

***Calculation and formatting using Web Assembly***

***Interns:***

Ologu Catinca-Ioana

&

Incze Zsolt-Tamás

**Content:**

**1. Project Description**

**2. Use Case**

**3. WebAssembly**

**4. ASP .NET API**

**5. JavaScript**

**6.** **Is WebAssembly a good solution?**

**7. Bibliography**

**1. Project Description**

The aim of the project is to find a more efficient way of doing calculations/value formatting using WebAssembly. Currently, all the operations are done in the backend, causing performance issues (delayed response, loading time).

Our goal is to see if we can do all these stuffs without using backend API calls to reduce the loading end execution time.

We implemented two separate functions with the same logic both in the backend and in WebAssembly and performed numerous tests to analyze which one is more performant.

**2. Use Case**

The implementation and research were done using several logical and arithmetical operations, like addition, subtraction, multiplication, division, logical and relational operations and a special number formatting for two cases, *integer* number and *floating-point* number.

The float number formatting was done following these requirements:

Table

Description automatically generated

Figure 1

And the integer formatting was done following these requirements:

**Table

Description automatically generated**

Figure 2

For the logical and relational operations part, we implemented the basic logical expressions between two entities (ex. and, or, not) and the relational expressions >, <, <=, >=.

**3. WebAssembly**

* WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.
* In a Nutshell. WebAssembly has huge implications for the web platform, it provides a way to run code written in multiple languages on the web at near native speed, with client apps running on the web that previously couldn't have done so.
* The pros of WASM:
* Gives us access to a set of low-level building blocks.
* Browsers will understand the binary format, which means we will be able to compile binary bundles that compress smaller than the text JavaScript we use today.
* Smaller payloads mean faster delivery.
* RealTime access
* The most popular compiler toolchain to develop a WebAssembly module is ***Emscripten***, which was also used in our project.
* The compiled code itself was written using the C programming language:

Text

Description automatically generated

Figure 3- C functions

* Using emscripten the compilation is done from the terminal with the command: emcc [source\_file\_path]-o [dest\_file\_path] -s EXPORTED\_RUNTIME\_METHODS='ccall','cwrap']. This generates exported functions and runtime method (cwrap, ccall), a wasm binary file, a default .html, and .js files
* In the browser we have the html page running.
* The html page is connected to javascript, which will use the methods compiled real-time.

Graphical user interface, diagram, Teams

Description automatically generated

Figure 4- WASM Flow

**4. ASP .NET API**

* To do some comparisons between WebAssembly and the backend we used C# and ASP.NET.
* One of the functions in the API looks like: Text

  Description automatically generated

Figure 5- API functions

* For the functionality of the method we use the following tags:

A screenshot of a computer

Description automatically generated with medium confidence

**5. JavaScript**

* For the connection between frontend and backend we used Javascript. JavaScript is used for dynamic interaction with the user and to connect with webAssembly binary files.
* Our C function is called by the *Module ccall* and the input is taken from the input cells from html.

Text

Description automatically generated

Figure 6-JavaScript function

* For the function which connects the API to the UI, we used the *fetch* API with values sent in json format

Text

Description automatically generated

Figure 7- JavaScript API function

**6. Is WebAssembly a good solution?**

That is a good question.

With WASM it was possible to achieve very good performance in terms of execution time compared to the classic backend api.

The following requirements were all achievable:

* Pass Objects as parameters in WebAssembly
* Modify Object properties
* Communication between 2 WebAssemblies

In conclusion, in the long term WebAssembly could be a good solution, but

a lot of time investment and documentation is needed on this subject.

**7. Bibliography**

* <https://www.wasm.builders/shravi_inamdar/wanna-build-calculator-using-c-and-compile-it-to-webassembly--1cf7>
* <https://livebook.manning.com/book/webassembly-in-action/b-ccall-cwrap-and-direct-function-calls/v-7/41>
* <https://1bestcsharp.blogspot.com/2017/01/javascript-addition-subtraction-multiplication-division.html>
* <https://www.youtube.com/watch?v=_pHgILVlx3c>