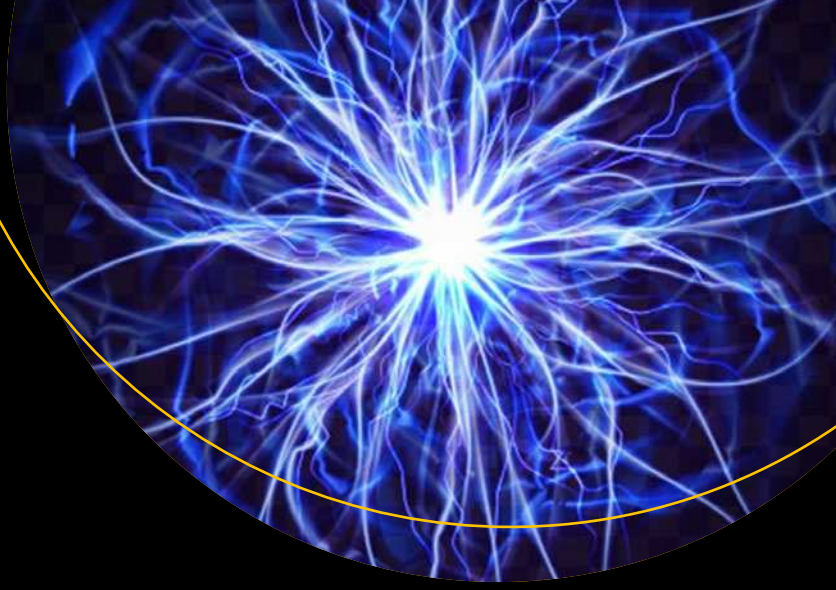

Microsoft Blazor

Building Web Applications in .NET

Second Edition

Peter Himschoot

Apress®

Microsoft Blazor

Building Web Applications in .NET

Second Edition

Peter Himschoot

Apress®

Microsoft Blazor: Building Web Applications in .NET

Peter Himschoot
Melle, Belgium

ISBN-13 (pbk): 978-1-4842-5927-6 ISBN-13 (electronic): 978-1-4842-5928-3
https://doi.org/10.1007/978-1-4842-5928-3

Copyright © 2020 by Peter Himschoot

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Jonathan Gennick
Development Editor: Laura Berendson
Coordinating Editor: Jill Balzano

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484259276. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

Table of Contents

About the Author

Peter Himschoot works as a lead trainer, architect, and strategist at U2U Training. Peter has a wide interest in software development, which includes applications for the Web, Windows, and mobile devices. Peter has trained thousands of developers, is a regular speaker at international conferences, and has been involved in many web and mobile development projects as a software architect. Peter has been a Microsoft Regional Director from 2003 to 2019, a group of trusted advisors to the developer and IT professional audiences, and to Microsoft. He can be reached on Twitter @peterhimschoot.

About the Technical Reviewer

Gerald Versluis (@jfversluis) is a software engineer at Microsoft from the Netherlands. With years of experience working with Xamarin, Azure, ASP.NET, and other .NET technologies, he has been involved in a number of different projects and has been building several real-world apps and solutions.

Not only does he like to code, but he is also passionate about spreading his knowledge, as well as gaining some in the bargain. Gerald involves himself in speaking, providing training sessions and writing blogs (https://blog.verslu.is) or articles, live coding, and contributing to open source projects in his spare time. He can be reached on Twitter @jfversluis and his website https://gerald.verslu.is.

Acknowledgments

When Jonathan Gennick from Apress asked me if I would be interested in writing a book on Blazor, I felt honored and of course I agreed that Blazor deserves a book. Writing a book is a group effort, so I thank Jonathan Gennick and Jill Balzano for giving me tips on styling and writing this book, and I thank Gerald Versluis for doing the technical review and pointing out sections that needed a bit more explaining. I also thank Magda Thielman and Lieven Iliano from U2U, my employer, for encouraging me to write this book.

I thoroughly enjoyed writing this book, and I hope you will enjoy reading and learning from it.

Second Edition

As the first edition of *Blazor Revealed* was published (using pre-release software), the Blazor team had made a bunch of changes to the razor syntax, stopping my examples in *Blazor Revealed* from working. Now that Blazor has been released and is completely official (YEAH!!!!), the time has come to publish an updated version of *Blazor Revealed*.

Should you get stuck with an example, I invite you to consult the accompanying code samples for comparison purposes.

Introduction

Back in 2018, I was attending the *Microsoft Most Valued Professional and Regional Directors Summit* where we were introduced to Blazor for the first time by *Steve Sanderson* and *Daniel Roth*. And I must admit, I was super excited about Blazor! We learned that Blazor is a framework that allows you to build single-page applications using C# and allows you to run any standard .NET library in the browser. Before Blazor, your options for building a SPA were Angular, React, Vue.js, and others using JavaScript or one of the other higher-level languages like TypeScript (which get compiled into JavaScript anyway). I was so excited then that I ended up writing the first edition of this book, and now I have updated it for you.

In this introduction, I will show you how browsers are now capable of running .NET assemblies in the browser using WebAssembly, Mono, and Blazor.

A Tale of Two Wars

Think about it. The browser is one of the primary applications on your computer. You use it every day. Companies who build browsers know that very well and are bidding for you to use their browser. At the beginning of mainstream Internet, everyone was using *Netscape*, and Microsoft wanted a share of the market, so in 1995 they built *Internet Explorer 1.0*, released as part of Windows 95 Plus! pack. Newer versions were released rapidly, and browsers started to add new features such as `<blink>` and `<marquee>` elements. This was the beginning of the first browser war, giving people (especially designers) headaches because some developers were building pages with blinking marquee controls 😊. But developers were also getting sore heads because of incompatibilities between browsers. *The first browser war was about having more HTML capabilities than the competition.*

But all of this is now behind us with the introduction of HTML5 and modern browsers like Google Chrome, Microsoft Edge, Firefox, Safari, and Opera. HTML5 not only defines a series of standard HTML elements but also rules on how these should render, making it a lot easier to build a website that looks the same in all modern browsers. Then, in 1995 *Brendan Eich* wrote a little programming language known as *JavaScript* (initially called *LiveScript*) in 10 days (What!?). It was called JavaScript because its syntax was very similar to Java.

JavaScript and Java are not related. Java and JavaScript have as much in common as ham and hamster (I don't know who formulated this first, but I love this phrasing).

Little did Mr. Eich know how this language would impact the modern web and even desktop application development. In 1995 *Jesse James Garett* wrote a white paper called *Ajax (Asynchronous JavaScript and XML)*, describing a set of technologies where JavaScript is used to load data from the server and that data is used to update the browser's HTML. This avoids full-page reloads and allows for client-side web applications, which are written in JavaScript that runs completely in the browser. One of the first companies to apply Ajax was Microsoft when they built *Outlook Web Access (OWA)*. OWA is a web application almost identical to the Outlook desktop application proving the power of Ajax. Soon other Ajax applications started to appear, with Google Maps stuck in my memory as one of the other keystone applications. Google Maps would download maps asynchronously and with some simple mouse interactions allowed you to zoom and pan the map. Before Google Maps, the server would do the map rendering and a browser displayed the map like any other image by downloading a bitmap from a server.

Building an Ajax website was a major undertaking that only big companies like Microsoft and Google could afford. This soon changed with the introduction of JavaScript libraries like jQuery and knockout.js (Knockout was also written by Steve Sanderson, the author of Blazor!). Today we build rich web apps with Angular, React, and Vue.js. All of them are using JavaScript or higher-level languages like TypeScript which get transpiled into JavaScript.

Transpiling will take one language and convert it into another language. This is very popular with TypeScript which gives you a modern high-level language. You need JavaScript to run it in a browser, so TypeScript gets "transpiled" into JavaScript.

This brings us back to JavaScript and the second browser war. JavaScript performance is paramount in modern browsers. Chrome, Edge, Firefox, Safari, and Opera are all competing with one another, trying to convince users that their browser is the fastest with cool-sounding names for their JavaScript engine like *V8* and *Chakra*. These engines use the latest optimization tricks like JIT compilation where JavaScript gets converted into native code as illustrated in Figure 1.

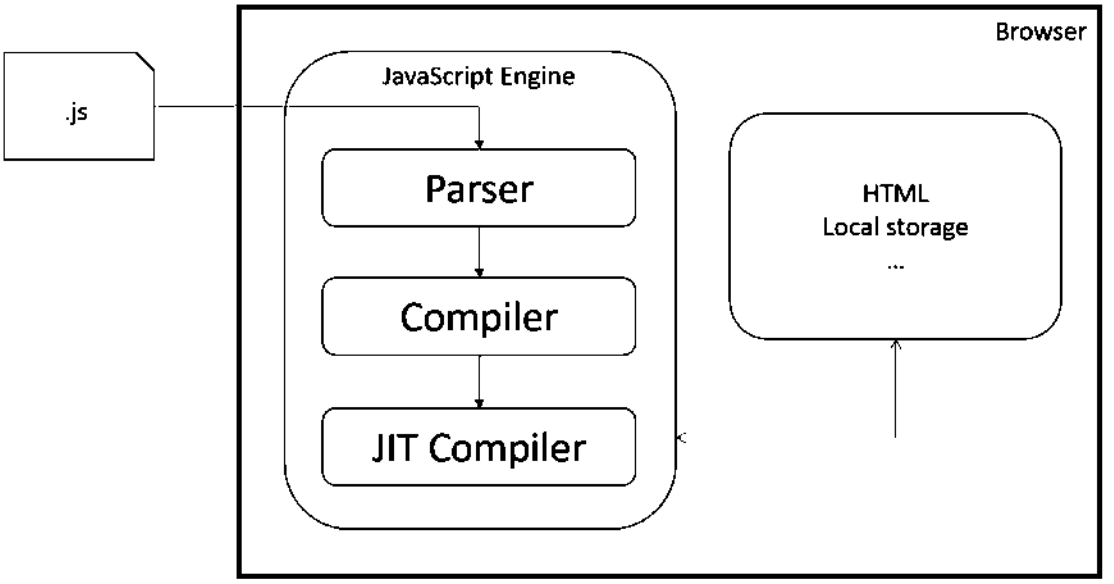

Figure 1. *The JavaScript execution process*

This process takes a lot of effort because JavaScript needs to be downloaded into the browser, where it gets parsed, then compiled into bytecode and then Just-In-Time converted into native code. So how can we make this process even faster?

The second browser war is all about JavaScript performance.

Introducing WebAssembly

WebAssembly allows you to take the parsing and compiling to the server. With WebAssembly, you compile your code in a format called WASM (an abbreviation of WebASseMbly), which gets downloaded by the browser where it gets Just-In-Time compiled into native code as in Figure 2. Open your browser and google "*webassembly demo zen garden.*" One of the links should take you to `https://s3.amazonaws.com/mozilla-games/ZenGarden/EpicZenGarden.html` where you can see an impressive ray-trace demo of a Japanese Zen garden with a screenshot in Figure 3.

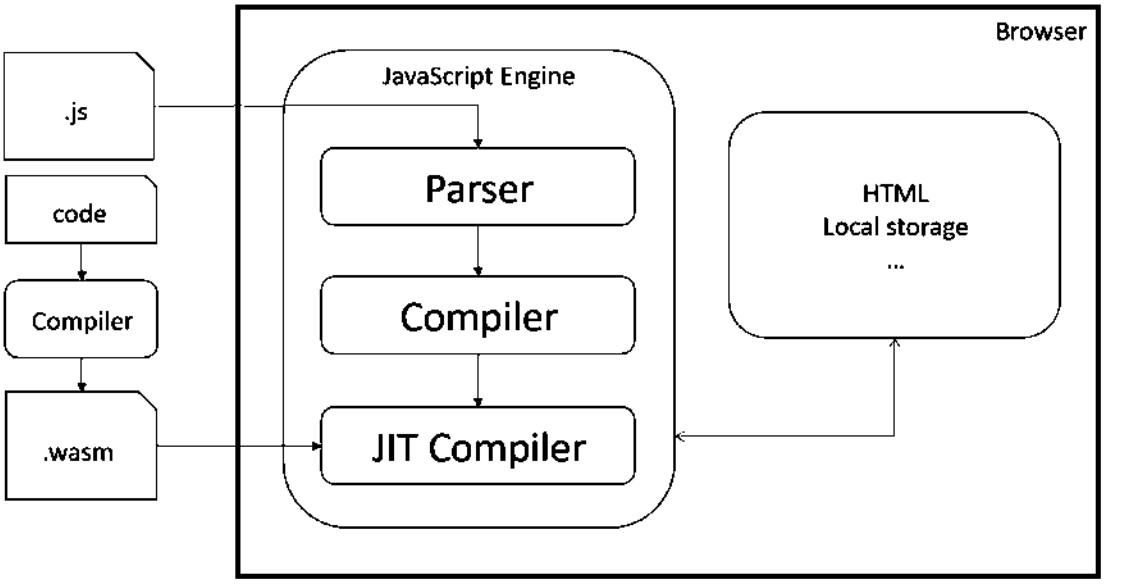

Figure 2. *The WebAssembly execution process*

Figure 3. *Japanese Zen garden*

From the official site webassembly.org

> *WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications.*

So WebAssembly as a new binary format optimized for browser execution, it is NOT JavaScript. There are compilers for languages like C++ and Rust which compile to WASM. Some people have compiled C++ applications to wasm, allowing to run them in the browser. There is even a Windows 2000 operating system compiled to wasm!

Which Browsers Support WebAssembly?

WebAssembly is supported by all major browsers: Chrome, Edge, Safari, Opera, and Firefox, including their mobile versions. As WebAssembly will become more and more important, we will see other modern browsers follow suit, but don't expect Internet Explorer to support WASM. You can check for this on `https://caniuse.com/#search=wasm`.

WebAssembly and Mono

Mono is an open source implementation of the .NET CLI specification, meaning that Mono is a platform for running .NET assemblies. Mono is used in *Xamarin* for building mobile applications that run on Windows, Android, and iOS mobile operating systems. You can also use it to build applications for macOS, Linux, Tizen, and others. Mono also allows you to run .NET on Linux (its original purpose) and is written in C++. This last part is important because we saw that you can compile C++ to WebAssembly. So, what happened is that the Mono team decided to try to compile Mono to WebAssembly, which they did successfully. There are two approaches. One is where you take your .NET code and you compile it together with the Mono runtime into one big WASM application. However, this approach takes a lot of time because you need to take several steps to compile everything into WASM, not so practical for day-to-day development. The other approach takes the Mono runtime and compiles it into WASM, and this runs in the browser where it will execute .NET Intermediate Language just like normal .NET does. The big advantage is that you can simply run .NET assemblies without having to compile them first into WASM. This is the approach currently taken by Blazor. But Blazor

is not the only one taking this approach. For example, there is the *Ooui* project which allows you to run *Xamarin.Forms* applications in the browser. The disadvantage of this is that it needs to download a lot of .NET assemblies. This can be solved by using *Tree Shaking* algorithms which remove all unused code from assemblies.

Interacting with the Browser with Blazor

WebAssembly with Mono allows you to run .NET code in the browser. *Steve Sanderson* used this to build Blazor. Blazor uses the popular ASP.NET MVC approach for building applications that run in the browser. With Blazor, you build Razor files (Blazor = Browser + Razor) which execute inside to browser to dynamically build a web page. With Blazor, you don't need JavaScript to build a web app which is good news for thousands of .NET developers who want to continue using C# (or F#).

How Does It Work?

Let's start with a simple razor file in Listing 1 which you can find when you create a new Blazor project (which we will do in the first chapter).

Listing 1. The `Counter` razor file

```
@page "/counter"

<h1>Counter</h1>

<p>Current count: @currentCount</p>

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

@code {
  private int currentCount = 0;

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

This file gets compiled into .NET code (you'll find out how later in this book) which is then executed by the Blazor engine. The result of this execution is a tree-like structure called the *render tree*. The render tree is then sent to JavaScript which updates the DOM to reflect the render tree (creating, updating, and removing HTML elements and attributes). Listing 1 will result in `h1`, `p` (with the value of `currentCount`), and `button` HTML elements. When you interact with the page, for example, when you click the button, this will trigger the button's click event which will invoke the `IncrementCount` method from Listing 1. The render tree is then regenerated, and any changes are sent again to JavaScript which will update the DOM. This process is illustrated in Figure 4.

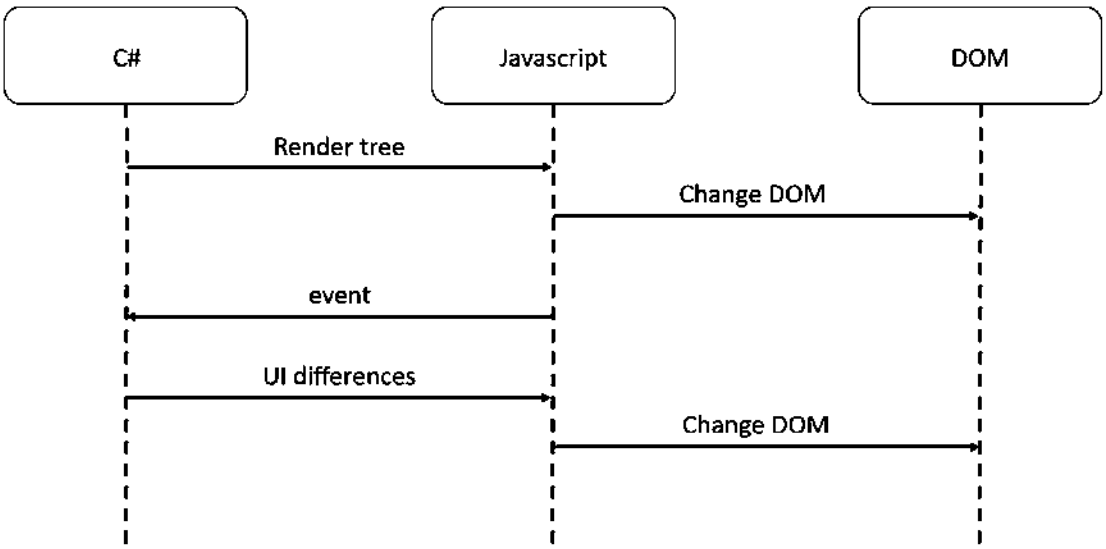

Figure 4. *The Blazor WebAssembly DOM generation process*

This model is very flexible. It allows you to build *Progressive Web Apps* and also can be embedded in *Electron* desktop applications, of which Visual Studio Code is a prime example.

In Chapter 1, section "The Client-Side Blazor Bootstrap Process," we will look at which files get downloaded. One of the drawbacks of Blazor WebAssembly is that this is a substantial download on first use (after this, most files can be cached by the browser or the Blazor runtime itself), especially the Mono runtime itself. If you want to avoid this big download, you can use Server-Side Blazor.