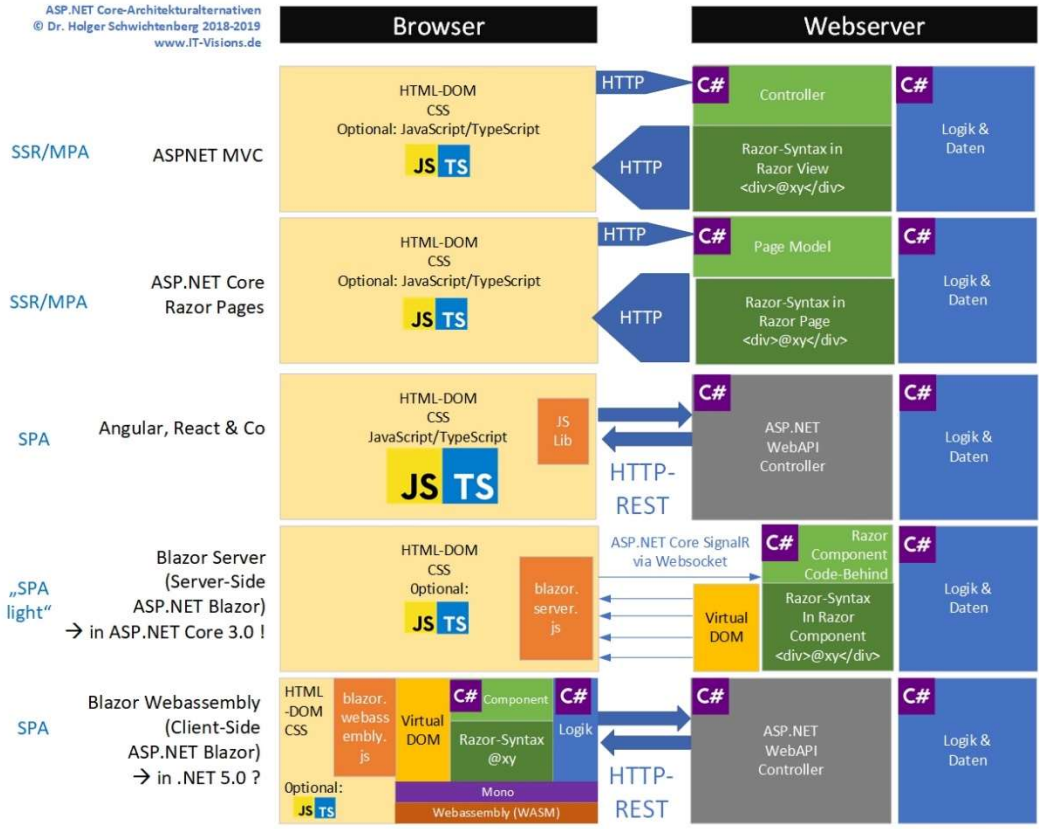


Abbildung: ASP.NET Core bietet mittlerweile fünf Architekturalternativen.

6.10 Blazor im Vergleich zu anderen Webframeworks

Es stellt sich die Frage, ob Blazor eine Alternative zu React, Angular, vueJS und anderen JavaScript- bzw. TypeScript-basierten Webframeworks darstellt. Dies kommt auf die Ausgangslage an. Bestehende, begeisterte JavaScript-Entwickler wird Blazor nicht locken können.

Blazor ist eine gute Alternative zu JavaScript für Entwicklungshäuser, die eine große bestehende Code-Basis und umfangreiches Know-how in .NET haben. Sie können nun moderne Webanwendungen mit großer Wiederverwendung entwickeln und so zum Beispiel bestehende Desktop-Anwendungen mit Windows Forms oder WPF ins Web bringen oder bestehende Multi-Page-Web-Applications modernisieren auf das Single-Page-Konzept. Bei einem bestehenden .NET-Backend hat Blazor den Vorteil des Teilens (Code Sharing) und Bewegens von Programmcode (Code Movement) zwischen Client und Server.

Beispiel für Code Sharing: Geschäftsobjektklassen, die auf dem Server geladen und in dem Client dargestellt werden. Auch für
Beispiel für Code Movement: Berechnungen, die ursprünglich auf dem Server ausgeführt wurden, werden zur Vermeidung von Netzwerkverkehr in den Client verlagert.

Blazor WebAssembly hat im Vergleich zu JavaScript-Frameworks mit Performance-Themen, der Anwendungsgröße und der (noch) fehlenden Modularisierung mit Lazy Loading zu kämpfen (siehe Tabelle). Das gibt auch Microsoft zu: "Blazor WebAssembly isn't going to win in any performance comparisons with JavaScript based frameworks like Angular or React." [*https://devblogs.microsoft.com/aspnet/blazor-webassembly-3-2-0-now-available/#comment-2276*].

Die folgende Tabelle stellt die Gemeinsamkeiten und Unterschiede zwischen Blazor, Microsoft Silverlight und JavaScript-basierten Single-Page-Webanwendungen dar.

	JavaScript-basierte SPA (z.B. Angular oder React)	Microsoft Silverlight	Blazor Server	Blazor WebAssembly	Blazor Hybrid	Blazor Native
Benutzer-schnittstellen-beschreibung	HTML	XAML	HTML	HTML	HTML	XAML
Formatierung	CSS	XAML	CSS	CSS	CSS	XAML
Benutzer-schnittstellen-steuerung	JavaScript (TypeScript)	C# oder Visual Basic .NET	C#, optional auch JavaScript (TypeScript)	C#, optional auch JavaScript (TypeScript)	C#, optional auch JavaScript (TypeScript)	C#
Unterstützte Browser	Alle wichtigen Browser	Browser mit Silverlight-Plug-In	Alle wichtigen Browser	Alle wichtigen Browser, außer Internet Explorer	-	-
Unterstützte Betriebssysteme	Alle Betriebssysteme	Windows, tlw. auch Linux und macOS	Alle Betriebssysteme	Alle Betriebssysteme	Windows, Linux, macOS	iOS und Android
Hersteller	Verschiedene Hersteller (z.B. Google und Facebook)	Microsoft	Microsoft	Microsoft	Microsoft	Microsoft
Entwicklungsstatus	Wird aktiv von vielen	Wird nicht mehr	Erste RTM-Version	Erste RTM-Version am	Noch experimentell	Noch experimentell

	Anbietern entwickelt.	weiterentwickelt.	vom 23.9.2019, wird aktiv weiterentwickelt	19.5.2020, wird aktiv weiterentwickelt		
Zusatzkomponenten und Werkzeuge	++	-	+	+	-	-
Performance	+	+	+	O	Offen	Offen
Startzeit	+	O	+	-	Offen	Offen
Modularisierung	+	+	O	O	Offen	Offen
Zukunft	+	--	+	+	Offen	Offen