ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Strumenti per il "modern" web development

Vincenzo Rubano

Alma Mater – Università di Bologna

Ambienti di sviluppo

Per ogni applicazione, distinguiamo almeno due ambienti di esecuzione:

- Ambiente di sviluppo (o locale), ovvero la macchina su cui viene implementata;
- Produzione, ovvero l'ambiente in cui l'applicazione viene eseguita quando vi accede l'utente finale.
- In alcuni casi può essere presente il cosiddetto ambiente di "staging", in cui le nuove funzionalità vengono testate prima del "deploy"

Di cosa si tratta?

Script, programmi e/o librerie da utilizzare mediante un terminale (appunto “a riga di comando”) o mediante un'IDE con uno scopo preciso

Piattaforme e servizi cloud per sviluppatori

Pensati per migliorare la produttività e risolvere problemi

Come si usano?

Nella maggior parte dei casi, vengono eseguiti nell'ambiente di sviluppo sul codice sorgente dell'applicazione; nell'ambiente di produzione, invece, viene caricato il codice risultante dall'esecuzione di questi strumenti.

Questa è una regola generale, ma ovviamente vi sono eccezioni!

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Sopravvivere all'evoluzione

Lo sviluppo front-end si evolve

Al contrario di quanto si possa pensare, i linguaggi che utilizziamo quotidianamente per sviluppare applicazioni Web (soprattutto JavaScript e CSS, ma anche HTML) sono in continua evoluzione secondo particolari meccanismi ben definiti.

Le novità introdotte sono spesso “desiderabili” durante lo sviluppo, ma purtroppo non è detto che esse siano supportate negli ambienti di esecuzione (i browser, Node.JS, etc) in cui l’applicazione debba essere eseguita.

Ciò può avvenire sia perché le novità non sono ancora state implementate, sia perché è necessario supportare versioni precedenti degli ambienti di esecuzione rispetto a quelle che ne integrano il supporto; come fare, allora?

CanIUse

Il sito CanIUse (www.caniuse.com) è una fonte di informazioni preziosissima!

Esso, infatti, indica a partire da quale versione di ogni browser una certa funzionalità (sia essa di HTML, JavaScript o CSS) è supportata, nonché una stima del “market share” di ciascun browser.

I dati sono messi a disposizione anche in modo strutturato, tramite delle comode API!

CSS DB

- Contiene una lista delle feature relative ai fogli di stile CSS, ed il loro grado di “maturità” nel processo di integrazione nello standard vero e proprio (che è diviso in più fasi).

Browserslist

- Descritto come “The config to share target browsers and Node.js versions between different front-end tools”, si tratta di uno strumento “molto potente” per descrivere il target (browser e version di node) che si intendono supportare.
- Queste informazioni vengono utilizzate da altri strumenti per determinare se una certa “caratteristica” del codice sorgente è supportata da tutti i browser (sfruttando caniuse e cssdb), determinare il grado di support (complete, parziale, con bug più o meno seri, etc), e decidere se e come trasformare il codice, o usare dei polyfill.

I polyfill

- In programmazione Web, per “polyfill” si intende un frammento di codice che implementa funzionalità che non fanno parte di quelle offerte da un browser, aggiungendo caratteristiche che ci si aspetta siano disponibili ma non lo sono. La necessità di un polyfill può essere dovuta a vari fattori, tra cui la totale assenza dell’implementazione originale, il fatto che essa contenga bug critici o sia incompleta.

- Molti polyfill utilizzati in JavaScript sono forniti dalla libreria “core-js”.

Problemi risolti

Esistono strumenti per risolvere i problemi più disparati, ma oggi ci occuperemo di:

- gestione delle dipendenze;
- CSS;
- Type checking;
- Linting;
- Transpilation;
- Minification;
- Bundling

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Gestione delle dipendenze

Cos’è la gestione delle dipendenze

Nel mondo della gestione del software esiste un luogo terrificante chiamato “dependency hell (inferno delle dipendenze)”. Quanto più cresce il vostro sistema e quanti più pacchetti integrate nel vostro software, tanto più facilmente vi ritroverete, un giorno, in questa valle di lacrime.

[Cit.: Versionamento Semantico 2.0.0 | Semantic Versioning (semver.org)]

Strumenti per la gestione delle dipendenze

Si tratta di strumenti che facilitano la gestione di tutti i pacchetti necessari allo sviluppo e/o all'esecuzione di un'applicazione Web. Il loro funzionamento è solitamente molto simile: ci limiteremo ad esaminare npm e yarn.

Automatizzano la gestione di molti aspetti problematici, come la selezione di versioni compatibili delle dipendenze.

DevDependencies VS. dependencies

- ***DevDependencies***: sono necessarie per lo sviluppo dell'applicazione, ma non per la sua esecuzione nell'ambiente di produzione. Esempio: vue-cli.
- ***Dependencies***: sono indispensabili per l'esecuzione dell'applicazione in ambiente di produzione, e ne consentono il funzionamento. Esempio: vue.

- NB: quando si aggiunge una dipendenza, di fatto si aggiungono al Progetto anche tutte le sue dipendenze (chiamate "transitive-dependencies"). Tutte le dipendenze di un Progetto formano il cosiddetto "grafo delle dipendenze".

NPM e Yarn

Sono due gestori di pacchetti molto simili pensati per gestire le dipendenze in ambiente node.js, ma consentono anche la creazione di comandi personalizzati per l'automazione di task ripetitivi.

Si basano su due file:

- package.json, che contiene i metadati del progetto, tutte le dipendenze (dev e non) ed eventualmente una specifica delle versioni accettabili;
- Un file "lock" (package-lock.json e yarn.lock, rispettivamente), che contiene l'elenco delle dipendenze e per ciascuna di esse quale versione è stata installata in un certo istante.

1, 2, 3… yarn

- Esistono ben 3 versioni di yarn attualmente in circolazione
- A partire dalla versione 2 è stato riscritto, e la sua implementazione è cambiata radicalmente (supporto per plugin, possibilità di occupare meno spazio su disco nella gestione di node_modules, etc.)
- Yarn 1 e yarn 2 sono molto diversi, yarn 2 e yarn 3 sono molto simili
- Se dovete scegliere: yarn 1 (sconsigliato), o yarn 3

Composer

- Simile a NPM e yarn, ma per applicazioni scritte in php
- Esso stesso scritto in php
- Offre funzionalità specifiche per questo linguaggio (generazione di classmap,, autoloading, etc.).

Perché il lock?

Un file “lock” è necessario per rendere l’installazione “riproducibile”, ovvero consentire ad un altro utente di installare le stesse identiche dipendenze con cui è stata testata l’applicazione su un’altra macchina. Le versioni accettabili, infatti, possono essere specificate sotto forma di range per agevolare gli aggiornamenti e l’installazione di versioni “compatibili”.

Alcuni comandi

- Inizializzare un progetto:
  - Npm init
  - Yarn init
- Aggiungere una dipendenza:
  - Npm install react
  - Yarn add react
- Aggiungere una dipendenza dev:
  - Npm install react –include=dev
  - Yarn add --dev react
- Installare tutte le dipendenze di un progetto:
  - Npm install
  - Yarn install
- Aggiornare tutte le dipendenze del progetto:
  - Npm update
  - Yarn upgrade

Il problema delle dipendenze

Dato un progetto con n dipendenze, vogliamo aggiungere una nuova dipendenza in modo tale che:

1. essa (e tutte le sue eventuali dipendenze) siano compatibili con le dipendenze già necessarie;
2. Sia possibile aggiornare tutte le dipendenze alle ultime versioni disponibili, garantendo però la loro compatibilità.

In ambito node.js, sfruttiamo il semantic versioning e la specifica dei “range accettabili”, ovvero indicare quali versioni di una dipendenza sono accettabili nel progetto all’istante attuale e “nel futuro”.

Semantic versioning

Quando si pubblica un pacchetto, di solito si specificano le versioni nel formato x.y.z dove:

- Un incremento nel valore di x (detto “major”) indica che la versione introduce cambiamenti nelle API pubbliche non retrocompatibili con le versioni precedenti;
- Un incremento nel valore di y (detto “minor”) indica che la nuova versione introduce cambiamenti nelle API retrocompatibili con le version precedent;
- Un cambiamento nel valore di z (detto “patch”) indica che la nuova ersione si limita a risolvere bug.

Esempi di range

In package.json:

“mylib”: “1.0.0”

Indica che può essere accettata solo la versione 1.0.0

“mylib”: “1.0.x”

Indica che possono essere accettate tutte le versioni “patch” (i.e. >=1.0||<1.1).

“mylib”: 1.x

Indica che possono essere accettate tutte le versioni patch e minor (i.e. >=1.0||<=2.0).

NB: || rappresenta l’operatore or, e può essere usato per esprimere range.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CSS

CSS Preprocessors

Strumenti che consentono di convertire codice scritto in un apposito linguaggio di scripting in fogli di stile CSS veri e propri. Creare un linguaggio di scripting consente di "estendere" le funzionalità rese disponibili da CSS mettendo a disposizione concetti quali gli operatori logici, le variabili, i mixins, l'ereditarietà, le funzioni e gli operatori matematici.

I più comuni sono: SASS, LESS e stylus.

SASS

SASS (Syntactically Awesome Style Sheet) è il primo CSS preprocessor, è stato rilasciato nel 2006; inizialmente richiedeva l'utilizzo di Ruby, ma nel tempo è stata implementata la libreria "libsas" che, essendo scritta in C, consente di interpretare questo linguaggio praticamente in ogni ambiente (node.js, php, etc).

I file hanno due possibili estensioni: .sas, che utilizza la sintassi originale (basata su indentazione), e ".scss", che utilizzano una sintassi CSS-like ma arricchita: ogni file .css è automaticamente un file .scss, ma non un file .sas.

LESS

LESS (Leaner Style Sheets) è stato rilasciato nel 2009. Implementa molti dei concetti presenti all'interno di SASS, e nel corso del tempo i due tool si sono influenzati a vicenda (la seconda sintassi di SASS è molto simile alla sintassi di LESS, per esempio).

Viene fornito sotto forma di libreria JavaScript eseguibile in ambiente Node, ma è disponibile anche una versione eseguibile direttamente nel browser.

PostCSS

Descritto sul sito ufficiale come “A tool for transforming CSS with JavaScript”, PostCSS permette di trasformare i fogli di stile in ambiente Node eseguendo diverse operazioni:

- Introdurre automaticamente i “prefissi” laddove necessari (autoprefixer), per abilitare attributi sperimentali;
- Utilizzare sintassi CSS non ancora pienamente supportata da tutti i browser (postcss-preset-env), trasformando i costrutti in loro equivalenti che lo siano;
- Eseguire automaticamente il “linting” dei file CSS (stylelint), per verificare il rispetto di regole stabilite (dallo sviluppatore del Progetto che lo usa);
- Usare i cosiddetti “CSS modules”, per avere uno “scoping locale” per i selettori (e non globale, come avviene di default);
- e molto altro: può essere esteso tramite plugins, e ne esistono di ogni tipo: conversion di unità, embedding di grafica, rimozione di CSS non necessario dato un certo markup, etc.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Linting

Cos’è il linting?

Con “linting” si intende il processo di analizzare staticamente il codice sorgente scritto in un certo linguaggio alla ricerca di potenziali “errori”. In particolare, nella programmazione qweb ciò spesso si traduce in un’analisi per individuare violazioni di regole stabilite dallo stesso programmatore del progetto.

Tali regole possono riguardare aspetti stilistici (indentazione, ritorni a capo, etc), così come l’uso (o il divieto di usare) certi o costrutti.

eslint

Eslint (the pluggable JavaScript linter) è uno dei linter più usati in ambiente Web. Il suo punto di forza sta nell'avere un "core" minimale, e poter essere esteso mediante plugin.

Il core ed i vari plugin forniscono un elevatissimo numero di regole, lasciando al programmatore la facoltà di decidere (attraverso un file di configurazione) quali utilizzare e quali no. Siccome il numero di regole è molto elevato (nell'ordinedi centinaia), esistono dei "preset" per renderne più facile l'applicazione.

Eslint – categorie di regole

In eslint, le regole sono raggruppate per tipologia:

- Possibili errori, ovvero regole che evitano di commettere errori logici e/o di sintassi;
- Best practice, relative ai migliori modi per implementare determinate funzionalità in modo da evitare problemi;
- Regole relative alle dichiarazioni di variabili;
- Regole stilistiche, relative allo stile di programmazione ed alla formattazione del codice; possono essere soggettive.
- Es6, regole relative all'uso di funzionalità introdotte in EcmaScript 6.

eslint --fix

Un altro punto di forza di eslint è quello di poter risolvere “automaticamente” molti dei problemi individuati, grazie al meccanismo denominato “autofix”. Tale meccanismo deve essere abilitato esplicitamente (nel file di configurazione o a riga di comando), giacchè potrebbe creare problemi con alcuni editor.

Prettier

Prettier (“the opinionated code formatter”) è uno strumento specializzato nell’eseguire il parsing del codice sorgente, e stamparlo applicando regole di formattazione ben precise. Supporta molteplici sintassi (typescript, JavaScript, JSX, JSON).

Al contrario di quanto avviene con eslint, le sue regole si basano su “opinioni” ben precise sul modo con cui dovrebbe essere scritto il codice, ed offre meno flessibilità a livello di configurazione.

Un esempio

```
foo(reallyLongArg(), omgSoManyParameters(),
IShouldRefactorThis(), isThereSeriouslyAnotherOne());
```

Sebbene questo codice sia perfettamente valido sintatticamente, la sua leggibilità è quantomeno discutibile…
Può essere riformattato come:

```
foo(
  reallyLongArg(),
  omgSoManyParameters(),
  IShouldRefactorThis(),
  isThereSeriouslyAnotherOne()
);
```

Ben più leggibile!

Perché tanta importanza allo stile?

Avere uno stile di programmazione “coerente” rende più agevole la lettura del codice. Ogni programmatore, però, tende ad applicare le regole che più preferisce, sempre che si preoccupi di questi aspetti. Ecco perché questi tool sono importanti!

Essi possono essere integrati in vari IDE, utilizzati a riga di comando, o addirittura diventare parte delle pipeline della cosiddetta “continuous integration (Ci)”, molto comune nei progetti open source.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Type checking

Esempio I

```
const el = document.getElementById('my-id')
el.classList.add('foo')
```

Il codice non è sempre corretto: se l'elemento con my-id non esiste, infatti, viene sollevata un'eccezione e l'esecuzione viene interrotta: qualunque istruzione successiva nello stesso blocco non sarà valutata, e senza una corretta gestione delle eccezioni potrebbe essere sospesa l'esecuzione dell'intero script.

Esempio II

```
function add(a, b) {
  return a+b
}

add(3, 4)
add(3,'4')
```

Cosa viene restituito?

I tipi in JavaScript

Per sua natura, il linguaggio JavaScript presenta un sistema di tipi dinamico, in cui cioè la tipizzazione viene eseguita durante l'esecuzione del codice. Questo rende difficile il debugging, giacchè non è possibile eseguire un'analisi statica dei tipi… o per lo meno essa è resa talmente problematica da non essere praticabile in tempi brevi. Vi sono design pattern, inoltre, in cui essa è di fatto impossibile.

Se su progetti di piccola scala questo aspetto potrebbe non essere problematico, con l'aumento delle righe di codice il problema diventa sempre più pressante; è molto facile, inoltre, far sì che il codice manifesti comportamenti diversi a causa dei parametri utilizzati per invocare le funzioni.

Tutto questo, però, può essere valutato solo a runtime… Eseguendo il codice in ogni possibile scenario, seguendo tutte le sue possibili ramificazioni… E presto diventa non sostenibile!

Typescript

Si tratta di un linguaggio open source sviluppato da Microsoft, il cui scopo è quello di rendere più agevole l'analisi statica dei tipi in JavaScript. Il linguaggio è definito come un super-set di JavaScript, perciò ogni costrutto valido in JavaScript è valido in typescript; vengono però aggiunti costrutti per l'annotazione dei tipi, nonché un sistema di type-inference che consente di conoscere (staticamente) il tipo di una variabile.

Un compilatore si occupa di "tradurre" i sorgenti typescript in JavaScript vero e proprio, eseguendo nel contempo l'analisi statica dei tipi: ciò consente di evitare molti bug.

Esempio I – in typescript

```
const el = document.getElementById('my-id')
el.classList.add('foo')
```

// Errore: el è di tipo HTMLElement|NULL, ed il tipo “HTMLElement|NULL” non ha la proprietà “classList”.

Soluzione: usare il “null coalescing operator”, oppure controllare esplicitamente il valore restituito dalla funzione.

Esempio II – in typescript

```
Function add(a: int, b: int): int {
  return a+b
}

add(3, 4) // corretto
add(3, '4') // errore: il secondo parametro è una
stringa, non un intero
```

Si può anche tipizzare la funzione in modo da mantenere la “dinamicità” sfruttando la composizione dei tipi…

Tipi in typescript I

Typescript fornisce alcuni tipi primitivi, ma è possibile derivare nuovi tipi per composizione.

Consente inoltre di definire i tipi corretti per array, dizionari, insiemi e tutte le altre “collections”, i “literal objects”, le classi, e permette di definire interfacce e dichiararne la conformità da parte di classi e oggetti… E molto altro!

Tipi in typescript II

Le definizioni per molti tipi (API relative al DOM, per esempio) sono “integrate” nel compilatore; è inoltre possibile utilizzare i tipi anche per le librerie di terze parti.

NB: a volte è necessario installare i tipi di una libreria separatamente; ciò è dovuto al modo con cui essa è implementata. Se una libreria non fornisce nativamente i suoi tipi, è possibile che essi siano resi disponibili da sviluppatori di terze parti.

Se una libreria non contiene le sue definizioni di tipo, è molto probabile che esse siano disponibili in package il cui nome è della forma “@types/libraryname”.

Analisi statica per altri linguaggi

- Strumenti simili sono disponibili anche per moltissimi altri linguaggi
- Spesso forniscono funzioni utili specificamente per quei linguaggi
- La quantità di problemi “individuabili” staticamente da questi strumenti può dipendere dalle proprietà del linguaggio

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Transpilation

Cos’è la transpilation?

Per transpilation si intende quel processo di conversione attraverso un programma (detto transpiler) di un codice sorgente scritto in un certo linguaggio in una sua forma equivalente scritta in un altro linguaggio. A differenza di un compilatore, il transpiler opera con due linguaggi il cui livello di atrazione è molto simile.

Babel I

Si tratta di un transpiler (a volte detto anche compilatore) per il linguaggio JavaScript, il cui scopo primario è quello di consentire l'utilizzo delle funzionalità più moderne introdotte nel linguaggio indipendentemente dal supporto offerto tramite i browser. Ciò viene ottenuto con diverse tecniche:

- Trasformando la sintassi dei costrutti;
- Iniettando i polyfill necessari.

Babel II

Esso è uno strumento molto potente, ma per sfruttarlo al massimo richiede una configurazione opportuna; può svolgere al meglio il suo compito in combinazione con altri tool (core-js, browserlist, etc).

Esistono plugin che offrono il supporto per i principali framework: react (responsabile della traduzione della sintassi JSX in JavaScript, per esempio), Vue (per trasformare i single file template ed altri costrutti), etc.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Minification

Cos’è la minification?

In programmazione, per “minification” si intende il processo di rimozione di tutti i caratteri non necessari dal codice sorgente di uno script scritto in un linguaggio interpretato, o da un contenuto scritto in un linguaggio di markup, senza alterarne le funzionalità. (o la semantica).

Lo scopo è quello di “minimizzare” la dimensione del codice sorgente, così da velocizzarne la trasmissione tramite una rete (come per esempio Internet); di solito si rimuovono i caratteri non necessari spazi bianchi, ritorni a capo, etc), ma possono essere anche eseguite “trasformazioni” che convertano determinati costrutti in forme equivalenti “più brevi”.

Minification vs. compression

La minification è diversa dalla compressione, in quanto il codice sorgente “minificato” può essere interpretato esattamente come la sua forma “non minimizzata”, senza cioè richiedere step aggiuntivi (come una fase di decompressione).

minifier

Esistono diversi strumenti più o meno specializzati nella minification di sorgenti scritti in vari linguaggi (HTML, JavaScript, CSS). Vediamo all'opera terser, uno dei minifier più utilizzati per ridurre le dimensioni del codice JavaScript.

Esempio di minification I

Consideriamo questo semplice frammento di codice JavaScript:

```
/**
 * Adds the given class to each DOM node in the given list
 *
 * @param elements: DOM nodes to add the class to
 * @param classname: CSS class to be added
 */
function addClass(elements, classname) {
  elements.forEach((element) => {
    element.classList.add(classname)
  })
}

const paragraphClass = 'para'
const paragraphs = document.querySelectorAll('p')
addClass(paragraphs, paragraphClass)
```

Esempio di minification II

Minificato mediante terser quel frammento può essere convertito in:

```
document.querySelectorAll("p").forEach((a=>{a.cla
ssList.add("para")}));
```

Esempio di minification III

Le due forme in cui è scritto il codice (non-minificata e minificata), sono del tutto equivalenti; mentre la prima occupa 415 bytes, la seconda ne occupa “solo” 71, che è inferiore circa dell’83%!

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Bundling

Cos’è il bundling?

Quando si sviluppa un’applicazione Web, per favorire la manutenibilità del codice è prassi comune “suddividere” il codice in più file.

Quando l’applicazione deve essere eseguita, però, avere molti file di piccole dimensioni può avere un impatto negativo sulle performance: il bundling, dunque, è quel processo che consiste nell’unificare più file dello stesso tipo (CSS, JS, etc) in un unico file secondo criteri.

Tale “raggruppamento” può avvenire in base a diversi criteri.

Il bundling non è un problema banale

Dal punto di vista implementativo, il bundling non è un problema affatto banale: specialmente quando si parla di JavaScript, infatti, occorre considerare diversi aspetti quali l'ordine di esecuzione (ed i side-effects), i potenziali conflitti di nomi (classi, variabili, metodi, etc), le dimensioni dell'output.

Esistono ovviamente diverse tecniche per affrontare questi problemi…

I bundler

Esistono molteplici bundler in circolazione.

Sebbene molti di essi condividano un certo insieme di funzionalità, ognuno può presentare peculiarità più o meno significative…

Ci limiteremo ad esaminarne n paio: webpack e rollup.

webpack

Bundler molto potente e flessibile, la cui configurazione può essere semplici o (molto) complessa a seconda dei casi d'uso. In generale, crea un "grafo delle dipendenze" del sorgente del progetto e, dopo aver eventualmente trasformato il codice (aggiornando il grafo opportunamente), procede alla creazione di uno o più artefatti secondo parametri configurabili.

Nonostante la potenziale complessità, è un bundler molto maturo, dotato di un ecosistema di plugin che lo rende molto interessante.

rollup

Nel corso del tempo, rollup si è fatto conoscere come un'alternativa a webpack più efficiente, più semplice da configurare, più performante (il bundling richiede tempo!), e per la generazione di artefatti più efficienti e di dimensioni inferiori, grazie all'uso di tecniche innovative (e.g. scope oisting).

Sebbene anche l'ecosistema intorno a rollup si stia evolvendo velocemente, spesso non è tanto maturo quanto webpack, perciò ci si potrebbe imbattere in vari bug da "aggirare" in modi non sempre banali, in attesa che vengano risolti.

Al contrario di webpack, inoltre, il suo core è estremamente minimale: ciò è sia un vantaggio che uno svantaggio.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Putting it all together…

Una pipeline

Gli strumenti a riga di comando disponibili sono molteplici, e spesso complementari.

Realisticamente, a parte doverose eccezioni, è difficile che essi vengano utilizzati “da soli”; più probabile, invece, è che vengano organizzati in una “build pipeline”, in modo tale che il codice sorgente venga controllato, “compilato” e “assemblato” secondo una certa configurazione.

Esempio di pipeline

Esaminiamo una possibile pipeline per file JavaScript. Ogni passaggio può generare sia errori che warning, i quali possono interrompere il processo.

- Partiamo dal file principale;
- Analizziamo le sue dipendenze (ricorsivamente);
- Processiamo i file, eseguendo linting e/o type-checking;
- Eseguiamo il transpiling di ogni codice (traducendo costrutti, iniettando polyfill in base ai target supportati, etc);
- Creiamo il “bundle” risultante, da eseguire sul browser.

Configurazione della pipeline

Queste pipeline possono diventare piuttosto complesse; di solito, esse vengono gestite attraverso un insieme di configurazioni per ottenere un risultato organico.

Spesso, inoltre, vi è un “overlapping” tra vari tool che offrono la stessa funzionalità (esliint e prettier, eslint e typescript, per esempio); è compito dello sviluppatore trovare la configurazione ideale, e non è sempre banale. Perciò…

Task management

Le tipologie di strumenti che abbiamo analizzato sono solo una parte di quelle disponibili, ma già fanno emergere l'esigenza di eseguire (e coordinare l'esecuzione) di vari compiti: come fare?

Come detto, spesso si lascia al bundler il compito di coordinare vari aspetti; esistono però altre possibilità, e non sono mutualmente esclusive:

- Usare i cosiddetti "npm scripts", che consentono di eseguire comandi complessi definiti all'interno del file "package.json" con alias più semplici;
- Usare i cosiddetti task-runner, come grunt e gulp.

Grunt VS. Gulp

- Si tratta di due task runner scritti in JavaScript per ambiente Node.JS
- Hanno funzionalità simili, ma implementazioni diverse
- Esemplificando e semplificando,
  - “grunt” per ogni passaggio prende in input un file e restituisce un file,
  - “gulp” si basa su uno stream con un solo input ed un solo output;
- Grunt si configura in maniera “statica”, in “gulp” si programmano i task da eseguire
- Generalmente, Gulp ha prestazioni migliori!
- Grunt sfrutta un approccio più “tradizionale”

Non bastano NPM scripts?

- Nella maggior parte dei casi, sì!
- Ma a volte ci sono esigenze particolari, per cui NPM scripts sarebbero troppo complessi, difficilmente manutenibili, e a volte impossibili da realizzare.

*-cli

Quello che spesso accade, dunque, è che i principali framework più usati per lo sviluppo Web mettano anche a disposizione delle pipeline già preconfigurate, offrendo un processo “ottimizzato” per quel determinato framework; è il caso di angular-cli, react-cli, vue-cli, per esempio.

Sarebbe importante capire il funzionamento di questi strumenti, però, così da evitare di vedere i command line tools come una “blackbox”: di fatto tutto è configurabile, e non è dettoche le configurazioni fornite dai framework soddisfino sempre le nostre esigenze o siano prive di bug.

E’ quindi importante sapere come/dove mettere le mani per risolvere!

Le source maps

Quando il codice viene trasformato, fare il debugging diventa complicato: come si determina a quale frammento del sorgente originale corrisponde il frammento di codice in esecuzione?

Si usano le cosiddette “source map”, file che consentono di mappare frammenti del codice trasformato (detto “generated code”) ai frammenti del codice sorgente (detto “original code”) che li hanno generati. Nel file generato, di solito, si inserisce l’URL della source map.

NB: generare la source-map durante un processo di compilazione può essere impegnativo da un punto di vista computazionale.

Prestazioni della pipeline

- Per quanto possano essere ottimizzati, questi strumenti richiedono un certo tempo per compiere il loro compito
- Questo tempo aumenta “più che linearmente” al crescere delle dimensioni del progetto (numero e lunghezza dei file su cui lavorare), nonché del numero di operazioni da compiere e la loro complessità;

esbuild

- Per questo è stato creato esbuild, un “bundler” nell’ordine di 10, 100 volte più veloce nell’eseguire i compiti;
- Scritto in Go
- Integra supporto per typescript, minification, JSX e generazione SourceMap, costituendo una pipeline integrata
- Predisposto per plugin esterni
- Stabile e funzionante, ma non ancora “maturo” quanto i suoi concorrenti.

Vite

- Innovativo sotto vari punti di vista
- Unisce le prestazioni di esbuild con i vantagi di rollup, traendo di fatto il meglio dai due
- Integra molte funzionalità pronte all'uso (CSS, JSX, transpilation, typescript, etc.)
- Più che un bundler è una pipeline complessa pronta all'uso, ma comunque estensibile e personalizzabile

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Static site generators (SSG)

Cosa sono gli SSG?

- Se una pipeline è accettabile per “compilare” un progetto JavaScript, perché non creare strumenti per “compilare” interi siti?
- Si tratta di strumenti che prendono in input una serie di file (di tipo e formato diversi), applicano dei template (con sintassi e modi diversi), e generano pagine in HTML (statico);
- Ne esistono tanti (troppi?), tutti simili ma diversi;
- Vengono chiamati “Static Site Generator (SSG)”

Hugo

- “The world's fastest framework for building websites”
- Prestazioni estremamente elevate (anche con siti da decine di migliaia di pagine)
- I template sono scritti sfruttando la sintassi di “Go Template”
- Le pagine possono essere generate da file markdown, JSON, YAML, TOML e XML (locali e remote);
- Le sue funzionalità possono essere estese mediante moduli scritti in Go

Eleventy

- Scritto in JavaScript, e basato su Node
- Estremamente (troppo?) modulare;
- Elevato numero di plugin disponibili;
- Facilmente estensibile mediante JavaScript;
- Supporta molte sintassi per la scrittura dei template;

Framework ibridi

Esistono anche approcci ibridi, che consentono di combinare la generazione di pagine statiche con il server-side-rendering

- Next, basato su React
- Nuxt, basato su Vue
- Remix
- Astro

Perché SSG?

- Offrono molti vantaggi rispetto ad altri approcci
- Prestazioni elevatissime;
- Facilità di hosting (l'output è HTML statico, torniamo alle origini del Web!);
- Scalabilità;
- Sicurezza (nessun backend, nessun interprete server-side, nessun DB coinvolto).

Perché non usare SSG?

- Gli stessi vantaggi offerti diventano i loro limiti principali
- Non sono adatti per siti di grandi dimensioni (centinaia di migliaia o milioni di pagine);
- Non si prestano particolarmente bene a scenari in cui bisogna gestire contenuti prevalentemente dinamici (prodotti dagli utenti, per esempio);
- Non avere un backend implica dipendere da servizi forniti da terzi (email, commenti, storage, etc.), e ciò può essere un problema da vari punti di vista (mancanza di controllo, GDPR, etc.).

Headless websites, decoupled architectures

- Trend del momento
- L'idea di base è quella di avere un CMS responsabile per la creazione di contenuti (backend), che permetta poi di "consumarli" in modi "machine-friendly" (di solito con API REST, serializzati in JSON);
- Si costruisce poi un'applicazione (o sito statico!) basato su tali contenuti, che non dipenda dal backend in alcun modo (eccetto le API pubbliche);
- Esistono moltissimi strumenti (cloud e self-hosted) per implementare tale architettura, non li elencheremo;
- Esistono svariati "standard" creati per supportare questo tipo di architetture: GraphQL e JSON:API ne sono due esempi

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Testing

Cosa è il testing?

- L’informatica è una scienza ricorsiva: perché non scrivere codice per testare la correttezza del nostro codice?
- E’ un tema complesso ma essenziale: il codice delle nostre app diventa sempre più articolato e complesso, testarne la correttezza consente di evitare “regressioni” (introduzione di bug a seguito di modifiche del codice) e di assicurarne la correttezza.
- Scrivere dei buoni test non è semplice, ma è un ottimo esercizio per scrivere codice migliore.

Given this then that

- Filosofia per scrivere i test della nostra app
- given this: si “innesca” un comportamento (chiamando metodi, istanziando oggetti, etc.);
- then that: si verifica che determinate che ci si aspetta siano vere lo siano davvero (mediante “assertions”).
- Attenzione: vogliamo testare il “contratto” (ovvero che sia vero ciò che il nostro codice promette), non una specifica implementazione!

Tipi di test e strumenti di supporto

- Esistono diversi tipi di test (unit test, integration tests, UI tests, etc.), a seconda di ciò che si vuole testare
- Esistono molteplici framework che semplificano la scrittura di test (moca, jest, nightwatch, PHPUnit, e moltissimi altri);
- Alcuni di questi consentono di testare comportamenti che si verificano soltanto in un browser (ad esempio perché utilizzano il DOM).

Esempio I

```
// Pseudo-code
function fact(n) {
  if (n === 0) return 1
  else return n*fact(n-1)
}

class FactorialTests {
  function testPositiveNumbersFactorial() {
    const f = fact(6)
    assert(f==720, “Factorial of 6 should be 720”)
    const f2 = fact(5)
    assert(f2==120, “Factorial of 5 should be 120”)
  }
…}
```

Esempio II

```
// Pseudo-code
class FactorialTests {…
  function test0Factorial() {
    const f = fact(0)
    assert(f==1, “Factorial of 0 should be 1”)
  }

  function testNegativeFactorial() {
    const f = fact(-3)
    // Crash! Svela un bug
  }
}
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Version control

Perché version control?

- Consente di tracciare e controllare ogni modifica al codice
- Può essere usato anche per documentazione, file di configurazione, ed ogni altro tipo di dato (meglio se testuale);
- Per ogni cambiamento, consente di conoscerne l'autore, la data in cui è stato effettuato, e le modifiche apportate.
- Facilita (di molto) la revision delle modifiche apportate.

Git

- Sicuramente il sistema di version control più usato
- Disponibile come strumento a riga di comando per linux, Mac e Windows, ma anche integrabile in vari IDE;
- Ogni progetto corrisponde ad una “repository”;
- Ogni repository può contenere più “branch” (rami) differenti del codice, di cui uno è il principale (master/main);
- Ogni branch può contenere differenze rispetto alle altre; possono essere sviluppate in parallelo
- I cambiamenti avvengono per mezzo di “commit” in locale;
- Si aggiungono i propri commit alla repository mediante “push”
- Si “aggiorna” la repository mediante “pull”.

Git e il cloud

- Intorno a git sono state sviluppate diverse piattaforme cloud che ne agevolano l'uso
- Github, GitLab, e BitBucket sono solo alcuni esempi
- Queste piattaforme forniscono funzionalità aggiuntive (wiki, issue tracking, etc.) e servizi (contiuous integration, notifiche, etc.) che "potenziano" le funzionalità di git, pur mantenendone i concetti chiave
- Sono anche ottimi sistemi di backup per il codice!
- Facilitano la collaborazione tra più sviluppatori ad uno stesso progetto

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Continuous integration (CI)

Cos’è la continuos integration?

- Semplificando, è quell’insieme di pratiche che consentono l'allineamento frequente (anche molte volte al giorno) dagli ambienti di lavoro degli sviluppatori verso l'ambiente condiviso (mainline).
- E’ possibile automatizzare (molte!) verifiche sulle modifiche (tutti i test devono essere superati, il codice supera la fase di linting, etc.) prima di determinare se queste possono essere accettabili o meno;
- Il version control è fondamentale
- Alla base di metodologie di sviluppo agile

Strumenti per continuous integration

- Ne esistono molteplici: self-hosted (Jenkins, Drone, etc.), come servizio (BitBucket Pipelines, Github Actions, etc.), o piattaforme dedicate (Travis CI, Bamboo, etc.).
- Si possono integrare altri controlli automatizzati (linting, testing, analisi del codice, etc.);
- I cambiamenti possono essere “accettati” in automatico al superamento di tutti i controlli, oppure richiedere una “revisione” per l’approvazione (chi vuole revisionare decine di righe modificate per poi scoprire che le modifiche rendono il progetto non funzionante?);
- Esiste anche “continuous deployment”, che consiste nel “distribuire” costantemente nell’ambiente di produzione i cambiamenti “accettati”

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Cloud e siti

Un nuovo trend

- Piaccia o no, con tutti i problemi e i vantaggi del caso il cloud si diffonde sempre più
- Sta rivoluzionando l'approccio con cui sviluppiamo applicazioni e siti Web
- Esistono tantissime piattaforme "simili ma diverse", ne elencheremo solo un paio particolarmente interessanti
- Di solito offrono piani gratuity, e piani a pagamento con maggiori funzionalità

Serverless

- Nuovo trend nel web development
- Consente di scrivere funzioni eseguite server-side, ma in un ambiente cloud gestito
- Si elimina la gestione dell'ambiente di esecuzione (installazione, configurazione, manutenzione, etc.) delegandola al fornitore del servizio;
- Consente di costtruire un backend senza dover gestire un server

Alcuni servizi e piattaforme

- Amazon Web Services (AWS);
- Google Cloud Platform (GCP)
- Azure (Microsoft);
- Heroku

Piattaforme e servizi per SS

- Permettono di sfruttare tutti i vantaggi forniti dagli SSG
- I siti vengono archiviati sotto forma di repository git
- Ad ogni commit, la piattaforma provvede a scaricare il codice, compilarlo ed effettuare il deployment
- Alcune di esse forniscono servizi per “superare” le limitazioni dei siti statici (funzioni serverless integrate, gestione di form, etc.).

Netlify e Cloudflare Pages

- Sono particolarmente interessanti
- La prima integra maggiori servizi utilizzabili più facilmente, la seconda fornisce prestazioni migliori grazie ad edge-computing (i nodi sono geograficamente più vicini agli utenti)
- La prima sfrutta infrastrutture AWS, la seconda quelle di CloudFlare (nessuno è indipendente, tutto è interconnesso!)
- Sfruttano Content Delivery Network (CDN) per offrire prestazioni eccellenti
- Hanno piani gratuiti molto validi

Cloudflare Workers

- Particolarmente interessante rispetto alla concorrenza
- Non fornisce un ambiente Node, ma esegue il codice JavaScript server-side sfruttando l'engine V8 (alla base di Node, Chrome ed altri tool)
- Ciò permette di avere prestazioni eccellenti, e non soffrire del cosìddetto "cold boot" (tempo aggiuntivo usato per attivare l'ambiente di esecuzione alla prima richiesta dopo il deployment)
- Edge computing, ma distribuzione e sincronizzazione tra i nodi sono gestiti automaticamente dall'infrastruttura
- Supporta qualsiasi languaggio che possa essere "compilato" in JavaScript o WebAssembly

Riferimenti I

- npm (npmjs.com)
- GitHub - yarnpkg/yarn: The 1.x line is frozen - features and bugfixes now happen on https://github.com/yarnpkg/berry
- Semantic Versioning 2.0.0 | Semantic Versioning (semver.org)
- Sass: Sass Basics (sass-lang.com)
- Getting started | Less.js (lesscss.org)

Riferimenti II

- PostCSS - a tool for transforming CSS with JavaScript
- GitHub - csstools/postcss-preset-env: Convert modern CSS into something browsers understand
- GitHub - css-modules/css-modules: Documentation about css-modules
- Cssdb
- Can I use... Support tables for HTML5, CSS3, etc

Riferimenti III

- ESLint - Pluggable JavaScript linter
- TypeScript: Typed JavaScript at Any Scale. (typescriptlang.org)
- Webpack
- rollup.js (rollupjs.org)
- GitHub - terser/terser:  JavaScript parser, mangler and compressor toolkit for ES6+

Riferimenti IV

- Grunt: The JavaScript Task Runner (gruntjs.com)
- gulp.js (gulpjs.com)
- Vue CLI (vuejs.org)
- Creare una Nuova App React – React (reactjs.org)
- Angular - CLI Overview and Command Reference

Riferimenti V

- esbuild - An extremely fast JavaScript bundler
- Home | Vite (vitejs.dev)
- Composer (getcomposer.org)
- Git (git-scm.com)

Riferimenti VI

- The world's fastest framework for building websites | Hugo (gohugo.io)
- Eleventy, a simpler static site generator (11ty.dev)
- Astro
- Cloudflare Pages
- Netlify: Develop & deploy the best web experiences in record time

Riferimenti VII

- Cloud Application Platform | Heroku
- Cloudflare Workers®
- Crea subito il tuo account Azure gratuito | Microsoft Azure
- Servizi di cloud computing: Amazon Web Services (AWS)
- Cloud computing, servizi di hosting e API | Google Cloud

Riferimenti VIII

- Jenkins
- Travis CI - Test and Deploy Your Code with Confidence (travis-ci.org)
- Drone CI – Automate Software Testing and Delivery

Riferimenti IX

- Bitbucket | The Git solution for professional teams
- GitHub: Where the world builds software · GitHub
- Features • GitHub Actions · GitHub
- Bitbucket Pipelines - Continuous delivery | Bitbucket
- Daring Fireball: Markdown Syntax Documentation

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Domande?