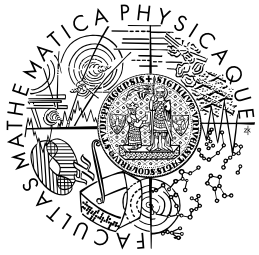

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Tomáš Husák

Client-side execution of PHP applications compiled to .NET

Department of Software Engineering

Supervisor of the bachelor thesis: RNDr. Filip Zavoral, Ph.D.

Study programme: Computer Science (B1801)

Study branch: ISDI (1801R049)

Prague 2021

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Dedication.

Title: Client-side execution of PHP applications compiled to .NET

Author: Tomáš Husák

Department: Department of Software Engineering

Supervisor: RNDr. Filip Zavoral, Ph.D., Department of Software Engineering

Abstract: Blazor is a new technology enabling to run .NET applications directly in the browser using WebAssembly, a recently created binary instruction format adopted by major web browsers. Whilst PHP is the most popular language in the realm of web applications, it cannot run directly in the browser. The Peach-Pie compiler provides a way to compile projects written in PHP into Common Intermediate Language (CIL), enabling them to run on the .NET platform.

This thesis aims to design and implement a convenient interface between Blazor and compiled PHP, enabling developers to create client-side PHP applications. These applications would be able to utilize the specifics of the client-side paradigm, such as fast response times, the possibility to preserve the application state between the requests more efficiently and the direct access to the Document Object Model (DOM) of the page. To demonstrate the usability of the implementation and the specific benefits of the solution, a pilot interactive application will be created.

Keywords: PHP .NET Blazor Peachpie

Contents

Introduction

Nowadays, web applications are mostly dynamic in order to offer a convenient user interface. Dynamic features of a web application are achieved by a programming language JavaScript which is supported by all modern web browsers like Google Chrome, Safari, Opera, and Mozzila. Despite advantages that have JavaScript, like manipulation with DOM, WebAPI, and easy syntax, there are also several severe disadvantages.

The first of them is a slow execution and effectiveness compared with other languages like C++ and CSharp. It became crucial for 3D graphics and, another time expensive computations. Since December 2019, when the W3 Consortium has begun recommending WebAssembly, this problem is solved because WebAssembly is almost as fast as native code in modern browsers. Another advantage of WebAssembly is being a target of compilation-friendly low-level languages like C or C++ due to its memory model. Although there are exists ways, how to compile other high-level languages like CSharp or PHP. A concrete project relates to a PHP to WebAssembly compilation is Pib.

The second problem of Javascript relates to developing a whole web application in more than one single language even though there are Node.js and the project Pib which has some limitations. PHP is the most popular server-side language. Since developers have to understand two programming languages, there is more space for mistakes.

One of this thesis's goals is to design a comfortable way how developers can write a whole web application using only PHP. It can be considered to reinvent a gear, but there is another way how to achieve similar results as the mentioned project and even more to add a new feature.

This thesis aims to integrate Peachpie and Blazor, which results in an interesting connection between PHP world of libraries for server-side processing and Blazor world, which is another way how to create a dynamic web application without JavaScript. Peachpie is able to compile PHP into CIL, and Blazor runs CSharp code on a client-side. So this approach has the potential to offer a possibility to PHP and CSharp developers to collaborate with their programming languages using a minimum knowledge of the integration. Another interesting functionality of this approach is full CSharp, PHP, and JavaScript interop which offers more options for developers and future extensions.

The first chapter addresses existing technologies used in the integration, alongside the existing project, which uses direct compilation to WebAssembly. The second chapter describes the problem of running PHP on the client-side and other problems related to used technologies. The third gives a detailed problem's solution. There are examples that demonstrate how to use all aspects of the created solution in chapter 4. In chapter 5, you can see benchmarks that explore the limits of implementation and compare them with the already existing project. And the last chapter relates to a conclusion of this solution.

1. Existing technologies

At the begining, I will introduce an existing solution of how to run PHP in a browser. The project Pib aims to using compiled php interpreter into WebAssembly which allows to evaluate a PHP code. The page has to import a specialized module php-wasm. A PHP code is evaluated by writing a specialized script block or manually by JavaScript and API. PHP can afterwards interact with JavaScript using a specialized API. At the first glance, that might be good enough solution, but they are several parts which can be problematic due to PHP semantics. The solution doesn't solving globals. This is resonable, because this is server's job, but you are not able to get information about a query part or handling forms without writing a JavaScript code. Next problem is a navigating. How a script can navigate to another script without an additional support code which has to be JavaScript. These problems can be solved by following technologies and their integration.

1.1 WebAssembly

Web is a new code format which can be run in today's browsers. It has a compact byte format and it's performance is near to a native code. WebAssembly is designed to be a compiling target of popular low-level languages like C or C++ due to it's memory model. It should be able to support languages with carbage collector in the future. Advantage of this format is a similarity with Javascript modules ES2015 after compilation into a machine code. This enables browsers to execute it by a JavaScript runtime. So it's security is as good as a code written in Javascript. Because of the same runtime, WebAssembly can call Javascript and vice versa.

Despite of supporting to run WebAssembly in browser, the browser can't load it as a normal ES2015 module yet. WebAssembly Javasrcipt API was created in order to be able to load a WebAssembly to browser using JavaScript.

1.2 Mono

Mono is a .NET runtime which aims to mobile platforms. Recently, they started to support com into WebAssembly. This support allows executing CIL inside browsers. The compilation has two modes. The first one is compilation Mono runtime with all using assemlies. The second only compiles Mono runtime which then can executes .dll files without further compilation of them into WebAssembly. A consequence of these compilation into WebAssembly is enabling to call Javascript and WebAPI from .NET.

1.3 Blazor

Blazor is a framework which provides a convinient way how to write dynamic web pages using CSharp. Blazor platform is devided into two Hos which have different approach how to create web applications. The first one is refered as

Blazor Server App and has a similar methology like a common website written in PHP. The interesting inovation using SignalR which is a comunication protocol between the server and a client. But this thesis uses the second model which Microsoft refers as Blazor WebAssembly App enabling an offline support after loading the app into a browser.

1.3.1 Blazor WebAssembly App

From now on i will use Blazor App to refer Blazor WebAssembly App. Blazor App can be devided into two parts. The first part serves the main WebAssembly application and its additional resources which can be requested during a runtime. The second part is WebAssembly wrapped together with an additional user code. The division enables to choose a place for implementation of bussiness logic. If there is a bad connection, we can move the mejority of bussiness logic to client and use the server for connection to a database, otherwise we can use client only for rendering the page. It consists of the following components. Kestrel with ASP.NET libraries provide server part of application. Mono runtime compiled to WebAssembly runs CSharp code inside browser. WebAssembly is important for be able to interact with DOM and JavaScript using CSharp without additional plugin which was necessary for older technologies like Microsoft Silverlight. Blazor's libraries provides constructs for manipulation with DOM and WebAPI together with rendering the page and JavaScript interop. And there are user code which using the libraries for creating dynamic pages with CSharp. A better imagination, how the app is situated on client side, can be represented by the figure 1.1 copied from Gli.

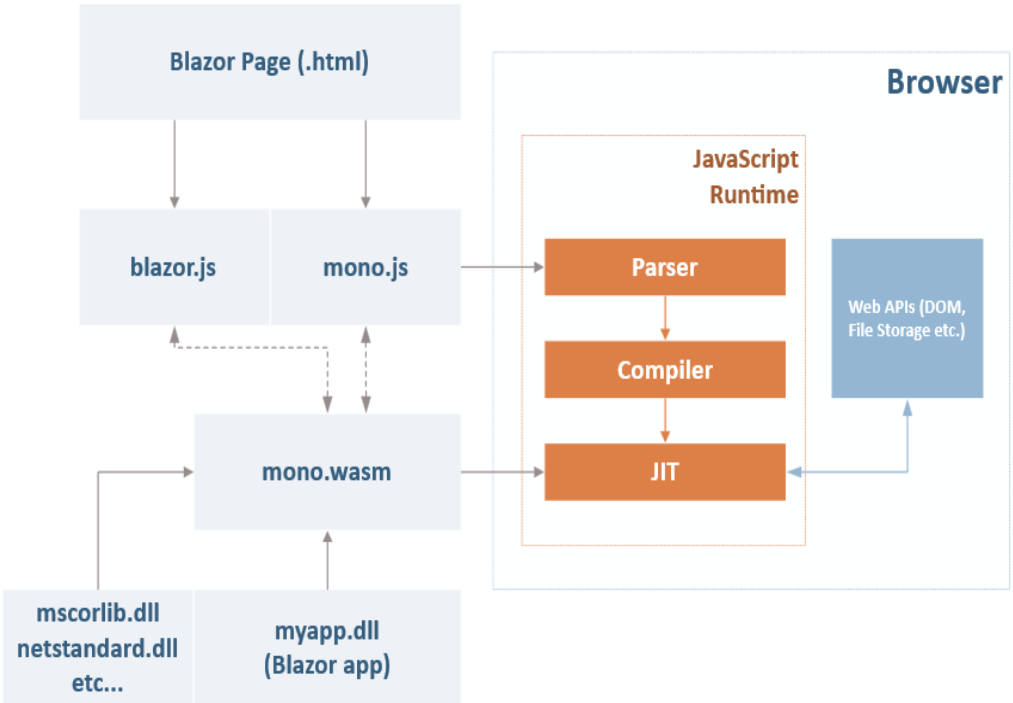

Figure 1.1: Running a Blazor WebAssembly App on client-side.

Technical details of interop with a browser is one part of the Blazor App. The main part is an architecture of the libraries. A common approach how to create a page is using the markup language Razor. There already exists Razor in common ASP.NET website where .cshtml extensions consist of this markup. Unfortunately the markup used in BlazorApp has the same name. From now on i will use Razor for the markup language which is content of .razor files in Blazor App. Because interleaving of HTML with other language turns out to be useful, the Razor uses spacial characters to identify CSharp code in HTML and convert it to rich content pages. A significant purpose of Razor is for generating CSharp structures, which represents parts of page, during compilation time. These structures have complex interface for rendering a page, so the markup is there in order to free user using complicated machanism for putting page together.

Blazor introduces Component which can represent a whole page or the part of it. Components can be arbitrarily put together in order to form a desired page. They can have different purposes, for example there is Router which takes care of routing the right page whenever the navigation is triggered. Alongside components there is a dispatcher which suplies aditional services like logger to components when they are creating. The last item, which isn' used transparently, is a WebAssemblyHost builder. The builder configures the application and prepare the renderer which is used by components to render their content.

Balzor presents own virtual DOM to reduce changing a DOM directly in a browser in order to its demending performance. Component works with RenderTreeBuilder which provides an interface for adding a content to the virtual DOM. An usage of RenderTreeBuilder is complex due to Blazor's diff algorithm which is used afterwards. RenderTreeBuilder is just a superstructure over Renderer which is responsible for updating the page. After all components used RenderTreeBuilder, the diff algorithm is used to minimize updating of browser's DOM. This algorithm used sequence numbers for parts of HTML to identify modified sections. Sequence numbers responds to an order of RenderTreeBuilder's instructions in the source code. A benefit of this information is detecting loops and conditional statements to generating smaller updates of DOM. It follows browser's DOM update which is executed by Blazor's JavaScript support code called through Mono runtime.

The process of bootstrapping Blazor App to a browser folows these steps. Kestrel gets a request for a page which is contained in Blazor App. The server responses with the index.html page which contains references to JavaScript support code (This code is refered as blazor.js and mono.js in the figure 1.1) responsible to load and run the runtime with application part. The runtime runs the application using main method in Blazor App. Remaing interactions are maintained by event handling. I distinguish two types of events. The first type is a navigation. The nav can be triggered by an anchor, form or filling up the url bar. The url bar is handled seperated by browser. The remainings ways can be influenced by JavaScript. Anchor is the only one which is handled by Blazor App by default. After clicking on an anchor, predefined methods in blazor.js tries to invoke navigation handler in Blazor App using a Mono WebAssembly gateway. This handler can be modified by user, but a default behavoir is implemented by a specialized component Router. The Router finds out all components, which implements an IComponent interface, by a reflection and tries to render the page

according to path matching RouteAttribute of component. The navigation can be redirected to server if there is on match. The second type is events invoked by UI like onchange. These event's callbacks call right CSharp callbacks thanks to RenderTreeBuilder, which can conect CSharp callback with element's event.

1.4 PeachPie

TODO: What is PeachPie

TODO: Compiling PHP to .NET

1.5 PHP

TODO: Classic web page in php

2. Problem analysis

Conclusion

Bibliography

URL https://daveaglick.com/posts/blazor-razor-webassembly-and-mono.

Hosting models. URL https://docs.microsoft.com/en-us/aspnet/core/blazor/hosting-models?view=aspnetcore-5.0.

The project pib. URL https://github.com/oraoto/pib?fbclid=IwAR3KZKXWCC3tlgQf886PF3GT_Hc8pmfCMI1-43gdQEdE5wYgpv070bRwXqI.

Webassebmly. URL https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts.

compilation. URL https://www.mono-project.com/news/2017/08/09/hello-webassembly/.

URL https://chrissainty.com/an-in-depth-look-at-routing-in-blazor/.

List of Figures

List of Tables

List of Abbreviations

A. Attachments

A.1 First Attachment