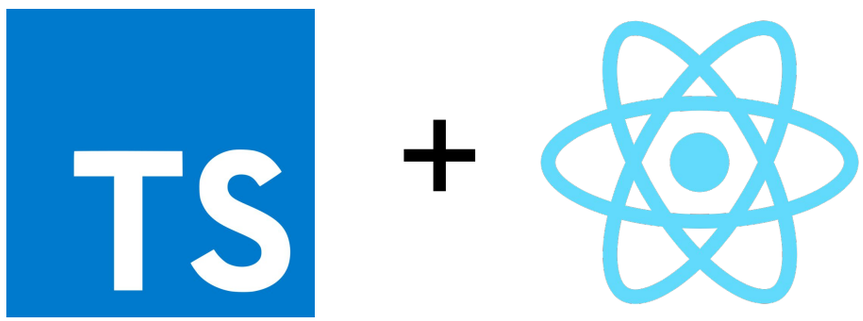
Sviluppo di un'interfaccia web con Typescript, React e Webpack.

Lorenzo Savini
Developer, Develer



develer

Lorenzo Savini savo@develer.com

Obiettivi del workshop

- Introdurre brevemente il concetto di bundling con Webpack;
- Vedere insieme le basi di TypeScript;
- Prendere familiarità con le varie differenze con un’applicazione ES6;
- Fare pratica con l’integrazione con React;

develer

Lorenzo Savini savo@develer.com

Le tecnologie

- **Typescript**: superset tipizzato del Javascript, sviluppato da Microsoft con licenza Apache 2.0. https://www.typescriptlang.org;
- **React**: libreria JavaScript per creare interfacce utente. https://reactjs.org;
- **WebPack**: tool per produrre bundle, pensato per applicazioni JavaScript moderne. https://webpack.js.org;
- **Antd**: [BONUS] una libreria di componenti React, sviluppata in TypeScript. https://github.com/ant-design/ant-design;

Lorenzo Savini savo@develer.com

Programma del workshop

1. Rapido recap del concetto di applicazioni JavaScript moderne;
2. TypeScript e le basi;
3. TypeScript e React;
4. Struttura del progetto;
5. Exercise time!

develer

Lorenzo Savini savo@develer.com

¿App JS moderne?

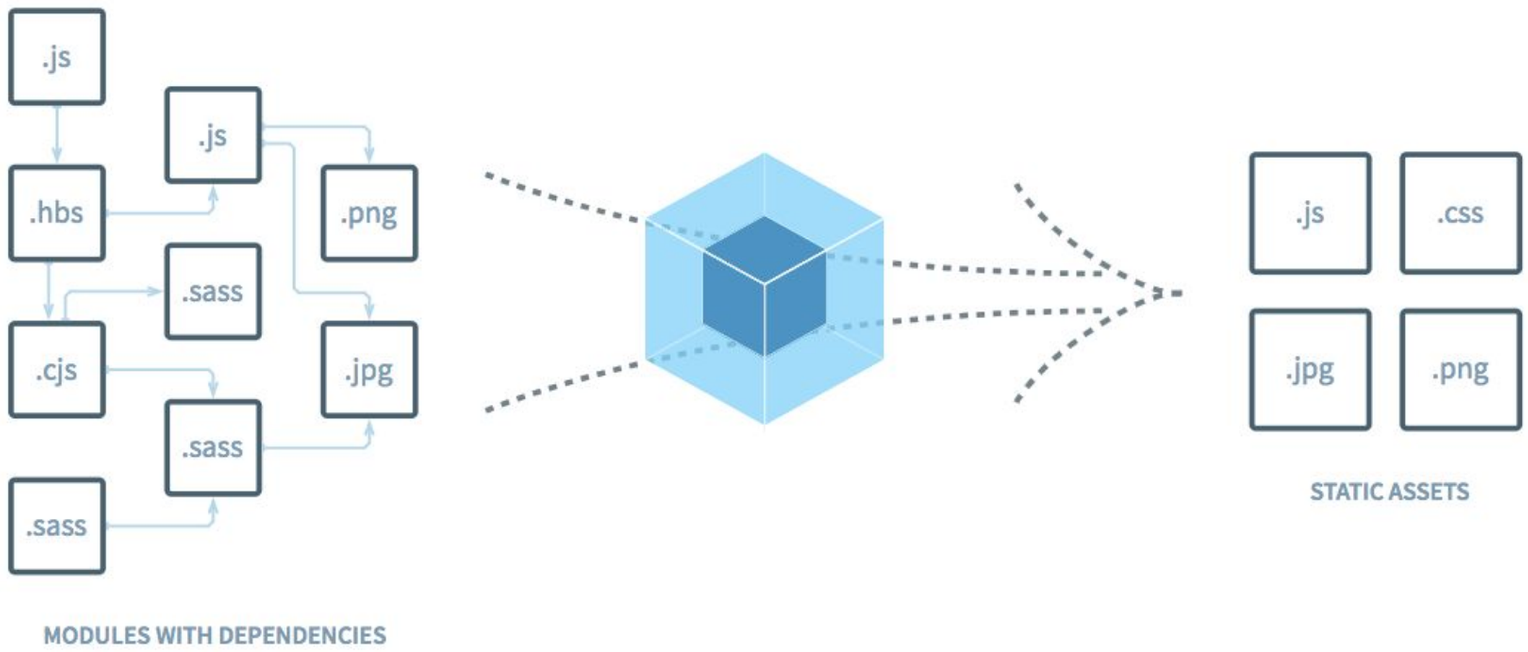
Un’applicazione JavaScript può essere definita “moderna” quando:

- 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...]

develer

Lorenzo Savini savo@develer.com

Webpack



Webpack è un tool per produrre dei **bundle** (destinati ai browser solitamente) partendo da codice sorgente, anche modulare. Si occupa di JavaScript ma anche TypeScript, JSX, SASS, CSS, immagini e molto altro.

develer

Lorenzo Savini savo@develer.com

Webpack

Si basa su 2 concetti: **loaders** e **plugins**.

- 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, …)

Lorenzo Savini savo@develer.com

TypeScript, finally

- 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**.

Lorenzo Savini savo@develer.com

TypeScript, finally

- 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.
- Ha un linter dedicato (tslint) che aiuta a migliorare leggibilità e manutenibilità del codice.

develer

Lorenzo Savini savo@develer.com

Type annotation

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

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


const func = (param1: string, param2: number): string => param1;
```

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

Lorenzo Savini savo@develer.com

Feedback su errori

Grazie alle annotazioni presenti nel codice, il compiler TS è in grado di analizzare il codice e segnalare errori. Vediamo un esempio di errore causato da un'errata assegnazione.
Definiamo "x" come string e proviamo a popolarlo con un intero.
Questo è il feedback di VScode:

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

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

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

Lo stesso messaggio può essere visualizzato tramite VScode facendo hover su "x".

Lorenzo Savini savo@develer.com

Built-in types

TypeScript espone alcuni tipi standard, qua alcuni esempi:

- 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";
```

La doc completa è disponibile qui

https://www.typescriptlang.org/docs/handbook/basic-types.html

Lorenzo Savini savo@develer.com

Enums

TypeScript espone anche la possibilità di dichiarare enums

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

che possono essere usate sia nelle type annotations, che come valori

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

move(Direction.Left);
```

Lorenzo Savini savo@develer.com

Interfacce

TypeScript permette di dichiarare interfacce

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

o estendere altre interfacce

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

Che possono essere utilizzate per annotare oggetti

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

Lorenzo Savini savo@develer.com

Interfacce

Si possono anche dichiarare interfacce indicizzate

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

Lorenzo Savini savo@develer.com

Null, undefined, optional

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

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

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

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

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

Lorenzo Savini savo@develer.com

Classi

TypeScript permette di dichiarare classi

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

che possono essere istanziate con un semplice

```
const x = new ToDoEntry(“test”);
```

Lorenzo Savini savo@develer.com

Generics

Aumentando la complessità, vediamo le possibilità che dà TypeScript con i generics, al fine di migliorare soprattutto la **riusabilità** del codice.
Ad esempio, una funzione che tratta un tipo generico e lo ritorna (la funzione identità ad esempio), può essere dichiarata come

```
interface IElem {
    some: string;
}

const func = <T  extends IElem>(elem: T): T => elem;

const elem = func<IElem>({"some": "string"});
```

Lorenzo Savini savo@develer.com

Modules

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";
```

TypeScript è in grado di importare anche da moduli non TypeScript ma ovviamente non potrà fornire supporto con type checking per il codice che si interfaccia con questi moduli, a meno che non siano forniti anche dei declaration file (".d.ts")

Lorenzo Savini savo@develer.com

Declaration files

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 files possono essere esposti dal pacchetto stesso o aggiungendo, fra le devDependencies del nostro package.json, il relativo pacchetto dal namespace **@types/** (“@types/react” per React stesso).

Come dicevamo precedentemente, questi declaration files avranno estensione “.d.ts”.

develer

Lorenzo Savini savo@develer.com

E molto altro...

TypeScript offre supporto a molte altre cose:

- ES6 Symbol;
- Namespaces;
- Decorators (sperimentali);
- Readonly;
- Mixins;
- Aliases;
- ecc...

Lorenzo Savini savo@develer.com

Considered harmful: abuso di Any

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.

Possiamo definire un Any **esplicito** quando viene inserito nel codice. È **implicito** quando il tipo di un elemento non viene annotato.

Per mitigare l’abuso, è possibile abilitare la regola “no-any” di tslint per evitare Any espliciti oppure abilitare la flag del compiler TS “noImplicitAny” dal tsconfig.json per prevenire Any impliciti.

develer

Lorenzo Savini savo@develer.com

Considered harmful: abuso di !

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.

Vediamo un esempio nella slide successiva...

develer

Lorenzo Savini savo@develer.com

Esempio di !

Dichiariamo queste 2 interfacce:

```
interface IEntry {
    some: string;
}


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

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

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

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

Per forzare TS a accettare questa funzione, è necessario il “!” dopo obj.entry:

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

Lorenzo Savini savo@develer.com

TypeScript & React

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

Lorenzo Savini savo@develer.com

Importiamo 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 utilizzata soltanto all’interno di file .tsx.

Lorenzo Savini savo@develer.com

Tipi base di React

I tipi base esposti da React sono

- React.Component: un Component React;
- React.PureComponent: un Pure Component React.
- React.ReactNode: un nodo dell'albero interno di React;

develer

Lorenzo Savini savo@develer.com

Dichiariamo un Component

```
// Questa interfaccia descrive le props che questo componente dovrà ricevere.
interface IProps {
    prop1: string;
}
// Questa interfaccia descrive la struttura dello stato interno del componente.
interface IState {
    some: string;
}

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

Di fatto, React.PropTypes viene sostituito da TypeScript stesso.

Lorenzo Savini savo@develer.com

Implementiamo il metodo render

```
export class MyComponent extends React.Component<IProps, IState> {
    // ...

    public render(): JSX.Element {
        return (
            <div>
                <p>{this.state.some}</p>
            </div>
        );
    }

    // ...
}
```

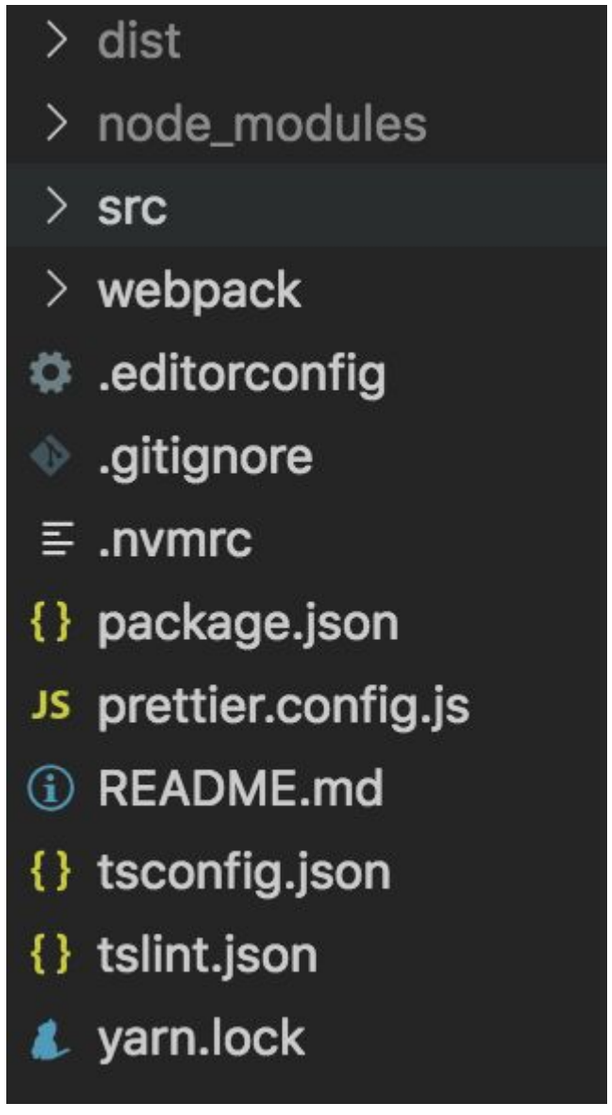
Lorenzo Savini savo@develer.com

Il nostro progetto

Abbiamo parlato delle tecnologie che andremo ad usare tra poco, vediamo ora invece alcuni punti chiave della struttura e della configurazione del nostro progetto.

Lorenzo Savini savo@develer.com

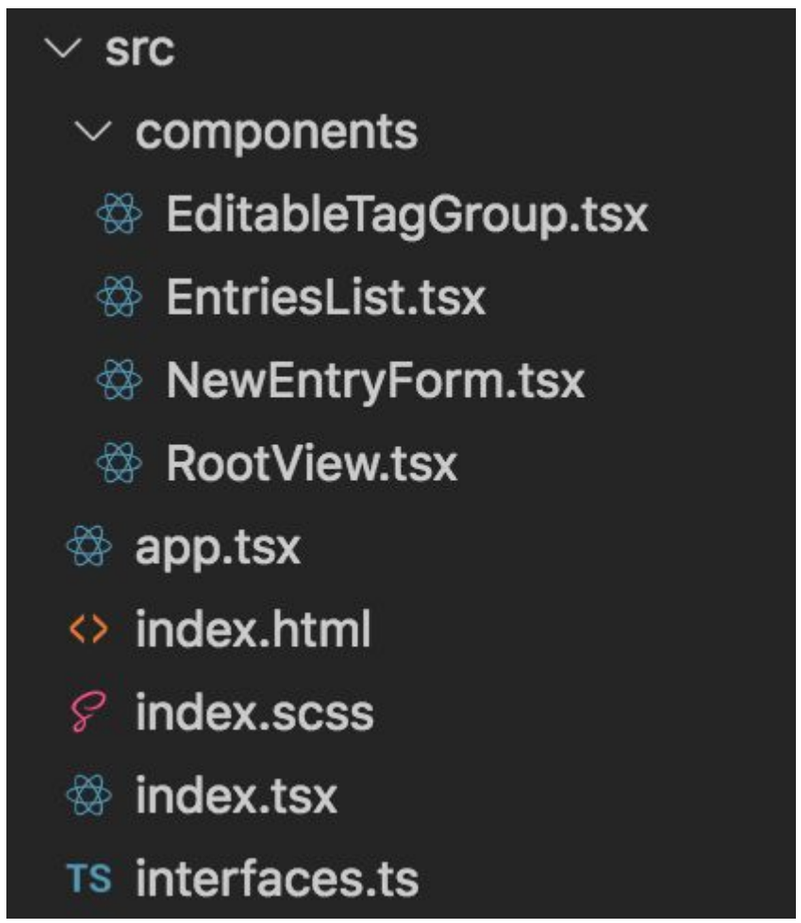
Struttura del progetto



- dist/: output di Webpack;
- src/: codice sorgente;
- webpack/: file di configurazione per Webpack;
- Vari file di configurazione che vedremo a breve.

Lorenzo Savini savo@develer.com

Struttura del codice sorgente



- components/: contiene i nostri componenti React;
- app.tsx: contiene il componente App;
- Vari entrypoint (index.html, index.scss, index.tsx);
- interfaces.ts: esporta la interfacce più comuni.

Lorenzo Savini savo@develer.com

Configurazione del progetto

I file di configurazione più importanti di un'interfaccia web in TypeScript/React/Webpack sono:

- package.json
- webpack.config.js
- tsconfig.json

Lorenzo Savini savo@develer.com

package.json

Il package.json è il file di configurazione del progetto.
Dà la possibilità di indicare le dipendenze della nostra applicazione e del relativo ambiente di sviluppo.
Oltre a questo, permette di dichiarare una serie di comandi (utilizzando la chiave “scripts”) che possono essere eseguiti tramite NPM o Yarn:

```
npm run-script build
yarn build
```

Lorenzo Savini savo@develer.com

webpack.config.js

Contiene tutta la configurazione relativa alla fase di build/bundling, incluse eventuali operazioni di **minifizzazione**, **transpilazione**, split del bundle in file multipli (classico approccio main vs vendor).

Diamo uno sguardo rapido al nostro config file.

develer

Lorenzo Savini savo@develer.com

tsconfig.json

Questo file di configurazione descrive le impostazione 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, qui lo schema completo:
  http://json.schemastore.org/tsconfig.

Lorenzo Savini savo@develer.com

Un sacco di boilerplate?

Il boilerplate risulta piuttosto standard per un'interfaccia web TypeScript/React/Webpack, ed ecco infatti che esistono tool per velocizzare il setup iniziale:

- **create-react-app**: tool ufficiale per creare SPA con React.
  https://create-react-app.dev
- **Yeoman**: esistono vari generator per lo stack TypeScript/React/Webpack.

Lorenzo Savini savo@develer.com

Exercise time!

Mettiamo in pratica quanto abbiamo visto finora.

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

develer

Lorenzo Savini savo@develer.com