Spis treści

Przedmowa

Tworzenie aplikacji internetowych zdominowało branżę oprogramowania od ponad 20 lat i prawdopodobnie nie zmieni się to przez wiele kolejnych lat. Giganci tej branży nadal intensywnie inwestują w zwiększanie mocy i elastyczności technologii internetowych, umożliwiając tworzenie coraz szerszej gamy zaawansowanego oprogramowania opartego na przeglądarkach internetowych. Podczas gdy natywne aplikacje mobilne i aplikacje rzeczywistości rozszerzonej / wirtualnej znajdują swoje miejsce wśród oprogramowania konsumenckiego, sieć WWW jest w przeważającej mierze domyślnym interfejsem użytkownika dla aplikacji biznesowych. Jeśli ktoś mógłby postawić tylko na jedną platformę aplikacji, powinien stawiać na aplikacje internetowe.

W ciągu tych samych 20 lat platforma .NET (która ukazała się po raz pierwszy w 2002 roku) utrzymała swoje miejsce jako najważniejszy zestaw narzędzi programistycznych firmy Microsoft. Podobnie jak sieć WWW, platforma .NET nadal zyskuje na sile. Została przebudowana od nowa w 2016 roku jako wieloplatformowa oraz w pełni otwarta platforma bazująca na chmurze i jest obecnie używana przez około 30% wszystkich profesjonalnych programistów[1]. Język C# zawsze był uważany za jeden z najbardziej wydajnych języków, zajmując miejsce w czołówce bogatych narzędzi programistycznych z technikami precyzyjnego uzupełniania kodu i najlepszymi narzędziami do debugowania, a obecnie ASP.NET Core jest jedną z najszybszych technologii internetowych po stronie serwera[2].

Celem technologii Blazor jest uwolnienie pełnej mocy platformy .NET dla aplikacji z interfejsem użytkownika opartym na przeglądarce. Zespół opracowujący .NET skupia swoje wysiłki na tworzeniu najbardziej wydajnych i naturalnych sposobów tworzenia aplikacji jednostronicowych. Obejmuje to model programowania Blazor oparty na składnikach, który bierze najlepsze aspekty z wielu nowoczesnych platform interfejsu użytkownika i osadza je w sposób naturalny w silnie typowanej platformie .NET. Poza tym oznacza

1 „Najpopularniejsze technologie" – ankieta serwisu Stack Overflow wśród programistów z 2022 roku, wyniki na dzień 18 sierpnia 2022, *https://oreil.ly/7UTEc*.

2 „Pomiary wydajności platform internetowych, runda 21, 19 lipca 2022", TechEmpower, *https://oreil.ly/EBawf*.

to połączenie z resztą ekosystemu .NET z wiodącymi w branży zintegrowanymi środowiskami programistycznymi i pierwszorzędnymi funkcjami pozwalającymi na debugowanie, testowanie i ponowne ładowanie uruchamianego kodu. Największą innowacją technologii Blazor mogą być elastyczne modele wykonywania programu, uruchamiające program po stronie serwera z przesyłaniem strumieniowym interfejsu użytkownika do przeglądarek przez gniazdo websocket, bezpośrednio w przeglądarce pod kontrolą WebAssembly lub jako kod natywny w aplikacjach mobilnych i desktopowych.

Poznaj Blazor zapewnia zarówno głębokie, jak i szerokie spojrzenie na programowanie aplikacji Blazor. W przeciwieństwie do wielu innych książek, nie koncentruje się tylko na łatwych częściach programowania w języku C# bez pokazywania złożoności świata rzeczywistego. Zamiast tego David przedstawia od samego początku cały zakres zagadnień związanych z tworzeniem aplikacji internetowych – w tym uwierzytelnianie, bezpieczeństwo, wydajność, lokalizację i wdrażanie (CI/CD). Przy odrobinie skupienia można łatwo przyswoić szeroką wiedzę Davida i być przygotowanym do zastosowania jej w rzeczywistych, własnych zadaniach.

David jest dobrze przygotowany, aby wyjaśniać nie tylko, jak rzeczy działają dzisiaj, ale także jak ewoluowały do obecnego stanu, a nawet jak mogą się zmienić w przyszłości. Od lat jest dobrze znaną postacią w społeczności Blazor, jest dobrze powiązany z liderami technologicznymi w firmie Microsoft i ma jeszcze dłuższą historię jako posiadacz tytułów Microsoft Most Valuable Professional (MVP) i Google Developer Expert (GDE) w dziedzinie technologii internetowych. W tej książce można znaleźć wiele szczegółów historycznych i anegdot, które rzucają światło na wyzwania, decyzje i osoby, które ukształtowały tworzenie aplikacji internetowych i platformę .NET w technologie, których będziemy używać. Entuzjazm Davida poprowadzi nas przez skomplikowaną postać tych technologii.

Moją największą motywacją przy tworzeniu pierwszego wydania Blazor z Danem Rothem i Ryanem Nowakiem była chęć uwolnienia interfejsu internetowego od jego monokultury. Doceniam język JavaScript i zbudowałem na nim wiele lat swojej kariery, ale jest tak wiele innych języków programowania, paradygmatów i społeczności, które mogą wnieść własne bogactwo do przeglądarki. Jestem pewien, że każdy znajdzie swoje własne sposoby na innowacje w tworzonym przez siebie oprogramowaniu, które będą korzystne dla użytkowników. Życzę każdemu wszystkiego najlepszego w pracach nad projektami Blazor i jestem pewien, że każdy znajdzie wiele inspiracji na stronach tej książki.

— Steve Sanderson
Software Engineer/Architect w firmie Microsoft, twórca technologii Blazor
Bristol, UK
Sierpień 2022

Wstęp

Witam w książce *Poznaj Blazor*. Prawdopodobnie jesteś tutaj, ponieważ słyszałeś kilka fajnych rzeczy o technologii Blazor i chcesz ją wypróbować. Czym więc ona jest? Blazor to oparta na otwartym kodzie źródłowym platforma internetowa do tworzenia interaktywnych składników internetowego interfejsu użytkownika po stronie klienta przy użyciu języka C#, HTML i kaskadowych arkuszy stylów (CSS)[1]. Jako funkcjonalność ASP.NET Core, Blazor rozszerza platformę programistyczną .NET o narzędzia i biblioteki do tworzenia aplikacji internetowych.

WebAssembly umożliwia uruchamianie w przeglądarce programów opracowanych w wielu językach programowania innych niż JavaScript. Blazor w pełni wykorzystuje WebAssembly i pozwala programującym w języku C# budować składniki interfejsu użytkownika po stronie klienta za pomocą .NET. Blazor jest platformą do tworzenia aplikacji jednostronicowych (SPA), podobną na przykład do Angular, React, VueJS i Svelte, ale opartą na języku C# zamiast JavaScript.

Jest to zatem platforma internetowa, ale co odróżnia ją od innych platform do tworzenia webowego interfejsu użytkownika po stronie klienta?

Dlaczego Blazor?

Blazor zmienia zasady gry zarówno dla programistów .NET, jak i ogólnie dla twórców aplikacji internetowych! Z tej książki można się dowiedzieć, jak korzystać z modelu osadzania Blazor w WebAssembly przy tworzeniu atrakcyjnych doświadczeń internetowych działających w czasie rzeczywistym. Wydaje się, że istnieją niezliczone powody, aby wybrać Blazor jako następną platformę do tworzenia aplikacji internetowych. Zacznijmy od tego, jak zmienia ona tworzenie aplikacji internetowych.

Na początku lat 90. surfowanie po Internecie przypominało czytanie ciągu połączonych dokumentów tekstowych – opracowanych przy użyciu podstawowego kodu HTML.

1 „Build beautiful web apps with Blazor", Microsoft, *https://oreil.ly/iIaWE*.

Nie było to wciągające ani spójne doświadczenie. Kiedy na scenie pojawiły się technologie CSS i JavaScript, możliwość dynamicznego reagowania na interakcje użytkownika dodała znacznie więcej uroku do korzystania z WWW. Chociaż strony internetowe zaczęły wyglądać ciekawiej, to ładowały się bardzo wolno, a ludzie spodziewali się powolnego interfejsu użytkownika, z widocznym odrysowywaniem/buforowaniem stron. Całkowicie normalne było obserwowanie stopniowo pojawiających się obrazów, gdy dane obrazu były buforowane przez HTTP w drodze do przeglądarki z szybkością połączenia modemowego. Ta cierpliwość szybko się wyczerpała. W ludzkiej naturze leży pragnienie jak najszybszego dostępu do różnych rzeczy, mam rację? Jeśli ktoś musi czekać w przeglądarce dłużej niż kilka sekund, zaczyna czuć się trochę nieswojo. W miarę jak treści internetowe stawały się coraz bardziej złożone, platformy programistyczne starały się oswajać tę złożoność.

Do tych technologii należy Blazor wraz z WebAssembly. Dzięki technologii Blazor możemy wykorzystywać ten sam kod w języku C# zarówno w scenariuszach po stronie klienta, jak i po stronie serwera, a jednocześnie korzystać z narzędzi z rodziny produktów Visual Studio, niezawodnego interfejsu wiersza poleceń .NET CLI oraz innych popularnych zintegrowanych środowisk programistycznych (IDE) obsługujących .NET. Ekosystem .NET kwitnie, wykorzystanie tej platformy gwałtownie rośnie, a atrakcyjność długoterminowego wsparcia technicznego (LTS) nadal jest czynnikiem napędzającym rozwój aplikacji dla przedsiębiorstw. W porównaniu z warunkami LTS dla innych platform SPA, takich jak Angular i React, platforma .NET wyróżnia się jako zdecydowany zwycięzca. Wynika to z faktu, że pomoc techniczna dla platformy .NET jest oferowana przez trzy lata od wypuszczenia każdej wersji LTS. Bardzo korzystne jest bycie na bieżąco z każdym nowym wydaniem. Więcej informacji można uzyskać w zasadach wsparcia technicznego dla platformy .NET (*https://oreil.ly/sQE70*).

Podobnie jak każda inna aplikacja internetowa, aplikacje internetowe Blazor mogą być tworzone jako progresywne aplikacje internetowe (PWA) do obsługi scenariuszy przy niestabilnym połączeniu internetowym. Mogą być również osadzane wewnątrz natywnych aplikacji komputerowych i instalowane na urządzeniu użytkownika. Aplikacje Blazor WebAssembly mogą definiować natywne zależności, na przykład związane z językami C i C++. Wszystko, co zostało skompilowane za pomocą Emscripten (*https://emscripten.org*), może być użyte w Blazor. Moim zdaniem platforma ta nie wymaga wielu kompromisów; jest bardzo chętnie wykorzystywana i przyjemna do programowania.

Kiedy wprowadzono technologię WebAssembly, początkowo budziła ona tylko umiarkowane oczekiwania ze strony społeczności programistów. W 2017 roku platforma WebAssembly została ustandaryzowana w otwarty sposób, co pozwoliło programistom badać nowe możliwości interaktywności i funkcjonalności w przeglądarkach poza samym wykorzystaniem języka JavaScript. Jest to ważne dla twórców aplikacji internetowych, ponieważ mogą łatwiej konkurować z lukratywną platformą aplikacji App Store. Język JavaScript nadal ewoluuje, dodając funkcje wykraczające poza standard ECMAScript. Wraz ze stworzeniem platformy Blazor dla .NET, język C # stał się prawdziwym konkurentem języka JavaScript.

Jako programista z ponad dziesięcioletnim doświadczeniem w tworzeniu rzeczywistych aplikacji internetowych mogę zaświadczyć, że wielokrotnie używałem platformy .NET do tworzenia aplikacji produkcyjnych w przedsiębiorstwach. Rozległość interfejsów API samej platformy .NET jest ogromna i jest ona używana w miliardach systemów komputerowych na całym świecie. Przez lata zbudowałem wiele aplikacji internetowych przy użyciu różnych technologii, w tym ASP.NET WebForms, AngularJS, Angular, VueJS, Svelte, a nawet React, a ostatnio ASP.NET Core Model View Controller, Razor Pages oraz Blazor. Blazor łączy siłę ugruntowanego ekosystemu z elastycznością oraz elegancją WWW i ma wiele do zaoferowania zarówno platformie .NET, jak i twórcom aplikacji internetowych.

Kto powinien przeczytać tę książkę

Ta książka jest przeznaczona dla programistów .NET i programistów aplikacji internetowych z podstawową wiedzą na temat HTML, CSS, Document Object Model i JavaScript, a także z pewnym doświadczeniem w tworzeniu aplikacji w .NET. Ta książka nie jest dla osób, które są całkowicie początkującymi w programowaniu. Na przykład, kiedy powiedziałem mamie, że piszę książkę, zapytała, o czym jest i czy jej się spodoba. Odpowiedziałem, że nie. Nie jest ani programistką .NET, ani twórczynią aplikacji internetowych, więc nie sądzę, by książka ta była dla niej przydatna. Jeśli ktoś jest jednak programistą .NET lub programistą aplikacji internetowych, będzie zadowolony.

Dla programistów .NET

Jeśli ktoś jest programistą .NET, którego ciekawi tworzenie aplikacji internetowych, ta książka szczegółowo opisuje, w jaki sposób można wykorzystać swoje istniejące umiejętności .NET i zastosować je do programowania aplikacji Blazor. Ta platforma aplikacji internetowych stanowi dużą szansę dla obecnych programistów .NET. Wszystkie popularne platformy SPA wykorzystujące język JavaScript, takie jak Angular, React, VueJS i Svelte, mają w technologii Blazor prawdziwego rywala. Tworzenie aplikacji Blazor powinno być wygodne, ponieważ Blazor jest oparty na .NET i C#. Można wykorzystywać te same biblioteki po stronie klienta i po stronie serwera, dzięki czemu programowanie jest naprawdę przyjemne.

Dla programistów aplikacji internetowych

Jeśli ktoś jest programistą aplikacji internetowych, który wcześniej pracował z .NET, ta książka rozszerza oba te zakresy wyuczonych umiejętności programowania. Całe dotychczasowe doświadczenie w .NET oraz znajomość podstaw programowania dla WWW będą nadal przydatne. Jeśli ktoś jest już programistą aplikacji SPA, ta książka pokaże lepszy zestaw narzędzi niż ten, do którego można być przyzwyczajonym. Omówimy też wiele nowych funkcji języka C#. Jeśli ktoś nie zna języka C#, książka ta pokaże idiomatyczne użycie języka C# i zdecydowane podejście do zasad programowania przy jego użyciu.

Jeśli ktoś się zastanawia, co oznacza *idiomatyczny C#*, to podobnie jak wszystkie inne języki programowania, język C# jest zbiorem idiomów programistycznych. Idiomy programistyczne to sposób na pisanie mądrzejszego i lepszego kodu, aby coś zrobić. Idiomatyczny C# to zestaw idiomów, które są używane, aby kod był bardziej czytelny i łatwiejszy w utrzymaniu.

Dotychczasowe doświadczenie w zakresie języka JavaScript i ogólne doświadczenie programistyczne związane z routingiem po stronie klienta oraz dogłębne zrozumienie protokołu HTTP, architektury mikrousług, wstrzykiwania zależności oraz aplikacji opartych na składnikach będą wciąż przydatne – wszystkie te rzeczy mają bezpośrednie zastosowanie do programowania aplikacji Blazor. Tworzenie aplikacji nie powinno być takie trudne i naprawdę wierzę, że Blazor to ułatwia. Dzięki bogatemu w funkcje wiązaniu danych, szablonom z silnym typowaniem, obsłudze zdarzeń w hierarchii składników, rejestrowaniu wpisów w dzienniku, lokalizacji, uwierzytelnianiu, obsłudze PWA i hostingowi mamy wszystkie elementy składowe do organizowania atrakcyjnych interfejsów internetowych.

Dlaczego napisałem tę książkę

Jeśli ktoś mnie pyta, dlaczego chciałem napisać książkę, udaję głęboki namysł, zanim po prostu odpowiadam, że wydawnictwo O'Reilly mnie o to poprosiło. Ale mówiąc poważnie, kiedy dostałem przyjacielską wiadomość od redaktora wydawnictwa O'Reilly z zapytaniem, czy byłbym zainteresowany napisaniem książki na temat Blazor, długo się nad tym zastanawiałem. Po pierwsze, fajnie było zostać o to poproszonym! Wiedziałem też jednak, że podjęcie się tego rodzaju projektu będzie oznaczało wstrzymanie kilku spraw. Musiałbym zrobić sobie przerwę od publicznych wystąpień, które były główną częścią mojego życia w ciągu ostatnich kilku lat. Uwielbiam pomaganie innym, a napisanie książki byłoby inną formą takiej pomocy. Pisanie książki oznaczałoby również zabranie nieco czasu poświęcanego mojej młodej rodzinie. Moja rodzina, a zwłaszcza moja żona, byli niezwykle serdeczni i wspierający w tej kwestii. Dostrzega ona moją zdolność do pomagania innym i podziela moją pasję. W końcu zdecydowałem się, że chcę napisać tę książkę. Pomaganie społeczności programistów pomaga również mnie samemu wzmocnić swoje zrozumienie konkretnej technologii. Kocham Blazor! Blazor jest ważną inwestycją dla zespołów programistycznych .NET i ASP.NET w firmie Microsoft. Wciąż napędzają innowacje, rozszerzając zasięg języka C# i ekosystemu .NET jako całości. Ta książka jest *pozycją obowiązkową dla programistów* i jest moim sposobem na odwdzięczenie się społeczności programistów, którą pokochałem. Bardzo zaangażowałem się w napisanie tej książki i sądzę, że widać w niej mój entuzjazm dla technologii Blazor.

Jak korzystać z tej książki

Nie jest to typowa książka wprowadzająca do technologii X. Jest to książka techniczna, która wprowadzi czytelnika w korzystanie z technologii Blazor do budowania aplikacji SPA za pomocą WebAssembly i C#. Istnieje wiele książek, które stosują podejście „krok po kroku" – ta książka nie jest jedną z nich.

Chciałem, aby osoba czytająca tę książkę miała odczucia podobne, jak przy dołączaniu do nowego zespołu. Zaczniemy od adaptacji do nowego stanowiska i przedstawienia istniejącej aplikacji oraz poznania różnych elementów dziedzinowych związanych z aplikacją. Przykładowa aplikacja „Learning Blazor" jest dość sporej wielkości rozwiązaniem z kilkunastoma projektami różnych rozmiarów. Każdy projekt zawiera określoną funkcjonalność aplikacji „Learning Blazor". Zbadamy te projekty jako przykłady tego, jak robić rzeczy w technologii Blazor. Będę przedstawiał wewnętrzne działanie aplikacji, pozwalając czytelnikowi poznać różne aspekty programowania aplikacji Blazor. Na koniec czytelnik zdobędzie doświadczenie w tworzeniu aplikacji Blazor i zrozumie, dlaczego podjęto pewne decyzje programistyczne oraz będzie miał działające przykłady realizacji różnych elementów. Cała książka może stanowić inspirację dla dalszych, własnych aplikacji.

Wszystkie przykłady w tej książce są pokazane za pomocą modelowej aplikacji „Learning Blazor". Kod źródłowy modelowej aplikacji z tej książki wraz z samą książką stanowią świetny zasób edukacyjny i przyszły punkt odniesienia. Repozytorium z kodem źródłowym jest dostępne w serwisie GitHub i opisane w podrozdziale „Kod musi żyć dalej" na stronie 18.

Plan i cele tej książki

Struktura tej książki jest następująca:

- Rozdział 1: „Wprowadzenie do Blazor" przedstawia podstawowe pojęcia i podstawy Blazor jako platformy do tworzenia aplikacji internetowych. Przedstawia również przykładową aplikację dla tej książki i omawia jej architekturę.
- Rozdział 2: „Wykonywanie aplikacji" zagłębia się w sposób działania aplikacji, począwszy od pierwszego żądania klienta do adresu URL statycznej witryny. Dowiemy się, jak kod HTML jest przedstawiany w przeglądarce, jak wywoływane są kolejne żądania dodatkowych zasobów i jak Blazor się uruchamia.
- Rozdział 3: „Podział na składniki" przechodzi do omówienia sposobu, w jaki użytkownik jest reprezentowany w aplikacji. Dowiemy się, jak weryfikować tożsamość użytkownika za pomocą zewnętrznych dostawców uwierzytelniania. Poznamy sposoby dostosowywania interfejsu stanu uwierzytelniania oraz różne podejścia do wiązania danych ze strukturami kontrolek Razor.
- Rozdział 4: „Dostosowywanie interfejsu logowania użytkownika" zawiera szczegółowe informacje na temat rejestrowania usług klienckich na potrzeby wstrzykiwania

zależności. Zapoznamy się z różnymi składnikami i sposobami korzystania z podejścia `RenderFragment` przy dostosowywaniu składników. Dowiemy się też, jak pisać i stosować sparametryzowaną syntezę mowy po stronie klienta, która jest w pełni funkcjonalna i konfigurowalna w Blazor WebAssembly.

- Rozdział 5: „Lokalizowanie aplikacji" pokazuje, jak można użyć bezpłatnego, zautomatyzowanego (opartego na sztucznej inteligencji) potoku ciągłego wdrażania do obsługi lokalizacji. Nauczymy się, jak korzystać z zapewnianego przez platformę typu `IStringLocalizer<T>` i powiązanych usług.
- Rozdział 6: „Przykłady funkcjonalności internetowych działających w czasie rzeczywistym" wprowadza funkcje internetowe w czasie rzeczywistym i pokazuje system powiadomień, stronę prezentowania strumienia „tweetów" na żywo i możliwości alertów. Ponadto dowiemy się, jak zbudować aplikację czatu za pomocą ASP.NET Core SignalR.
- Rozdział 7: „Korzystanie z generatorów kodu źródłowego" pokazuje, jak generatory kodu źródłowego poprawiają działanie wywołań międzyoperacyjnych (interop) pomiędzy Blazor a JavaScript. Dowiemy się, dlaczego generatory kodu źródłowego C# są tak przydatne w programowaniu aplikacji i jak pozwalają zaoszczędzić mnóstwo czasu.
- Rozdział 8: „Sprawdzanie poprawności danych wejściowych" zajmuje się sposobem działania formularzy. Zajmiemy się zaawansowanym sprawdzaniem poprawności danych wejściowych z `<form>`. Przyjrzymy się również, jak dołączyć funkcję rozpoznawania mowy do formularza, aby dać użytkownikom inną opcję wprowadzania danych wejściowych. Dowiemy się, jak korzystać z `EditContext` i wiązania modelu formularza. Rozdział 8 demonstruje również wzorzec niestandardowego sprawdzania poprawności stanu, który uzyskuje aktualizacje na bieżąco przy użyciu Reactive Extensions for .NET.
- Rozdział 9: „Testowanie wszystkiego" nauczy, jak pisać testy jednostkowe, testy składnikowe, a nawet testy kompleksowe, aby upewnić się, że aplikacja działa poprawnie. Te testy można zautomatyzować, aby były uruchamiane za każdym razem, gdy aplikacja jest przekazywana do repozytorium GitHub przy użyciu GitHub Actions.

Konwencje wykorzystywane w tej książce

Następujące konwencje typograficzne są używane w tej książce:

Kursywa
: Wskazuje nowe pojęcia, adresy URL, adresy e-mail, nazwy plików i rozszerzenia plików.

`Stała szerokość liter`
: Używana w przykładach programów, a także w treści akapitów przy odwołaniach do elementów programów, takich jak nazwy zmiennych lub funkcji, bazy danych, typy danych, zmienne środowiskowe, instrukcje i słowa kluczowe.

Ten element oznacza wskazówkę lub sugestię.

Ten element oznacza ogólną uwagę.

Ten element wskazuje ostrzeżenie lub przestrogę

Korzystanie z przykładów kodu

Materiały dodatkowe (przykłady kodu, ćwiczenia, itd.) są dostępne do pobrania pod adresem *https://oreil.ly/learning-blazor-code*.

Pytania techniczne lub problemy związane z korzystaniem z przykładów kodu można kierować na adres e-mail *bookquestions@oreilly.com*.

Ta książka ma pomóc czytelnikom w realizacji własnych projektów. Ogólnie rzecz biorąc, jeśli jakiś kod przykładowy jest oferowany w tej książce, można z niego korzystać w swoich programach i dokumentacji. Nie trzeba uzyskiwać od nas pozwolenia, o ile nie kopiuje się znacznej ilości kodu. Na przykład napisanie programu, który wykorzystuje kilka fragmentów kodu z tej książki, nie wymaga zezwolenia. Sprzedawanie lub dystrybucja przykładów kodu z książek wydawnictwa O'Reilly wymaga pozwolenia. Cytowanie tej książki i przykładów kodu w odpowiedzi na czyjeś pytanie nie wymaga pozwolenia. Włączenie znaczącej ilości kodu przykładowego z tej książki do dokumentacji swojego produktu wymaga pozwolenia.

Doceniamy powołanie się na źródło, ale w zasadzie go nie wymagamy. Zwykle zawiera ono tytuł, autora, wydawcę i numer ISBN książki. Na przykład: „*Poznaj Blazor*, David Pine (O'Reilly/APN Promise). Copyright 2023 David Pine, 978-83-7541-508-7".

Jeśli ktoś ma wątpliwości, czy użycie przykładów kodu z tej książki wymaga dodatkowego zezwolenia, może się z nami skontaktować pod adresem *permissions@oreilly.com*.

Jak się z nami skontaktować

Komentarze i pytania dotyczące tej książki należy kierować na adres wydawcy:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (w USA lub Kanadzie)
707-829-0515 (połączenie międzynarodowe lub lokalne)
707-829-0104 (fax)

Mamy stronę WWW dla tej książki, gdzie umieszczamy erratę, przykłady i wszelkie dodatkowe informacje. Można uzyskać dostęp do tej strony poprzez adres *https://oreil.ly/learning-blazor.*

Aby przesłać komentarz lub zadać pytanie techniczne dotyczące tej książki, należy wysłać wiadomość pod adres e-mail *bookquestions@oreilly.com*.

Wiadomości i informacje na temat naszych książek i kursów można znaleźć pod adresem *https://oreilly.com.*

Można nas znaleźć w serwisie LinkedIn: *https://linkedin.com/company/oreilly-media*

Można nas śledzić w serwisie Twitter: *https://twitter.com/oreillymedia*

Można nas oglądać na YouTube: *https://www.youtube.com/oreillymedia*

Podziękowania

Kiedyś pojechałem do Serbii w ramach konferencji programistycznej ITkonekt. Podróżowałem samochodem wspólnie z trzema niesamowitymi osobami. Jedną z nich był Jon Galloway, który w tym czasie był dyrektorem wykonawczym .NET Foundation. Drugą był Jonathan LeBlanc będący jedyną znaną mi osobą, która zdobyła nagrodę Emmy za technologię (i który jest teraz również autorem w wydawnictwie O'Reilly). Trzecią osobą był Håkon Wium Lie, który jest znany jako twórca CSS i jest byłym dyrektorem technicznym firmy Opera. Była to świetna okazja, aby uczyć się od nich wszystkich.

W każdym razie podczas tej podróży wyszło na jaw, że z naszej czwórki w samochodzie byłem jedynym, który nie napisał książki. Natychmiast zachęcili mnie do naprawienia tego braku. Kazali mi podzielić się swoją wiedzą ze światem i napisać książkę. Byłem wzruszony, gdy usłyszałem, że moi szacowni przyjaciele i koledzy wierzą we mnie. Nie napisałem książki od razu, ale dużo się nad tym zastanawiałem i czekałem, aż nadejdzie odpowiedni czas. Czyli teraz! Chciałbym podziękować Jonowi, Jonathanowi i Håkonowi za wiarę we mnie i bycie inspiracją dla społeczności programistów.

Dziękuję Benowi Feldzie, za dodanie plików SVG i aktualizacji stylów do komponentów kafelków na stronie startowej modelowej aplikacji. Dziękuję Maxowi Schmittowi za pomoc w uproszczeniu mojego sposobu korzystania z platformy testowej Playwright z przepływu zadań przy sprawdzaniu poprawności kompilacji aplikacji modelowej. Dziękuję Billy'emu Mumby za obszerną pracę nad funkcją listy zadań w aplikacji modelowej z wykorzystaniem bazowego magazynu danych. Praca, którą wykonał, aby wzmocnić Azure Cosmos DB Repository .NET SDK (*https://oreil.ly/1rmND*), jest niezwykle cenna dla społeczności programistów. Dziękuję Weihanowi Li za jego wkład w wykorzystanie przez aplikację modelową generatora kodu źródłowego Blazor, a mianowicie Blazorators (*https://oreil.ly/uBU3o*). Dziękuję Vsevolodowi Šliachtenko za współpracę i pracę ze mną nad elementem GitHub Action do tłumaczenia zasobów platformy Azure (*https://oreil.ly/1pGdr*). Pomógł mi pięknie zaimplementować porcjowanie zapytań. Dziękuję botowi GitHub za automatyzację ponad 73 000 wierszy kodu (wg stanu na lipiec 2022 r.) dla projektu „Learning Blazor".

Chciałbym podziękować mojemu mentorowi i dobremu przyjacielowi Davidowi Fowlerowi. David jest moim mentorem od dłuższego czasu i wszystkie jego cenne lekcje są bliskie mojemu sercu. David jest współautorem kodu upraszczającego przykład minimalnego interfejsu API w moim projekcie otwartego klienta HTTP dla serwisu „Have I Been Pwned". Nasze rozmowy są często najważniejszym wydarzeniem mojego tygodnia; Dzielę się z nim kodem, doświadczeniami, wyzwaniami zawodowymi i przemyśleniami, a on dzieli się swoją mądrością. Jest inspiracją dla mnie i wielu innych i jestem niezmiernie wdzięczny, że mogę się od niego uczyć.

Dziękuję zespołowi wydawnictwa O'Reilly za wsparcie i zachętę. Chciałbym oficjalnie podziękować wszystkim recenzentom tej książki: Ricie Fernando, Carol Tumey, Erikowi Hopfowi, Geraldowi Versluisowi, Johnowi Kennedy'emu, Chadowi Olsonowi i Egilowi Hansenowi. Bez ich niestrudzonych i dokładnych recenzji – od recenzji redakcyjnych analizujących każde słowo po dogłębne recenzje techniczne, zapewniające, że każdy wiersz kodu jest tak prosty i elegancki, jak to tylko możliwe – ta książka nie stałaby się tak dokładna i pomocna. Jakość jest poparta dziesięcioleciami profesjonalnego doświadczenia w świecie rzeczywistym i jestem zachwycony wynikiem.

Chciałbym podziękować Steve'owi Sandersonowi za stworzenie technologii Blazor. Bardzo lubię pisać aplikacje przy użyciu tej technologii. Chciałbym również podziękować licznym członkom społeczności open source i .NET na całym świecie. Jesteście dla mnie inspiracją – dziękuję!

Na koniec chcę podziękować swojej rodzinie. Bez wsparcia mojej niesamowitej żony, Jennifer, nic z tego nie byłoby możliwe. Zachęca mnie do bycia najlepszą możliwą wersją samego siebie. Wierzyła we mnie o wiele dłużej, niż ja sam wierzyłem w siebie. Chcę podziękować moim trzem synom, Lyricowi, Londynowi i Lennyxowi. Nieustannie przypominają mi o przyszłości i dobru, które znajdujemy na świecie. Każde dziecko wyjątkowo nosi w sobie odrobinę dociekliwej natury, ciekawości i radości. Bez ich iskry i wsparcia nie byłoby tej książki. Dziękuję!

Rozdział 1

Wprowadzenie do Blazor

Technologia Node.js zmieniła świat tworzenia nowoczesnych aplikacji internetowych. Jej sukces jest oczywiście częściowo przypisywany popularności języka JavaScript. JavaScript działa obecnie dzięki Node zarówno na kliencie, jak i na serwerze. Właśnie dlatego Blazor odniesie taki sukces – język C# może teraz działać w przeglądarce za pomocą WebAssembly. Dla programistów .NET stanowi to ogromny potencjał, ponieważ obecnie istnieje wiele aplikacji serwerowych napisanych w języku C#. Dla programistów .NET istnieje wiele możliwości tworzenia niesamowitych interfejsów użytkownika za pomocą Blazor.

Po raz pierwszy programiści .NET mogą wykorzystywać swoje istniejące umiejętności pisania w języku C# do tworzenia wszelkiego rodzaju aplikacji w sieci WWW. Zaciera to granice między programistami warstwy prezentacyjnej (frontend) i warstwy wewnętrznej (backend) oraz rozszerza możliwości tworzenia aplikacji dla WWW. W przypadku nowoczesnego tworzenia aplikacji internetowych chcemy, aby aplikacje były responsywne w przeglądarkach zarówno na komputerach, jak i na urządzeniach przenośnych. Nowoczesne aplikacje internetowe są znacznie bardziej wyrafinowane i bogate w treści niż ich poprzednicy i oferują funkcjonalność internetową czasu rzeczywistego, możliwości progresywnych aplikacji internetowych (PWA) i pięknie zorganizowane interakcje z użytkownikami.

W tym rozdziale dowiemy się o początkach tworzenia aplikacji internetowych przy pomocy platformy .NET i narodzinach Blazor. Poznamy rodzaje platform do tworzenia aplikacji jednostronicowych (SPA) i zobaczymy, jak platforma .NET ugruntowała swoje miejsce w ekosystemie WWW. Odpowiem na wiele potencjalnych pytań związanych z tym, *dlaczego* technologia Blazor jest sensowną opcją i omówię jej modele hostingu. Na koniec po raz pierwszy przyjrzymy się przykładowej aplikacji „Learning Blazor". Ta przykładowa aplikacja będzie używana w całej książce, a każdy rozdział zademonstruje różne funkcje technologii Blazor i wykorzysta tę aplikację do ich omówienia.

Powstanie technologii Blazor

W 1996 roku technologia Active Server Pages (ASP) firmy Microsoft zaoferowała pierwszy język skryptowy działający po stronie serwera i silnik dla dynamicznych stron internetowych. Wraz z ewolucją platformy .NET Framework narodziła się technologia ASP.NET, a wraz z nią pojawiła się technologia ASP.NET Web Forms (WebForms). Technologia WebForms była (i nadal jest) używana przez wiele osób, którym podobają się możliwości .NET.

Gdy technologia ASP.NET Model View Controller (MVC) została wydana po raz pierwszy w 2006 roku, sprawiła, że technologia WebForms w porównaniu wydawała się wolniejsza. Technologia MVC przybliżyła programistów ASP.NET do mniej abstrakcyjnego tworzenia stron internetowych. Dzięki ściślejszemu dostosowaniu do standardów sieciowych technologia MVC wprowadziła wzorzec model-widok-kontroler (model-view-controller) do ASP.NET, co pomogło rozwiązać problem zarządzania stanem ASP.NET po przesłaniu danych na serwer. W tamtym czasie był to bolesny punkt w społeczności programistów. Programistom nie podobał się fakt, że technologia WebForms przenosiła dodatkowy stan dla wszystkich kontrolek na stronie wraz z danymi przesyłanymi przez `<form>`. Technologia WebForms obsługiwała stan poprzez ViewState (stan widoku) i inne mechanizmy zarządzania stanem, które były sprzeczne z naturą protokołu HTTP. Technologia MVC skupiła się na możliwościach testowania, kładąc nacisk na znaczenie stabilności oprogramowania. Była to wyraźna zmiana paradygmatu w porównaniu z WebForms.

W 2010 roku wprowadzono silnik widoków Razor, który miał służyć jako jedna z kilku opcji silnika widoku do użycia z ASP.NET MVC. Razor opiera się na składni znacznikowej, która miesza kod HTML oraz C# i jest używana do tworzenia szablonów. Jako produkt uboczny MVC, technologia ASP.NET Web API zyskała na popularności, a programiści chętnie korzystali z mocy .NET. Interfejsy Web API zaczęły być traktowane jako standard tworzenia usług HTTP opartych na .NET. Przez cały ten czas silnik widoków Razor ewoluował, wzmacniał się i dojrzewał.

Ostatecznie na scenie pojawiła się technologia Razor Pages z silnikiem widoków Razor wykorzystującym MVC jako podstawę. Innowacje związane z ASP.NET Core sprawiły, że wiele z tego było możliwe. Dążenie zespołu do *wydajności jako funkcji* jest widoczne w wynikach testów porównawczych TechEmpower (*https://oreil.ly/Ff8lV*), w których ASP.NET Core nadal wspina się coraz wyżej. Kestrel to wieloplatformowy serwer WWW, który jest domyślnie włączany w szablonach projektów ASP.NET Core. Jest to jeden z najszybszych istniejących serwerów WWW (wg stanu na 2022 rok) – zdolny do obsługi ponad 4 milionów żądań na sekundę.

ASP.NET Core oferuje pełną obsługę wszystkich podstawowych elementów, których można oczekiwać od nowoczesnej technologii programistycznej, takich jak (ale nie wyłącznie) wstrzykiwanie zależności, silnie typowane konfiguracje, bogate w funkcje rejestrowanie, lokalizacja, uwierzytelnianie, autoryzacja i hosting. Technologia Razor Pages

skłania się bardziej ku prawdziwemu podziałowi na składniki i opiera się na infrastrukturze Web API.

Po Razor Pages przyszedł czas na technologię Blazor, której nazwa była zainspirowana połączeniem słów „browser" i „Razor". Blazor jest pierwszą platformą SPA w .NET. Blazor korzysta z WebAssembly (Wasm), który jest binarnym formatem instrukcji dla maszyny wirtualnej opartej na stosie. WebAssembly (*https://webassembly.org*) zaprojektowano jako przenośny cel kompilacji dla języków programowania, umożliwiając wdrażanie aplikacji klienckich i serwerowych w Internecie. WebAssembly pozwala aplikacjom internetowym .NET naprawdę konkurować z platformami SPA opartymi na języku JavaScript. Dzięki Blazor możemy uruchamiać program w języku C# na przeglądarce klienta za pośrednictwem WebAssembly i środowiska uruchomieniowego Mono .NET.

Według Steve'a Sandersona, stworzył on Blazor, ponieważ chciał uruchamiać .NET w WebAssembly. Przełom nastąpił, gdy odkrył Dot Net Anywhere (DNA), alternatywne środowiska uruchomieniowe .NET, które można było łatwo skompilować do WebAssembly za pomocą Emscripten (*https://emscripten.org*), kompletnego łańcucha narzędzi kompilujących do WebAssembly, ze szczególnym uwzględnieniem szybkości, rozmiaru i platformy internetowej.

Była to droga, która doprowadziła do powstania jednego z pierwszych działających prototypów wykonujących kod .NET w przeglądarce bez użycia wtyczki. Po tym, jak Steve Sanderson przedstawił niesamowitą demonstrację tej działającej aplikacji .NET w przeglądarce, inne zespoły w firmie Microsoft zaczęły wspierać ten pomysł. Doprowadziło to .NET o krok dalej jako ekosystem i krok bliżej do tego, co znamy dziś jako Blazor.

Teraz, gdy omówiliśmy, jak powstała technologia Blazor, zajmijmy się tym, jak jest ona w stanie ożywić aplikacje i jakie są różne sposoby ich hostingu.

Hosting w Blazor

Istnieją trzy podstawowe modele hostingu Blazor: Blazor Server, Blazor WebAssembly i Blazor Hybrid. Podczas gdy ta książka omawia model Blazor WebAssembly, modele Blazor Server i Blazor Hybrid są również ważnymi alternatywnymi podejściami.

Blazor Server

W przypadku Blazor Server, gdy przeglądarka klienta wysyła początkowe żądanie do serwera WWW, serwer wykonuje kod .NET, aby dynamicznie wygenerować odpowiedź HTML. Kod HTML jest zwracany, a kolejne żądania są wysyłane w celu pobrania CSS i JavaScript zgodnie z zapisami w dokumencie HTML. Po załadowaniu i uruchomieniu skryptów routing po stronie klienta i inne aktualizacje interfejsu użytkownika są możliwe dzięki utrzymywanemu połączeniu ASP.NET Core SignalR. ASP.NET Core SignalR oferuje dwukierunkową komunikację między klientem a serwerem, wysyłając wiadomości w czasie rzeczywistym. Ta technologia jest używana do przekazywania informacji

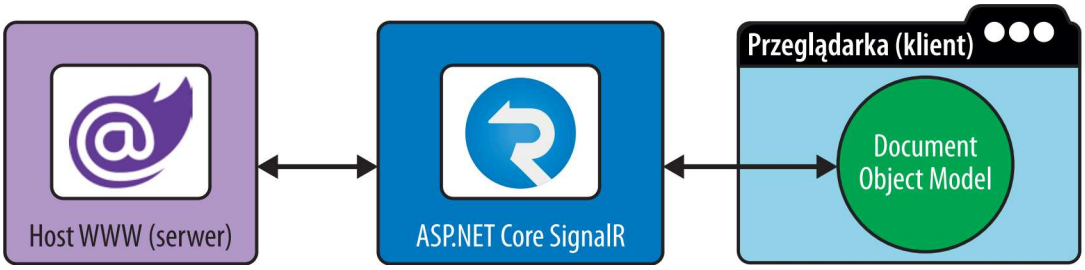
o zmianach w modelu DOM (Document Object Model) w przeglądarce klienta – bez odświeżania strony.

Istnieją zalety korzystania z Blazor Server jako modelu hostingowego w stosunku do Blazor WebAssembly:

- Rozmiar pobieranych elementów jest mniejszy niż w przypadku Blazor WebAssembly, ponieważ aplikacja jest renderowana na serwerze.
- Kod składników nie jest przekazywany do klientów, a jedynie wynikowy kod HTML i trochę kodu JavaScript do komunikacji z serwerem.
- Możliwości serwerowe są obecne w modelu hostingu Blazor Server, ponieważ aplikacja technicznie działa na serwerze.

Dodatkowe informacje na temat Blazor Server można znaleźć w dokumentacji firmy Microsoft: „Modele hostingu ASP.NET Core Blazor" (*https://oreil.ly/rwMaU*).

Rysunek 1-1 przedstawia część serwerową i kliencką. Serwer jest miejscem, w którym wykonywany jest kod Blazor składający się ze składników Razor uruchamianych na platformie .NET. Klient jest odpowiedzialny za renderowanie kodu HTML. Kod JavaScript po stronie klienta przekazuje interakcje użytkownika na serwer, który następnie wykonuje logikę aplikacji przed wysłaniem listy zmian w kodzie HTML (różnic) z powrotem do klienta w celu zaktualizowania jego widoku.



Rysunek 1-1 *Model hostingu Blazor Server*

Blazor WebAssembly

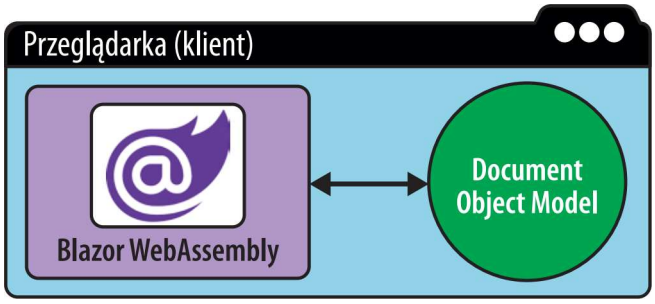
W przypadku Blazor WebAssembly, gdy przeglądarka klienta wysyła początkowe żądanie do serwera WWW, serwer zwraca statyczny widok HTML tego, co aplikacja wyświetliłaby użytkownikowi, gdyby była już uruchomiona; Zapewnia to użytkownikom krótszy czas pierwszego renderowania i umożliwia wyszukiwarkom indeksowanie zawartości aplikacji. Gdy użytkownik wyświetla statycznie wyrenderowaną zawartość, zasoby potrzebne do wykonywania aplikacji na kliencie są pobierane w tle. Częścią kodu HTML aplikacji Blazor WebAssembly będzie element `<link>` żądający pliku *blazor.webassembly.js*. Ten plik jest uruchamiany i rozpoczyna ładowanie WebAssembly, co działa jako rozruch aplikacji, wysyłając żądania dla plików binarnych .NET z serwera. Po pobraniu aplikacji lokalnie i uruchomieniu jej w przeglądarce zmiany modelu DOM, takie jak aktualizowanie wartości danych na stronie, są wprowadzane w miarę pobierania nowych danych z wywołań

interfejsów API. Jest to szczegółowo omówione w podrozdziale „Rozruch aplikacji” na stronie 25.

Ważne jest, aby brać model hostingu pod uwagę. W przypadku hostingu Blazor WebAssembly cały kod C# jest wykonywany na kliencie. Oznacza to, że należy unikać używania kodu, który wymaga funkcjonalności istniejących po stronie serwera, i należy unikać korzystania z poufnych danych, takich jak hasła, klucze interfejsów API lub inne poufne informacje.

Przy korzystaniu z modelu hostingu Blazor WebAssembly możemy utworzyć aplikację obsługiwaną przez Blazor ASP.NET Core lub autonomiczną aplikację, która może być publikowana jako zestaw plików statycznych (oczywiście nie będzie to obsługiwać wstępnego renderowania po stronie serwera na potrzeby wyszukiwarek i sprawniejszego interfejsu użytkownika). W przypadku rozwiązania obsługiwanego przez ASP.NET Core platforma ta jest odpowiedzialna za obsługę aplikacji, a także zapewnianie interfejsu Web API w architekturze klient/serwer. Aplikacja przykładowa dla tej książki korzysta z modelu *autonomicznego* i jest wdrażana w usłudze Azure Static Web Apps. Innymi słowy, aplikacja jest obsługiwana jako zestaw plików statycznych. Dane używane do obsługi aplikacji są dostępne jako kilka punktów końcowych interfejsu Web API, które są wdrażane jako kontenery lub jako proste, odporne na uszkodzenia, przechodnie interfejsy API z monitorowaniem. Korzystamy również z usługi Azure Functions jako architektury bezserwerowej dla lokalnych, bieżących i aktualnych danych pogodowych.

Rysunek 1-2 pokazuje tylko klienta. Klient odpowiada za wszystko w tym scenariuszu, a strony internetowe mogą być dostarczane w formie statycznej.



Rysunek 1-2 *Model hostingu Blazor WebAssembly*

W przypadku podejścia autonomicznego przydatna jest możliwość korzystania z funkcjonalności chmury bezserwerowej za pośrednictwem usługi Azure Functions. Tego rodzaju możliwości mikrousługowe doskonale współpracują z interfejsami ASP.NET Core Web API oraz autonomicznymi scenariuszami Blazor WebAssembly i razem służą jako pożądany cel wdrażania za pomocą usługi Azure Static Web Apps. Statyczne serwery WWW dostarczają pliki statyczne, co jest mniej kosztowne obliczeniowo niż przetwarzanie żądania, które następnie musi dynamicznie wyrenderować kod HTML zwracany następnie jako odpowiedź.

Chociaż ta książka koncentruje się na tworzeniu aplikacji Blazor WebAssembly, która jest hostowana jako pliki statyczne, ważne jest, aby pamiętać, że nie jest to jedyna opcja. Wolę tworzyć aplikacje Blazor WebAssembly, które są hostowane statycznie. Dodatkowe informacje na temat tego modelu hostingu można znaleźć w dokumentacji firmy Microsoft: „Modele hostingu ASP.NET Core Blazor" (*https://oreil.ly/xuL8J*).

W przypadku modelu hostingu Blazor WebAssembly możemy pisać kod C#, który działa po stronie klienta. Jeśli chodzi o WebAssembly, „binarny format instrukcji" oznacza, że mówimy o kodzie bajtowym. WebAssembly bazuje na „maszynie wirtualnej opartej na stosie". Instrukcje są dodawane na stos, a wyniki są ściągane ze stosu. WebAssembly jest „przenośnym celem kompilacji". Oznacza to, że możliwe jest kompilowanie do WebAssembly kodu w językach C, C ++, Rust, C# i innych językach programowania nieużywanych tradycyjnie w programowaniu internetowym. Daje to w wyniku pliki binarne WebAssembly, które można uruchamiać w sieci WWW w oparciu o otwarte standardy, ale pochodzące z języków programowania innych niż JavaScript.

Blazor Hybrid

Model Blazor Hybrid wykracza poza zakres tej książki. Jego cel jest ukierunkowany na tworzenie natywnych interfejsów klienckich dla komputerów stacjonarnych oraz urządzeń przenośnych i działa dobrze z technologią MAUI (.NET Multiplatform App UI). Więcej informacji na temat Blazor Hybrid można znaleźć w dokumentacji Microsoft „ASP.NET Core Blazor Hybrid" (*https://oreil.ly/pubzs*).

Aplikacje jednostronicowe zdefiniowane na nowo

Blazor jest jedyną istniejącą platformą jednostronicowych aplikacji SPA opartą na .NET. Fakt, że możemy używać .NET do pisania aplikacji SPA, jest nie do przecenienia. Istnieje wiele popularnych platform SPA opartych na JavaScript, w tym (ale nie wyłącznie) następujące:

- Angular (*https://angular.io*)
- React (*https://reactjs.org*)
- VueJS (*https://vuejs.org*)
- Svelte (*https://svelte.dev*)

Wszystkie są oparte na JavaScript, podczas gdy Blazor nie jest. Lista ta nie jest wyczerpująca – istnieje o wiele więcej platform SPA opartych na JavaScript, a jeśli o to chodzi, nawet jeszcze więcej ogólnych platform aplikacji internetowych opartych na JavaScript! Język JavaScript rządził w przeglądarkach jako wyłączny język programowania przez ponad 20 lat. Jest to bardzo elastyczny język programowania i należy do najpopularniejszych

na świecie. W swoich początkach prototyp tego języka został stworzony w ciągu kilku tygodni przez Brendana Eicha – to niesamowite, jak daleko zaszedł od tego czasu.

Serwis Stack Overflow przeprowadza coroczną ankietę wśród zawodowych programistów i w 2021 roku ponad 58000 zawodowych programistów oraz ponad 83000 programistów w ogóle zagłosowało na JavaScript jako najczęściej używany język programowania. Był to dziewiąty rok z rzędu, w którym JavaScript był najczęściej używanym językiem programowania. Na drugim miejscu (z niewielką różnicą głosów) znalazł się HTML/CSS[1]. Jeśli połączyć te wyniki, programowanie aplikacji internetowych ma solidną przyszłość.

Jedną z postrzeganych wad języka JavaScript jest to, że bez ustalonych typów programiści muszą albo kodować defensywnie, albo stawić czoła potencjalnym konsekwencjom błędów w czasie wykonywania programu. Jednym ze sposobów rozwiązania tego problemu jest użycie języka TypeScript.

Język TypeScript został stworzony przez Andersa Hejlsberga (który był także głównym architektem języka C#, głównym inżynierem języka Turbo Pascal i głównym architektem Delphi – jest geniuszem języków programowania!). TypeScript zapewnia system typów, który umożliwia narzędziom obsługującym ten język wnioskowanie na temat zamierzonych celów poszczególnych elementów kodu.

Za pomocą języka TypeScript można pisać ogólny kod bezpieczny pod względem typów, korzystając ze wszystkich najnowszych standardów ECMAScript i prototypowych funkcji. Najlepsze jest to, że kod jest wstecznie kompatybilny ze standardem ES3. Język TypeScript jest nadzbiorem JavaScript, co oznacza, że każdy prawidłowy kod JavaScript jest również prawidłowym kodem TypeScript. TypeScript zapewnia statyczne typowanie (system typów) i zaawansowane usługi językowe udostępniające swoje funkcje zintegrowanym środowiskom programistycznym. Dzięki temu programowanie w JavaScript jest mniej podatne na błędy, czego nie można lekceważyć. TypeScript jest bardziej narzędziem programistycznym niż językiem programowania, ale ma niesamowite funkcje językowe. Po kompilacji wszystkie typy znikają, a pozostaje tylko kod JavaScript. Można traktować TypeScript jako sposób na znaczne ułatwienie debugowania i refaktoryzacji oraz zwiększenie niezawodności przy programowaniu w języku JavaScript. Dzięki TypeScript mamy jedno z najbardziej zaawansowanych narzędzi do analizy przepływu i znacznie bardziej zaawansowane funkcje językowe niż w samym JavaScript. Wszyscy twórcy aplikacji internetowych wiedzą, że Angular i React rywalizują ze sobą o tytuł najpopularniejszej platformy do tworzenia aplikacji SPA opartych na JavaScript. Uważam, że duża część przewagi konkurencyjnej w przypadku Angular jest bezpośrednio powiązana ze znacznie wcześniejszym przyjęciem języka TypeScript w porównaniu z React.

Blazor, w przeciwieństwie do aplikacji SPA opartych na JavaScript, opiera się na platformie .NET. Podczas gdy TypeScript może zwiększyć produktywność programistom piszącym kod JavaScript, jednym z głównych powodów, dla których Blazor ma świetlaną przyszłość, jest współdziałanie z językiem C#. Język C# od dawna ma większość zalet, które język TypeScript zaoferował programowaniu w JavaScript i nie tylko. Język C# ma nie

1 „Stack Overflow Developer Survey 2021", Stack Overflow, *https://oreil.ly/bngvt*.

tylko doskonały system typów, ale jest jeszcze lepszy w wyłapywaniu błędów w czasie kompilacji. Statyczny system typów TypeScript jest oparty na „kaczym typowaniu" (jeśli coś wygląda jak kaczka i brzmi jak kaczka, traktuj to jak kaczkę), podczas gdy C# ma ścisły system typów zapewniający, że obiekt, który przekazujemy, jest wystąpieniem konkretnego typu. Język C# zawsze priorytetowo traktował środowisko programistyczne z obsługą analizy przepływu, uzupełnianiem instrukcji, rozbudowanym ekosystemem i niezawodną refaktoryzacją. C# to nowoczesny, zorientowany obiektowo i bezpieczny pod względem typów język programowania, który stale ewoluuje i dojrzewa, dalej poszerzając swoje możliwości. Jest oparty na otwartym kodzie źródłowym, a środowisko programistów często inspiruje jego twórców i wpływa na nowe funkcje języka.

Biorąc to wszystko pod uwagę, Blazor zapewnia również współpracę z JavaScript. Możemy wywoływać funkcje JavaScript z kodu Blazor i możemy wywołać kod .NET ze swojego kodu JavaScript. Jest to przydatna funkcja umożliwiająca wykorzystanie istniejących funkcji języka JavaScript oraz interfejsów API napisanych w JavaScript.

Dlaczego warto korzystać z Blazor

Istnieją interesujące nowe scenariusze specyficzne dla WebAssembly, które nie były realistycznie osiągalne za pomocą samego języka JavaScript. Łatwo sobie wyobrazić aplikacje dostarczane przez Internet do przeglądarki, obsługiwane przez WebAssembly w przypadkach bardziej złożonych i wymagających dużych zasobów. Jeśli ktoś wcześniej nie słyszał o AutoCAD, jest to oprogramowanie do projektowania wspomaganego komputerowo, z którego korzystają architekci, inżynierowie i specjaliści budowlani przy tworzeniu rysunków dwuwymiarowych i trójwymiarowych. Jest to aplikacja komputerowa, ale wyobraźmy sobie, że możemy uruchomić taki program bezpośrednio w przeglądarce internetowej. Wyobraźmy sobie edycję audio i wideo albo granie w rozbudowane i wymagające dużych zasobów gry w przeglądarce. WebAssembly pozwala nam nieco przeobrazić sieć WWW. Platforma aplikacji internetowych może być kolejnym mechanizmem dostarczania nowej generacji oprogramowania. Platforma do tworzenia aplikacji internetowych nadal ewoluuje, rośnie i dojrzewa. Systemy przetwarzania i pozyskiwania danych oparte na Internecie rozwijają się dzięki łączności ze światem. Platforma do tworzenia aplikacji internetowych służy jako środek łączący wyobraźnię programisty i pragnienia użytkownika.

Programiści mogą nadal rozszerzać swoje umiejętności w zakresie C# i Razor na tworzenie aplikacji SPA, zamiast uczyć się dodatkowego języka i platformy renderowania. Programiści C#, którzy wcześniej nie byli skłonni do pisania aplikacji SPA, teraz przechodzą z MVC na SPA po prostu dlatego, że „to wciąż język C#". Dodatkowo potencjał współdzielenia kodu między różnymi aplikacjami jest ogromny. Zamiast pilnować, aby kontrakty interfejsów API w języku C# na serwerze były ręcznie zgrane z definicjami w języku TypeScript, można po prostu użyć tego samego pliku kontraktów wraz ze wszystkimi atrybutami sprawdzania poprawności przy użyciu `DataAnnotation`.

Wierzę, że w nadchodzących latach zobaczymy coraz więcej aplikacji obsługiwanych przez WebAssembly. Blazor WebAssembly będzie często wybieranym rozwiązaniem w ramach .NET.

Potencjał .NET w przeglądarce

W swojej pierwszej pracy programistycznej po studiach byłem najmłodszym programistą w zespole programistów i architektów. Doskonale pamiętam, jak siedziałem sam w biurze; sąsiednie biurka były puste. Ale wszystkie okoliczne biura były wypełnione resztą zespołu.

Pracowałem w branży motoryzacyjnej i wdrażaliśmy standard komunikacji niskiego poziomu, znany jako protokoły diagnostyki pokładowej (OBD). Robiliśmy to przy pomocy klasy .NET o nazwie `SerialPort`. Pisaliśmy aplikacje, które przeprowadzały stanowe testy emisji spalin. W Stanach Zjednoczonych większość stanów wymaga, aby pojazdy w określonym wieku miały przeprowadzane coroczne testy emisji spalin, żeby sprawdzać, czy mogą być zarejestrowane. Pomysł jest dość prosty: sprawdzać różne stany pojazdu. Na przykład pojazd może mieć sprzęt wyzwalający zmiany stanu, które rozprzestrzeniają się poprzez oprogramowanie układowe, a informacja jest przekazywana na bieżąco. System OBD znajduje się w komputerach pokładowych pojazdu, które mogą przekazywać te informacje zainteresowanym stronom. Na przykład lampka „sprawdź silnik" jest kodem diagnostycznym z systemu OBD.

Aplikacje te były tworzone głównie jako aplikacje Windows Forms (WinForms), ale było też kilka aplikacji będących usługami internetowymi. Oznaczało to jednak, że aplikacja była wtedy ograniczona do platformy .NET Framework i Windows – innymi słowy, nie była wieloplatformowa. Aplikacja musiała komunikować się z różnymi usługami internetowymi, aby pobierać i zachowywać dane. W tamtym czasie było niewyobrażalne, aby coś takiego napisać i wdrożyć jako aplikację internetową; musiała to być aplikacja WinForms działająca w systemie Windows.

Teraz jednak bardzo łatwo sobie wyobrazić, że ta aplikacja mogłaby zostać przepisana jako aplikacja internetowa za pomocą Blazor WebAssembly. Środowisko uruchomieniowe Mono .NET umożliwia pisanie międzyplatformowych aplikacji .NET.

Spróbujmy wyobrazić sobie, jak proste mogłoby być zaimplementowanie tego samego obiektu `SerialPort` z .NET, którego używaliśmy w WinForms, tym razem w aplikacji Blazor WebAssembly. Odpowiednia implementacja mogłaby się hipotetycznie opierać na komunikacji WebAssembly z natywnymi interfejsami Web Serial API napisanymi w JavaScript. Tego rodzaju wieloplatformowa funkcjonalność istnieje już w innych implementacjach, takich jak klasa `HttpClient` z .NET zaimplementowana w Blazor WebAssembly. W przypadku Blazor WebAssembly naszym celem kompilacji jest WebAssembly, a implementacją środowiska uruchomieniowego Mono jest interfejs Web API `fetch`. Platforma .NET ma teraz dostęp do całej gamy elementów programistycznych dostępnych przez sieć WWW.