Angular 10 Core

June 20th , 2020

- Sergiy Morenets, 2020

DEVELOPER 16 YEARS

TRAINER 7 YEARS

WRITER 4 BOOKS

- Sergiy Morenets, 2020

FOUNDER

ITSimulator

SPEAKER

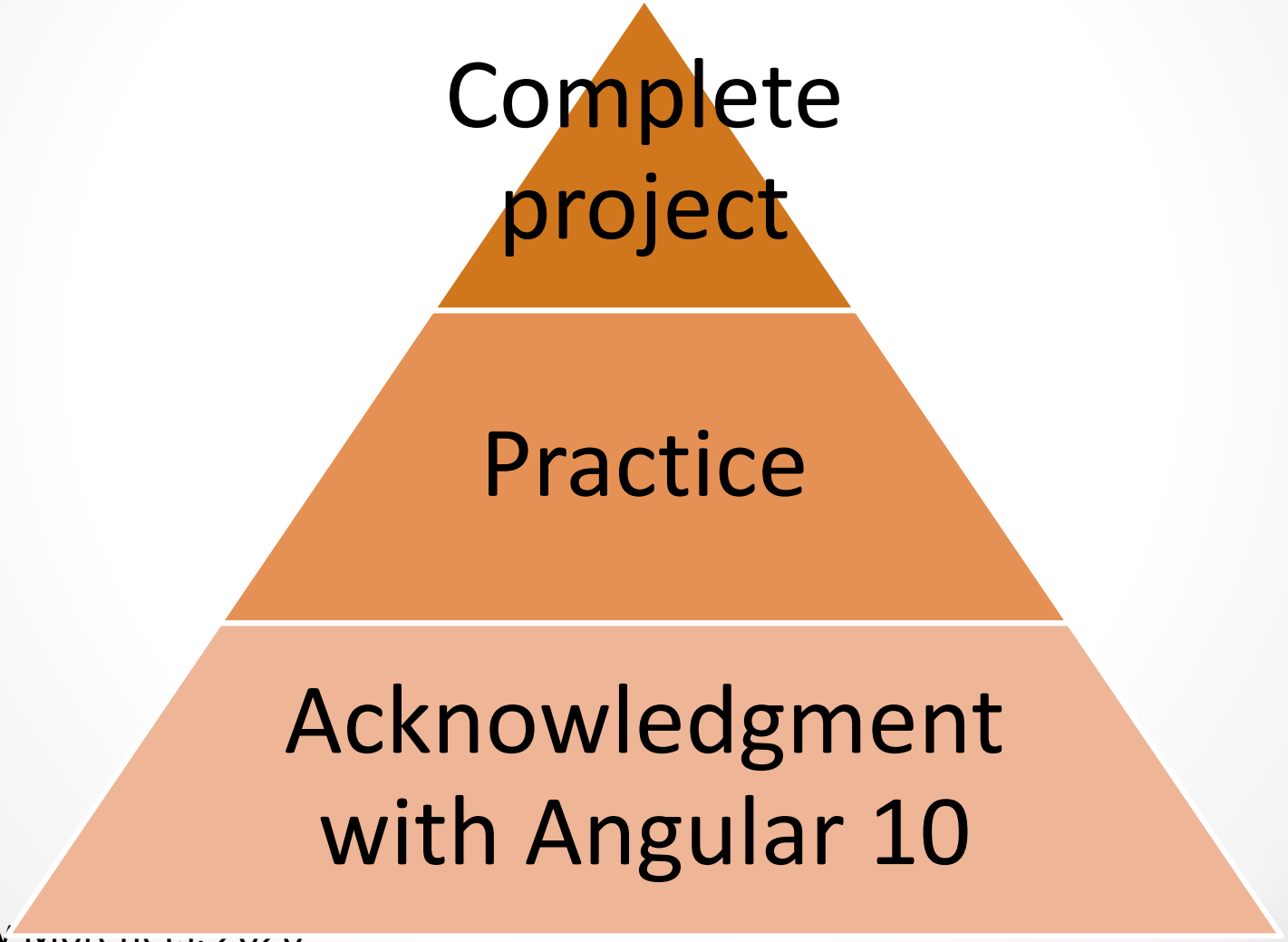
JAVA DAY
MINSK 2013

Dev(Talks):

Java
frameworks
day


@sergey-morenets
DEVOX
POLAND

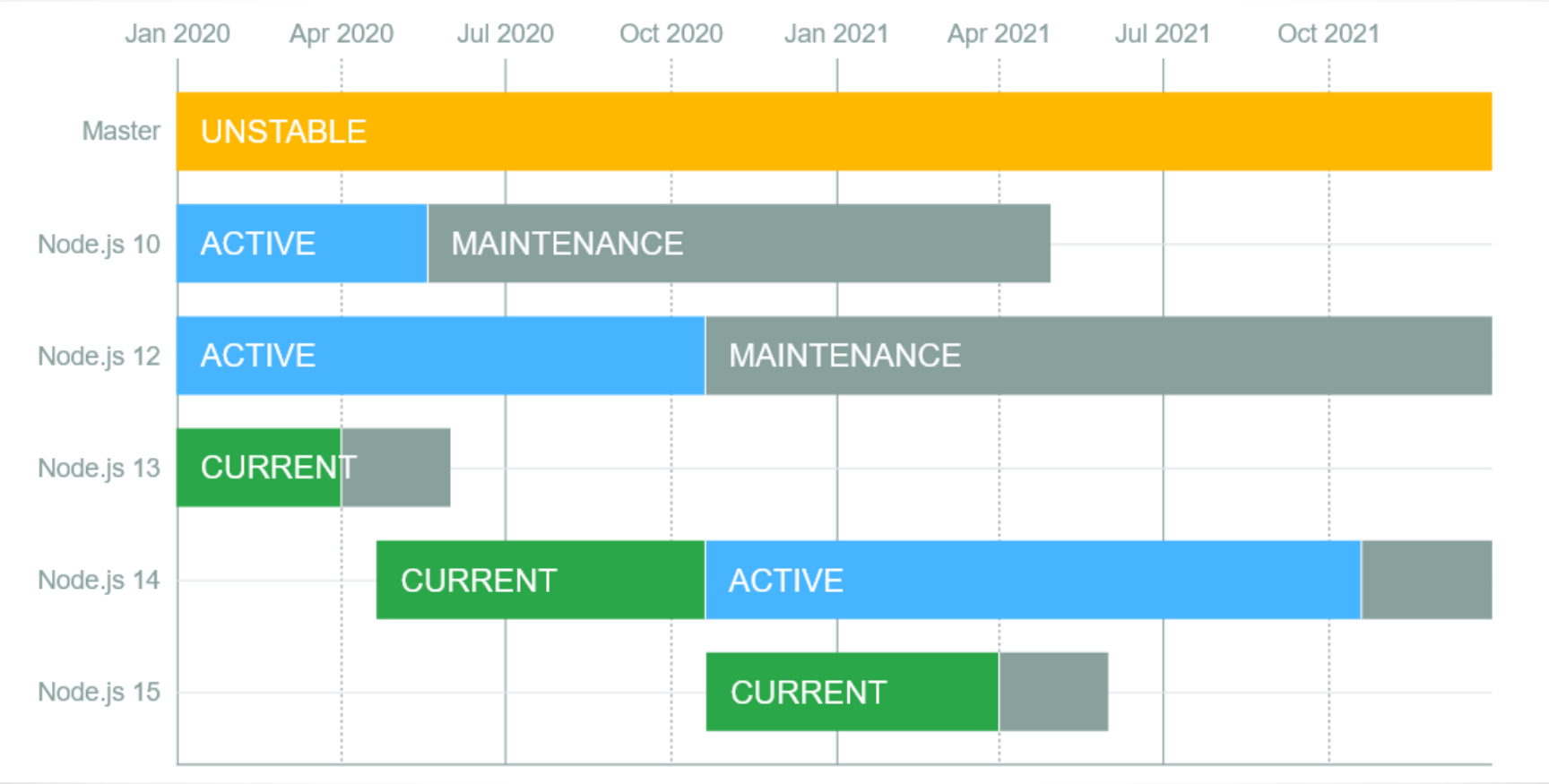
Goals


Complete project
Practice
Acknowledgment with Angular 10

Agenda

- ECMAScript versions
- TypeScript. OOP and typing
- Module bundlers. Webpack
- Angular CLI
- Components and templates
- Services and DI
- Pipes
- Form validation
- Directives
- Ivy renderer
- Working with HTTP

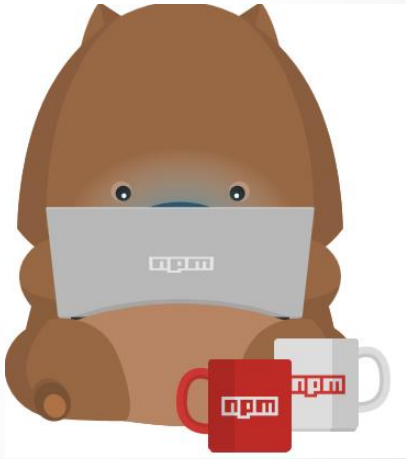
NodeJS. Versions



- Sergiy Morenets, 2020

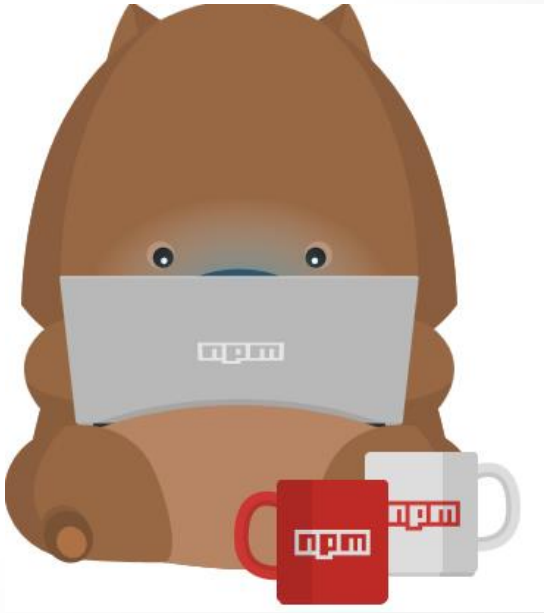
NPM

- Package manager for JavaScript
- Bundled together with **Node**
- Package(or module) is directory with files
- Splits prod and dev dependencies
- **package.json** contains dependencies and scripts
- Stores dependencies in **node_modules** folder
- Hosts over 250 000 packages



package-lock.json

- Automatically generated during any operation that modified node_modules folder or package.json
- Any install operation will produce the same dependency trees
- Should be committed into CVS repository
- Can’t be published



NPM. Dependency versions

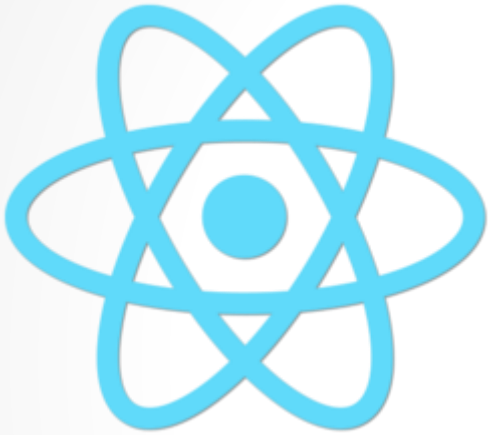
Example	Description
1.0	Exactly version 1.0
>=1.0	1.0 or greater
~1.0	The same as 1.1.x
1.1.x	1.1.0, 1.1.1, etc
*	Any version
1.0 – 5.0	Any version between 1.0 and 5.0
^1.0	Any version that doesn't modify left-most non zero digit, for example 1.2 or 1.5 but not 2.0
< 2.0 || > 5.0	Any version less than 2.0 or greater than 5.0

Task #1. Installation.

1. Install **Node**/npm. Check **NPM** version using “npm –v” command.
2. Install **Python**(2.7 or higher)
3. Install **Git**
4. Create **new folder** and run *npm init* command in this folder
5. Install *jquery* package by running *npm install --save-dev jquery* command

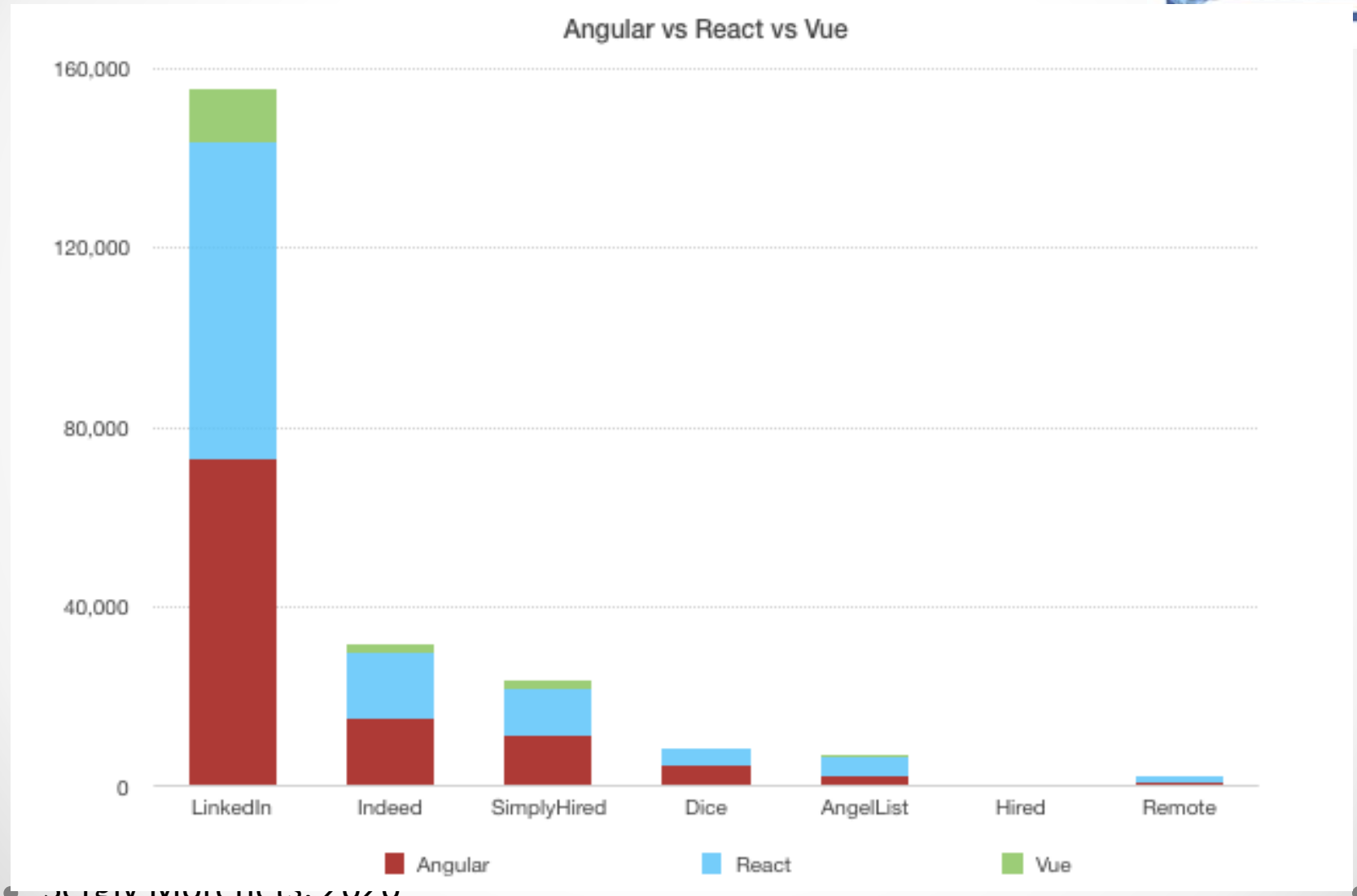
- Sergiy Morenets, 2020

Frameworks



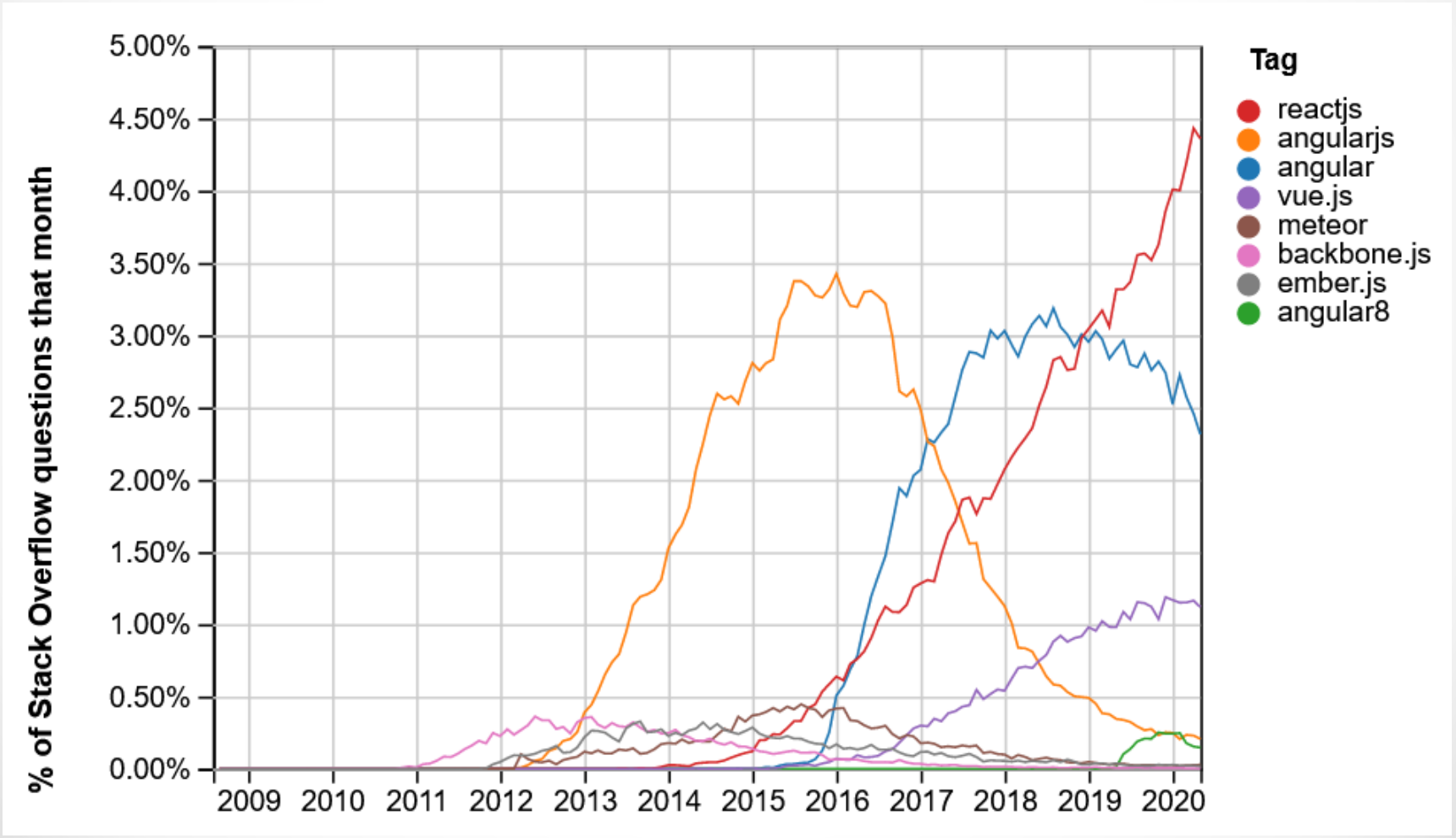
- Sergiy Morenets, 2020

Job markets. Popularity.2019



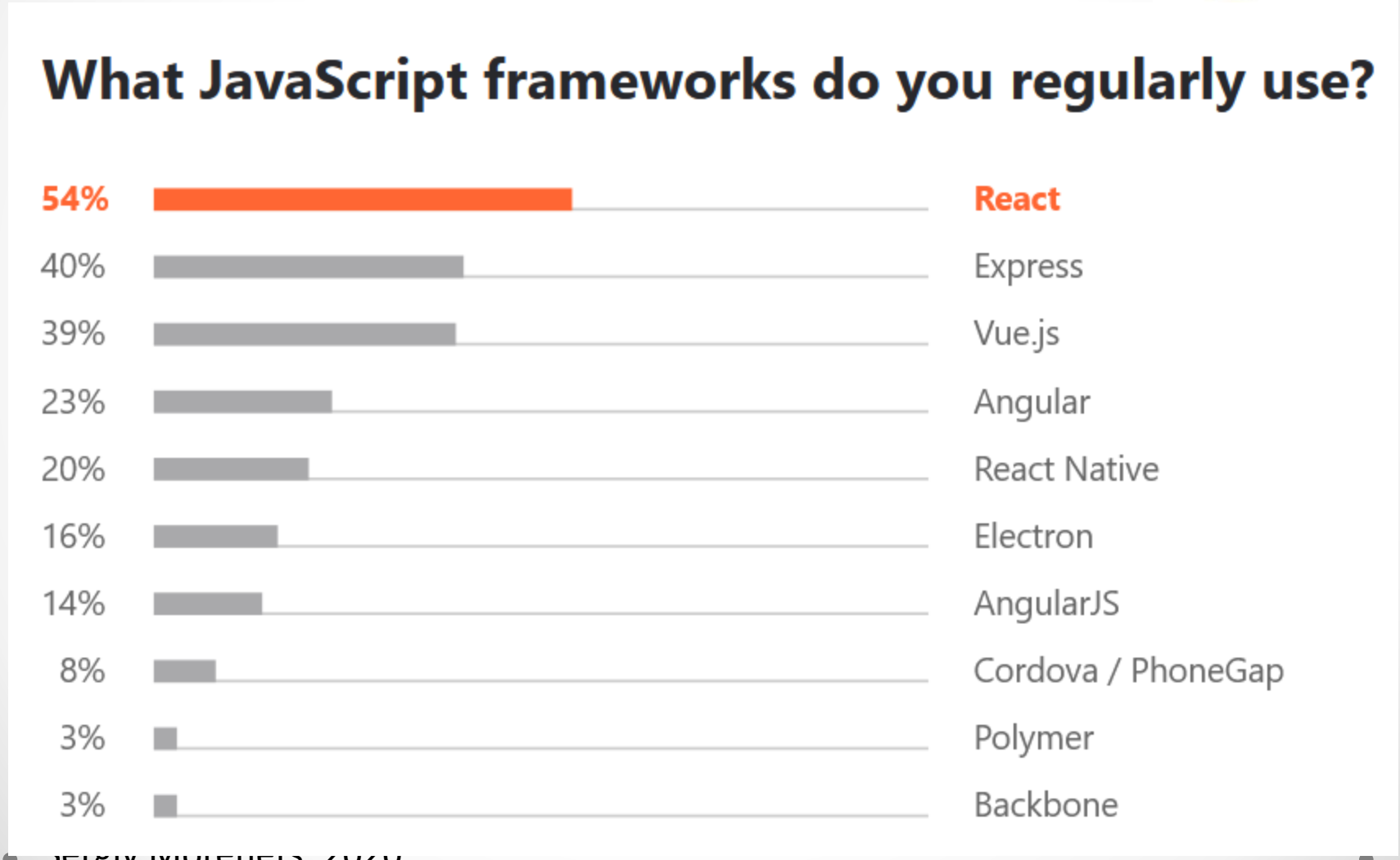
- Sergiy Morenets, 2020

StackOverflow trends

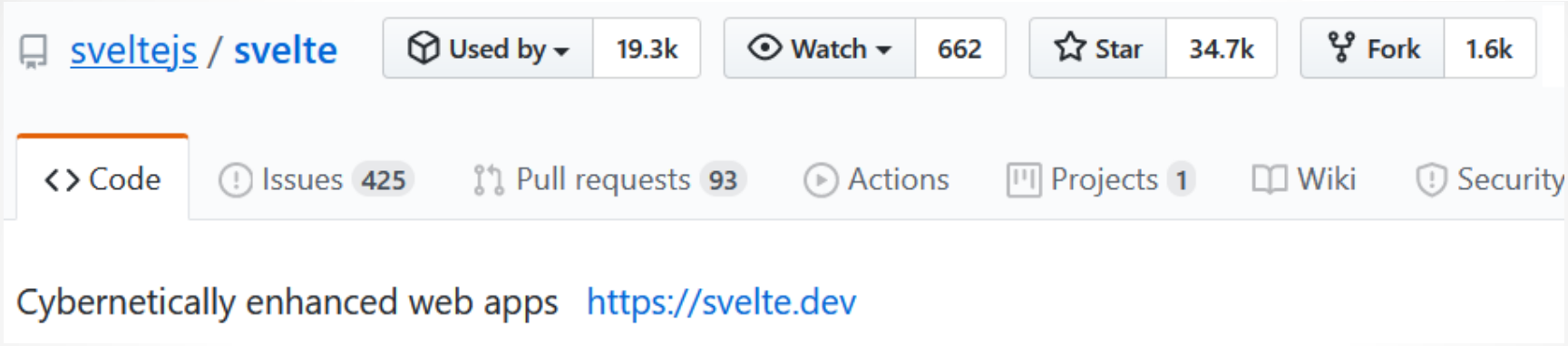


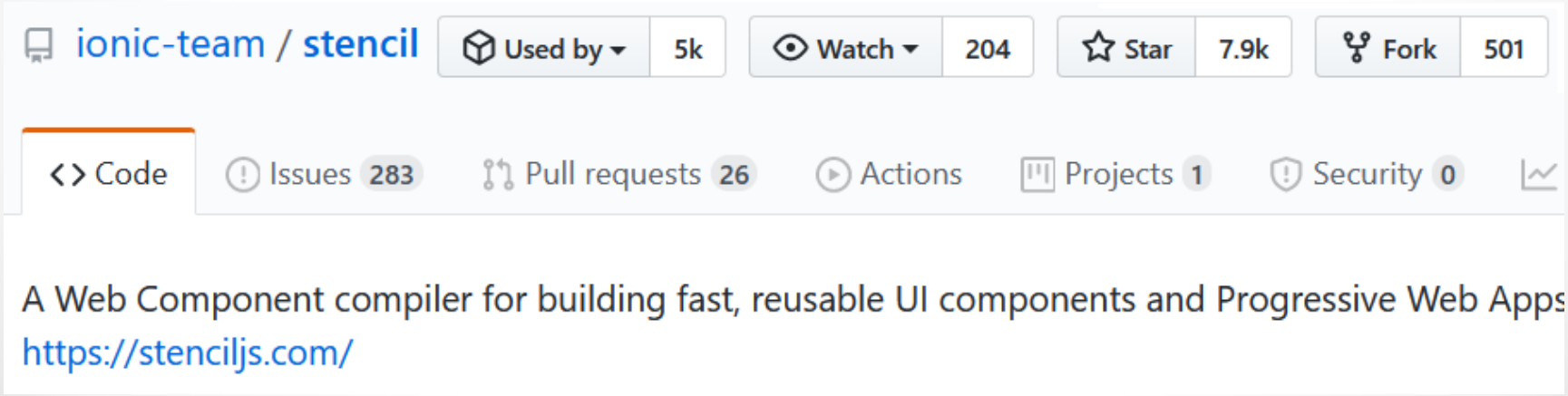
- Sergiy Morenets, 2020

Jetbrains. 2019 survey



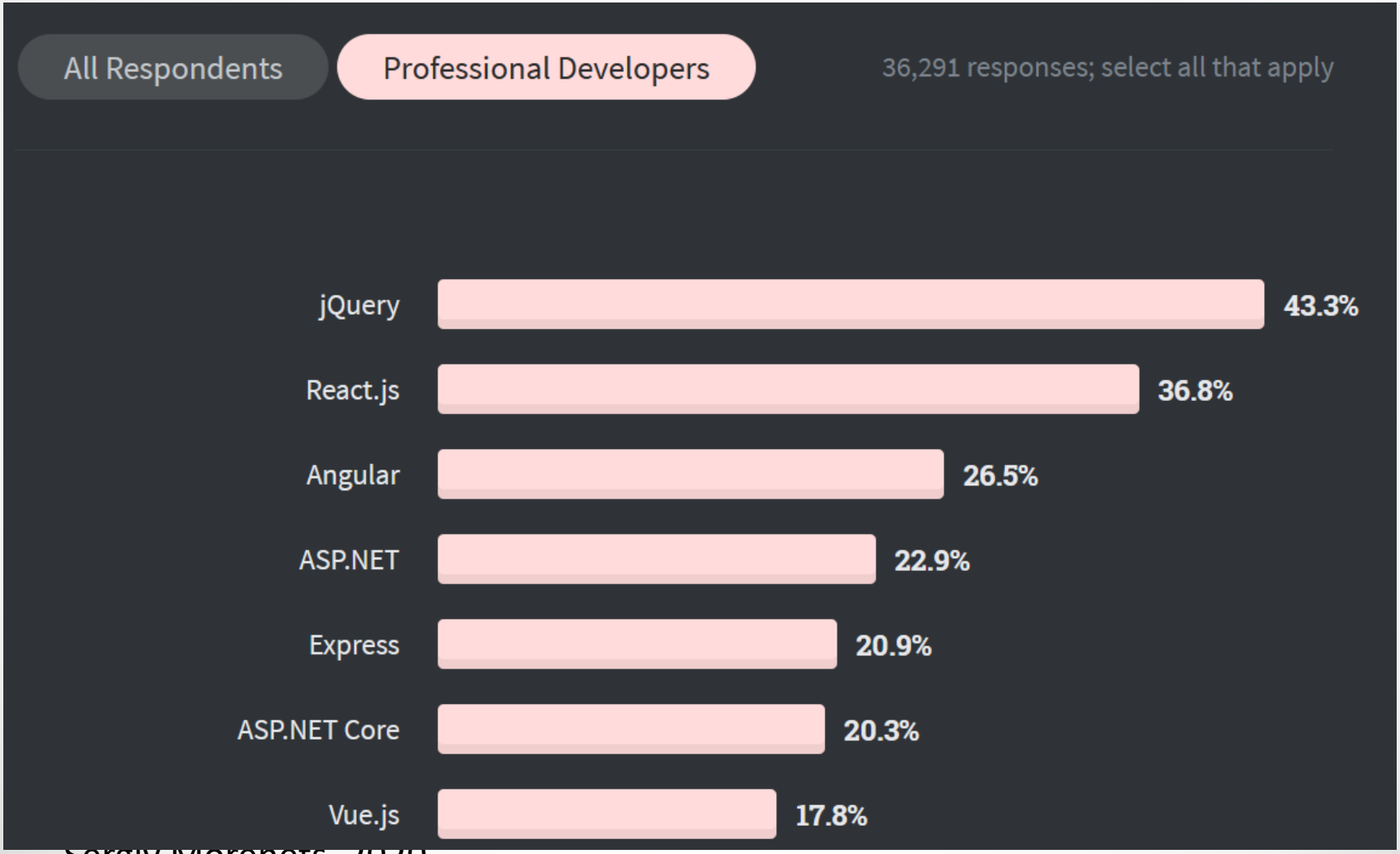
New players



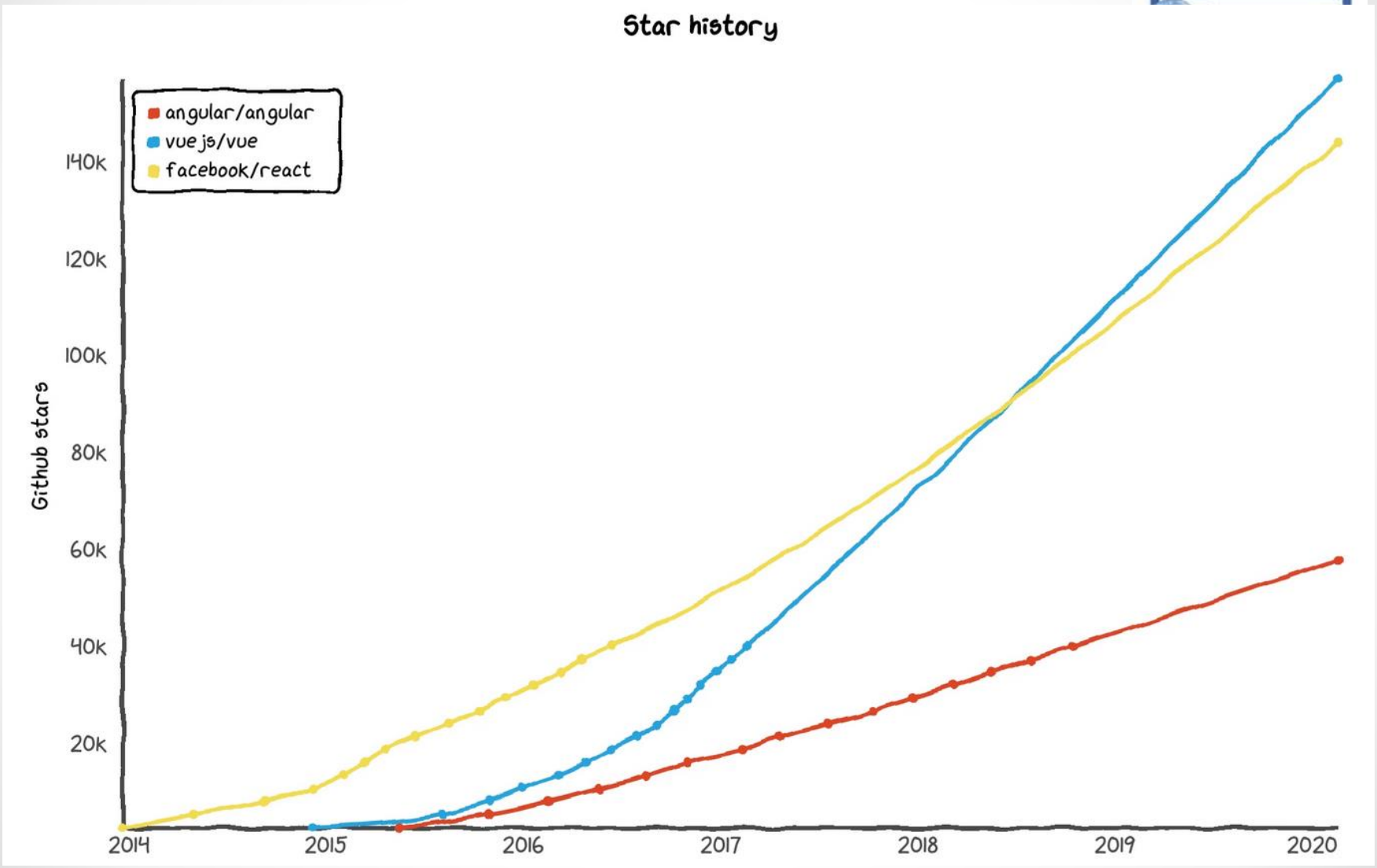


- Sergiy Morenets, 2020

Stackoverflow. 2020 survey



- Sergiy Morenets, 2020



- Sergiy Morenets, 2020

JS frameworks. Statistics

	Angular	React	Vue
# Watchers	3.2k	6.6k	6.0k
# Stars	57k	144k	157k
# Forks	15.9k	27.6k	23.7k
# Contributors	1,089	1,361	289

- Sergiy Morenets, 2020

Languages history

- JavaScript(previously Mocha, LiveScript) developed in 1995 for Netscape Communicator
- Microsoft developed Jscript for Internet Explorer
- Works on ECMA standard began in 1996
- Standardized as ECMAScript in 1997
- ECMAScript implementations include ActionScript/JScript
- ECMAScript 3 released in 1999
- ECMAScript 4 took almost 8 years of development and was finally scrapped
- ECMAScript 5 published in 2009

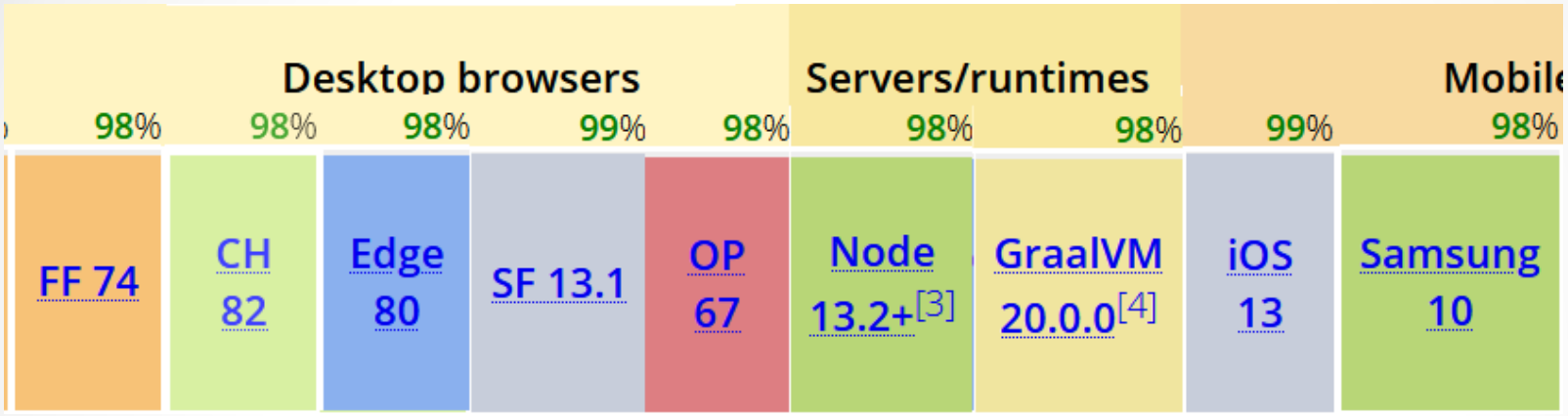
Languages history

- ECMAScript 6 or ES6 approved in June 2015
- ES6 requires transpiler(for example, **Traceur** or **Babel**)
- TypeScript released in 2012 with improvements to ES5
- ECMAScript 2016 includes software tools: Ecmarkup, Ecmarkdown, and Grammarkdown
- ECMAScript 2017 includes asynchronous functions, atomics, and Shared memory model
- ECMAScript 2018 includes new regular expression features and asynchronous iteration/generators
- ECMAScript 2019 includes optional catch binding and class fields

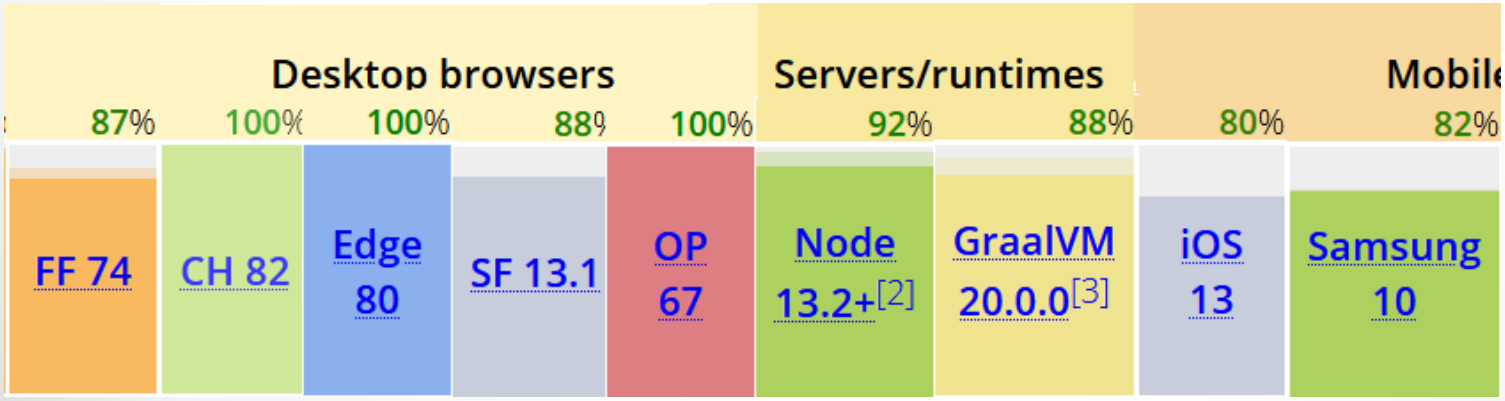
ECMAScript editions

Edition	Official name	Date published
ES9	ES2018	June 2018
ES8	ES2017	June 2017
ES7	ES2016	June 2016
ES6	ES2015	June 2015
ES5.1	ES5.1	June 2011
ES5	ES5	December 2009
ES4	ES4	Abandoned
ES3	ES3	December 1999
ES2	ES2	June 1998
ES1	ES1	June 1997

Browser support. 2020



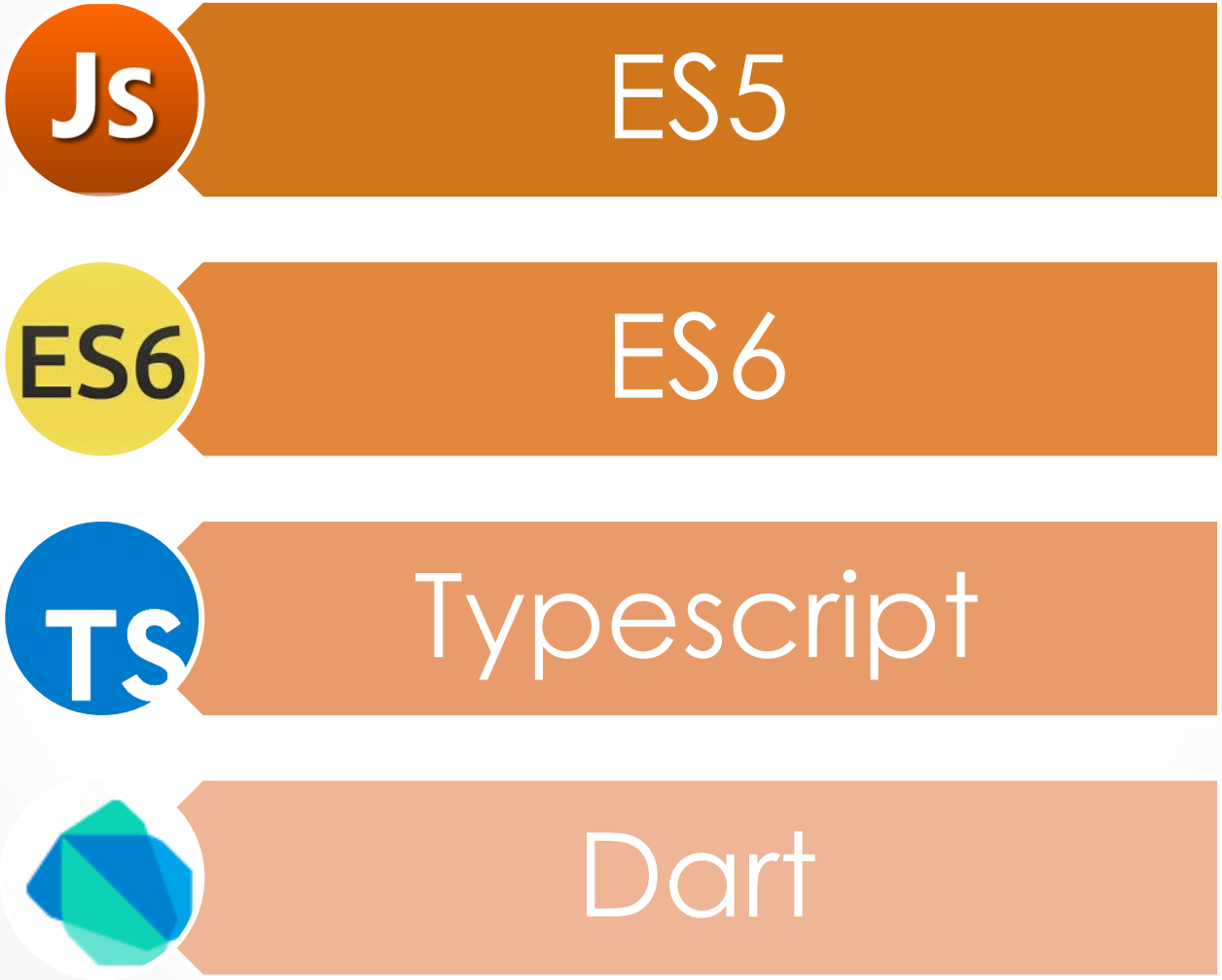
ES6



ES2016+

- Sergiy Morenets, 2020

Angular



- Sergiy Morenets, 2020

JavaScript. ES5


```
function greet(name, sirname) {
    if (!name) {
        var text = sirname;
    }
    console.log(text);
}

greet(null, "Jack");
```

ES6. Variable declaration


```
let name = "John";
let value = 1;
value = 3;

function greet(name, sirname) {
    if (!name) {
        let text = sirname;
    }
    console.log(text);
}

greet(null, "Jack");
```

- Sergiy Morenets, 2020

ES6. Constants


```
const PI = 3.14;

PI = 3.1415;
```


Compilation error

```
const VALUES = {
    value: 3.14
};

VALUES.value = 3.1415;
```

ES6. Variable declaration. Puzzle


```
console.log(data);
let data = 20;
```

```
let i = 1;

if (i >= 1) {
    let i = 2;
    console.log(i);
}

console.log(i);
```

```
const width = 20;
width++;

console.log(width);
```

JavaScript. ES5


```
function greet(name, greeting) {
    console.log(greeting + ', ' + name);
}

greet('Peter');
```

→ Undefined, Peter

```
function greet(name, greeting) {
    greeting = greeting || 'Hello';
    console.log(greeting + ', ' + name);
}

greet('Peter');
```

→ **Hello**, Peter

ES6.Default parameters


```
function greet(name, greeting = 'Hello') {
    console.log(greeting + ', ' + name);
}

greet('Peter');
```

→ **Hello**, Peter

JavaScript. ES5


```
function print(value) {
    for (var i = 0; i < arguments.length; i++) {
        console.log(arguments[i]);
    }
}

print('1', '2', '3');
```

→ 1, 2, 3

ES6.Rest operator


```
function print(...values) {
    for (let value of values) {
        console.log(value);
    }
}

print("1", "2", "3");
```

ES6.Spread operator

```
let developers = ['John', 'Peter'];
let employees = ['Tom', ...developers];
console.log(employees);
```

['Tom', 'John', 'Peter']

```
const product = {
    name: 'Phone',
    vendor : 'Samsung',
    price: 300
};
```

```
const discountedProduct = {
    ...product,
    price: 100
};
```

ES6.Spread operator and arrays


```
const phones = ['Samsung', 'Galaxy'];
const laptops = ['Dell', 'HP'];

let products = phones.concat(['Nikon']);
products = products.concat(laptops);
```

```
const products = [...phones, 'Nikon', ...laptops];
```

ES6.Destructuring


```
let [i, j] = [1, 2];
```

```
function getDetails() {
    return {
        name: 'John',
        age: 20
    };
}

let {name, age} = getDetails();

console.log(name + ' ' + age);
```

- Sergiy Morenets, 2020

ES6.Destructuring. Examples


```
const items = ['JS', 'TS'];

for(const item of items) {
    console.log(`Item value is ${item}`);
}
```

JS, TS

```
for(const [idx, item] of items) {
    console.log(`Item value is ${item} and index ${idx}`);
}
```

S

J, T

```
for(const [idx, item] of items.entries()) {
    console.log(`Item value is ${item} and index ${idx}`);
}
```

0, 1

ES6.Destructuring. Puzzle


```
const person = {
    name: 'John',
    age: 20
};
```

```
const {name, b, c} = person;
console.log(name);
console.log(b);
console.log(c);
```

```
const {name, age: b} = person;
console.log(name);
console.log(b);
```

ES6.Object literal upgrades


```
const product = {
    name: 'Phone',
    vendor : 'Samsung',
    price: 300
};
```

```
const name = 'Phone';
const vendor = 'Samsung';
const price = 200;
```

```
const product = {
    name: name,
    vendor: vendor,
    price: 300
};
```

```
const product = {
    name,
    vendor,
    price: 300
};
```

Task #2. ES6 features.

1. Open your browser (Firefox, Chrome) Web console in Developer Tools, enable **command line** if needed.
2. Declare several variables using "**let <var>=<value>"** syntax
3. Declare new **constant**. Try to reassign new value to it
4. Write a function that accepts default parameters. Try to invoke it passing different number of arguments.
5. Write a function that uses Rest operator. How will you access its arguments?

IDE

- Sergiy Morenets, 2020

WebStorm

- ✓ JavaScript/TypeScript/CSS/HTML dev environment
- ✓ Supports **LiveEdit** and JavaScript debugging
- ✓ **NodeJS** server-side support
- ✓ Supports **Angular**/React/Meteor web-frameworks
- ✓ Supports **Angular** (starting since 2016.1)
- ✓ Angular templates
- ✓ Integration with **VCS** and bug-tracking systems
- ✓ Supports build/code quality/unit-testing tools
- ✓ Developed by **JetBrains**
- ✓ Commercial product

ES6. Strict declaration

```
item = 1;

console.log(item);

const data = {
    name: 'Phone',
    name: 'Computer'
}

console.log(data.name);
```

1
Computer

```
'use strict';

item = 1;

console.log(item);

const data = {
    name: 'Phone',
    name: 'Computer'
}

console.log(data.name);
```

ReferenceError: item is not defined

Task #3. Working with WebStorm

1. Open **WebStorm (2020.1** or later is recommended**)** and create empty project. Create new empty JavaScript file, for example, **sample.js**. Put “use strict”; as first line in the file.
2. Open Webstorm preferences (File->Settings), open **Language & Frameworks** -> JavaScript panel. Choose “**ECMAScript 6**” option.
3. Repeat steps 2-5 from **Task #2**. You should run your script using Node.js interpretator.
4. Try to debug your code in IDE.

ES6.Objects


```
const person = {
    name: 'John',
    age: 20
};

console.log(person.age);
```

```
const person = {
    name: 'John',
    age: 20,

    printAge() {
        console.log('Age is ' + this.age);
    }
};
```

Requires schema declaration per each object

ES6.Classes


```
class User {
    constructor(name) {
        this.name = name;
    }

    printName() {
        console.log('My name is ' + this.name);
    }
}

const user = new User('Peter');
user.printName();
```

Declare name property

ES6.Getters and setters


```
class User {
    constructor(firstName, secondName) {
        this._firstName = firstName;
        this._secondName = secondName;
    }

    get fullName() {
        return this._firstName + ' ' + this._secondName;
    }
```

```
const user = new User('Peter', 'Johnson');
console.log(user.fullName);
```

Use in property-style

ES2019. Class fields


```
class User {
    name = '';
}

const user = new User();
user.name = 'John';
```

No need to use constructor for field declaration

```
class User {
    #name = '';
}

const user = new User();
user.#name = 'John';
```

Private field

SyntaxError: Private field '#name' must be declared in an enclosing class

ES6.Inheritance


```
class User {
    constructor(name) {
        this.name = name;
    }
}
```

```
class Developer extends User {
    constructor(name, skills) {
        super(name);
        this.skills = skills;
    }

    static getRole() {
        return 'developer';
    }
}
```

```
const user = new Developer('John', 'JS');
console.log(Developer.getRole());
```

- Sergiy Morenets, 2020

ES6.Arrow functions


```
function double(x, callback) {
    let result = x * 2;
    callback(result);
    return result;
}

double(1, function (x) {
    console.log('Result is ' + x);
});
```

Anonymous function

```
double(1, x => console.log('Result is ' + x));
```

```
const printer = name => console.log(`Name is ${name}`);
printer( name: 'John');
```

Arrow functions pitfalls


```
const employee = {
    name: 'John',
    age: 30,
    greet : function () {
        console.log(`My name is ${this.name}`);
    }
}

employee.greet();
```

Hi, my name is John

TypeError: Cannot read property 'name' of undefined

```
const employee = {
    name: 'John',
    age: 30,
    greet : () => console.log(`My name is ${this.name}`)
};
```

ES6.Collections


```
const map = new Map();
map.set('name', 'John');
map.set('name', 'Peter');
```

Value replacement

```
console.log(map.has('name'));
console.log(map.get('name'));
console.log(map.size);
map.delete('name');
```

```
const names = new Set();
names.add('1');
names.add('2');
names.add(1);

console.log(names.size);
```

Non-ordered

```
for (const name of names) {
    console.log(name);
}
```

```
const set = new Set(['1', '2']);
```

ES6. Style guide

- ✓ Developed by AirBnb
- ✓ Types, objects and functions
- ✓ Classes, modules and properties
- ✓ Naming conventions
- ✓ Testing & performance

https://github.com/airbnb/javascript

Task #4. ES6 Classes

1. Try to create class's hierarchy using **WebStorm** as editor.
2. You should add properties, getters/setters, methods (including **static** ones) and constructors to the classes.
3. Try to instantiate several instances of these classes and invoke underlying methods.
4. Try to declare private fields in the class. Can you access them outside of the class?

Modules

- No real standard until ES6
- Server-side attempt is NodeJs using CommonJS
- Client-side API in RequireJS uses Asynchronous Module definition API
- Supports asynchronous loading and cyclic dependencies in import/export

ES6.Modules


```
export function sum(x, y) {
    return x + y;
}
```

```
export class MathUtils {
    static divide(x, y) {
        return x / y;
    }
}
```

export.js

Can be used outside of export.js

```
import {MathUtils, sum} from "./export.js";

console.log(sum(1,2));
console.log(MathUtils.divide(2, 3));
```

ES6. Modules


```
import {Context} from "./model/Context";
import {Context} from "./service/Context";
```

```
import {Context} from "./model/Context";
import {Context as ServiceContext} from "./service/Context";

const context1 = new Context();
const context2 = new ServiceContext();
```

ES6.Modules

- ES6 modules are running in strict mode by default (no need for 'use strict' anymore).
- Top-level value of **this** is undefined.
- Top-level variables are local to the module.
- ES6 modules are loaded and executed asynchronously after the browser finished parsing the HTML
- Supported since Safari 10.1, Chrome 61, Firefox 54 and Edge 15 (over 75% of modern browsers, for other you can use polyfills)

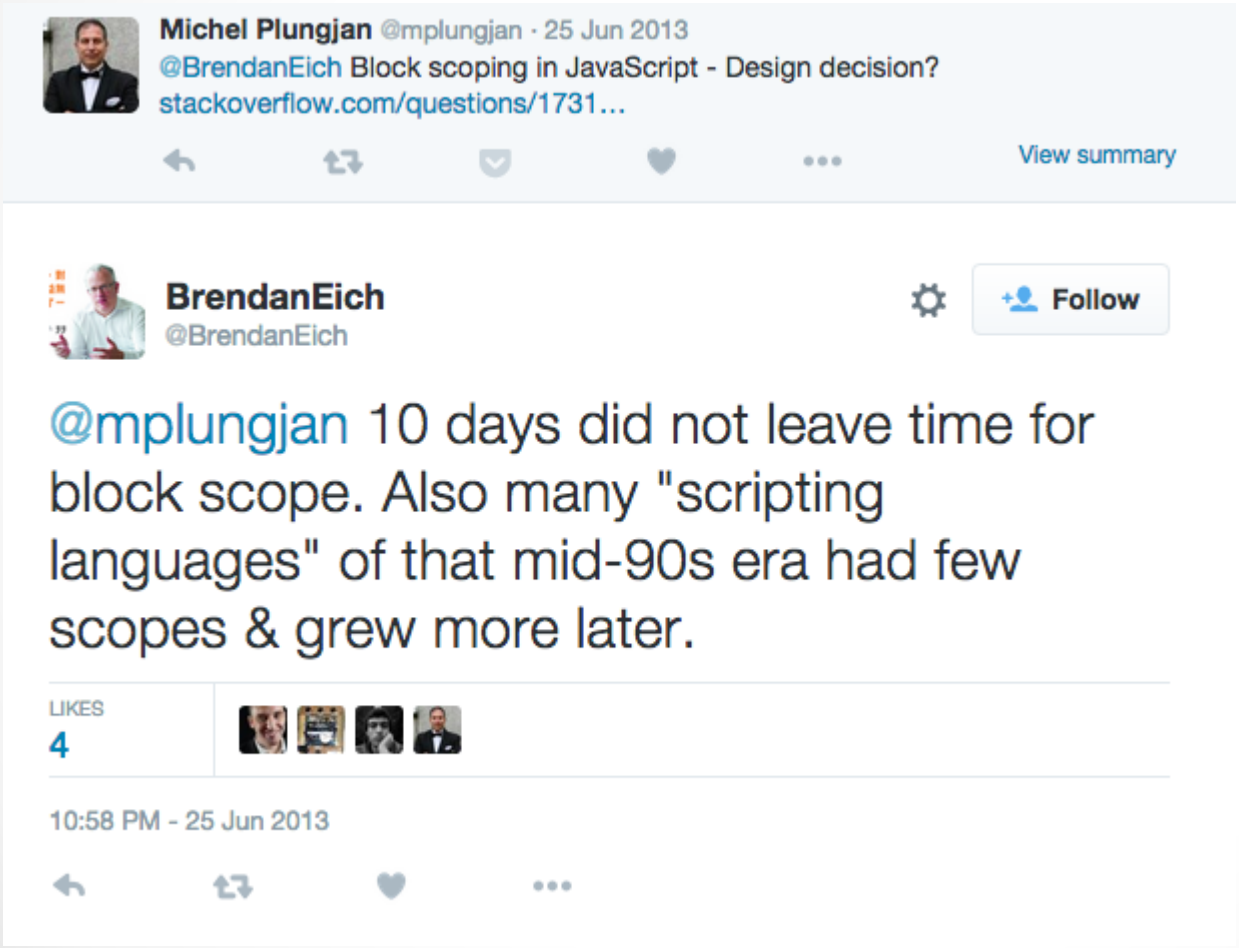
ES6.Modules


```
<script type="module">
  import {addTextToBody} from './utils.js';

  addTextToBody('Modules are pretty cool.');
</script>
```

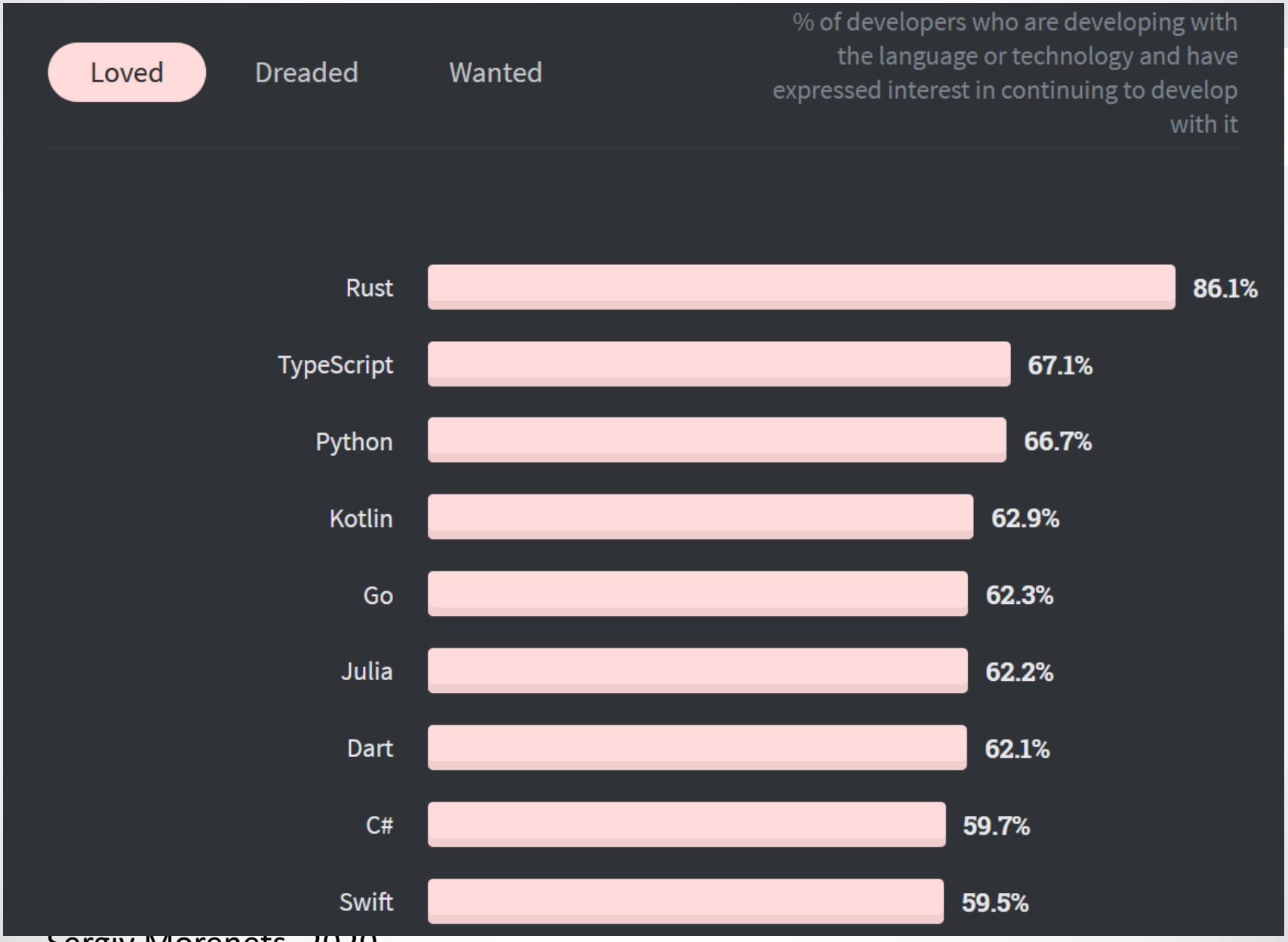
```
// utils.js
export function addTextToBody(text) {
  const div = document.createElement('div');
  div.textContent = text;
  document.body.appendChild(div);
}
```

- Sergiy Morenets, 2020

JavaScript

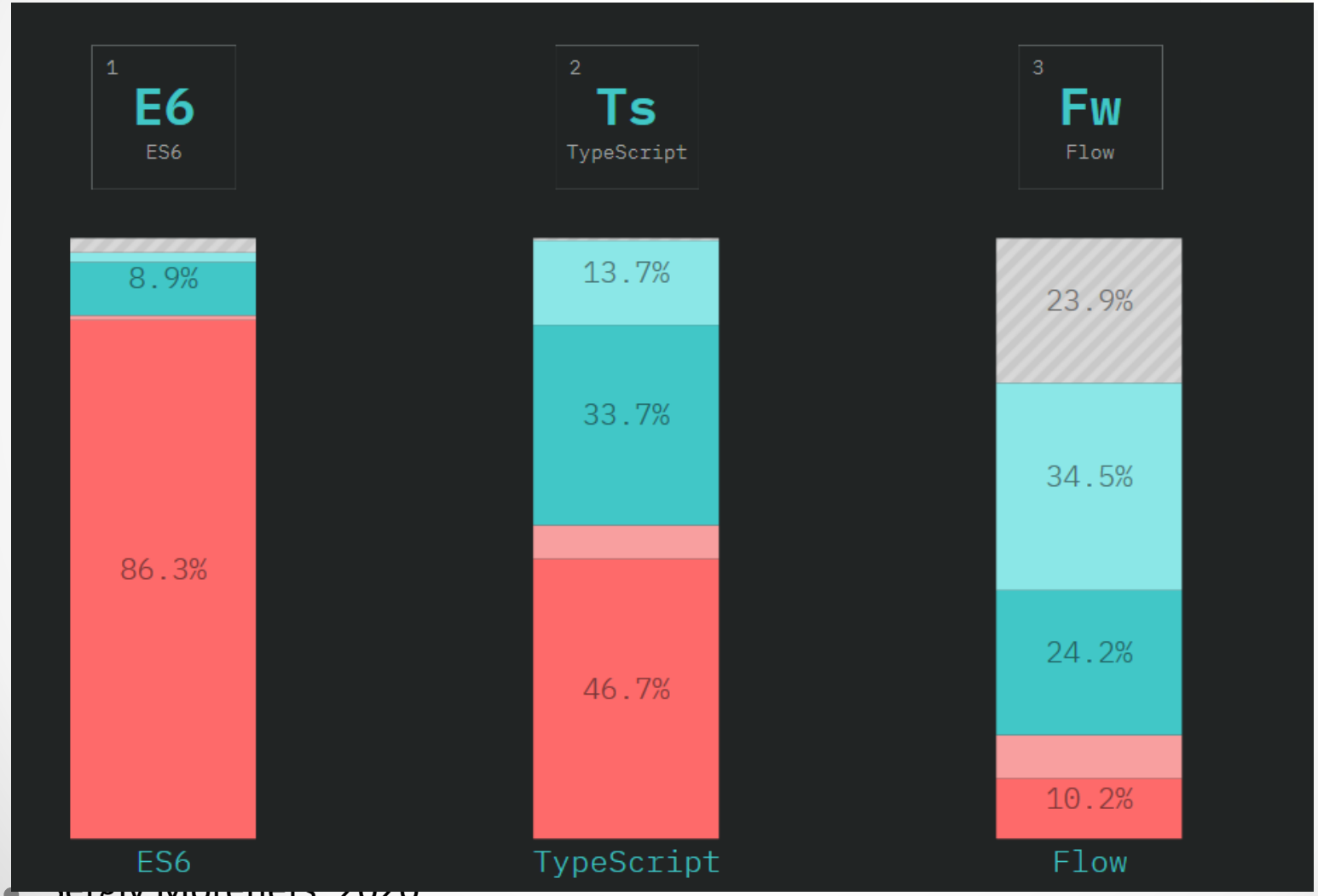


- Sergiy Morenets, 2020



- Sergiy Morenets, 2020

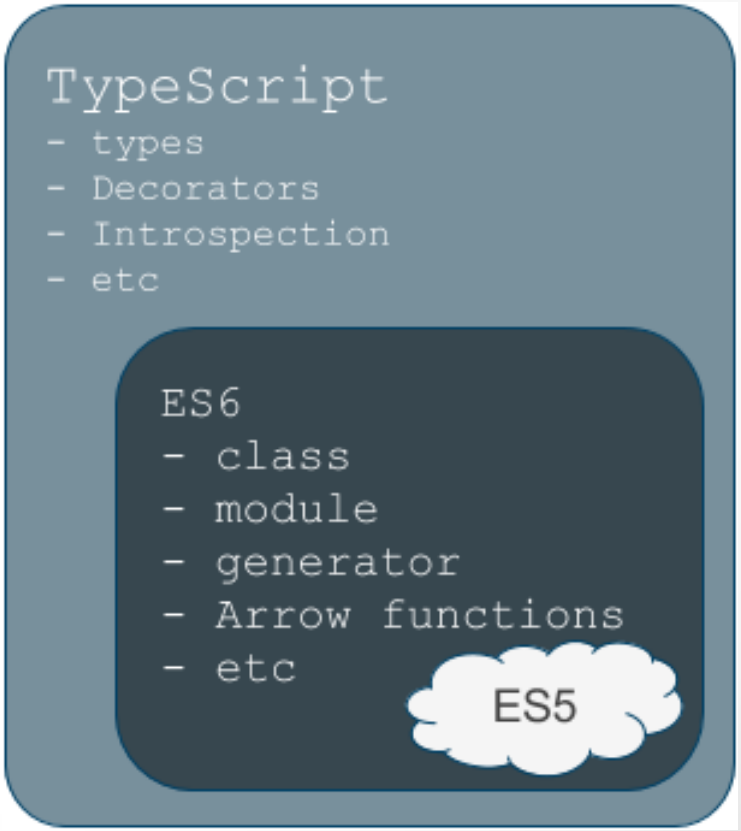
State of JS 2018



- Sergiy Morenets, 2020

TypeScript

- ✓ Typed superset of ES6
- ✓ Strong typing with IDE support
- ✓ Developed since 2012
- ✓ Has .ts extension
- ✓ Current version 3.9.5



Types. Advantages

- ✓ Introduce rules and constraints
- ✓ Resolve common errors
- ✓ Allows to fix errors at compile-time (not runtime)
- ✓ Easier refactoring
- ✓ IDE support

```
function addition(a, b) {
  if(typeof a != 'number' || typeof b != 'number') {
    return a + b;
  } else {
    Throw(…)
   }
}
```

- Sergey Morenets, 2020

TypeScript. Variables and types


```
let value: number = 10;
let text: string = 'TypeScript';

let value = 10;
let text = 'TypeScript';

const PI = 3.14;
PI = 3.15;

var name;
name = 'Some text';

let name: any;
name = 'Some text';
```

Primitive types (also boolean)

Type is auto-detected by TypeScript compiler (type inference)

Migration from legacy JavaScript projects

TypeScript. Type any


```
let item: any;
item = 'Some text';
item = false;
```

item can hold any value

```
item.run();
```

You can invoke any function

TypeScript. Objects


```
export class DataHolder {
   data: any;          data: { name: string, id: string} ;

  constructor() {
    this.data = { name: 'Phone', id: '1'};
  }
```

```
data: {};
```

OR

```
data: object;
```

OR

```
data: Object;
```

Allow non-primitive types

```
export class DataHolder {
  data: { [key: string]: string };
```

- Sergiy Morenets, 2020

TypeScript. Type unknown


```
let item: unknown;
```

```
item = 'Text';
item = 1;
```

item can hold any value

```
item.run();
```

You need to **cast** unknown type

```
let amount: number;
amount = item;
```

```
amount = item as number;
```

```
class Info {
    private dataType: unknown;
```

Protect from external usage

TypeScript. Arrays


```
let languages: string[];

languages = ['JavaScript', 'TypeScript', 'ECMAScript'];

languages[0] = 'Java';          languages.push('Python');

const javaLangs = languages.filter(
    (lang: string) => lang.startsWith('Java'));
```

TypeScript. Array tuples


```typescript
let personData: [number, string] = [20, 'John'];

personData[0] = 10;

personData[0] = false;
personData = [1];

personData.push(1, 2, 3);
```

TypeScript. Arrays and iteration


```
const items = ['JS', 'TS'];

for(const item of items) {
    console.log(item);
}
```

JS, TS

```
for(const item in items) {
    console.log(item);
}
```

0, 1

Generator function

JS, TS

```
function *languages() {
    yield 'JS';
    yield 'TS';
}
```

```
for(const lang of languages()) {
    console.log(lang);
}
```

TypeScript. Functions

List of arguments

Return type

```
function factorial(value: number) : number {
    if(value <= 0) {
        throw new Error('Invalid argument: ' + value);
    }
    if(value == 1) {
        return 1;
    }

    return value * factorial( value: value - 1);
}
```

Function never returns a value

```
function throwError(text: string) : never {
    throw new Error(text);
}
```

TypeScript. Type unions


```
let arg: number | string;
arg = 1;
arg = '1';
```

```
let desc = null;
```

Type any

```
let desc: string | null = null;
```

```
function log(msg: string,
             overwrite: boolean| null | undefined) : void {
    if(overwrite) {
```

```
let employeeType: 'Dev' | 'QA' | 'DevOps';
employeeType = 'PM';
```

TypeScript. Types


```
type StringOptional = string | null | undefined;

function print(value: StringOptional) : void {

}
```

```
type EmployeeType = 'Dev' | 'QA' | 'DevOps';
const type: EmployeeType = 'Dev';
```

TypeScript. Asserts


```
let item: any;

console.log(item.toLowecrCase());
```

Assumes item is string

Verifies any statement

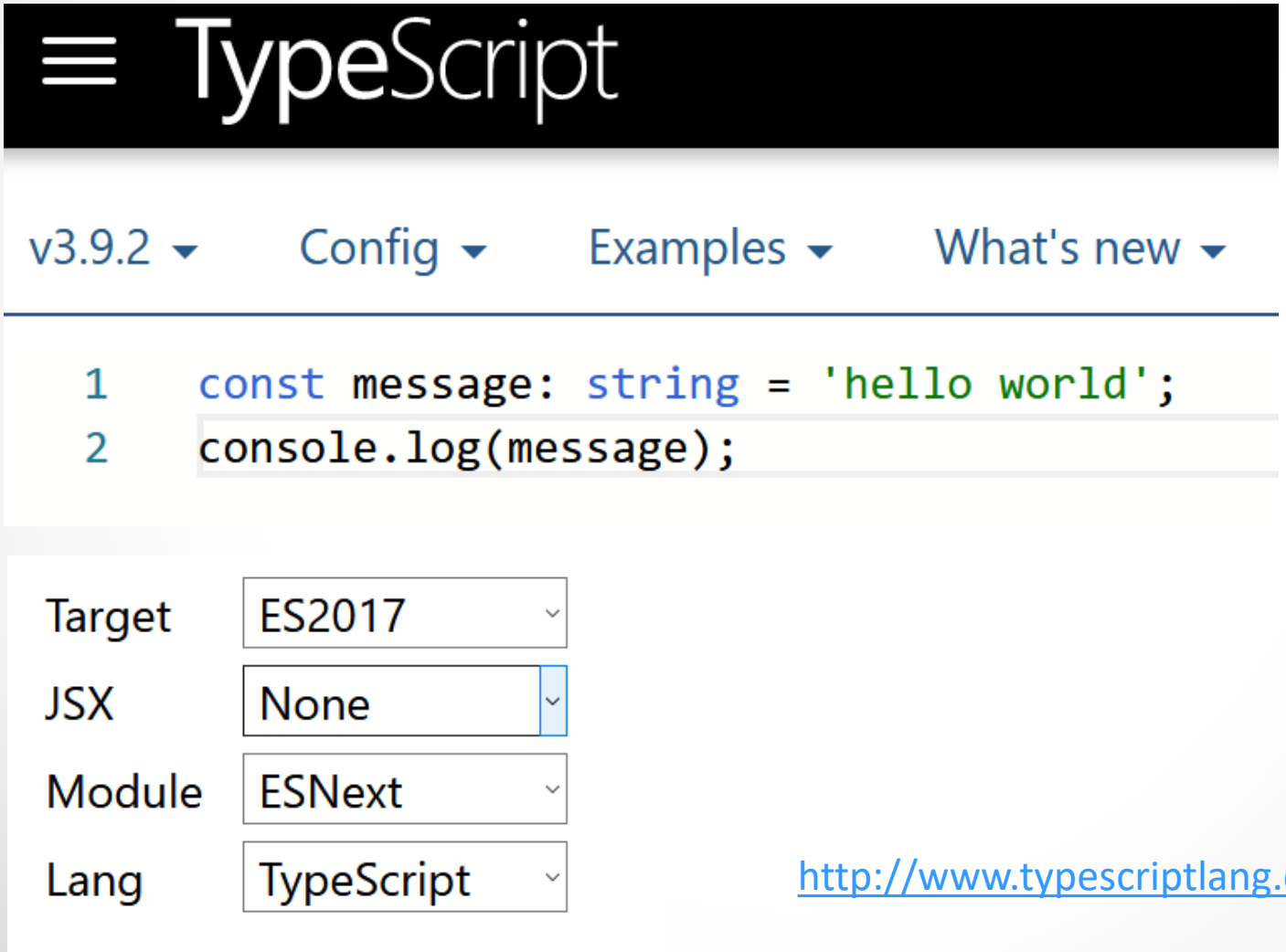
```
function assert(input: any): asserts input {
    if (!input) {
        throw new Error('Not a truthy value');
    }
}
```

TS2551: Property 'toLowecrCase' does not exist on type 'string'. Did you mean 'toLowerCase'?

```
let item: any;
assert(typeof item === 'string');
console.log(item.toLowecrCase());
```

Narrows item type to **string**

Typescript playground



http://www.typescriptlang.org/play/

tsconfig.json structure

property	Description
baseUrl	Base URL for non-resolvable modules
outDir	Output folder to compilation results
sourceMap	Generates .map files
declaration	Generate corresponding .d.ts files
module	Module code generation (ES6, ES2015, CommonJS, System, UMD, ESNext)
emitDecoratorMetadata	Emit design-type metadata for declared instructions
target	ECMAScript target version(ES3, ES5, ES6, ES2016, ES2017, ESNext)
lib	List of library files to include in compilation(DOM, ES5, ES6, WebWorker)
watch	Run compilers in watch mode

tsconfig.json structure

property	Description
strict	Enable all strict check-types (TS 2.3 and later)
strictNullChecks	Enable strict null checks
strictFunctionTypes	Enable strict checking of function types
noImplicitAny	Raised an error if compiler finds implicit any type. False value is useful when migrating from Javascript to Typescript
noUnusedLocals	Report errors on unused locals
noUnusedParameters	Report errors on unused function parameters

```
const quantity: number = null;
```

strictNullChecks: true

```
function print(msg) {}
```

noImplicitAny: true

Task #5. TypeScript basics

1. Create new TypeScript file in **WebStorm**, for example ‘run.ts’
2. Check box “**Recompile on changes**” in ‘Settings’
3. Declare several variables of different types. What will happen if you assign value of incorrect type? What is the type of this variable if its type is not explicitly defined?
4. Create new functions. How do you specify its arguments/return types? Try to invoke this function
5. Create new type

TypeScript. Objects and tuples


```
let personData: [number, string] = [20, 'John'];
```

Tuples

Don’t provide property explanation

Not reusable

```
data: { name: string, id: string} ;
```

```
data: object;
```

Implicit anonymous class

```
data: Object;
```

TypeScript. Objects and tuples


```
const person = {
    name: 'John',
    id: 20
}
```

```
function handlePersonData(person: object) : void {
    console.log(`Person name: ${person.name}`);
    console.log(`Person id: ${person.id}`);
}
```

No declared properties

```
function handlePersonData(person: { name: string, id: number })
    : void {
    console.log(`Person name: ${person.name}`);
    console.log(`Person id: ${person.id}`);
}
```

TypeScript. Classes and objects


```
class Person {
    age: number;

    constructor(age: number) {
        this.age = age;
    }

    getAge(): number {
        return this.age;
    }
}
```

Property

Constructor

Function

Reference class-level attributes

```
const person = new Person(20);  let person: Person;
console.log(person.getAge());
```

TypeScript. Access modifiers


```
class Person {
    private readonly age: number;

    constructor(age: number) {
        this.age = age;
    }

    public getAge(): number {
        return this.age;
    }
}
```

Access allowed only inside class

Visible outside of the class

```
const person = new Person(20);
console.log(person.age);
```

Access restricted

TypeScript. Classes and objects


```
class Employee {
    age: number;
    name: string;
}
```

```
let employee: Employee = {
    age: 25,
    name: 'John'
}
```

```
let employee2: Employee = {
    name: 'Peter'
}
```

← No age property

```
let employee: {
    age: number;
    name: string;
}
```

← We can assign any object with age/number properties

TypeScript. Classes and objects


```
class Server {
    status: boolean;

    startDate: Date;

    start() : void {
        console.log('Process started');
    }
}

const server: Server = {
    status: false,
    startDate: new Date()
};
```

start() should be declared

TypeScript. Constructors


```
class User {
    constructor(public name: string) {
    }
}
```

```
class User {
    public name: string;

    constructor(name: string) {
        this.name = name;
    }
}
```

TypeScript. Optional fields


```
const users: any[] = [{name: 'John'},
    {id: 1, name: 'Peter'}];
```

```
class User {
    id: number;

    name: string;
}
```

Optional field →

```
class User {
    id?: number;

    name: string;
}
```

```
const users: User[] = [{name: 'John'},
    {id: 1, name: 'Peter'}];
```

TypeScript. Enumerations


```
enum WorkStatus {
    Started, InProgress, Completed, Aborted
}
```

```
const workStatus = WorkStatus.Started;
```

```
let status: 'Started' | 'InProgress' | 'Completed'
  | 'Aborted';

status = 'Aborted';
```

```
status = 'None';
```

Doesn't compile

TypeScript. Puzzler

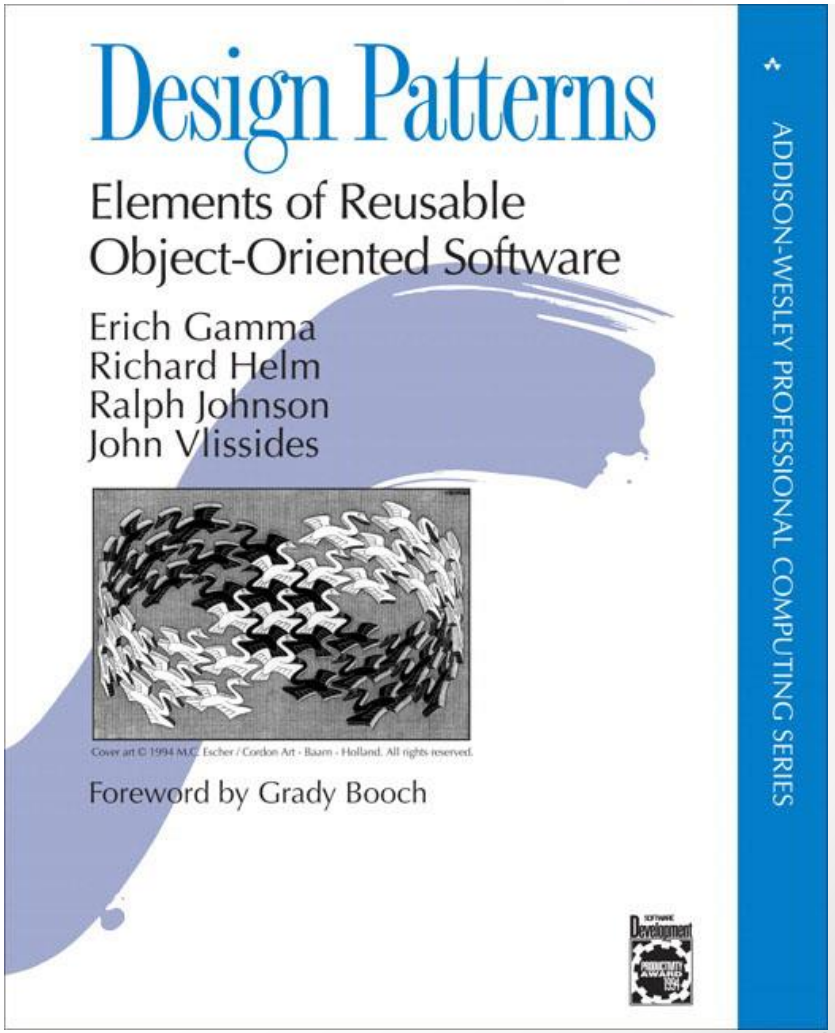
```
class Product {
    id: number;

    name: string;

    price: number;
}
const product = new Product();
product.name = 'PC';
console.log(product.price);
```

undefined

Task #6. TypeScript classes

1. Create new class Book. Which properties/methods/ constructors can you add for it? Try to instantiate this class.
2. Try to **overload** methods/constructors.
3. Which modifiers should you use for Book properties/methods? Declare a property with **private** modifier in the class. Can you access it outside of the class?

TypeScript. OOP and patterns



- Sergiy Morenets, 2020

TypeScript. Generalization


```
class Developer {
    vacationDays() : number {
        return 16;
    }
}
```

```
class Manager {
    vacationDays() : number {
        return 8;
    }
}
```

Developer/Manager

```
function totalVacationDays(employees: any[]) : number {
    return employees.map(emp => emp.vacationDays())
        .reduce((num1, num2) => num1 + num2);
}
```

TypeScript. Abstract classes


```typescript
abstract class Employee {
    public abstract type: string;

    public abstract vacationDays(): number;
}

class Developer extends Employee {
    type = 'Dev';

    vacationDays(): number {
        return 16;
    }
}
```

TypeScript. Interfaces


```
interface Employee {
    vacationDays(): number;
}
```

Abstract method

Can’t have constructors

```
class Developer implements Employee {
    vacationDays(): number {
        return 16;
    }
}
```

[new Developer(), new Manager()]

```
function totalVacationDays(employees: Employee[]): number {
    return employees.map(employee => employee.vacationDays())
        .reduce((i, j) => i + j);
}
```

TypeScript. Interfaces


```
interface Item {
    price: number;

    description: string;
}
```

```
class ProductItem implements Item {
    price: number;

    description: string;
}
```

```
const phone: Item = {
    description: 'Mobile phone',
    price: 400
};
```

```
phone.price = 500;
```

```
interface Item {
    readonly price: number;

    readonly description: string;
}
```

TypeScript. Arrow functions

```
execute(param: number, successCallback, errorCallback):
    void {
    try {
        this.exec(param);
        successCallback(param);
    } catch (e) {
        errorCallback(e);
    }
}
```

Signature is inknown

```
execute(param: number, successCallback: ((x:number) => void),
        errorCallback: (err: Error | string) => void):
    void {
```

Types of arguments are returned value

TypeScript. Puzzler

```
class Utils {
    public static isValid (code: number) {
        return code === 10;
    }
}
```

```
const code: any = '10';
console.log(Utils.isValid(code));
```

→ false

```
public static isValid (code: number) {
    return Number(code) == 10;
}
```

- Sergiy Morenets, 2020

TypeScript. Generic types


```
function asArray (item) {
    return [item];
}
```

What is the type of argument/ return value?

```
function asArray (item: any): any[] {
    return [item];
}
```

Return type should be the same as type of argument

```
function asArray<X> (item: X): X[] {
    return [item];
}
```

```
const arr = asArray('1');
```

TypeScript. Generic types


```
class ArrayUtils<X> {
    asArray (item: X): X[] {
        return [item];
    }
}
```

```
const arrUtils = new ArrayUtils();
const arr = arrUtils.asArray('1');
```

What is the type **arr**?

Specify generic type

```
const arrUtils = new ArrayUtils<string>();
```

TypeScript. Generic types

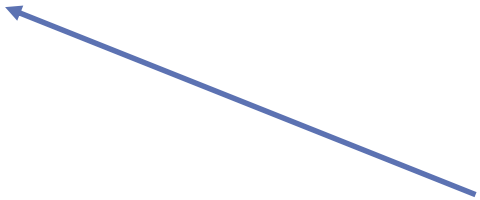

```
class ArrayUtils<X = string | number> {
    asArray (item: X): X[] {
        return [item];
    }
}
interface Order {
    paid: boolean;
}
class OrderUtils {
    getPaid<O extends Order>(orders: O[])
        : O[] {
        return orders.filter(o => o.paid);
```

Default generic type

We can pass only Order
or its implementations

TypeScript. Generic types


```
class OrderUtils {
    getPaid(orders: Order[])
        : Order[] {
        return orders.filter(o => o.paid);
    }
}
```



Can we use this implementation and get rid of generic types?

TypeScript. Type safety


```
function log(msg : string): void {
    console.log(msg);
}
```

```
class LogMessage {
    text: string;

    level: string;
}
```

```
function log(msg : string | LogMessage | number): void {
```

```
function log(msg : string | LogMessage | number): void {
    if(typeof msg === 'string') {
        console.log(msg);
    } else if(msg instanceof LogMessage) {
        console.log('Level:' + msg.level + ", message: " +
        msg.text)
```

TypeScript. Data privacy


```
class Movie {
    constructor(private genre: string) {
    }
}
```

Transpiling into JS2015

```
class Movie {
    constructor(genre) {
        this.genre = genre;
    }
}
```

Can be accessed outside of Movie class

TypeScript. Data privacy

```
class Movie {
    readonly #genre: string;

    constructor(genre: string) {
        this.#genre = genre;
    }
```

Private names (ECMAScript private fields)

Transpiling into JS2015

```
var _genre;
class Movie {
    constructor(genre) {
        _genre.set(this, void 0);
        __classPrivateFieldSet(this, _genre, genre);
    }
}
_genre = new WeakMap();
```

JS2020

```
class Movie {
    constructor(genre) {
        this.#genre = genre;
    }
    #genre;
}
```

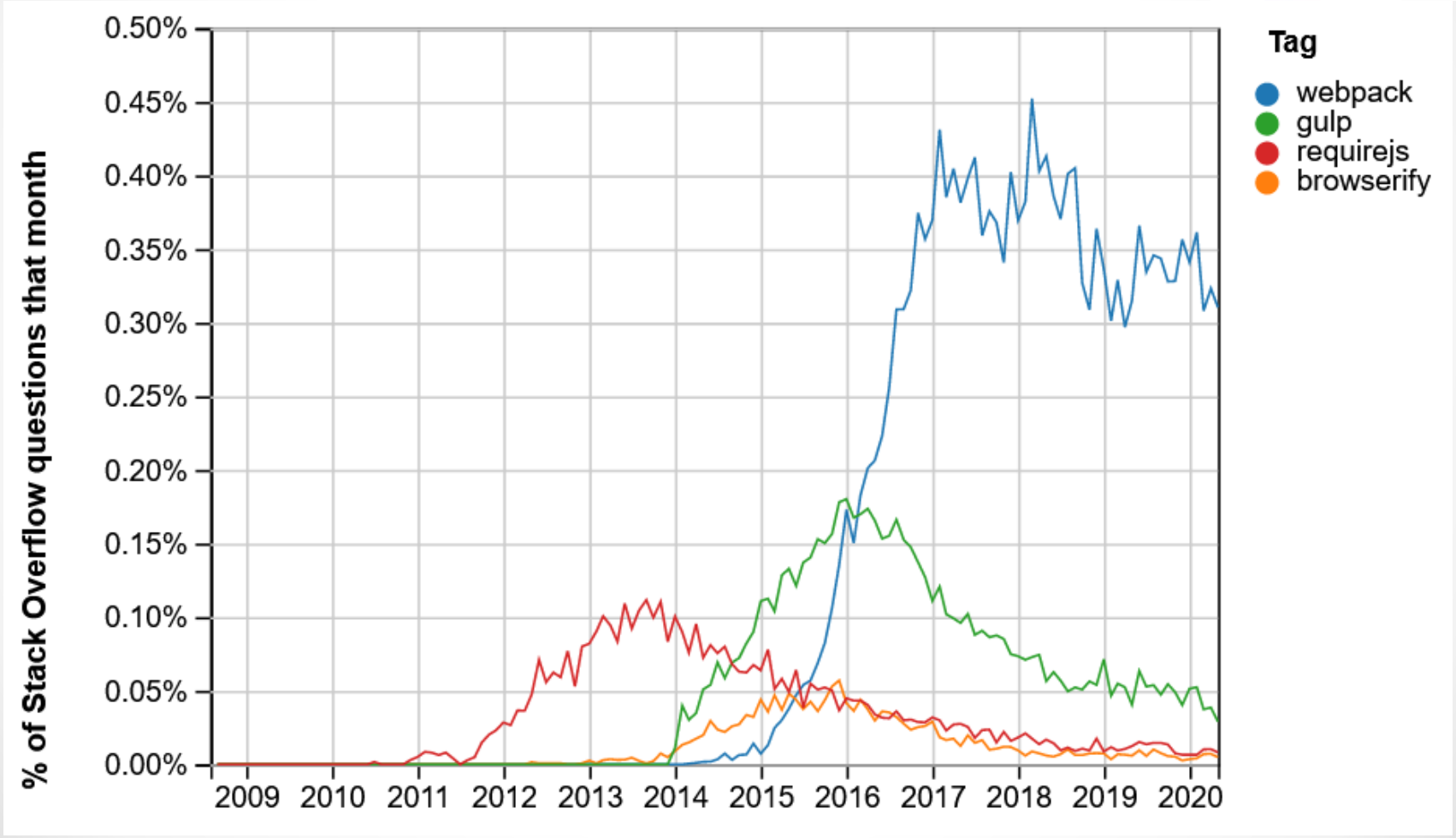
Task #7. Interfaces and generic types

1. Create new classes hierarchy that contains square, triangle and rectangle. Add required fields for them
2. Try to extract common interface and implement it in all the classes. Which method will this interface contain?
3. Review generated JavaScript file. How are interfaces translated into JavaScript?
4. Create array that contains square, triangle and rectangle

Module bundlers/loaders

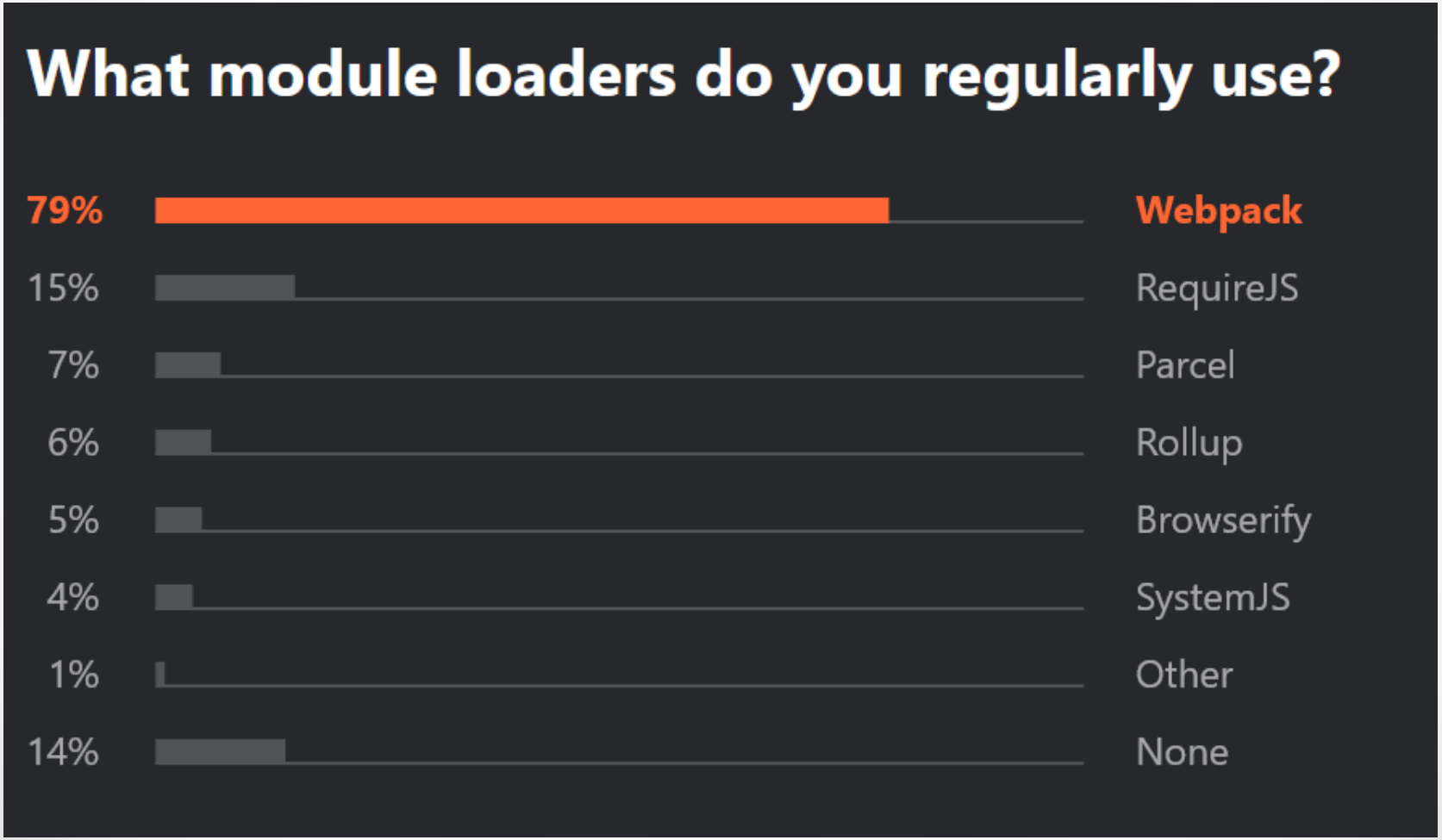
- ✓ Current module bundlers are SystemJS, Webpack, Rollup, Browserify, JSPM and Parcel
- ✓ Main goals are code splitting and asset management
- ✓ Extensible by plugins

Module bundlers



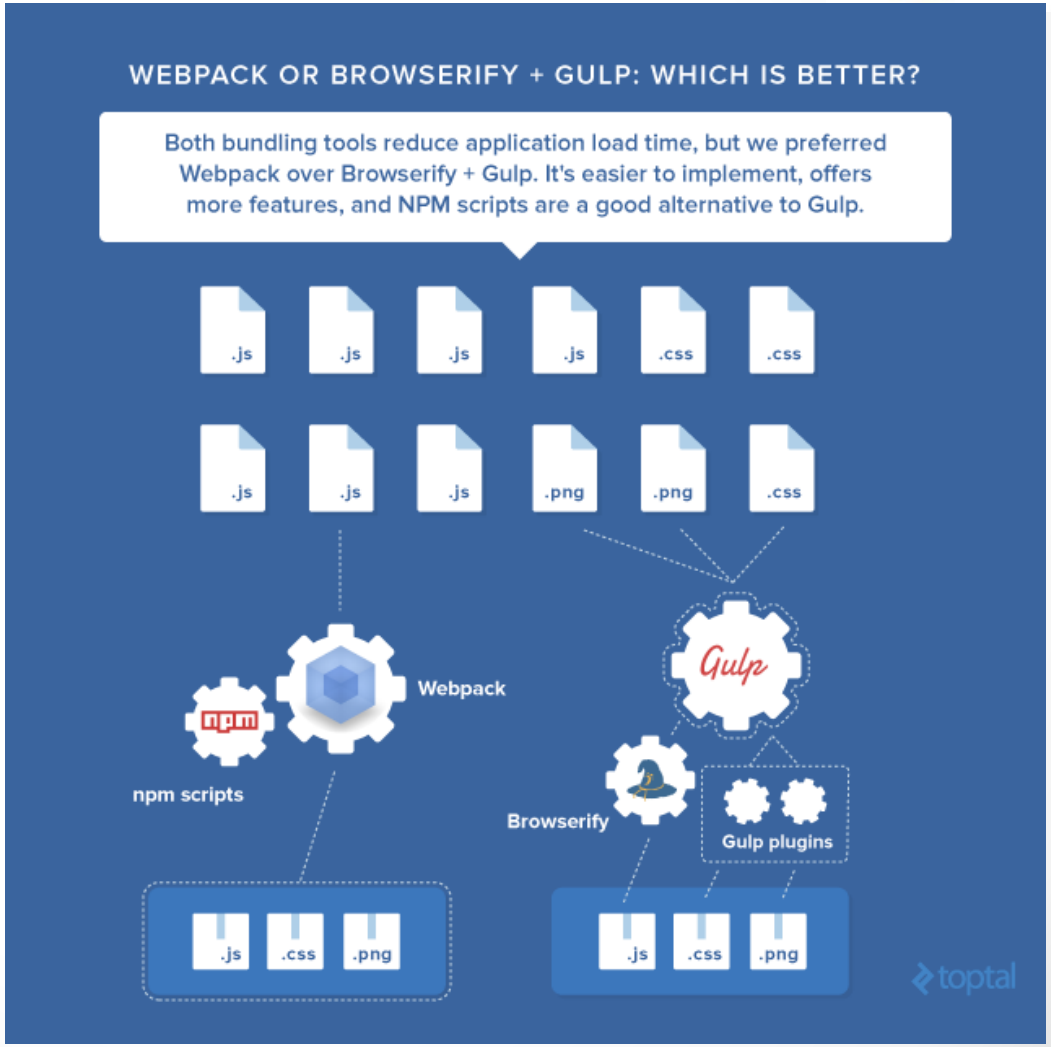
- Sergiy Morenets, 2020

Module bundlers



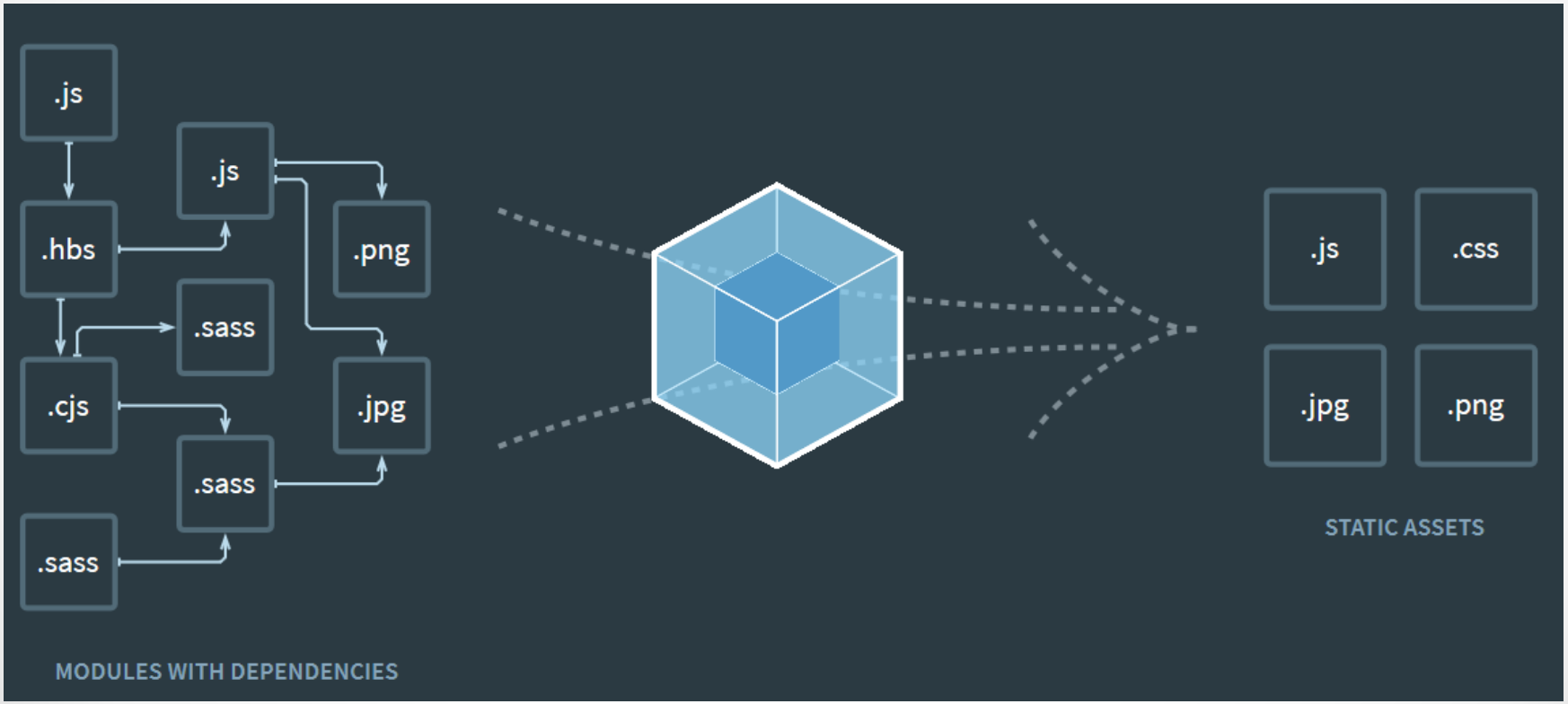
- Sergiy Morenets, 2020

Bundling tools



- Sergiy Morenets, 2020

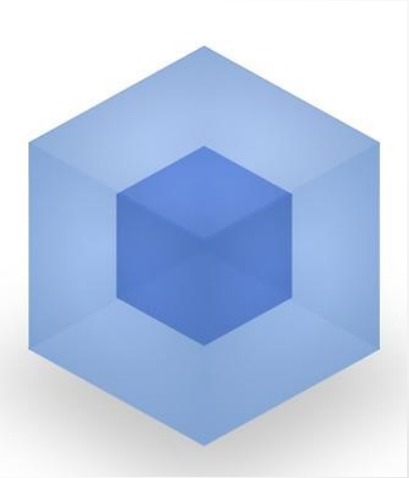
Webpack



- Sergiy Morenets, 2020

Webpack

- Started in 2012 by Tobias Kopper
- Static module bundler for JavaScript applications
- Useful to bundle SPA applications into small chunks(vendor + app)
- Creates graph of dependencies and packages application modules into bundle(s) during build
- Treats resources(images, css) as dependency
- Highly customizable



Webpack

- Since 4.0 doesn't require explicit configuration
- Processes JavaScript/JSON out-of-the-box. So **loaders** are required to convert other files into valid modules
- Plugins allow to optimize bundles and manage assets
- By default runs in the production mode that implies built-in optimization(minification, tree shaking)
- To support tree shaking you need ES2015 modules and minifier to eliminate the dead code
- Supports Hot Module replacement
- Can split bundles into chunks

Webpack.config.js

```
const path = require('path');

module.exports = {
  entry: './path/to/my/entry/file.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'my-first-webpack.bundle.js'
  }
};
```

- Sergiy Morenets, 2020

Task #8. Webpack

1. Install **webpack**: *npm install -g webpack*
2. Install **concurrently** library: *npm install concurrently --save-dev*
3. Add new file **webpack.config.js** to the root folder of your project
4. Remove all **<script>** blocks from index.html. Add this line to the end of the <body> tag: *<script src="dist/bundle.js"></script>*
5. Run **npm start** command

Angular 1.0

- ✓ Originally developed by Miško Hevery and Adam Abrons
- ✓ Initial release in 2009
- ✓ Maintained by **Google** and used in **100+** projects

- Sergiy Morenets, 2020

Angular 1

- ✓ MVC
- ✓ Functional programming
- ✓ Data binding(two-way)
- ✓ Templates
- ✓ Dependency injection
- ✓ Directives
- ✓ Promises
- ✓ Built-in services
- ✓ Testability
- ✓ Integration with unit-testing/build tools
- ✓ A lot of libraries ….

Angular 2

- ✓ Development started in Sep 2014 and first release in Sep 2016
- ✓ Completely new code base
- ✓ Up to 5x times faster
- ✓ Can be written in ES5, ES6, TypeScript **and Dart**
- ✓ Component-oriented
- ✓ Captured best ideas from ReactJS/EmberJS/Aurelia/Vue.js
- ✓ HTML page is set of components where each component can have sub-components

Angular 4

- Released in March 2017
- Current version 4.4.7
- Performance improvements, including **AOT** and bundle size
- Enhanced ***ngIf** and ***ngFor**
- Angular Universal
- TypeScript 2.3 compatible
- Angular Material 2

Benchmarks. Duration(ms)

Angular 4 is slower and bigger file size than Angular 2 #15748

Name	angular-v4.4.3-keyed	react-v16.0.0	angular-v1.6.3-keyed	ember-v2.13.0-keyed	react-lite-v0.15.30-keyed	knockout-v3.4.1-keyed
create rows Duration for creating 1000 rows after the page loaded.	195.1 ± 35.3 (1.4)	193.9 ± 6.1 (1.4)	267.0 ± 0.0 (1.9)	383.8 ± 20.5 (2.8)	176.7 ± 9.8 (1.3)	380.9 ± 12.6 (2.7)
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	187.1 ± 11.5 (1.4)	172.9 ± 5.0 (1.3)	262.5 ± 0.0 (2.0)	277.5 ± 10.5 (2.1)	231.2 ± 6.8 (1.8)	349.9 ± 18.3 (2.7)
partial update Time to update the text of every 10th row (with 5 warmup iterations).	17.5 ± 7.6 (1.1)	15.6 ± 5.4 (1.0)	12.0 ± 0.0 (1.0)	18.9 ± 1.8 (1.2)	31.4 ± 2.2 (2.0)	10.7 ± 0.6 (1.0)

https://github.com/krausest/js-framework-benchmark

Angular 5

- Released in November 2017
- Current version 5.2.11
- Build optimizer and performance improvements
- Progressive web apps
- Material Design for server-side rendering
- Based on TypeScript 2.4

Angular 6

- ✓ Released in March 2018
- ✓ Current version 6.1.10
- ✓ Angular Elements
- ✓ Service workers support
- ✓ Ivy engine development started
- ✓ Bazel compiler
- ✓ Migration to RxJS 6
- ✓ Tree shaking support
- ✓ Based on TypeScript 2.6

Angular 7

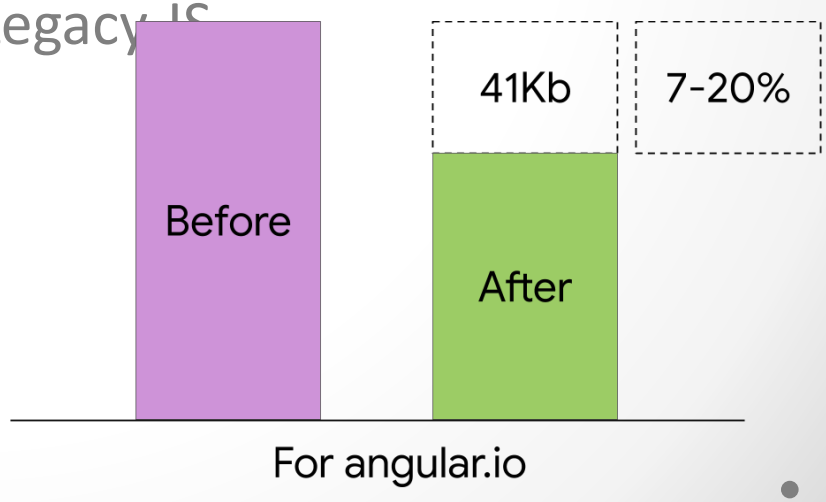
- ✓ Released in October 2018
- ✓ Current version 7.2.3
- ✓ Angular CLI prompts
- ✓ Application performance improvements
- ✓ Angular Material changes (Virtual Scrolling, Drag’n’Drop)
- ✓ Added support for RxJS 6.3, Node v10 and TypeScript 3.x

Angular 8

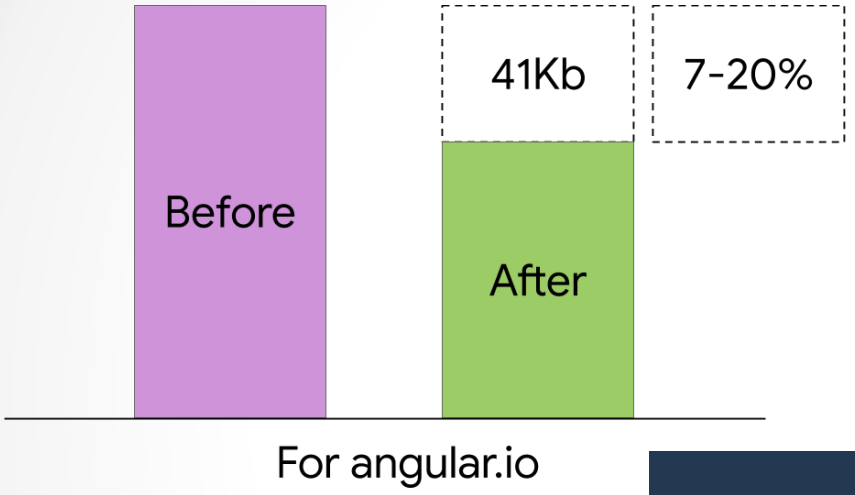
- ✓ Released in May 2019
- ✓ Current version 8.2
- ✓ Different support of legacy (ES5) and modern (ES2015+) JavaScript during build process
- ✓ Opt-in Ivy preview
- ✓ Improved Web Worker bundling
- ✓ Improved Bazel support

Angular 8. Differential loading

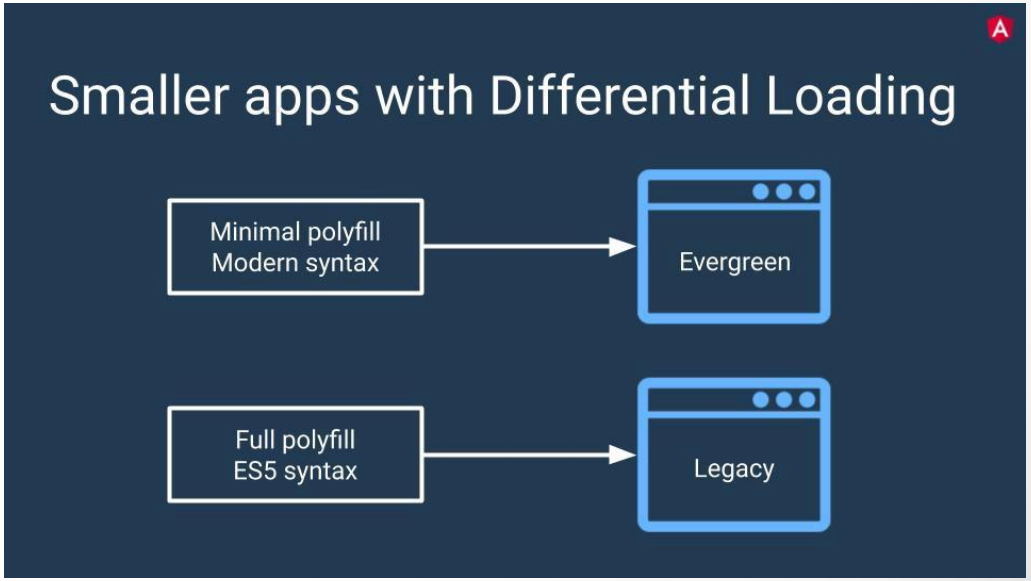
- Angular generates modern ES2015 and legacy ES5 build by default
- When browser is loading the application it uses necessary build
- No need for polyfills for modern browsers
- <script type="module" src="…"> // Modern JS
- <script nomodule src="…"> // Legacy JS



Angular 8. Differential loading



For angular.io



- Sergiy Morenets, 2020

Angular 9. Roadmap

9.0.0-next.0 (2019-07-31)

9.0.0-next.15 (2019-10-30)

9.0.0-rc.14 (2020-02-03)

9.0.0 (2020-02-06)

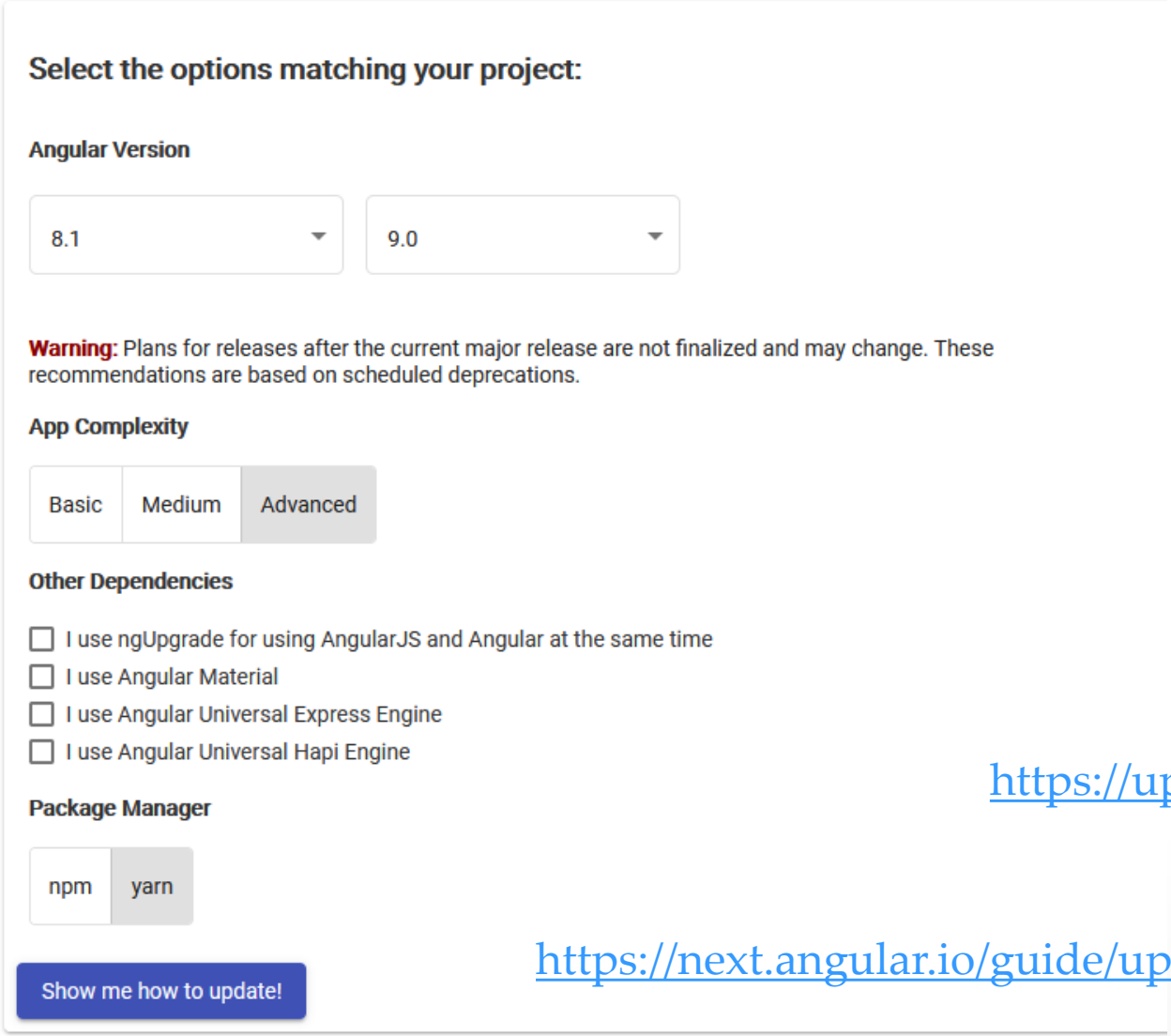
The following table contains our current target release dates for the next two major versions of Angular:

DATE	STABLE RELEASE	COMPATIBILITY
October/November 2019	9.0.0	^8.0.0
May 2020	10.0.0	^9.0.0

Angular 9


angular 9 announcement
angular 9 beta
angular 9 RC1
angular 9 RC9
angular 9 RC20

Angular 9 migration



https://update.angular.io/

https://next.angular.io/guide/updating-to-version-9

Angular 9

- ✓ Released in February 2020
- ✓ Current version 9.1
- ✓ Ivy renderer by default (replacing old View Engine)
- ✓ AOT mode by default
- ✓ Selector-less directives
- ✓ Language Service improvements
- ✓ I18N
- ✓ Renderer type and related API removed (deprecated in v4)
- ✓ 101 features including 34 Ivy-related
- ✓ 95 performance improvements including 84 Ivy-related
- ✓ TypeScript 3.8

Angular 10

- Released in June 2020
- Current version 10.0 RC 6
- TypeScript 3.9
- Angular Language service improvements
- Performance improvements

Angular. Project migration

- Sergiy Morenets, 2020

Benchmarks 2019. Duration(ms)

Name Duration for...	aurelia-v1.3.0-keyed	vue-v2.6.2-keyed	react-v16.8.6-keyed	angular-v8.0.1-keyed	angular-ivy-v7.1.4-keyed	angularjs-v1.7.8-keyed	knockout-v3.5.0-keyed	ember-v3.11.1-keyed
create rows creating 1,000 rows	149.5 ± 1.9 (1.00)	162.3 ± 3.4 (1.09)	165.7 ± 3.8 (1.11)	164.3 ± 7.6 (1.10)	178.2 ± 5.1 (1.19)	187.3 ± 5.6 (1.25)	293.1 ± 10.0 (1.96)	323.6 ± 6.1 (2.16)
replace all rows updating all 1,000 rows (5 warmup runs).	130.8 ± 1.7 (1.02)	128.2 ± 2.5 (1.00)	127.8 ± 1.5 (1.00)	134.4 ± 2.4 (1.05)	134.7 ± 1.5 (1.05)	156.5 ± 2.2 (1.22)	219.2 ± 4.2 (1.72)	175.1 ± 2.8 (1.37)
partial update updating every 10th row for 1,000 rows (3 warmup runs). 16x CPU slowdown.	145.2 ± 4.3 (1.00)	230.9 ± 6.1 (1.59)	162.2 ± 5.7 (1.12)	147.0 ± 7.3 (1.01)	161.5 ± 3.1 (1.11)	164.2 ± 8.1 (1.13)	148.0 ± 4.9 (1.02)	173.4 ± 8.1 (1.19)
select row highlighting a selected row. (5 warmup runs). 16x CPU slowdown.	48.0 ± 2.2 (1.76)	104.7 ± 1.7 (3.83)	31.1 ± 1.5 (1.14)	27.3 ± 2.3 (1.00)	41.7 ± 3.3 (1.53)	41.4 ± 2.5 (1.51)	83.9 ± 2.6 (3.07)	49.5 ± 1.8 (1.81)

- Sergiy Morenets, 2020

Benchmarks 2019. Duration(ms)

Startup metrics (lighthouse with mobile simulation)

Name	aurelia-v1.3.0-keyed	vue-v2.6.2-keyed	react-v16.8.6-keyed	angular-v8.0.1-keyed	angular-ivy-v7.1.4-keyed	angularjs-v1.7.8-keyed	knockout-v3.5.0-keyed	ember-v3.11.1-keyed
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	3,436.4 ± 14.8 (1.55)	2,313.3 ± 39.8 (1.04)	2,515.6 ± 14.1 (1.13)	2,885.4 ± 9.3 (1.30)	3,161.7 ± 5.0 (1.42)	2,861.4 ± 4.8 (1.29)	2,220.0 ± 76.1 (1.00)	4,542.8 ± 10.9 (2.05)
script bootup time the total ms required to parse/compile/evaluate all the page's scripts	260.9 ± 28.9 (4.70)	59.6 ± 28.6 (1.07)	55.6 ± 45.2 (1.00)	159.8 ± 8.8 (2.88)	75.2 ± 39.4 (1.35)	128.4 ± 10.4 (2.31)	63.7 ± 33.7 (1.15)	213.8 ± 9.4 (3.85)
total kilobyte weight network transfer cost (post-compression) of all the resources loaded into the page.	439.0 ± 0.0 (2.11)	211.2 ± 0.0 (1.01)	260.8 ± 0.0 (1.25)	295.5 ± 0.0 (1.42)	388.3 ± 0.0 (1.87)	324.4 ± 0.0 (1.56)	208.0 ± 0.0 (1.00)	577.4 ± 0.0 (2.78)

- Sergiy Morenets, 2020

Benchmarks 2019. Duration(ms)

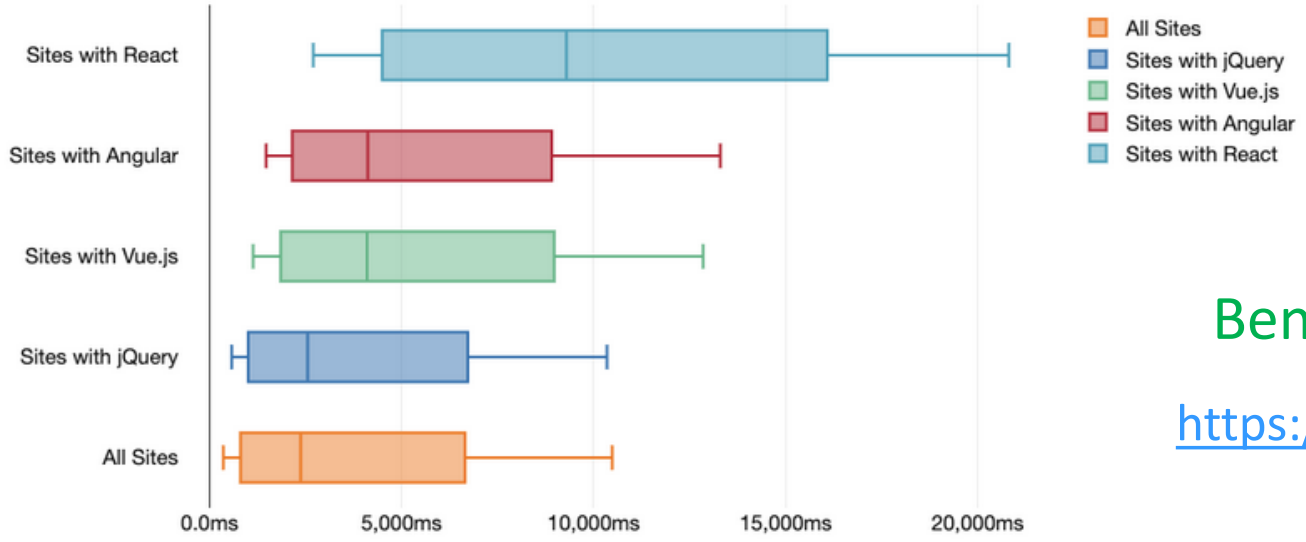
Memory allocation in MBs ± 95% confidence interval

Name	aurelia-v1.3.0-keyed	vue-v2.6.2-keyed	react-v16.8.6-keyed	angular-v8.0.1-keyed	angular-ivy-v7.1.4-keyed	angularjs-v1.7.8-keyed	knockout-v3.5.0-keyed	ember-v3.11.1-keyed
ready memory Memory usage after page load.	3.6 ± 0.0 (1.72)	2.1 ± 0.0 (1.00)	2.3 ± 0.0 (1.09)	4.8 ± 0.0 (2.24)	2.7 ± 0.0 (1.27)	2.8 ± 0.0 (1.33)	2.1 ± 0.0 (1.00)	4.7 ± 0.0 (2.20)
run memory Memory usage after adding 1000 rows.	8.3 ± 0.0 (1.31)	7.1 ± 0.0 (1.12)	6.9 ± 0.0 (1.09)	9.1 ± 0.0 (1.45)	6.3 ± 0.0 (1.00)	10.6 ± 0.0 (1.68)	22.2 ± 0.0 (3.51)	13.7 ± 0.0 (2.17)
update eatch 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	8.6 ± 0.0 (1.27)	7.5 ± 0.0 (1.11)	8.0 ± 0.0 (1.19)	9.5 ± 0.0 (1.41)	6.8 ± 0.0 (1.00)	10.9 ± 0.0 (1.62)	22.7 ± 0.1 (3.36)	14.0 ± 0.2 (2.07)
replace 1k rows (5 cycles) Memory usage after clicking create 1000 rows 5 times	9.0 ± 0.0 (1.23)	7.7 ± 0.0 (1.05)	8.9 ± 0.0 (1.21)	9.9 ± 0.1 (1.35)	7.3 ± 0.0 (1.00)	11.5 ± 0.0 (1.57)	14.4 ± 0.1 (1.97)	14.7 ± 0.0 (2.01)

- Sergiy Morenets, 2020

Scripting related CPU time (in milliseconds) for mobile devices, in percentiles

	10TH	25TH	50TH	75TH	90TH
All Sites	356.4ms	959.7ms	2,372.1ms	5,367.3ms	10,485.8ms
Sites with jQuery	575.3ms	1,147.4ms	2,555.9ms	5,511.0ms	10,349.4ms
Sites with Vue.js	1,130.0ms	2,087.9ms	4,100.4ms	7,676.1ms	12,849.4ms
Sites with Angular	1,471.3ms	2,380.1ms	4,118.6ms	7,450.8ms	13,296.4ms
Sites with React	2,700.1ms	5,090.3ms	9,287.6ms	14,509.6ms	20,813.3ms

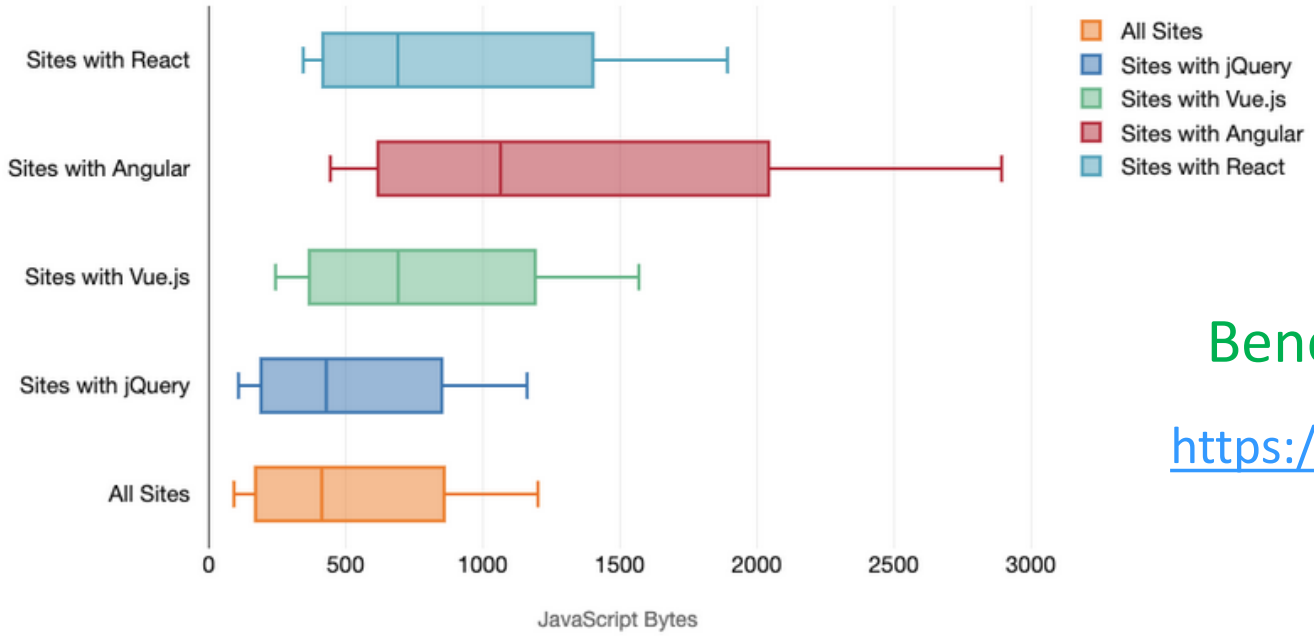


Benchmarks 2020

https://httparchive.org/

JavaScript Bytes Served to Mobile Devices, by Percentile

	10TH	25TH	50TH	75TH	90TH
All Sites	93.4kb	196.6kb	413.5kb	746.8kb	1,201.6kb
Sites with jQuery	110.3kb	219.8kb	430.4kb	748.6kb	1,162.3kb
Sites with Vue.js	244.7kb	409.3kb	692.1kb	1,065.5kb	1,570.7kb
Sites with Angular	445.1kb	675.6kb	1,066.4kb	1,761.5kb	2,893.2kb
Sites with React	345.8kb	441.6kb	690.3kb	1,238.5kb	1,893.6kb



Benchmarks 2020

https://httparchive.org/

Angular

- ✓ Object-oriented programming
- ✓ Static typing
- ✓ Generics
- ✓ Lambdas
- ✓ Improved DI (services)
- ✓ Iterators
- ✓ Decorators
- ✓ Angular CLI
- ✓ Angular Universal

Angular CLI

- ✓ Avoid routine tasks
- ✓ Setup application from scratch
- ✓ Continuation of **Yeoman** best ideas
- ✓ Command-line utility to provide packaging, build tools, unit-testing, etc
- ✓ Similar to **ember-cli** of EmberJS
- ✓ Uses Webpack Terser plugin to minify the code
- ✓ Current version 9.1.0

Angular-cli. Commands

Command	Description
ng build	Build the project and puts bundles into output folder
ng generate	Generates(or modified) files based on predefined schemas
ng new	Creates new Angular workspace
ng serve	Builds and run the project using webpack-dev-server, detecting file changes
ng test	Run unit-tests of the project
ng eject	Extract Webpack configuration and scripts
ng lint	Lint your project using tslint
ng e2e	Runs end-to-end tests
ng update	Updates application and its dependencies
ng config	Changes or retrieves Angular configuration
ng add	Adds library(s) to the project dependencies

Angular-cli. Examples

- ng **new** new-ws
- ng generate **application** new-application
- ng generate **class** new-class
- ng generate **component** new-component
- ng generate **directive** new-directive
- ng generate **enum** new-enum
- ng generate **interface** new-interface
- ng generate **library** new-library
- ng generate **module** new-module
- ng generate **pipe** new-pipe
- ng generate **service** new-service

ng g s new-service

Schematics

- ✓ Workflow tool for project transformations (creating new elements or updating configuration)
- ✓ Custom template generator (for example, instead of default Angular template)
- ✓ Automate boilerplate code
- ✓ Used by Angular CLI

Linting


```
@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.css']
})
export class BookComponent{
```

TSLint: missing whitespace (one-line)

```
import {Book} from "../book/book";
```

TSLint: " should be ' (quotemark)

```
10      item: string = 'Value';
```

TSLint: Type string trivially inferred from a string literal, remove type annotation (no-inferrable-types)

```
if (this.item == 'Value') {
```

TSLint: == should be === (triple-equals)

```
  console.log("Matches");
}
```

TSLint: " should be ' (quotemark)

Linting. TSLint and ESLint

- TSLinit was default linting tool in Angular
- TSLint was deprecated in February 2019 and closed in January 2020
- ESLint became default linter for JavaScript/TypeScript
- ESLint will be default linter in Angular 10

Linting. ESLint migration

- ✓ *npx tslint-to-eslint-config* is migration command
- ✓ Default parser for ESLint became @typescript-eslint/parser

Build results

- ✓ polyfills.js ⟵ Compatibility with old browsers
- ✓ scripts.js ⟵ Scripts declared in “scripts” section of angular.json
- ✓ runtime.js ⟵ Webpack loader
- ✓ styles.css ⟵ Styles declared in “styles” section of angular.json
- ✓ main.js ⟵ Application code & 3rd party modules

Task #9. Angular CLI

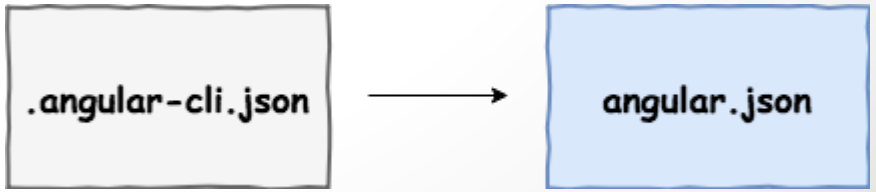
1. Install globally Angular CLI
2. Create new Angular project. What is the size and overall file/folder number in **node_modules** folder?
3. Run web development server: *ng serve* in the project folder
4. Disable specifications (tests) for your new components
5. Run *ng build* command and review contents of the **dist** folder

Configuration files

File	Purpose
angular.json	Angular CLI configuration
package.json	NPM package configuration
tslint.json	Linting configuration
tsconfig.json	TypeScript compiler configuration
karma.conf.js	Karma configuration
protractor.conf.js	Protractor(e2e testing) configuration
environment.ts	Default environment configuration(dev | prod)

Angular.json structure

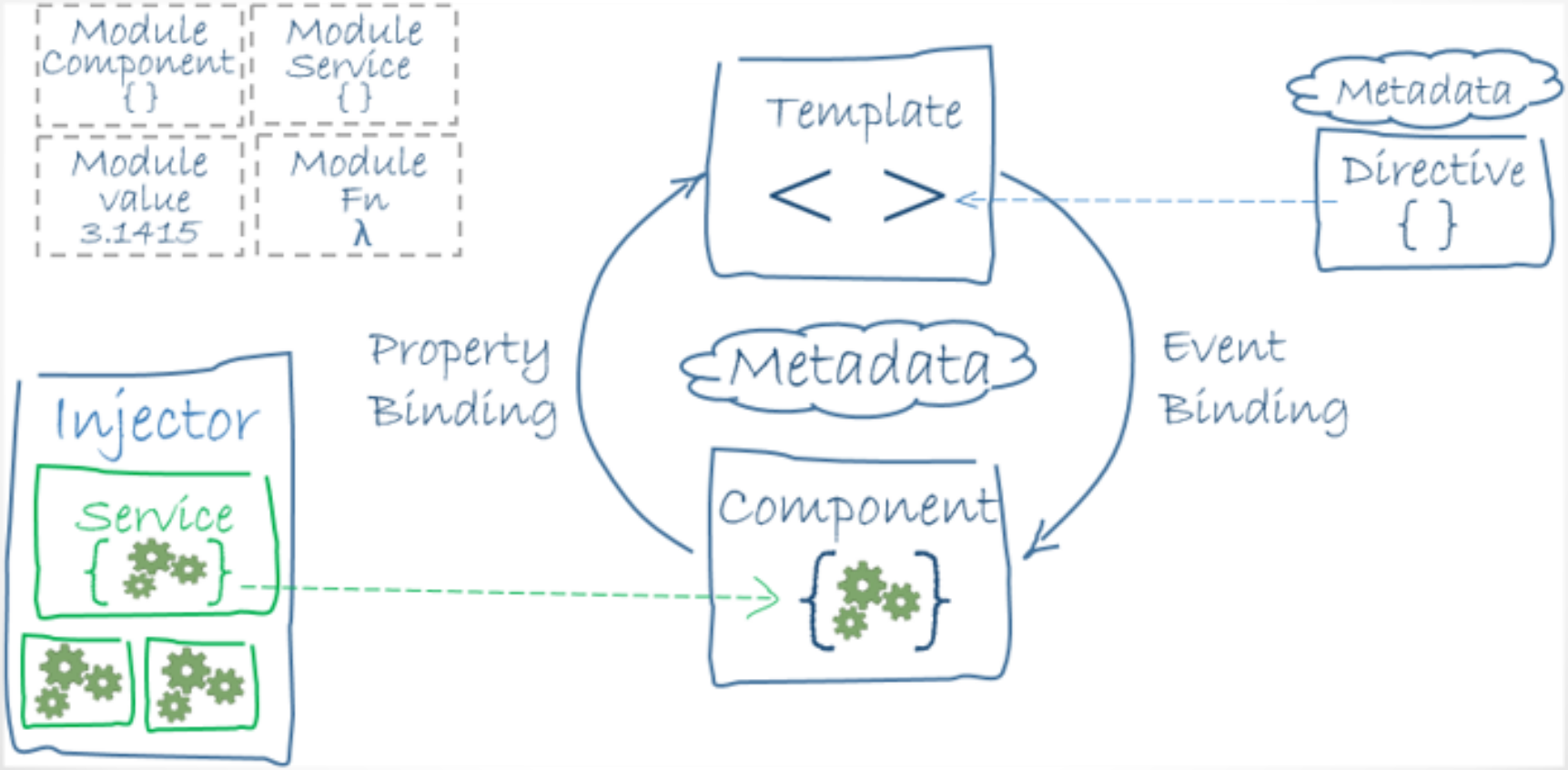
property	Description
schematics	Configuration of Schematics packages, workflow tool and part of Angular DevKit
packageManager	npm or yarn
projects	Configuration of workspace projects
root	Main directory with project files
sourceRoot	Directory with source files (**src** by default)
projectType	application or library
prefix	Custom prefix for our components/directives
build	Build settings (custom styles, scripts)



Angular. Modules

- Application should have at least one **root** module and multiple **feature** modules
- Root module is responsible for **bootstrapping** and launching the application
- Other type of modules are routing, service, features, shared, core and widget modules
- **FormsModule**, HttpModule and RouterModule are built-in modules
- Angular modules functionally separates the project
- Module can export components, directives and pipes

Angular. Building blocks



- Sergiy Morenets, 2020

Angular packages

Module name	Description
@angular/core	Provides DI, decorators, types for components, views, rendering, change detection and event handling
@angular/common	Basic directives and pipes
@angular/compiler	View and module compiler
@angular/forms	Form validation and data binding
@angular/http	Communication with back-end services over HTTP
@angular/platform-browser	Support for different browsers
@angular/language-service	Auto-completion, error checking and navigation for code editor
@angular/router	Routing and navigation

3rd party packages

Module name	Description
zone.js	Isolated zoning
RxJS	Supports asynchronous programming
core.js	Supports decorators metadata (Reflect API) and provides polyfills for them. Can be used instread of reflect-metadata

Angular. Module


```
import {BrowserModule} from '@angular/platform-browser';
import {NgModule} from '@angular/core';
import {AppComponent} from './app.component';

@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
  bootstrap: [AppComponent]
})
export class AppModule {
}
```

Decorator

All modules components, pipes, directives

Root component in index.html

src/app/app.module.ts

Angular. Decorators

- ✓ Can be class decorators **(@NgModule**), property decorator (@Input), method decorator (@HostListener) and parameters decorator (@Inject)
- ✓ Stores metadata information about Angular element in declarative way
- ✓ Supported by TypeScript

```
"emitDecoratorMetadata": true,
"experimentalDecorators": true,
```

tsconfig.json

Angular. NgModule

Attribute	Description
providers	List of objects that can be injected
declarations	List of directives/pipes/components of the module
imports	List of imported modules
exports	List of exported/pipes/modules that can be reused in the module that imports current module
bootstrap	List of bootstrapped components
id	Identifier or path of the module
entryComponents	List of dynamically constructed components

Angular. Bootstrapping


```
import {AppModule} from './app/app.module';
import {platformBrowserDynamic} from '@angular/platform-browser-dynamic';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

Can be production/development

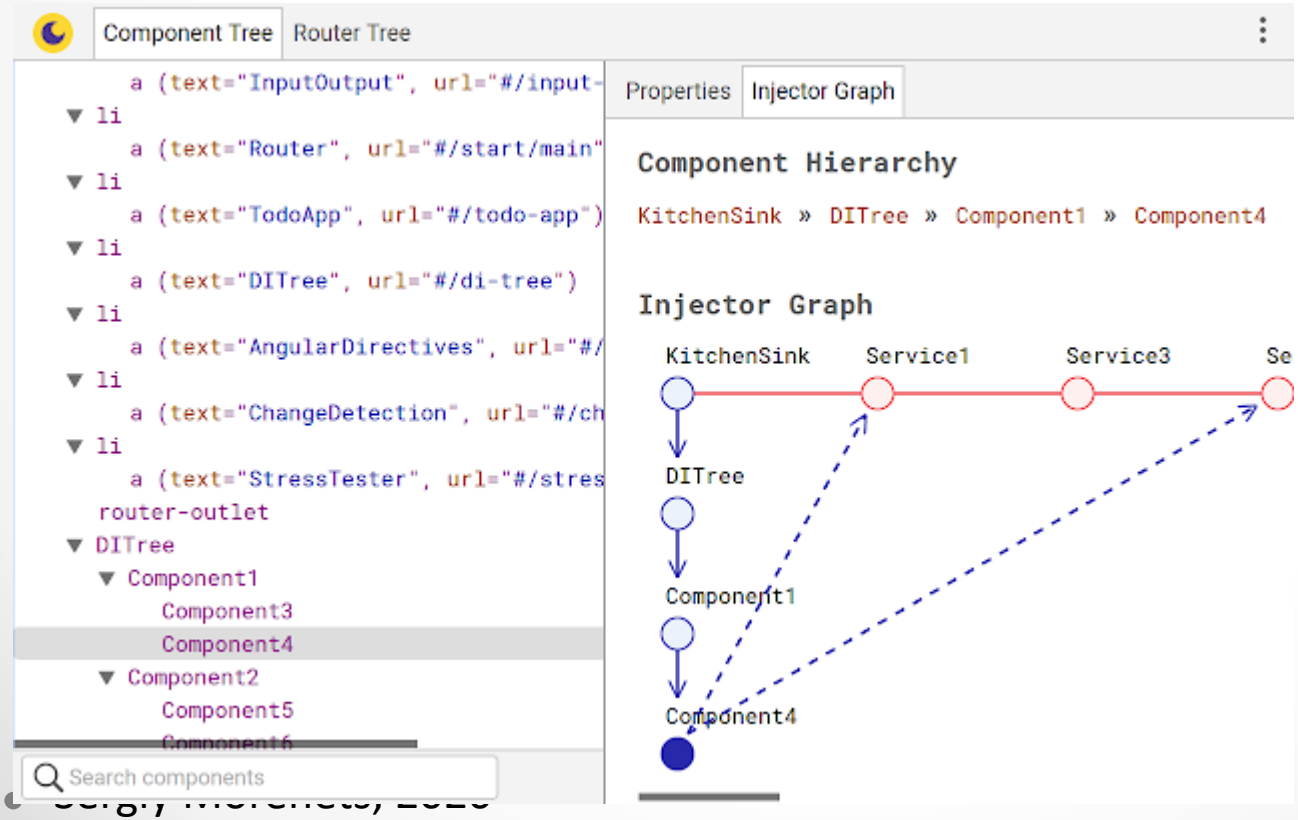
src/main.ts

Element naming

- EventsComponent
- FontDirective
- UsersModule
- OrderPipe
- PaymentService

Augury extension

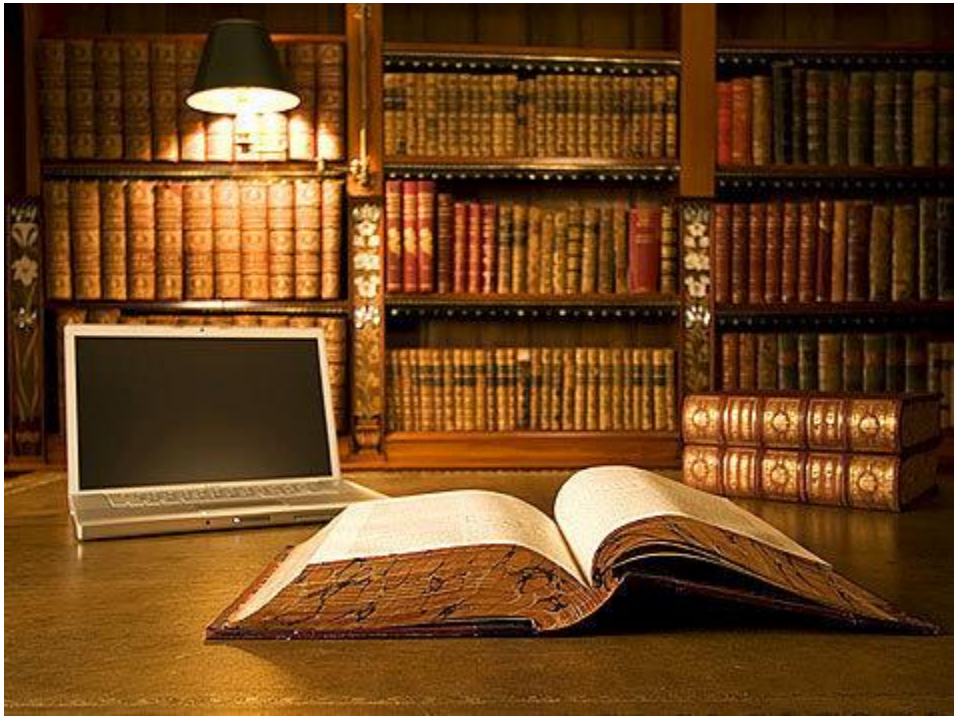
- ✓ Browser extension for debugging/profiling Angular applications
- ✓ Visualization of the components tree



Task #10. WebStorm and Angular

1. Open created Angular project in **WebStorm** (File -> Open)
2. Review project structure and contents. Which Angular modules do you have now?
3. Set "**strictNullCheck**" property to "true" in "compilerOptions" element in tsconfig.json.
4. Open **app.module.ts** and go to the declaration of standard Angular types

Business domain



- Sergiy Morenets, 2020

Index.html

```html
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Angular</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
    <app-root></app-root>
</body>
</html>
```

Bootstrap component selector (AppComponent)

```
<app-panel></app-panel>
<app-dashboard></app-dashboard>
```

app.component.html

Angular components

- ✓ Contains model, view (template) and controller (model and view update)
- ✓ Recognized by **@Component** decorator

Definition(class)

- app.component.cs

Template

- app.component.html

Styles

- app.component.css

Specification (tests)

- app.component.spec.ts

Component declaration

Global prefix

```
@Component({
  selector: 'app-root',
  template: '<div>Body</div>',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
```

Inlined template

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
```

template

styles (optional)

Components. Data binding

app.component.ts

```
export class AppComponent {
  title = 'Hello, world!';

  user: object = {name: 'John'};
```

```
export class AppComponent {
  title: string;

  constructor() {
    this.title = 'Hello, world!';
  }
```

app.component.html

```
<div>{{title}}</div>
<div>{{user.name}}</div>
```

Data binding. Safe programming

app.component.ts

```ts
export class OrderComponent implements OnInit {
  order: Order;
```

app.component.html

```html
<div>ID: {{order.id}}</div>
<div>Amount: {{order.amount}}</div>
```

```
ERROR TypeError: Cannot read property 'id' of undefined core.js:6185
    at OrderComponent_Template (order.component.html:2)
    at executeTemplate (core.js:11949)
```

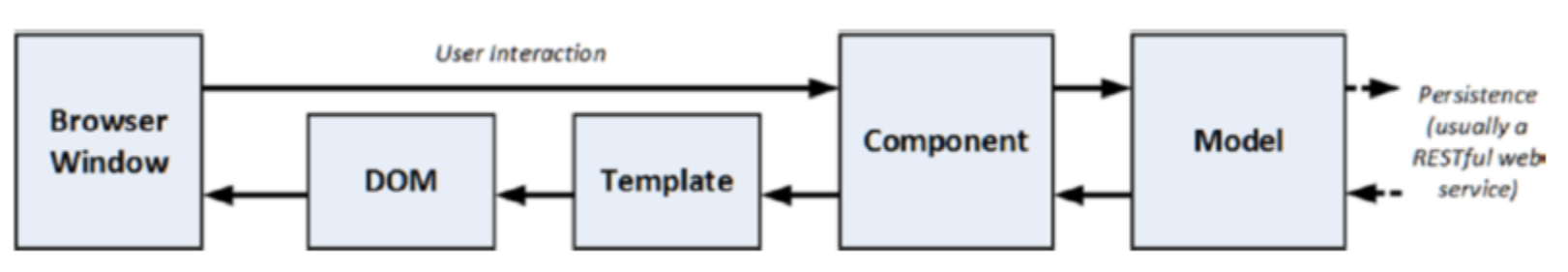
```html
<div>ID: {{order?.id}}</div>
<div>Amount: {{order?.amount}}</div>
```

Angular feature

- Sergiy Morenets, 2020

Data binding

- Each HTML element is converted into corresponding DOM node
- Each element attribute has matching DOM property
- Special DOM properties can be accessed using [] syntax, for example, [textContent], [selected] or [hidden]



Two-way data binding. ngModel

```
export class OrderComponent implements OnInit {
  extendedMode: boolean;
```

One-way binding

```
<div>Extended mode: {{extendedMode}}</div>

<input [ngModel]="extendedMode">
```

```
error NG8002: Can't bind to 'ngModel' since it isn't a known property of 'input'.
```

```
@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
```

```
<input [ngModel]="extendedMode">
```

One-way binding

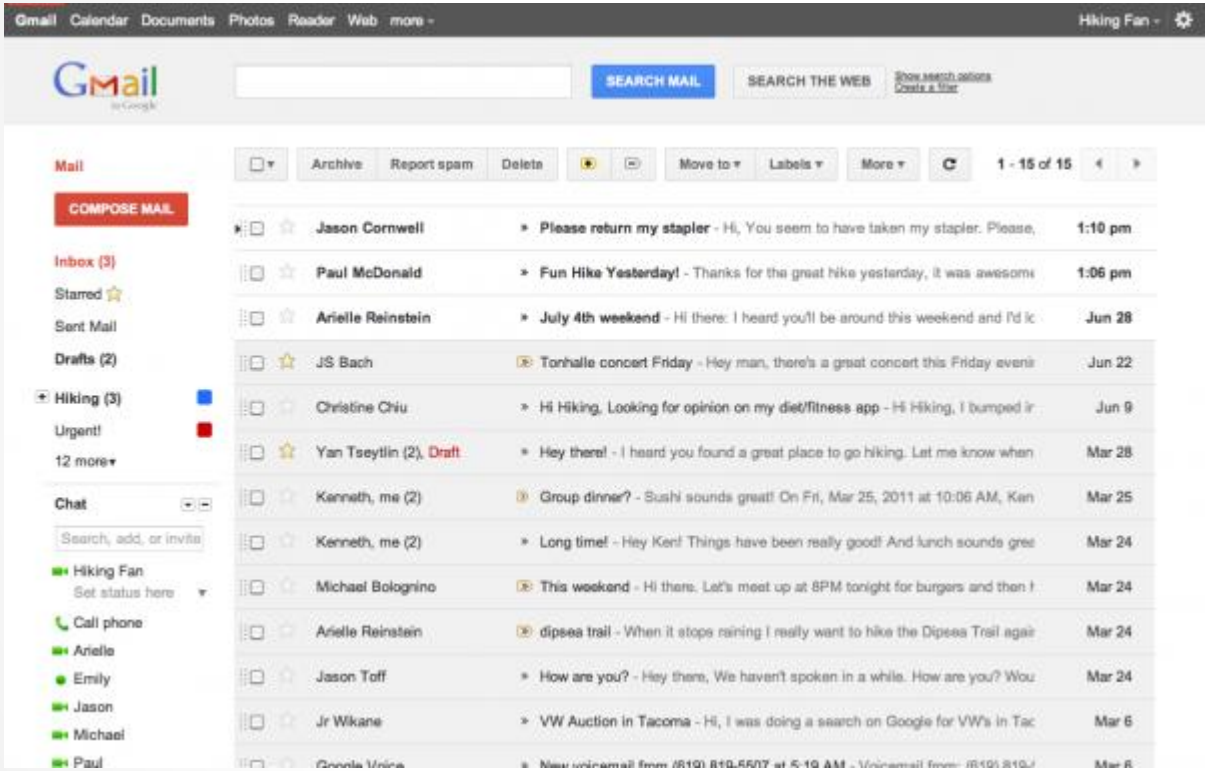
Two-way data binding. ngModel


```
<div>Extended mode: {{extendedMode}}</div>
<label> Extended mode:
  <input [(ngModel)]="extendedMode">
</label>
```

Two-way binding:

1. Model -> Template
2. Template -> Model

Components composition



Page should be split into multiple components (tree)

- Sergiy Morenets, 2020

Task #11. Data binding

1. Create object that stores **book** attributes (title, year, author, pages, description) in your component.
2. Update component template to display property values.
3. What will happen if you put incorrect property name in the template? Review browser console in all the cases.
4. Create new class **Book** and use it to store book properties instead of object.