Documentation (frontend en backend)

Write the documentation in english so that more people can understand it.
Further more most programming terms dont have a clear definition in the dutch dictionary.

C# code conventions

Explicit typing

Make use of explicit typing, do this so that you can descern the type that is being used at a glance.
This inproves readeblilty of the code.

```
int orderNumber = 10
string orderName = "Henks Pizza"

CustomerOrder? order = GetOrder(ordernumber);
decimal customerBalance = GetCustomerBalance();

// Don't
var customerOrder = new CustomerOrder();

// do
Pizza customerOrder = new CustomerOrder();

// allowed
CustomerOrder customerOrder = new();
```

No naked ifs

Naked ifs are the practice of not using curly brackets when you are only executing one line of code after the if statement (see example).

Diff unfriendly
When you are commit changes and have people look at pull request, you want to introduce as little clutter as possible. When you add an extra line to the naked, if you introduce a lot respective changes in the file.

Accidental logic changes
You can accidentally add a new line to a naked if and logic is not executed the correct way.
This can lead to unneccery bugs.

```
bool condition = true

// Don't
if(condition)
        Console.Writeline("Don't")

// Do
```

```
if(condition)
{
        Console.Writeline("Do")
}
```

Naming bools

When naming booleans don't use the true state as perspective, so name IsX and not IsNotX.
This increases clarity of the function of the variable.

```
// Don't
bool isNotLoading = false
// Don't
bool loading = true
// Do
bool isLoading = true
```

Don't return direct into return statement.

Don't return the result of a function or a statement direct into the return statement.
This increases readability of the function and the possibility to debug the function.

```
// Don't
public string ExampleFunction()
{
        return GetCoolString();
}

// Do
public string CreateCoolString()
{
     string result = GetCoolString();
     return result;
}
```

The most complex variable left

This for increasing readability.

```
public void TestFunc()
{
        List<int> ids = new List<int>() {0,1,2,4};

        // Don't
        if(0 =≠ ids.Count ) {
                // code ..
        }

        // Do
```

```
        if(ids.Count ≠ 0) {
                // code ..
        }
}
```

Try to reduce nesting

Try to reduce the amount of nesting in your code.
If there is a lot of nesting in your code, it is difficult to see what is going on.
Make use of Extraction and inversion to reduce nesting

"If you need more than 3 levels of **indentation**,
you're **screwed** anyway,
and you should **fix** your program." - Linux kernel style guide

Don't:

```
public int Calcutate(int bottom, int top)
{
        if(top > bottom)
        {
                int sum = 0
                for (int number = bottom; number ≤ top; number++)
                {
                        if (number & 2 == 0)
                        {
                                sum += number;
                        }
                }

                return sum;
        }
        else
        {
                 return 0
        }

}
```

Do:

```
public int Calcutate(int bottom, int top)
{
        if(top < bottom)
        {
                return 0;
        }

        int sum = 0
        for (int number = bottom; number ≤ top; number++)
```

```
        {
                sum += filterNumber(number);
        }

        return sum;
}

private int FilterNumber(int number)
{
        if (number & 2 == 0)
        {
                return number;
        }

        return 0;
}
```

Use IEnumbreble in public interface methods

Reduces the amount of conversions that need to take place if people want to make use of the functions.

```
// Don't
public void WithoutIEnumerable(List<string> collection) {
        // do something with collection
}

// Do
public void WithIEnumerable(IEnumerable<string> collection) {
        // do something with collection
}
```

Frontend conventions

Dont use the "Any" type

This makes the code less save and less predictable

Try to reduce the amount arrow functions

Use arrow functions only for annomynus functions.
The function key word uses less characters and is more clearer than the arrow variant.

```
// Don't
const calculate = (input1: number, input2: number): number => {
        // code here..
}
```

```
// Do
function calculate(input1: number, input2: number) -> number {
        // code here
}
```

Always specify the return type

This improves readability and allows you to see what kind of type a function returns at a glance.

```
// Don't
function calcutale(input1: number, input2: number) {
        // code here
        return 42
}

// Do

function calcutale(input1: number, input2: number) -> number {
        // code here
        return 42
}
```