



BAITUSSALAM
—TECH PARK—



Class Agenda

**Application Performance
Optimization, Cross Browser
Compatibility, Generators and
Decorators**

Topics We Will Cover

1. How to optimize frontend performance
2. Cross-browser compatibility techniques with Babel
3. Performance profiling and debugging tools

Importance of Performance and Compatibility

- **Faster loading times improve user experience.**
- **Cross-browser compatibility ensures accessibility for all users.**
- **Profiling and debugging help maintain high performance.**

Optimizing Frontend Performance

- **Minification:** Reduce file size by removing unnecessary characters.
- **Image Optimization:** Compress images without losing quality.
- **Lazy Loading:** Load content only when needed.
- **Caching:** Store assets locally to reduce server requests.

Cross-Browser Compatibility

Ensuring Accessibility for All Users

- Vendor prefixes for CSS properties.
- Use Babel for JavaScript transpilation.

Vendor Prefixes for CSS Properties

CSS vendor prefixes, sometimes called **CSS browser prefixes**, are a way for browser makers to add support for new CSS features before those features are fully supported in all browsers.

Android:

-webkit-

Chrome:

-webkit-

Firefox:

-moz-

Internet Explorer:

-ms-

iOS:

-webkit-

Opera:

-o-

Safari:

-webkit-

CSS Browser Prefixes Examples

```
style.css > body
1  .btn {
2      -webkit-transition: all 4s ease;
3      -moz-transition: all 4s ease;
4      -ms-transition: all 4s ease;
5      -o-transition: all 4s ease;
6      transition: all 4s ease;
7  }
```


What is Babel

Babel is a JavaScript transpiler that converts edge JavaScript into plain old ES5 JavaScript that can run in any browser (even the old ones).

It makes available all the syntactical sugar that was added to JavaScript with ES6 and ESNext, including classes, fat arrows, multiline strings, generators, and many more.

```
// Babel Input: ES2015 arrow function
[1, 2, 3].map(n => n + 1);

// Babel Output: ES5 equivalent
[1, 2, 3].map(function(n) {
  return n + 1;
});
```

JavaScript Classes in Babel Compiler

Private Class with Babel Compiler

```

1 class BankAccount {
2   #balance
3
4   constructor(initialBalance) {
5     this.#balance = initialBalance;
6   }
7
8   getBalance() {
9     return this.#balance;
10  }
11
12  deposit(amount) {
13    this.#balance += amount;
14  }
15 }

```

```

< const account = new BankAccount(100);

```

```

1 function _classPrivateFieldInitSpec(e, t, a) { _checkPrivateRedeclaration(e,
t), t.set(e, a); }
2 function _checkPrivateRedeclaration(e, t) { if (t.has(e)) throw new
TypeError("Cannot initialize the same private elements twice on an object");
}
3 function _classPrivateFieldGet(s, a) { return s.get(_assertClassBrand(s,
a)); }
4 function _classPrivateFieldSet(s, a, r) { return s.set(_assertClassBrand(s,
a), r), r; }
5 function _assertClassBrand(e, t, n) { if ("function" == typeof e ? e === t :
e.has(t)) return arguments.length < 3 ? t : n; throw new TypeError("Private
element is not present on this object"); }
6 var _balance = /*#__PURE__*/new WeakMap();
7 class BankAccount {
8   constructor(initialBalance) {
9     _classPrivateFieldInitSpec(this, _balance, void 0);
10    _classPrivateFieldSet(_balance, this, initialBalance);
11  }
12  getBalance() {
13    return _classPrivateFieldGet(_balance, this);
14  }
15  deposit(amount) {
16    _classPrivateFieldSet(_balance, this, _classPrivateFieldGet(_balance,
this) + amount);
17  }
18 }
19 const account = new BankAccount(100);

```

Performance Profiling and Debugging Tools

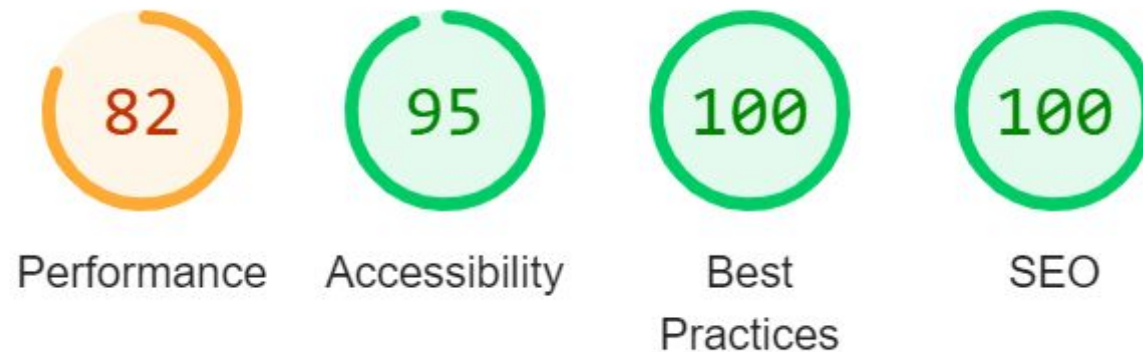
Chrome DevTools: Performance tab, Lighthouse.

Firefox Developer Tools: Performance analysis.

- **Inspect network requests.**
- **Analyze rendering performance.**
- **Use Lighthouse for audits.**

Lighthouse
















Lighthouse is an open-source, automated tool for improving the performance, quality, and correctness of your web apps.



Network Tab

As a user, we can use it for **resource analysis**, meaning how much of our data and other resources are being consumed.

It can also show how many requests and responses are coming and going from the source to the destination.

Name	Status	Type	Initiator	Size	Time
 fleck-worklet.bundle.min.js	200	script	Other	0 B	19.15 s
 css-tricks.com	304	document	Other	192 B	677 ms
 style.css?ver=cache_bust=1721331254185	200	stylesheet	css-tricks.com/:63	(memory cache)	0 ms
 mediaelementplayer-legacy.min.css	200	stylesheet	css-tricks.com/:330	(memory cache)	0 ms
 wp-mediaelement.min.css	200	stylesheet	css-tricks.com/:331	(memory cache)	0 ms
 breeze-prefetch-links.min.js?ver=2.1.11	200	script	css-tricks.com/:355	(memory cache)	0 ms
 wp-polyfill-inert.min.js	200	script	css-tricks.com/:356	(memory cache)	0 ms
 regenerator-runtime.min.js	200	script	css-tricks.com/:357	(memory cache)	1 ms
 wp-polyfill.min.js	200	script	css-tricks.com/:358	(memory cache)	0 ms
 hooks.min.js	200	script	css-tricks.com/:359	(memory cache)	0 ms
 i18n.min.js	200	script	css-tricks.com/:360	(memory cache)	0 ms
 i18n-loader.js?minify=true&ver=b5d2a25b...	200	script	css-tricks.com/:366	(memory cache)	0 ms
 83ad7bf5c295733f8dfc1b5003cc573a?s=80...	200	png	css-tricks.com/:497	(memory cache)	0 ms
 1a1b5ca4b139c0c8caaea26ef437d943?s=8...	200	jpeg	css-tricks.com/:555	(memory cache)	0 ms
 uQbduxW-_400x400-80x80.jpg	200	jpeg	css-tricks.com/:738	(memory cache)	0 ms
73 requests 380 kB transferred 3.0 MB resources Finish: 9.36 s DOMContentLoaded: 1.33 s Load: 7.60 s					

ES6 - Generators

- **Special type of function that can pause and resume execution.**
- **Uses function* syntax to define.**
- **Provides a more efficient way to handle iteration.**
- **Simplify asynchronous programming.**
- **Handle infinite sequences.**

Generator Basic Example

- Call generator function to get an iterator
- Use `.next()` method to get values

```
JS app.js > ...  
1  function* simpleGenerator() {  
2    yield "Hello"  
3    yield "World"  
4  }  
5  
6  const generator = simpleGenerator()  
7  
8  console.log("value", generator.next().value) // Hello  
9  console.log("value", generator.next().value) // World  
10
```


yield keyword

The **yield** keyword pauses generator function execution and the value of the expression following the yield keyword is returned to the generator's caller.

yield has 3 methods

1. next()
2. return()
3. throw()

```
function* countUpTo(max) {  
  let count = 0  
  
  while (count < max) {  
    yield count++  
  }  
}  
  
const counter = countUpTo(3)  
console.log(counter.next().value) // Output: 0  
console.log(counter.next().value) // Output: 1  
console.log(counter.next().value) // Output: 2
```

Passing parameter to yield

```
11 function* countUpTo(max) {
12     let count = 0
13
14     while (count < max) {
15         let increment = yield count
16         if (increment !== null) {
17             count = count + increment
18         } else {
19             count++
20         }
21     }
22 }
23
24 const counter = countUpTo(6)
25 console.log(counter.next().value) // Output: 0
26 console.log(counter.next(2)) // passing parameter to yield
27 console.log(counter.next().value) // Output: 3
28
```

Typescript Decorators

In JavaScript/Typescript, **decorators** are functions that can modify classes, methods, properties, or even parameters. They provide a way to add behavior or metadata to various parts of your code without altering the source code.

- Simplify code by adding reusable annotations
- Enhance readability and maintainability
- Common in frameworks like Angular for dependency injection and metadata

Types of Decorators

- Class Decorators
- Method Decorators
- Accessor Decorators
- Property Decorators
- Parameter Decorators

Simple Decorator

Decorators are typically used with classes and prefixed with the **@ symbol**

```
function logger(target: any, key: string, descriptor: PropertyDescriptor): PropertyDescriptor {  
    const originalMethod = descriptor.value; // Save the original method  
  
    // Redefine the method with custom behavior  
    descriptor.value = function (...args: any[]) {  
        console.log(`Before ${key} is called`);  
        const result = originalMethod.apply(this, args);  
        console.log(`After ${key} is called`);  
        return result;  
    };  
  
    return descriptor;  
}
```

```
class User {  
    constructor(private name: string, private age: number) {}  
  
    greet() {  
        console.log("start: greet")  
        console.log(`Hello, my name is ${this.name}.`)  
        console.log("end: greet")  
    }  
  
    printAge() {  
        console.log("start: printAge")  
        console.log(`I am ${this.age} years old`)  
        console.log("end: printAge")  
    }  
}  
  
const user = new User("Ron", 25)  
user.greet()  
user.printAge()
```

Decorators Example

The End

