LORENZO SAVINI
Software developer

Sviluppo di un'interfaccia web con TypeScript, React e Webpack

Develer webinar

07/10/2020

develer

OBIETTIVI DEL WEBINAR

- Introdurre il concetto di bundling con Webpack
- Prendere familiarità con le basi di TypeScript e le differenze con un'applicazione ES6
- Vedere da vicino l'integrazione con React

LE TECNOLOGIE

- TypeScript
- React
- Webpack

PROGRAMMA DEL WEBINAR

- Applicazioni JavaScript moderne
- Introduzione a Webpack
- TypeScript e le basi
- TypeScript & React
- Struttura del progetto

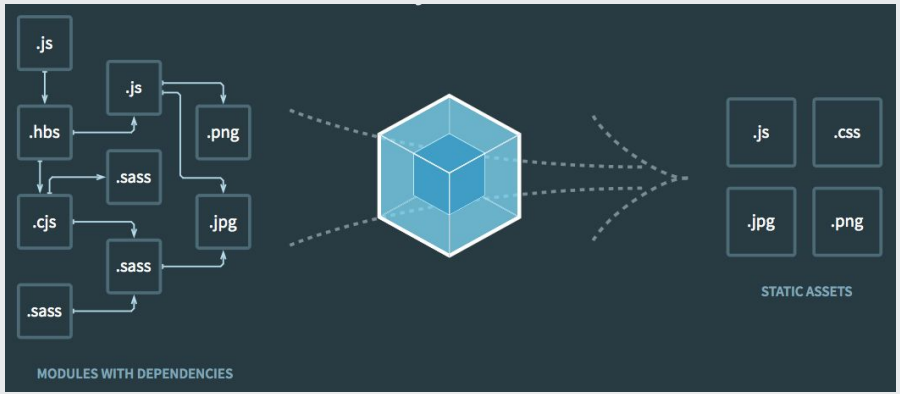
¿App JS moderne?

Un'applicazione JavaScript può essere definita "moderna" quando:

- è scritta in una recente versione di ECMAScript (beh, oppure TypeScript);
- utilizza un tool di build e bundling;
- è modulare, e quindi deve essere prodotto un bundle finale;
- utilizza un package manager (e non una miriade di tag script in fondo al body);
- viene transpilata a build time;
- [...altre caratteristiche non rilevanti...]

Webpack

Webpack



Webpack è un tool per produrre dei bundle, partendo da codice sorgente modulare.

Si occupa di JavaScript ma anche TypeScript, JSX, SASS, CSS, immagini e molto altro.

Webpack

Si basa su 2 concetti:

- **Loaders**: utili per pre-processare i file sorgente, ne esistono per TypeScript, per gli styles, per i file, ...
- **Plugins**: per eseguire altre azioni durante la fase di building/bundling (generazione manifest, iniezione di variabili d'ambiente, ...)

TypeScript

TypeScript

- TypeScript è un **superset tipizzato del JavaScript**, ovvero un dialetto che espone più API del JavaScript stesso;
- Permette di definire il tipo di variabili, parametri delle funzioni, ritorno delle funzioni, semplificando molto lo sviluppo di interfacce web complesse e la comprensione del codice;
- Lo **static** type checking avviene a **compile time**;
- Il type system è **strutturale**;

TypeScript

- Il suo compiler genera codice JavaScript comprensibile ai moderni browser;
- Supporta JSX (richiede configurazione);
- I file di codice TypeScript hanno estensione propria. `.tsx` per i file che richiedono anche sintassi JSX, `.ts` per i file che invece non richiedono JSX;
- Linter integrabile in `eslint`.

Type annotation - TypeScript

```ts
const x: string = "fake";

const sum = (
    param1: number,
    param2: number
): number => param1 + param2;

const obj: {
   field: string | number;
} = {
   field: "fake",
};
```

È l'operazione di definire il tipo di una variabile, parametro o property.

Feedback su errori

- Grazie alle annotazioni presenti nel codice, il compiler TS è in grado di analizzare il codice e segnalare errori.

Feedback su errori - TypeScript

Definiamo `x` come `string` e proviamo a popolarlo con un intero.

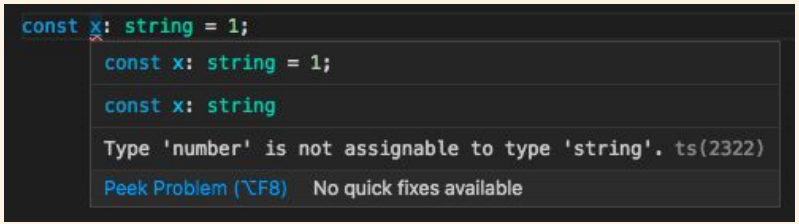
Questo è il feedback di Visual Studio Code:

```
const x: string = 1;
```

Feedback su errori - TypeScript

```
ERROR in [at-loader] ./src/components/EntriesList.tsx:66:11
    TS2322: Type '1' is not assignable to type 'string'.
```

Mentre qui vediamo il feedback del compiler TS, che marcherà anche la compilazione come fallita.



Lo stesso messaggio può essere visualizzato tramite VS Code facendo hover su `x`.

Built-in types

- TypeScript espone alcuni tipi standard

Built-in types - TypeScript

Boolean

```
const x: boolean = true;
```

String

```
const y: string = "asd";
```

Number

```
const k: number = 1;
```

Array

```
const j: string[] = ["a", "b"];
```

Numeric literal types

```
const x: 1 = 1;
```

String literal types

```
const s: "foo" | "bar" = "bar";
```

Enums - TypeScript

TypeScript espone la possibilità di dichiarare enums

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

function move(direction: Direction): void {
    // ...
}

move(Direction.Left);
```

Null e undefined - TypeScript

Ovviamente si può anche indicare se un elemento potrebbe essere `null` o `undefined`.

```
const x: string | null = "fake";
```

Optional - TypeScript

Oppure indicare un parametro di funzione o una property come opzionale, usando il suffisso ?

```
const func = (
    required: string, optional?: number
): false => false;

interface Entry {
    required: string;
    optional?: number;
}
```

Programmazione ad oggetti

- TypeScript permette di dichiarare classi, interfacce

Interfacce - TypeScript

TypeScript permette di dichiarare interfacce

```
interface ToDoEntry {
   description: string;
   dueDate: Date;
}

interface ToDoAdvancedEntry extends ToDoEntry {
   tags: string[];
   doThings(param: string): void;
}
```

Interfacce - TypeScript

Possono essere utilizzate per annotare oggetti

```
const entry: ToDoAdvancedEntry = {
    description: "test",
    dueDate: new Date(),
    tags: [],
    doThings(param: string): void {
        console.log(param);
    }
}
```

Interfacce - TypeScript

Si possono anche dichiarare interfacce indicizzate

```
interface MyDictionary {
    [key: string]: any;
}
```

Classi - TypeScript

TypeScript permette di dichiarare classi

```
class ToDoEntry {
    private field: string = "fooBar";
    constructor(field: string) {
        this.field = field;
    }
    public isCompleted(): boolean {
        return true;
    };
}

const toDoEntry = new ToDoEntry("test");
```

Generics - TypeScript

Aumentando la complessità, vediamo le possibilità che dà TypeScript con i generics, al fine di migliorare soprattutto la riusabilità del codice.

```
interface Elem {
    some: string;
}

const identity = <T  extends Elem>(
    elem: T
): T => elem;

const elem = identity<Elem>(
    {some: "string"}
);
```

Modules - TypeScript

Per facilitare lo sviluppo di app modulari, TypeScript fornisce le primitive `export` e `import`.

`Export` permette di esporre `const`/`let`, funzioni, classi, interfacce dall'interno di un modulo.

```
// file: src/entry.ts
export interface IEntry {
    some: string;
}

// file: src/index.ts
import { IEntry } from "./entry";
```

Declaration files - TypeScript

Pacchetti non scritti in TypeScript possono comunque fornire dei declaration files. Questo permette a TypeScript di fornire type hinting su ciò che espone tale pacchetto.

Questi declaration file possono essere esposti dal pacchetto stesso o aggiungendo, fra le `devDependencies` del nostro package.json, il relativo pacchetto dal namespace `@types/`.

I declaration files hanno estensione `.d.ts`.

E molto altro... - TypeScript

TypeScript offre supporto a molte altre cose:

- ES6 Symbol
- Namespaces
- Decorators
- Readonly
- Mixins
- Aliases
- ecc...

https://www.typescriptlang.org/docs

Considered harmful!

- Vediamo alcuni esempi di bad practices con TypeScript

Abuso di Any - TypeScript

TypeScript supporta il tipo `Any`, che indica al compiler TS che tale variabile può effettivamente essere qualunque cosa.

Da *non usare* come soluzione alternativa quando il tipo corretto risulta troppo complesso da dichiarare.

`Any` esplicito `->` viene inserito nel codice.
`Any` implicito `->` il tipo di un elemento non annotato.

Abuso di Any - TypeScript

Per mitigare l'abuso, è possibile abilitare:

1. La regola `@typescript-eslint/no-explicit-any` di eslint per evitare `Any` espliciti
2. La flag del compiler TS `noImplicitAny` dal tsconfig.json per prevenire `Any` impliciti

Abuso di ! - TypeScript

TypeScript supporta anche il suffisso `!`, che è utile nei casi in cui è previsto che un dato elemento possa essere `null` o `undefined`.

A volte, quando il compilatore non è abbastanza intelligente da capire che avete già scritto del codice per utilizzare quell'elemento in sicurezza, può essere effettivamente necessario.

Abuso di ! - TypeScript

```
interface IEntry {
    some: string;
}

interface IObj {
    entry: IEntry | null;
}
```

Abuso di ! - TypeScript

```
const printSome = (obj: IObj) => {
    console.log(obj.entry.some);
}
```

Se proviamo ad accedere a `obj.entry.some`, la compilazione fallirà perché TS rileverà che `obj.entry` potrebbe essere `null`.

```
ERROR in [at-loader] ./src/app.tsx:11:46
    TS2531: Object is possibly 'null'.
```

Abuso di ! - TypeScript

Per forzare TS ad accettare questa funzione, è necessario un `!` dopo `obj.entry`.

Soluzione? Validare la presenza di `obj.entry`.

```
const printSome = (obj: IObj) => {
    console.log(obj.entry!.some);
};
```

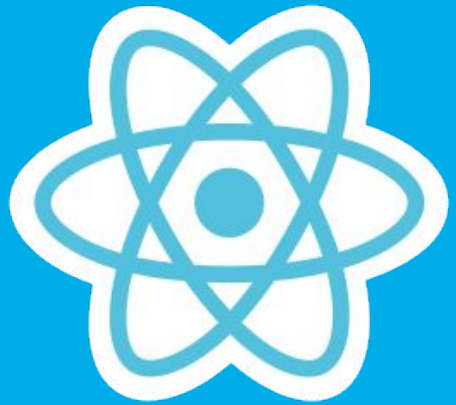
```
const printSome = (obj: IObj) => {
    const text = obj.entry
        ? obj.entry.some : "N/A";
    console.log(text);
};
```

Altre bad practices comuni - TypeScript

Potete trovare altre bad practices direttamente nella doc di TS:

https://www.typescriptlang.org/docs/handbook/declaration-files/do-s-and-don-ts.html

TypeScript & React



TypeScript & React

- Vediamo ora i vantaggi dell'adozione di TypeScript all'interno di applicazioni React

TypeScript & React

Come abbiamo già detto, TypeScript:

- supporta nativamente JSX (serve soltanto una flag del compiler)
- ha bisogno del pacchetto `@types/react` per annotare ciò che React espone

Importiamo React - TypeScript & React

All’interno della nostra applicazione, è possibile importare React utilizzando

```
import * as React from "react";
```

Questo import è necessario anche se non si referenzia React direttamente e può essere utilizzato soltanto all’interno di file `.tsx`.

I tipi base esposti da React sono:

- `React.Component`
- `React.PureComponent`
- `React.ReactNode`

Dichiariamo un Component - TypeScript & React

```
interface IProps {
  prop1: string;
}

interface IState {
  some: string;
}

export class MyComponent extends React.Component<IProps, IState> {
  constructor(props: IProps) {
      super(props);
      this.state = {some: "fake"};
  }
}
```

Implementiamo il metodo render - TypeScript & React

```
export class MyComponent extends React.Component<IProps,
IState> {
    // ...
    public render(): JSX.Element {
        return (
            <div>
                <p>{this.state.some}</p>
            </div>
        );
    }
    // ...
}
```

Getting started

Vediamo insieme un piccolo progetto per creare una To Do list

Requisiti - Getting started

- Node (`lts/erbium`)
- Yarn (o npm)

Repository - Getting started

https://github.com/savo92/ws-typescript-react

Setup - Getting started

```
$ yarn install
$ yarn start
# poi aprite http://localhost:8080
# sul vostro browser
```

Setup - Getting started

```
$ tree -L 1 -I node_modules
.
├── README.md
├── dist
├── package.json
├── prettier.config.js
├── src
├── tsconfig.json
├── webpack
└── yarn.lock
```

package.json - Getting started

```
"dependencies": {
    "@ant-design/icons": "^4.2.2",
    "antd": "^4.6.5",
    "react": "^16.9.0",
    "react-dom": "^16.9.0"
 }
```

package.json - Getting started

```
"devDependencies": {
    "@babel/core": "^7.5.5",
    "@babel/preset-env": "^7.5.5",
    "@babel/preset-react": "^7.0.0",
    "@types/node": "^12",
    "@types/react": "^16",
    "@types/react-dom": "^16",
    "typescript": "^4.0.3",
    "webpack": "^4.39.2",
    "webpack-cli": "^3.3.7",
    "webpack-dev-server": "^3.8.0",
    "webpack-manifest-plugin": "^2.0.4",
    [...]
```

```
    [...]
    "babel-loader": "^8.0.6",
    "css-loader": "^4.3.0",
    "file-loader": "^6.1.0",
    "html-webpack-plugin": "^4.4.1",
    "mini-css-extract-plugin": "^0.11.2",
    "node-sass": "^4.12.0",
    "optimize-css-assets-webpack-plugin": "^5.0.3",
    "sass-loader": "^10.0.2",
    "source-map-loader": "^1.1.0",
    "style-loader": "^1.0.0",
    "terser-webpack-plugin": "^4.2.2",
    "ts-loader": "^8.0.4"
}
```

tsconfig.json - Getting started

Questo file di configurazione descrive le impostazioni del compilatore TypeScript.
Oltre alle cartelle di destinazione e di importazione, è possibile :

- Abilitare **JSX**;
- Importare **librerie** (`es2015`, `es5`, `dom`, `esnext`, ...);
- Definire la versione **target** di ECMAScript;
- Configurare la strategia di risoluzione dei moduli;
- Abilitare altre **flag** del compiler (come `noImplicitAny` o `noUnusedLocals`);

E molto altro, lo schema completo: https://json.schemastore.org/tsconfig.

tsconfig.json - Getting started

```
{
   "compilerOptions": {
       "jsx": "react",
       "module": "es6",
       "moduleResolution": "node",
       "noImplicitAny": true,
       "noUnusedLocals": true,
       "outDir": "./dist/",
       "sourceMap": true,
       "strict": true,
       "target": "es6",
       "lib": ["dom", "es2015", "es5", "esnext.asynciterable"]
   },
   "include": ["./src/**/*"]
}
```

Webpack - Getting started

```
$ tree -L 1 webpack
webpack
├── rules.js
├── webpack.config.js
└── webpack.dev.config.js
```

Webpack rules - Getting started

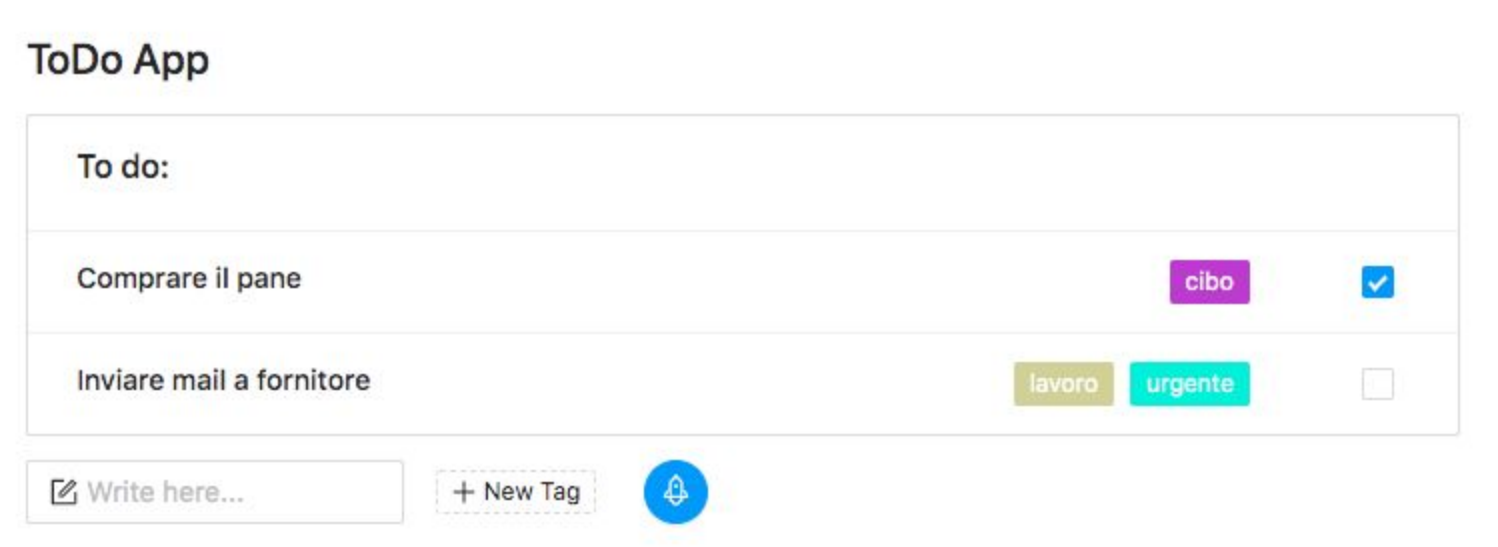
```
const styleLoader =
    process.env.NODE_ENV !== "production"
        ? "style-loader"
        : MiniCssExtractPlugin.loader;

const sassRule = {
    test: /\.scss$/,
    use: [
        { loader: styleLoader },
        { loader: "css-loader" },
        { loader: "sass-loader" },
    ],
};
```

```
const sourceMapRule = {
    enforce: "pre",
    exclude: [/node_modules/, /dist/],
    loader: "source-map-loader",
    test: /\.js$/,
};

const tsRule = {
    loader: "ts-loader",
    test: /\.tsx?$/,
};

const fileRule = {
    test: /\.(eot|jpeg|jpg|png|svg|ttf|woff|woff2)/,
    use: 'file-loader',
};
```

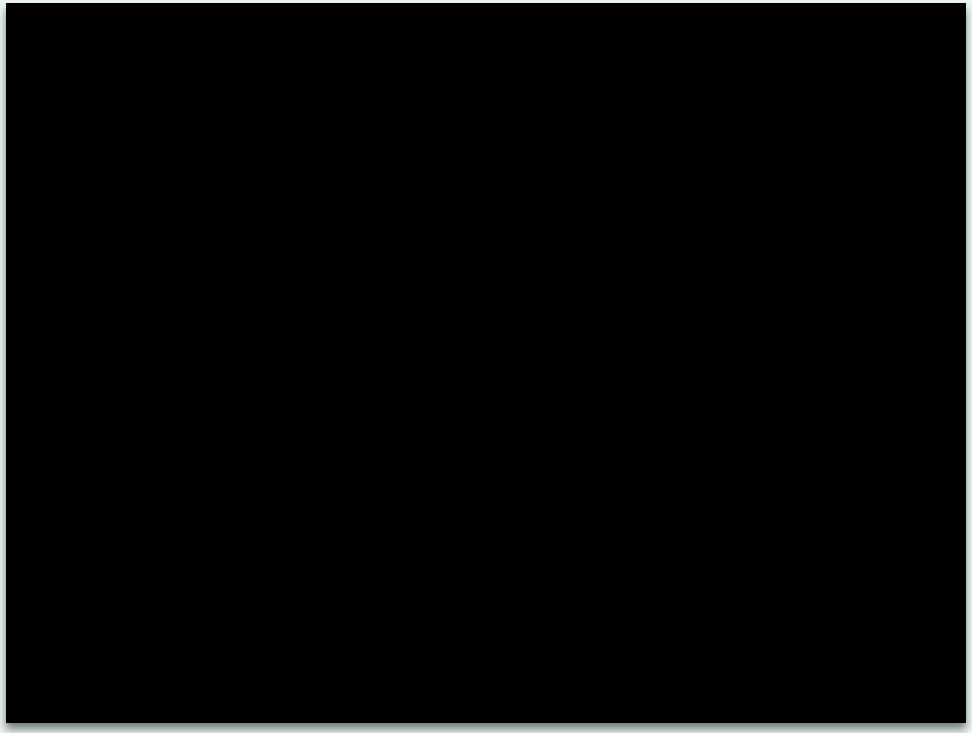
Source code - Getting started

```
$ tree src
src
├── app.tsx
├── components
│   ├── EditableTagGroup.tsx
│   ├── EntriesList.tsx
│   ├── NewEntryForm.tsx
│   └── RootView.tsx
├── index.html
├── index.scss
├── index.tsx
└── interfaces.ts
```

Demo! - Getting started



Demo! - Getting started



Lorenzo Savini

savo@develer.com

Vuoi rimanere aggiornato sugli eventi Develer?
Seguici nei nostri canali social:

develer

www.develer.com