

Dr. Holger Schwichtenberg

ASP.NET Core Blazor 5.0

Blazor WebAssembly

und Blazor Server

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



Buchversion:	3.19.1 (09.06.2021)
Verlag:	www.IT-Visions.de , Fahrenberg 40b, D-45257 Essen
Sprachkorrektur:	Matthias Bloch, M.A.
ISBN:	978-3-934279-35-3
Bezugsquelle gedruckt:	www.amazon.de/exec/obidos/ASIN/393427935X/itvisions-21
Bezugsquelle Kindle:	www.amazon.de/exec/obidos/ASIN/B08J5L3WQ7/itvisions-21
Bezugsquelle PDF:	www.leanpub.com/Blazor50

Für Heidi, Felix und Maja

1 Inhaltsverzeichnis (Hauptkapitel)

1	Inhaltsverzeichnis (Hauptkapitel)	4
2	Inhaltsverzeichnis (Details)	5
3	Vorwort	17
4	Über den Autor	19
5	Über dieses Buch	20
6	Programmcodebeispiel zum Download	24
7	Was ist ASP.NET Core Blazor?	41
8	Projektaufbau	82
9	Übersetzung und Debugging	102
10	Komponenten (Razor Components)	114
11	Routing und Navigation	134
12	Razor-Syntax	147
13	Ereignisse und Ereignisbehandlung	156
14	Komponenteneinbettung / Unterkomponenten	166
15	Zustandsverwaltung	188
16	Formulare/Eingabemasken	205
17	Klassenbibliotheken und Razor Class Libraries (RCL)	239
18	Dependency Injection (DI)	255
19	Interoperabilität mit JavaScript	264
20	Zugriff auf Webservices/WebAPIs	295
21	Benachrichtigungen mit ASP.NET Core SignalR in Blazor	315
22	Serversystem- und Browserinformationen	324
23	Authentifizierung und Benutzerverwaltung	333
24	Installation von Blazor-Anwendungen	362
25	Tipps, Tricks und Tools	371
26	Fallbeispiel "MiracleList"	445
27	Umstieg von ASP.NET Webforms zu ASP.NET Core Blazor	521
28	Ausblick auf Blazor 6.0	556
29	Hybride Apps mit Blazor Desktop	565
30	Mobile Apps mit Mobile Blazor Bindings	582
31	Quellen im Internet	590
32	Versionsgeschichte dieses Buchs	591
33	Stichwortverzeichnis (Index)	600
34	Werbung in eigener Sache ☺	611

2 Inhaltsverzeichnis (Details)

1	Inhaltsverzeichnis (Hauptkapitel).....	4
2	Inhaltsverzeichnis (Details).....	5
3	Vorwort	17
4	Über den Autor.....	19
5	Über dieses Buch.....	20
5.1	Versionsgeschichte dieses Buchs	20
5.2	Bezugsquelle für Aktualisierungen	20
5.3	Geplante Kapitel.....	20
5.4	Programmiersprache in diesem Buch.....	21
5.5	Notwendige Vorkenntnisse	21
5.6	Ihre Belohnung, wenn Sie helfen, dieses Buch zu verbessern!	23
6	Programmcodebeispiel zum Download.....	24
6.1	Webadresse für Downloads.....	24
6.2	Übersicht über die Beispiele.....	24
6.3	Eingesetztes CSS-Framework	26
6.4	Das Fallbeispiel "MiracleList"	26
6.4.1	Szenario	26
6.4.2	Technische Basis für MiracleList	26
6.4.3	Webadressen.....	27
6.4.4	Projektmappenaufbau	27
6.4.5	MiracleList-Bildschirmmasken	28
6.5	Weitere Beispiele innerhalb der MiracleList-Webanwendung.....	32
6.6	Visual Studio 2019	33
6.7	.NET SDK	35
6.8	Testen Ihrer PC-Konfiguration.....	37
6.9	Util.Log()	39
6.10	Qualitätssicherung der Programmcodebeispiele.....	39
7	Was ist ASP.NET Core Blazor?.....	41
7.1	Blazor-Arten.....	41
7.2	Geschichte von Blazor.....	43
7.3	Technischer Support für Blazor	44
7.3.1	Support für Blazor Server.....	44
7.3.2	Support für Blazor WebAssembly.....	45

7.4	Blazor Server.....	46
7.4.1	Rendering und Interaktion bei Blazor Server	46
7.4.2	Softwarearchitektur	47
7.4.3	Netzwerkverkehr bei Blazor Server	47
7.4.4	Ressourcenbedarf und Skalierbarkeit	48
7.4.5	Vor- und Nachteile von Blazor Server	50
7.5	Blazor WebAssembly	51
7.5.1	Konzept von Blazor WebAssembly	51
7.5.2	WebAssembly (WASM)	52
7.5.3	Sandbox des Browsers / API-Einschränkungen	53
7.6	NuGet-Pakete für Blazor	55
7.7	Neuerungen in Blazor 5.0.....	55
7.8	Fehlende Funktionen in Blazor 5.0	58
7.9	Vergleich zwischen Blazor WebAssembly und Blazor Server	59
7.9.1	Die wichtigsten Unterschiede.....	59
7.9.2	Browserunterstützung für Blazor	65
7.9.3	Performance-Vergleich	67
7.9.4	Anwendungsgröße und Startzeiten.....	72
7.9.5	Speicherlimit	74
7.9.6	Fazit.....	76
7.10	Blazor im Vergleich zu anderen Spielarten von ASP.NET Core	77
7.11	Blazor im Vergleich zu anderen Webframeworks.....	78
7.12	Uno Platform	80
8	Projektaufbau	82
8.1	Inhalte der Standard-Projektvorlage.....	82
8.2	Blazor-Projekte anlegen	83
8.3	Aufbau eines Blazor Server-Projekts	84
8.3.1	Ordner und Dateien	84
8.3.2	Projektdatei.....	86
8.3.3	Start-Code.....	86
8.3.4	Genauere Fehlermeldungen aktivieren (Detailed Errors)	89
8.3.5	Struktur des Seitenaufbaus	90
8.4	Aufbau eines Blazor WebAssembly-Projekts	91
8.4.1	Ordner und Dateien	94
8.4.2	Aufbau der index.html.....	94
8.4.3	Projektdatei.....	94

8.4.4	Start-Code im Client-Projekt.....	95
8.4.5	Start-Code im Server-Projekt.....	96
8.4.6	Struktur des Seitenaufbaus	98
8.5	Anlegen einer kompletten mehrschichtigen Projektstruktur mit der .NET CLI	98
9	Übersetzung und Debugging	102
9.1	Übersetzung.....	102
9.2	Start von Blazor-Projekten in Visual Studio	103
9.3	Automatische Neukompilierung bei Änderungen	104
9.4	Visual Studio-Debugging in Blazor Server-Projekten	107
9.5	Visual Studio-Debugging in Blazor WebAssembly-Projekten	108
9.6	Ablaufverfolgung des ASP.NET Core Webservers.....	111
9.7	Browser-Debugging in Blazor WebAssembly	112
9.8	Veröffentlichen.....	113
10	Komponenten (Razor Components).....	114
10.1	Einsatz von Razor Components.....	114
10.2	Namen für Komponenten	114
10.3	Grundkonzepte am Beispiel einer "Hello World"-Komponente	114
10.4	Markup-Dateien mit Inline-Code	116
10.5	Komponenten mit Code-Behind-Datei.....	118
10.5.1	Code-Behind-Dateien auf Basis von Vererbung	119
10.5.2	Code-Behind-Dateien auf Basis von partiellen Klassen.....	122
10.6	Komponenten ohne Markup nur mit Programmcode	124
10.7	CSS-Isolation (ab Blazor 5.0)	127
10.7.1	Scoped Styles anlegen	127
10.7.2	Scoped Styles einbinden.....	127
10.7.3	CSS-Isolation mit Razor Class Libraries.....	128
10.8	Nicht-visuelle Komponenten.....	129
10.9	Layoutseiten (Masterpages)	131
11	Routing und Navigation	134
11.1	Routendefinitionen	134
11.2	Routen mit Parametern	135
11.2.1	Navigation per <a>-Tag	137
11.2.2	URL-Fragmente	137
11.2.3	Komponente <NavLink>.....	137
11.2.4	Navigation im Code.....	138
11.3	Auswerten der aktuellen URL	139

11.3.1	Fallunterscheidung bei Mehrfachrouten	140
11.3.2	Query Strings.....	140
11.3.3	Komplette URL-Auswertung	141
11.3.4	Routerkonfiguration	145
12	Razor-Syntax.....	147
12.1	Unterschiede zwischen Razor in Blazor und Razor in MVC und Razor Pages.....	147
12.2	Ausdrücken vs. Befehlsfolgen.....	151
12.3	Asynchrone Ausdrücke in der Razor-Syntax	152
12.4	Praxisaufgabe zur Razor-Syntax	153
12.4.1	Aufgabenstellung.....	153
12.4.2	Lösung.....	154
13	Ereignisse und Ereignisbehandlung	156
13.1	Komponentenlebenszyklusereignisse.....	156
13.2	Reaktion auf HTML-Ereignisse	157
13.3	Eigene Parameter bei HTML-Ereignissen.....	158
13.4	Standardparameter.....	159
13.5	Standardereignisbehandlung unterbinden.....	162
13.6	Ereignisweitergabe unterbinden	163
13.7	Komplexe Befehlsfolgen als Ereignisbehandlung.....	165
14	Komponenteneinbettung / Unterkomponenten.....	166
14.1	Einbetten von Razor Components	166
14.2	Parameter für eingebettete Razor Components	167
14.2.1	Parameter mit Value Type.....	167
14.2.2	Parameter mit Reference Type	171
14.3	Kaskadierende Parameter	173
14.4	Vorlagenbasierte Komponenten (Templated Components)	177
14.4.1	Einfache vorlagenbasierte Komponente	177
14.4.2	Vorlagenbasierte Komponente mit mehreren Fragmenten	178
14.4.3	Datenübergabe an Renderfragment via Kontext.....	179
14.4.4	Komplexe Objekte als Kontext	180
14.4.5	Vorlagenbasierte Komponente mit generischem Typparameter.....	182
14.4.6	Verschachtelung von Renderfragmenten.....	183
14.4.7	Standardwerte für Renderfragmente.....	184
14.4.8	Praxislösung: Repeater	185
15	Zustandsverwaltung	188
15.1	Komponentenzustand	188

15.2	Testanwendung: Counter.....	188
15.3	Verlust des Zustandes.....	189
15.4	Bewahrung des Zustandes	191
15.5	Sitzungszustand.....	191
15.6	Generischer Sitzungszustand.....	192
15.7	Nutzung des Webbrowserspeichers.....	195
15.7.1	NuGet-Pakete	195
15.7.2	Einsatz von Blazored.LocalStorage.....	196
15.7.3	Verschlüsselter Browserspeicher (Protected Browser Storage)	199
15.8	Cookies.....	202
15.9	Persistierung in Datenbanken oder anderen persistenten Speichern auf dem Server ...	203
16	Formulare/Eingabemasken.....	205
16.1	Formulare auf Basis der Blazor-Eingabekomponenten.....	205
16.1.1	Verfügbare Blazor-Eingabekomponenten	205
16.1.2	Komponente <EditForm>.....	206
16.1.3	Datenbindung	207
16.1.4	Datenannotationen.....	207
16.1.5	Validierungskomponenten in Blazor	209
16.1.6	Eigene Validierungsannotationen.....	209
16.1.7	Beispiel.....	210
16.1.8	Optionsfelder (<InputRadio> und <InputRadioGroup>)	215
16.1.9	Dateien hochladen (<InputFile>)	217
16.1.10	Praxisbeispiel: Dateien-Upload bei Blazor Server	218
16.2	Formulare auf Basis von Standard-HTML-Tags.....	224
16.2.1	Datenbindung	224
16.2.2	Reaktion auf einzelne Buchstabeneingaben	224
16.2.3	Auswahlsteuerelemente (<select>).....	229
16.2.4	Datenbindung mit Datumsangaben	230
16.2.5	Praxisbeispiel	232
16.3	Tipps & Tricks zu Formularen	236
16.3.1	Direkter Objektverweis	236
16.3.2	Fokus setzen	237
17	Klassenbibliotheken und Razor Class Libraries (RCL).....	239
17.1	.NET Standard.....	239
17.2	Referenzierbarkeit	240
17.3	Anlegen der Klassenbibliotheken.....	241

17.4	Einsatzgebiete der verschiedenen Klassenbibliotheksarten.....	243
17.5	Razor Class Libraries (RCL)	243
17.5.1	Erstellen einer Razor Class Library.....	243
17.5.2	Inhalte einer Razor Class Library.....	245
17.5.3	Referenzierung einer Razor Class Library	245
17.5.4	Einbettung von Razor Components aus einer Razor Class Library.....	247
17.5.5	Routing zu Razor Components aus einer Razor Class Library	247
17.5.6	Nutzung von statischen Ressourcen aus einer Razor Class Library	248
17.6	Lazy Loading von DLLs (ab Blazor 5.0)	249
17.6.1	Integration von Lazy Loading in die Standardprojektvorlage	250
17.6.2	Integration von Lazy Loading in das MiracleList-Fallbeispiel.....	252
17.7	Code-Sharing zwischen Blazor Server und Blazor WebAssembly	253
18	Dependency Injection (DI).....	255
18.1	Microsoft.Extensions.DependencyInjection.....	255
18.2	Implementierung eines Dienstes.....	255
18.3	Lebensdauer von Instanzen	256
18.4	Registrierung von Diensten	257
18.5	Automatisch registrierte Dienste	258
18.6	Injectiion im Template mit <code>@inject</code>	261
18.7	Injectiion in Properties mit <code>[Inject]</code>	261
18.8	Injectiion in einen Konstruktor (Konstruktorinjection)	261
18.9	Manuelle Beschaffung von Instanzen	262
19	Interoperabilität mit JavaScript	264
19.1	Motivation	264
19.2	Einschränkung der JavaScript-Interoperabilität in Blazor Server	264
19.3	Aufruf von C# zu JavaScript	265
19.4	Eigene JavaScript-Dateien verwenden	266
19.4.1	Globale Skripteinbindung.....	266
19.4.2	JavaScript-Isolation / <code>IJSObjectReference</code> (ab Blazor 5.0)	269
19.5	Aufruf von JavaScript zu C# (statische Methoden).....	270
19.6	Praxisbeispiel	271
19.7	Aufruf von JavaScript zu C# (Instanzmethoden)	274
19.8	Praxisbeispiel: Nutzung einer vorhandenen JavaScript-Bibliothek am Beispiel <code>Chartist.js</code>	275
19.9	Praxisbeispiel: Integration mit Angular über Web Components	279
19.9.1	Impementierung einer Web Component mit Angular Elements.....	280

19.9.2	Nutzung der Web Component in JavaScript	282
19.9.3	Einbindung von Web Components in Blazor	283
19.10	Praxisbeispiel: Angular-Datagrid in Blazor	286
19.10.1	Implementierung der Web Component in Angular	286
19.10.2	Nutzung der Web Component in JavaScript	289
19.10.3	Einbindung der Web Component <angular-grid> in Blazor	289
20	Zugriff auf Webservices/WebAPIs	295
20.1	Daten- und Ressourcenzugriffe in Blazor WebAssembly	295
20.2	Daten- und Ressourcenzugriffe in Blazor Server und Blazor Desktop	296
20.3	Webservice-Arten.....	296
20.4	Szenario.....	296
20.5	REST-Dienste (WebAPIs)	298
20.5.1	WebAPIs erstellen mit ASP.NET Core	298
20.5.2	WebAPIs testen mit Postman	299
20.5.3	Blazor-Zugriff auf Web APIs.....	300
20.5.4	Cross-Origin Resource Sharing für das WebAPI	302
20.5.5	Metadaten mit Open API Specification.....	302
20.5.6	Client-Generierung aus Metadaten mit Visual Studio	304
20.5.7	Client-Generierung aus Metadaten mit NSwagStudio	306
20.6	Google RPC-Dienste	307
20.6.1	gRPC-Dienst erstellen	307
20.6.2	Blazor-Zugriff auf gRPC-Dienste	311
20.7	Leistungsvergleich zwischen WebAPI und gRPC	312
20.8	SOAP und die Windows Communication Foundation (WCF).....	313
21	Benachrichtigungen mit ASP.NET Core SignalR in Blazor	315
21.1	Implementierung im Webserver	316
21.2	Implementierung im Blazor-Client.....	317
21.3	Aktivieren der Websockets-Unterstützung.....	321
21.4	Diagnose von ASP.NET Core SignalR	323
22	Serversystem- und Browserinformationen	324
22.1	HttpContext	324
22.2	Allgemeine Systeminformationen	325
22.3	Host Environment-Informationen	330
22.4	Verwenden von Environmentnamen	330
23	Authentifizierung und Benutzerverwaltung	333
23.1	Verwenden von ASP.NET Core Identity in Blazor Server	333

23.1.1	Aktivieren der Authentifizierung in Blazor Server	333
23.1.2	Individuelle Benutzerkontenverwaltung.....	334
23.1.3	Anpassung der Benutzerverwaltungsseiten	336
23.1.4	Weitere Anpassung von ASP.NET Identity	339
23.2	Authentifizierung in Blazor WebAssembly	339
23.2.1	OIDC-Authentifizierung bei Projekten ohne "ASP.NET Core Hosted".....	340
23.2.2	Weitere OIDC-Optionen	344
23.2.3	Authentifizierung bei Projekten mit "ASP.NET Core Hosted"	344
23.3	Eigene Authentifizierungsprovider (AuthenticationStateProvider).....	348
23.3.1	AuthenticationStateProvider für Debugging-Zwecke	349
23.3.2	AuthenticationStateProvider in MiracleList	349
23.3.3	Nutzung des eigenen AuthenticationStateProvider	356
23.4	Autorisierung.....	357
23.4.1	Zugriff auf den angemeldeten Benutzer	358
23.4.2	Autorisierung von Komponenten	359
23.4.3	Inhalte für angemeldete Benutzer.....	359
23.4.4	Umleitung für nicht-autorisierte Benutzer.....	359
24	Installation von Blazor-Anwendungen.....	362
24.1	Deployment	362
24.2	Installation auf Azure-Diensten	363
24.2.1	WebSockets	363
24.2.2	Brotli-Komprimierung.....	364
24.3	Installationsvoraussetzungen für eigene Windows-Webserver (IIS).....	365
24.4	Installation auf einem eigenen Windows-Webserver (IIS).....	366
24.5	Self-Hosting (Kestrel Webserver)	370
25	Tipps, Tricks und Tools	371
25.1	Ermitteln der Blazor-Art und -Versionsnummer.....	371
25.2	Ausgaben in die Browserkonsole	373
25.3	HTML-Kopf beeinflussen	376
25.4	BlazorFiddle	377
25.5	Render-Modi bei Blazor Server.....	378
25.5.1	Server Prerendered	379
25.5.2	Server	379
25.5.3	Static.....	380
25.5.4	Static mit Parametern	380
25.5.5	Mehrere Blazor Server-Komponenten in einer Seite.....	381

25.6	Render-Modi bei Blazor WebAssembly.....	382
25.7	Anwendungskonfigurationsdateien	383
25.7.1	AppSettings.json	383
25.7.2	Listen in Konfigurationseinträgen	385
25.7.3	Andere Konfigurationsformate.....	385
25.8	Steuerung des Rendering.....	386
25.9	Leistungsoptimierung beim Rendern großer Datenmenge durch Virtualisierung	390
25.9.1	Szenario	392
25.9.2	Suksessives Rendern mit <Virtualize>.....	393
25.9.3	Suksessives Laden per ItemsProvider-Funktion.....	395
25.10	Überwachung von Blazor Server-Anwendungen	396
25.10.1	Überwachung der Netzwerklatenz bei Blazor Server.....	396
25.10.2	Überwachung der Sitzungen/Circuits bei Blazor Server	398
25.11	Hintergrundaufgaben und Fortschrittsanzeige.....	403
25.12	Zeitgesteuerte Aufgaben via Timer	406
25.12.1	Praxisbeispiel: Zeitgesteuerte Seitenaktualisierung.....	407
25.12.2	Praxisbeispiel: Countdown	408
25.12.3	Eigene Timer-Komponente für Blazor	409
25.13	Meldungskomponente	410
25.14	Modale Dialoge mit Bootstrap	415
25.15	Benachrichtigungen.....	417
25.15.1	Desktop-Benachrichtigungen	418
25.15.2	Benachrichtigungen im Browserfenster (Toasts)	421
25.16	Zugriff auf die lokale Zeit des Webbrowsers	421
25.17	Custom Boot Resource Loading.....	422
25.18	Progressive Web Applications (PWA) mit Blazor WebAssembly.....	423
25.19	Verzögerter Start einer Blazor WebAssembly-Anwendung.....	427
25.20	Integration von Blazor-Anwendungen in andere Webanwendungen	428
25.21	Mehrsprachigkeit (Lokalisierung/Globalisierung/ Internationalisierung)	429
25.21.1	Festlegung der aktuellen Sprache	429
25.21.2	Texte aus Ressourcendateien.....	429
25.21.3	Beispiel.....	429
25.21.4	Notwendige Infrastruktur bei Blazor WebAssembly.....	433
25.21.5	Notwendige Infrastruktur bei Blazor Server.....	434
25.21.6	Darstellung von Datums- und Zahleingabefeldern.....	436
25.22	Optimierung der Download-Größe bei Blazor WebAssembly	436

25.23	Micro-Apps mit Blazor	438
25.23.1	Projektaufbau.....	439
25.23.2	Kommunikation.....	441
26	Fallbeispiel "MiracleList"	445
26.1	Das MiracleList-Backend.....	445
26.1.1	Softwarearchitektur	448
26.1.2	Entitätsklassen/Geschäftsobjekte/Datentransferobjekte	451
26.1.3	Entity Framework Core-Kontextklasse	453
26.1.4	Lebensdauer der Kontextklasse in ASP.NET Core-Anwendungen.....	454
26.1.5	Geschäftslogik.....	455
26.1.6	Web API.....	463
26.2	MiracleList-Frontend mit Blazor Server	473
26.2.1	Softwarearchitektur	473
26.2.2	Startcode (Program.cs und Startup.cs)	476
26.2.3	Startseite (_Host.cshtml)	478
26.2.4	Anmeldeansicht (Login.razor).....	480
26.2.5	Authentication State Provider (MLAuthenticationStateProvider).....	483
26.2.6	Hauptansicht (Index.razor).....	487
26.2.7	Bearbeitungsformular (TaskEdit.razor).....	496
26.3	MiracleList-Frontend mit Blazor WebAssembly	498
26.3.1	Softwarearchitektur	498
26.3.2	Projektdatei.....	500
26.3.3	Startseite mit Ladeanimation (index.html)	502
26.3.4	Blazor WebAssembly-Startcode (Program.cs).....	503
26.3.5	Anmeldeansicht (Login.razor).....	505
26.3.6	Authentication State Provider (MLAuthenticationStateProvider).....	508
26.3.7	Hauptansicht (Index.razor).....	511
26.3.8	Bearbeitungsformular (TaskEdit.razor).....	518
27	Umstieg von ASP.NET Webforms zu ASP.NET Core Blazor	521
27.1	Migrationspfade und Herausforderungen.....	521
27.2	Grundsätzliche Vorgehensweise bei der Migration.....	523
27.3	Umstieg von Webforms auf Blazor an einem Beispiel	524
27.3.1	Umsetzung der Datentabelle mit Webforms und Blazor	525
27.3.2	Umstellung der Benutzerschnittstellenbeschreibung.....	531
27.3.3	Umstellung der Benutzerschnittstellensteuerung	531
27.4	Einsatz von Telerik UI for Blazor	532

27.4.1	Komponenten in Telerik UI for Blazor	532
27.4.2	Bezugsquelle und Preise.....	533
27.4.3	Installation von Telerik UI for Blazor	534
27.4.4	Erstellen eines Projekts mit Telerik UI for Blazor	536
27.4.5	Integration von Telerik UI for Blazor in bestehende Blazor-Projekte.....	539
27.4.6	Umsetzung der Flugdatentabelle mit Telerik UI for Blazor	540
27.5	Einsatz von DevExpress UI for Blazor	546
27.5.1	Komponenten in DevExpress UI for Blazor.....	546
27.5.2	Bezugsquelle und Preise.....	546
27.5.3	Installation von DevExpress UI for Blazor	547
27.5.4	Erstellen eines Projekts mit DevExpress UI for Blazor.....	548
27.5.5	Umsetzung der Flugdatentabelle mit DevExpress UI for Blazor	550
28	Ausblick auf Blazor 6.0.....	556
28.1	Blazor Desktop.....	556
28.2	Schnelleres Kompilieren (seit .NET 6 Preview 2).....	556
28.3	Dynamic Component (seit .NET 6 Preview 1).....	556
28.4	Hot Reloading (seit .NET 6 Preview 3).....	558
28.5	Ahead-of-Time-Kompilierung (AOT) (seit .NET 6 Preview 4).....	560
28.6	Error Boundaries (seit .NET 6 Preview 4)	561
28.7	Weitere Verbesserungen.....	563
29	Hybride Apps mit Blazor Desktop	565
29.1	Hybride Apps	565
29.2	Architektur von Blazor Desktop.....	565
29.3	Blazor Desktop in WPF und Windows Forms	567
29.4	Beispiel: Blazor-Anwendung in Windows Forms.....	568
29.5	Implementierungsdetails	571
29.6	Integration in WPF	579
29.7	Verteilungsoptionen	579
29.8	Blazor Desktop in .NET MAUI.....	579
30	Mobile Apps mit Mobile Blazor Bindings	582
30.1	Architektur der Blazor Mobile Bindings	582
30.2	Versionsgeschichte.....	583
30.3	Projekt anlegen.....	583
30.4	Übersetzung und Debugging für Android	584
30.5	Aufbau der Projektvorlage	586
30.6	Verwenden von HTML-Rendering	587

30.7	Verfügbare Steuerelemente	588
31	Quellen im Internet	590
32	Versionsgeschichte dieses Buchs	591
33	Stichwortverzeichnis (Index)	600
34	Werbung in eigener Sache ☺	611
34.1	Dienstleistungen	611
34.2	Aktion "Buch für Buchrezension"	612
34.3	Aktion "Buch-Abo"	613

3 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, AngularJS, Angular und diversen anderen Bibliotheken und Frameworks). 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 beide Blazor-Varianten (Blazor Server seit September 2019 und Blazor WebAssembly seit Mai 2020) in einer stabilen Version verfügbar. Dieses Buch behandelt die Version **Blazor 5.0**, die dritte Version von Blazor Server und die zweite Version von Blazor WebAssembly.

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

Mit Blazor konnten wir schon einige SPA-Webanwendungen in Rekordzeit entwickeln, denn man ist als Entwickler damit sehr produktiv. Wenn man lange Erfahrung mit den verschiedenen Microsoft-Webframeworks einerseits und JavaScript-basierten Frameworks andererseits hat, ist man mit Blazor wirklich sehr schnell. 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 bei unseren Kunden im deutschsprachigen Raum immer mehr klassische Windows-Desktop-Entwickler, 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 noch einige Hürden zu überwinden, wie immer bei den ersten Versionen einer neuen Technik.

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.

Das Buch setzt aber voraus, dass Sie .NET, C# und Visual Studio schon kennen. Ich biete dazu aber auch weitere Bücher an, siehe Kapitel "Notwendige Vorkenntnisse".

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 49,99 Euro (der Autor erhält 22,05 Euro):
www.amazon.de/exec/obidos/ASIN/393427935X/itvisions-21
- Kindle-E-Book bei Amazon.de für 44,99 Euro (der Autor erhält 14,99 Euro):
www.amazon.de/exec/obidos/ASIN/B08J5L3WQ7/itvisions-21
- PDF inkl. Updates bei Leanpub.com ab 44,99 Dollar (der Autor erhält ca. 28,75 Euro):
www.leanpub.com/Blazor50

Tipp: Sie können jederzeit Aktualisierungen des PDF-Buchs (gleiche Hauptversion) kostenfrei bei Leanpub.com beziehen. Amazon erlaubt dies leider nicht! 😞 Ich biete daher Käufer bei Amazon die PDF-Version zum Sonderpreis von 19 Dollar an:

www.leanpub.com/Blazor50/c/Picard

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, ist 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: www.dotnet-doktor.de/Leserfeedback

Tipp: Ich belohne Sie mit E-Books für gemeldete Fehler, siehe Kapitel "Über dieses Buch/Ihre Belohnung, wenn Sie helfen, dieses Buch zu verbessern".

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** zu Blazor und .NET oder anderen Themen rund um die Entwicklung und den Betrieb von Anwendungen (Desktop, Web und Mobile) sowie Server und Cloud benötigen, stehe ich Ihnen im Rahmen meiner beruflichen Tätigkeit für die Firmen www.IT-Visions.de (Beratung, Schulung, Support) und MAXIMAGO GmbH (Softwareentwicklung, siehe www.MAXIMAGO.de) gerne zur Verfügung. Bitte wenden Sie sich für ein Angebot an das jeweilige Kundenteam. Bitte kontaktieren Sie die Firmen aber nicht für Feedback und Verbesserungsvorschläge zu diesem Buch, da dieses Buch reine Privatsache ist.

Die Beispiele zu diesem Buch können Sie herunterladen auf der von mir ehrenamtlich betriebenen **Leser-Website** unter: www.dotnet-doktor.de/Leser

Dort müssen Sie sich zunächst registrieren. Bei der Registrierung wird ein Lösungswort abgefragt. Bitte geben Sie dort **Picard** ein (siehe auch Kapitel "Programmcodebeispiele zum Download").

Herzliche Grüße aus Essen, dem Herzen der Metropole Ruhrgebiet

Holger Schwichtenberg

4 Über den Autor

- Studienabschluss Diplom-Wirtschaftsinformatik an der Universität Essen
- Promotion an der Universität Essen im Fachgebiet komponentenbasierter Softwareentwicklung
- Seit 1996 selbstständig als unabhängiger Berater, Dozent, Softwarearchitekt und Fachjournalist
- Fachlicher Leiter des Expertenteams bei www.IT-Visions.de in Essen
- Chief Technology Expert (CTE) der Softwareentwicklung bei der MAXIMAGO GmbH in Dortmund (www.MAXIMAGO.de)
- Über 80 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), kontinuierlich ausgezeichnet seit 2004
 - Microsoft Certified Solution Developer (MCSD)
- Thematische Schwerpunkte:
 - Softwarearchitektur, mehrschichtige Softwareentwicklung, Softwarekomponenten, SOA
 - Visual Studio, Continous Integration (CI) und Continous Delivery (CD) mit Azure DevOps
 - Microsoft .NET (.NET Framework, .NET Core), C#, Visual Basic
 - .NET-Architektur/Auswahl von .NET-Techniken
 - Einführung von .NET und Visual Studio/Migration auf .NET
 - Webanwendungsentwicklung und Cross-Plattform-Anwendungen mit HTML, ASP.NET (Core), 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
 - Objekt-Relationales Mapping (ORM), insbesondere ADO.NET Entity Framework und Entity Framework Core
 - PowerShell
 - Architektur- und Code-Reviews
 - Performance-Analysen und -Optimierung
 - Entwicklungsrichtlinien
- Ehrenamtliche Community-Tätigkeiten:
 - Vortragender für die International .NET Association (INETA)
 - Betrieb diverser Community-Sites: 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



www.IT-Visions.de
Dr. Holger Schwichtenberg



MAXIMAGO

- 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

5 Über dieses Buch

5.1 Versionsgeschichte dieses Buchs

Die Versionsgeschichte dieses Buch finden Sie in einem eigenen Kapitel am Ende des Buchs.

Hinweis: Die Versionsgeschichte ist eine wichtige Referenz für die Leser, die sich aktuelle Versionen des Buchs beschaffen (z.B. über Leanpub.com) und wissen wollen, was sich geändert hat. Wenn Sie das Buch erstmalig lesen, müssen Sie die Versionsgeschichte nicht lesen.

5.2 Bezugsquelle für Aktualisierungen

Leanpub-Kunden 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 19,00 Dollar (zzgl. 5% Mehrwertsteuer, ab 1.1.2021: 7%) unter folgender Webadresse beziehen:

<https://leanpub.com/ASPNETBlazor/c/Picard>

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 und damit auch ein neues Buchprojekt bei Amazon und Leanpub erstellt werden muss.

5.3 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.

- AdditionalAttributes und @attributes für Komponenten
- Autorisierung mit Rollen
- Blazor WebForms Components (A collection of Blazor components that emulate the web forms components of the same name) →
<https://github.com/FritzAndFriends/BlazorWebFormsComponents>
- Cross-Platform-HTML-Apps mit Blazor Electron
- DotNet.createJSObjectReference()
- Eigene Basisklassen für Razor Components
- Fehlerbehandlung / blazor-error-ui → **Gehört zum Standardlieferumgang der Projektvorlagen und wird bereits verwendet in MiracleList.**
- Fluent Validator (<https://github.com/Blazored/FluentValidation>)
- Host Environments in Blazor Server verwenden
- IComponentActivator zur Kontrolle der Komponenteninstanziierung (ab Blazor 5.0)
- Installation von Blazor-Anwendungen auf Linux-Servern
- IL-Linker-Konfigurationsdatei (LinkerConfig.xml)

- JavaScript-Interop ohne Marshalling mit IJSInProcessRuntime und IJSUnmarshalledRuntime (in Blazor WebAssembly)
- Razen Blazor Components (kostenfrei) → <https://razor.radzen.com>
- Blazorise Components (kostenfrei) → <https://efroliddemo.blazorise.com>
- Synchrone JavaScript-Interop mit IJSInProcessRuntime (in Blazor WebAssembly ab Blazor 5.0)
- LESS und SCSS mit Blazor
- Leistungsoptimierung (@key, Inlining, ShouldRender u.a.)
- Microsoft.AspNetCore.Components.Web.Extensions (ab Blazor 5.0)
- 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)
- PreseveWhiteSpace
- Routing-Anpassungen (<https://chrissainty.com/building-a-custom-router-for-blazor/amp/>)
- Testen von Blazor-Anwendungen (Microsoft.AspNetCore.Components.Testing und BUnit)
→ **Beispiele zu BUnit finden Sie bereits im MiracleList-Beispiel, das Sie herunterladen können!**
- TypeScript in Blazor Apps
- Validierung: Anpassung der CSS-Klassen (ab Blazor 5.0)
- 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.

5.4 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/fsboler/Bolero>], die außerhalb von Microsoft entwickelt wird und daher kein offizielles Produkt von Microsoft ist.

5.5 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

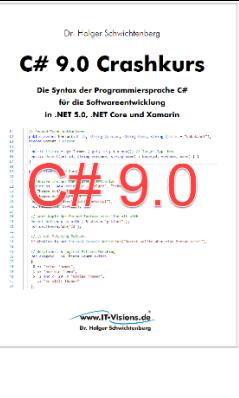
- ASP.NET Core
- C#
- Entity Framework Core
- Visual Studio

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

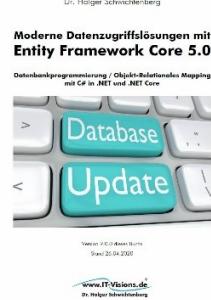
Für die ersten fünf Punkte ist dies:

	<p>Moderne Webanwendungen für .NET-Entwickler: Server-Anwendungen, Web APIs, SPAs & HTML-Cross-Platform-Anwendungen mit ASP.NET, ASP.NET Core, JavaScript/TypeScript und Angular</p> <p>ISBN 3960090153</p> <p>https://www.amazon.de/exec/obidos/ASIN/3960090153/itvisions21</p> <ul style="list-style-type: none"> ▪ Gedruckt: 49,90 Euro ▪ Kindle: 39,99 Euro
---	--

Wenn Sie C# lernen wollen, möchte ich Ihnen mein Buch "C# 9.0 Crashkurs" empfehlen:

	<p>C# 9.0 Crashkurs</p> <ul style="list-style-type: none"> ▪ Gedruckt (Print-on-Demand) bei Amazon.de für 24,99 Euro (der Autor erhält 10,52 Euro): www.amazon.de/exec/obidos/ASIN/3934279406/itvisions-21 ▪ Kindle-E-Book bei Amazon.de für 9,99 Euro (der Autor erhält 6,29 Euro): www.amazon.de/exec/obidos/ASIN/B08KPLNBZS/itvisions-21 ▪ PDF-E-Book bei Leanpub.com ab 11,99 Dollar (der Autor erhält ca. 14,00 Euro): www.leanpub.com/CSharp9
--	---

Um Entity Framework Core zu lernen, möchte ich Ihnen mein Buch "Moderne Datenzugriffslösungen mit Entity Framework Core 5.0" empfehlen:

	<p>Moderne Datenzugriffslösungen mit Entity Framework Core 5.0</p> <ul style="list-style-type: none"> ▪ 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
---	--

5.6 Ihre Belohnung, wenn Sie helfen, dieses Buch zu verbessern!

Wenn Sie Fehler in diesem Buch finden, bin ich Ihnen nicht nur wirklich sehr dankbar, sondern Sie bekommen auch eine Belohnung in Form von aktualisierten oder weiteren E-Books.

Fehlerart	E-Book-Guthaben
Inhaltlicher Fehler	Pro Fehler 20 Euro
Sprachlicher Fehler	Pro Fehler 4 Euro

Ein Beispiel: Wenn Sie einen inhaltlichen Fehler und fünf Rechtschreibfehler in diesem Buch finden, dann haben Sie bei mir 45 Euro gut. Dafür können Sie dann eins meiner selbstverlegten Bücher als E-Book bekommen.

Die selbstverlegten Bücher finden Sie unter www.IT-Visions.de/Verlag

Melden Sie die Fehler unter www.dotnet-doktor.de/Leserfeedback

Schreiben Sie dabei, welches E-Book Sie wünschen. Das Buch schicke ich Ihnen dann per E-Mail zu.

Tipp: Auch Fehler auf meiner persönlichen Website www.dotnet-doktor.de und der Firmenwebsite www.IT-Visions.de zählen mit!

Ich freue mich auf Ihre Fehlermeldung!

Holger Schwichtenberg

P.S. Falls Sie Ihre Fehlermeldung sich auf eine Ausgabe des Buchs bezieht, die älter als ein Jahr ist und der Fehler in der aktuellsten Ausgabe schon behoben ist, dann zählt das leider nicht.

6 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.

6.1 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 **Picard** 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.

6.2 Übersicht über die Beispiele

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

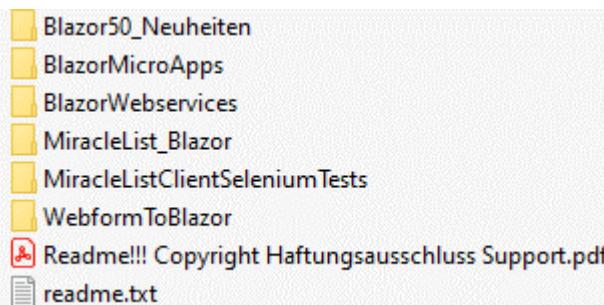


Abbildung: Inhalt des Download-Pakets zu diesem Buch

Es war eine bewusste Entscheidung, in diesem Buch nicht für jedes einzelne Beispiel ein eigenes Blazor-Projekt zu schaffen, sondern die meisten 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, praxisnahe 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 Fallbeispiels "MiracleList"

/src/MiracleList_CSBlazor	Alle	Blazor WebAssembly-Beispiele einschließlich der Blazor WebAssembly-Implementierung des Fallbeispiels "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/SamplesRCL	Alle	Ergänzende Beispiele aus dem Buch, die nicht in das MiracleList-Szenario fallen. Dieses Projekt wird im Fallbeispiel von Blazor Server und Blazor WebAssembly gemeinsam verwendet. Diese Beispiele sind im Menü "Blazor-Beispiele außerhalb der MiracleList" aufrufbar,
/src/MiracleList_Backend	Fallbeispiel	Backend (ASP.NET Core WebAPI und ASP.NET Razor Pages)
/Blazor60_Neuheiten	Alle	Für neue Funktionen in Blazor 6.0 gibt es derzeit ein eigenes Beispielprojekt NET6BW.csproj mit neuen Funktionen in Blazor 6.0 (z.B. DynamicComponent und ErrorBoundaries).
/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.

6.3 Eingesetztes CSS-Framework

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

6.4 Das Fallbeispiel "MiracleList"

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

6.4.1 Szenario

Das Fallbeispiel ist sehr praxisnah, weil das Anwendungsszenario ein Nachbau einer existierenden, sehr erfolgreichen Anwendung ist.

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. Es gibt drei Implementierungen: eine für Blazor Server, eine für Blazor WebAssembly und eine mit Angular und TypeScript. In diesem Buch werden nur die ersten beiden Varianten besprochen.



Abbildung: Das MiracleList-Logo

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.

6.4.2 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.

6.4.3 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>
- Frontend mit Angular und TypeScript (Webanwendung und Cross-Platform-Desktop-Clients für Windows, Linux und macOS):
<https://miraclelist.azurewebsites.net>
- Backend mit ASP.NET Core für Blazor WebAssembly- und Angular-Clients:
<https://miraclelistbackend.azurewebsites.net>

6.4.4 Projektmappenaufbau

Die MiracleList-Projektmappe (MiracleList.sln) besteht aus zahlreichen Projekten, die logisch in fünf Ordner 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 ein Diagramm der Geschäftsobjektklassen

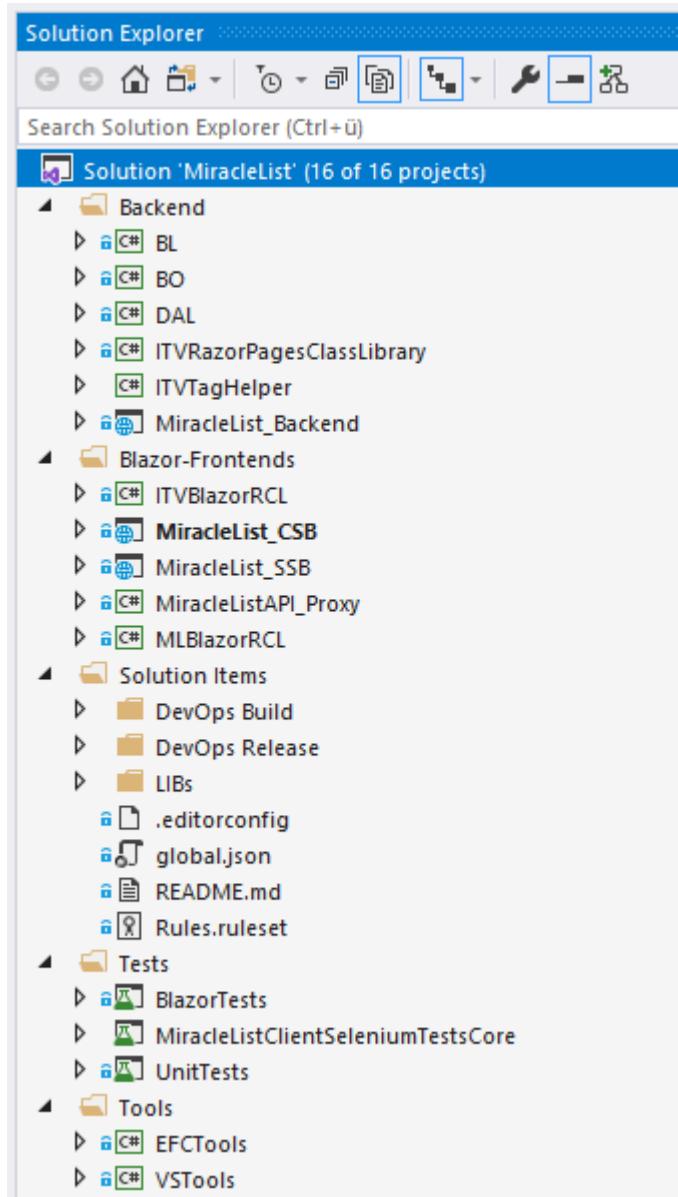


Abbildung: Aufbau der Projektmappe

6.4.5 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.

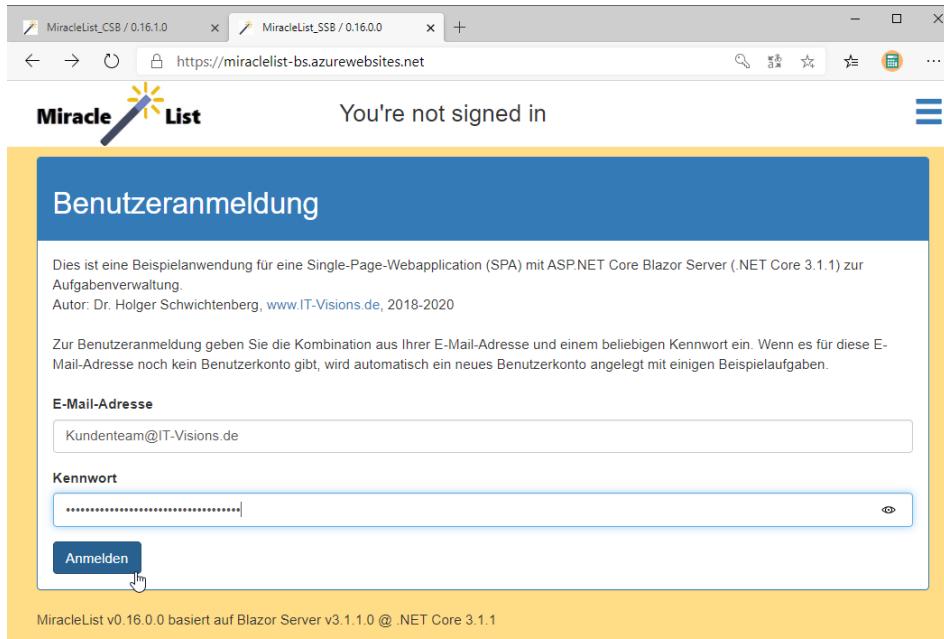
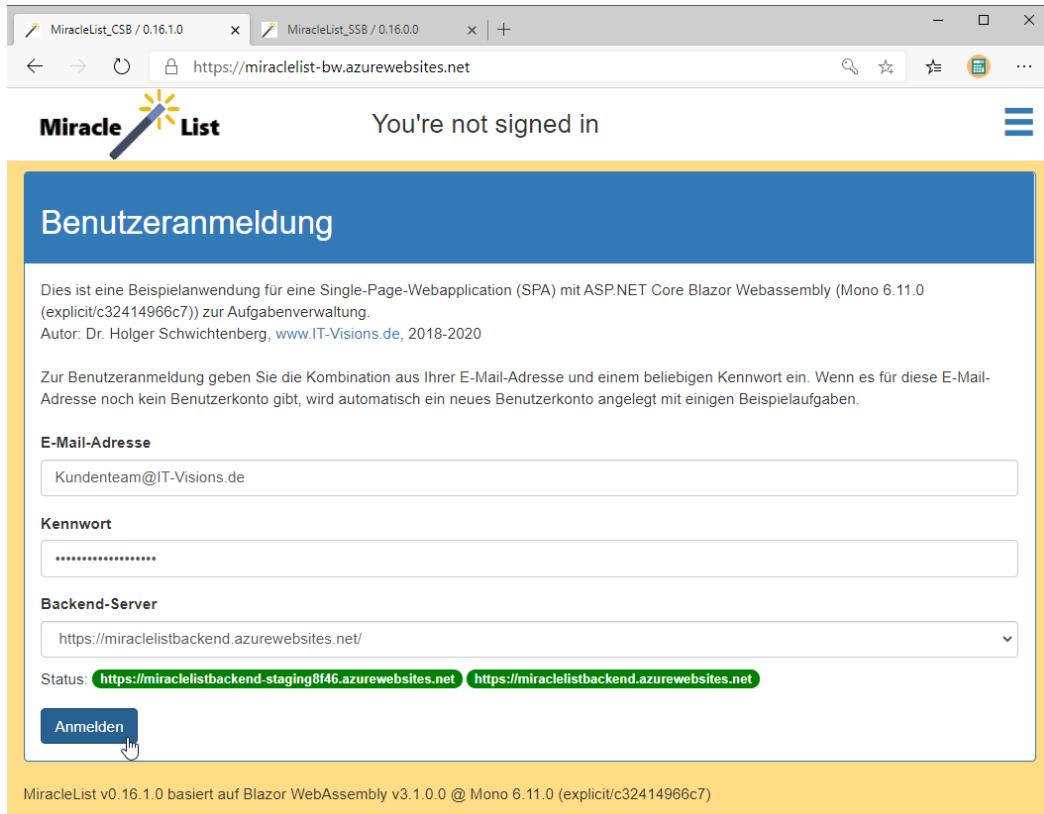


Abbildung: Anmeldeseite bei der Blazor Server-Implementierung des MiracleList-Frontends

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.



Dies ist eine Beispielanwendung für eine Single-Page-Webapplication (SPA) mit ASP.NET Core Blazor Webassembly (Mono 6.11.0 (explicit/c32414966c7)) zur Aufgabenverwaltung.
Autor: Dr. Holger Schwichtenberg, www.IT-Visions.de, 2018-2020

Zur Benutzeranmeldung geben Sie die Kombination aus Ihrer E-Mail-Adresse und einem beliebigen Kennwort ein. Wenn es für diese E-Mail-Adresse noch kein Benutzerkonto gibt, wird automatisch ein neues Benutzerkonto angelegt mit einigen Beispielaufgaben.

E-Mail-Adresse
Kundenteam@IT-Visions.de

Kennwort

Backend-Server
https://miraclelistbackend.azurewebsites.net/

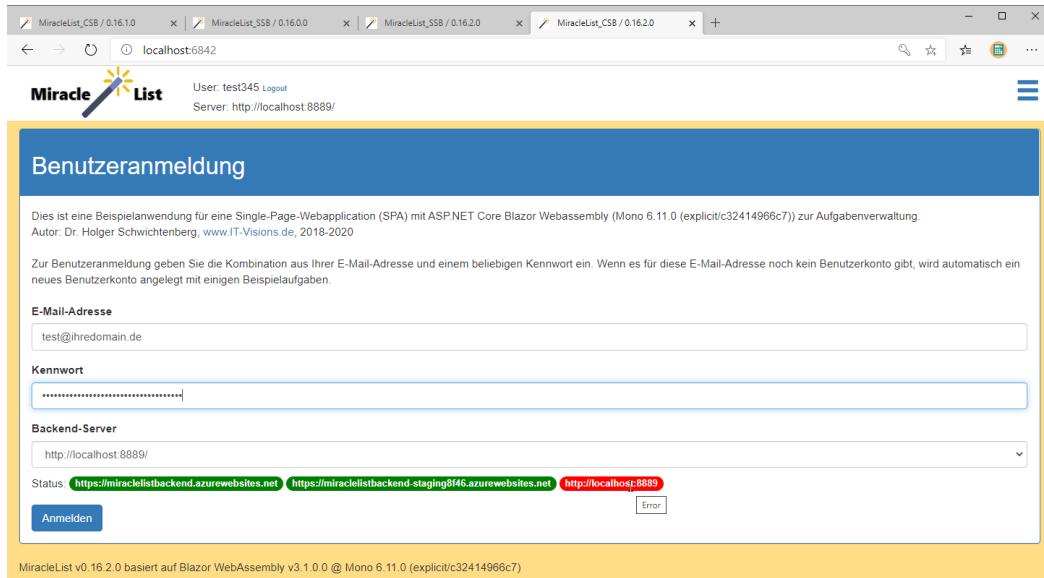
Status: <https://miraclelistbackend-staging8f46.azurewebsites.net> <https://miraclelistbackend.azurewebsites.net>

Anmelden

MiracleList v0.16.1.0 basiert auf Blazor WebAssembly v3.1.0.0 @ Mono 6.11.0 (explicit/c32414966c7)

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!



Dies ist eine Beispielanwendung für eine Single-Page-Webapplication (SPA) mit ASP.NET Core Blazor Webassembly (Mono 6.11.0 (explicit/c32414966c7)) zur Aufgabenverwaltung.
Autor: Dr. Holger Schwichtenberg, www.IT-Visions.de, 2018-2020

Zur Benutzeranmeldung geben Sie die Kombination aus Ihrer E-Mail-Adresse und einem beliebigen Kennwort ein. Wenn es für diese E-Mail-Adresse noch kein Benutzerkonto gibt, wird automatisch ein neues Benutzerkonto angelegt mit einigen Beispielaufgaben.

E-Mail-Adresse
test@ihredomain.de

Kennwort

Backend-Server
http://localhost:8889/

Status: <https://miraclelistbackend.azurewebsites.net> <https://miraclelistbackend-staging8f46.azurewebsites.net> <http://localhost:8889>

Anmelden

MiracleList v0.16.2.0 basiert auf Blazor WebAssembly v3.1.0.0 @ Mono 6.11.0 (explicit/c32414966c7)

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 sowie 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.

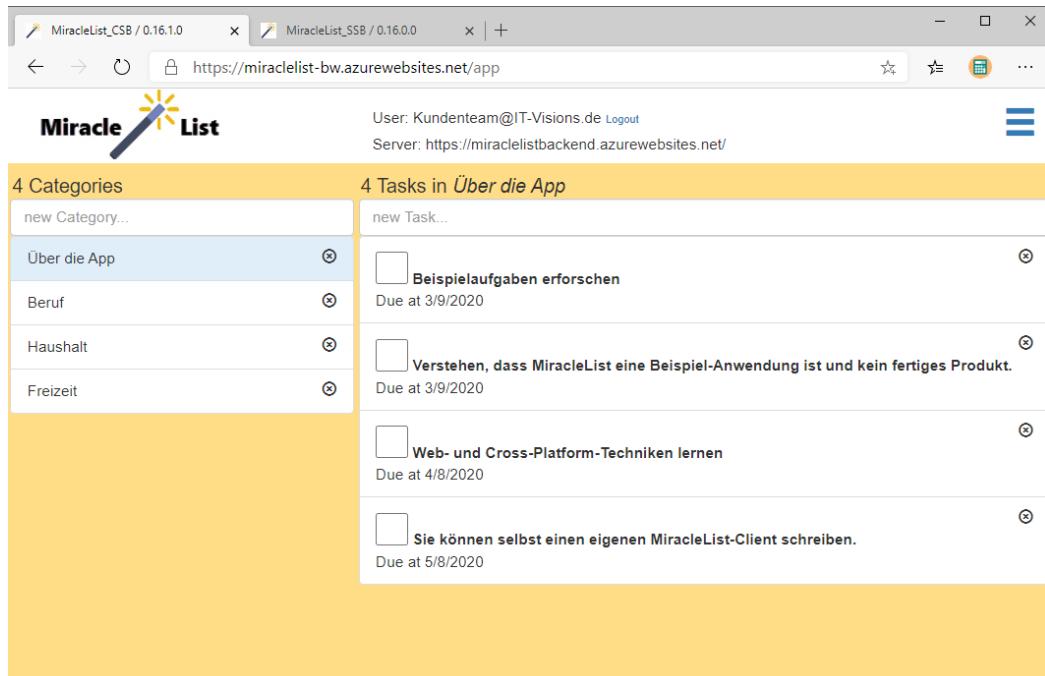


Abbildung: Hauptansicht der MiracleList-Webanwendung

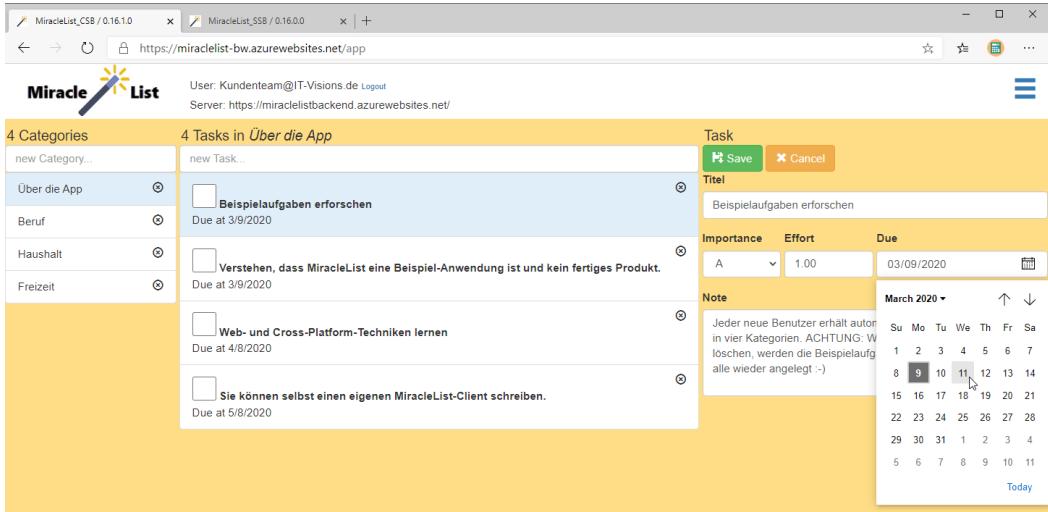


Abbildung: Bearbeitungsansicht der MiracleList-Webanwendung

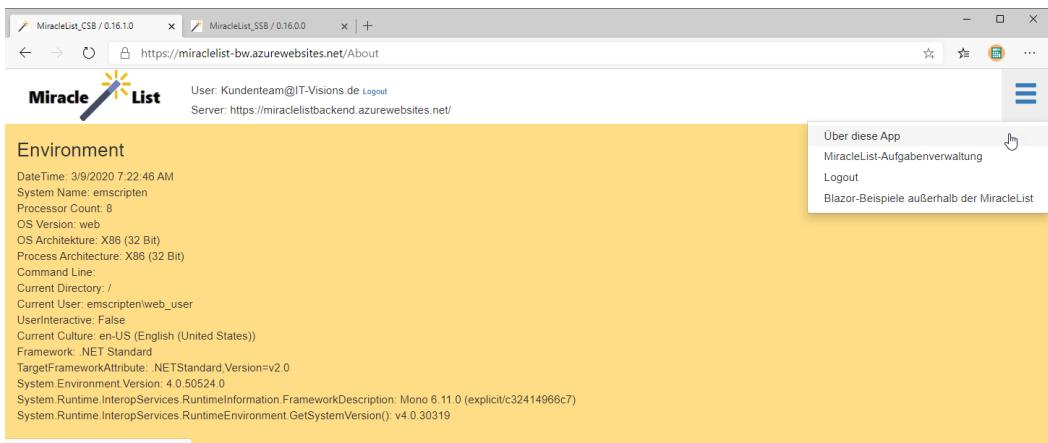


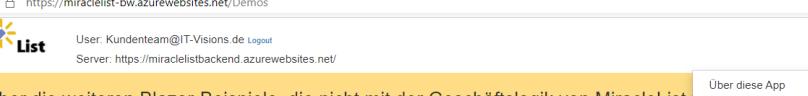
Abbildung: Technische Daten sieht man im Menü "Über diese App"

6.5 Weitere Beispiele innerhalb der MiracleList-Webanwendung

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 in 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.



Übersicht über die weiteren Blazor-Beispiele, die nicht mit der Geschäftslogik von MiracleList

- Kapitel 9: Code-Behind mit Vererbung
- Kapitel 9: Code-Behind mit partieller Klasse
- Kapitel 9: Komponenten ohne Razor nur aus Programmcode
- Kapitel 10: Routing und Navigation
- Kapitel 11: Razor-Syntax
- Kapitel 11: Razor-Listen
- Kapitel 12: Razor-Ereignisse
- Kapitel 13: Komponenteneinbettung
- Kapitel 14: Komponentenzustand
- Kapitel 15: Einfache Formulare
- Kapitel 16: Fehlerbehandlung
- Kapitel 16: Verwendung einer RCL
- Kapitel 17: Dependency Injection
- Kapitel 19: JavaScript Interop
- Kapitel 20: Systeminformationen
- Kapitel 23: Countdown und Fortschrittsanzeige
- Performancemessungen

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/Samples. 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.

6.6 Visual Studio 2019

Für die Entwicklung von Blazor-Anwendungen 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-Anwendungen ist (für Einzelpersonen und Organisation bis fünf Entwickler) die 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	Visual Studio Professional	Visual Studio Enterprise
	Kostenloser Download	Kaufen	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.

Visual Studio Installer

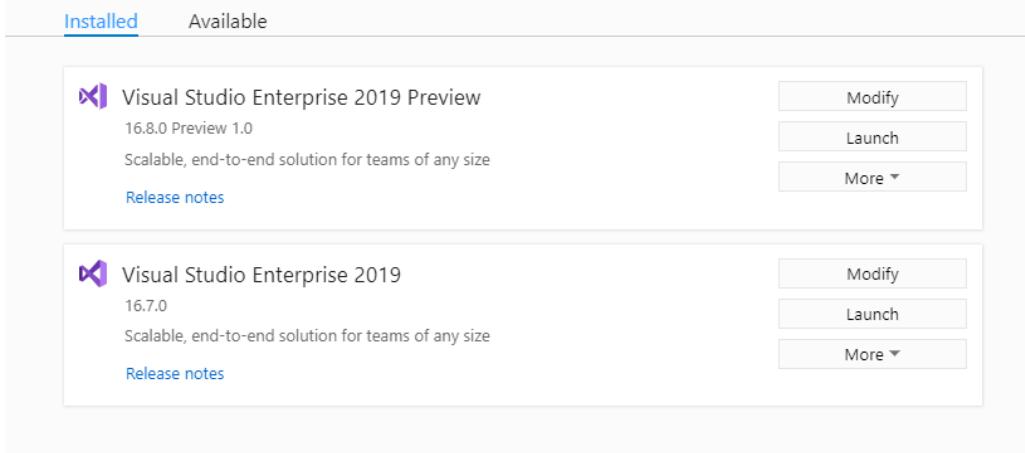


Abbildung: Parallelinstallation der stabilen Version 16.7 und der Preview 1 von 16.8

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.

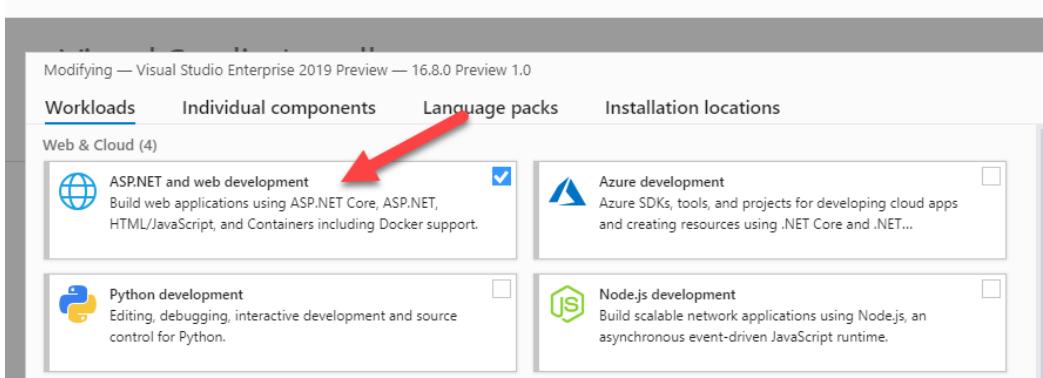


Abbildung: Notwendige Workloads für ASP.NET Core Blazor

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.

6.7 .NET SDK

Ein typisches Problem, das 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.

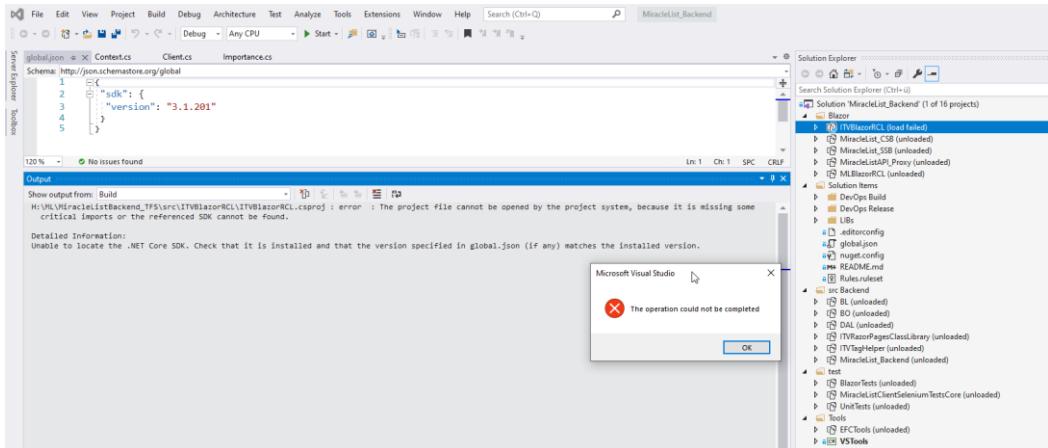


Abbildung: Die notwendige SDK-Version fehlt

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

- Im Standard wird die aktuellste .NET SDK-Version verwendet (einschließlich Preview-Versionen). Dies kann zu unerwünschten Effekten beim Kompilieren führen, z.B. aufgrund von 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`

```
PS T:\> dotnet --list-sdks
2.1.803 [C:\Program Files\dotnet\sdk]
3.1.102 [C:\Program Files\dotnet\sdk]
3.1.201 [C:\Program Files\dotnet\sdk]
3.1.402 [C:\Program Files\dotnet\sdk]
5.0.100-preview.8.20417.9 [C:\Program Files\dotnet\sdk]
5.0.101 [C:\Program Files\dotnet\sdk]
5.0.200-preview.20601.7 [C:\Program Files\dotnet\sdk]
PS T:\> |
```

Abbildung: Liste der auf einem 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.

6.8 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 Server" oder "Blazor WebAssembly" im Dialog "File/New Project" in Visual Studio.

Tipp: Geben Sie im Suchdialog "Blazor" ein, siehe Abbildung.

Wählen Sie hier zum Test "Blazor WebAssembly". Nach der Auswahl des Namens (geben Sie ein "MiracleListBW" für das Projekt und "MiracleList" für die Projektmappe) und des von Ihnen wählbaren Zielpfades (wie immer sollte man einen Dateisystempfad ohne Leerzeichen wählen, das macht die Verwendung von Kommandozeilenwerkzeugen einfacher), kommt ein weiterer Dialog. Hier ist ".NET 5.0", "No Authentication", "Configure for HTTPS" und "Progressive Web App" zu wählen.

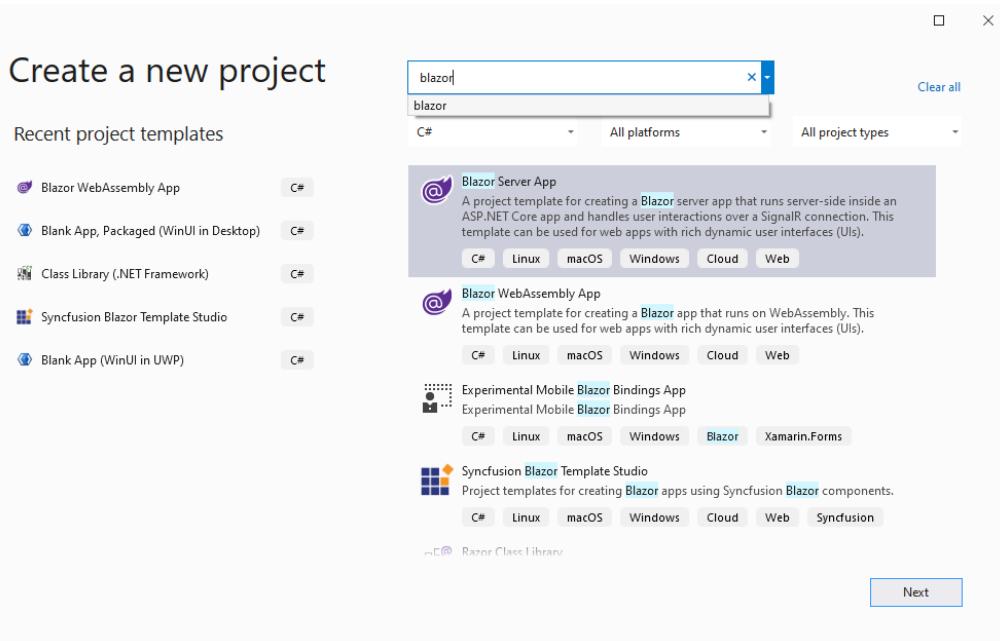


Abbildung: Wahl der Projektvorlage

Configure your new project

Blazor WebAssembly App C# Linux macOS Windows Cloud Web

Project name

Location

Solution

Solution name ?

Place solution and project in the same directory



Abbildung: Einstellungen für die neue Blazor WebAssembly-Anwendung

Additional information

Blazor WebAssembly App C# Linux macOS Windows Cloud Web

Target Framework

Authentication Type

Configure for HTTPS
 ASP.NET Core hosted
 Progressive Web Application



Abbildung: Weitere 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 die 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).

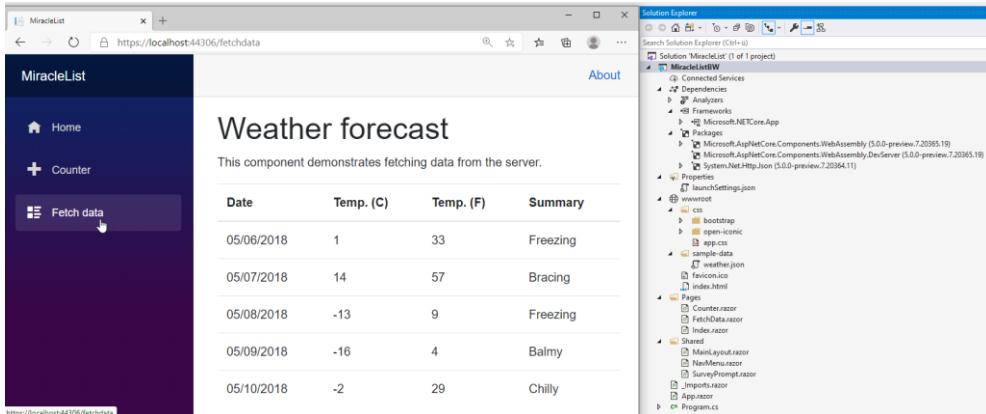


Abbildung: Beispielanwendung von Microsoft

Hinweis: Es gibt leider mannigfaltige Gründe, warum es auf Ihrem System zu einem Fehler kommt, den der Autor dieses Buch nicht hatte. 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/Support.

6.9 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".

6.10 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.

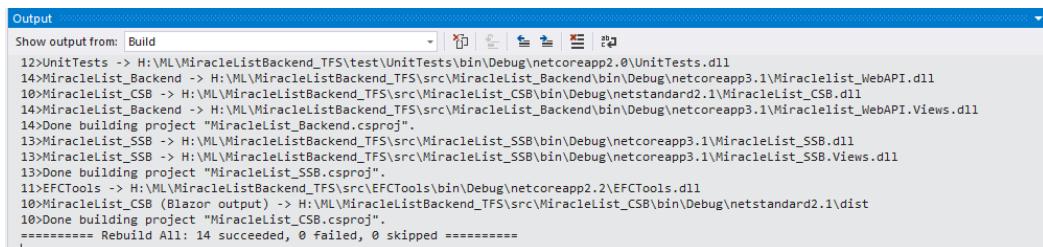


Abbildung: Beweis, dass alle Projekte in Visual Studio fehlerfrei übersetzt

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.

Falls dennoch Beispiele bei Ihnen nicht laufen, kontaktieren Sie mich bitte unter

www.dotnet-doktor.de/Leserfeedback

mit einer sehr genauen Fehlerbeschreibung und Ihrer genauen Systemkonfiguration. Ich bemühe mich, Ihnen binnen zwei Wochen zu antworten. Im Einzelfall kann es wegen dienstlicher oder privater Abwesenheit aber auch länger dauern.

7 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).

7.1 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 Unterschieden beider Modelle werden Sie in weiteren Unterkapiteln erfahren.