Framework front-end: sviluppo, deploy e CLI

Angelo Di Iorio

Università di Bologna

CLI tools

- I framework includono un vasto ecosistema di **strumenti** per **supportare** e **semplificare** lo sviluppo
- Solitamente sono **pacchetti Node.js** installati e usati da **linea di comando**
- Permettono di creare facilmente applicazioni basate su quel framework:
  - Viene generata una struttura base dell'applicazione a cui aggiungere componenti
  - Vengono forniti comandi per **aggiungere**, **validare** e **testare** componenti
- Generare la struttura di base di un'applicazione vuol dire quindi:
  - Creare i **sorgenti**, sia HTML, CSS e JS che **template specifici** per quel framework
  - Installare strumenti che saranno poi usati per lo **sviluppo** e il **deploy dell'applicazione finale**

Dai sorgenti al browser (1)

Vue

```
<template>
  <div class="hello">
    <p>Ciao <b>{{nome}}</b>,
        benvenuto in Vue.js!</p></div>
</template>

...
```

...

React

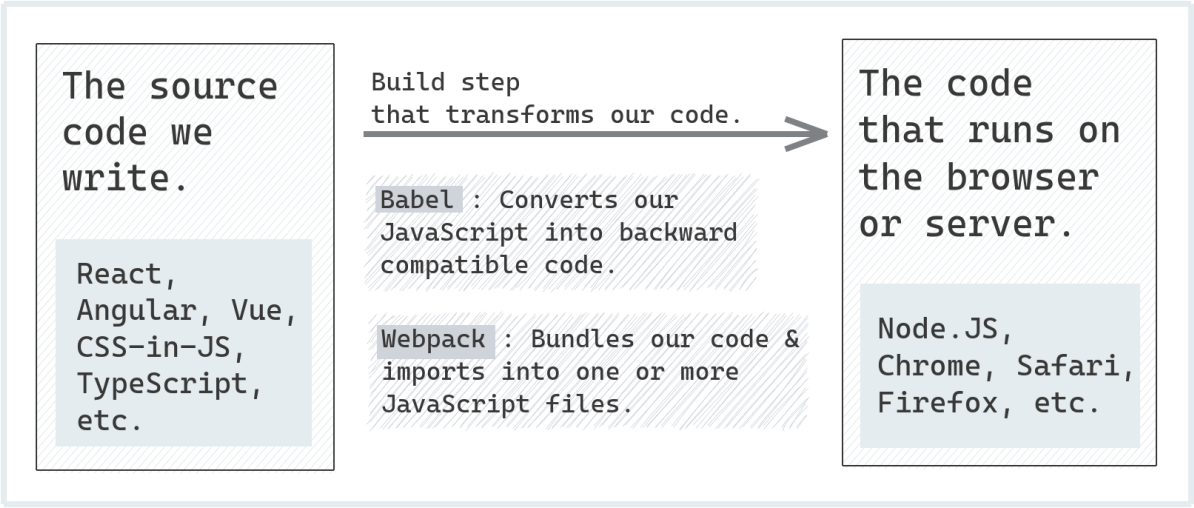
```
var Benvenuti = function() {
       return <h1>Benvenuti</h1>
       };

ReactDOM.render(
    <Benvenuti />,
 document.getElementById('welcome'));
```

Ciao **Angelo**, benvenuto in Vue.js!

Programmazione	Semestre: 1
Informatica	Semestre: 1
Tecnologie Web	Semestre: 2

Dai sorgenti al browser (2)

- I framework includono strumenti per tradurre il codice sorgente nell'applicazione eseguita sul browser
- Questi strumenti possono essere installati e usati anche separatamente
- I moduli CLI di ogni framework combinano e semplificano il loro uso, facendosi carico di installazione e configurazione



Dai sorgenti al browser: operazioni base

- Di seguito alcune delle operazioni tipiche e i pacchetti *Node* che le eseguono (non sono gli unici, altri svolgono gli stessi task)
- I tool CLI includono e invocano gli script inclusi in questi pacchetti

Operazione	Tool/Package
Linting	*eslint*
Compilazione, più precisamente transpilazione	*Babel, polyfill(s)*
Analisi e installazione delle dipendenze	*Webpack*
Creazione del bundle finale da eseguire sul browser	

- Oltre a questi strumenti, si usa spesso un server di test che aggiorna automaticamente l'applicazione dopo ogni modifica ai sorgenti, senza dover ripetere manualmente tutti i passaggi

Linting

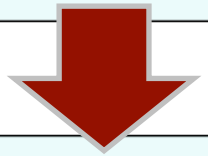
- L'operazione di "linting" consiste **nell'analizzare staticamente** il codice sorgente scritto in un certo linguaggio di programmazione alla ricerca di potenziali errori o imprecisioni (non nasce in ambito Web ma è usato comunemente)
- Diversi controlli:
  - Correttezza sintattica
  - Aspetti stilistici (indentazione, ritorno a capo)
  - Convenzioni (uso o divieto di alcuni costrutti)
- *eslint (the pluggable JavaScript linter)* è uno dei linter più usati in ambiente Web
- Il suo punto di forza è avere un **«core» minimale estensibile tramite plugin** e un vasto insieme di **regole configurabili** per la validazione
- Siccome il numero di regole è molto elevato, esistono delle configurazioni predefinite, chiamate "preset" per renderne più facile l'applicazione

Transpilazione e Babel

- Il termine **transpilazione** indica la traduzione da un linguaggio di programmazione ad un altro dello stesso livello di astrazione, a differenza della compilazione che traduce da un linguaggio di altro livello ad uno di livello più basso
- Babel è un *transpiler* JavaScript che converte il codice ES6+ in JavaScript compatibile con le versioni precedenti
- Permette di configurare i **browser *target*** e genera codice JS che può essere eseguito su quel/i browser
- Esegue sia trasformazioni generiche e indipendenti dal framework (es. ES6+) che specifiche (come React JSX o Vue Template) attraverso plugin
- Per ottenere questo risultato esegue principalmente due operazioni:
  1. Traduce da diverse sintassi (backward compatibility)
  2. Aggiunge polyfill

Babel: backward compatibility

```
const add = (x, y) => { return x + y };

const num1 = 1;
const num2 = 2;

let sum = add(num1, num2);

console.log(`Sum of ${num1} and ${num2} is ${sum}.`);
```



```
var add = function add(x, y) {
  return x + y;
};

var num1 = 1;
var num2 = 2;

var sum = add(num1, num2);

console.log("Sum of " + num1 + " and " + num2 + " is " + sum + ".");
```

Polyfill

- In programmazione Web, per "*polyfill*" si intende un frammento di codice che implementa funzionalità non disponibili in un browser, anche se ci si aspetta che siano implementate
- La necessità di un polyfill può essere dovuta a vari fattori:
  - funzionalità non implementate, ad esempio in versioni precedenti dei browser
  - implementazione parziale
  - bug critici
- Alcuni esempi:
  - Implementazioni di `JSON.stringify` o `JSON.parse` su browser che non supportano l'oggetto `JSON`
  - Simulazione di `WebWorker` su browser datati
  - Risoluzioni di bug su browser specifici
  - …
- Molti polyfill sono inclusi nel package *core-js*

Webpack

- Webpack (https://webpack.js.org/) è un **bundler** di moduli statici
- Si occupa di generare **asset statici** a partire da gruppi di file sorgenti organizzati in strutture complesse
- Costruisce il **grafo delle dipendenze** - a partire dagli import nei sorgenti e dalle dipendenze tra le librerie – e copia tutti i file necessari al funzionamento dell'applicazione
- I file JS prodotti possono essere caricati su un server e sono visualizzati/usati dal browser quando gli utenti accedono all'applicazione
- **Architettura modulare**, formata da **loaders** e **plugin** che si occupano di generare le singole parti del pacchetto finale (es. loader SASS, SVG, ecc.)
- Esistono diverse alternative a Webpack, ad esempio Browserify (https://browserify.org/)

Server di test

- Il pacchetto *webpack-dev-server* è incluso in Webpack e permette di lanciare l'applicazione su un server di test
- Ottimizzato per l'accesso veloce ai bundle prodotti da Webpack
- Include un modulo "*watchdog*" che controlla i file sorgenti e dopo ogni modifica ri-esegue la compilazione
- Configurabile attraverso l'oggetto principale di configurazione di Webpack (*webpack.config.js*)
- Anche in questo caso, esistono molti altri pacchetti che svolgono lo stesso task
- Vue, React e Angular usano *webpack* e *web-pack-dev-server*

Esempio: command-line Vue

CLI e sviluppo

- *Vue Command-Line-Interface* è un pacchetto Node.js (`npm install -g @vue/cli`) che include diversi strumenti da linea di commando per velocizzare lo sviluppo
- I comandi di base:
  - **`vue create hello-world`**: crea un nuovo progetto con tutte le dipendenze
  - **`npm run serve`** (esegue `vue serve`): fa partire un server di test che osserva le modifiche sui file e aggiorna direttamente la pagina
    - Molto utile quindi per sviluppare, non ottimizzato per fare il deploy in produzione
- Guida completa su: https://cli.vuejs.org/guide/

CLI e deploy

- Oltre a lanciare il server di test un'applicazione Vue include altri due script:
  - **`npm run lint`** (esegue `vue lint`): controlla la correttezza sintattica e strutturale dei template
  - **`npm run build`** (esegue `vue build`): produce una versione del progetto completa di tutte le librerie e pronta per essere spostata su un server
    - output nella directory `./dist/`
    - ATTENZIONE: NON funziona se si apre il file `index.html` da filesystem. Necessario un server HTTP, ad esempio il pacchetto `serve` di Node.js