DESPLIEGUE DE APLICACIONES WEB

2. Utilización de servicios de red y automatización

En 2009 se creó Node. Js, un entorno de programación para escribir aplicaciones basado en el motor V8 de JavaScript de Chrome. Sus usos más habituales son el desarrollo de servicios web, los servidores de aplicaciones (como pueden ser chats o servidores de juegos) y las herramientas para la línea de comandos. Por estos motivos, Node. Js ha convertido en un componente fundamental en el desarrollo de aplicaciones modernas.

Como JavaScript es uno de los lenguajes fundamentales para todo desarrollador de aplicaciones web, aprender a desarrollar aplicaciones con Node. Js y utilizar las herramientas de la línea de comandos es bastante sencillo.

Estas herramientas facilitan mucho el trabajo de los desarrolladores y aumentan la productividad, ya que permiten generar los archivos optimizados por el despliegue a partir de los archivos de producción.

minimización

Término proveniente del inglés *minification* . Eliminación de todos los espacios y saltos de línea y sustitución de los nombres de variables y funciones por otros con el mínimo número de caracteres posibles. Así, `totalProductes = preUnitari * quantitatProducte;` se convierte en algo parecido a `a=b*c;`.

Entre muchas otras funciones, estas herramientas facilitan las tareas siguientes:

- Preprocesar ficheros Less y Sass para generar CSS .
- Compilar código ES6 o transcripción en ES5 (la versión de JavaScript que utilizan la mayoría de los navegadores).
- Copiar archivos entre carpetas (por ejemplo, desde las carpetas de las bibliotecas de desarrollo en las carpetas públicas de la aplicación para desplegar).

Aunque estas herramientas no se ejecutan en el lado del cliente, su finalidad es procesar el código para generar la versión de la aplicación web para desplegar.

Cabe destacar que como se trata de herramientas que funcionan completamente mediante la línea de comandos, pueden utilizarse en cualquier entorno, como puede ser la terminal del sistema operativo, un servidor remoto al que se conecte utilizando SSH o una máquina virtual gestionada por Vagrant.

Estas herramientas pueden instalarse de diferentes maneras:

- Descargándolas desde una página web.
- Importante-desde un repositorio de control de versiones (como puede ser GitHub).
- Utilitzant un gestor de paquets com npm o Yarn.

escollit aquesta versió del llenguatge), concatenar els fitxers JS generats per reduir el nombre de peticions i minimitzar el fitxer normal. En el cas de projectes grans, pot ser molt fàcil ometre algun pas o concatenar els fitxers en un ordre incorrecte.

La solució a aquest problema és utilitzar alguna eina d'automatització com GULP o GRUNT, que permeten configurar i executar de forma ordenada totes les tasques necessàries. D'aquesta manera, només cal executar l'automatitzador per generar tots els fitxers necessaris per al desplegament.

2.1. Instal·lació de Node.js i npm

El primer pas per poder utilitzar qualsevol eina basada en Node.js és instal·lar Node.js. Podeu descarregar l'instal·lador des de la pàgina web corresponent: nodejs.org/es.

APT

Acrònim d'*advanced packaging tool* (eina avançada d'empaquetat). Sistema de gestió de paquets creat pel projecte Debian (inclòs a Ubuntu) que permet instal·lar i eliminar programes mitjançant la línia d'ordres.

Cal destacar que en el cas d'algunes distribucions de Linux el programari Node.js pot venir preinstal·lat o pot instal·lar-se mitjançant un gestor de paquets (com APT), però aquesta versió acostuma a estar molt més endarrerida que les versions que es troben a la pàgina de Node.js.

Un cop instal·lat a l'equip, per comprovar que la instal·lació s'ha portat a terme correctament, heu d'accedir al símbol del sistema o la terminal (segons el sistema operatiu) i introduir:

```
node -v
```

El nom del programa Node.js a Linux és `nodejs`, per evitar conflictes amb altres paquets.

O en el cas de **Linux**:

```
nodejs -v
```

npm

És, per defecte, el gestor de paquets per a Node.js (www.npmjs.com) .

La instal·lació de Node.js també acostuma a incloure el gestor de paquets npm, que permet actualitzar-lo i instal·lar biblioteques, eines i mòduls per utilitzar en els programes

forma independent.

La instal·lació a Linux mitjançant els gestors de paquets pot ser problemàtica, ja que habitualment no tenen les versions més recents de Node.js ni npm als seus repositoris. Com que les versions més antigues de npm són rebutjades pels servidors, no és possible instal·lar nous paquets.

En el cas d'**Ubuntu**, per actualitzar la versió de Node.js i npm cal escriure a la terminal:

```
sudo apt-get install curl
sudo apt-get purge nodejs npm
curl -sL https://deb.nodesource.com/setup_6.x | sudo bash -
sudo apt-get install -y nodejs
```

2.1.1. Creació d'un servidor HTTP amb Node.js

Per comprovar el funcionament de Node.js, podeu provar el programa següent. Consisteix en la creació d'un servidor HTTP que respon a qualsevol petició amb el missatge "Hola món" en format text pla.

Creeu un fitxer anomenat hola-mon.js, amb el contingut següent:

```
var http = require('http');
http.createServer(function(peticio, resposta) {
        resposta.writeHead(200, {'Content-Type':'text/plain;charset=utf-8'});
        resposta.end('Hola món');
}).listen(8080, '127.0.0.1');
console.log('Servidor executant-se a http://127.0.0.1:8080/');
```

Executeu-lo, escrivint a la línia d'ordres:

```
node hola-mon
```

O en el cas de **Linux**:

```
nodejs hola-mon
```

> Els programes desenvolupats amb Node.js s'anomenen mòduls.

Aquest codi realitza les accions següents:

- Importa el mòdul `http`.
- Crea un servidor que escolta pel port `8080` i l'adreça IP `127.0.0.1`.
- La resposta del servidor a totes les peticions és la següent: escriu a la capçalera el codi 200, defineix el tipus de contingut com a text pla amb el joc de caràcters UTF-8 i afegeix com a cos de la resposta el missatge "Hola món".
- Mostra un missatge a la terminal per indicar a l'usuari que s'ha iniciat el servidor.

Si obriu l'URL 127.0.0.1:8080 al vostre navegador, veureu que es mostra una pàgina en blanc amb el missatge "Hola Món". Si inspeccioneu el codi, el resultat serà semblant al següent (pot variar segons el navegador que feu servir):

DESPLIEGUE DE APLICACIONES WEB

```
    <link rel="alternate stylesheet" type="text/css" href="resource://gre-resources,
  </head>
  <body>
    <pre>Hola Món</pre>
  </body>
</html>
```

Com es pot apreciar, és molt senzill crear un servidor amb Node.js, perquè es tracta d'un entorn de programació orientat a la creació de serveis web.

2.2. Gestors de paquets

A l'hora d'instal·lar una biblioteca o mòdul per utilitzar a les aplicacions, disposeu de diverses opcions, però una de les més recomanables és utilitzar un gestor de paquets especialitzat en el desenvolupament d'aplicacions web.

L'avantatge d'utilitzar els gestors de paquets és que la informació d'aquestes biblioteques passa a ser gestionada pel gestor de paquets. Això facilita la distribució de l'aplicació i l'actualització de les dependències, ja que a partir del fitxer de configuració (per exemple, el fitxer package.json per a npm) podeu tornar a descarregar tots els fitxers necessaris per al projecte. A més, els gestors s'encarreguen de descarregar també totes les dependències. És a dir, si una biblioteca requereix cinc mòduls diferents, aquests mòduls es descarreguen automàticament.

Entre els gestors de paquets més utilitzats hi ha **npm** i **Yarn**. npm va ser creat poc després de l'aparició de Node.js, té més de 300.000 paquets enregistrats i és utilitzat per més de 5 milions de desenvolupadors. Per la seva banda, Yarn va ser desenvolupat pels enginyers de Facebook en col·laboració amb els enginyers d'Exponent, Google i Tilde.

Un altre gestor de paquets molt popular però que **no es recomana utilitzar** és **Bower**, ja que es va deixar des desenvolupar a finals de l'any 2016. Aquest gestor de paquets permetia fer una instal·lació "plana" de les dependències, de manera que s'evitava la duplicació de biblioteques, al contrari que npm, que descarrega les dependències de cada paquet individualment (es troben imbricades). Habitualment s'utilitzava npm per als paquets utilitzats a la banda del servidor i Bower per als paquets utilitzats a la banda del client.

2.2.1. npm

> **npm**
>
> Podeu trobar més informació sobre el gestor de paquets npm a l'enllaç següent: www.npmjs.com.

npm és el gestor de paquets més utilitzat i es troba inclòs a la instal·lació de Node.js. Per comprovar que es troba instal·lat correctament al vostre equip podeu escriure a la línia d'ordres:

```
npm -v
```

amb l'ordre següent:

```
npm install express
```

> La instal·lació de paquets de forma global pot requerir executar l'ordre com a administrador o superusuari.

En executar aquesta ordre, es descarrega la biblioteca Express juntament amb totes les seves dependències, que queden emmagatzemades dins d'una carpeta a l'arrel del projecte anomenada *node_modules*.

A més a més, pot actualitzar-se a si mateix amb l'ordre següent:

```
npm install npm@latest -g
```

Fixeu-vos en el paràmetre `-g`. Indica que aquesta ordre s'ha d'executar per a la instal·lació global de npm. Utilitzant aquest mateix paràmetre poden instal·lar-se paquets globalment. Això és especialment útil a l'hora d'instal·lar eines per a la línia d'ordres.

Vegeu com s'instal·la **Babel**, el compilador de TypeScript i ES6:

```
npm install babel-cli -g
```

Aquest gestor de paquets utilitza el fitxer package.json per llegir la informació del projecte, incloent-hi els autors, la versió i la llicència. Si esteu començant el vostre projecte des de zero, podeu crear-lo manualment amb un editor de text pla o utilitzant l'ordre `npm init` i omplint les dades.

Vegeu un exemple d'inicialització del projecte "hola-mon" a continuació:

```
xavier@Ubuntu-Node:~/hola-mon$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (hola-mon)
version: (1.0.0)
description: Primer programa amb Node.js
entry point: (hola-mon.js)
test command:
git repository:
keywords:
author: Xavier Garcia
license: (ISC)
About to write to /home/xavier/hola-mon/package.json:

{
  "name": "hola-mon",
  "version": "1.0.0",
```

```
    "express": "^4.14.0"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "el vostre nom",
  "license": "ISC"
}
```

Fixeu-vos que a `dependencies` apareixen els mòduls i les biblioteques instal·lades al vostre projecte mitjançant npm. Així, si heu instal·lat la biblioteca Express a la llista de dependències, apareix amb el número de versió.

Aquest fitxer es pot modificar manualment amb un editor de text per modificar els continguts, afegir noves dependències o eliminar les que hi ha, tot i que en aquest últim cas cal recordar que no s'eliminen els fitxers afegits al projecte (continguts a la carpeta *node_modules*). Per eliminar-los, cal utilitzar l'ordre `uninstall`. Per exemple, per eliminar el paquet Express del directori *node_modules* i totes les seves dependències cal d'escriure:

```
npm uninstall express
```

Cal destacar que l'actualització del fitxer package.json no es realitza automàticament en utilitzar les ordres `npm install` ni `npm uninstall`. Per a aquesta actualització s'han d'utilitzar les opcions següents:

- `-S`, `–save`: el paquet apareix o és eliminat de `dependencies`.
- `-D`, `–save-dev`: el paquet apareix o és eliminat de `devDependencies`.
- `-O`, `–save-optional`: el paquet apareix o és eliminat d'`optionalDependiencies`.

És a dir, si proveu d'instal·lar el paquet Express amb l'opció `-S`, apareix automàticament dins del fitxer package.json a la secció `dependencies`:

```
npm install Express -S
```

I si ho desinstal·leu amb la mateixa opció, serà eliminat de la secció `dependencies`:

```
npm uninstall Express -S
```

Opcions d'npm

Podeu trobar totes les opcions del client npm a l'enllaç següent: docs.npmjs.com/cli.

Els sistemes de control de versions (per exemple, Git o Subversion) permeten mantenir el control de tots els canvis realitzats en els fitxers d'un projecte al llarg del temps.

És important que les dependències estiguin enregistrades correctament al fitxer package.json. Això permet executar l'ordre `npm install` sense especificar cap altre argument i

fitxers del directori *node_modules*.

Proveu d'afegir novament la biblioteca Express al vostre projecte com a dependència (opció `-S`) i, seguidament, esborreu la carpeta *node_modules*, amb les següents ordres:

```
npm install Express -S
rm -Rf node_modules
```

Executeu l'ordre `npm install`, sense cap altre argument per instal·lar totes les dependències definides al fitxer package.json:

```
npm install
```

Com podeu comprovar, llistant el contingut del directori, s'ha tornat a generar la carpeta *node_modules* i s'han afegit la biblioteca Express i totes les seves dependències.

2.2.2. Yarn

Yarn (yarnpkg.com) pot instal·lar-se mitjançant npm però no es recomana: la instal·lació no es realitza de forma determinista, el paquet no és signat i només es fa una comprovació d'integritat, com a mesura de seguretat. En conseqüència, els desenvolupadors de Yarn estimen que aquest tipus d'instal·lació és un risc de seguretat per a aplicacions grans.

Tot i així, en cas de voler instal·lar-lo utilitzant npm, l'ordre seria la següent:

```
npm install yarn -g
```

Fixeu-vos que al començament de la instal·lació mitjançant npm apareix un avís similar al següent:

```
npm WARN deprecated yarn@0.24.4: It is recommended to install Yarn using the native
```

És a dir, es recomana instal·lar Yarn utilitzant el sistema nadiu de cada sistema operatiu que pot trobar-se a l'enllaç següent: yarnpkg.com/en/docs/install.

Yarn va ser desenvolupat per Facebook per solucionar els problemes que tenien en utilitzar npm com a gestor de paquets. Quan treballaven en projectes grans, es trobaven que per a cada canvi en algun dels fitxers es produïen enviaments de canvis al repositori amb centenars de milers de línies (que s'havien regenerat automàticament), i per solucionar els problemes produïts es perdia un dia de feina d'un dels enginyers.

Per altra banda, no hi havia cap garantia que les dependències es descarreguessin en el mateix ordre en tots els equips, de manera que podien produir-se inconsistències: segons l'ordre de descàrrega, la versió d'alguna dependència podia canviar (per exemple, perquè es tractava d'una subdependència).

> **React Native** és un entorn de desenvolupament per crear aplicacions mòbils natives utilitzant JavaScript i React (altre entorn de desenvolupament). Tant React com React Native són propietat de Facebook.

A més a més, el nombre de fitxers descarregats per npm en alguns projectes era exageradament gran. Per exemple, en instal·lar React Native, que tenia 62 dependències, s'havien de descarregar, a la carpeta *node_modules*, més de 120.000 fitxers.

Per resoldre aquests problemes, Facebook, juntament amb Exponent, Google i Tilde, va crear Yarn com un gestor de paquets més ràpid i fiable del qual destaquen les característiques següents:

- Cau fora de línia: Yarn descarrega, al disc de l'equip, una còpia de cada paquet utilitzat i es diferencien per registre d'origen (per exemple, npm) i versió (per exemple, 4.1.4). D'aquesta manera, es poden fer instal·lacions fora de línia i només es descarreguen els paquets que no es trobin al cau.
- Instal·lacions deterministes: la instal·lació dels paquets en qualsevol equip sempre es du a terme en el mateix ordre i amb la mateixa versió dels paquets.
- Ràpid: la instal·lació de paquets és més rapida perquè es realitza en paral·lel i fa servir el cau per als paquets que s'han descarregat prèviament.
- Segur: abans d'executar el codi de qualsevol paquet Yarn, realitza una comprovació d'integritat del fitxer.

Tot i que pot semblar que aquest gestor és superior a npm, sembla que en alguns casos no funciona correctament. Respecte a les instal·lacions deterministes, es poden obtenir a npm mitjançant l'opció `shrinkwrap` o indicant al fitxer package.json la versió exacta que es vol utilitzar.

2.3. Preprocessadors de CSS

Els fulls d'estil en cascada (en anglès, *cascading style sheets* o CSS) són un element indispensable per canviar la representació de qualsevol pàgina o aplicació web, ja que aquest llenguatge és el responsable de modificar l'aspecte que mostren els elements HTML. Gràcies al CSS, es pot modificar la font utilitzada en un text, la mida o el color, indicar el tipus de vora que es vol afegir, etc. En les versions més recents, fins i tot es pot animar qualsevol element HTML.

Malauradament, CSS és un llenguatge de marques i no inclou funcionalitats avançades com la utilització de variables, operadors i funcions definides pels usuaris. Això provoca greus problemes de manteniment, ja que en qualsevol lloc web és habitual haver de reutilitzar elements i configuracions (per exemple, el color principal de l'empresa, l'estil de les ombres o el radi d'una vora arrodonida).

Exemple de canvi del color principal al lloc web d'una empresa

En el lloc web d'una empresa s'ha estat fent servir el color vermell (`#ff00000`) com a color principal de l'empresa (corresponent al de la seva marca) per als botons, els enllaços, les capçaleres i el text destacat.

Un dia s'adonen que això no és correcte, que el color principal de la seva marca és (`#ff2020`) i s'ha de canviar a tot arreu. Això implica modificar tots els fitxers CSS i fer un reemplaçament on s'hagi fet servir aquest color, però només quan es mostra com a color de la marca (per exemple, sense modificar el color quan es fa servir per mostrar errors).

missatges d'error amb el nou color.

Un cas més extrem podria consistir a canviar els colors d'un entorn de treball complet, com per exemple Bootstrap, per ajustar-lo al de l'empresa. Això implicaria canviar no sols el color principal, sinó també els múltiples colors que es poden trobar a l'entorn, que són lleugerament més clars o més foscos que el color principal. Aquest tipus de canvi requeriria modificar centenars de línies de codi (a la versió 3.3.7 de Bootstrap hi ha més de 500 elements amb propietats que modifiquen el color).

Per solucionar aquests problemes es van crear els preprocessadors de CSS.

Característiques dels preprocessadors

Podeu trobar exemples de la implementació de les característiques dels preprocessadors a l'enllaç següent: goo.gl/GSKSx8.

Els preprocessadors de CSS fan servir un llenguatge propi que augmenta el llenguatge CSS. Llavors, es preprocessen aquests fitxers i es generen els fitxers amb el codi CSS. Entre les característiques que es poden trobar en aquests llenguatges hi ha les següents:

- **Declaració de variables**: permeten emmagatzemar valors com colors o dimensions (amplada, alçària, mida de font, etc.).
- **Operadors**: permeten realitzar operacions entre nombres o variables.
- ***Mixins***: permeten afegir blocs de codi CSS configurats utilitzant arguments (de forma similar a les funcions).
- **Funcions predefinides**: cada preprocessador inclou les seves, però és habitual trobar la funció `lighten`, que permet aclarir un color, o `darken`, que permet enfosquir-lo.
- **Importació**: permet importar altres fitxers, de manera que és més fàcil organitzar-los. Per exemple, el fitxer "principal" pot importar els fitxers "variables", "colors", "mides" i "pagina". El resultat de preprocessar "principal" serà un únic fitxer CSS amb el contingut dels 5 fitxers.
- **Imbricació** (en anglès, *nesting*): permet imbricar el codi generant una jerarquia en forma d'arbre, de manera que el codi generat és més senzill d'entendre i de modificar.
- **Condicionals i bucles**: permeten la creació de bucles i aplicar unes regles o unes altres de forma condicional.

Cal destacar que no totes les noves característiques que formen part de l'especificació de CSS es troben implementades a tots els navegadors. En alguns casos es requereix utilitzar un prefix especial perquè encara es troben en fase experimental o pot ser que no siguin implementades. Per aquesta raó, alguns preprocessadors inclouen automàticament les versions prefixades de les regles més recents i, quan és possible, alguna alternativa per als navegadors més antics.

S'ha de tenir en compte que segons quin preprocessador s'utilitza es pot treballar directament amb codi CSS estàndard o no, ja que no tots l'admeten.

Els preprocessadors més populars són els següents:

- **Sass**: és més potent que Less però més complicat d'aprendre, tot i que amb la nova sintaxi és molt més similar a Less.
- **Less**: inclou menys característiques que Sass, tot i que les més importants es troben presents i és més fàcil d'aprendre perquè la seva sintaxi és molt similar a la de CSS.

també ofereix més opcions en tots els àmbits.

Com que Sass i Less fa més temps que circulen, és habitual trobar que els entorns de treball i biblioteques ofereixen, a banda dels fitxers CSS, els fitxers en format Sass o Less, perquè es puguin fer les pròpies compilacions. Per exemple, Bootstrap ofereix l'opció de descarregar els fitxers en tots dos formats, de manera que canviar el color principal de l'entorn només requereix canviar una única variable i preprocessar-lo.

2.3.1. Sass

> Les sigles Sass signifiquen "fulls d'estil sintàcticament impressionants" (en anglès, *syntactically awesome stylesheets*).

Sass (sass-lang.com) és un projecte de codi lliure que ha estat en desenvolupament durant més de deu anys i ha assentat les bases dels preprocessadors CSS moderns.

Actualment accepta dues sintaxis diferents:

- **Sass** (extensió ".sass"): té la sintaxi original, no és directament compatible amb CSS.
- **SCSS** (extensió ".scss"): té una nova sintaxi més similar a CSS i Less.

Convé tenir en compte que la sintaxi Sass no és compatible directament amb CSS, al contrari que SCSS. És a dir, si el contingut d'un fitxer Sass és codi CSS, es produeix un error en preprocessar-lo; en canvi, amb SCSS és preprocessa sense problema.

Sass inclou les següents característiques: variables, imbricació, importació i parcials, *mixins*, herència, operadors i funcions predefinides.

> **Preprocessador Sass en línia**
>
> Per preprocessar el codi Sass en línia i veure'n el resultat immediatament es pot visitar l'enllaç següent: www.sassmeister.com.

Les **variables** s'identifiquen perquè van prefixades pel símbol `$`. Per exemple, per establir el color i la mida de la font utilitzant variables es fa de la manera següent:

```
$mida-font: 15px;
$color: #ff0000;

p {
  font-size: $mida-font;
  color: $color;
}
```

I, en ser preprocessat, generaria la sortida següent:

```
p {
  font-size: 15px;
  color: #ff0000;
}
```

elements com a llistes desordenades (`ul`), elements de les llistes (`li`) i enllaços (`a`).

La **imbricació** permet imbricar elements dintre d'altres elements.

En el cas d'una barra de navegació (element `nav`), es poden afegir els estils de tots els elements que s'apliquen només quan aquests es troben dintre de la barra de navegació:

```
nav {
  ul {
    list-style: none;
    margin: 0;
    padding: 0;
    color: light-grey;
  }

  li {
    display: inline-block;
  }

  a {
    text-decoration:none;
    display: block;
    padding: 5px;
  }
}
```

Com es pot apreciar, es veu molt clarament com s'organitzen els estils. Vegeu, a continuació, el codi generat en preprocessar-lo:

```
nav ul {
  list-style: none;
  margin: 0;
  padding: 0;
  color:light-grey;
}

nav li {
  display: inline-block;
}

nav a {
  text-decoration: none;
  display: block;
  padding: 5px;
}
```

En aquest cas no hi ha gaire diferència, però es perd la claredat que proporciona visualitzar els continguts jerarquitzats.

Sass permet **importar fitxers sencers o parcials** mitjançant la directriu `@import`. La diferència es troba en el fet que els fitxers parcials no són preprocessats en fitxers individuals, ja que són inclosos en altres fitxers. El nom dels fitxers parcials sempre ha de començar per una barra baixa (_).

Exemple d'importació de fitxers

manera següent:

```
@import 'parcial';
@import 'colors';
```

Fixeu-vos que en cap dels dos casos no s'hi ha afegit l'extensió i s'ha prescindit de la barra baixa en el nom del parcial. Sass s'encarrega automàticament de cercar els fitxers correctes.

Per declarar un ***mixin*** cal utilitzar la directriu `@mixin`, i per incloure-la cal fer servir la directriu `@include`. Aquests *mixins* es poden tractar com a funcions que són cridades passant un argument i que retornen el codi per inserir.

Per exemple, per afegir una ombra a diferents elements tenint en compte els prefixos dels diferents navegadors (`-webkit-`, `-moz-`, `-o-` i `-ms-`).

```
@mixin afegir-ombra($opacitat) {
  -webkit-box-shadow: 10px 10px 5px 0px rgba(0,0,0,$opacitat);
  -moz-box-shadow: 10px 10px 5px 0px rgba(0,0,0,$opacitat);
  box-shadow: 10px 10px 5px 0px rgba(0,0,0,$opacitat);
}

p {
  margin: 20px;
  @include afegir-ombra(0.75);
}

span {
  @include afegir-ombra(0.50);
}

div {
  @include afegir-ombra(1);
}
```

Fixeu-vos que dins del *mixin* s'hi han afegit les versions prefixades de la regla CSS, de manera que es genera el codi compatible amb el màxim de navegadors possibles. El resultat de preprocessar aquest codi és el següent:

```
p {
  margin: 20px;
  -webkit-box-shadow: 10px 10px 5px 0px rgba(0, 0, 0, 0.75);
  -moz-box-shadow: 10px 10px 5px 0px rgba(0, 0, 0, 0.75);
  box-shadow: 10px 10px 5px 0px rgba(0, 0, 0, 0.75);
}

span {
  -webkit-box-shadow: 10px 10px 5px 0px rgba(0, 0, 0, 0.5);
  -moz-box-shadow: 10px 10px 5px 0px rgba(0, 0, 0, 0.5);
  box-shadow: 10px 10px 5px 0px rgba(0, 0, 0, 0.5);
}

div {
  -webkit-box-shadow: 10px 10px 5px 0px black;
  -moz-box-shadow: 10px 10px 5px 0px black;
  box-shadow: 10px 10px 5px 0px black;
}
```

Com es pot apreciar, la versió SCSS, a banda de ser més clara, permet canviar la mida de totes les ombres només modificant el *mixin*, i totes les regles s'actualitzarien quan es preprocessés el fitxer.

ple, es pot crear un selector `icona`, amb les propietats bàsiques de totes les icones, i després fer que cada icona concreta sigui herència d'aquesta (imprimir, previsualitzar, esborrar, etc.), com es pot veure a continuació:

```
.icona {
        width: 16px;
        height: 16px;
        margin: 2px;
}

.imprimir {
  @extend .icona;
  background-image: url('imprimir.png');
}

.previsualitzar {
  @extend .icona;
  background-image: url('previsualitzar.png');
}

.esborrar {
  @extend .icona;
  background-image: url('esborrar.png');
}

.desar {
  @extend .icona;
  background-image: url('desar.png');
}
```

Utilitzant l'herència no s'ha de modificar la definició d' `.icona` en cap moment, de manera que es poden afegir nous botons en qualsevol altre punt del fitxer, o fins i tot en fitxers diferents, i farien servir la mateixa definició. El resultat seria el següent:

```
.icona, .imprimir, .previsualitzar, .esborrar, .desar {
  width: 16px;
  height: 16px;
  margin: 2px;
}

.imprimir {
  background-image: url("imprimir.png");
}

.previsualitzar {
  background-image: url("previsualitzar.png");
}

.esborrar {
  background-image: url("esborrar.png");
}

.desar {
  background-image: url("desar.png");
}
```

Com es pot apreciar, per cada nou botó que s'afegeix en CSS, s'ha de modificar la regla que defineix `.icona` per incloure la nova classe, i això faria que afegir noves icones fos més enrevessat, sobretot si no es trobessin definides en el mateix fitxer CSS.

Sass inclou els **operadors** matemàtics de suma, resta, multiplicació, divisió i mòdul. Aquests operadors es poden utilitzar per fer càlculs. Per exemple, per ajustar la mida de

```
$mida-font: 14px;

p {
  font-size: $mida-font;
}

small {
  font-size: $mida-font / 3 * 2;
}

h1 {
  font-size: $mida-font * 2;
}

h2 {
  font-size: $mida-font * 1.5;
}

h3 {
  font-size: $mida-font * 1.2;
}

h4 {
  font-size: $mida-font + 1;
}
```

Fixeu-vos que es poden combinar els operadors amb l'ús de variables. El resultat és el següent:

```
p {
  font-size: 14px;
}

small {
  font-size: 9.33333px;
}

h1 {
  font-size: 28px;
}

h2 {
  font-size: 21px;
}

h3 {
  font-size: 16.8px;
}

h4 {
  font-size: 15px;
}
```

L'avantatge d'usar els operadors per a aquest tipus de càlcul és que en cas d'haver de modificar la mida de la font només caldria modificar el valor de la variable; la resta s'actualitzaria automàticament.

Funcions predefinides de Sass

Podeu trobar una llista completa d'aquestes funcions a l'enllaç següent: goo.gl/mcZFoZ.

per crear variacions d'un color, com es pot veure a continuació:

```
$color-principal: blue;
$color-vora: darken($color-principal, 20%);
$color-enllac: lighten($color-principal, 20%);

div {
  color: $color-principal;
  border: 1px solid $color-vora;

  a {
    color:$color-enllac;
  }
}
```

Exemple d'ús de funcions predefinides en Sass

En l'exemple següent s'estableix el color blau per als continguts del contenidor `div`, però afegint una vora de color blau enfosquida un 20% i els enllaços de color blau aclarits un 20%.

El codi CSS generat és el següent:

```
div {
  color: blue;
  border: 1px solid #000099;
}

div a {
  color: #6666ff;
}
```

Si en lloc d'utilitzar el color blau, es vol utilitzar el color vermell o qualsevol altre color, només cal canviar el valor de la variable `$color-principal`, i automàticament tot el lloc web passa a ser del nou color, incloses les vores i els enllaços (aquest és el sistema que utilitzen les plantilles i entorns de treball com Bootstrap).

Ruby és un llenguatge de programació empleat a la banda del servidor. El gestor de paquets d'aquest llenguatge s'anomena *RubyGems* i els paquets s'anomenen *gemmes*.

Originalment, per instal·lar Sass era necessari tenir instal·lat el llenguatge de programació Ruby, ja que Sass s'instal·la com una gemma. Així, segons el sistema operatiu, s'ha d'instal·lar Ruby en primer lloc (a Ubuntu, per exemple, ja es troba preinstal·lat).

Tot i que es pot instal·lar el preprocesador de Sass com un paquet de Node.js mitjançant npm, és possible que no funcioni si el sistema no té les biblioteques del llenguatge necessàries instal·lades (Ruby o, en versions més recents, C). En cas que sigui necessari, podeu instal·lar Ruby seguint les instruccions per al vostre sistema operatiu a l'enllaç següent: goo.gl/Bo6ehR.

Per instal·lar el preprocessador mitjançant npm, escriviu a la línia d'ordres:

Una vegada instal·lat, per utilitzar-lo, haureu de crear un fitxer de text que serà executat per Node.js, amb el codi següent:

```
var fs = require("fs");
var sass = require("node-sass");

sass.render(
  {
    file: "nom-del-vostre-fitxer.scss"
  },
  function(error, result) {
    if (!error) {
      // No s'han trobat errors, es desarà al disc
      fs.writeFile('resultat.css', result.css, function(err) {
        if (!err) {
          // Fitxer escrit al disc
          console.log("Preprocessament realitzat amb èxit");
        }
      });
    }
  }
);
```

Primer de tot, es carreguen els dos mòduls necessaris per executar el programa:

- **fs**: correspon al mòdul `FileSystem` (encarregat de les operacions de lectura i escriptura al disc).
- **node-sass**: correspon al mòdul preprocessador Sass.

Seguidament, s'invoca la funció `render` del mòdul `node-sass`, a la qual s'ha de passar, com a primer paràmetre, un objecte amb la configuració on s'indica el nom del fitxer (aquí heu de posar el nom del vostre fitxer Sass) i, com a segon paràmetre, la funció que serà executada en finalitzar el preprocessament.

Aquesta funció rep automàticament dos paràmetres: el primer és un objecte, si s'ha produït cap error, i en cas que no se n'hagi produït cap, un segon paràmetre amb el resultat del preprocessament. Com que el resultat es troba a la memòria, cal desar-lo en un fitxer. Això s'aconsegueix invocant la funció `writeFile` del mòdul `fs`, indicant el nom del fitxer de destí, el contingut del fitxer (obtingut de la propietat `css` del paràmetre rebut `result`) i una funció cridada automàticament en finalitzar el procés de desar les dades que, en aquest cas, serveix per comprovar si s'ha desat amb èxit o no.

Una vegada s'ha desat el fitxer, es pot executar com qualsevol altre programa de node. Per exemple, si heu anomenat "compilar-sass.js" al fitxer, l'ordre és la següent:

```
node compilar-sass.js
```

Recordeu que a **Linux** el nom del programa és diferent i, per tant, l'ordre és aquesta altra:

```
nodejs compilar-sass.js
```

Una vegada executat el programa, dintre del mateix directori hi ha un fitxer anomenat "resultat.css" (el nom s'ha indicat dins del programa), amb el codi Sass preprocessat.

2.3.2. Less

que poden interpretar-se en diferents dispositius.

Less (lesscss.org) és un preprocessador que pot executar-se a Node.js, al navegador (encara que només es recomana durant el desenvolupament) i a Rhino. Cal tenir en compte que, en conjunt, Sass és més complet que Less, però tant l'un com l'altre tenen les característiques bàsiques més importants.

Preprocessar Less en línia

Per preprocessar el codi Less en línia i veure'n el resultat, es pot visitar l'enllaç següent: less2css.org

A Less, s'hi poden trobar les característiques següents: variables, imbricació, importació, *mixins*, herència, operadors i funcions predefinides. A Less, per declarar **variables** s'utilitza el símbol @.

Exemple de definició de variables a Less

Vegeu com es defineixen dues variables per establir el color i la mida de la font dels elements de tipus paràgraf:

```
@mida-font: 15px;
@color: #ff0000;

p {
  font-size: @mida-font;
  color: @color;
}
```

El codi CSS preprocessat és el següent:

```
p {
  font-size: 15px;
  color: #ff0000;
}
```

La **imbricació** d'elements és idèntica a com es codifica a Sass (amb la sintaxi SCSS). Per imbricar els estils necessaris per formatar una barra de navegació, el codi Less és el següent:

```
nav {
  ul {
    list-style: none;
    margin: 0;
    padding: 0;
    color: light-grey;
  }

  li {
    display: inline-block;
  }

  a {
    text-decoration:none;
    display: block;
```

I el resultat CSS és el que trobeu a continuació:

```
nav ul {
  list-style: none;
  margin: 0;
  padding: 0;
  color: light-grey;
}
nav li {
  display: inline-block;
}
nav a {
  text-decoration: none;
  display: block;
  padding: 5px;
}
```

Less també inclou la capacitat d'importar altres fitxers que, en ser processats, són inclosos directament al fitxer CSS. Per fer-ho, s'utilitza la directriu `@import`, igual que a Sass, com es pot comprovar en l'exemple següent:

```
@import 'parcial';
@import 'colors.css';
```

Si no s'afegeix cap extensió al fitxer Less, s'interpreta que l'extensió ha de ser `.less`. En qualsevol cas (a excepció de l'extensió `.css`, que no és preprocessada), s'interpreta el fitxer com a codi Less.

En el cas dels ***mixins***, a Less no cal utilitzar cap directriu: es declaren com si fossin una classe però especificant els paràmetres que accepten si són necessaris.

Exemple de mixin a Less

Per crear un *mixin* que afegeixi una ombra a un element es fa de la manera següent:

```
.afegir-ombra(@opacitat) {
  -webkit-box-shadow: 10px 10px 5px 0px rgba(0,0,0,@opacitat);
  -moz-box-shadow: 10px 10px 5px 0px rgba(0,0,0,@opacitat);
  box-shadow: 10px 10px 5px 0px rgba(0,0,0,@opacitat);
}

p {
  margin: 20px;
  .afegir-ombra(0.75);
}

span {
  .afegir-ombra(0.50);
}

div {
  .afegir-ombra(1);
}
```

Fixeu-vos que per afegir el *mixin* als elements tampoc no es fa servir cap directriu, s'afegeixen com si fossin classes imbricades.

El codi resultant de preprocessar és aquest:

```
  -webkit-box-shadow: 10px 10px 5px 0px rgba(0, 0, 0, 0.75);
  -moz-box-shadow: 10px 10px 5px 0px rgba(0, 0, 0, 0.75);
  box-shadow: 10px 10px 5px 0px rgba(0, 0, 0, 0.75);
}
span {
  -webkit-box-shadow: 10px 10px 5px 0px rgba(0, 0, 0, 0.5);
  -moz-box-shadow: 10px 10px 5px 0px rgba(0, 0, 0, 0.5);
  box-shadow: 10px 10px 5px 0px rgba(0, 0, 0, 0.5);
}
div {
  -webkit-box-shadow: 10px 10px 5px 0px #000000;
  -moz-box-shadow: 10px 10px 5px 0px #000000;
  box-shadow: 10px 10px 5px 0px #000000;
}
```

Less també inclou la possibilitat d'**heretar** propietats d'altres classes, però en lloc d'utilitzar una directriu, s'utilitza el selector de pseudoclasse `extend`.

Exemple de sistema d'icones a Less

Un sistema d'icones en el qual totes les icones siguin herència de la classe `icona` que conté les propietats comunes podria implementar-se així:

```
.icona {
  width: 16px;
  height: 16px;
  margin: 2px;
}

.imprimir {
  &:extend(.icona);
  background-image: url('imprimir.png');
}

.previsualitzar {
  &:extend(.icona);
  background-image: url('previsualitzar.png');
}

.esborrar {
  &:extend(.icona);
  background-image: url('esborrar.png');
}

.desar {
  &:extend(.icona);
  background-image: url('desar.png');
}
```

El codi CSS generat seria el següent:

```
.icona,
.imprimir,
.previsualitzar,
.esborrar,
.desar {
  width: 16px;
  height: 16px;
  margin: 2px;
}
.imprimir {
  background-image: url('imprimir.png');
}
.previsualitzar {
  background-image: url('previsualitzar.png');
}
.esborrar {
  background-image: url('esborrar.png');
}
.desar {
```

El conjunt d'**operadors** matemàtics que ofereix no inclou el mòdul, només inclou la suma, la resta, la multiplicació i la divisió. Per exemple, per definir la mida de la font de diferents elements a partir d'una variable es fa de la manera següent:

```
@mida-font: 14px;

p {
  font-size: @mida-font;
}

small {
  font-size: @mida-font / 3 * 2;
}

h1 {
  font-size: @mida-font * 2;
}

h2 {
  font-size: @mida-font * 1.5;
}

h3 {
  font-size: @mida-font * 1.2;
}

h4 {
  font-size: @mida-font + 1;
}
```

Com es pot apreciar, a excepció del prefix de la variable, el codi és exactament el mateix que a Sass. El codi generat és el següent:

```
p {
  font-size: 14px;
}
small {
  font-size: 9.33333333px;
}
h1 {
  font-size: 28px;
}
h2 {
  font-size: 21px;
}
h3 {
  font-size: 16.8px;
}
h4 {
  font-size: 15px;
}
```

Funcions predefinides de Less

Podeu trobar una llista completa d'aquestes funcions a l'enllaç següent: lesscss.org/functions.

Igual que Sass, Less inclou moltes **funcions predefinides** per tractar el color, les cadenes de text o les llistes. La forma d'utilitzar-les és la mateixa, només cal invocar la funció passant com a argument els paràmetres requerits. Per exemple, per obtenir un color acla-

Vegeu, a continuació, com es poden combinar aquestes funcions per crear un esquema de color reutilitzable a partir d'un color principal:

```
@color-principal: blue;
@color-vora: darken(@color-principal, 20%);
@color-enllac: lighten(@color-principal, 20%);

div {
  color: @color-principal;
  border: 1px solid @color-vora;

  a {
    color:@color-enllac;
  }
}
```

El codi resultant és el següent:

```
div {
  color: blue;
  border: 1px solid #000099;
}
div a {
  color: #6666ff;
}
```

Fixeu-vos que si es canvia el color principal, els colors de la vora i dels enllaços també canvien. Per exemple, si canviem el color principal a vermell (`@color-principal: red`), el codi CSS generat passa a ser el següent:

```
div {
  color: red;
  border: 1px solid #990000;
}
div a {
  color: #ff6666;
}
```

Es pot instal·lar directament el gestor de paquets npm sense cap altre requisit de la terminal, amb l'ordre següent:

```
npm install less -g
```

> A macOS i UNIX l'ordre per executar Less és `lessc`, perquè `less` es correspon a una eina del sistema operatiu.

Per utilitzar-lo només cal introduir l'ordre `less`, indicant el nom del fitxer:

```
less estils.less
```

El resultat es mostra per pantalla. Si voleu que es desi en un fitxer, heu d'especificar el nom del fitxer com a segon paràmetre. Per exemple:

Alternativament, es pot importar com a mòdul de Node.js per crear els propis programes, per exemple, per automatitzar processos.

Exemple de programa Less desat en disc

Vegeu un programa d'exemple que carrega el fitxer indicat a la configuració i el desa al disc:

```
var fs = require("fs");
var less = require("less");
var entrada = fs.readFileSync('nom-del-vostre-fitxer.less', 'utf8');

less.render(entrada,
  {},
  function(error, result) {
    if (!error) {
      // No s'han trobat errors, es desarà al disc
      fs.writeFile('resultat.css', result.css, function(err) {
        if (!err) {
          // Fitxer escrit al disc
          console.log("Preprocessat realitzat amb èxit");
        }
      });
    }
  }
);
```

Cal destacar que mentre que en utilitzar el mòdul `node-sass` només cal indicar el nom del fitxer amb el codi d'entrada, en treballar amb el mòdul `less` cal llegir primer el fitxer (fs.readFileSync). A més, la lectura s'ha de fer de forma síncrona (per defecte és asíncrona), perquè en cas contrari podria executar-se el processament abans de finalitzar la lectura del fitxer d'entrada.

Less ofereix l'opció de preprocessar-se al navegador. Aquesta opció pot ser interessant durant el desenvolupament o si per alguna raó no es poden generar els fitxers CSS al servidor. Cal tenir en compte que aquesta opció provoca un retard en la presentació de la pàgina, ja que no es mostra la pàgina fins que no s'ha descarregat la biblioteca i s'han processat els fitxers CSS.

Per afegir la compilació al navegador, primer heu d'afegir la càrrega del fitxer less.js a la capçalera del fitxer HTML. Aquest fitxer pot estar allotjat al propi servidor o ser un enllaç a una xarxa de lliurament de continguts, per exemple.

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/less.js/2.7.2/less.min.js"></sc
```

A continuació, també dintre de la capçalera de la pàgina, heu d'afegir els enllaços als vostres fulls d'estil amb el format següent:

```
<link rel="stylesheet/less" type="text/css" href="estils.less" />
```

Haureu d'afegir una línia per a cada fitxer Less, especificant, a l'atribut `href`, l'URL on es troba el fitxer que voleu preprocessar.

Una vegada es carregui el fitxer less.min.js, cercarà tots els fitxers marcats com a `stylesheet/less`, els preprocessarà i els afegirà com a fulls d'estil CSS.

Stylus (stylus-lang.com) és un preprocessador que inclou totes les característiques de Sass i Less (està inspirat en tots dos) però també hi afegeix les seves pròpies característiques. Com que la seva sintaxi és força diferent de la de Sass i Less i ofereix gran quantitat d'opcions, Stylus és un preprocessador més difícil de dominar.

Una diferència important amb altres preprocessadors és que els punts i comes, les comes i els claudàtors són opcionals. Per altra banda, el sagnat ha de fer-se correctament, ja que en cas contrari el resultat no seria l'esperat. Una altra diferència és que les variables a Stylus poden declarar-se al mateix lloc que es fan servir i no requereixen cap prefix especial (tot i que es pot fer servir el símbol $).

Funcions predefinides d'Stylus

Podeu trobar una llista completa d'aquestes funcions a l'enllaç següent: goo.gl/tiI93I.

Pel que fa a les funcions, Stylus no només té funcions predefinides sinó que permet crear funcions pròpies que retornin valors i no només mostrin codi CSS. El conjunt d'operadors admesos és molt superior que el d'altres processadors, ja que inclou operadors binaris, unaris, lògics i fins i tot admet rangs de llistes.

Stylus s'instal·la mitjançant el gestor de paquets npm amb l'ordre següent:

```
npm install stylus -g
```

Una vegada instal·lat, es pot utilitzar l'ordre `stylus` indicant el nom del fitxer amb el codi Stylus i el nom del fitxer CSS de destí. Per comprovar-ho, creeu un fitxer anomenat prova.styl amb el contingut següent:

```
color_principal= blue
color_vora = darken(color_principal, 20%)
color_enllac = lighten(color_principal, 20%)

div
  color: color_principal
  border: 1px solid color_vora

  a
    color:color_enllac
```

Fixeu-vos que, per assignar el valor a les variables, s'ha fet servir el signe igual (=) i el nom de les variables s'ha separat amb una barra per sota en lloc d'un guió (a Stylus no es pot fer servir el guió en els noms de variables).

Per generar el fitxer CSS escriviu a la línia d'ordres:

```
stylus prova.styl resultat.css
```

Es generarà el fitxer resultat.css a la mateixa carpeta amb el contingut següent:

```
div {
  color: #00f;
  border: 1px solid @00c;
}
div a {
```

API JavaScript per Stylus

Podeu trobar tota la documentació d'API JavasScript a l'enllaç següent: goo.gl/ZeM6Fu.

En lloc de fer servir la línia d'ordres, es pot crear un programa en Node.js per crear configuracions de preprocessament més complexes. Per comprovar-ho, creeu un fitxer anomenat processar-stylus.js, amb el contingut següent:

```
var fs = require("fs");
var stylus = require("stylus");
var entrada = fs.readFileSync('prova.styl', 'utf8');

stylus(entrada)
        .render(function(error, css) {
    if (!error) {
      // No s'han trobat errors, es desarà al disc
      fs.writeFile('resultat.css', css, function(err) {
        if (!err) {
          // Fitxer escrit al disc
          console.log("Preprocessat realitzat amb èxit");
        }
      });
    }
  }
);
```

En executar aquest programa, es preprocessa el fitxer prova.styl i el resultat es desa a "resultat.css". Per comprovar-ho, escriviu a la línia d'ordres:

```
node processar-stylus.js
```

En el cas de **Linux**, hi heu d'escriure:

```
nodejs processar-stylus.js
```

Cal destacar que Stylus ofereix més opcions que altres preprocessadors, també pel que fa a l'API de JavaScript, fet que permet crear programes més complexos per gestionar el preprocessament dels fitxers.

2.4. Compiladors JavaScript

Tot i que la idea d'un compilador per a un llenguatge interpretat com a JavaScript pot semblar estranya, es tracta d'una eina molt utilitzada.

Un dels problemes de JavaScript és que quan s'executa en el navegador d'usuari, no es pot controlar l'entorn d'execució. Pot tractar-se d'un navegador en un ordinador d'escriptori o d'un dispositiu mòbil, i no hi ha cap garantia que aquest navegador estigui actualitzat, ni tan sols que el seu fabricant hagi inclòs les característiques necessàries per al bon funcionament de la vostra aplicació.

S'ha de tenir en compte que el ritme d'adaptació dels navegadors a les noves versions de JavaScript és molt lent. Per exemple, ES5 va trigar 10 anys a ser completament admès per la majoria dels navegadors, és a dir, el mateix any que va ser llançada la següent versió.

admesa més àmpliament. En aquest àmbit, el compilador més utilitzat es Babel.js, que permet compilar programes desenvolupats en ES2015 i posteriors a ES5.

> JSDOC és un format de documentació per a JavaScript similar al que es pot trobar a altres llenguatges com Javadoc per a Java i PHPDoc per a PHP.

Un altre compilador de JavaScript (no gaire utilitzat) és el Closure Compiler (goo.gl/4AR0x5), desenvolupat per Google, que permet optimitzar el codi i, mitjançant els comentaris inclosos com a documentació en format JSDOC, inspeccionar determinats comportaments que en cas que no es complissin, mostrarien avisos durant la compilació. Per exemple, si s'anota una variable com si fos una constant i s'intenta canviar el valor més endavant, es mostra un avís durant la compilació.

> **Llistat de llenguatges que compilen a JavaScript**
>
> Podeu trobar el llistat en l'enllaç següent: goo.gl/A5sNah.

Per altra banda, al llarg del temps s'han creat múltiples llenguatges que són compilats a JavaScript. D'aquesta manera, poden ser desenvolupats en aquests llenguatges i ser utilitzats al navegador (recordeu que l'únic llenguatge que pot executar-se en aquest entorn és JavaScript). Entre aquests llenguatges hi ha CofeeScript, TypeScript i Dart.

Fora de l'entorn dels navegadors cal destacar el compilador Rhino, utilitzat per compilar JavaScript a Java, que s'utilitza incrustat en altres dispositius i permet programar-los utilitzant JavaScript.

2.4.1. ECMAScript 2015, ES6 i Babel

Quan es parla d'ES6 es fa referència a la sisena edició de JavaScript, tot i que el nom oficial de l'estàndard és **ECMAScript 2015** (i posteriors). És a dir, JavaScript és la implementació de l'estàndard ECMAScript.

La cinquena edició del llenguatge s'anomenava ES5 (correspon a JavaScript 5), però a la sisena edició van canviar el nom diverses vegades fins que finalment van escollir ECMAScript 2015 (abreviat com a ES2015) i van decidir ampliar el llenguatge anualment.

Cada any, el mes de juny, es publica una ampliació de l'especificació i en canvien el nom: ES2015 el juny de 2015, ES2016 el juny de 2016, ES2017 el juny de 2017, etc.

Quan es treballa amb alguna característica avançada del llenguatge, és important saber en quina versió s'ha afegit per poder determinar si està disponible o no, ja que algunes característiques només existeixen com a especificació i no es troben implementades en cap navegador ni compilador.

> **Noves característiques d'ECMAScript 2015**
>
> Podeu trobar una llista d'aquestes característiques a l'enllaç següent: es6-features.org.

DESPLIEGUE DE APLICACIONES WEB

Tot i que moltes de les característiques no es troben implementades als navegadors, hi ha alguns desenvolupadors que prefereixen utilitzar aquestes versions del llenguatge, ja que eventualment aquest és el llenguatge que cal utilitzar al web. A més, aquestes versions ofereixen moltes característiques interessants, com ara la creació de classes, la declaració de constants, el context de bloc, la importació de mòduls, l'encapsulació i la utilització de funcions fletxa o lambdes.

És més, molts dels entorns de treball més populars com AngularJS, React, Vue.js, etc., estan programats en ES2015 o TypeScript, ja que a l'hora de treballar en projectes grans és habitual haver de fer un processament dels fitxers i no costa gens afegir la compilació de ES2015 al flux de treball.

Compatibilitat de ES2015 amb Node.js

Es pot trobar informació detallada sobre la implementació de totes les característiques de ES2015 a Node.js a l'enllaç següent: node.green.

Cal destacar que Node.js inclou pràcticament la totalitat de l'especificació de ES2015, és a dir, els mòduls de Node.js estan desenvolupats en ES2015.

El compilador més utilitzat per compilar el codi de ES2015+ a ES5 és Babel (babeljs.io). Per instal·lar-lo mitjançant el gestor de paquets npm podeu utilitzar l'ordre següent:

```
npm install --save-dev babel-cli babel-preset-env
npm install --save-dev babel-preset-latest
```

Fixeu-vos que la instal·lació recomanada de Babel es fa localment, afegint-se a la llista de dependències del projecte (al fitxer package.json). Tingueu en compte que si treballeu amb ES2015+, el vostre projecte no funcionarà correctament en tots els navegadors si no es compila abans a ES5.

Un cop instal·lats aquests paquets, podeu començar a utilitzar Babel per compilar JavaScript.

Per comprovar que funciona correctament, creeu un fitxer anomenat hola-mon-es2015.js, amb el contingut següent:

```
let missatge = "Hola món!";
console.log(missatge);
```

A continuació compileu-lo, amb l'ordre següent:

```
babel hola-mon-es2015.js --presets=es2015
```

Com que no s'ha especificat un fitxer de sortida, mostrarà per pantalla la compilació, que serà semblant a aquesta:

```
"use strict";

var missatge = "Hola món!";
console.log(missatge);
```

la línia d'ordres, dins del fitxer package.json i al fitxer .babelrc.

Per afegir l'opció al fitxer .babelrc només heu d'editar-lo i afegir la propietat `presets` dintre de l'objecte amb les opcions de Babel. En cas que sigui buit, el contingut ha de ser el següent:

```
{
  "presets": ["latest"]
}
```

Un cop afegit al fitxer, l'exportació es fa automàticament utilitzant la predefinició més recent.

Per desar el fitxer en lloc de mostrar-lo per pantalla, s'ha de fer servir l'opció `-out-file` seguida del nom del fitxer de destí. Per exemple:

```
babel hola-mon-es2015.js --out-file hola-mon-compilat.js
```

El resultat és un fitxer anomenat hola-mon-compiled.js amb el mateix contingut que es mostra per pantalla: el programa compilat a ES5.

2.4.2. TypeScript

TypeScript (www.typescriptlang.org) és un llenguatge de programació desenvolupat i mantingut per Microsoft basat en l'especificació ES2015 al qual afegeix (opcionalment) tipificació estàtica, és a dir, es comprova que els tipus de variables i paràmetres siguin correctes durant la compilació.

Tot i que la tipificació estàtica és opcional, és molt recomanable utilitzar-la, ja que ajuda a prevenir errors que d'altra manera poden passar desapercebuts fàcilment.

Vegeu un exemple de funció tipada en TypeScript:

```
function multiplicar(a: number, b: number): number {
  return a * b;
}
```

Aquesta funció accepta dos paràmetres anomenats `a` i `b`, tots dos de tipus `number`, i el valor retornat és de tipus `number`.

Cal destacar que TypeScript interpreta JavaScript correctament, de manera que en una aplicació desenvolupada en aquest llenguatge pot utilitzar-se juntament amb JavaScript (incloses biblioteques) en el mateix fitxer.

TypeScript fa servir el seu propi compilador, que pot instal·lar-se utilitzant el gestor de paquets npm, escrivint la següent ordre a la línia d'ordres:

```
npm install typescript -g
```

Una vegada instal·lat, per compilar un fitxer només cal utilitzar l'ordre `tsc` seguida del nom del fitxer. Per comprovar-lo creu un fitxer anomenat hola-mon.ts, amb el contingut següent:

```
}

mostrarMissatge('Hola món!');
```

Executeu l'ordre següent:

```
tsc hola-mon.ts
```

Es crea un fitxer anomenat hola-mon.js al mateix directori amb el contingut següent:

```
function mostrarMissatge(missatge) {
    console.log(missatge);
}
mostrarMissatge('Hola món!');
```

Com veieu, s'ha eliminat l'anotació que obligava a passar una cadena de text com a paràmetre de la funció mostrarMissatge.

Per comprovar que el programa funciona correctament podeu executar-lo com un programa de Node.js, amb l'ordre següent:

```
node hola-mon.js
```

O la següent, si es tracta de **Linux**:

```
nodejs hola-mon.js
```

Si desitgeu comprovar el funcionament de la tipificació estàtica, podeu modificar la invocació de la funció al fitxer hola-mon.ts per la següent:

```
mostrarMissatge(42);
```

A continuació, intenteu compilar-lo de nou:

```
tsc hola-mon.ts
```

Aquest cop el resultat de la compilació dona un error similar al següent:

```
hola-mon.ts(5,17): error TS2345: Argument of type '42' is not assignable to paramet
```

És a dir, en aquest cas `42` no és assignable a un paràmetre de tipus `string`.

2.5. Eines d'automatització

Quan es desenvolupen aplicacions web és molt important optimitzar els fitxers generats, perquè es redueix el pes i el nombre de peticions HTTP que s'han de realitzar per carregar l'aplicació.

Els navegadors només processen entre 4 i 8 peticions HTTP al mateix temps, i el temps mitjà per obtenir la resposta a cada petició és al voltant dels 100 ms en línies d'alta veloci-

guts d'una pàgina web es troba en el nombre de peticions i no pas en el pes dels continguts.

Aquest és un dels problemes més greus d'utilitzar sistemes gestors de continguts (CMS) com WordPress o Joomla, ja que acostumen a realitzar 60 peticions (superant-ne el centenar en alguns llocs) només per descarregar fitxers JS i CSS. Aquest nombre de peticions es podria reduir a 2 o 3 fitxers en la majoria dels casos.

Convé destacar que, molt sovint, quan es parla de desenvolupar un lloc o una aplicació web, s'acostuma a pensar en un nombre molt petit de fitxers: un parell de fitxers CSS, un fitxer amb el codi JavaScript, un fitxer HTML i algunes imatges decoratives. Però avui dia això està molt lluny de la realitat, atesa la complexitat de les aplicacions que s'han de desenvolupar, les dependències necessàries (compiladors de JavaScript, preprocessadors de CSS, gestió de fitxer, etc.) i el nombre i tipus de fitxers diferents amb els quals s'ha de treballar.

A més, s'ha de tenir en compte que molts d'aquests fitxers no formen part del desplegament i es generen molts fitxers intermedis. Per exemple, en el cas del preprocessament de fitxer Less es podrien fer els passos següents:

1. Es preprocessen els fitxers Less i es generen els fitxers CSS.
2. Els fitxers CSS es concatenen per formar un únic fitxer.
3. Es minimitza el fitxer final.

Cal ressaltar que, si canvia l'ordre de concatenació dels fitxers, també pot canviar el resultat final (els estils CSS s'apliquen en cascada), així que és necessari que el procés es realitzi sempre en el mateix ordre. Per altra banda, al desplegament només cal afegir-hi el fitxer final minimitzat; la resta de fitxers CSS es poden descartar, i els fitxers Less no han de sortir de l'entorn de desplegament.

Fixeu-vos que aquest cas s'aplica als fitxers Less propis, però segurament haureu d'incloure en aquest procés els fitxers Less i CSS inclosos en altres biblioteques. Per exemple, si el projecte inclou Bootstrap i FontAwesome, aquests també s'han d'afegir al vostre fitxer CSS:

Variables Less de Bootstrap

Podeu trobar totes les variables i els seus valors per defecte a l'enllaç següent:goo.gl/mrHCG8.

1. En el cas de Bootstrap, s'ha de descarregar la versió Less i importar-la al vostre fitxer Less principal per poder modificar directament les variables i processar-lo juntament amb els vostres fitxers.
2. El fitxer CSS de FontAwesome es pot afegir directament al començament de la concatenació de fitxers CSS.

Continuant amb el mateix exemple, el següent pas és copiar els fitxers d'imatges i fonts necessàries a la carpeta de desplegament, tenint en compte que aquestes imatges i fonts poden canviar al llarg del desenvolupament, ja sigui perquè s'han modificat algunes imatges o perquè s'ha actualitzat la biblioteca que les incloïa.

Després, s'hi han d'afegir els fitxers JavaScript, seguint un procés similar al dels fitxers Less:

1. Biblioteques principals utilitzades per l'aplicació (per exemple, jQuery, AngularJS, Vue.js, etc.)
2. Codi d'altres biblioteques i connectors necessaris
3. Codi propi

3. Minimitzar el fitxer resultant.

En alguns casos, cal generar els fitxers optimitzats amb diferents continguts (és a dir, repetir tots aquests processos amb diferents configuracions).

Per exemple:

- Per obtenir un fitxer que només inclogui el codi de la secció d'administració del lloc i que no serà carregat per usuaris no autoritzats.
- Per generar un fitxer amb els continguts comuns a totes les pàgines del lloc (que després pot ser concatenat a altres fitxers més especialitzats).
- Per generar fitxers únics per a diferents tipus de continguts de l'aplicació: uns fitxers per a la pàgina principal, un altre per a la pàgina de cerca, un altre per al panell d'administració, etc.

Com es pot apreciar, aquest procés és molt feixuc, pot consumir molt de temps i és molt fàcil de cometre errors. A més, s'ha de repetir cada vegada que es fan canvis al codi, cosa que fa inviable realitzar totes aquestes operacions manualment. Durant molts anys, s'han fet servir eines per automatitzar els processos en altres llenguatges: Ant, Graddle o Maven.

> Els programes que permeten automatitzar el flux de treball a JavaScript també són coneguts com a executors de tasques (*task runners*) o automatitzador de tasques.

A JavaScript, s'ha aprofitat la potència de Node.js per crear eines pròpies que s'integren perfectament amb el flux de treball, ja que es programen i configuren amb JavaScript i permeten realitzar tot tipus de tasques. Les dues eines més populars per portar a terme aquesta automatització són: Grunt i Gulp.

La decisió de fer servir qualsevol d'aquestes eines generalment recau en la decisió del vostre equip de desenvolupament o de l'entorn de programació escollit. Per exemple, les aplicacions desenvolupades amb Laravel feien servir Gulp com a automatitzador, però a les versions més recents ha estat reemplaçat per Webpack (un gestor de paquets).

2.5.1. Gulp

Gulp (gulpjs.com) és una eina basada en Node.js que permet automatitzar el processament de tasques. La configuració es realitza mitjançant un fitxer que conté el codi JavaScript, en el qual s'afegeixen els mòduls necessaris i la configuració de les diferents tasques.

Per instal·lar Gulp mitjançant el gestor de paquets npm, s'han d'introduir les ordres següents:

```
npm install gulp-cli -g
npm install gulp -D
```

DESPLIEGUE DE APLICACIONES WEB

Gulp requereix un fitxer anomenat gulpfile.js. Aquest fitxer ha de contenir un programa de Node.js que serà utilitzat pel client de Gulp per portar a terme les tasques, a part de ser el responsable d'importar els connectors necessaris. Cal tenir en compte que els connectors s'han d'instal·lar per separat. Així, si voleu preprocessar fitxers Less i minimitzar-los, cal instal·lar primerament els connectors "gulp-less", "gulp-csso" i "gulp-concat", tal com podeu veure a continuació:

```
npm install gulp-less gulp-csso gulp-concat -g
```

Un cop instal·lats els connectors necessaris, es pot crear una tasca que processi tots els fitxers Less d'una carpeta, que concateni el fitxer CSS resultant i el minimitzi. El contingut de gulpfile.js seria similar al següent:

```
var gulp = require('gulp');
var less = require('gulp-less');
var minifyCSS = require('gulp-csso');
var concat = require('gulp-concat');

gulp.task('css', function(){
  return gulp.src('less/*.less')
    .pipe(less())
    .pipe(concat('estils.css'))
    .pipe(minifyCSS())
    .pipe(gulp.dest('build/css'))
});

gulp.task('default', [ 'css' ]);
```

Com podeu apreciar, Gulp fa servir un sistema de canonades: la sortida d'una funció és l'entrada de la següent. Així doncs, la sortida de la funció `src` és l'entrada de la funció `less`, la seva sortida, al seu torn, és l'entrada de la funció `concat`, i així successivament.

Per comprovar que funciona correctament, creeu un directori anomenat *less*, amb dos fitxers anomenats full1.less i full2.less.

Afegiu al fitxer full1.less el codi següent:

```
@mida-font: 15px;
@color: #ff0000;

p {
  font-size: @mida-font;
  color: @color;
}
```

I al fitxer full2.less, afegiu-hi el següent:

```
@color-principal: blue;
@color-vora: darken(@color-principal, 20%);
@color-enllac: lighten(@color-principal, 20%);

div {
  color: @color-principal;
  border: 1px solid @color-vora;

  a {
    color:@color-enllac;
  }
}
```

```
gulp
```

La sortida serà similar a la següent:

```
[10:33:06] Using gulpfile ~/gulpfile.js
[10:33:06] Starting 'css'...
[10:33:06] Finished 'css' after 84 ms
[10:33:06] Starting 'default'...
[10:33:06] Finished 'default' after 23 µs
```

Quan s'executa Gulp sense cap paràmetre, s'executa la tasca per defecte (`default`), que en aquest cas només inclou la tasca `css` i, per consegüent, el resultat és l'execució d'aquesta tasca.

El resultat és la creació d'un fitxer anomenat estils.css dintre del directori *build/css*. Si algun d'aquests directoris no existeix, es crea automàticament. El contingut d'aquest fitxer és el següent: preprocessat, concatenat i minimitzat.

La sortida serà similar a la següent:

```
p{font-size:15px;color:red}div{color:#00f;border:1px solid #009}div a{color:#66f}
```

Si s'executa Gulp indicant el nom de la tasca `css`, el resultat és el mateix:

```
gulp css
```

Fixeu-vos que en aquest cas la sortida només indica que s'ha iniciat la tasca `css`:

```
[10:41:28] Using gulpfile ~/gulpfile.js
[10:41:28] Starting 'css'...
[10:41:28] Finished 'css' after 89 ms
```

Per afegir noves tasques al fitxer gulpfile.js, només cal invocar la funció `task` del mòdul `gulp` passant com a primer argument el nom de la tasca i com a segon argument una funció que conté el codi necessari per executar-la. Per exemple, per afegir una tasca que mostri un missatge per la pantalla només cal afegir al fitxer gulpfile.js el següent codi:

```
gulp.task('hola', function() {
  console.log("Hola món!");
});
```

Seguidament podreu executar la tasca "hola" amb l'ordre següent:

```
gulp hola
```

I el resultat mostrat per la pantalla serà el següent:

```
[10:49:08] Using gulpfile ~/gulpfile.js
[10:49:08] Starting 'hola'...
Hola món!
[10:49:08] Finished 'hola' after 477 µs
```

següent:

```
gulp.task('default', [ 'css', 'hola' ]);
```

Si ara executeu l'ordre `gulp` sense cap paràmetre, el resultat serà similar al següent:

```
[10:52:23] Using gulpfile ~/gulpfile.js
[10:52:23] Starting 'css'...
[10:52:23] Starting 'hola'...
Hola món!
[10:52:23] Finished 'hola' after 620 µs
[10:52:24] Finished 'css' after 92 ms
[10:52:24] Starting 'default'...
[10:52:24] Finished 'default' after 31 µs
```

Convé destacar un detall molt important: les tasques s'inicien paral·lelament, és a dir, totes les tasques s'inicien alhora i la segona tasca finalitza abans que la primera, perquè, com que és molt simple, requereix menys temps de processament. És un avantatge respecte a Grunt, ja que aquest sistema és més ràpid que executar les tasques seqüencialment.

Per altra banda, s'ha de tenir molt clar que cada tasca ha de ser independent de la resta per evitar conflictes. Per exemple, si teniu una tasca que concatena els fitxers CSS i una altra que els preprocessa, la tasca de preprocessament començarà abans que els fitxers siguin concatenats i el resultat serà incorrecte.

Val la pena recordar que gulpfile.js és un fitxer de JavaScript executat sobre Node.js i, consegüentment, podeu utilitzar qualsevol dels seus mòduls segons les necessitats. Per exemple, seria possible utilitzar el mòdul FileSystem per desar la data i hora de l'última compilació, iniciar una bateria de proves unitàries o generar la documentació actualitzada.

2.5.2. Grunt

Grunt (gruntjs.com) és una de les primeres eines per automatitzar el flux de treball que es va desenvolupar sobre Node.js. Actualment té més de 5.000 connectors, fet que fa que ja existeixi el connector adequat per a, pràcticament, qualsevol tasca.

La principal diferència amb Gulp és que en lloc d'implementar les tasques mitjançant codi, aquest està basat en configuracions (tot i que el fitxer de configuració també és un fitxer en JavaScript).

Per instal·lar Grunt mitjançant el gestor de paquets npm escriviu les ordres següents:

```
npm install grunt-cli -g
npm install grunt -D
```

Com es pot apreciar, el client de Grunt s'instal·la globalment, mentre que el mòdul Grunt s'instal·la localment i s'afegeix com a dependència de desenvolupament al fitxer package.json.

A continuació, s'han d'instal·lar els connectors necessaris. Per exemple, si es vol crear una tasca per preprocessar fitxers Less, concatenar-los i minimitzar-los, cal instal·lar els con-

```
npm install grunt-contrib-cssmin --save-dev
npm install grunt-contrib-less --save-dev
```

Un cop instal·lats els connectors, podeu crear el fitxer Gruntfile.js amb el contingut següent:

```
module.exports = function(grunt) {

  grunt.initConfig({
    less: {
        dist: {
          src: ['less/*.less'],
          dest: 'build/css/estils.css'
        }
    },

    cssmin: {
      target: {
        files: [{
          expand: true,
          cwd: 'build/css',
          src: ['*.css', '!*.min.css'],
          dest: 'build/css',
          ext: '.min.css'
        }]
      }
    }


  });

  grunt.loadNpmTasks('grunt-contrib-cssmin');
  grunt.loadNpmTasks('grunt-contrib-less');

  grunt.registerTask('default', ['less', 'cssmin']);
};
```

Cal tenir en compte que tot el codi de la configuració es troba dintre de la funció exportada (`module.exports = function (grunt)`). Dintre d'aquesta funció es poden distingir 3 seccions:

- **Inicialització** (`grunt.initConfig`): conté la configuració de les tasques (que corresponen als connectors). En aquest cas, `less` i `cssmin`. Aquí s'indiquen els fitxers d'origen i destí de cada connector, entre altres opcions.
- **Càrrega dels connectors** (`grunt.loadNpmTasks`): carrega els connectors.
- **Enregistrament de la tasca** (`grunt.registerTask`): s'enregistra la tasca amb el nom i la llista de tasques i l'ordre en què s'han d'executar.

Un cop creat el fitxer Gruntile.js, per poder comprovar que heu realitzat les tasques correctament, heu de crear un directori anomenat *less* i dos fitxers anomenats full1.less i full2.less (si no els heu creat anteriorment).

Afegiu al fitxer full1.less el codi següent:

```
@mida-font: 15px;
@color: #ff0000;

p {
  font-size: @mida-font;
  color: @color;
}
```

I al fitxer full2.less, afegiu-hi el següent:

```
@color-enllac: lighten(@color-principal, 20%);

div {
  color: @color-principal;
  border: 1px solid @color-vora;

  a {
    color:@color-enllac;
  }
}
```

Arribats a aquest punt només cal executar l'ordre `grunt` a la línia d'ordres per comprovar que es realitza el processament correctament. La sortida serà similar a la següent:

```
Running "less:dist" (less) task
>> 1 stylesheet created.

Running "cssmin:target" (cssmin) task
>> 1 file created. 124 B → 81 B

Done.
```

Una vegada finalitzat, s'haurà creat un directori anomenat *build/css* que contindrà dos fitxers: estils.css i estils.min.css, amb la versió minimitzada i el contingut següent:

```
p{font-size:15px;color:red}div{color:#00f;border:1px solid #009}div a{color:#66f}
```

Fixeu-vos que, al contrari que en el cas de Gulp, el processament de les tasques és seqüencial, és a dir, fins que no acaba la tasca `less` no s'inicia la tasca `cssmin`.