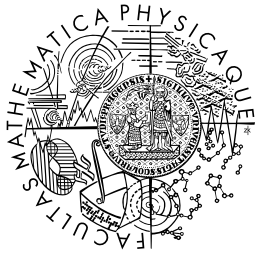

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

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:

Write an abstract.

Keywords: PHP .NET Blazor Peachpie

Contents

Introduction

We can divide web applications into two types by roles of server and client. However, they use different technologies for their purpose. We will start with common parts. An internet protocol, HTTP, usually carries out the communication between a server and a client. A client uses a web browser for requesting a server. A markup language HTML is essential for describing a page's structure. A browser is responsible for interpreting and rendering the page's content. It is a web application's environment for further interaction. The need to adjust content by different styles initiates standardizing CSS language, which enriches pages with a wide graphical content.

The first type is server-based web applications. A server prepares the page, makes additional computations related to the request, and sends it back to the client. Having a business logic on a server-side is the main objective. The most popular language for server-side scripting becomes PHP.

Add a statistic for the popularity of PHP

The second type is client-based web applications, where major business logic is moved to a client's browser. However, the combination of CSS and HTML is sometimes sufficient. This type of application needs dedicated technologies, which allow manipulation with a page structure, reacts to on-page events, and controls the browser's behavior. Many languages were enabling the manipulation, but they were not usually supported by most browsers like Google Chrome, Safari, Opera, and Mozzila. The scripting language Javascript became a browser standard from these supports.

However, Javascript is a powerful language. There are language-specific features, which are harder replaceable by the language. Despite the urge, many technologies like Silverlight, which runs C# code in a browser, or Adobe Flash Player with Actionscript were deprecated due to insufficient support across the browsers. There appeared a portable binary-code format for executing programs, WebAssembly (abbreviated WASM) [1] in 2015. WebAssembly aims to secure high-performance applications on web pages. Interop with Javascript makes the format as powerful as the language. The advantage of WebAssembly is a being compilation target for many programming languages. Since December 2019, when the W3 Consortium has begun recommending WebAssembly, it is easy to migrate other languages to the browsers supporting this recommendation.

Many projects use the WASM as a target of compilation. For example, the project PHP in browser [2]. It enables running PHP script inside our browser using predefined Javascript API or standard tag for HTML script. Another project is an open-source framework Blazor [3] developed by Microsoft. It provides a runtime, libraries, and interop with Javascript for creating dynamic web pages using C#.

Add a statistic for popularity of .NET

The .NET and PHP popularity led to the creation of the Peachpie compiler. Peachpie [4] tries to make use of the .NET platform and offers PHP compilation to .NET. It is a modern compiler enabling interop between PHP and C#.

The project opens the PHP door to Blazor. An integration between Peachie and Blazor can yield to following benefits. A community of PHP developers

is significant. Thus, many PHP libraries apply to working with client's data, cooperation with databases, and other server tasks. The possibility to migrate the language PHP together with its conventions to a browser will impact developing dynamic web applications due to the PHP community and the libraries. It can join PHP and C# developers to collaborate with their programming languages using a minimum knowledge of the integration. Another interesting functionality of this idea is a full C#, PHP, and JavaScript interop which offers more options for developers and future extensions.

This thesis uses the compilation of PHP scripts to .NET in order to execute PHP in a browser powered by Blazor. The approach tries to achieve two goals. The first goal is to enable web development on a client-side with PHP. There are not libraries supporting this integration. The second goal is to design the support to offer a convenient way to combine a PHP code with a Blazor.

The first chapter addresses the analysis of related work, alongside descriptions of the technologies used in the integration. The second chapter analyses 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, we can see benchmarks that explore the limits of the implementation and compare them with the already existing project. And the last chapter relates to a conclusion of this solution.

1. Existing technologies

This chapter describes a classical web application, written in PHP and Javascript, and its conventions helping us design a convenient mechanism of migrating PHP to a browser. It presents a WebAssembly specification, which follows the existing project using the format to achieve running PHP in a browser. Give short information about the language CSharp and Mono runtime used in Blazor. Introduces Blazor and Peachpie, which is integrated by the proposed solution.

1.1 PHP and server-side web application

The basic principle of web pages is a request-response architecture, where a client sends a request for the web page using an HTTP protocol and receives a response with requested data. A PHP language design makes handling these requests on the server easier.

1.1.1 Server-side web application

The previously mentioned principle has several consequences, which are important for our solution. It is necessary to hold the client's application state on the server because the HTTP is stateless. An interaction with a page not using next requests is limited by CSS. HTML presents a tag form as a way how to send additional information to the server after the first request. The form tag contains other tags representing fields, which are filled by the client. Afterward, the form tag enables to encode these data and send it to the server. Get and post methods are relevant methods, how to perform manner of sending the data. Get method encodes the data into an URL. Post method encodes it in the request body, which does not appear in the URL.

1.1.2 Language specification

PHP [5] is interpreted language maintained by The PHP Group. PHP was designed for generating a web page on the server-side. Since PHP is a scripting language, its entry-point is the first line of the executing script.

The type system is dynamic. Variables represent just references to the heap, where all types of objects reside. An unusual thing is superglobals [6], which are built-in variables accessible from all scopes of the script. Following superglobals are relevant to the thesis. The GET variable stores parsed query part of the URL. The POST variable stores variables which are sent by post method. The FILES variable contains uploaded files. There is also a SESSION variable, which holds a user session. This variable will become needless because of the proposed solution.

We can divide code in several ways. Global functions are the most notable characteristic of PHP despite wide-spread object-oriented programming. They are defined in the global scope and accessible from anywhere. The next option is an object inspired by object-oriented programming. There is also namespace

support. And the highest level of division is the module representing the bag of code related to specific behavior like gd2 extension for graphics.

There is not common to use asynchronous functions in a typical PHP application. One-way pass of the application is a standard convention because of request-response semantic. The iconic design pattern is Front Controller. Usually, the main script invokes other parts of the program, based on the request, to deal with it and send the response back.

An HTML interleaving has appeared to be a helpful method for data binding. The feature allows inserting a PHP code between HTML. These fragments do not have to form individual independent blocks of code closed in curly brackets. We can see usage of the interleaving in listing 1.1.

Listing 1.1: HTML interleaving.

```
1 <body>
2 <h1>Superglobal GET</h1>
3 <?php
4   foreach($_GET as $key => $value) {
5 ?>
6   <p><?php echo $key; ?> => <?php echo $value; ?></p>
7 <?php } ?>
8 </body>
```

Uploaded files sent by the client reside in two places. The file's information is stored in FILES. The uploaded file is saved as a temporary file, and standard reading operations can obtain the content.

1.2 Javascript and client-side web application

Control over the rendered page and WebAPI provided by a browser is a point of interest on the client-side. This functionality usually uses Javascript functions as wrappers.

1.2.1 Client-side web application

The process of generating a web page follows several steps. A browser parses the HTML line by line. If a script occurs, the browser starts to execute the code. The order of processing is important for manipulation with an HTML structure. This limitation can be solved by web events mentioned later, but it is a convention to add scripts to the end of the body part after all HTML tags.

We can image a web page as an XML tree. Its nodes are tags or text fragments, and its edges connect nodes with their children. One representation of this tree is Document Object Model (abbreviated DOM). Each node is represented by an object with special parameters relating to HTML and CSS. The nodes can contain other nodes representing their children. Afterward, the document node represents the whole document together with its root node.

Events are the most common method of how to react to changing a web page state. Every event can have some handlers(listeners). Whenever an event occurs, it calls all its listeners. There are many event types, but we will mention the ones that are important for us. HTML tags are the most common entities, which

can have some events. For example, a button has an event onclick which triggers when a client clicks on the button. Other events can represent a state of a page like onload which fires when the whole HTML document is parsed.

A browser provides more APIs valuable for the application, like fetching extra data from a server or local storage. These APIs are mentioned as Web API [7].

1.2.2 Language specification

Javascript is a high-level language usually executed by a browser's dedicated Javascript engine but can also be run on a desktop by Node. It has dynamic typing. It is often used as a wrapper of Web APIs due to Javascript's essentiality in dynamic web pages. These APIs are accessible in global scope as an object or global functions. An example of API available in Javascript can be DOM API mentioned in the previous section. Javascript supports an event-driven style that helps to react to events conveniently.

Javascript is single-threaded, which can be confusing with its constructs for promises. Promise is a structure representing an unfinished process. These processes can be chained. However, the structure can give an illusion of multi-threading. It uses the scheduler for planning the next task executed by the main thread. The single thread is critical for blocking operations which causes thread freezing.

1.3 WebAssembly

WebAssembly [9] is a new code format that can be run in today's browsers. It has a compact byte format, and its 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 its memory model. It should be able to support languages with carbage collector in the future. The 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 its security is as good as a code written in Javascript. Because of the same runtime, WebAssembly can call Javascript and vice versa.

Threads [10] support is currently discussed nowadays and appears to be realistic. After all, new versions of Google Chrome experiments with proper multi-threading support despite the chance of vulnerability. A replacement of multi-threading can be web workers [11]. The worker's limitation is communication with UI thread only by messages.

Despite supporting to run WebAssembly in a browser, the browser cannot load it as a standard ES2015 module yet. WebAssembly JavaScript API was created in order to be able to load a WebAssembly to a browser using JavaScript.

1.4 Project PHP in browser

The project [2] aims to use compiled PHP interpreter into WebAssembly, which allows evaluating 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 afterward interact with JavaScript using a specialized API. At first glance, that might be a good enough solution, but they are several parts that can be problematic due to PHP semantics. The solution doesn't solve superglobals. This is reasonable because this is the server's job, but you are not able to get information about a query part or handling forms without writing a JavaScript code. The next problem is 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.

1.5 CSharp and Mono

We have to introduce .NET [17] in other to understand the following sections fully. .NET is a free and open-source project primarily developed by Microsoft. It is a cross-platform successor to .NET Framework. It consists of core libraries and runtime, which runs a unique code format CIL standing for Common Intermediate language on various environments. The runtime is often named CLR (Common Language Runtime) and represents a virtual machine interpreting the code into the machine code. The libraries can represent whole frameworks like ASP.NET, which aims to web development. The CIL is a compilation target of languages like CSharp, Visual Basic, or FSharp. The most common granularity of .NET projects is an assembly formed from a bunch of code representing a library or an executable program.

1.5.1 Mono

Mono is a .NET runtime that aims to mobile platforms. Recently, they started to support compilation [12] into WebAssembly. This support allows executing CIL inside browsers. The compilation has two modes. The first one is compilation Mono runtime with all using assemblies. The second only compile Mono runtime, which then can execute .dll files without further compilation of them into WebAssembly. A consequence of these compilations into WebAssembly is enabling to call Javascript and WebAPI from .NET.

1.5.2 Language specification

CSharp is a high-level language using strong typing and a garbage collector. It has a multi-paradigm, but its common characteristic is the objected-oriented style. These features cause that CSharp is a good language for a huge project which needs discipline from developers to hold the code understandable and manageable. CSharp is used on the server-side as well as PHP.

1.6 Blazor

Blazor is a framework that provides a convenient way how to write dynamic web pages using CSharp. Blazor platform is divided into two hosting models 13 which have different approaches to creating web applications. The first one is referred

to as Blazor Server App and has a similar methodology to a standard website written in PHP. An interesting innovation is SignalR which is a communication protocol between the server and a client. However, this thesis uses the second model, which Microsoft refers to as Blazor WebAssembly App enabling offline support after loading the app into a browser.

1.6.1 Blazor WebAssembly App

From now on, I will use Blazor App to refer Blazor WebAssembly App. Blazor App can be divided into two parts. The first part serves the main WebAssembly application and its additional resources, which can be requested during runtime. The second part is WebAssembly wrapped together with an additional user code. The division enables to choose of a place for the implementation of business logic. If there is a bad connection, we can move the majority of business logic to the client and use the server for connection to a database; otherwise, we can use the client only for rendering the page. It consists of the following components. Kestrel with ASP.NET libraries provides the server part of an application. Mono runtime compiled to WebAssembly runs CSharp code inside a browser. WebAssembly is essential for being able to interact with DOM and JavaScript using CSharp without an additional plugin, which was necessary for older technologies like Microsoft Silverlight. Blazor's libraries provide constructs for manipulation with DOM and WebAPI together with rendering the page and JavaScript interop. And there is a user's code that using the libraries for creating dynamic pages with CSharp. A better imagination, how the app is situated on the client-side, can be represented by the figure 1.1 copied from the article 15.

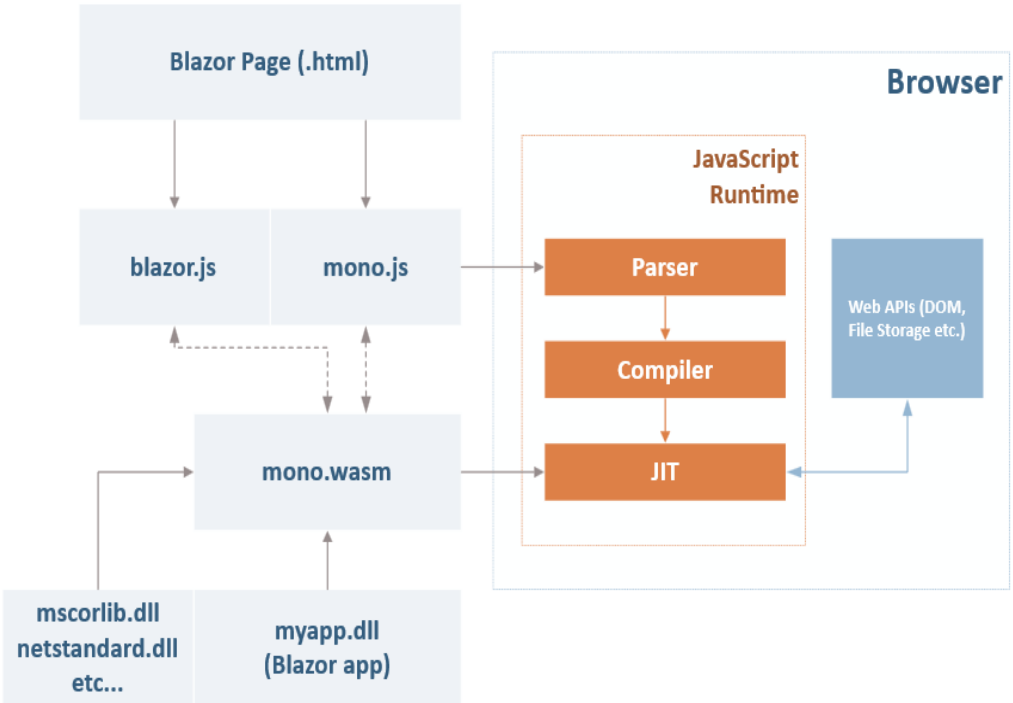

Figure 1.1: Running a Blazor WebAssembly App on client-side.

The interop with a browser are one part of the Blazor App. The main part is the architecture of the libraries. A common approach how to create a page is using the markup language Razor. There already exists Razor in standard 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 the content of .razor files in Blazor App. Because interleaving HTML with other languages turns out to be helpful, the Razor uses special 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 represent parts of a page, during compilation time. These structures have a complex interface for rendering a page, so the markup is there in order to free users using a complicated mechanism for putting a page together. We can see an example of web page in listing 1.2.

Listing 1.2: Example of Razor page.

```
@page "/example"
@inject HttpClient Http

<h1>Example</h1>
@if (!loaded)
{
  <p>Loading...</p>
}
else
{
  <p>Ticks: @ticks</p>
}

@code {
  private bool loaded = false;
  private int ticks = 0;

  protected override async Task OnInitializedAsync() {
    ticks = await Http.GetFromJsonAsync<int>("ticks.json");
    loaded = true;
  }
}
```

Razor provides a special sign at which follows with dedicated keywords. We can see a page keyword determining the path of the page. An inject keyword represents a service injection. An if keyword determines a standard condition, and a code keyword contains a regular CSharp code. These features are afterward transformed into CSharp entities forming a special class.

Blazor introduces a Component class that can represent a whole page or the part of it. Components can be arbitrarily put together in order to form the desired page. We can see the generated Component from listing 1.2 in listing 1.3.

Listing 1.3: Razor page generated to the CSharp class.

```
[Route("/example")]
public class Index : ComponentBase {
  private bool loaded = false;
```

```
4     private int ticks = 0;
5
6     [Inject] private HttpClient Http { get; set; }
7
8     protected override void BuildRenderTree(RenderTreeBuilder
          __builder) {
9       __builder.AddMarkupContent(0, "<h1>Example</h1>");
10      if (!loaded)
11      {
12        __builder.AddMarkupContent(1, "<p>Loading...</p>");
13        return;
14      }
15      __builder.OpenElement(2, "p");
16      __builder.AddContent(3, "Ticks: ");
17      __builder.AddContent(4, ticks);
18      __builder.CloseElement();
19    }
20
21    protected override async Task OnInitializedAsync() {
22      ticks = await Http.GetFromJsonAsync<int>("ticks.json");
23      loaded = true;
24    }
25  }
```

We can assign the Razor keywords to parts of the code in the listing. Page keyword stands for Route attribute. Inject keyword stands for parameter attribute. The parameter is assigned by a dispatcher, mentioned later, during the initialization. Code keyword is a part of class content. Another markup is transformed into calling a specialized method in the BuildRenderTree function, which renders the page. There will be more information about rendering later in this section.

The components can have different purposes. For example, a Router takes care of routing the right page whenever the navigation is triggered. Alongside components, a dispatcher supplies additional services like logger to components when they are creating. The dispather is a specialized class initialized in the begining of the application.

The last item, which is not used transparently, is a WebAssemblyHost builder. The builder configures the application and prepares the renderer used by components to render their content.

Balzor presents its own virtual DOM to reduce changing a DOM directly in a browser to its demanding performance. A component works with RenderTreeBuilder, which provides an interface for adding content to the virtual DOM. The usage of RenderTreeBuilder is complex due to Blazor's diff algorithm, which is used afterward. RenderTreeBuilder is just a superstructure over Renderer, which is responsible for updating the page. The diff algorithm is used to minimize the browser's DOM update after all components used RenderTreeBuilder to render their content. This algorithm used sequence numbers for parts of HTML to identify modified sections. Sequence numbers respond 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 fol-

lows the browser's DOM update, which is executed by Blazor's JavaScript support code called through Mono runtime.

The process of bootstrapping the Blazor App to a browser follows these steps. Kestrel gets a request for a page that is contained in Blazor App. The server responses with the index.html page, which contains references to JavaScript support code (This code is referred to as blazor.js and mono.js in the figure 1.1) responsible for loading and running the runtime with the application part. The runtime runs the application using the Main method in Blazor App. The remaining interactions are maintained by event handling. I distinguish two types of events. The first type is navigation. The navigation [14] can be triggered by an anchor, form, or filling up the URL bar. The URL bar is handled separated by a browser. JavaScript can influence the remainings elements. Blazor App handles only an anchor by. After clicking on an anchor, predefined methods in blazor.js try to invoke navigation handler in Blazor App using a Mono WebAssembly gateway. A user can modify this handler, but a specialized component Router implements a default behavior. 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 a component. The navigation can be redirected to the server if there is no match. The second type is events invoked by UI like onchange. These event's callbacks call right CSharp callbacks thanks to RenderTreeBuilder, connecting CSharp callback with element's event.

Add cutting unneccessarly references to make the app smaller.

1.7 Peachpie

Peachpie [16] is a modern compiler based on Roslyn and Phalanger project. It allows compiling PHP into a .NET assembly, which can be executed alongside standard .NET libraries. Peachpie introduces several structures representing states, scripts, and variables of PHP written in Csharp. The first of them is a context representing one request to PHP code. The context consists of superglobals, global variables, declared functions, declared and included scripts. The possibility of saving the context and using it later is a significant advantage used in the solution. The context can also be considered as a configuration of the incoming script's execution. All information about a request can be arranged to mock every situation on the server-side. The compiler offers a dedicated type of assembly for PHP libraries. Using this assembly can add additional functions, which can provide an extra nonstandard functionality as an interaction with a browser. Another advantage of the compiler is the great interoperability between PHP and .NET. An option to work with Csharp objects, attributes and calling methods will become crucial for achieving advanced interaction between Blazor and PHP.

Saving script information in the assembly

Inheriting the CSharp classes

However, there are limitations following from differences in the languages and the stage of development. Availability of PHP extensions depends on binding these functions to CSharp code which gives equivalent results. The time and memory complexity of this code can be tricky in Blazor. The previously men-

tioned interoperability has limits as well. Csharp constructs like structs and asynchronous methods are undefined in PHP.

2. Problem analysis

This chapter describes the main problems together with using technologies to help to solve them. In the end, we will introduce the proposed solution to the problem.

Current problems of migrating PHP to a browser divides into different types of tasks that have to be done. In the beginning, we have to load scripts alongside the Blazor app into a browser. Afterward, navigation to the scripts is essential for making the PHP website client-side. The ability to find the desired script should be adopted to Blazor's environment in order to combine PHP scripts with Razor pages. Mocking the server role on the client-side will be another important feature to make this migration familiar with standard PHP usage. It consists of managing superglobals like GET or POST, file management. Because we can save the application state, we bring a new feature of choosing the preservation of the script's context to the next evaluation. The interaction with a user is a challenge for PHP due to its server role. And the last problem will be with evaluating the PHP code. Rendering the whole page is a demanding computation. It can be critical for sections, which tend to change their content often.

2.0.1 Proposed solution

The main idea of migrating PHP to a browser is to integrate Peachpie with Blazor. In the beginning, we have to think of how we can put the scripts into a browser. Peachpie can solve it. It compiles our scripts into .NET assembly, which consists of all information about the scripts. We can reference it from a Blazor app as a standard CSharp code when we have the assembly. We have to ensure that compiler will think the application uses the assembly due to cutting unnecessarily assemblies mentioned in the Blazor section. The process results in loading the assembly alongside the Blazor application into a browser. However, this could be considered as the major part of the problem. How to reference and evaluate the scripts is not clear, and there will be many decisions that will not be silver bullets.

We can use a Blazor component class in order to represent a particular script in the Blazor application. This component should be able to find and evaluate a specified script from the assembly. This approach can benefit from the component's reusability. Afterward, we will be able to compose the component with others to make the desired layout. Before that, we have to make finding and evaluation the script clear.

From this time, we have to distinguish between the purposes of the PHP code. We will create two components for each of them. We make the reason clear later in the section and describe detail in the following chapter.

The first purpose is to free the script from Blazor. It is done by finding the script by the name obtained from a component's parameter or the URL. The script's evaluation is done via caching its output and adding it as markup text to the RenderTreeBuilder. It is is a good approach for transparent use of Blazor, but it is ineffective for often rendering.

User interaction -¿ forms

Superglobals get, post, files

The second purpose is to offer the complete interface of Blazor. An inheritance can achieve this. The PHP class can inherit the CSharp component class due to Peachpie. The consequence of this is accessible Blazor functions for rendering the content.

However, these purposes are different. They can be combined due to a Component interface.

Full control over the rendering

Conclusion

Bibliography

Webassembly. URL https://en.wikipedia.org/wiki/WebAssembly.

Threads. URL https://developers.google.com/web/updates/2018/10/wasm-threads.

Webworkers. URL https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers.

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

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

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

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

Peachpie. URL https://docs.peachpie.io.

.net core. URL https://en.wikipedia.org/wiki/.NET_Core.

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

Blazor's homepage. URL https://dotnet.microsoft.com/apps/aspnet/web-apps/blazor.

Peachpie's homapage. URL https://docs.peachpie.io.

Php. URL https://en.wikipedia.org/wiki/PHP.

Php manual. URL https://www.php.net/manual/en/.

Webapi. URL https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Introduction.

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

List of Figures

List of Tables

List of Abbreviations

A. Attachments

A.1 First Attachment