howest
hogeschool

Web Technology: Typescript

Dieter Mourisse

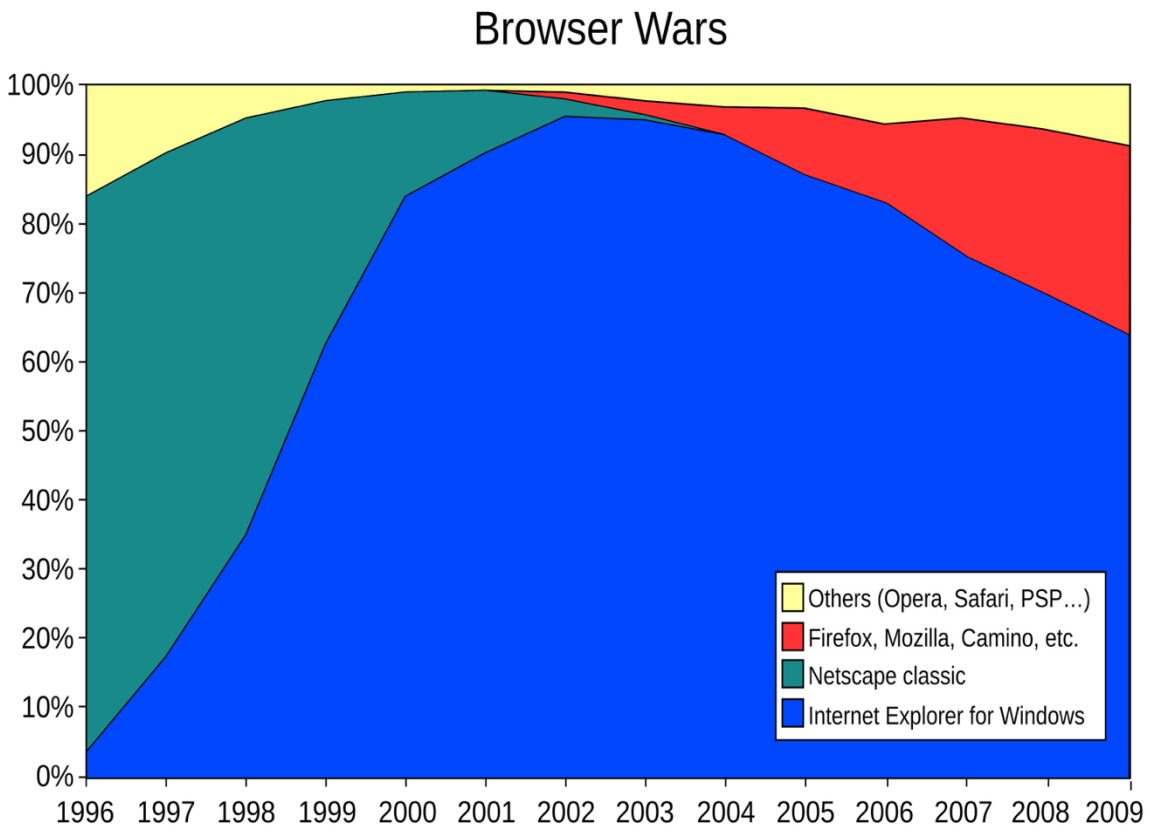
howest
hogeschool

History of JavaScript

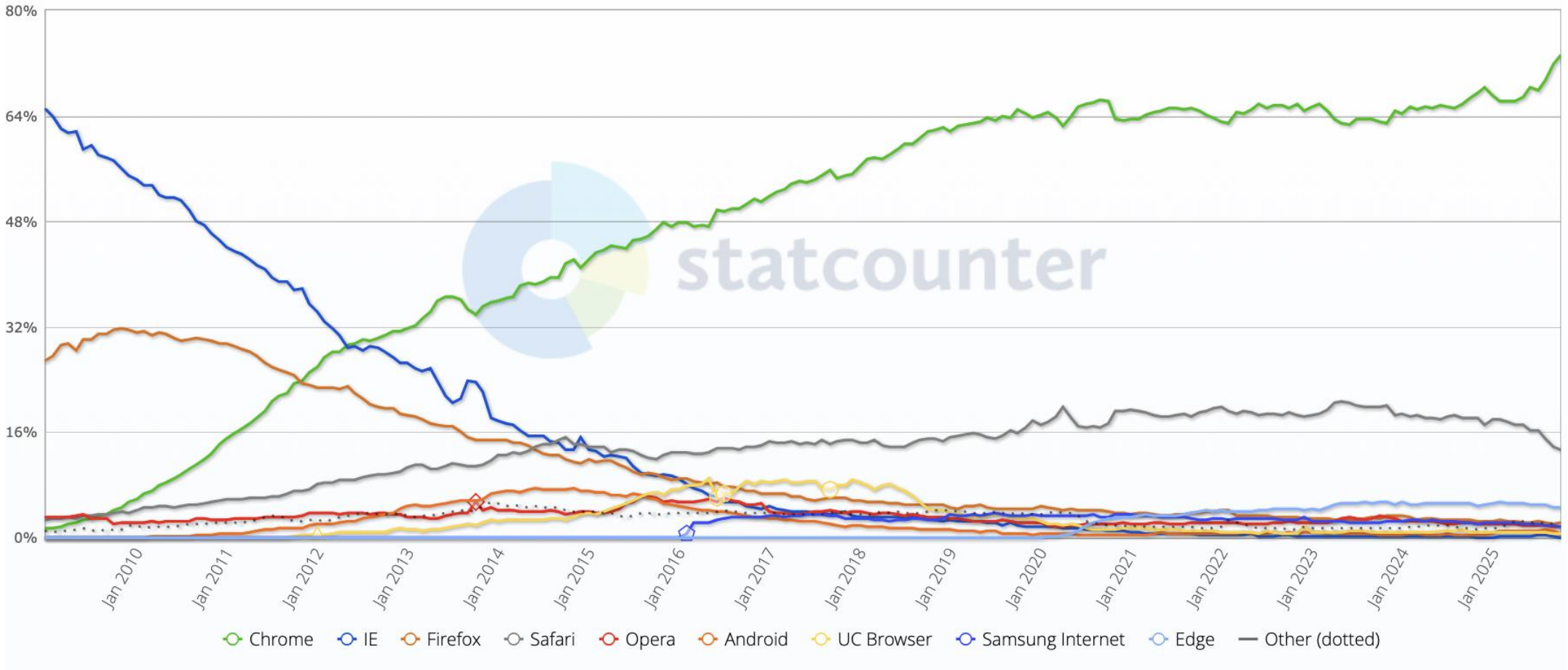
Some history

Netscape Navigator (1994 - 2008)

Some history



Some history



Some history

During these formative years of the Web, web pages could only be static, lacking the capability for dynamic behaviour after the page was loaded in the browser.

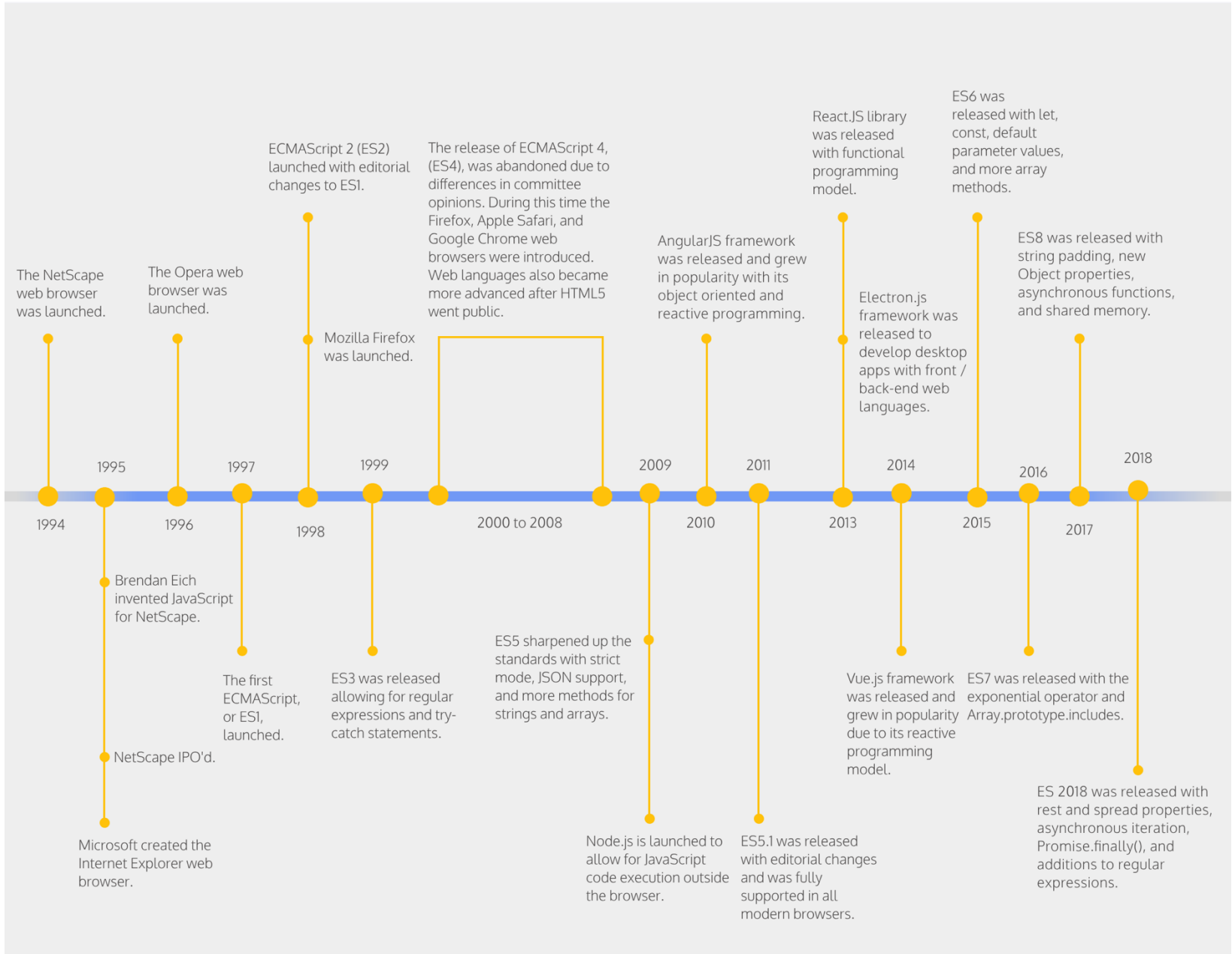
There was a desire in the flourishing web development scene to remove this limitation, so in 1995, Netscape decided to add a programming language to Navigator.

The goal was a "language for the masses", "to help nonprogrammers create dynamic, interactive Web sites".

The choice of the JavaScript name has caused confusion, implying that it is directly related to Java. At the time, the dot-com boom had begun, and Java was a popular new language, so Eich considered the JavaScript name a marketing ploy by Netscape.

Brendan Eich
creator of JavaScript

https://en.wikipedia.org/wiki/JavaScript


1994
The NetScape web browser was launched.
1995
Brendan Eich invented JavaScript for NetScape.
NetScape IPO'd.
Microsoft created the Internet Explorer web browser.
1996
The Opera web browser was launched.
1997
The first ECMAScript, or ES1, launched.
1998
ECMAScript 2 (ES2) launched with editorial changes to ES1.
Mozilla Firefox was launched.
1999
ES3 was released allowing for regular expressions and try-catch statements.
2000 to 2008
The release of ECMAScript 4, (ES4), was abandoned due to differences in committee opinions. During this time the Firefox, Apple Safari, and Google Chrome web browsers were introduced. Web languages also became more advanced after HTML5 went public.
2009
ES5 sharpened up the standards with strict mode, JSON support, and more methods for strings and arrays.
Node.js is launched to allow for JavaScript code execution outside the browser.
2010
AngularJS framework was released and grew in popularity with its object oriented and reactive programming.
2011
ES5.1 was released with editorial changes and was fully supported in all modern browsers.
2013
React.JS library was released with functional programming model.
2014
Electron.js framework was released to develop desktop apps with front / back-end web languages.
Vue.js framework was released and grew in popularity due to its reactive programming model.
2015
ES6 was released with let, const, default parameter values, and more array methods.
2016
ES7 was released with the exponential operator and Array.prototype.includes.
2017
ES8 was released with string padding, new Object properties, asynchronous functions, and shared memory.
2018
ES 2018 was released with rest and spread properties, asynchronous iteration, Promise.finally(), and additions to regular expressions.

howest
hogeschool

Typescript

Type sytems

Strongly Typed vs Weakly Typed

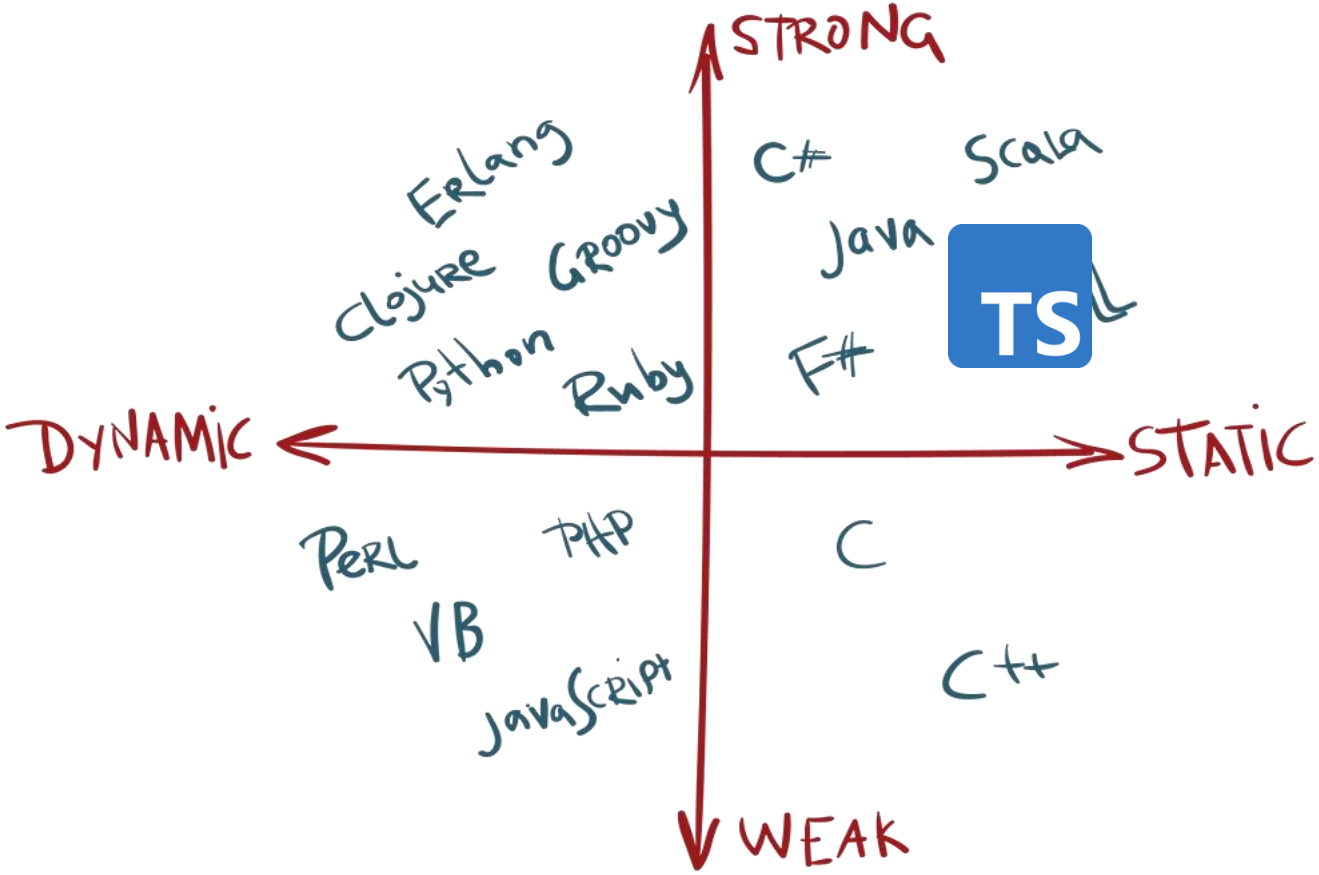
In a strongly typed language variables have a fixed type, and implicit type conversion is usually not allowed

In a weakly typed language implicit type conversion is allowed, and the language tries to guess what you mean by automatically converting types.

Statically typed vs Dynamically Typed

In a statically typed language the variable types are known and checked at compile time.

In a dynamically typed language the variable types are determined and checked at runtime.


STRONG
Erlang
C#
Scala
Clojure
Groovy
Java
TS
Python
Ruby
F#
DYNAMIC
STATIC
Perl
PHP
C
VB
JavaScript
C++
WEAK

Advantages of strongly typed languages

Early Error Detection

In strongly typed languages, many errors are caught at **compile time**, rather than at runtime. This leads to safer and more reliable code since you can identify and fix problems before running the application.

Better Tooling and IDE Support

Strong typing provides clear and explicit type information, which allows development tools (IDEs) to offer better support for **autocomplete, refactoring, and type-checking**. This makes development faster and reduces the chance of errors.

Improve Code readability and Maintainability

When types are explicitly defined, the code becomes more self-documenting, meaning that other developers (or even your future self) can understand the intended structure and purpose of variables, functions, and objects more easily.

Typescript

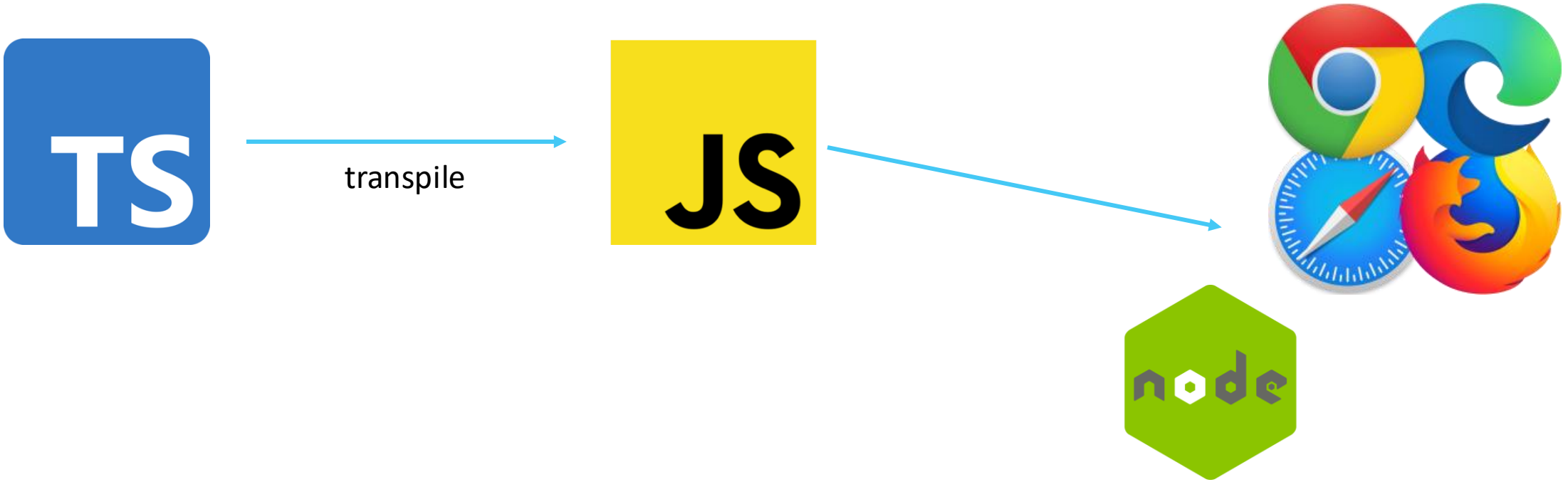
TypeScript stands in an unusual relationship to JavaScript. TypeScript offers all of JavaScript's features, and an additional layer on top of these: TypeScript's type system.

For example, JavaScript provides language primitives like string and number, but it doesn't check that you've consistently assigned these. TypeScript does.

This means that your existing working JavaScript code is also TypeScript code. The main benefit of TypeScript is that it can highlight unexpected behaviour in your code, lowering the chance of bugs.

Typescript

The browser and node.js do not understand TypeScript



Setup (node.js)

Terminal

```
bash-3.2$ npm install typescript
bash-3.2$ npx tsc –version
Version 5.6.3
bash-3.2$ npx tsc -init --module "ESNext"
```

This creates a *tsconfig.json* file. The presence of a *tsconfig.json* file in a directory indicates that the directory is the root of a TypeScript project. The *tsconfig.json* file specifies the root files and the compiler options required to compile the project. We also set the module system to ES Modules.

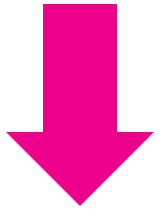
Transpiling (node.js)

script.ts

```
const myName : string = "Dieter";
console.log(`My name is ${myName}`);
```

Terminal

```
bash-3.2$ npx tsc script.ts
```



script.js

```
var myName = "Dieter";
console.log("My name is ".concat(myName));
```


howest hogeschool

Deno

Deno is an alternative to Node.js that supports TypeScript out of the box.

It was created by Ryan Dahl, who also created node.js and he sees it as a “second attempt” to design a JavaScript runtime.

Typescript files can be run by using the *deno run* command.

Terminal

```
bash-3.2$ deno run script.ts
```

howest
hogeschool

Typescript features

https://www.typescriptlang.org/docs/handbook

Basic typing

```
let age: number = 25;
let firstName: string = "Alice";
let isStudent: boolean = true;
```

Denoted after a colon (:)

Helps clarify the intended types of variables, providing better code understanding and catching potential errors during development

Basic typing: compound

Homogenous of variable length: Array

```
let numbers: number[] = [1, 2, 3, 4, 5];
let names: string[] = ["Alice", "Bob", "Charlie"];
```

```
let numbers: Array<number> = [1, 2, 3, 4, 5];
let names: Array<string> = ["Alice", "Bob", "Charlie"];
```

generic array type

Heterogeneous of fixed length: Tuple

```
let employee: [number, string] = [1, "Alice"];
let coordinates: [number, number] = [10, 20];
```


howest hogeschool

Basic typing: enum

```
enum Direction {
  Up,
  Down,
  Left,
  Right,
}


let playerDirection: Direction = Direction.Right;
```

Basic typing: multiple types

Any

```
let dynamicData: any = 10;
dynamicData = "Hello, TypeScript!";
```

Union types

```
// Union type example
let id: number | string;
id = 123; // Valid
id = "ABC"; // Valid
```

Interface

```typescript
// Interface defining a 'Person' object structure
interface Person {
  name: string;
  age: number;
  greet: () => void;
}

// Implementing the 'Person' interface
let user: Person = {
  name: "Alice",
  age: 25,
  greet() {
    console.log(`Hello, I'm ${this.name}!`);
  },
};

user.greet(); // Output: "Hello, I'm Alice!"
```

Optional properties

```
interface Person {
  name: string;
  age?: number;
}

const person: Person = {
  name: "Alice"
}
```

The question mark can be used for optional properties.

Types by Inference

Typescript will often be able to infer the types of a variable automatically withouth you having to explicitly set it.

TypeScript knows the JavaScript language and will generate types for you in many cases. For example in creating a variable and assigning it to a particular value, TypeScript will use the value as its type.

```
let helloWorld = "Hello World";
        let helloWorld: string
```

Good practice to always set it explicitly.

Function types

```
function add(x: number, y: number): number {
  return x + y;
}

let myAdd = function (x: number, y: number): number {
  return x + y;
};
```

We can add types to each of the parameters and then to the function itself to add a return type. TypeScript can figure the return type out by looking at the return statements, so we can also optionally leave this off in many cases.

howest
hogeschool

DOM Manipulation

Project structure

Project structure

Terminal

```
bash-3.2$ deno bundle --sourcemap –output assets/js/main.js ts/main.ts
```

will transpile ts files to 1 single js file and move it to the *assets/js* directory

Project structure


```
<script src="assets/js/main.js" defer></script>
```

Project structure

Debugging information for developper tools

Line number in *ts* file

```
16:49:22.307 ▶ Uncaught (in promise) TypeError: can't access property "content",  main.ts:14:33
               $ALBUMTEMPLATE is null
                 getAlbumTemplate main.ts:14
                 fillAlbums       main.ts:24
                 init             main.ts:30
                 async*           main.ts:34
               [Learn More]
```


howest hogeschool

querySelector

querySelector returns an element of type *Element | null*

```
const $element : Element | null = document.querySelector("#album-template");
```

Can be casted to a more specific element by using *as*

```
const $element : HTMLTemplateElement = document.querySelector("#album-template") as HTMLTemplateElement;
```

Full list of all HTML Elements:

https://developer.mozilla.org/en-US/docs/Web/API/HTML_DOM_API#html_element_interfaces_2

querySelector using generics (not recommended)

querySelector also supports generics to specify the type

```
const $element : HTMLTemplateElement | null = document.querySelector<HTMLTemplateElement>("#album-template");
```

To avoid the *null* type, the null assertion operation (*!*) can be used

```
const $element : HTMLTemplateElement = document.querySelector<HTMLTemplateElement>("#album-template")!;
```

howest
hogeschool

Events

List of Event Types

https://developer.mozilla.org/en-US/docs/Web/API/Event#interfaces_based_on_event

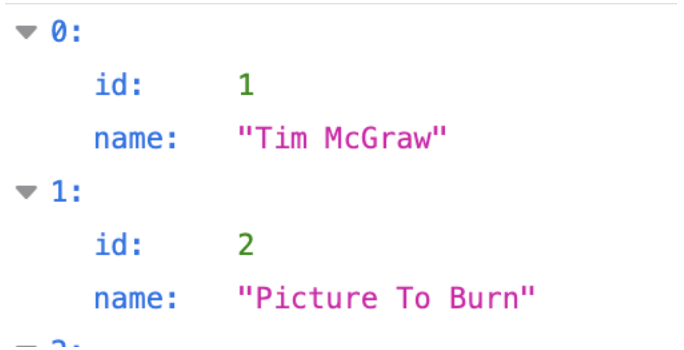
howest
hogeschool

Fetch

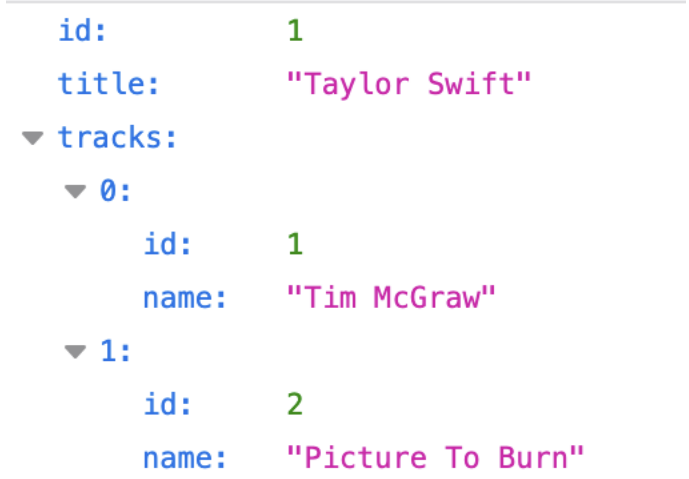
Create interface for json server response

/albums

/songs



/albums/{id}




```
interface Album {
  id: number,
  title: string,
  release_date: string
}
```

```
interface Song {
  id: number,
  name: string
}
```

```
interface AlbumDetails {
  id: number,
  title: string,
  tracks: Song[]
}
```


howest hogeschool

Do request

```
async function getAlbum(id: number) : Promise<Album> {
  const response : Response = await fetch(`${BASEURL}/albums/${id}`);
  const album : Album = await response.json();
  return album;
}
```