HTML5 & CSS3

La guida **completa** per il Web Design

Un libro per imparare velocemente la programmazione e lo sviluppo lato client, con cenni di Javascript, jQuery, JSON e XML

TONY CHAN

HTML5 & CSS3

la guida **completa** per il *Web Design*

Un libro per imparare velocemente la programmazione e lo sviluppo lato client, con cenni di Javascript, jQuery, JSON e XML

Di **Tony Chan**

riparazioni o perdite monetarie dovute alle informazioni contenute in questo libro.

Avviso legale:

Questo libro è protetto da copyright, ed è solo per uso personale. Non è possibile modificare, distribuire, vendere, utilizzare, citare o parafrasarne il contenuto senza il consenso diretto dell'autore o dell'editore.

Avviso di esclusione di responsabilità:

Si prega di notare che il contenuto di questo libro è solo a scopo educativo e d'intrattenimento. È stato compiuto ogni sforzo per presentare informazioni accurate, aggiornate, affidabili e complete. Nessuna garanzia di alcun tipo è dichiarata o implicita. I lettori riconoscono che l'autore non s'impegna a fornire consulenza legale, finanziaria, medica o professionale. Il contenuto di questo libro è in parte derivato da varie fonti. Consultare un professionista autorizzato prima di tentare qualsiasi tecnica descritta.

Leggendo questo testo, il lettore accetta che in nessun caso l'autore sarà ritenuto responsabile per eventuali perdite, dirette o indirette, subite a seguito dell'uso delle informazioni contenute in questo documento, incluse omissioni o inesattezze.

 # L'autore

Tony Chan è nato a South Gate, in California, Stati Uniti, da genitori immigrati cinesi. A 11 anni scrive il suo primo programma software in Basic, a 17 insegna agli amici programmazione e a 20 si laurea in informatica.

Recentemente ha partecipato come consulente nella stesura di alcuni testi sull'intelligenza artificiale e sul linguaggio macchina. Ha poi deciso di pubblicare i suoi libri sulla programmazione sia negli Stati Uniti, dove vive e lavora, ma anche in Italia, che ama e che visita spesso. Un giorno, un pasticcere a Roma gli ha dedicato la "*Torta touch*", che ricordava la forma di uno smartphone e bisognava anche "sbloccarla" toccandola per poterla mangiare.
Di Tony si dice che in inverno si serva di un rig per minare criptovalute come riscaldamento del suo soggiorno…

Indice dei contenuti

 # Introduzione

Sono passati tanti anni e diverse battaglie dalla standardizzazione di HTML5 da parte del World Wide Web Consortium, meglio conosciuto come W3C, ossia l'organizzazione non governativa che si occupa proprio della definizione di protocolli per il Web. Parliamo di circa dieci anni, che in internet sono un'era geologica. In questo periodo HTML5 e CSS3 hanno invaso la rete e, secondo le statistiche di w3techs.com, oggi il 91% dei siti utilizza l'evoluzione di HTML e CSS. La rivoluzione immaginata qualche anno fa si è concretizzata ed entro breve ce ne sarà un'altra, legata al Web 3.0 o alla transizione verso di esso. In questo passaggio avremo anche l'ulteriore miglioramento di HTML alla sua sesta edizione.
Questo significa che dovremo avere ben salda la conoscenza delle basi di HTML5, CSS3 e delle loro successive implementazioni (es. HTML 5.3) le quali hanno gettato le basi per questa generazione di applicazioni Web. Ci hanno consentito la creazione di siti più semplici da sviluppare, più facili da mantenere e da utilizzare, con nuovi elementi per definirne la struttura e incorporarne i contenuti, senza dover ricorrere a markup o plug-in aggiuntivi. CSS3 ci ha regalato selettori avanzati, miglioramenti grafici e nel supporto dei caratteri i quali hanno reso i nostri siti visivamente più accattivanti senza utilizzare tecniche di sostituzione delle immagini o dei caratteri, codice JavaScript o strumenti grafici, mentre Il supporto offline ci ha consentito di iniziare a creare applicazioni funzionanti senza la connessione ad Internet.

Per la maggior parte del libro, ci concentreremo sulle specifiche di HTML5, CSS3 e su come utilizzare le varie tecniche con uno stratagemma: immagineremo di dover creare e implementare un ambizioso sito Web di un ristorante. Il proprietario e gestore, il sig. Mario, ogni tanto ci chiederà anche di aggiungere alcune funzionalità e noi lo faremo proprio sfruttando HTML5 e CSS3.
Questo libro è rivolto a chi ha già delle basi minime di conoscenza, tuttavia nel capitolo zero abbiamo un veloce corso accelerato proprio sulle fondamenta di HTML e una breve introduzione a Javascript e jQuery, necessari per una migliore

comprensione e utilizzo di alcune tecniche che studieremo. In pratica tutto ciò permetterà a chiunque di iniziare a seguire i vari capitoli fino alla fine. Anche perché l'impostazione del libro ci obbligherà a scrivere del codice e provare ciò che è illustrato, cercando di semplificare al massimo anche concetti più complessi, proprio per fare in modo di immergere e coinvolgere il lettore in un progetto concreto, che potremo quindi provare sul campo, essenzialmente come in un corso intensivo, o meglio, immersivo.

Ogni capitolo di questo libro si concentra su un gruppo specifico di problemi che possiamo risolvere con HTML5 e CSS3 ed è raggruppato per argomento. Essenzialmente, piuttosto che affrontare i concetti separatamente tra i due linguaggi, abbiamo preferito unirli per affrontare un percorso di studio stimolante e interessante.

Inizieremo con una breve panoramica su HTML5 e CSS3 e daremo un'occhiata ad alcuni dei nuovi tag strutturali da utilizzare per descrivere meglio il contenuto della nostra pagina Web. Quindi lavoreremo con i moduli (form) con i quali avremo la possibilità di scoprire alcune utili funzionalità come *messa a fuoco automatica* (autofocus) e *segnaposto* (placeholder). Proseguendo, potremo giocare con i nuovi selettori di CSS3 e così apprendere come applicare gli stili agli elementi senza aggiungere markup extra (editing o formattazione) al nostro contenuto.

Quindi esploreremo il supporto audio e video di HTML5 e impareremo come utilizzare la tela (*canvas*) per disegnare forme. Vedremo anche come giocare con le ombre, i gradienti e le trasformazioni di CSS3, nonché come imparare a lavorare con i caratteri.

Nell'ultima parte del libro ci dedicheremo a quella sfumatura di HTML5 come l'archiviazione Web, i database con *IndexedDB* e il supporto offline per creare applicazioni lato client. Useremo i Web Sockets per parlare con un semplice servizio di chat e vedremo come HTML5 rende possibile inviare messaggi e dati tra domini. Avremo anche la possibilità di giocare con l'API di geolocalizzazione e imparare a manipolare la cronologia del browser. Concluderemo con le altre novità e con tutto ciò che non siamo riusciti a vedere nel corso dei vari capitoli.

Andando avanti noterete che ogni tanto mantengo i termini, la sintassi di alcuni concetti o della grammatica dell'HTML in inglese, a volte traducendoli tra parentesi (ove possibile). Questo perché praticamente tutti i linguaggi di programmazione sono in inglese, ed è corretto, nella maggior parte dei casi, evitare la traduzione, dato che ben pochi discuteranno di un listato o di un'istruzione in italiano, è dunque un bene abituarsi e sintonizzarsi fin dall'inizio.
Adesso faremo una breve panoramica su alcuni software, gratuiti e assolutamente non invasivi, che ci aiuteranno nel percorso di studio.

Strumenti (Tools) di lavoro

Sebbene, in linea teorica, per scrivere codice HTML sia sufficiente disporre di un editor di testi (come il *blocco note*), quando si comincia a scrivere dei listati più complessi o importanti, gli strumenti giusti possono fare davvero la differenza, il consiglio è quello di usufruire fin da subito di un buon software.
Senza la pretesa di essere esaustivi, nelle pagine che seguono proporremo una panoramica degli strumenti più diffusi:

Editor di testo

Una prima alternativa al Blocco Note è costituita degli editor di testo "più evoluti". Ne ricordiamo tre: Notepad ++, Atom e Visual Studio.
Il primo di questi, **Notepad ++** è un editor gratuito che può essere scaricato dal sito https://notepad-plus-plus.org/.
La sua interfaccia è semplice e di facile utilizzo. Dispone di una serie di caratteristiche davvero utili: evidenziazione della sintassi, raggruppamento di porzioni omogenee di codice (Syntax Folding) in modo da poter nascondere o visualizzare porzioni di un documento lungo, evidenziazione della sintassi e Syntax Folding personalizzato dall'utente, evidenziazione delle parentesi, ricerca/sostituisci mediante espressioni regolari (Perl Compatible Regular Expression), autocompletamento della sintassi, segnalibri, visualizzazione a schede, visualizzazione di documenti affiancati per il confronto.
Atom è un editor gratuito scaricabile dal sito https://atom.io/ disponibile per più piattaforme (OS X, Windows e Linux). Può

essere completato con diversi pacchetti open source e dispone di supporto al sistema di controllo di versione Git.
Fra i punti di forza di Atom ci sono: autocompletamento, evidenziazione della sintassi, funzionalità di ricerca e sostituzione fra più file, possibilità di aprirne diversi in pannelli affiancati per poterli confrontare.
Visual Studio Code è l'editor che abbiamo usato per scrivere gli esempi di questo libro. È un editor sviluppato da Microsoft per più piattaforme (OS X, Windows e Linux). Si tratta di uno strumento gratuito scaricabile dalla pagina https://code.visualstudio.com/.
Dispone già di Git ed è integrabile con ulteriori pacchetti. Fra i suoi punti di forza ci sono: autocompletamento, evidenziazione della sintassi, funzionalità di ricerca e sostituzione, possibilità di impostare breakpoint, lavorare direttamente con file e cartelle senza la necessità di creare progetti.
Ci servirà anche un *linter*, il quale è un programma che in genere si integra con un editor di codice e permette di evidenziare gli errori di sintassi o in generale di scrittura del codice. Attualmente, uno dei *linter* per HTML più diffusi è HTML-Lint (https://html-lint.com/), per il CSS abbiamo Stylelint (https://stylelint.io/).
I tre software appena segnalati andranno benissimo anche per testare quei listati dove avremo insieme HTML e Javascript, PHP o altri. In questi casi sicuramente solo il blocco note non sarà sufficiente.
AppStore: oltre ai software elencati, in tutti i dispositivi aggiornati con Windows, MacOS, iOS e Android, negli *appstore* dedicati esistono numerosi editor gratuiti anche in prova, più o meno validi, comodi e veloci, da utilizzare per esercitarsi e studiare. Vale la pena darci un'occhiata per trovare quello che fa per noi.

Server virtuale

Per i nostri esperimenti qualche volta avremo bisogno di simulare un server, l'alternativa è quella di pagare un software professionale. Comunque, per iniziare andrà benissimo utilizzare un servizio gratuito e open source come quello offerto da **XAMPP** (https://www.apachefriends.org/it/index.html).

Sarà sicuramente un'ottima alternativa all'installazione di Apache e all'eventuale aggiunta di MySQL e PHP.

Una panoramica su HTML5 e CSS3

Molte delle nuove funzionalità dell'HTML sono incentrate sulla creazione di una piattaforma migliore per le applicazioni basate sul Web. Da tag più descrittivi, animazioni e supporto multimediale migliorato, e molto altro ancora.

Ogni versione di HTML introduce un nuovo markup, cioè quell'insieme di regole funzionali alla rappresentazione o all'impaginazione di un testo, ma mai prima della quarta versione ci sono state così tante nuove aggiunte che si riferiscono direttamente alla descrizione del contenuto. Impareremo gli elementi per definire intestazioni, piè di pagina, sezioni di navigazione, barre laterali e articoli nel capitolo 1, nuovi tag e attributi strutturali. Impareremo anche i contatori, le barre di avanzamento e come gli attributi dei dati personalizzati possono aiutarci a eseguire il markup dei dati.

L'interfaccia utente è una parte così importante delle applicazioni Web e infatti tratteremo ampliamente questo argomento in vari capitoli. Lo stesso vale per i miglioramenti grafici, infatti CSS3 ci consentirà di aggiungere ombre e sfumature agli elementi senza ricorrere a immagini di sfondo o markup extra. Inoltre, potremo utilizzare le trasformazioni per arrotondare gli angoli o inclinare e ruotare gli elementi. Vedremo come funzionano tutte queste cose nel capitolo 4.

I browser Web ci impediscono di utilizzare gli script su un dominio per influenzare o interagire con gli script di un altro dominio. Questa restrizione protegge gli utenti finali dal *cross-site scripting*, che è stato utilizzato per eseguire malware o altra robaccia ai visitatori ignari del sito.

Tuttavia, questo impedisce a tutti gli script di funzionare, anche quando li scriviamo noi stessi e sappiamo di poterci fidare del contenuto. HTML5 include una soluzione alternativa sicura e semplice da implementare. Vedremo come farlo funzionare in *Talking Across Domains*.

HTML5 offre supporto per i *Web Sockets*, che forniscono una connessione permanente a un server. Invece di interrogare costantemente un back-end per gli aggiornamenti sullo stato di avanzamento, la nostra pagina Web può iscriversi a un socket e il back-end può inviare notifiche ai nostri utenti. Inoltre

giocheremo un po' in Chat sempre con i Web Sockets.
Tendiamo a pensare a HTML5 come a una tecnologia Web, ma con l'aggiunta delle API *Web Storage* e *IndexedDB*, possiamo creare applicazioni nel browser in grado di mantenere i dati interamente sul computer del client. Vedremo come utilizzare queste API nel capitolo 6.
HTML5 permette migliori controlli dell'interfaccia utente. In precedenza, solitamente ci si rivolgeva a JavaScript e CSS per costruire cursori, selettori di date del calendario o selettori di colori. Questi sono tutti definiti come elementi reali in HTML5, proprio come i menù a discesa, le caselle di controllo e pulsanti di opzione. Impareremo come usarli nel capitolo 2, con la creazione di moduli Web intuitivi. Questi argomenti sono basilari, soprattutto se stiamo sviluppando applicazioni basate sul Web, cioè lo scopo principale di questo libro. Oltre a una migliore usabilità senza fare affidamento alle librerie JavaScript, c'è un altro vantaggio: una migliore accessibilità per tutti, ad esempio anche per i lettori di schermo. La navigazione di un sito, è molto più facile da trovare se puoi cercare il tag *nav* invece di un *div* specifico o di un elenco non ordinato. Piè di pagina, barre laterali e altri contenuti possono essere facilmente riordinati o saltati del tutto. L'analisi delle pagine in generale diventa molto meno complicata, il che può portare a esperienze migliori per le persone che si affidano alle tecnologie assistive. Nel capitolo 3, parleremo anche di miglioramento dell'accessibilità, impareremo come utilizzare questi nuovi attributi.
Infine, parlando di CSS3 vedremo che ha selettori che ci consentono di identificare righe pari e dispari nelle tabelle, tutte le caselle di controllo selezionate o anche l'ultimo paragrafo di un gruppo. Potremo ottenere di più con meno codice e meno markup. Questo rende anche molto più facile dare uno stile all'HTML che non possiamo modificare. Sempre nel capitolo 3, vedremo come migliorare le interfacce utente con CSS3, vedremo come utilizzare questi selettori in modo efficace.
Alla fine di ogni capitolo oltre ad avere un piccolo riassunto, troveremo degli esercizi utili a ripassare gli argomenti incontrati.

Tag HTML deprecati

Quando avremo a che fare con del codice non proprio aggiornato dovremo sempre tenere a mente che HTML5 ha introdotto molti nuovi elementi, ma soprattutto le specifiche ne deprecano alcuni comuni che, come detto, potremmo trovare in svariate pagine Web.

Innanzitutto, diversi elementi di presentazione sono scomparsi. Se li trovassimo nel nostro codice dovremmo sbarazzarcene subito e sostituirli con quelli più aggiornati. Eccone la lista:

- `basefont`
- `big`
- `center`
- `font`
- `frame`
- `frameset`
- `noframes`
- `s`
- `strike`
- `tt`
- `u`

Alcuni di questi tag sono perlopiù dimenticati, ma diversi siti obsoleti creati con editor ancora più vecchi contengono ancora alcuni di questi caratteri e tag. A parte gli elementi di presentazione, il supporto per le cornici è stato da tempo rimosso e quindi nella lista sono presenti anche loro.

Da segnalare che alcuni elementi sono stati sostituiti:

- `acronym` viene sostituito da `abbr`.
- `applet` viene sostituito da `object`.
- `dir` viene sostituito da `ul`.

Oltre agli elementi deprecati, ci sono molti attributi che non sono più validi. Questi includono attributi di presentazione

come i seguenti:

- `align`
- `bgcolor`
- `cellpadding`, `cellspacing`, e `border` di `table`
- `height` e `width`
- `hspace` e `vspace`
- `link`, `vlink`, `alink`, l'attributo `text` del tag `body`
- `profile` del tag `head`
- `scrolling` dell'elemento `iframe`
- `target` utlizzato per i link
- `valign`

Impareremo a nostre spese che, dopo più di dieci anni dall'introduzione di HTML5, ancora si trovano questi relitti in giro per il Web.
Adesso partiremo con un doveroso accenno sui fondamentali di HTML e su altri concetti utili al percorso di studio.

 # 0 - Le basi

Le basi dell'HTML

La sequenza degli eventi

Cenni di jQuery

Da questo punto in poi entriamo un po' di più nel concreto. Parleremo dell'HTML, acronimo di HyperText Markup Language, tradotto: Linguaggio a Marcatori per Ipertesti, uno degli strumenti di sviluppo per il Web più popolare in assoluto. Nel resto del libro parleremo ampliamente di HTML5 e CSS3, mentre, in questo capitolo ci occuperemo delle basi dell'HTML e di alcuni cenni su Javascript e jQuery (una delle librerie più conosciute di Javascript). Tutto ciò ci servirà per affrontare diversi temi più avanti, e soprattutto, occorre immediatamente entrare nell'ordine delle idee che HTML lavora benissimo in sinergia con altri linguaggi. Inoltre, come già sappiamo, è usato per la creazione di siti internet, quindi è evidente che abbiamo bisogno di interagire con gli altri strumenti di sviluppo per il Web, e JavaScript è uno di questi. Tuttavia la sezione relativa a jQuery può essere tranquillamente saltata ed eventualmente ripassata in seguito poiché potrebbe risultare un po' ostica. Il consiglio è di provare comunque, ne ricaveremo sicuramente dei vantaggi nel prosieguo dello studio. Andiamo avanti, ci sono alcune cose che dovremo conoscere, come ad esempio la posizione di un eventuale codice ospite nel documento HTML, sapere che può posizionarsi sia internamente sia esternamente in un file separato che sarà indicato nel nostro listato. Ora vediamo la struttura standard dell'HTML, poi proveremo a lavorare con un mini listato Javascript:

```
    <html>
   <head>
</head>
   <body>
</body>
    </html>
```

Se volessimo inserire del codice *interno*, aggiungeremo i tag *<script>* e *</script>* e posizioneremo il codice tra questi due, in questo modo il browser potrà distinguere, ad esempio JavaScript, dal resto del codice HMTL o CSS nella pagina internet.

Essendoci ovviamente diversi linguaggi di programmazione dal lato client (come ad esempio VBScript, PHP) è sempre meglio specificare quale intenderemo utilizzare, lo faremo tra i tag <script> in questa maniera:

```
   <script type="text/javascript">
// Il nostro primo listato Javascript e HTML
   </script>
```

Possiamo inserire il vostro codice sia all'interno del tag <head> che all'interno del tag <body>.

Proviamo con un programmino semplice, creiamo un file HTML con questo codice anche usando semplicemente il notepad, come già accennato inizialmente:

```
   <html>
<head>
  <title>Il mio grandioso programma interno</title>
  <script type="text/javascript">
  var data = new Date()
  alert(data)
  </script>
</head>
<body>
   </body>
   </html>
```

Se apriamo questo file .html usando il nostro browser, potremmo vedere una finestra che mostra la data corrente.

Come detto, possiamo inserire il nostro codice JavaScript esternamente in un file separato, e quindi aggiungere un link al file dal documento HTML. Se vogliamo mantenere lo stesso codice di JavaScript dell'esempio precedente esternamente, allora copiamo le seguenti linee di codice in un nuovo file:

```
   var data = new Date()
alert(data)
```

Salviamo il file come *datadioggi.js* Ora, copiamo le seguenti linee di codice in un nuovo file:

```
   <html>
   <head>
   <title>Il mio grandioso programma esterno</title>
   <script type="text/javascript"
   src="datadioggi.js">
```

```
</head>
<body>
</body>
</html>
```

Salviamo questo file come *datadioggi.html*. Quindi, assicuriamoci che i file *datadioggi.html* e *datadioggi.js* siano posizionati nella stessa cartella. Altrimenti quando impostiamo il valore dell'attributo *src* nel tag <script> dovremmo specificare l'intero percorso del file .js.

Ovviamente starà a noi decidere se utilizzare il codice esternamente o internamente in base alla lunghezza del listato o anche alla quantità di pagine che lo utilizzano. In quest'ultimo caso, ad esempio, se avessimo trenta pagine con la medesima funzione sarà sicuramente una buona idea quella di lasciare il codice fuori in un file .js, senza ripeterlo per trenta volte. E' semplicemente una questione di praticità, pulizia e ordine.

ALCUNI CODICI HTML DA SAPERE:

<html> </html>	Inizio e fine del listato, <*linguaggio*>.
<head> </head>	Informazioni descrittive.
<title> </title>	Titolo della pagina.
<body> </body>	Contenuto della pagina (corpo).
	Contenitore per varie parti di un sito suddivise per *id,* in disuso con HTML5.

PROPRIETA' DELLA PAGINA: SFONDI E COLORI

Immagine di sfondo <BODY BACKGROUND="URL">

Colore di sfondo <BODY BGCOLOR="#******">

Colore del testo <BODY TEXT="#******">

Colore dei collegamenti <BODY LINK=”#******”>

Colore dei coll. cliccati <BODY VLINK=”#******”>

Colore del coll. selezionato <BODY ALINK=”#******”>

FORMATTAZIONE DEL TESTO

Neretto:	<B></B>	
Corsivo:	<I></I>	
Sottolineare:	<U></U>	
Apice:		
Pendice:		
Macc.da scrivere:	<TT></TT>	
Paragrafo:	<P></P>	
Preformattato:	<PRE></PRE>	(compresi gli spazi)
Centrare:	<CENTER></CENTER>	(testo e immagini)
Dimensioni Font:	<FONT SIZE=?> </FONT>	(da 1 a 7)
Dimensioni testo:	<FONT SIZE=”+|-?”> </FONT>	(da –3 a +3)
Colore Font:	<FONT COLOR=”#*****”></FONT>	
Tipologia Font:	<FONT FACE=“Courier”> </FONT>	

CSS

Abbiamo parlato brevemente di HTML e quindi sprechiamo qualche riga anche per il CSS. La sigla CSS sta per Cascading Style Sheets,

tradotto: fogli di stile. Praticamente il CSS è un linguaggio che gestisce l'aspetto delle pagine Web e lavora in combinazione con l'HTML il quale invece gestisce i contenuti delle pagine.
Con il CSS si stabiliscono le regole inerenti la formattazione e lo stile (font, colori, dimensioni, ecc) separati da quelli relativi al contenuto.
Vengono chiamati a cascata perché i fogli di stile su cui lavorare sono svariati, ma solo uno eredita a cascata tutte le proprietà dei fogli precedenti. Esistono tre tipi di fogli di stile CSS: esterni, interni, in linea.
Gli stili esterni controllano l'aspetto degli oggetti presenti su diverse pagine di un sito. Gli stili interni controllano l'aspetto di una singola pagina. Gli stili in linea controllano solo un elemento su di una singola pagina, anche una sola parola.
Potete usare fogli di stile multipli, i quali sanno già automaticamente in che modo dovranno rinviare l'uno all'altro; in generale, quelli in linea hanno la precedenza sui fogli interni, i quali a loro volta ce l'hanno sugli esterni.
La creazione di un foglio di stile e la scrittura dei comandi di CSS avviene mediante l'utilizzo di un editor di testo. Perché le regole inserite entrino in vigore e la vostra pagina web le rispetti, dovete collegare il foglio di stile al vostro HTML. Vediamo un piccolo esempio di CSS applicato al titolo della pagina:

```
#titolo {text-shadow: 5px 5px 5px #FF0000;}
```

Con questo codice il nostro titolo avrà la sua ombra, entreremo nel dettaglio più avanti nei prossimi capitoli.
Per approfondire ulteriormente l'argomento ed esplorare l'enorme quantità di possibilità offerte consiglio una ricerca su internet, come queste pagine web dell'organizzazione Mozilla:
https://developer.mozilla.org/it/docs/Learn/Getting_started_with_the_web/CSS_basics
Prima di iniziare il capitolo pensiamo al nostro browser, a tutte le volte che lo abbiamo eseguito e a quanto abbiamo atteso i caricamenti dei vari siti. In qualche caso, dopo un nostro click avremmo visto la fase di costruzione della pagina immaginando il codice che stava dietro. Sicuramente in pochi si saranno soffermati su questi aspetti, soprattutto negli ultimi anni, dove la velocità (con la fibra ottica) e l'evoluzione verso il Web 3.0 sta portando a diversi cambiamenti ma anche a standardizzazioni di molti siti web. Tuttavia, nonostante i numerosi futuri progressi, lo studio e l'apprendimento di queste fasi fondamentali non sono cambiati.

Parliamo dell'ordine della sequenza di eventi o "*ciclo di vita*", e in particolare di come il nostro codice HTML ci si relaziona. Il ciclo di vita inizia nel momento in cui la pagina è richiesta, attraverso le interazioni eseguite dall'utente, fino alla chiusura della stessa. Inizieremo esplorando come viene costruita la pagina elaborando il codice HTML. Procederemo osservando l'esecuzione del codice JavaScript e HTML, esamineremo come sono gestiti gli eventi e infine, durante questo percorso, esploreremo alcuni concetti fondamentali delle applicazioni web come il BOM, il DOM e il ciclo di eventi.

Impariamo l'ordine

L'ordine o la successione regolare di una serie di eventi in un listato di codice per il Web, ad esempio Javascript, è molto importante e, parlando dell'esempio più semplice, cioè un sito internet, deve interfacciarsi con le API, il codice HTML, il CSS e così via. Inizia con l'utente che digita un URL nella barra degli indirizzi del browser o fa clic su un collegamento. Se volessimo dare un'occhiata alle ultime news dal mondo, apriremo il nostro browser e digiteremo, ad esempio, l'indirizzo www.corriere.it, a questo punto il browser, fungendo da tramite, formulerà una richiesta che sarà inviata a un server, che la elaborerà, inoltrando poi una risposta. Nel momento in cui il browser riceve questa risposta l'applicazione web inizierà il suo ciclo di vita. L'ordine di tutti questi eventi ha una sua sequenza, un suo ordine, che impareremo a conoscere e che bisognerà tenere a mente.

Poiché le applicazioni Web lato client (come i siti internet) sono applicazioni che si avvalgono di una GUI (Graphical User Interface, tradotto: interfaccia grafica), il loro ciclo di vita è contraddistinto da passaggi del tutto analoghi. Ad esempio, un programma per la videoscrittura, per il fotoritocco o anche un videogame, è eseguito in due fasi, la prima sarà la costruzione della pagina, dove l'interfaccia utente sarà modulata al meglio per rispondere allo scopo del programma. La seconda sarà la gestione degli eventi, dove una serie di questi resta sopita, finché l'utente non esegue le sue richieste, innescando così i gestori degli eventi.

Il ciclo di vita dell'applicazione termina quando l'utente termina e chiude il programma o la pagina del sito sul nostro browser.

Andiamo avanti. Anche gli eventi stessi, di cui abbiamo parlato pocanzi, hanno un ciclo di vita, un loro ordine. Precisamente sono

tre fasi:
- la fase di *cattura* dell'evento

- la fase *obiettivo* dell'evento

- la fase cosiddetta di "*bollitura*" dell'evento

L'immagine appena vista illustra come si sviluppa il ciclo di un classico evento come il click di un pulsante su schermo. Inizialmente l'istruzione addetta all'ascolto cattura ed indirizza il nostro evento verso l'area o l'elemento designato, dopodiché, nella fase di "bollitura" l'evento si unirà al codice associato alla sua cattura. Vediamo un esempio su cui dovremo tornare per capire appieno tutti i meccanismi:

```
<html>

<body id="abc">
  <div id="provaclick">
    <button id="pulsante">Cliccami</button>
     </div>

    <script type="text/javascript">
      document.getElementById('abc')
             .addEventListener('click', function() {
             alert('Corpo');
         })
         document.getElementById('provaclick')

         .addEventListener('click', function() {
             alert('Div');
         }, true)

      document.getElementById('pulsante')
             .addEventListener('click', function() {
             alert('Pulsante');
         })

    </script>
</body>
 </html>
```

Nel codice qui illustrato, *corpo* (body) e *pulsante* inizialmente sono nella fase di bollitura mentre *div* è impostato sulla fase di cattura. Quando si fa click sul pulsante, si ripartirà dall'alto, come nell'illustrazione precedente arriveremo poi all'elemento *body* (abc) ma saremo ancora nella fase di cattura, quindi non verrà mostrato il testo *Corpo*. Andremo avanti, verso *div* e qui il parametro dell'ascoltatore di eventi è *true*, perciò la funzione verrà eseguita mostrando a questo punto il testo "*Div*". Raggiunto il pulsante si passa dalla fase di *cattura* alla fase *obiettivo*, e quindi verrà mostrato il testo "*Pulsante*", ed infine alla fase di *bollitura,* e qui sarà il momento del testo "*Corpo*".

Proseguiamo con lo stesso argomento osservando la sequenza di eventi che si avvia ogni volta che una pagina web viene eseguita.

La sequenza di eventi in una pagina Web

Quando apriamo il browser e scriviamo un sito sulla barra degli indirizzi dando poi l'invio, prima ancora che la nostra pagina sia visualizzata e che noi possiamo interagirci, essa dovrà essere "costruita" mediante le informazioni contenute nel codice HTML, incluse le istruzioni dei CSS (fogli di stile, essenzialmente l'aspetto della pagina, come visto nel precedente capitolo) e ovviamente il codice JavaScript. Precisamente questo è quello che accade in sequenza:

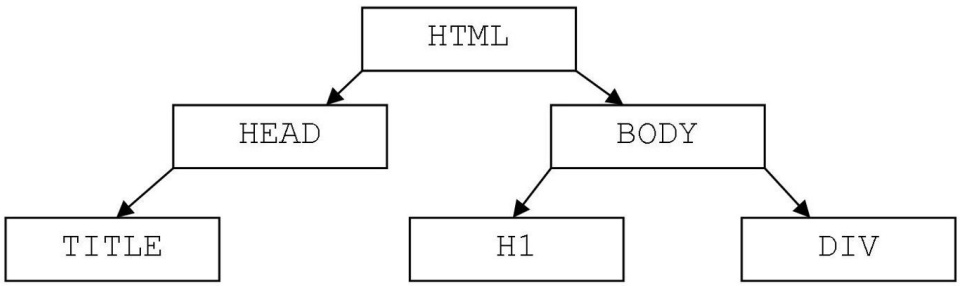
1) Analisi dell'HTML e creazione del DOM (Document Object Model, ne parliamo brevemente più avanti nel capitolo)

2) Esecuzione dei CSS

3) Esecuzione del codice JavaScript

Il passaggio 1 e 2 vengono eseguiti quando il browser elabora i nodi HTML e il passaggio 3 viene eseguito ogni volta che si incontra un tipo speciale di elemento HTML, l'elemento script (che contiene o fa riferimento al codice JavaScript). Durante la fase di creazione della pagina, il browser può alternare questi tre passaggi tutte le volte che è necessario.

Il DOM è una rappresentazione strutturata della pagina HTML in cui ogni elemento è rappresentato come nodo. Ad esempio, qui sotto si può vedere come viene visualizzato schematicamente il DOM prima dell'esecuzione di uno script:



Si può notare che come questo tipo di struttura ad albero suddivisa in oggetti *padre*, da cui si generano oggetti *figlio* e i *rami* che li collegano, sia di facile lettura e comprensione.
Durante la costruzione della pagina, il browser può incontrare un tipo speciale di elemento HTML, l'elemento script, che, come abbiamo già visto, viene utilizzato per includere il codice JavaScript. Quando ciò accade, il browser sospende la costruzione del DOM dal codice HTML e avvia l'esecuzione del codice JavaScript.
Tutto il codice JavaScript contenuto nell'elemento script viene eseguito dal motore JavaScript del browser; ad esempio, Spidermonkey di Firefox, Chrome e Opera's V8 o Chakra di Edge (IE). Poiché lo scopo principale del codice JavaScript è fornire dinamicità alla pagina, il browser fornisce un'API tramite un *oggetto globale* che può essere utilizzato dal motore JavaScript per interagire e modificare la pagina.
L'oggetto globale principale che il browser espone al motore JavaScript è l'oggetto *finestra* (o meglio: *window*), dove è contenuta una pagina. In essa sono accessibili tutti gli altri oggetti globali, le variabili e le API del browser. Una delle proprietà più importanti dell'oggetto finestra globale è il documento, che rappresenta il DOM della pagina corrente. Utilizzando questo oggetto, il codice JavaScript può alterare il DOM della pagina in qualsiasi misura, modificando o rimuovendo elementi esistenti, e persino creandone e inserendone di nuovi.

```
  <body id="abc">
       <div id="provaclick">
           <button id="pulsante">Cliccami</button>
       </div>
       <script type="text/javascript">
           document.getElementById('abc')
azione da eseguire…
```

Questa porzione di codice utilizzato qui utilizza l'oggetto *documento globale* per selezionare prima un elemento con l'ID dal DOM (in questo caso l'elemento "*abc*") e esegundo poi un'azione. Possiamo quindi utilizzare il codice JavaScript per apportare tutti i tipi di modifiche a quell'elemento, come cambiare il suo contenuto testuale, modificare i suoi attributi, creare dinamicamente e aggiungere nuovi *figli* ad esso e persino rimuovendo l'elemento dal DOM.

BOM e DOM

Il modello BOM, ossia Browser Objet Model, permette l'accesso e la possibilità di editare la finestra (Window) del browser. Lo sviluppatore potrà modificare il testo della barra di stato della finestra, muoverla, e agire su diversi parametri della stessa. Non esiste uno standard come per il DOM, quindi ogni browser ne ha uno suo.

Più imortante ed incisivo per noi è il DOM (Document Object Model), il quale definisce uno standard per l'accesso ai documenti. Il modello DOM è una piattaforma e un'interfaccia indipendente dal linguaggio che consente a programmi e script di accedere dinamicamente e aggiornare il contenuto, la struttura e lo stile di un documento.

Lo standard W3C (*World Wide Web Consortium*) del DOM è suddiviso in 3 parti differenti:

Core DOM - modello standard per tutti i tipi di documenti

XML DOM - modello standard per documenti XML
HTML DOM - modello standard per documenti HTML

A noi interessa l'HTML DOM, il quale è un modello di oggetti standard e un'interfaccia di programmazione che definisce gli elementi HTML come oggetti, le loro proprietà, i metodi per accedere a tutti gli elementi e i loro eventi.
Ne consegue che con questa piattaforma potremo ottenere, modificare, aggiungere o eliminare elementi HTML i quali sono organizzati all'interno del documento con una struttura ad albero.
I metodi HTML DOM sono azioni che possiamo eseguire (sugli elementi), mentre le proprietà HTML DOM sono valori (di elementi) che puoi impostare o modificare.
È possibile accedere al DOM HTML con JavaScript (e con altri linguaggi di programmazione), qui tutti gli elementi sono definiti come oggetti e l'interfaccia di programmazione è costituita dalle proprietà e dai metodi di ogni oggetto. Una proprietà è un valore che puoi ottenere o impostare (come modificare il contenuto di un elemento HTML).
Un metodo è un'azione che puoi eseguire (come aggiungere o eliminare un elemento HTML), qui sotto vediamo la modifica

del contenuto (*innerHTML*) <z> dell'elemento con id="prova":

```
   <html>
<body>
 <z id="prova"></z>
  <script>
     document.getElementById ("prova").innerHTML = "La
nostra prima stringa";
 </script>
</body>
</html>
```

Notiamo che *getElementById* è un metodo, mentre *innerHTML* è una proprietà, con il primo avremo Il modo più comune per accedere a un elemento HTML, tramite l'elemento id, come nell'esempio sopra, dove abbiamo utilizzato il metodo *getElementById* per trovare l'elemento *id="prova"* .

La proprietà innerHTML

Il modo più semplice per ottenere il contenuto di un elemento è utilizzare la proprietà *innerHTML*, essa è utile per ottenere o sostituire e modificare il contenuto degli elementi HTML, inclusi *<html>* e *<body>* (in questo caso non sarà possibile sostituirli ovviamente). Vediamo:

```
document.getElementById(id).innerHTML = new HTML
```

Questo esempio cambia il contenuto dell'elemento *<z>:*

```
<html>
<body>
  <z id="prova2">Ecco il nostro esempio</z>
  <script>
  document.getElementById("prova2").innerHTML = "Nuovo testo";
  </script>
</body>
</html>
```

Il documento HTML sopra contiene un elemento *<p>* con *id="prova2",* usiamo il DOM HTML per ottenere l'elemento con *id="prova2"* e poi JavaScript cambia il contenuto (*innerHTML*) di quell'elemento in "Nuovo testo". Quest'altro esempio cambia il contenuto dell'elemento *<a1>*:

```
<!DOCTYPE html>
<html>
  <body>
  <a1 id="pr1">Piè pagina</a1>
  <script>
  var element = document.getElementById("pr1");
  element.innerHTML = "Nuovo piè pagina";
  </script>
</body>
</html>
```

Il documento HTML sopra contiene un elemento *<a1>* con *id="pr1*", usiamo il DOM HTML per ottenere l'elemento

con *id="pr1",* dopodiché JavaScript cambia il contenuto (*innerHTML*) di quell'elemento in *"Nuovo piè pagina"*.

Modifica dello stile HTML

Il DOM HTML consente a JavaScript di modificare lo stile degli elementi HTML. Per modificare lo stile di un elemento HTML, utilizza questa sintassi:

```
document.getElementById(id).style.property = nuovo stile
```

L'esempio seguente cambia lo stile di un elemento *<ab>:*

```
<html>
<body>
  <ab id="pr5">Buongiorno</ab>
  <script>
  document.getElementById("pr5").style.color = "blue";
  </script>
  <ab>Colore cambiato!</ab>
</body>
</html>
```

Utilizzo di eventi

Il DOM HTML consente di eseguire codice quando si verifica un evento. Essi vengono generati dal browser quando "qualcosa" accade agli elementi HTML, ad esempio quando si fa clic su un elemento, la pagina è stata caricata, i campi di input vengono modificati.

Questo esempio cambia lo stile dell'elemento HTML con *id="id9",* quando l'utente fa clic su un pulsante:

```
    <!DOCTYPE html>
<html>
  <body>
  <a3 id="id9">Intestazione</a3>
  <button                type="button"
onclick="document.getElementById('id9').style.color    =
'green'">
   Cliccami!</button>
  </body>
   </html>
```

Creazione di nuovi elementi HTML (nodi)

Per aggiungere un nuovo elemento al DOM HTML, è necessario prima creare l'elemento (nodo dell'elemento), quindi aggiungerlo a un elemento esistente:

```
   <div id="d5">
  <a id="a1">Paragrafo</a>
  <a id="a2">Un altro paragrafo</a>
   </div>
   <script>
   var pfo = document.createElement("a");
   var nodo = document.createTextNode("Ecco quello
nuovo");
   pfo.appendChild(nodo);
   var element = document.getElementById("d5");
   element.appendChild(pfo);
   </script>
```

Questo codice crea un nuovo elemento *<a>:*

```
   var pfo = document.createElement("a");
```

Per aggiungere testo all'elemento *<a>,* bisogna prima creare un nodo di testo, dopo lo aggiungeremo all'elemento *<a>:*

```
   var nodo = document.createTextNode("Ecco quello
nuovo");
para.appendChild(nodo);
```

Infine è necessario aggiungere il nuovo elemento a un elemento esistente, prima cercandolo tramite l'id e poi aggiungendolo:

```
   var element = document.getElementById("d5");
element.appendChild(pfo);
```

Rimozione di elementi HTML esistenti

Per rimuovere un elemento HTML, utilizzeremo il metodo *remove():*

```
   <div>
  <z id="z2">Paragrafo 10</z>
  <z id="z3">Paragrafo 30</z>
</div>
<script>
```

```
var elmnt = document.getElementById("z2");
elmnt.remove();
</script>
```

Il documento HTML contiene un elemento <*div*> con due nodi figlio (due elementi <*z*>):

```
    <div>
  <z id="z2">Paragrafo 10</z>
  <z id="z3">Paragrafo 30</z>
</div>
```

Trova l'elemento che desideri rimuovere e poi lo eliminiamo con il metodo *remove ()*:

```
    var elmnt = document.getElementById("z2");
elmnt.remove();
```

Cenni su jQuery

Utilizzare codice HTML che funzioni bene su tutti i principali browser Web in modo pulito e conciso è un compito difficile. Oltre a Javascript esistono molte librerie che rendono questo processo meno doloroso e jQuery è una delle più popolari. È facile da usare, ha un'ampia gamma di librerie esistenti ed è adatto per creare facilmente soluzioni di fallback.

Questa parte potrebbe risultare un po' ostica, soprattutto ai neofiti, quindi volendo si può tranquillamente passare oltre, per tornarci in seguito, dopo aver acquisto familiarità con i concetti basilari di HTML e della sua quinta versione.

Proseguiamo con l'introduzione, jQuery è una libreria open source, scritta sempre in Javascript, perlopiù orientata verso le pagine Web, le sue animazioni, la gestione del codice HTML, gli eventi, e tanto altro.

JQuery (https://jquery.com/) ci interessa dato che è molto diffuso su internet per alcune sue peculiarità:

- Fornisce un'API intuitiva e di facile apprendimento per eseguire le attività più comuni dello sviluppo del sito web.
- È abbastanza semplice scrivere plugin per migliorare le capacità di jQuery, e ci sono tantissime librerie di plugin disponibili gratuitamente su Internet.
- Esiste un'ampia comunità di sviluppatori pronta ad aiutarci, purtroppo soprattutto in inglese. Comunque esiste anche qualcosa in italiano.
- Probabilmente è la libreria più utilizzata per Javascript.

Quindi ci sono tanti buoni motivi per dare un'occhiata alle sue

funzionalità, nonostante si tratti di una libreria un po' datata. Per utilizzare jQuery dovremo scaricare l'ultima versione dal sito jquery.com, controllando se vi sono eventuali incompatibilità con il browser che intendiamo utilizzare. Vi sono due versioni, una compressa, l'altra no. Inizialmente useremo la seconda, sarà tutto più semplice. La versione compressa ha dei vantaggi in termini di velocità che potranno essere sfruttati in tanti altri casi che vedremo in un secondo momento senza il supporto di questo testo.
Ora, importiamo in maniera molto semplice la libreria nella pagina web usando un tag sorgente nell'intestazione del documento:

```
<script src="jquery-3.6.js"></script> // ovviamente controlliamo la versione scaricata e l'esatta stringa da inserire.
```

Se scegliessimo di caricare jQuery tramite una CDN (Content Delivery Network, un'infrastruttura di rete estesa a livello globale, utile per velocizzare alcuni processi), non sarà necessario scaricare jQuery; dovremo semplicemente aggiungere il seguente tag sorgente nella sezione head della pagina web:

```
<script src=http://ajax.googleapis.com/ajax/libs/jquery/3.6/jquery.min.js></script>
```

Se carichiamo la pagina, potremo verificare se jQuery è installato. Per confermarlo, digitiamo *jQuery* nella console e attendiamoci che non sia restituito alcun errore.
Per chi ha anche solo una minima base di Javascript ora vedremo la sintassi di jQuery e le differenze con il papà Javascript.

Selettori

jQuery non è Javascript e le differenze di base iniziano con i selettori. La sintassi per indicare un id che usavamo, cioè: id = "Mario" diventerà: $(#"Mario"). Il simbolo del dollaro è un alias della funzione jQuery e ha lo scopo di risparmiare sulla digitazione, difatti: jQuery(#"Mario") equivale alla sintassi precedente. Per le classi avremo: $('.tipo'); infine per gli attributi: $('[tempo]'); o anche: $('temperatura[meteo]');

Si noti che in tutti questi casi, la sintassi di selezione è identica alla sintassi utilizzata con CSS. Pertanto, per selezionare tutti gli elementi *prova* dal documento, digitermo: $(‘prova’);

Pseudo Selettori

Proprio come con le pseudo classi CSS, possiamo anche utilizzare i pseudo selettori per selezionare gli elementi. Essi sono sempre preceduti da due punti. Di seguito sono riportati alcuni degli pseudo selettori più utili:

- *:not()* trova gli elementi che non corrispondono alla selezione.

- *:gt()* trova gli elementi con indice maggiore del numero fornito.

- *:even* trova elementi pari in una selezione.

- *:odd* trova elementi con numero dispari in una selezione.

- *:checked* trova i pulsanti di opzione o caselle di controllo selezionate.

- *:selected* trova le opzioni nelle caselle di selezione scelte.

- *:contains()* trova elementi che contengono una data porzione di testo.

- :*first* trova il primo elemento di un set.
- *:last* trova l'ultimo elemento di un set.
- *:focus* trova l'elemento che ha il focus (ad es. selezionato dal mouse).

Ad esempio con: `$('body prova:even');` selezioneremo tutti gli elementi *prova* di numero pari nell'elemento *body,* qui invece: `$('parte:first');` troveremo il primo elemento denominato *parte* nella pagina web. Ultimo esempio, dove cerchiamo tutti gli elementi *prova*, tranne il primo: `$("prova:gt(0)");`

Un'altra caratteristica notevole degli pseudo selettori jQuery è che è anche possibile scriverne di propri. Per approfondire è meglio fare riferimento al sito ufficiale https://api.jquery.com/ che è in inglese, ma possiamo sempre utilizzare la traduzione di Google.

Alcuni metodi jQuery

Abbiamo visto in precedenza come è possibile selezionare elementi figli di altri elementi. Per esempio, quanto segue trova tutti gli elementi *prova* che sono figli del elemento

pagina: `$("pagina prova:even");`
La selezione di elementi nel contesto di un sottoalbero specifico del DOM è molto comune. Per ad esempio, nel nostro sito potremmo voler selezionare sempre gli elementi nel contesto del tag principale per la pagina. Ciò garantirà che anche se la pagina dei tuoi prodotti sia incorporata in una pagina web più grande, selezionerà solo gli elementi rilevanti per essa.
Poiché la selezione di elementi nel contesto di altri elementi è così comune, jQuery ne fornisce due meccanismi aggiuntivi per raggiungerlo. Il primo meccanismo utilizza il metodo *find*: `$('#prodotti').find('prova');`
Questo prima troverà l'elemento con l'ID *prodotti* e poi cercherà all'interno del risultato qualsiasi elemento *prova*.
L'altro modo per ottenere esattamente lo stesso risultato è utilizzare il secondo parametro opzionale per jQuery dopo la stringa di selezione: `$('prova', '#prodotti')` . Il secondo parametro fornisce il contesto all'interno del quale dovrebbe avvenire la selezione.
Ritorniamo un attimo su: `$('#prodotti').find('prova');` vediamo che jQuery non sta semplicemente restituendo un oggetto DOM perché essi non supportano un metodo di ricerca, in questo caso particolare, jQuery ha restituito il proprio tipo di oggetto avvolto sugli oggetti DOM, ed è l'oggetto jQuery su cui viene eseguito find.
In effetti, ogni selezione jQuery restituisce un oggetto specifico di jQuery, non un oggetto DOM. Tuttavia, l'oggetto restituito è in grado di mascherarsi come un array e di accedere a specifici indici restituendo gli oggetti DOM nativi. vediamo un esempio dove assegneremo un oggetto DOM nativo alla variabile *domOgg*: `var domOgg = $('prova')[0];`
È sempre possibile convertire un oggetto DOM nativo in un oggetto jQuery incorporandolo nel file struttura di selezione: `$(domOgg);`
Questo ti dà quindi accesso a tutte le funzionalità aggiuntive fornite da jQuery. Ad esempio, questa riga restituisce il testo dell'elemento: `$(domOgg).text()`, mentre questa chiamata chiede se il testo dell'elemento contiene la stringa "*telefono*": `$(domOgg).is(':contains("telefono")');`
In genere lavorereremo con oggetti jQuery piuttosto che con oggetti DOM nativi, ma possiamo sempre usare questa tecnica

se avremo bisogno di convertire oggetti DOM nativi in oggetti jQuery.
- *hide()* e *show()* rendono facile nascondere e mostrare gli elementi dell'interfaccia utente. Possiamo nascondere uno o più elementi in una pagina come questa: `$("h1").hide();`
Per mostrarli, chiamiamo semplicemente il metodo *show()*. Useremo il metodo *hide()* più avanti per nascondere le sezioni della pagina che devono apparire solo quando JavaScript è disabilitato, come trascrizioni o altri contenuti.
- *html()* ci servirà per ottenere e impostare il contenuto interno dell'elemento specificato, Qui, impostiamo il contenuto tra i tag h1 con *"Ciao da Mario!*": `$("messaggio").html("Ciao da Mario!");`
- *val()* imposta e recupera il valore da un campo modulo, cioè da un form, praticamente come il metodo *html()*.
- *attr()* ci consente di recuperare e impostare attributi sugli elementi.
- *append()* aggiunge un nuovo elemento figlio dopo gli elementi esistenti. Vediamo come in esempio, prima il codice di un form basico:

```
   <form id="aggiungi">
 <label for="task" >Da_fare</label>
  <input type="testo" id="attività">
  <input type="submit" value="Aggiungi">
 </form>
<ul id="collegamenti">
</ul>
```

Da qui possiamo creare nuovi elementi nell'elenco aggiungendoli quando invieremo il modulo:

```
   $(function(){
  $("#aggiungi").submit(function(evento){
    evento.preventDefault();
       var prova = $("<li>" + $("#aggiunto").val() +
"</li>" );
    $("#collegamenti").append(prova);
 });
});
```

- *prepend()* funziona allo stesso modo del metodo *append()* ma inserisce il nuovo elemento prima di quelli esistenti.
- *wrap()* esegue il wrapping dell'elemento selezionato con l'elemento rappresentato dall'oggetto jQuery specificato:

```
   var      prova      =      $("#messaggio").wrap("<div>
<h2>Messaggio</h2></div>" ).parent();
```

Creeremo alcune strutture complesse a livello di codice utilizzando queste tecniche.

CSS e Classi

Possiamo usare il metodo *css()* per definire gli stili sugli elementi, come questo:

```
   $("label").css("color" , "#f00");
```

Possiamo definirli uno alla volta, ma possiamo anche usare un hash JavaScript per assegnare molte regole CSS all'elemento:

```
   $("h1").css( {"color" : "red" ,
             "text-decoration" : "underline" }
         );
```

Tuttavia, non è una buona idea mescolare il CSS con gli script. Possiamo usare i metodi *addClass()* e *removeClass()* di jQuery per aggiungere e rimuovere classi quando si verificano determinati eventi. Possiamo quindi associare gli stili a queste classi. Possiamo cambiare lo sfondo dei nostri campi modulo quando ricevono e perdono il focus combinando eventi e classi jQuery.

```
   $("input").focus(function(evento){
  $(this).addClass("prova");
});

   $("input").blur(function(evento){
   $(this).removeClass("prova");
});
```

Questo è un esempio banale che può essere sostituito dalla pseudoclasse *:focus* in CSS3.

I metodi sugli oggetti jQuery restituiscono oggetti jQuery, il che significa che potremo concatenare i metodi indefinitamente, in questo modo:

```
   $("h2"  ).addClass("hidden"  ).removeClass("visible"
);
```

Di tanto in tanto, avremo anche bisogno di creare nuovi elementi HTML in modo da poterli inserire nel nostro documento, vediamo un altro esempio in cui il concatenamento che ci aiuta a costruire e manipolare rapidamente le strutture:

```
   var prova = $("<p>Ciao da Mario!</p>");
prova.css("color" , "#f00").insertAfter("#header");
```

Faremo attenzione a non abusarne, perché potrebbe rendere il codice più difficile da interpretare ed eventualmente correggere.

Eventi jQuery e il metodo bind()

Spesso abbiamo bisogno di attivare eventi quando gli utenti interagiscono con la nostra pagina e jQuery lo rende molto semplice. In jQuery, molti eventi comuni sono semplicemente metodi sull'oggetto che accetta una funzione. Ad esempio, possiamo fare in modo che tutti i collegamenti su una pagina con la classe di popup si aprano in una nuova finestra come questa:

```
   var coll = $("#link a" );
coll.click(function(evento){
  var prova = $(this).attr('href');
  evento.preventDefault();
  window.open(prova);
});
```

All'interno del nostro gestore di eventi jQuery, possiamo accedere all'elemento con cui stiamo lavorando usando la parola chiave *this*. Alla terza riga, lo passiamo alla funzione jQuery in modo da poter chiamare il metodo *attr()* su di esso per recuperare rapidamente l'indirizzo di destinazione del collegamento.
Usiamo la funzione *preventDefault()* per impedire l'attivazione dell'evento originale in modo che non interferisca con ciò che stiamo facendo.
Alcuni eventi non sono supportati direttamente da jQuery e possiamo usare il metodo *bind()* per gestirli. Ad esempio, quando si implementa la parte Drag and Drop della specifica HTML5, è necessario annullare l'evento *ondragover*. Vediamo:

```
   target = $("#droparea")
target.bind('dragover' , function(evento) {
   if (evento.preventDefault) evento.preventDefault();
   return false;
});
```

Da notare che eliminiamo il prefisso on dall'istruzione, rimane solo *dragover*.
Quando utilizziamo una qualsiasi delle funzioni dell'evento

jQuery come *bind()* o *click(),* jQuery avvolge l'evento JavaScript nel proprio oggetto e copia solo alcune delle proprietà. A volte abbiamo bisogno di arrivare all'evento reale in modo da poter accedere a quelle proprietà che non sono state clonate. Gli eventi jQuery ci danno accesso all'evento originale con la proprietà *originalEvent* opportunamente denominata. Possiamo accedere alla proprietà dei dati dell'evento *onmessage* in questo modo:

```
$(window).bind("message" ,function(evento){
var messaggio = event.originalEvent.data;
});
```

È possibile utilizzare questa tecnica per chiamare qualsiasi proprietà o metodo dell'evento originale.

Chiaramente le immense possibilità offerte da jQuery non finiscono qui, il suggerimento è quello di apprendere le basi di HTML e poi, eventualmente approfondire inanzitutto Javascript e di conseguenza anche jQuery.

Quiz & esercizi

1) Creare un listato che utilizzando jQuery possa gestire i seguenti eventi:

- al click del mouse su di un pulsante cambia il colore di sfondo

- il doppio click sposta lo stesso pulsante

- il passaggio del mouse sopra il pulsante lo fa ingrandire

2) Creare un listato che utilizzando jQuery possa realizzare le seguenti funzioni su tre bottoni:

- al click sul primo bottone cambia il testo sui pulsanti

- il click sul secondo cambia colore del titolo della pagina

- il passaggio del mouse sopra terzo pulsante il primo cambia colore

3) Creare un listato che mostri o nasconda un testo su una pagine web in base al passaggio del mouse.

4) Attraverso i pseudo selettori potremo manipolare il DOM o selezionare elementi di un vettore?

5) Il selettore è obbligatorio o esistono delle scorciatoie per utilizzare jQuery?

 # 1 - Iniziamo con HTML5 e CSS3

La struttura di una pagina Web

Ecco il CSS!

La barra di avanzamento

Da qui in poi inizieremo a fare sul serio, infatti, nei primi capitoli di questo libro parleremo di come utilizzare le funzionalità di HTML5 e CSS3 per migliorare le interfacce dei siti per i nostri utenti. Vedremo come creare form più accattivanti, migliorare l'aspetto grafico generale e implementare l'accessibilità delle nostre pagine per tutti i vari dispositivi esistenti come smartphone, tablet, notebook, smartwatch o anche smart tv, gli elettrodomestici smart, e così via. Impareremo anche come utilizzare la creazione di contenuti per migliorare l'usabilità dei nostri fogli di stile per la stampa e studieremo le modifiche "al volo" con l'attributo *contenteditable*. Per prima cosa, però, daremo un'occhiata a come i nuovi elementi di HTML5 possono aiutarci a strutturare meglio le nostre pagine con dei tag specifici.
Oltre a questi nuovi tag strutturali, parleremo anche dell'elemento *meter* e discuteremo di come utilizzare la nuova funzione degli attributi personalizzati in modo da poter incorporare i dati nei nostri elementi invece di dover dirottare classi o attributi esistenti. In poche parole, scopriremo come ogni tag abbia la sua giusta collocazione.
In questo capitolo esploreremo nuovi elementi e caratteristiche proprie di HTML5, nello specifico i seguenti tag:

<*article*> Definisce un articolo o un contenuto completo.
<*aside*> Definisce il contenuto secondario o correlato.
<*footer*> Definisce un'area del piè di pagina di una pagina o di una sezione.

<*header*> Definisce un'area di intestazione di una pagina o di una sezione.
<*meter*> Descrive un importo all'interno di un intervallo.
<*nav*> Definisce un'area di navigazione di una pagina o di una sezione.

<*progress*> Controllo che mostra i progressi in tempo reale verso un obiettivo.
<*section*> Definisce un'area logica di una pagina o un raggruppamento di contenuti.

Infine, dato che spesso ne sentiremo parlare, specifichiamo cosa s'intende per **markup semantico**: sappiamo già che l'HTML è un linguaggio di markup, cioè essenzialmente un linguaggio di formattazione dove ci sono diverse regole per la corretta rappresentazione o impaginazione di un testo. Il markup è semantico quando si renderanno riconoscibili gli aspetti relativi al significato di ogni elemento e del suo contenuto. Tutto questo servirà ad esempio ai motori di ricerca per ottenere le informazioni richieste su un particolare argomento. Praticamente è come se avessimo un indice a cui rivolgerci per attingere a tutte le informazioni che ci servono, tra poche righe vedremo meglio a cosa ci stiamo riferendo.

Creiamo il sito di un ristorante

Un posto in cui troveremo sicuramente molti contenuti che necessitano di un markup strutturato potrebbe essere il sito di un ristorante. Avremo intestazioni, piè di pagina, più tipi di navigazione (ricette, recensioni e collegamenti interni) e, naturalmente, articoli o post. Useremo il markup HTML5 per simulare la prima pagina del sito del Ristorante **Mario**.
Creeremo la classica pagina Web cercando di rendere subito visibili la maggior parte delle informazioni più importanti, e ovviamente, invece di codificarla con molti tag div, ne utilizzeremo di specifici per descrivere meglio queste aree.
A tal proposito iniziamo subito parlando di un grave problema che affligge molti sviluppatori Web, cioè la sindrome cronica del *Divitis*, la quale li induce a racchiudere elementi con tag div aggiuntivi con id come banner, barre laterali, articoli e piè di pagina. Qualche anno fa questa pratica era dilagante. Gli sviluppatori si scambiano *Divitis* molto rapidamente e poiché i div sono invisibili ad occhio nudo, anche i casi lievi di *Divitis* potrebbero passare inosservati per anni. Osserviamo insieme un esempio di questa perniciosa piaga:

```
   <div id="barra_di_prova">
<div id="barra_laterale">
 <ul>
  <li><a href="/">Mario</a></li>
```

```
  <li><a href="/">Home</a></li>
  <li><a href="/">Index</a></li>
 </ul>
</div>
</div>
```

Qui abbiamo un elenco non ordinato, che è già un elemento di blocco, racchiuso con due tag div i quali anche loro sono elementi di blocco. Gli attributi id su questi elementi wrapper (contenitori) ci dicono cosa fanno, ma possiamo rimuovere almeno uno di questi per ottenere lo stesso risultato. L'uso eccessivo del markup porta al codice *boilerplate* (inutili e ridondanti ripetizioni) e a pagine difficili da definire e mantenere.
Per fortuna HTML5 fornisce una cura sotto forma di nuovi *tag semantici* che descrivono perfettamente il loro contenuto. Poiché così tanti sviluppatori hanno creato barre laterali, intestazioni, piè di pagina e sezioni nei loro progetti, la specifica HTML5 introduce nuovi tag progettati specificamente per dividere una pagina in regioni logiche. Come già anticipato tutto questo servirà molto ai motori di ricerca per indicizzare meglio i contenuti e anche per "guarire" da *Divitis* e *boilerplate code*.
Iniziamo creando una pagina *index.html* e inserendo questo modello HTML5 di base nel file:

```
   <!DOCTYPE html>
<html lang="it-IT">
<head>
 <meta http-equiv="Content-Type" content="text/html;
 charset=utf-8"/>
 <title>Ristorante Mario</title>
</head>
 <body>
  </body>
</html>
```

Osserviamo subito che in HTML5 dovremo indicare il *doctype*, che serve al browser per capire le regole da seguire in base al linguaggio che andremo ad utilizzare, difatti potremo integrare l'HTML con Javascript, SQL o il PHP, ad esempio e in quel caso dovremo specificarlo con il *doctype*.

La struttura di una pagina con HTML5

Oltre al *doctype* avremo le intestazioni (*header*), da non confondere con le intestazioni come *h1* o *h2*, le quali possono

includere ogni tipo di contenuto, dal logo dell'azienda alla casella di ricerca. L'intestazione del nostro ristorante conterrà per ora solo il suo nome:

```
   <header id="intestazione">
<h1>Ristorante Mario</h1>
</header>
```

Non siamo limitati ad avere solo un'intestazione su una pagina. Ogni singola sezione o articolo potrà averla, quindi sarà utile utilizzare l'attributo ID, come nella prima riga del listato appena visto, per identificare in modo specifico i nostri elementi, inoltre un ID univoco ne semplifica lo stile con CSS o la loro individuazione con altri linguaggi come PHP o JavaScript.
L'elemento piè di pagina (*footer*) definisce le informazioni per un documento o una sezione adiacente. Sicuramente avremo già visto il piè di pagina su tanti siti Web, di solito conteniene informazioni come la data del copyright e informazioni sul gestore o sul proprietario. La specifica dice che possiamo avere diversi piè di pagina anche in un documento, quindi questo significa che potremmo usarli anche all'interno dei nostri post o articoli.
Per ora, definiremo solo un semplice piè di pagina, poiché potremo averne più di uno, gli daremo un ID proprio come abbiamo fatto con l'intestazione. Ci aiuterà a identificarlo in modo univoco nel momento in cui vorremmo aggiungere stili a questo elemento e ai suoi figli:

```
   <footer id="piè_pagina">
<p>&copy; Mario SRL 2022</p>
</footer>
```

Questo piè di pagina contiene semplicemente una data di copyright. Tuttavia, come le intestazioni, il piè di pagina spesso comprende altri elementi, inclusi quelli di navigazione.
La *navigazione* è fondamentale per il successo di un sito Web. Le persone semplicemente non rimarranno in giro se rendiamo troppo difficile per loro trovare ciò che stanno cercando, quindi ha senso che la navigazione ottenga il proprio tag HTML.
Aggiungiamo una sezione di navigazione all'intestazione del nostro documento. Aggiungeremo collegamenti alla home

page del sito, ai post, a una pagina che elenca le recensioni e a un collegamento ai contatti:

```
   <header id="intestazione">
<h1>Ristorante Mario</h1>
 <nav>
  <ul>
   <li><a href="/">ultimi post</a></li>
   <li><a href="archivi">Archivi</a></li>
   <li><a href="recensioni">Recensioni</a></li>
   <li><a href="contatti">Contatatti</a></li>
  </ul>
 </nav>
</header>
```

Nella nostra pagina avremo più elementi di navigazione, come per l'intestazione e il piè di pagina. Capiterà di trovarli spesso mischiati assieme e adesso potremo identificarli in modo esplicito. Il *footer* del nostro sito deve contenere collegamenti alla home page (principale), alla pagina "chi siamo" dell'azienda e collegamenti alle informazioni generali e alle politiche sulla privacy. Li aggiungeremo come un altro elenco non ordinato all'interno dell'elemento a piè di pagina del sito:

```
   <footer id="piè_pagina">
<p>&copy; Mario SRL 2022</p>
 <nav>
  <ul>
                                                    <li><a
href="http://ristorantemario.it/">Principale</a></li>
   <li><a href="info">Chi siamo</a></li>
   <li><a href="informazioni.html">Informazioni</a></li>
   <li><a href="privacy.html">Privacy</a></li>
  </ul>
 </nav>
</footer>
```

Procedendo con i paragrafi ci prenderemo cura dell'aspetto delle barre di navigazione usufruendo del CSS. Quindi per ora concentriamoci sullo scopo di questi nuovi elementi, cioè non concentrarsi sull'aspetto, ma descrivere il contenuto.
Quindi, discorso estetico a parte parliamo delle sezioni, le quali sono le regioni logiche di una pagina, infatti l'elemento *section* servirà per sostituire il tag *div* spesso abusato per il medesimo scopo:

```
   <section id="post">
</section>
```

Riassumendo, in pratica *section* ci servirà a raggruppare logicamente i nostri contenuti.
Nel sito del ristorante abbiamo anche creato una sezione che conterrà tutti i post. Tuttavia, ogni post non dovrebbe essere nella propria sezione, utilizzeremo un tag più appropriato per questo, cioè ll tag *article*, il quale è l'elemento perfetto per descrivere il contenuto effettivo di una pagina Web. Con così tanti elementi su di essa, inclusi intestazioni, piè di pagina, elementi di navigazione, pubblicità, widget, e segnalibri dei social media, potrebbe essere facile dimenticare che le persone visitano il nostro sito perché sono interessate al contenuto che stiamo fornendo. Il tag *article* ci aiuterà proprio a descrivere quel contenuto.
Ciascuno dei nostri articoli avrà un'intestazione, alcuni contenuti e un piè di pagina. Possiamo definire un intero articolo così:

```
   <article class="post">
<header>
  <h2>Oggi sveliamo il segreto della Cacio e Pepe</h2>
  <p>Inserito da Chef Mario il
   <time datetime="2022-06-13T10:22">13 Giugno 2022 alle
10:22
   </time>
  </p>
</header>
 <p> Per creare la crema con il pecorino romano ed il
pepe bisognerà, mentre la pasta è sul fuoco, mischiare
l'acqua di cottura con il cacio e mischiare, scolare la
pasta qualche minuto prima e saltarla con la crema
ottenuta aggiustando, se necessario, con l'acqua di
cottura.
 </p>
   <footer>
  </footer>
</article>
```

Abbiamo la possibilità di servirci degli elementi di intestazione e piè di pagina all'interno dei nostri articoli, il che renderà molto più semplice descrivere quelle sezioni specifiche. Potremo anche dividere il nostro articolo in più parti usando l'elemento *section*.
A volte avremo contenuti che aggiungono qualcosa in più al nostro articolo principale, come citazioni estraibili, diagrammi, pensieri aggiuntivi o collegamenti correlati. Potremo utilizzare il tag *aside* per identificarli correttamente:

```
   <aside>
  <p>
   “Chi mette la panna nella cacio e pepe ha un suo girone dedicato all’infermo.”
  </p>
</aside>
```

Questo tag andremo a posizionarlo subito prima del nostro post.
Nel nostro sito avremo anche una barra laterale sul lato destro che contiene i link agli archivi dei post e delle ricette. Non andremo assolutamente ad usare il tag *aside* per definire la barra laterale del nostro sito. Potremmo farlo in questo modo, ma va contro lo spirito delle specifiche. Il lato è progettato per mostrare il contenuto relativo a un articolo. È un buon posto per mostrare link correlati, un glossario o una citazione da estrarre.
Per contrassegnare la nostra barra laterale che contiene il nostro elenco di archivi precedenti, utilizzeremo semplicemente un altro tag di sezione e un tag di navigazione:

```
   <section id=“barralaterale”>
 <nav>
   <h3>Archivi</h3>
    <ul>
      <li><a href=“2022/01”>Gennaio 2022</a></li>
      <li><a href=“2022/02”>Febbraio 2022</a></li>
      <li><a href=“2022/03”>Marzo 2022</a></li>
      <li><a href=“2022/04”>Aprile 2022</a></li>
      <li><a href=“2022/05”>Maggio 2022</a></li>
   </ul>
</nav>

   </section>
```

Per il momento abbiamo la struttura del sito di Mario. Ora andremo ad applicare il CSS agli elementi appena visti.

Finalmente il CSS!

Questo piccolo intervento che ci apprestiamo ad eseguire ci farà comprendere le potenzialità del CSS. Per prima cosa, creiamo un nuovo file per il foglio di stile chiamato *style.css* che allegheremo al nostro documento HTML inserendolo nell'intestazione del documento:

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

Proseguiamo centrando il contenuto della pagina e poi impostiamo alcuni stili di carattere di base, poi definiremo la larghezza dell'intestazione:

```
   body{
  width:1080px;
   margin: 18px auto;
  font-family: Arial, "MS Trebuchet", sans-serif;
  }
   p{
  margin:0 0 22px 0;
}
   p, li{
  line-height: 22px;
   }
   header#intestazione{
 width: 100%;
   }
```

Sfruttiamo il CSS per i collegamenti di navigazione trasformando gli elenchi puntati in barre di navigazione orizzontali:

```
   header#intestazione nav ul, #piè_pagina nav ul{
  list-style: none;
  margin: 0;
  padding: 0;
}
   #intestazione nav ul li, footer#piè_pagina nav ul li{
   padding:0;
   margin: 0 22px 0 0;
```

```
   display:inline;
}
```

La sezione dei post deve essere spostata a sinistra e assegnata a una larghezza, e dobbiamo anche spostare il *callout* (richiamo, elemento utile a rendere visibile qualcosa) all'interno dell'articolo. Mentre lo facciamo, aumentiamo la dimensione del carattere per il *callout*:

```
   section#post{
   float: left;
   width: 75%;
}

   section#post aside{
    float: right;
    width: 34%;
    margin-left: 5%;
    font-size: 21px;
    line-height: 42px;
}
```

Avremo anche bisogno di rendere mobile la barra laterale e definirne la larghezza:

```
   section#barralaterale{
   float: left;
   width: 25%;
}
```

Infine applicheremo il CSS al piè di pagina. Procederemo concellando i *float* nel *footer* in modo che si trovi in fondo alla pagina:

```
   footer#piè_pagina{
  clear: eboth;
  width: 100%;
  display: block;
  text-align: center;
}
```

Questi sono solo stili di base, utili a comprendere le fondamenta del CSS. Sicuramente, quando ci abitueremo ad usarli, non potremo più farne a meno.
Adesso apriamo una piccola parentesi sull'istruzione *meter*.

Mostrare i progressi

Se è necessario implementare un misuratore o una barra di avanzamento del caricamento in un'applicazione Web, è necessario esaminare il misuratore e gli elementi di avanzamento introdotti in HTML5. L'elemento *meter* ci

permetterà di descrivere semanticamente un punto fisso effettivo su un metro con un valore minimo e massimo. Non potremo usare il misuratore per cose con valori minimi o massimi arbitrari come altezza e peso, a meno che stiamo parlando di qualcosa di specifico in cui abbiamo impostato un limite preciso. Ad esempio, se volessimo mostrare quanta gente apprezza il nostro ristorante e vogliamo mostrare quanto siamo vicini al nostro obiettivo di 1.000 persone, possiamo descriverlo facilmente:

```
   <section id="misura">
<header>
<h3>Clienti che hanno apprezzato il nostro ristorante</h3>
</header>
<meter title="Clienti" id="clienti_voto"
          value="358" min="0" max="1000" >
   358 persone
</meter>
<p>Vota anche tu, raggiungiamo le 1.000 persone!</p>
</section>
```

La cosiddetta barra di avanzamento è stata progettata per mostrare i progressi attivi visualizzati come se stessimo caricando un file. *Meter*, in confronto, è progettato per mostrare una misura quello è non attualmente in movimento. Il markup per una barra di avanzamento è molto simile all'elemento *meter*:

```
   <progress id="progbarra" max=100><span>0</span>%
</progress>
```

Vediamo adesso un esempio diverso in HTML con delle barre di colori differenti:

```
   <div class="avanzamento">

   <div class="progbarra w-25 bg-danger"
role="progressbar" aria-valuenow="25" aria-valuemin="0"
aria-valuemax="100"></div>

   </div>

   <div class="avanzamento">

   <div class="progbarra w-50 bg-success"
role="progressbar" aria-valuenow="50" aria-valuemin="0"
aria-valuemax="100"></div>

   </div>
```

```
    <div class="avanzamento">
    <div       class="progbarra       w-75       bg-info"
role="progressbar" aria-valuenow="75" aria-valuemin="0"
aria-
    valuemax="100"></div>
    </div>
```

Quiz & esercizi

1) Per una migliore indicizzazione del nostro sito Web sfrutteremo un buon markup semantico o pay per click di Google?
2) Cosa viene utilizzato per applicare correttamente uno stile CSS solo ad una precisa porzione della pagina?
3) Completa questo codice:

```
   <header>
<h3>testo testo testo</h3>
```

4) Come faremo ad aggiungere un foglio di stile alla nostra pagina?
5) Nel tag *nav* cosa dovremmo inserire?
6) In HTML5 il doctype andrà posizionato all'inizio o alla fine?
7) Perché usare i nuovi tag invece di *div*?

Riassunto

Abbiamo iniziato con i tag fondamentali di ogni pagina Web, i quali serviranno ad avere un sito conforme a tutti gli standard. Abbiamo anche visto le prime righe di CSS, è solamente l'inizio ed è sostanzialmente semplice, quindi impariamo bene queste semplici regole di base.

 # 2 - I moduli con HTML5

Migliorare un form

Utilizziamo il segnaposto

Contenteditable

Abbiamo visto nel capitolo precedente come impostare correttamente la struttura di una pagina Web. Adesso inizieremo ad esplorare delle funzionalità leggermente più avanzate, partiremo dai cosiddetti *form* o moduli, parecchio migliorati rispetto a ciò che offriva l'HTML precedentemente. Molti, per questo scopo, potrebbero pensare giustamente di rivolgersi a jQuery, oppure di utilizzare una combinazione di HTML, CSS e JavaScript. Ma questo vorrebbe dire che dovremmo lavorare con: dispositivi di scorrimento, controlli del calendario, caselle numeriche, campi di completamento automatico, assicurandoci che i controlli inclusi nella pagina non siano in conflitto con nessuno degli altri controlli o con qualsiasi altra libreria JavaScript nella pagina. Ci renderemo subito conto di esserci infilati da soli in un mare di problemi, per utilizzare un eufemismo.

Per fortuna il nostro HTML5 ci viene incontro e in questo capitolo creeremo un paio di moduli Web utilizzando alcuni nuovi tipi di campi modulo e proveremo anche il segnaposto (*placeholder*) del testo.

Infine, discuteremo come utilizzare il nuovo attributo *contenteditable* per trasformare qualsiasi campo HTML in un controllo di input dell'utente.

HTML5 ha introdotto diversi nuovi tipi di input che abbiamo utilizzato per descrivere meglio il tipo di dati che i nostri utenti dovranno inserire. Oltre ai campi di testo standard, ai pulsanti di opzione e agli elementi delle caselle di controllo, ci serviremo di elementi come campi e-mail, calendari, selettori di colori, caselle numeriche e dispositivi di scorrimento. I browser hanno usufruito di questi nuovi *campi* per visualizzare controlli migliori per l'utente senza la necessità di JavaScript, lo stesso dicasi per i dispositivi mobili e i tablet i quali oltretutto hanno avuto anche un adeguamento

specifico a loro dedicato. Andiamo ad osservare in un esempio concreto tutto ciò.

Implementiamo il nostro sito

Mario ha avuto un'idea geniale. Per creare nuove ricette e sperimentare nuove tecniche di cottura vorrebbe avvalersi della collaborazione di diversi chef in giro per il mondo. Quindi ha pensato di creare un'applicazione web per la gestione di questi progetti e per consentire ai suoi aiuti in cucina di tenere il passo più facilmente con tutte le novità del mondo della ristorazione. Ogni progetto ha un nome, un indirizzo e-mail di contatto e un URL di staging in modo che i gestori possano visualizzare in anteprima il sito Web mentre viene creato. Ci sono anche campi per la data di inizio, la priorità e il numero stimato di ore che il progetto dovrebbe impiegare per essere completato. Infine, il responsabile della cucina vorrebbe assegnare a ciascun progetto un colore in modo da poterli identificare rapidamente quando li esamina. Lo vedremo nella pagine delle preferenze del progetto utilizzando i *campi* (fieldset) di HTML5.

Il modulo base

Creiamo un semplice modulo che esegue la classica richiesta *post*. Andiamo a vedere un codice di esempio:

```
   <form method="post" action"/ricette/1">
<fieldset id="info_personali">
 <legend>Prove di nuove ricette e tecniche</legend>
<ol>
 <li>
    <label for="name">Nome</label>
    <input type="text" name="nome" autofocus id="nome">
</li>
<li>
  <input type="submit" value="Invia">
</li>
</ol>
</fieldset>
</form>
```

Stiamo contrassegnando questo modulo con etichette (label) avvolte in un elenco ordinato, questo perché le etichette sono essenziali per creare moduli accessibili. L'attributo *for* dell'etichetta fa riferimento all'*id* dell'elemento del modulo associato. Questo aiuterà chi è dall'altra parte del modulo a identificare i campi in una pagina. L'elenco ordinato fornisce

un buon modo per elencare i campi senza ricorrere a complesse strutture di tabelle o div. Avremo così anche un modo per contrassegnare l'ordine in cui desideriamo che le persone compilino i campi.
Per il nostro form probabilmente avremo bisogno dello strumento *slider*. Di solito viene utilizzato per consentire agli utenti di diminuire o aumentare un valore numerico e potrebbero essere un ottimo modo per consentire rapidamente agli chefs di visualizzare e modificare la precedenza o l'importanza del progetto, della ricetta o del messaggio da inviare. Visualizzeremo un dispositivo di scorrimento con il tipo di intervallo grazie all'uso di *range*, vediamo:

```
   <label for="p">Importanza</label>
<input type="range" min="0" max="5" name="importanza" value="0" id="importanza">
```

Inserendo questo codice nel modulo, all'interno di un nuovo elemento *li* proprio come il campo precedente. Imposteremo l'intervallo minimo e massimo per il dispositivo di scorrimento e limiteremo il valore del campo del modulo.
Oltre allo *slider* ci farebbe comodo anche uno *spinbox,* utile ad esempio per indicare velocemente le ore di realizzazione di una ricetta. Lo *spinbox* è una casella numerica contenente un controllo con frecce che incrementano o decrementano il valore nella casella. L'input type questa volta è *number*. In questo modo, le ore possono essere facilmente selezionate:

```
   <label for="ore_lavoro">Ore necessarie</label>
<input type="number" name="ore_lav" min="0" max="24" id="ore_lav">
```

La casella numerica consente anche di digitare per impostazione predefinita e, come i dispositivi di scorrimento dell'intervallo, possiamo impostare valori minimo e massimo. Tuttavia, gli intervalli minimo e massimo non verranno applicati a nessun valore digitato nel campo.
Ovviamente potremo modificare a nostro piacimento i parametri visualizzati nel codice di esempio. Il valore predefinito è 1 ma potrà essere qualsiasi valore numerico.
Molto importante sarà registrare la data di inizio del progetto e vogliamo renderlo il più semplice possibile, vediamo come usando *date*:

```
    <label for="data_ins">Data inserimento</label>
<input    type="date"    name="data_ins"    id="data_ins"
value="01-06-2022">
```

Siamo arrivati ai campi relativi l'email e l'URL. La specifica HTML5 afferma che il tipo di input *e-mail* è progettato per contenere un singolo indirizzo o anche un elenco, ecco la sintassi:

```
    <label for="email">Contatto e-mail</label>
<input type="email" name="email" id="email">
```

I dispositivi mobili traggono il massimo vantaggio da questo tipo di *campo modulo*, perché i layout della tastiera virtuale cambiano per facilitare l'immissione degli indirizzi e-mail.
C'è un tipo di campo progettato per gestire anche gli URL. Questo è particolarmente utile se il tuo visitatore utilizza uno smartphone, perché mostra un layout della tastiera molto diverso, visualizzando pulsanti di supporto per inserire rapidamente indirizzi Web. Aggiungere il campo URL cosiddetto di *staging* è semplice:

```
    <label for="url">URL di staging</label>
<input type="url" name="url" id="url">
```

Le tastiere virtuali utilizzano questo tipo di campo anche per visualizzare un layout diverso. Infine, faremo in modo di inserire un codice colore ai progetti:

```
    <label          for="project_color">Colore          del
progetto</label>

    <input type="color" name="colore" id="colore">
```

Infine, adesso vedremo come davvero velocizzare l'immissione dei dati posizionando il cursore dell'utente nel primo campo del modulo appena la pagina è caricata.

Tutto quello che dovremo fare è aggiungere l'attributo *autofocus* a qualsiasi campo modulo, come, ad esempio:

```
    <label for="nome">Nome</label>

    <input type="text" name="nome" autofocus id="nome">
```

È possibile avere un solo attributo di messa a fuoco automatica (*autofocus*) su una pagina affinché funzioni nel modo in cui è stato pensato. Se ce ne fosse più di uno, il

browser concentrerà il cursore dell'utente sull'ultimo campo del modulo con *autofocus*, rendendo inutile l'assegnazione degli altri campi.

La registrazione di un account

Il nostro sito del Ristorante Mario richiederà ad alcuni utenti di registrare un account e uno dei maggiori problemi con le registrazioni è che gli utenti continuano a provare a utilizzare password non sicure. Ma c'è la soluzione, useremo un *placeholder* (segnaposto) del testo per fornire agli utenti una piccola guida sui nostri requisiti per la password. Per motivi di coerenza, aggiungeremo il testo segnaposto anche agli altri campi.

Per farlo aggiungeremo semplicemente l'attributo segnaposto a ciascun campo di input, in questo modo:

```
<input id="email" type="email"
name="email" placeholder="utente@esempio.it">
```

Il markup dell'intero modulo è simile a questo, con testo segnaposto per ogni campo:

```
<form id="creare_account" action="/signup"
method="post">
  <fieldset id="registrazione">
  <legend>Crea un nuovo account</legend>
  <ol>
    <li>
        <label for="nome">Nome di battesimo</label>
        <input id="name" type="text" autofocus="true"
        name="nome" placeholder="'Tony'">
  </li>
  <li>
    <label for="cognome">Cognome</label>
    <input id="cognome" type="text"
      name="cognome" placeholder="'Chan'">
 </li>
<li>
<label for="email">E-mail</label>
```

I segnaposto (*placeholder*) possono aiutare gli utenti a capire cosa stai chiedendo loro di fare suggerendo il dato da inserire nel campo richiesto:

```
    <input id="email" type="email"
name="email" placeholder="email@esempio.it">
  </li>
<li>
  <label for="password">Password</label>
   <input id="password" type="password" name="password"
value=""            autocomplete="off"   placeholder="10
caratteri" />
  </li>
<li>
  <label for="conf_password">Conferma password</label>
        <input    id="conf_password"    type="password"
name="conf_password" value=""
autocomplete="off" placeholder="Digita di nuovo la tua
password" />
   </li>
  <li><input type="submit" value="Iscriviti"></li>
</ol>
</fieldset>
</form>
```

Qualcuno avrà osservato che abbiamo aggiunto l'attributo di completamento automatico ai campi della password nel modulo appena visto. Infatti HTML5 introduce un attributo di completamento automatico che indica ai browser Web di non tentare di compilare automaticamente i dati per il campo. Alcuni browser ricordano i dati che gli utenti hanno precedentemente digitato e, in alcuni casi, vorremmo che il browser non consentisse agli utenti di farlo, per varie ragioni, in questo caso ad esempio per evitare di suggerire le password utilizzate da altri utenti.
Dal momento che stiamo usando ancora una volta l'elemento dell'elenco ordinato per contenere i campi del modulo, aggiungeremo un po' di CSS di base per rendere il modulo graficamente un po' più gradevole, vediamo come:

```
   fieldset{
  width: 230px;
}
fieldset ol{
  list-style: none;
  padding:0;
  margin:3px;
}
fieldset ol li{
  margin:0 0 10px 0;
  padding:0;
}
fieldset input{
```

```
  display:block;
}
```

Da notare che nelle ultime due righe di codice abbiamo inserito le istruzioni per far in modo che gli input rimangano nella propria riga.
Adesso andremo oltre e vedremo come fare in modo che un utente del nostro sito modifichi alcune informazioni su se stesso senza dover passare ad un altro modulo. Tradizionalmente implementiamo la modifica sul posto osservando le aree di testo per i clic e sostituendo tali aree con campi di testo. Questi campi rimandano il testo modificato al server tramite Ajax. Il tag *contenteditable* di HTML5 si occupa automaticamente della parte di immissione dei dati. Dovremo ancora scrivere del JavaScript per inviare i dati al server in modo da poterli salvare, ma non dobbiamo più creare e attivare i moduli nascosti.
Mario ha pensato bene di consentire agli utenti di rivedere il profilo del proprio account, quindi il nome, il cognome, la città, lo stato e l'indirizzo email. Prima di iniziare, è bene sapere che l'implementazione di una funzionalità che si basa su JavaScript senza prima includere una soluzione lato server va contro tutto ciò che la bibbia del programmatore insegna e predica. Ovviamente lo stiamo facendo così perché vogliamo concentrarci sulle caratteristiche dell'attributo *contenteditable*. Quindi, essenzialmente la lezione da imparare sarebbe quella di creare sempre la soluzione che non richieda JavaScript, quindi creare la versione basata sugli script e infine assicurarsi di scrivere test automatici per entrambi i percorsi in modo da avere maggiori probabilità di rilevare bug se ne modificassimo una versione e non l'altra.

Utilizzare contenteditable

HTML5 prevede l'attributo *contenteditable* che è disponibile su quasi tutti gli elementi. La semplice aggiunta di questo attributo trasforma una porzione specifica della pagina in modificabile, ovviamente saremo noi a decidere quale specificando, dove previsto, che *contenteditable* sia attivo assegnandogli il valore booleano *true*. Vediamo l'esempio:

```
    <h1>Informazioni utente</h1>
    <div id="infout"></div>
    <ul>
   <li>
   <b>Nome</b>
     <span     id="nome"     contenteditable="true">Paolo
Rossi</span>
   </li>
<li>
   <b>Città</b>
   <span id="citta" contenteditable="true">Roma</span>
   </li>
<li>
   <b>Indirizzo</b>
   <span   id="ind"   contenteditable="true">Via   Cavour
12</span>
   </li>
<li>
   <b>Email</b>
<span                                          id="email"
contenteditable="true">prossi@gmail.com</span>
</li>
    </ul>
```

Sarà una buona idea quella di utilizzere alcuni selettori CSS3 per identificare i campi modificabili in modo che cambino colore quando i nostri utenti passano il mouse sopra o li selezionano:

```
    ul{list-style:none;}
li{clear:both;}
```

```
li>b, li>span{
  display: block;
  float: left;
  width: 105px;
}
li>span{
  widht: 520px;
  margin-left: 22px;
}
li>span[contenteditable=true]:hover{
  background-color: #ffc;
}
li>span[contenteditable=true]:focus{
  background-color: #ffa;
  border: 1px shaded #000;
}
```

Ora che abbiamo i dati dovremmo fare in modo di mantenerli per i prossimi accessi, altrimenti bisognerà ripetere ogni volta la registrazione.
Sebbene gli utenti possano modificare i dati, le modifiche andranno perse se aggiornano la pagina o escono. Abbiamo bisogno di un modo per inviare tali modifiche al nostro back-end e possiamo farlo facilmente attraverso la libreria di Javascript jQuery:

```
 $(function(){
var stato = $("#stato");
$("span[contenteditable=true]").blur(function(){
var campo = $(this).attr("id");
var valore = $(this).text();
$.post("http://localhost:1234/users/1",
   campo + "=" + valore,
   function(dati){
     status.text(dati);
   }
);
});
 });
```

Aggiungeremo un *listener* (ascoltatore) di eventi a ogni intervallo della pagina che avrà l'attributo *contenteditable* impostato su true. Quindi, tutto ciò che dovremmo fare è inviare i dati al nostro script lato server.

Quiz & esercizi

1) Il segnaposto mi servirà per migliorare l'accesibilità del sito?
2) Perché ho bisogno di indicare l'ID nei vari tag?
3) Completa questo codice:

```
<li><input type="submit" value="
```

4) Scrivere un listato dove un selettore CSS modifica la dimensione del campo modulo al passaggio del mouse.

5) Per attivare l'attributo *contenteditable* di cosa avrò bisogno?
6) Un valore booleano può essere negativo?

7) Cosa utilizzeremo per fare in modo che, all'avvio di una pagina, sia evidenziato un campo piuttosto che un altro?
8) In un modulo, se volessi inserire calendario dovrò modificare l'input id oppure l'input type? e cosa dovrò inserire tra virgolette?

Riassunto

In questo capitolo si è parlato di moduli o forme, di alcune istruzioni e attributi utili come contenteditable e anche dell'utilizzo del segnaposto, il quale dà un suggerimento agli utenti. Nel prossimo capitolo finalmente approfondiremo il CSS dato che finora lo abbiamo solo visto di sfuggita.

 # 3 - CSS3

Approfondiamo il CSS

Gli standard del Web

Il CSS per tutti i dispositivi

Nel primo capitolo ne abbiamo osservato le potenzialità. Nonostante ciò la maggior parte degli sviluppatori non sa o non vuole sapere che CSS3 ha alcuni eccellenti selettori utili ad applicare stili differenti in diverse aree delle nostre pagine web. Rammentiamo che un selettore è un modello che si usa per aiutarci a trovare elementi nel documento HTML in modo, come detto, da poterci applicare degli stili specifici. Per quale motivo si utilizzano poco? principalmente per pigrizia, oramai una gran parte dei siti hanno uno stile derivato da un mediocre template prefabbricato acquistato per pochi euro o addrittura gratuito. Ma a noi i template non piacciono tanto, quindi in questo paragrafo useremo questi selettori per definire lo stile di una tabella. Quindi daremo un'occhiata a come possiamo utilizzare alcune altre funzionalità CSS3 per migliorare i fogli di stile di stampa del nostro sito e divideremo il contenuto in più colonne.
Prima di tutto dovremo introdurre il concetto di *pseudoclasse*. La troveremo nel CSS, ed è un modo per selezionare elementi basati su informazioni che si trovano al di fuori del documento o che non possono essere espresse usando i normali selettori. Abbiamo già visto nell'ultimo esempio qui che le pseudoclassi come :*hover* ci sono servite per cambiare il colore di un collegamento quando l'utente ci passa sopra con il puntatore del mouse.

Osserveremo come ci renderanno molto più semplice la localizzazione degli elementi, e lo faremo sul campo servendoci di alcuni esempi.

Impostiamo lo stile di una ricevuta fiscale

Il nostro eclettico chef Mario si è evoluto e adesso spedisce alcuni prodotti confezionati come sughi pronti e marmellate in tutto il mondo. Avremo bisogno di creare un sistema che produca una ricevuta fiscale ad ogni acquisto in modo da poter gestire poi la contabilità (producendo la relativa fattura elettronica, ma questo non lo vedremo). Cercheremo di implementare il modello standard in HTML di una ricevuta migliorandone la leggibilità e lo stile. Nel foglio avremo i prezzi, le quantità, i totali, un subtotale, e un totale generale per l'ordine.

Il listato per la tabella standard potrebbe essere così:

```
    <table>
   <tr>
   <i>Prodotto</th>
   <i>Prezzo</th>
   <i>Quantità</th>
   <i>Totale</th>
</tr>
   <tr>
   <td>Sugo alla marinara</td>
   <td>€ 8,00</td>
   <td>3</td>
   <td>€ 24,00</td>
   </tr>
   <tr>
  <td>Marmellata di more</td>
  <td>€ 10,00</td>
  <td>4</td>
  <td>€ 40,00</td>
 </tr>
 <tr>
  <td>Spedizione</td>
 <td>€ 8,00</td>
  <td>1</td>
  <td>€ 8,00</td>
 </tr>
<tr>
```

```
  <td colspan="3">totale parziale</td>
  <td>€ 72,00</td>
 </tr>
<tr>
  <td colspan="3">IVA 22%</td>
  <td>€ 15,84</td>
 </tr>
<tr>
  <td colspan="3">Totale dovuto</td>
  <td>€ 87,84</td>
 </tr>
</table>
```

Abbiamo la nostra ricevuta, la quale ha certamente bisogno di un restyling. Inanzitutto sarebbe più facile da leggere se ogni altra riga fosse colorata in modo diverso, come già avevamo proggettato. Inoltre il *totale dovuto* dovrebbe avere un colore diverso in modo che risalti maggiormente.
Intanto, per prima cosa, elimineremo quell'orribile bordo predefinito della tabella, poi daremo anche uno stile all'intestazione dandogli uno sfondo nero con testo bianco:

```
    table{
     width: 800px;
border-collapse: collapse;
 }

    th, td{
   border: none;
}

    th{
  background-color: #000;
  color: #fff;
}
```

Adesso la nostra ricevuta si presenta più pulita, così potremo iniziare a usare le pseudoclassi per definire lo stile di singole righe e colonne.
Infatti, come già detto, aggiungeremo delle come sfondo di ogni riga delle strisce di due colori diversi. Questo tipo di stile viene eseguito al meglio in CSS, anche se precedentemente significava aggiungere nomi di classi aggiuntivi alle righe della tabella come "dispari" e "pari". Non vogliamo assolutamente inquinare il *markup* (marcatore) della nostra tabella in questo modo, perché la specifica HTML5 ci incoraggia a evitare di usare i nomi delle classi che definiscono la presentazione. Usando alcuni nuovi selettori, potremo ottenere ciò che vogliamo senza modificare affatto il

nostro *markup*, separando la presentazione dal contenuto.
Il selettore *nth-of-type* trova ogni *nth* elemento di un tipo specifico utilizzando una formula o parole chiave. A breve entreremo nella formula in modo più dettagliato, ma prima concentriamoci sulle parole chiave, perché sono immediatamente più facili da capire.
Troveremo ogni riga pari della tabella e le assegneremo un colore di sfondo. Quindi faremo la stessa cosa con le righe dispari. Questo è la maniera più semplice per cambiare lo sfondo delle righe della tabella con due colori diversi. Utilizzeremo le parole chiave *even* (pari) e *odd* (dispari) che CSS ci mette a disposizione:

```
    tr:nth-of-type(even){
    background-color: #F3F3F3;
}
tr:nth-of-type(odd) {
    background-color:#ddd;
}
```

Essenzialmente questo selettore dice: *trovami ogni riga pari della tabella e colorala, quindi trova ogni riga dispari e colora anche quella*. Non avremo quindi bisogno di ricorrere a script o nomi di classi extra sulle righe. Ora andremo a lavorare sull'allineamento delle colonne nella tabella.

Per impostazione predefinita, tutte le colonne nella nostra tabella della ricevuta sono allineate a sinistra. Le allineremo tutte a destra tranne la prima. In questo modo, le nostre colonne prezzo e quantità saranno allineate più facili da leggere. Per farlo, possiamo usare *nth-child*, ma prima dobbiamo imparare come funziona.
Il selettore *nth-child* cerca gli elementi figlio di un elemento e, come *nth-of-type*, può utilizzare parole chiave o anche una formula, come faremo.
La formula che utilizzeremo è *an+b*, dove *b* è l'offset (slittamento o avanzo rispetto al valore di riferimento)e *a* è un multiplo. Questa descrizione non è decisamente chiara senza un contesto, quindi esaminiamola nel concreto della nostra tabella.
Se volessimo selezionare tutte le righe della tabella, potremmo usare questo selettore:

```
    table tr:nth-child(n)
```

Non stiamo usando alcun multiplo, né stiamo usando un *offset,* cioè nessun valore al di fuori di quello di riferimento. Tuttavia, se volessimo selezionare tutte le righe della tabella ad eccezione della prima, cioè quella contenente le intestazioni di colonna, utilizzeremo questo selettore che utilizza un *offset*:

```
table tr:nth-child(n+2)
```

Invece se volessimo selezionare ogni altra riga della nostra tabella, useremmo un multiplo o *2n*. Potremmo anche usare l'offset in modo da poter iniziare più in basso nella tabella, infatti con la formula *2n+4* troveremmo ogni altra riga, a partire dalla quarta, come nel secondo esempio:

```
   table tr:nth-child(2n)
table tr:nth-child(2n+4)
```

Se volessimo selezionare ogni terza riga, useremmo *3n*.
Infine, potremmo allineare ogni colonna tranne la prima con questa regola:

```
   td:nth-child(n+2){
  text-align: right;
}
```

A questo punto, la nostra tavola prende davvero forma, Ora, stiliamo l'ultima riga della tabella.
La fattura sembra abbastanza buona in questo momento, ma uno dei gestori vorrebbe che la riga inferiore della tabella fosse più audace delle altre righe in modo che risalti di più. Possiamo usare *last-child* anche per quello, che afferra l'ultimo figlio di un gruppo.
Applicare un margine inferiore ai paragrafi in modo che siano distanziati uniformemente su una pagina è una pratica comune tra molti sviluppatori Web. Questo a volte può portare a un margine inferiore extra alla fine di un gruppo e ciò potrebbe essere indesiderabile. Ad esempio, se i paragrafi sono seduti all'interno di una barra laterale o di un riquadro di richiamo, potremmo voler rimuovere il margine inferiore dell'ultimo paragrafo in modo che non ci sia spazio sprecato tra la parte inferiore dell'ultimo paragrafo e il bordo del riquadro. Il selettore dell'ultimo figlio è lo strumento perfetto per questo. Possiamo usarlo per rimuovere il margine dall'ultimo paragrafo.

```
   p{ margin-bottom: 22px }
#sidebar p:last-child{ margin-bottom: 0; }
```

Usiamo questa stessa tecnica per mettere in grassetto il contenuto dell'ultima riga:

```
   tr:last-child {
  font-weight: bolder;
}
```

Facciamo la stessa cosa con l'ultima colonna della tabella. Questo aiuterà anche a risaltare i totali di linea:

```
   td:last-child {
  font-weight: bolder;
}
```

Infine, aumenteremo la dimensione del carattere del totale utilizzando l'ultimo *figlio* con i selettori discendenti. Troveremo l'ultima colonna dell'ultima riga e la modelleremo così:

```
   tr:last-child td:last-child{
  font-size:24px;
}
```

Abbiamo quasi finito, ma ci sono ancora alcune cose da fare con le ultime tre righe della tabella. Ad esempio potremmo evidenziare la riga di spedizione della tabella quando c'è una tariffa di spedizione scontata. Useremo l'ennesimo figlio per individuare rapidamente quella riga. Abbiamo visto come possiamo usare *nth-child* e la formula *an+b* per selezionare elementi figlio specifici qui. Il selettore dell'ennesimo figlio funziona esattamente allo stesso modo, tranne per il fatto che conta all'indietro attraverso i figli, a partire dall'ultimo figlio per primo. Ciò semplifica l'acquisizione del penultimo elemento in un gruppo. Si scopre che dobbiamo fare proprio questo con la nostra tabella delle fatture.
Quindi, per trovare la nostra riga di spedizione, useremmo questo codice:

```
   tr:nth-last-child(2){
  color: red;
}
```

Qui stiamo solo specificando un figlio specifico, il penultimo.
Rimane un piccolo dubbio sull'estetica. In precedenza,

abbiamo allineato a destra tutte le colonne tranne la prima colonna e, sebbene ciò sembri corretto per le righe della tabella con le descrizioni degli articoli e i prezzi, le ultime tre righe della tabella al momento sembrano un po' bruttine. Allineiamo a destra anche le tre righe inferiori. Possiamo farlo usando l'ennesimo figlio con un valore negativo per n e un valore positivo per a nella nostra formula, in questo modo:

```
    tr:nth-last-child(-n+3) td{
  text-align: right;
}
```

Possiamo vederlo come un selettore di intervallo, avrà l'offset (slittamento o avanzo rispetto al valore di riferimento) di 3 e poiché stiamo usando l'ennesimo figlio, recupererà ogni elemento prima dell'offset. Se stavamo usando *nth-child*, questa formula prenderebbe ogni riga fino all'offset.

In conclusione, la tabella con il nuovo stile sarà sicuramente molto meglio.

I CSS possono dare uno stile agli elementi esistenti, ma possono anche inserire contenuto in un documento. Ci sono alcuni casi in cui la generazione di contenuto con CSS ha senso e il più ovvio è aggiungere l'URL di un collegamento ipertestuale accanto al testo del collegamento quando un utente stampa la pagina. Quando osserviamo un documento sullo schermo, potremo semplicemente passare il mouse sopra un collegamento e vedere dove va guardando la barra di stato. Tuttavia, quando guarderemo la stampa di una pagina, non abbiamo assolutamente idea di dove vadano quei collegamenti. Questo potrebbe essere un problema per Mario il quale vorrebbe che il suo sito sia sempre in ordine e all'avanguardia, come la sua cucina.

La pagina stessa non ha altro che un elenco di collegamenti su di essa in questo momento:

```
    <ul>

    <li>

    <a     href="cucina/haccp.html">Istruzioni     sulla
compilazione dei moduli HACCP</a>

    </li>

    <li>
```

```
    <a  href="cucina/regolesanita.html">Regole  sanitarie
di base</a>
</li>
<li>
<a href="cucina/linkfornitori.html">Elenco fornitori</a>
</li>
</ul>
</body>
```

Come detto, se dovessimo stampare la pagina avremmo sicuramente dei problemi con la visualizzazione e il posizionamento corretto dei link, vediamo la possibile soluzione.

Applicare il CSS ad un'anteprima di stampa

Quando aggiungiamo un foglio di stile a una pagina, possiamo specificare il tipo di supporto a cui si applicano gli stili. Il più delle volte, utilizziamo il tipo di schermo. Tuttavia, possiamo utilizzare il tipo di stampa per definire un foglio di stile che viene caricato solo quando la pagina viene stampata (o quando l'utente utilizza la funzione di anteprima di stampa).

```
    <link        rel="stylesheet"        href="print.css"
type="text/css" media= "print">
```

Possiamo quindi creare un file di foglio di stile print.css con questa semplice regola:

```
    a:after {
content: " (" attr(href) ") ";
}
```

Queste istruzioni recuperano ogni collegamento sulla pagina e aggiungono il valore di *href* tra parentesi dopo il testo del collegamento. Quando lo stamperemo da un browser sulla pagina avremo solo i link che ci interessano.

Adeguarsi agli standard

Come per quasi tutto ciò che è presente online anche i siti internet hanno un loro standard e delle specifiche da seguire per rendere la navigazione un'ottima esperienza per tutti.
Accessibility for Rich Internet Applications (WIA-ARIA) è una specifica che fornisce modi per migliorare l'accessibilità dei siti internet e in particolare delle applicazioni Web. È decisamente utile se dovessimo sviluppare applicazioni con del codice aggiunto in JavaScript o altri linguaggi.

Alcune parti della specifica WIA-ARIA sono state integrate in HTML5, mentre altre rimangono separate e possono comunque essere integrate.
La maggior parte dei siti Web condivide una struttura comune: c'è un'intestazione, una sezione di navigazione, alcuni contenuti principali e un piè di pagina. Sono quasi tutti organizzati proprio così, in modo lineare. Sfortunatamente, questo significa che, ad esempio, uno screen reader potrebbe dover leggere il sito al suo utente in quell'ordine. Poiché la maggior parte dei siti ripete la stessa intestazione e navigazione su ogni pagina, l'utente dovrà ascoltare questi elementi ogni volta che visiterà un'altra pagina.
Potremmo risolvera in tanti modi, ma servirsi dell'attributo *role* (ruolo) di HTML5 ci consente di assegnare una "responsabilità" a ciascun elemento della nostra pagina. Quindi, uno screen reader potrà facilmente analizzare la pagina e classificare tutte queste responsabilità in modo da poter creare un semplice indice per la pagina. Ad esempio, potrà trovare tutti i ruoli di navigazione nella pagina e presentarli all'utente in modo che possa navigare rapidamente nell'applicazione.
Questi ruoli sono una specifica di WIA-ARIA3 incorporati in HTML5, sono divisi in ruoli di riferimento e ruoli del documento.

Ruoli di riferimento

I ruoli di riferimento identificano i "punti di interesse" sul nostro sito, come l'intestazione, l'area di ricerca o di navigazione. Ma vediamoli nel dettaglio:

Ruolo di riferimento	Utilizzo
application	Un'area della pagina che contiene un'applicazione Web.
article	Una composizione parte indipendente di un documento.
banner	Identifica l'area banner della pagina.
complementary	Identifica una pagina che integra il contenuto principale ma comunque è significativa di suo (complementare).
contentinfo	Identifica informazioni, tipo il copyright.
definition	Identifica una definizione di un termine o di un soggetto.
directory	Identifica un elenco di riferimenti a un gruppo, ad esempio un sommario. Utilizzato per contenuto statico.
document	Un'area che contiene il testo del documento.
heading	Identifica un'intestazione per una sezione di una pagina.
img	Una sezione che contiene elementi di

	un’immagine.
list	Identifica un gruppo di voci di elenco non interattive.
listitem	Un singolo membro di un gruppo di voci di elenco.
main	Identifica il contenuto principale della nostra pagina.
math	Identifica un’espressione matematica
navigation	Identifica gli elementi di navigazione sulla pagina.
note	Contenuto tra parentesi o accessorio.
presentation	Contenuto destinato alla presentazione.
row	Identifica una riga di celle in una griglia.
rowheader	Riga con informazioni di intestazione in una griglia.
search	Identifica l’area di ricerca della nostra pagina.

Possiamo applicare alcuni di questi ruoli al nostro sito di Mario su cui abbiamo lavorato usando il markup semantico. Per l’intestazione generale, applicheremo il ruolo *banner*:

```
<header id="intestazione" role="banner">
<h1>Ristorante Mario<h1>
</header>
```

Ecco come appare il listato con l'aggiunta di *role="banner"* all'esistente tag *intestazione*. Possiamo identificare la nostra navigazione allo stesso modo:

```
   <nav role="navigation">
  <ul>
  <li><a href="/">Ultimi post</a></li>
  <li><a href="archivi">Archivi</a></li>
  <li><a href="recensioni">Recensioni</a></li>
  <li><a href="contatti">Contattaci</a></li>
  </ul>
   </nav>
```

La specifica HTML5 dice che alcuni elementi hanno ruoli predefiniti e non possono essere sovrascritti. L'elemento *nav* deve avere il ruolo di navigazione e tecnicamente non ha bisogno di essere specificato.

Le aree principali e la barra laterale possono essere identificate come segue:

```
   <section id="post" role="main">
   </section>
<section id="barra laterale" role="complementary">
 <nav>
   <h3>Archivi</h3>
    <ul>
      <li><a href="2022/01">Gennaio 2022</a></li>
      <li><a href="2022/02">Febbraio 2022</a></li>
      <li><a href="2022/03">Marzo 2022</a></li>
      <li><a href="2022/04">Aprile 2022</a></li>
      <li><a href="2022/05">Maggio 2022</a></li></ul>
  </nav>
</section><!– barra laterale –>
```

Identifichiamo la pubblicazione e le informazioni sul copyright nel nostro piè di pagina:

```
   <footer id="pie_pagina " role="contentinfo">
  <p>&copy; Mario SRL 2022</p>
</footer><!– pie_pagina –>
```

Se avessimo una casella di ricerca nel sito, potremmo identificare anche quell'area.
Ora che abbiamo individuato i punti di riferimento, facciamo

un ulteriore passo avanti e iniziamo a riconoscere alcuni degli elementi del documento.
Molti dei ruoli del documento sono definiti implicitamente dal tag HTML, come articoli e intestazioni. Tuttavia, il ruolo del documento non lo è ed è molto utile, specialmente nelle applicazioni con un mix di contenuto dinamico e statico. Ad esempio, un client di posta elettronica basato sul Web può avere il ruolo del documento allegato all'elemento che contiene il corpo del messaggio di posta elettronica. Questo è utile perché i lettori di schermo hanno spesso metodi diversi per navigare usando la tastiera. Quando l'attenzione dell'utilità per la lettura dello schermo è su un elemento dell'applicazione, potrebbe essere necessario consentire la pressione dei tasti all'applicazione Web. Tuttavia, quando lo stato attivo è sul contenuto statico, potrebbe consentire alle associazioni di tasti dell'utilità per la lettura dello schermo di funzionare in modo diverso.
Possiamo applicare il ruolo come "documento" al nostro sito aggiungendolo all'elemento *body*:

```
<body role="document">
```

Questo può aiutare a garantire che questa pagina venga trattata come contenuto statico.
Adesso vediamo cosa ha di nuovo in mente Mario.

Creiamo una newsletter

Il nostro chef Mario è un vulcano e ha intenzione di pubblicare una newsletter per tutti i suoi clienti registrati. Il gestore dello spazio web ha deciso di inserire la newsletter sul sito intranet e sta pianificando di inviare e-mail ai clienti con un collegamento per visualizzare la newsletter nei loro browser.
Vorremmo che la newsletter assomigli a una vera e propria newsletter, con due colonne invece di una. Chi ha mai provato a dividere del testo in più colonne usando *div* e *float*, sa quanto può essere difficile. Il primo grande ostacolo che incontreremo è che dovremmo decidere manualmente dove dividere il testo. In un software come *InDesign* o *Illustrator*, è possibile "collegare" le caselle di testo in modo che quando una si riempie di testo, il testo scorre nell'area di testo collegata. Difficile trovare qualcosa di simile sul Web, ma potremmo prendere un elemento e dividere il suo contenuto in più

colonne, ognuna con la stessa larghezza. Inanzitutto partiremo con il markup per la newsletter, sarà in un HTML abbastanza semplice.

Poiché il suo contenuto cambierà una volta scritto, useremo solo del *testo segnaposto* (placeholeder) per il contenuto. Potremmo anche usare i nuovi elementi di markup HTML5 come *section* e simili per questa newsletter, ma li abbiamo già visti e quindi qui subentrate voi e, come esercizio, riscriverete il codice della newsletter utilizzando i nuovi elementi. Adesso vediamo il listato standard in HTML base:

```
    <body>
    <div id="intestazione">
    <h1>Newsletter di Mario</h1>
    <p>Numero 23</p>
    </div>
    <div id="newsletter">
    <div id="Le news di Mario">
    <div>
    <h2>Le nuove ricette</h2>
    </div>
    <div>
    <p>
    …
</p>
  <div class="callout">
  <h4>La tecnica del mese</h4>
<p>
"…"
</p>
</div>
<p>
…
</p>
</div>
</div>
    <div id="Il nostro personale">
<div>
<h2>Pino il maitre</h2>
</div>
<div>
```

```
<p>
…
</p>
<p>
…
</p>
</div>
</div>
</div>
<div id="piè di pagina">
  <h6>Contatto email di Chef Mario
     <a href="mailto: chefmario@ristorantemario.it">
chefmario@ristorantemario.it</a>.
 </h6>
</div>
</body>
```

Per dividerlo in un layout a due colonne, tutto ciò che dobbiamo fare è aggiungerlo al nostro foglio di stile:

```
   #newsletter{
  -moz-column-count: 2;
  -webkit-column-count: 2;
  -moz-column-gap: 25px;
  -webkit-column-gap: 25px;
  -moz-column-rule: 1px solid #ddccb5;
  -webkit-column-rule: 1px solid #ddccb5;
}
```

Ora sicuramente abbiamo qualcosa di molto più accattivante a livello grafico. Potremo aggiungere altri contenuti e il browser determinerà automaticamente come dividerli in modo uniforme. Inoltre, da notare come gli elementi flottanti si adattano alle colonne che li contengono.
Adesso andremo a vedere un aspetto molto importante inerente l'autosettaggio delle dimensioni dello schermo in base al dispositivo utilizzato dall'utente. Le *media query* di CSS3 ci consentiranno di modificare la presentazione di una pagina in base alle dimensioni dello schermo utilizzate dai nostri visitatori. Potremmo utilizzarle per determinare quanto segue:

- *Risoluzione* del dispositivo.
- *Orientamento* verticale o orizzontale del dispositivo.
- *Larghezza e altezza* del dispositivo.
- *Larghezza e altezza* della finestra del browser.

Per questo motivo, le media query ci consentiranno di creare facilmente fogli di stile alternativi per gli utenti mobili.
Chiaramente dovremo considerare il fatto che molti potrebbero

guardare le pagine Web tramite lo smartphone e quindi bisognerà ridefinire la pagina usando il markup semantico. Per fortuna lo faremo molto rapidamente.
Il sito attualmente ha un layout a due colonne, con un'area di contenuto principale e una barra laterale. Il modo più semplice per renderlo più leggibile sugli smartphone è rimuovere gli elementi mobili in modo che la barra laterale cada sotto il contenuto principale. In questo modo, il lettore non dovrà scorrere lateralmente sul dispositivo.
Per fare in modo che funzioni, aggiungeremo questo codice in fondo al foglio di stile del nostro sito:

```
   @media only screen and (max-device-width: 720px) {
 body{
    width: 690px;
 }
section#sidebar, section#posts{
  float: none;
  width: 100%;
  }
}
```

Questa porzione di codice si attiverà quando la condizione tra le parantesi si avvererà, ossia un dispositivo con schermo da 720 pixel. In questo caso, verrà ridimensionato il corpo della pagina e trasformeremo il nostro layout a due colonne in uno a colonna singola.
Potremmo anche usare le media query quando colleghiamo il foglio di stile, così possiamo mantenere il nostro foglio di stile mobile in un file separato, come questo:

```
   <link rel="stylesheet" type="text/css"
href="CSS/mobile.css" media="only screen and (max-
device-width: 720px)">
```

Con ciò, il nostro sito diventa immediatamente più leggibile sui dispositivi mobili. Ovviamente potremo utilizzare questo approccio creando fogli di stile per tablet o dispositivi smart con display di varie dimensioni, in modo che il contenuto sia sempre leggibile.

Quiz & esercizi

1) Come già accennato riscrivete il listato della newsletter utilizzando gli elementi di HTML 5.
2) Cosa viene utilizzato per applicare correttamente uno stile CSS solo ad una precisa porzione della pagina?

3) Qual è lo scopo dei selettori in CSS? scrivere un listato dove se ne usa almeno uno.
4) Cosa indica questo codice: `<section id="" role="complementary">` ?

5) Perché devo scrivere i ruoli nella pagina Web?

6) Posso utilizzare la media query per migliorare un'anteprima di stampa?

7) Scrivere un listato con del testo di almeno 6 righe che possa essere visualizzato in uno schermo di un dispositivo smart, con il display di 200x200 pixel

Riassunto

Abbiamo iniziato con i tag fondamentali di ogni pagina Web, i quali serviranno ad avere un sito conforme a tutti gli standard. Abbiamo anche visto le prime righe di CSS, è solamente l'inizio ed è sostanzialmente semplice, quindi impariamo bene queste semplici regole di base.
Molto importanti sono anche le specifiche WIA-ARIA3 utili migliorare l'accesibilità alle nostre pagine Web.

 # 4 - La grafica con HTML5

Disegnare con canvas

Miglioramenti grafici

Font e grafici

Un'altra funzionalità di HTML 5 che pochi utilizzano, come i pseudoselettori del CSS3, riguarda l'inaspettata nonché divertente possibilità di disegnare (letteralmente). Infatti solitamente, se avessimo bisogno di un'immagine in un'applicazione Web, ci serviremo di un software grafico qualsiasi, creeremmo o editeremmo un'immagine e la incorporeremmo nella nostra pagina con un tag *img*. In caso volessimo qualcosa di più movimentato potremmo usufruire di Three.Js, una libreria Javascript per la grafica 3D. Ma restiamo sul bidimensionale, in HTML5 possiamo avvalerci dell'elemento *canvas* il quale ci consentirà di creare immagini e anche animazioni nel browser servendoci sempre di JavaScript. Potremo utilizzare l'elemento *canvas* per creare forme semplici o complesse o anche creare grafici e diagrammi senza ricorrere a librerie lato server, il desueto Flash o altri plug-in. Proveremo entrambe queste cose fra poco.
Per prima cosa acquisiremo familiarità con il modo in cui utilizziamo JavaScript e l'elemento *canvas* insieme disegnando alcune semplici forme mentre costruiamo una versione del logo del Ristorante Mario. Quindi ci impegneremo per rendere il nostro sito più accattivante graficamente in alcuni dettagli a prima vista insignificanti.

Infine utilizzeremo una libreria di grafici progettata specificamente per lavorare con l'area di disegno per creare un grafico a barre, ad esempio, delle statistiche del browser.

Disegnamo sulla tela (canvas)

L'elemento *canvas* è un contenitore molto simile all'elemento *script*. È letteralmente una tabula rasa sulla quale potremo lavorare. Iniziamo definendo una tela (*canvas*) con una larghezza e un'altezza, come questa:

```
<canvas id="tela" width="180" height="180">
```

Ricordiamoci che non potremo usare CSS per controllare o alterare la larghezza e l'altezza di un elemento *canvas* senza

distorcerne il contenuto, quindi dovremo avere ben chiare fin dall'inizio le sue dimensioni.
Ci serviremo di JavaScript per mettere le forme sulla nostra tela, anche qui ricordiamoci di inserire queste perzioni di codice all'interno dell'elemento <*script*>:

```
   var tela = document.getElementById('tela');
var contesto = tela.getContext('2d');
```

Una volta che abbiamo il contesto della tela, aggiungeremo semplicemente elementi a quel contesto. Per aggiungere una casella di un bel color lavanda, imposteremo il colore di riempimento e quindi creeremo la casella:

```
   contesto.fillStyle = "rgb(34,113,179)";
contesto.fillRect (10, 10, 100, 100);
```

Il contesto 2D della tela è una griglia, con l'angolo in alto a sinistra come origine predefinita. Quando creeremo una forma, specificheremo le coordinate X e Y iniziali, la larghezza e l'altezza. Ogni forma viene aggiunta al proprio livello, quindi potremo creare due quadrati con una dimensione di 10 pixel, in questo modo:

```
   contesto.fillStyle = "rgb(34,113,179)" ;
contesto.fillRect (10, 10, 100, 100);
contesto.fillStyle = "rgb(0,200,0)" ;
contesto.fillRect (20, 20, 100, 100);
```

Oppure per dei rettangoli:

```
   contesto.fillStyle = "rgb(34,113,179)";
contesto.fillRect (10, 10, 100, 50);
contesto.fillStyle = "rgb(34,113,179)" ;
contesto.fillRect (10, 10, 100, 50);
contesto.fillStyle = "rgb(0,200,0)" ;
contesto.fillRect (0, 20, 100, 50);
```

Ed ecco come appariranno le due prove:

Questa è la base di partenza, adesso proveremo a disegnare il logo del Ristorante a cui Mario tiente tanto, dovrà essere un eccellente lavoro, altrimenti rischieremo di essere licenziati.
Il logo è costituito da una stringa di testo, un percorso diciamo un po' arzigogolato, e da due triangoli. Iniziamo creando un nuovo documento HTML5, aggiungendo un elemento *canvas* alla pagina, e quindi creando una funzione JavaScript per disegnare il logo, che rileva se possiamo utilizzare il canvas 2D.

```
   var disegna = function(){
var tela = document.getElementById('logo');
var contesto = tela.getContext('2d');
};
```

Invochiamo questo metodo dopo aver prima verificato l'esistenza dell' elemento tela, in questo modo:

```
   $(function(){
 var tela = document.getElementById('logo');
 if (tela.getContext){
   disegna();
 }
});
```

Si noti che qui stiamo usando di nuovo la funzione jQuery per garantire che l'evento si attivi quando il documento è pronto. Stiamo cercando un elemento nella pagina con l'ID del logo, quindi è meglio assicurarsi di aggiungere il nostro elemento canvas al documento in modo che possa essere trovato e il nostro rilevamento funzionerà.

```
   <canvas id="logo" width="380" height="100">
  <h1>Ristorante Mario</h1>
</canvas>
```

Successivamente, aggiungiamo il testo "Ristorante Mario" alla tela.
L'aggiunta di testo alla tela implica la scelta di un carattere, una

dimensione del carattere e un allineamento, quindi l'applicazione del testo alle coordinate appropriate sulla griglia. Possiamo aggiungere il testo "Ristorante Mario" alla nostra tela in questo modo, specificando il font (abbiamo scelto Verdana), la grandezza (29 pixel), e lo stile (qui abbiamo il grassetto e anche il corsivo, cioè *bold* e *italic*):

```
   contesto.font = 'bold italic 29px verdana;
contesto.textBaseline = 'top';
contesto.fillText('Ristorante Mario', 75, 0);
```

Definiamo il tipo di testo e ne impostiamo la linea di base, o allineamento verticale, prima di applicarlo all'area di disegno. Stiamo usando il metodo *fillText* in modo da ottenere il testo dipinto con il colore di riempimento e lo stiamo impostando a 75 pixel a destra in modo da poter fare spazio al a quello che dovrebbe essere il logo vero e proprio che disegneremo in seguito.
Tracciamo linee sulla tela come per il gioco "collega i punti". Specifichiamo un punto di partenza sulla griglia dell'area di lavoro e quindi specifichiamo punti aggiuntivi in cui spostarci. Mentre ci muoviamo sulla tela, i punti vengono collegati. Usiamo il metodo *beginPath()* per iniziare a disegnare una linea, quindi creiamo il nostro percorso, in questo modo:

```
   contesto.lineWidth = 4;
contesto.beginPath();
contesto.moveTo(-90, 20);
contesto.lineTo(45, 2);
contesto.lineTo(70, 35);
contesto.lineTo(360, 35);
contesto.stroke();
contesto.closePath();
```

Quando avremo finito di spostarci sulla tela, chiameremo il metodo per disegnare la linea e infine attraverso il metodo *closePath* finiremo il disegno.
Adesso cercheremo di creare un'immagine stilizzata che dovrà poi essere il logo del ristorante.
L'idea è quella di disegnare due triangoli a specchio, uno trasparente e l'altro colorato internamente. Quando disegniamo forme e percorsi, possiamo specificare le coordinate X e Y dall'origine nell'angolo in alto a sinistra della tela, ma possiamo anche spostare l'origine in una nuova posizione.
Iniziamo a divertirci, vediamo cosa uscirà fuori:

```
    contesto.save();
contesto.translate(0,0);
contesto.fillRect(0,20,80,4);
```

Notiamo che prima di spostare l'origine, chiamiamo il metodo *save()*. Questo salva lo stato precedente della tela in modo da poterlo ripristinare facilmente. È come un punto di ripristino e possiamo pensarlo come una pila. Ogni volta che chiameremo save(), otterremo una nuova voce. Quando avremo finito, chiamando *restore(),* ripristineremo l'ultimo il punto di salvataggio.
Ora proviamo a disegnare il triangolo, cercando di creare un effetto ombra rispetto alla figura opposta:

```
    contesto.lineWidth = 4;
contesto.beginPath();
contesto.moveTo(80, 37);
contesto.lineTo(0, 5);
contesto.lineTo(0, 20);
contesto.lineTo(0, 35);
contesto.fill();
contesto.closePath();
contesto.restore();
```

Come visto disegniamo le nostre linee e, poiché in precedenza abbiamo spostato l'origine, siamo relativi all'angolo in alto a sinistra di ciò che abbiamo appena disegnato. E ora, aggiungiamo un po' di colore.

Aggiungiamo il colore

Spostando l'origine, nella pagina precedente, abbiamo visto brevemente come impostare il colore del tratto e del riempimento per gli strumenti di disegno. Potremmo impostare il colore di tutto sul verde azzurro semplicemente aggiungendo questo codice prima di disegnare qualsiasi cosa:

```
    contesto.fillStyle = "#256d7b" ;
contesto.strokeStyle = "#256d7b" ;
```

Ma è un po' noioso. Possiamo creare gradienti (sfumature) e assegnarli a tratti e riempimenti in questo modo:

```
    contesto.fillStyle = "#256d7b" ;
contesto.strokeStyle = "#256d7b" ;
var gradiente = contesto.createLinearGradient(0, 0, 0,
40);
gradiente.addColorStop(0, '#256d7b'); // verde scuro
gradiente.addColorStop(1, '#66ff00'); // verde chiaro
contesto.fillStyle = gradiente;
contesto.strokeStyle = gradiente;
```

Creiamo semplicemente un oggetto sfumatura e impostiamo le interruzioni di colore della sfumatura. In questo esempio, stiamo solo passando tra due tonalità di verde, ma potremmo creare un arcobaleno se lo volessimo.
Ricordiamoci che dovremo impostare il colore delle cose prima di disegnarle. Ed ecco il risultato ottenuto:

A questo punto, ci renderemo conto di aver assolutamente bisogno di un vero grafico, dato che il logo ottenuto, possiamo dire, è un discreto obbrobrio. A parte il discutibile risultato abbiamo visto come agire direttamente a livello grafico, e con un po' di esercizio e di occhio potremo sicuramente ottenere di meglio. Osserviamo adesso come apportare miglioramenti su tutti gli altri elementi dato che CSS3 ci offre diversi modi per farlo. Potremo utilizzare i nostri caratteri personalizzati sulle pagine, creare elementi con angoli arrotondati e ombre esterne, usare i gradienti come sfondi e persino ruotare gli elementi in modo che le cose non appaiano così a blocchi e noiose tutto il tempo. Possiamo fare tutte queste cose senza ricorrere a Photoshop o ad altri programmi di grafica. Inizieremo ammorbidendo l'aspetto di un modulo arrotondando alcuni angoli. Quindi, costruiremo un prototipo di banner per una fiera imminente, dove impareremo come aggiungere ombre, rotazioni, gradienti e opacità. Infine, parleremo di come utilizzare la funzione *@font-face* di CSS3 in modo da poter utilizzare caratteri più accattivanti.
Qualche anno fa sul Web era tutto un rettangolo per impostazione predefinita. I campi modulo, le tabelle e persino le sezioni delle pagine web avevano tutti un aspetto spigoloso e a blocchi.
Quindi molti designer si sono rivolti a tecniche diverse nel corso degli anni per aggiungere angoli arrotondati a questi elementi per ammorbidire un po' l'interfaccia, poi con CSS3 che ha il supporto per arrotondare facilmente gli angoli tutto è diventato più semplice.

Diamo una limatina agli angoli delle finestre

Adesso proveremo ad applicare dei miglioramenti grafici al nostro sito, per prima cosa daremo una limatina agli angoli della finestra di login degli utenti registrati. Giriamo quegli angoli usando prima solo CSS3. Per il modulo di accesso, utilizzeremo un codice HTML molto semplice:

```
 <form action="/login" method="post">
<fieldset id="login">
<legend>Accedi</legend>
<ol>
  <li>
    <label for="email">E-mail</label>
    <input id="email" type="email" name="email">
  </li>
  <li>
  <label for="password">Password</label>
  <input id="password" type="password"
        name="password" value="" autocomplete="off"/>
</li>
<li><input type="submit" value="Accedi"></li>
</ol>
</fieldset>
 </form>
```

Modelleremo un po' il modulo per dargli un aspetto leggermente migliore con il CSS:

```
 fieldset{
widht: 230px;
border: none;
background-color: #ddd;
 }
 fieldset legend{
background-color: #ddd;
padding: 0 65px 0 2px;
 }
 fieldset>ol{list-style: none;
padding:0;
margin: 2px;
```

```
    }
    fieldset>ol>li{
  margin: 0 0 10px 0;
  padding: 0;
    }
    fildset input{
  display:block;
    }
    input{
  widht: 210px;
  background-color: #fff;
  border: 1px solid #bbb;
    }
input[type="invia"]{
  widht: 205px;
  padding: 0;
  background-color: #bbb;
}
```

Notiamo subito che con l'istruzione relativa al campo input *display:block* faremo in modo che non ci siano altri elementi sulla stessa riga. Inoltre, questi stili di base rimuovono i punti elenco e assicurano che i campi di input siano tutti della stessa dimensione. Con tutto ciò possiamo quindi applicare gli effetti di arrotondamento ai nostri elementi.
Ovviamente gli angoli arrotondati sono solo il primo passo di ciò che possiamo fare con CSS3. Possiamo aggiungere ombre esterne agli elementi per farli risaltare dal resto del contenuto, usare i gradienti per rendere gli sfondi più definiti e usare le trasformazioni per ruotare gli elementi. Metteremo insieme diverse di queste tecniche per simulare un banner pubblicitario che Mario inserirà nel proprio sito per promuovere i prodotti che vende online. Non useremo Photoshop, faremo un *badge* (un elemento che serve per risaltare qualcosa), l'ombra e persino la trasparenza tutto in CSS. L'unica cosa di cui avremo bisogno è un immagine di sfondo.

Il banner

Sappiamo bene che oramai i banner praticamente non si usano più, tuttavia questo esercizio ci sarà utile per conoscere alcuni concetti. Iniziamo marcando la struttura di base della pagina in HTML:

```
   <div id="pubblicità">
 <section id="prod">
   <h3>Mario vi propone:</h3>
   <h2>Marmellate e Sughi pronti</h2>
</section>
   <section id="informazioni">
 </section>
</div>
```

Possiamo modellare le basi con questo (l'immagine *mario1.jpg* la dovrete aggiungere voi a vostro piacimento):

```
   #pubblicità{
  background-color: #000;
  width: 1080px;
  float: left;
  background-image:url('immagini/mario1.jpg');
  background-position: center;
  height: 260px;
}
   #badge{
text-align: center;
width: 200px;
border: 2px solid blue;
}
   #Informazioni{
margin: 20px;
padding: 20px;
width: 680px;
height: 180px;
}
   #badge, #info{
float: left;
background-color: #fff;
}
   #badge h2{ margin: 0;
color: red;
font-size: 41px;
}
   #badge h3{
margin: 0;
background-color: blue;
color: #fff;
}
```

Una volta applicato il foglio di stile alla nostra pagina, abbiamo il nostro *badge* e la regione dei contenuti visualizzati fianco a fianco, nella pagina seguente, quindi iniziamo a modellare.
Possiamo aggiungere definizione al *badge* cambiando lo sfondo

bianco con una sfumatura sottile che va dal bianco al grigio chiaro.

```
   #badge{
  background-image:-moz-linear-gradient(
    top, #fff, #efefef
);
background-image:-webkit-gradient(
  linear, left top, left bottom,
  color-stop(0, #fff),
  color-stop(1, #efef)
);
background-image: linear-gradient(
   top, #fff, #efefef
 );
}
```

Nel nostro esempio, dobbiamo solo passare dal bianco al grigio, ma se avessimo la necessità di aggiungere altri colori, dovremmo solo impostare un'altra interruzione di colore nella definizione.

Aggiungere l'ombra

Possiamo facilmente far sembrare che il badge sia posizionato sopra il banner aggiungendo un'ombra esterna. Tradizionalmente, faremmo questa ombra in Photoshop aggiungendola all'immagine o inserendola come immagine di sfondo. Tuttavia, la proprietà *box-shadow* di CSS3 ci consente di definire rapidamente un'ombra sui nostri elementi.
Applicheremo questa regola al nostro foglio di stile per dare un'ombra al badge:

```
   #badge{
  -moz-box-shadow: 5px 5px 5px #333;
  -webkit-box-shadow: 5px 5px 5px #333;
  -o-box-shadow: 5px 5px 5px #333;
  box-shadow: 5px 5px 5px #333;
}
```

La proprietà *box-shadow* ha quattro parametri. Il primo è l'offset orizzontale. Un numero positivo significa che l'ombra cadrà a destra dell'oggetto; un numero negativo significa che cade a sinistra. Il secondo parametro è l'offset verticale. Con l'offset verticale, i numeri positivi fanno apparire l'ombra sotto il riquadro, mentre i valori negativi fanno apparire l'ombra sopra l'elemento.
Il terzo parametro è il raggio di sfocatura. Un valore di 0 fornisce un valore molto nitido e un valore più alto rende

l'ombra più sfocata. Il parametro finale definisce il colore dell'ombra.
Dovremmo giocare un po' con questi valori per farci un'idea di come funzionano, così da trovare quelli più appropriati per noi. Quando lavoriamo con le ombre, dovremmo prenderci un momento per indagare su come funzionano realmente le ombre nel mondo fisico. Proviamo una torcia e puntiamola sugli oggetti, oppure usciamo ad osservare come il sole proietta ombre su ciò che ci circonda. Questo uso della prospettiva è importante, perché la creazione di ombre incoerenti può rendere l'interfaccia più confusa, soprattutto se si applicano le ombre a più elementi in modo errato. L'approccio più semplice che possiamo adottare è utilizzare le stesse impostazioni per ogni ombra che creeremo.
Potremo anche aggiungere l'ombra al testo allo stesso modo che abbiamo appena visto:

```
h1{text-shadow: 2px 2px 2px #bbbbbb;}
```

Le trasformazioni

le cosiddette "trasformazioni" di CSS3 ci serviranno per ruotare, ridimensionare e inclinare gli elementi proprio come faresti con programmi di grafica vettoriale come Flash, Illustrator o Inkscape. Questo può aiutare a far risaltare un po' di più gli elementi ed è un altro modo per creare un la pagina web non sembra così "quadrata".
Ruotiamo leggermente il badge in modo che fuoriesca dal bordo dritto del banner.

```
   #badge{
 -moz-transform: rotate (-7,5deg);
 -o-transform: rotate(-7,5deg);
 -webkit-transform: rotate(-7,5deg);
 -ms-transform:rotate(-7,5deg);
 transform: rotate(-7,5deg);
}
```

La rotazione con CSS3 è piuttosto semplice. Tutto quello che dobbiamo fare è fornire il grado di rotazione e il rendering funziona. Vengono ruotati anche tutti gli elementi contenuti all'interno dell'elemento che ruotiamo.
Ruotare è facile come arrotondare gli angoli, ma non ne abusiamo. L'obiettivo della progettazione dell'interfaccia è renderla utilizzabile. Se ruotiamo elementi che contengono

molti contenuti, assicuriamoci che i nostri spettatori possano leggere il contenuto senza girare la testa troppo in una direzione!

Lo sfondo trasparente

I grafici hanno utilizzato livelli semitrasparenti dietro il testo per un po' di tempo e questo processo di solito comporta la creazione di un'immagine completa in Photoshop o la sovrapposizione di un PNG trasparente sopra un altro elemento con CSS. CSS3 ci consente di definire i colori di sfondo con una nuova sintassi che supporta la trasparenza.
Chi ha un po' di familiarità con lo sviluppo Web, saprà come definire i colori usando codici esadecimali. Si può definire la quantità di rosso, verde e blu usando coppie di numeri, dove 00 è "tutto spento" o "niente" e FF è "tutto acceso". Quindi, ad esempio, il colore rosso sarebbe FF0000 o "tutto acceso per il rosso, tutto spento per il blu e tutto spento per il verde".
CSS3 introduce le funzioni RGB e RGBA. La prima funziona come la controparte esadecimale, ma utilizza valori da 0 a 255 per ogni colore. Quindi il colore rosso sarà rgb(255,0,0).
La funzione RGBA funziona all'incirca allo stesso modo, ma ci vuole un quarto parametro per definire la quantità di opacità, da 0 a 1. Se usiamo 0, non vedremo alcun colore, perché sarà completamente trasparente. Per rendere semitrasparente la casella bianca, aggiungeremo questa regola di stile:

```
   #Info{
 background-color: rgba(255,255,255,0,95);
}
```

Quando si lavora con valori di trasparenza come questo, le impostazioni di contrasto degli utenti a volte possono influire sull'aspetto risultante, quindi assicuriamoci di sperimentare il valore e di controllare su display differenti per assicurarci di ottenere un risultato coerente.
Mentre lavoriamo con la sezione delle informazioni del nostro banner, giriamo un po' gli angoli:

```
   #Info{
  moz-border-radius: 12px;
  web kit-border-radius: 12px;
  o-border-radius: 12px;
  border-radius: 12px;
}
```

Oltre all'aspetto grafico dovremo dare importanza anche alla scelta di un font. Tutti i libri che abbiamo letto hanno caratteri selezionati accuratamente da persone che capiscono come la loro scelta assieme a quella sulla spaziatura giusta possa rendere più facile o più invitante la lettura. Questi concetti ovviamente sono altrettanto validi nel Web.
I caratteri che sceglieremo per trasmettere un messaggio agli utenti influenzerà il modo in cui verrà interpretato.

I font

Alcuni font non sono gratuiti. Come per le fotografie stock o altro materiale protetto da copyright, siamo tenuti a rispettare i diritti e le licenze del materiale che utilizzeremo sul nostro sito Web. Se acquisteremo un font, di solito avremo il diritto di usarlo nel nostro logo e nelle immagini sulle nostre pagine. Questi sono chiamati diritti di utilizzo. Tuttavia, l'approccio *@font-face* mette in gioco un diverso tipo di licenza: i diritti di ridistribuzione.
Quando incorporiamo un font in una pagina, gli utenti dovranno scaricare quel font, il che significa che il nostro sito sta distribuendo questo font ad altri. Dovremo essere assolutamente sicuri che i caratteri che stiamo utilizzando consentano la distribuzione.
Typekit di Adobe ha una vasta libreria di font gratuiti disponibili e inoltre fornisce strumenti e codice che semplifica l'integrazione con il nostro sito Web. Non è tutto gratuito, ma la gran parte è abbastanza conveniente tutto sommato.
Google fornisce l'API di *Google Font*, che è simile a Typekit ma contiene solo font open source, in totale ne conta circa 800.
Entrambi questi servizi utilizzano JavaScript per caricare i caratteri, quindi dovremo assicurarci che i nostri contenuti siano facili da leggere per tutti gli utenti.
Essenzialmente, finché ci ricorderemo di trattare i caratteri come tutti le altre risorse, non dovremmo incorrere in nessun problema.
L'implementazione di Microsoft utilizzava un formato di carattere chiamato Embedded OpenType (EOT) e la maggior parte dei caratteri oggi sono in formato TrueType o OpenType.
Il direttore del marketing di del sito di Mario ha deciso che l'azienda dovrebbe standardizzare un carattere sia per la stampa che per il Web. Proveremo ad utilizzarne uno chiamato *Rubik*

(http://www.fontsquirrel.com/fonts/rubik), un font semplice che è completamente gratuito per uso commerciale. Come prova, applicheremo questo font al sito di Mario. In questo modo, potremo vedere il carattere in azione.
Il carattere che stiamo esaminando è disponibile su FontSquirrel nel formato TrueType, che dovrebbe andare più che bene.
L'uso del carattere prevede due passaggi: la definizione del carattere e l'associazione del carattere agli elementi. Nel foglio di stile del sito aggiungiamo questo codice:

```
   @font-face {
 font-family: 'Rubik-Regular';
 src: url('fonts/Rubik-Regular.ttf);
format('truetype'),


   font-weight: normal;
}
```

Definiamo prima la famiglia di caratteri, assegnandole un nome e quindi fornendo le origini dei caratteri. Inseriamo prima la versione Embedded OpenType, se presente, quindi forniamo gli altri font. Il browser di un utente continuerà a cercare i font finché non ne troverà uno che funzioni.
Ora che abbiamo definito la famiglia di caratteri, possiamo usarla nel nostro foglio di stile. Cambieremo il nostro stile del carattere originale in modo che assomigli a questo:

```
   body{
 font-family: "Rubik-Regular";
}
```

Con questa semplice modifica, il testo della nostra pagina verrà visualizzato con il nuovo carattere.

Visualizzare un grafico

Dopo tutto questo lavoro Mario vorrebbe conoscere alcune statistiche del suo sito Web e il modo migliore per farlo sarà quello di visualizzare un grafico con le statistiche. I programmatori back-end saranno in grado di ottenere i dati in tempo reale, ma prima vorrebbero vedere se è possibile trovare un modo per visualizzare il grafico nel browser, quindi ci hanno fornito alcuni dati di test.
Ci sono molti modi per disegnare grafici su una pagina Web.

Gli sviluppatori hanno usato spesso Flash per i grafici, ma ciò ha il limite di non funzionare su alcuni dispositivi mobili come gli smartphone. Esistono soluzioni lato server che funzionano bene, ma potrebbero richiedere un utilizzo eccessivo del processore se si lavora con dati in tempo reale. Una soluzione lato client basata su standard come l'area di disegno è un'ottima opzione. Abbiamo già visto come disegnare quadrati, ma fare qualcosa di più complesso richiederà sicuramente JavaScript. Abbiamo bisogno di una libreria grafica che ci aiuti.
La libreria *RGraph* rende ridicolmente semplice disegnare grafici usando l'area HTML5. È una pura soluzione in JavaScript, tuttavia, non funzionerà per chi lo ha disabilitato, ma in questo caso, nemmeno il canvas.
Ecco il codice per un grafico a barre molto semplice:

```
   <canvas   width="520"   height="260"   id="prova">[no
canvas support]</canvas>

   <script type="text/javascript" charset="utf-8">
  var grafico = new RGraph.Bar('prova', [55,20,13,12]);
  grafico.Set('chart.gutter', 55);
  grafico.Set('chart.colors', ['red']);
  grafico.Set('chart.title', "I miei sport preferiti");
   grafico.Set('chart.labels', ["Calcio" , "Pugilato" ,
"Ciclismo" , "Rugby" ]);
  grafico.Draw();
</script>
```

Tutto quello che dobbiamo fare è creare un paio di array (insiemi) con JavaScript e la libreria disegnerà per noi il grafico sulla tela.
Potremmo codificare i valori per le statistiche del browser nel codice JavaScript, ma solo gli utenti con JavaScript sarebbero in grado di vedere i valori. Invece, mettiamo i dati direttamente sulla pagina come testo. Possiamo leggere i dati con JavaScript e inviarli alla libreria grafica in un secondo momento:

```
   <div id="dati">
  <h1>Console in Italia</h1>
   <ul>
     <li>
      <p data-name="Playstation 5" data-percent="28">
         Playstation 5 - 32%
     </p>
  </li>
<li>
   <p data-name="Playstation 4" data-percent="22">
      Playstation 4 - 29%
   </p>
```

```
 </li>
<li>
  <p data-name="Xbox" data-percent="25">
    Xbox - 20%
   </p>
 </li>
<li>
    <p data-name="Nintendo" data-percent="20">
       Nintendo - 20%
    </p>
  </li>
</ul>
</div>
```

Utilizziamo gli attributi dei dati HTML5 per memorizzare i nomi delle console e le percentuali (assolutamente inventate). Sebbene disponiamo di tali informazioni nel testo, è molto più facile lavorarci a livello di codice poiché non dovremo analizzare le stringhe. Ora, trasformiamo questo markup in un grafico.

Utilizzeremo un grafico a barre, quindi dovremo richiedere sia la libreria del grafico a barre RGraph che la libreria principale RGraph. Useremo anche jQuery per estrarre i dati dal documento. Nella sezione head della pagina HTML, dobbiamo caricare le librerie di cui abbiamo bisogno:

```
   <script type="text/javascript"
                                          charset="utf-8"
src="http://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jq
uery.min.js">
</script>
<script src="javascripts/RGraph.common.js" ></script>
<script src="javascripts/RGraph.bar.js" ></script>
```

Per costruire il grafico, dobbiamo prendere il titolo del grafico, le etichette ei dati dal documento HTML e passarlo alla libreria RGraph. RGraph accetta gli array sia per le etichette che per i dati. Possiamo usare jQuery per costruire rapidamente quegli array.

```
   function disGraf(){
 var titolo = $('#dati h1' ).text();
 var et = $("#dati>ul>li>p[data-name]").map(function(){
   return$(this).attr("data-name");
});
 var p = $("#dati>ul>li>p[data-percent]").map(function(){
   return parseInt($(this).attr("data-name"));
});
 var grafico =new RGraph.Bar('console', p);
 grafico.Set('chart.gutter', 50);
 grafico.Set('chart.colors', ['green' ]);
```

```
 grafico.Set('chart.title', titolo);
 grafico.Set('chart.labels', et);
 grafico.Draw();
}
```

Innanzitutto, nella seconda riga prendiamo il testo per l'intestazione. Quindi, selezioniamo tutti gli elementi che hanno l'attributo *data-name*. Usiamo la funzione *map* di jQuery per trasformare i valori di quegli elementi in un array. Usiamo di nuovo la stessa logica più avanti per recuperare un insieme delle percentuali.
Con i dati raccolti, RGraph non avrà problemi a tracciare il nostro grafico.

Quiz & esercizi

1) Agire sul codice mostrato all'inizio per migliorare il logo di Mario, il quale non è affatto soddisfatto del risltato.
2) Correggi tutti gli errori nel codice del CSS del capitolo.
3) Completa questo codice per ottenere un quadrato giallo:

```
<canvas id="tela" width="200" height="200">
var tela = document.getElementById('tela');
var contesto = tela.getContext('2d');
contesto.fillStyle = "";
contesto.fillRect ;
```

4) Applicare un foglio di stile alla pagina con il nostro quadrato giallo per smussare gli angoli.
5) Aggiungere l'ombra a quadrato.
6) Ruotare il quadrato di 45 gradi.
7) Scrivere del testo sotto il quadrato utilizzando un font qualunque di Fontsquirrel.com
8) Scrivere un listato per un grafico dei nostri 4 sport preferiti con le relative percentuali delle ore dedicate.
9) Usare quattro font differenti per ogni grafico.

Riassunto

Abbiamo dimostrato che con HTML5 potremo anche disegnare direttamente sullo schermo, avendo a disposizione delle istruzioni potenti e divertenti allo stesso tempo. Oltre a questo c'è stato un approfondimento sull'aspetto grafico in genere, in modo da poter visualizzare particolari grafici di una pagina Web più accattivanti, e abbiamo imparato che anche il font visualizzato ha una sua importanza da non sottovalutare.

 # 5 - Audio e video

Le basi della programmazione
Le variabili, le costanti e i metodi
Le iterazioni

Oggi, nel Web moderno, praticamente il 90% di chi ci si approccia guarderà un video. Questo dato è stato raggiunto in un tempo relativamente breve, circa 20 anni e chissà cosa ci offrirà il Web nel 2043! Ma torniamo ai nostri giorni, dove audio e video sono una parte molto importante di Internet. Podcast, anteprime audio e video dimostrativi sono ovunque e fino a pochi anni fa erano utilizzabili solo servendosi di plug-in nel browser. Poi HTML5 ha introdotto nuovi metodi per incorporare file audio e video in una pagina i quali sono divenuti degli standard. In questo capitolo esploreremo alcuni metodi che possiamo utilizzare per incorporare il contenuto audio e video.
Tutto è partito dai primi che hanno incorporato file MIDI nelle home page servendosi del tag *embed* per fare riferimento al file, in questo modo:

```
   <embed src="metallica.mid" autostart="true"
  loop="true" controller="true" ></embed>
```

Il tag *embed* non è mai diventato uno standard, quindi le persone hanno iniziato a utilizzare invece il tag *object*, che è uno standard W3C accettato. Per supportare i browser meno recenti che non comprendono il tag object, si vedeva un tag embed annidato all'interno del tag *object*, come questo:

```
   <object>
<param name="src" value="litfiba.mid">
<param name="autoplay" value="false">
<param name="controller" value="true">
<embed src="queen.mid" autostart="false"
  loop="false" controller="true" ></embed>
</object>
```

I creatori della specifica HTML5 ritenevano che il browser dovesse supportare audio e video in modo nativo piuttosto che fare affidamento su un plug-in che richiede molto codice HTML standard. E fortunatamente hanno avuto ragione, infatti

sono lentamente scomparsi i vari Flash player, Silverlight, Real Player e molti altri. Questo ci ha portato a gestire i codec e i cosiddetti *containers* (contenitori) personalmente.
In Internet quando parliamo di *containers* e di codec in realtà ci stiamo riferendo a dei video. Potremmo pensare a un video come a quelli del nostro smartphone, dei file MKV o MP4, ma c'è dell'altro dietro. Infatti i contenitori, necessari per l'aggiunta di video o audio al nostro sito Web, sono come un involucro che contiene flussi audio, video e talvolta metadati aggiuntivi come i sottotitoli (es. MKV). Questi flussi audio e video devono essere codificati, ed è qui che entrano in gioco i codec. Video e audio possono essere codificati in centinaia di modi diversi, ma quando si tratta di video HTML5, solo pochi contano.
Quando guardiamo un video, il nostro video player deve decodificarlo. Sfortunatamente, il lettore che stiamo utilizzando potrebbe non essere in grado di decodificare il video che desideriamo guardare. Alcuni lettori utilizzano un software per decodificare il video, che può essere più lento o richiedere più risorse per la CPU. Altri giocatori utilizzano decoder hardware e sono quindi limitati a ciò che possono riprodurre. In questo momento, ci sono cinque formati video che dobbiamo conoscere se vogliamo iniziare a utilizzare il tag video di HTML5: H.264, H.265, Theora, VP9 e AV1, infine come se gli standard concorrenti per i video non complicassero abbastanza le cose, dobbiamo anche preoccuparci degli standard concorrenti per l'audio.

Codec video

- **H.264** è un codec di alta qualità standardizzato nel 2003 e creato dal gruppo MPEG. Per supportare dispositivi di fascia bassa come i telefoni cellulari, e allo stesso tempo gestire video per dispositivi ad alta definizione, la specifica H.264 è suddivisa in vari profili. Questi profili condividono una serie di caratteristiche comuni, ma i profili di fascia alta offrono opzioni aggiuntive che migliorano la qualità. H.264 è uno standard grazie al supporto di Microsoft e Apple, che sono licenziatari, tuttavia, non è una tecnologia aperta. È brevettato e il suo utilizzo è soggetto a condizioni di licenza. I produttori di contenuti devono pagare una royalty per codificare i video utilizzando H.264, ma queste royalty non si applicano ai

contenuti resi disponibili gratuitamente agli utenti finali.
- **H.265** è Il successore di H.264 ed è stato concepito come il codec che ci accompagnerà al web 3.0 con i video in alta risoluzione 4k e anche 8k.
I fautori del software libero hanno sempre temuto che alla fine i titolari dei diritti possano iniziare a richiedere royalties elevate dai produttori di contenuti. Questa preoccupazione ha portato, per nostra fortuna, alla creazione e alla promozione di codec alternativi, ma comunque senza compromessi sulla qualità.
- **Theora** è un codec royalty-free sviluppato dalla Xiph.Org Foundation. Sebbene i produttori di contenuti possano creare video di qualità simile con Theora, i produttori di dispositivi sono stati lenti ad adottarlo. Firefox, Chrome e Opera possono riprodurre video codificati con Theora su qualsiasi piattaforma senza software aggiuntivo, da qualche anno anche Edge, Safari e i dispositivi iOS. Apple e Microsoft sono diffidenti nei confronti dei "brevetti sottomarini", un termine usato per descrivere i brevetti in cui la domanda di brevetto ritarda di proposito la pubblicazione e il rilascio del brevetto per restare bassi mentre altri implementano la tecnologia. Quando è il momento giusto, il richiedente il brevetto "emerge" e inizia a chiedere royalties su un mercato ignaro, ma questo non è mai accaduto e Theora resta un ottimo codec, con qualità simile a H.264.
- **VP9** è il successore di VP8 di Google ed è un codec completamente aperto e privo di royalty con una qualità simile a H.265 il quale sta per essere completamente sostituito dal codec AV1 visti i numerosi vantaggi in termini di qualità ed efficienza. Se volete approfondire basterà una ricerca sui motori di ricerca.

Codec audio

- **AAC** questo è il formato audio che Apple utilizza nel suo iTunes Store. È progettato per avere una qualità audio migliore rispetto agli MP3 per circa la stessa dimensione del file e offre anche più profili audio simili a H.264. Inoltre, come H.264, non è un codec gratuito e ha costi di licenza associati.
Tutti i prodotti Apple riproducono file AAC.
- **MP3** Lo conosciamo tutti, ma è ancora soggetto al pagamento di royalties, tuttavia in giro per la rete si trovano

diversi software per la codifica gratuita, magari con qualche limite con la qualità.
- **Vorbis (OGG)** Questo formato open source esente da royalty ed è ampliamente supportato. Lo troverai utilizzato anche con i codec video Theora e VP9. I file Vorbis hanno una qualità audio molto buona ma non sono ampiamente supportati dai lettori musicali hardware.

Contenitori e codec

Un contenitore è un file di *metadati* (quindi dei dati con delle informazioni specifiche di contenuto) che identifica e accomuna file audio o video. Un contenitore in realtà non contiene alcuna informazione su come vengono codificati i suoi contenuti. In sostanza, un contenitore "avvolge" i flussi audio e video. I contenitori possono spesso includere qualsiasi abbinamento di media codificati. Vedremo spesso queste combinazioni quando si tratterà di lavorare con i video sul Web:

- Il contenitore *OGG*, con video Theora e audio Vorbis.
- Il contenitore *MKV*, che utilizza anche AV1 e H.265.
- Il contenitore *MP4*, con video H.264, H.265 e audio AAC.
- Il contenitore *WebM*, che utilizza video VP9 e audio Vorbis.

Mario, grande appassionato di musica, sta sviluppando un sito per mostrare alcuni loop audio esenti da royalty da utilizzare negli screencast e vorrebbe vedere una pagina demo simulata di una singola raccolta di loop. Quando avremo finito, avremo un elenco dei loop audio e un visitatore sarà in grado di ascoltarli rapidamente. Non dobbiamo preoccuparci di trovare audio per questo progetto, perché l'ingegnere del suono del cliente ci ha già fornito i campioni di cui avremo bisogno in entrambi i formati MP3 e OGG e sono quattro campioni: batteria, organo, basso e chitarra. Dobbiamo descrivere ciascuno di questi esempi usando il markup HTML. Ecco il markup per il loop per il Kazoo:

```
 <article class ="campione">
<header><h2>Kazoo</h2></header>
<audio id="controlli kazoo">
<source src="sounds/ogg/kazoo.ogg" type="audio/ogg">
```

```
  <source src="sounds/mp3/kazoo.mp3" type="audio/mpeg">
  <a href="sounds/mp3/kazoo.mp3">Scarica kazoo.mp3</a>
  </audio>
   </article>
```

Definiamo prima l'elemento audio e gli diciamo che vogliamo che alcuni controlli vengano visualizzati. Successivamente, definiamo più origini per il file. Definiamo prima le versioni MP3 e OGG del campione, quindi visualizziamo un collegamento per consentire al visitatore di scaricare direttamente il file MP3 se il browser non supporta l'elemento audio.

Mettiamolo all'interno di un modello HTML5 con gli altri tre campioni audio:

```
   <article class ="campione">
  <header><h2>Kazoo</h2></header>
  <audio id="controlli kazoo">
  <source src="sounds/ogg/kazoo.ogg" type="audio/ogg">
  <source src="sounds/mp3/kazoo.mp3" type="audio/mpeg">
  <a href="sounds/mp3/kazoo.mp3">Scarica kazoo.mp3</a>
  </audio>
   </article>
   <article class="campione">
  <header><h2>Flauto traverso</h2></header>
 <audio id="controlli del flauto traverso">
   <source src="sounds/ogg/flauto.ogg" type="audio/ogg">
              <source     src="sounds/mp3/flauto.mp3"
type="audio/mpeg">
            <a    href="sounds/mp3/flauto.mp3">Scarica
flauto.mp3</a>
 </audio>
</article>
<article class="campione">
 <header><h2>Zampogna</h2></header>
 <audio id="comandi della zampogna">
              <source     src="sounds/ogg/zampogna.ogg"
type="audio/ogg">
              <source     src="sounds/mp3/zampogna.mp3"
type="audio/mpeg">
           <a    href="sounds/mp3/zampogna.mp3">Scarica
zampogna.mp3</a>
 </audio>
```

```
</article>
<article class="campione">
 <header><h2>Oboe</h2></header>
 <audio id="controlli oboe">
   <source src="sounds/ogg/oboe.ogg" type="audio/ogg">
   <source src="sounds/mp3/oboe.mp3" type="audio/mpeg">
   <a href="sounds/mp3/oboe.mp3">Scarica oboe.mp3</a>
  </audio>
 </article>
</body>
</html>
```

Quando apriamo la pagina nel browser ogni voce nell'elenco avrà il proprio lettore audio. Il browser stesso gestisce la riproduzione dell'audio quando si premerà il pulsante Riproduci.

Più potenza con le API

In questo capitolo, abbiamo solo osservato la punta dell'iceberg riguardo le API JavaScript per gli elementi audio e video. La versione completa è in grado di rilevare i tipi di file audio che il browser può riprodurre e fornisce metodi per controllare la riproduzione degli elementi audio.

Nella parte in cui abbiamo giocato con l'audio, qui, costruendo un'applicazione Web con più campioni sonori, potremmo usare l'API JavaScript per riprodurre tutti i suoni (all'incirca) contemporaneamente. Ecco un approccio davvero semplificato:

```
   var prova = $("<p><input type='button' value='Riproduci tutto' /></p>") prova.click(function(){
$("audio").each(function(){
   this.play();
  })
});

   $("body").append(prova);
```

Stiamo creando un pulsante "Riproduci tutto" che, se premuto, scorre tutti gli elementi audio sulla pagina e chiama il metodo *play()* su ciascun elemento.

Possiamo fare cose simili con i video. Esistono metodi per avviare e mettere in pausa gli elementi o anche interrogare l'ora corrente.

L'API *Web Audio* invece è un altro esempio delle potenzialità che abbiamo a disposizione. Infatti ci fornirà un sistema molto solido che ci permetterà di controllare l'audio sul Web, quindi

ad esempio di applicare alcuni effetti spaziali (riverbero, panning), aggiungerne altri, scegliere le sorgenti audio, e altro ancora. Sulla rete vi sono diversi tutorial per usufrire della sue potenzialità.
Ve sono tantissime altre possibilità, è una buona idea controllare sui siti ufficiali, come ad esempio quello di Mozilla.org:
https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API
E' tutto in inglese, ma potremo utilizzare il traduttore.

Quiz & esercizi

1) Cosa significa codificare un file audio o video?
2) Perché non posso utilizzare liberamente file .mp3 nei miei podcast?
3) Quale codec dovrò utilizzare per inserire un filmato in 8K nella mia pagina Web?
4) Il vantaggio di usufruire di un contenitore piuttosto che un altro?
5) Creare una pagina Web con alcuni video dimostrativi in 8K

Riassunto

La gestione dell'audio e dei video nel Web oramai è data per scontata, anche se presuppone una minima conoscenza di alcuni fondamentali che abbiamo osservato velocemente in questo paragrafo.

 # 6 - Lavorare con dati lato client

Utilizzare Web Storage

Scambiare dati con il client

la cache e il manifest

Fino a questo punto ci siamo occupati in buona parte di argomenti relativi all'estetica o comunque alla grafica, tralasciando un po' quella parte che si occupa di aspetti più pratici. Abbiamo parlato di impaginazione con HTML5 e CSS3, ma ora rivolgiamo la nostra attenzione ad alcune delle tecnologie e delle funzionalità associate a HTML5. La messaggistica incrociata e il supporto offline, ad esempio, ci consentono di comunicare tra domini e creare soluzioni che consentono ai nostri utenti di lavorare offline.
HTML5 ha introdotto alcune nuove opzioni per l'archiviazione dei dati sul client: *Web Storage* (utilizzando *localStorage* o *sessionStorage*) e *IndexedDB*. Sono facili da usare, incredibilmente potenti, ragionevolmente sicuri e sono implementati praticamente da tutti i browser.

Sono anche utili per la creazione di applicazioni mobili che possono essere eseguite nel browser ma non sono connesse a Internet.

Attualmente molte applicazioni Web richiamano un server per salvare i dati dell'utente, ma con questi nuovi meccanismi di archiviazione, una connessione Internet non è più una dipendenza assoluta. I dati dell'utente possono essere archiviati localmente e sottoposti a backup quando necessario.

Combinando questi metodi con le nuove funzionalità offline di HTML5, potremo creare applicazioni database complete direttamente nel browser che funzionano su un'ampia varietà di piattaforme, come i desktop, gli smartphone e i tablet.
Ma facciamo un passo indietro. Quando il World Wide Web è stato concepito, era inteso come un deposito per documenti, principalmente documenti accademici. Il Web era costituito da collegamenti ipertestuali, che consentivano a un documento di fornire un collegamento a un altro documento e ha quindi

consentito una comoda navigazione da un'informazione all'altra. Non passò molto tempo prima che le pagine Web iniziassero a fornire funzionalità più avanzate, come gli acquisti online. Ciò richiedeva che le pagine Web rispondessero dinamicamente all'interazione dell'utente, ciò è stato facilitato da tecnologie come l'API DOM, che consente a una pagina Web di essere modificato dopo il caricamento.
Non era ancora possibile creare pagine o applicazioni Web che esibivano le funzionalità tipiche delle applicazioni desktop native. Ad esempio, le pagine Web non disponevano delle seguenti funzionalità:
- La capacità di interagire con il file system al di là del semplice tipo di input di file.
- La capacità di memorizzare grandi quantità di dati o informazioni di configurazione all'interno del browser (sul client).
- La capacità di funzionare senza una connessione di rete. Anche se una pagina è memorizzata nella cache all'interno di un browser, in genere non è utilizzabile senza una connessione di rete.
- La possibilità di eseguire un'elaborazione intensiva su un thread in background senza alcun impatto per l'esperienza dell'utente. Poiché tutta l'elaborazione JavaScript avviene su un singolo thread, un'elaborazione intensiva causerà il "blocco" della pagina Web.
- La possibilità di richiedere dati aggiuntivi da un server dopo che una pagina è stata caricata senza eseguire un aggiornamento dell'intera pagina Web.
Potremo pensare a queste funzionalità come ai servizi accessibili a un'applicazione software. Per un applicazione desktop tradizionale, il sistema operativo fornisce questi servizi. Le pagine Web non possono interagire direttamente con il sistema operativo; possono interagire solo con il browser. Pertanto, a meno che il browser non fornisse questi servizi, le pagine web sarebbero sempre vincolate dal raggiungimento di un livello di sofisticazione superiore.
Nonostante i limiti storici di HTML e JavaScript, c'è stata una forte spinta verso applicazioni Web basate su browser rispetto alle applicazioni native. Le applicazioni web basate su browser sono estremamente convenienti perché è possibile accedere

alla stessa applicazione web su qualsiasi dispositivo, da qualsiasi luogo, in qualsiasi momento. Questa spinta si è solo intensificata con il passaggio al cloud computing, come sempre più dati vengono inseriti nei computer cloud disponibili a livello globale.
Le applicazioni Web basate su browser hanno anche il vantaggio di non dover essere installate e possono essere aggiornati automaticamente senza alcuna azione da parte dell'utente.
Per consentire alle applicazioni web di raggiungere lo stesso livello di sofisticazione delle applicazioni native, la specifica HTML5 fornisce un set di API JavaScript per implementare tutte le funzionalità appena elencate, insieme a molti altri. Questa sezione tratterà in dettaglio queste API, a partire dall'archiviazione lato client.

Archiviazione lato client

Il protocollo HTTP utilizzato per recuperare risorse da un server Web è stato progettato come protocollo *stateless*.
Questo protocollo inizia con il browser che stabilisce una connessione di rete al server Web e richiede una risorsa. Il server Web trova la risorsa e la restituisce al browser come risposta. Dopo ciò la connessione tra il server e il browser viene chiusa e non viene mantenuto alcun ulteriore collegamento.
Ovviamente questa è una visione leggermente semplificata del protocollo, anche perché stabilire le connessioni può richiedere molto tempo. Inizialmente il protocollo HTTP è stato esteso per supportare un'opzione *keep-alive* (essenzialmente per mantenere il collegamento attivo). Ciò significava che potevano essere fatte più richieste sulla stessa connessione. Non era comunque possibile fare affidamento su questa funzione per continuare una conversazione con il server Web, perché non dipendeva dall'utente per quanto tempo il server manteneva la connessione.
E' chiaro quindi che questo modello non poteva funzionare, anche per diversi altri motivi. Ad esempio, considerando un sito di ecommerce, il server dovrebbe conservare un carrello per l'utente mentre questo si muove da una pagina all'altra,

ovviamente questo si baserà sul server Web che memorizza ogni particolare browser e la cronologia delle loro azioni, come il prodotto nel carrello. Essenzialmente ci stiamo riferendo a quella che, comunemente, viene definita una *sessione*.
Per supportare ciò, è stata introdotta una tecnologia chiamata cookie. Un cookie è semplicemente una coppia di dati chiave/valore che può essere inviata dal server Web al browser su una risposta HTTP. Il browser farà quindi memorizzare le informazioni in questo cookie per un periodo di tempo configurabile (di solito come piccoli file su il file system). Ogni volta che il browser invierà una richiesta allo stesso server Web in futuro, farà includere tutti i cookie che gli sono stati inviati.
Sebbene i cookie possano essere utilizzati per memorizzare qualsiasi informazione testuale, in genere vengono utilizzati per memorizzare un ID di sessione univoco per ciascun browser. Il server Web genera questo ID di sessione ogni volta che riceve una richiesta da un browser, che quindi salva e ripropone automaticamente alle richieste future. Il server Web può quindi memorizzare le informazioni rispetto a questo ID di sessione e fornire tali informazioni al browser se necessario, ad esempio quando l'utente decide di effettuare il check-out.
Sebbene i cookie siano ottimi per memorizzare piccole quantità di dati, hanno diverse limitazioni. Il limite di principio è che i browser devono consentire solo 20 cookie per nome di dominio e ogni cookie è limitato a 4096 byte di dati. Se si lavora attraverso le somme, un dominio potrebbe essere in grado di memorizzare solo 80 kilobyte all'interno del browser, che, per gli standard moderni, non è granchè.
Ci sono, tuttavia, ottime ragioni per cui un'applicazione Web potrebbe voler archiviare quantità maggiori di dati sul client, pensiamo alle prestazioni, se i dati fossero archiviati sul client, sarebbero molto più veloci da elaborare e visualizzare nel browser rispetto ad un caricamento dal server Web. Infatti, nonostante l'aumento della velocità nella rete, l'accesso ai dati in locale è ancora molto più veloce. Infine anche la disponibilità è importante, se i dati fossero memorizzati sul client, sarebbe possibile accedervi anche quando il browser non è connesso ad internet.

Ottimo, ecco perché HTML5 dispone non di una, ma di ben tre API distinte per la memorizzazione dei dati all'interno del browser:
- *Archiviazione Web*: E' l'API di archiviazione più vecchia e ha un'eccellente supporto su tutti i browser, risultando abbastanza semplice da maneggiare.
- *Web SQL*: questo standard propone un'API basata su database relazionale, ha dei punti a favore, ma non è ampiamente diffuso e soprattutto è stato deprecato, quindi non ci perderemo tempo.
- *IndexedDB*: Un'API molto più sofisticata rispetto a quella di archiviazione Web, essenzialmente è molto più completa ma anche più complessa.
Prima di procedere diamo una breve occhiata a JSON di Javascript, il quale ci aiuterà proprio con *IndexedDB*.

JSON e XML

JSON (*JavaScript Object Notation*) è un formato per scambiare dati in maniera che siano leggibili e chiari per chi li scrive e per chi dovrà interpretarli, cioè il computer. Possiamo anche semplificare dicendo che è il modo con cui si possono riprodurre degli oggetti javascript con una sintassi ridotta, in modo che successivamente sia riproducibile in una stringa. JSON è indipendente da Javascript e si basa su di una struttura convenzionale, riconoscibile da chi ha familiarità con qualsiasi altro linguaggio di programmazione, proprio perché JSON ha come scopo l'interscambio di dati. Dal 2009 è parte integrante di Javascript, dove essenzialmente convertirà in una stringa il nostro oggetto che, ad esempio, intendiamo inviare ad un altro computer in rete, o memorizzarlo in un file. Questo processo di conversione è definito *serializzazione*, mentre il processo inverso è chiamato *deserializzazione*.
Per convertire un oggetto in una stringa, è necessario un formato dati che specifichi come l'oggetto dovrebbe essere mappato su una stringa di caratteri, ad esempio come si fa a denotare la proprietà e i valori di un oggetto e come si codificano i vari tipi di dati come numeri e matrici? Storicamente la maggior parte dei formati di dati sono binari, ciò significa che non è possibile per un essere umano leggere i dati formattati e ottenere una comprensione della sua struttura o del significato, dato che in genere, come molti sapranno,

questi dati si presentano come lunghe serie di 0 e 1.
Negli ultimi anni, c'è stato un passaggio verso i formati di dati in testo normale. L'esempio principale è XML, che utilizza una struttura e una sintassi simili all'HTML per differenziare le proprietà dai loro valori.
Non c'è nulla di male nell'usare XML per serializzare oggetti JavaScript e oramai molte applicazioni web la utilizzano. Ma perché mai dovremmo convertire i nostri dati in XML? proprio perché questo linguaggio è utilizzato per trasportare dati, anche tra programmi applicativi o tra sistemi operativi differenti. Quindi, riassumendo, se avessimo la necessità di esportare dei dati, ad esempio, dal listato Javascript che gestisce un sito di ecommerce, ad un applicativo installato sul nostro computer per analizzare le statistiche di vendita, succederà questo: tramite JSON i dati verranno convertiti in stringhe le quali saranno così comprensibili ed utilizzabili da una libreria di conversione in XML. A quel punto l'applicativo importerà il file XML e sarà in grado di visualizzare le statistiche. Senza questo procedimento e questi ponti di collegamento tutto sarebbe molto più criptico e chiuso.
Vediamo un esempio di come è strutturato un file XML:

```
    <?xml version="1.0" codifica="UTF-8"?>
    <prodotti>
    <nomeProdotto>Miele</nomeProdotto>
    <prezzo>10 euro</prezzo>
    <peso>500 grammi</peso>
    <gusto>Arancia</gusto>
    <quantità>25</quantità>
<inserito>01-01-2022</inserito>
<azienda produttrice>
<nome>L'Alveare</nome>
```

Una struttura di questo tipo è molto simile al codice Javascript, anche se diversa. Per la conversione avremo bisogno di una libreria (in rete ve ne sono diverse, ad esempio: https://goessner.net/download/prj/jsonxml/) dato che, come accennato, Javascript non supporta XML.
Il file visto precedentemente in Javascript apparirebbe così:

```
    prova = {
nome Prodotto: "Miele",
```

```
prezzo:“10 euro”,
peso:“500 grammi”,
gusto:“Arancia”,
quantità:25,
inserito: data Oggi
  azienda produttrice:{
  nome: “L’Alveare”
  },
}
```

Per procedere con la conversione abbiamo bisogno della funzione *JSON.stringify()* :

```
   prodotti = JSON.stringify(prova)
“{“nomeProdotto”:“Miele”,      “prezzo”:“10      euro”,
“peso”:“500  grammi”,  “gusto”:“Arancia”,  “quantità”:25,
“inserito”:“01-01-2022”,     “azienda     produttrice”:
{“nome”:“L’Alveare”},”}”
```

Si noti che le proprietà e i valori vengono tutti mantenuti, e i nomi delle proprietà sono automaticamente racchiusi tra virgolette. Quindi, quando useremo *stringify*:
- Le stringhe appariranno sempre tra virgolette doppie.
- I numeri non hanno bisogno delle virgolette; ad esempio, la proprietà *quantità* ha un valore di 25.
- I valori booleani sono rappresentati senza virgolette con le parole chiave *true* o *false,* lo stesso vale anche per *undefined* o *null*.
- Gli oggetti possono incapsulare oggetti figlio.
- Gli Array possono essere utilizzati come valore per qualsiasi proprietà con le parentesi quadre, mentre gli elementi sono separati da virgole.

JSON consente di rappresentare tutti i tipi di dati JavaScript nella versione serializzata, e, quando la stringa viene riconvertita in un oggetto, le varie proprietà mantengono tutti i dati originali.
Per riconvertire la stringa in un oggetto useremo la funzione *JSON.parse()*:

```
   recupero = JSON.parse(prodotti);
typeof recupero
recupero.nomeProdotto
recupero.quantità
```

Sebbene il formato JSON sia ottimo per la serializzazione delle proprietà, non può essere utilizzato con i metodi,

verranno semplicemente ignorati quando si utilizza *JSON.stringify()*.

L'API di archiviazione Web

L'API di archiviazione Web è di gran lunga la più semplice delle tre API di archiviazione dati specificate in HTML5 e, come accennato, ha anche il miglior supporto tra tutti i principali fornitori di browser. Tuttavia, l'API di archiviazione Web presenta alcune limitazioni, infatti può essere utilizzato solo per la memorizzazione di dati testuali (stringhe Javascript). Non è possibile memorizzare altri tipi di dati come *oggetti* JavaScript (essenzialmente una porzione di codice, una funzione, che esegue un compito). Inoltre i browser possono limitare un dominio a 5 MB di spazio di archiviazione. Anche se si tratta di un'enorme quantità di dati rispetto ai cookie, potrebbe non essere fattibile per tutti gli scenari.

Il limite di 5 MB è un vincolo ancora più grande di quanto possa sembrare. Questo è a causa del fatto che ogni carattere in una stringa JavaScript utilizza 2 byte di memoria.

JavaScript utilizza una codifica dei caratteri chiamata UTF-16, che consente qualsiasi Unicode carattere da rappresentare in 2 o più byte. Altre codifiche come UTF-8 ora sono molto più comuni e usano solo 1 byte per l'Unicode più comune caratteri (a condizione che tu stia utilizzando un alfabeto occidentale).

L'API di archiviazione web è straordinariamente semplice da usare; si basa su semplici coppie chiave/valore. Per vederla in azione, apriremo una pagina Web, ad esempio prova.html utilizzando l'indirizzo relativo al nostro server Web (es. localhost:1234/prova.html) e inseriremo il codice seguente nella console JavaScript:

```
localStorage.setItem("prova", "Saluti da Mario");
```

In questo caso, *prova* è la chiave e *Saluti da Mario* è il valore. Come possiamo vedere, l'oggetto *localStorage* fornito dal browser espone l'API di archiviazione Web.

NOTA Il browser espone anche l'API di archiviazione Web tramite un compagno oggetto chiamato *sessionStorage*. Tutti i dati archiviati tramite localStorage sono conservati a tempo indeterminato (o fino alla cancellazione da parte dell'utente), mentre i dati conservati tramite sessionStorage verranno cancellati automaticamente alla chiusura del browser.

Dovremmo essere sempre consapevoli del fatto che i dati archiviati tramite l'API Web storage non sono crittografati; pertanto, non è appropriato per i dati sensibili.
Se apriamo la scheda Risorse degli strumenti per sviluppatori ed espandi l'opzione Archiviazione locale, tu vedrà che la coppia chiave/valore è stata acquisita. Si noti che questi dati sono associati all'origine *localhost:1234*. Solo pagine servite da questo origine avrà accesso a questi dati.
Il metodo complementare per *setItem* è *getItem*. Ciò consente di recuperare un elemento in base a una chiave. Per esempio:

```
    localStorage.getItem("prova");
"Saluti da Mario"
```

Questo metodo restituirà sempre una stringa JavaScript o non definita se non è presente alcun valore archiviato la chiave specificata.
Altri due metodi utili sono inclusi nell'API. Il metodo *removeItem* può essere utilizzato per rimuovere un valore in base a una chiave. Per esempio:

```
localStorage.removeItem("prova");
```

Infine, il metodo *Clear* può essere utilizzato per rimuovere tutti i dati memorizzati dall'origine:

```
localStorage.clear();
```

Questi semplici metodi sono tutto ciò che è necessario per utilizzare l'archiviazione web.
Quindi essenzialmente ll meccanismo *localStorage* fornisce agli sviluppatori un metodo molto semplice per rendere persistenti i dati sulla macchina del client. Come già visto *localStorage* è semplicemente un archivio con i campi nome/valore integrati nel browser web.
Le informazioni archiviate in *localStorage* persistono tra le sessioni del browser e non possono essere lette da altri siti Web, perché sono limitate al dominio che stai attualmente visitando.
Mario crede che un sito più accattivante e personalizzabile possa attirare altri utenti e quindi vorrebbe che siano in grado di modificare la dimensione del testo, lo sfondo e il colore del testo del sito. Verrà implementato utilizzando *localStorage* in modo che quando salveremo le modifiche, queste rimarranno

da una sessione del browser all’altra.

Modulo per le impostazioni, salvataggio e recupero

Creiamo un modulo usando del markup semantico HTML5 e alcuni dei nuovi controlli dei moduli che abbiamo imparato nel capitolo 2. Vogliamo consentire all'utente di cambiare il colore di primo piano, cambiare il colore di sfondo e regolare la dimensione del carattere. Useremo solo codici colore HTML per il colore:

```
   <p><strong>Preferenze</strong></p>
<form id="preferenze" action="save_prefs"
    method="post" accept-charset="utf-8">
<fieldset id="colors" class="">
  <legend>Colori</legend>
 <ol>
  <li>
  <label for="colore_sf">Colore di sfondo</label>
  <input type="color" name="colore_sf"
       value="" id="colore_sf" >
 </li>
<li>
<label for="colore_tx">Colore del testo</label>
  <input type="color" name="colore_tx"
     value="" id="colore_tx" >
 </li>
<li>
<label for="dim_testo">Dimensione del testo</label>
<select name="dim_testo" id="dim_testo">
 <option value="16">16px</option>
 <option value ="18">18px</option>
 <option value ="22">22px</option>
 <option value ="30">30px</option>
 </select>
</ol>
</fieldset>
<input type="submit" value="Salva modifiche">
</form>
```

Per lavorare con *localstorage* utilizzeremo JavaScript per usifruire di *window.localStorage()*. Il suo uso è piuttosto semplice come vedremo:

```
   localStorage.setItem("colore_sf", $("#colore_sf"
).val());
```

Recuperare un valore è altrettanto facile:

```
   var coloreSf = localStorage.getItem("colore_sf");
```

Creiamo un metodo per salvare tutte le impostazioni dal modulo:

```
   function salva_pref(){
  localStorage.setItem("colore_sf" ,
$("#colore_sf").val());
 localStorage.setItem("colore_tx" , $("#colore_tx"
).val());
  localStorage.setItem("dim_testo" , $("#dim_testo"
).val());
  applica();
}
```

Quindi, costruiamo un metodo simile che caricherà i dati dal sistema di archiviazione locale e li inserirà nei campi del modulo:

```
   funzione carica_pref(){
  var coloreSf = localStorage.getItem("colore_sf");
  var colore_tx = localStorage.getItem("colore_tx" );
  var dim_testo = localStorage.getItem("dim_testo");

  $("#colore_sf".val(coloreSf);
 $("#colore_tx").val(colore_tx);
 $("#dim_testo").val(dim_testo);

  applica();
}
```

Questo metodo ne chiama anche uno ulteriore che applicherà le impostazioni alla pagina stessa, che scriveremo in seguito.
Ora che possiamo recuperare le impostazioni da *localStorage*, dobbiamo applicarle alla pagina. Le preferenze con cui stiamo lavorando sono tutte legate ai CSS in qualche modo e possiamo usare jQuery per modificare gli stili di qualsiasi elemento.

```
   function applica(){
  $("body").css("coloreSf", $("#coloreSf" ).val());
  $("body").css("colore_tx" , $("#colore_tx").val());
    $("body").css("dim_testo"  ,  $("#dim_testo".val()  +
"px");
}
```

Infine, dovremmo eseguire tutto questo quando il documento sarà pronto:

```
   $(function(){
  carica_pref();
  $('form#preferenze').submit(function(evento){
     evento.preventDefault();
     salva_pref();
```

```
  });
});
```

I metodi *localStorage* e *sessionStorage* ci offrono un modo semplice per memorizzare semplici coppie nome/valore sul computer del client, ma a volte abbiamo bisogno di più. Infatti Mario vorrebbe implementare il sito di ecommerce con una parte dedicata alla gestione dei clienti, in pratica un CRM (Custom Relationship Managment).

Applicare il tag corretto

Prima di ogni cosa dobbiamo ripassare il processo di assegnazione dei tag. Infatti per usufrire dei vari meccanismi di memorizzazione del capitolo dovremo racchiudere i dati in un'area adeguata definita con il tag *template*. Per iniziare a utilizzare il tag *template* (modello), è sufficiente aggiungerlo ovunque nell'HTML e includere la relativa struttura HTML al suo interno. In genere forniamo anche un ID per il template in modo da poterlo individuare. Pertanto, vediamo un esempio dove iniziamo aggiungendo quanto segue immediatamente prima del tag principale di chiusura:

```
   <template id="rigaContatti">
<td></td>
<td></td>
<td></td>
<td></td>
<td><ora></ora>
<div class="overlay"></div></td>
</template>
```

Ricaricando la pagina Web, noteremo che questo codice HTML non viene visualizzato all'interno del browser. Inoltre, sebbene sia possibile selezionare l'elemento *template* utilizzando i selettori DOM e jQuery, non sarà possibile selezionarne i figli. Per esempio:

```
   $('#rigaContatti td')
[]
```

Quando si utilizza *template*, è necessario compilarlo con i dati appropriati. Chiaramente, potremmo semplicemente impostare il nome del contatto nel primo figlio, il numero di telefono nel secondo figlio e così via.

Ove possibile, è meglio trovare soluzioni generiche a problemi comuni. Pertanto, scriveremo un algoritmo che accetterà

qualsiasi modello e oggetto e popolerà il template con i dati nell'oggetto in base a un insieme di convenzioni.
Per raggiungere questo obiettivo, utilizzeremo gli *attributi dei dati*. Per creare l'algoritmo generico menzionato nella sezione precedente, è necessario un modo per contrassegnare i tag nel modello con i nomi delle proprietà dell'oggetto che dovrebbe essere utilizzato per popolarli.
Ci sono diversi modi per farlo. Ad esempio, se un elemento td deve essere popolato con i dati nella proprietà contactName, è possibile specificare il *td* come segue:

```
<td id="nomeContatto"></td>
```

L'ovvio problema con questo approccio è che gli ID all'interno del documento dovranno essere univoci, e quindi dovremmo assicurarci che i nomi delle proprietà nei nostri oggetti non siano mai in conflitto con gli ID degli elementi nel documento.
Un approccio alternativo consiste nello specificare il nome della proprietà come classe:

```
<td class="nomeContatto"></td>
```

Non c'è nulla di intrinsecamente sbagliato nell'usare i nomi delle classi per scopi non inerenti al CSS. Tuttavia, un problema ovvio con questa soluzione è che il nostro CSS potrebbe contenere una classe che corrisponde a una delle proprietà.
Un altro approccio che è stato utilizzato storicamente consiste nel servirsi di attributi supportati dalla specifica ma non utilizzati dal browser, come ad esempio *rel*:

```
<td rel="nomeContatto"></td>
```

Il problema con questo approccio è che, sebbene i browser non lo utilizzino, l'attributo *rel* ha un significato e viene utilizzato dai motori di ricerca quando viene utilizzato su un tag.
Storicamente, non c'è stato un buon modo per collegare i dati specifici del programma a un elemento. Fortunatamente HTML5 offre un modo molto migliore per risolvere questo problema. È possibile specificare i propri attributi su qualsiasi elemento, a condizione che siano preceduti da data-. Possiamo vederlo nel seguente esempio:

```
    <template id="rigaContatti">
    <td data-property-name="nomiContatti"></td>
    <td data-property-name="telefono"></td>
<td data-property-name="indirizzoEmail"></td>
<td data-property-name="indirizzo"></td>
<td><time data-property-name="dataContatto"></time>
<div  data-property-name="notes"  class="overlay"></div>
</td>
</template>
```

Gli attributi dei dati dovrebbero seguire le convenzioni di denominazione mostrate qui, in particolare:
- Devono iniziare con *data-*.
- Dovrebbero contenere solo caratteri minuscoli.
- Dovrebbero usare i trattini per separare le parole.

Proprio come qualsiasi attributo, a quelli dei dati possono essere assegnati valori. In questo caso, il valore è stato definito come il nome della proprietà dell'oggetto che dovrebbe essere utilizzato per popolare il testo dell'elemento.

Archiviare altri tipi di dati

Come abbiamo già osservato in precedenza, l'API di archiviazione Web può essere utilizzata solo per la memorizzazione di dati testuali; non per memorizzare dati strutturati come oggetti JavaScript. Questo rappresenta un problema per la nostra applicazione CRM perché idealmente vorremmo archiviare gli oggetti di contatto JavaScript nell'archivio web in modo che vengano mantenuti quando la pagina viene aggiornata o il browser viene chiuso.
L'API di archiviazione Web non si lamenterà se dovessimo specificare come valore per una chiave un oggetto JavaScript, verrà semplicemente convertito in una stringa attraverso il metodo *toString*. Questo di solito significa che il valore della chiave sarà *[object Object]* dato che questo è il valore predefinito di *toString*.

Fortunatamente, c'è una soluzione semplice per ovviare a questo probema di possibile confusione: potremo usare la funzione *JSON.stringify* per convertire gli oggetti JavaScript in stringhe con codifica JSON e quindi archiviare queste stringhe nell'archiviazione Web. Quando avremo bisogno di recuperare i dati dall'archivio web, li riconvertiremo in oggetti JavaScript.

Ad esempio, se desideriamo salvare un “oggetto contatto”, potremo creare la seguente funzione in prova.js:

```
    function memo(contatto) {
var x = JSON.stringify(contatto);
localStorage.setItem('contatti', x);
    }
```

Questa funzione aggiunge un singolo contatto all’archivio web. Poiché ovviamente è necessario memorizzare più contatti, potremmo decidere di creare un array (insieme) per contenere gli oggetti. Inoltre, sarà necessario recuperare l’array esistente dall’archivio Web prima di aggiungere un nuovo contatto. La seguente funzione fornisce quindi la funzionalità necessaria per memorizzare più contatti:

```
    function memo(contatto) {
var contattiMem = localStorage.getItem('contatti');
var contatti = [];
  if (contattiMem) {
  contatti = JSON.parse(contattiMem);
  }
contatti.push(contatto);
localStorage.setItem('contatti',
JSON.stringify(contatti));
    }
```

La prima riga di questa funzione estrae l’array esistente dall’archivio Web. Se esiste, converte la stringa in un array JavaScript. Se nessun contatto è stato salvato, crea semplicemente uno spazio vuoto nell’array per mantenere i contatti. Quindi inseriamo il nuovo contatto nell’array e lo salviamo sul Web.

È anche possibile estrarre oggetti da *localStorage* utilizzando la tradizionale notazione punto, ad esempio *localStorage.contatti*. Per usare questo approccio, le chiavi degli elementi devono essere conformi agli standard dei nomi delle proprietà di JavaScript.

Il metodo *setItem* sovrascriverà qualsiasi voce esistente per la stessa chiave, quindi, ogni volta un nuovo il contatto viene salvato, l'array memorizzato nella memoria Web verrà completamente sostituito.

La funzione *memo* va aggiunta alla funzione *ContactsScreen*, subito dopo la riga seguente:

```
var initialized = false;
```

Il metodo di salvataggio dovrebbe quindi essere modificato per invocare questa funzione:

```
save: function(evento) {
  if                    ($(evento.target).parents('form')
[0].checkValidity()) {
  var parte =
                          $(screen).find('#rigaContatti')
[0].content.cloneNode(true);
  var riga = $('<td>').append(parte);
  var contact = this.serializeForm();
  memo(contatto);
  riga = bind(riga, contatto);
  $(riga).find('time').setTime();
  $(screen).find('table tbody').append(riga);
  $(screen).find('form :input[nome]').val('');
  $(screen).find('#dettaglioContatti').toggle("blind");
  this.updateTableCount();
}
}
```

Ora è necessario aggiungere funzionalità per caricare i contatti esistenti quando si accede alla pagina Web *Contatti.html* carichi. Ciò può essere ottenuto aggiungendo il seguente metodo a *prova.js* (subito dopo il file metodo di salvataggio):

```
caricaContatti: function() {
var contattiMem = localStorage.getItem('contatti');
 if (contattiMem) {
   contatti = JSON.parse(contattiMem);
   $.each(contatti, function(i, v) {
         var  parte  =  $(screen).find('#rigaContatti')
[0].content.
    cloneNode(true);
    var riga = $('<td>').append(parte);
    riga = bind(riga, v);
    $(riga).find('time').setTime();
    $(screen).find('table tbody').append(riga);
```

```
  });
}
}
```

Questa funzione è relativamente semplice; estrae semplicemente l'array di contatti dall'archivio Web, scorre gli elementi e li aggiunge ciascuno alla tabella sullo schermo utilizzando il codice precedente.
Questa funzione dovrebbe quindi essere invocata alla fine del metodo *init* con la seguente riga di codice:

```
this.caricaContatti();
```

Se ora salviamo un contatto e riapriamo la pagina Web *contatti.html*, dovremmo vedere che sono stati tutti salvati.
Adesso avremo bisogno anche della possibilità di eliminare un contatto, vediamo come dovremo procedere.
Apriamo il file *prova.js* nel nostro editor di testo, apporteremo una modifica al processo di salvataggio. Infatti quando salveremo un nuovo contatto, gli assegneremo un ID univoco. Questo ci permetterà identificare il contatto in maniera specifica. L'ID che creeremo sarà basato sul valore relativo al *tempo* in millisecondi. Per farlo, aggiungiamo una nuova riga di codice dopo: `var contatti = this.serializeForm();`

```
var riga = $('<td>').append(parte);
```

Dove imposteremo una proprietà ID sul contatto per avere un valore derivato dalla chiamata di funzione *$.now()*. Questo è un altro vantaggio derivato dall'utilizzo di jQuery e restituisce l'ora corrente in millisecondi utile al nostro scopo.
A questo punto dato che tutti i contatti esistenti non avranno proprietà ID, dovremo eliminarli. Utilizzeremo la chiamata al metodo *localStorage.clear()* dalla riga di comando per eliminare tutti i dati dalla memoria Web.
Quando le righe di contatto vengono aggiunte alla tabella in *loadContacts*, verrà aggiunto un attributo di dati a l'elemento *td* che specifica l'ID del contatto nella riga. Verrà utilizzato il listener (ascoltatore) di eventi di eliminazione per determinare l'ID del contatto da eliminare.
Ora, lo cambieremo in modo che l'elemento td abbia un attributo chiamato *data-id* con il valore dal nostro ID, come abbiamo visto nel paragrafo precedente.
Il tutto è stato diviso in tre righe di codice separate. Il file

prova.js ha già un metodo di eliminazione che rimuove un contatto dalla tabella. È necessario aggiungere il codice prima della riga seguente per identificare l'ID dati associato a td:

```
$(evento.target).parents('td').remove();
```

Assegneremo l'ID a una variabile chiamata *idContatti*.
Cerchiamo i contatti dall'archivio Web e convertiamoli nuovamente in un array JavaScript utilizzando *JSON.parse*. Utilizzeremo il metodo del filtro sull'array per conservare gli elementi che non corrispondono a questo ID. Assegnamo l'array appena creato a una variabile chiamata *contattiNuovi*.
Aggiorniamo i contatti nell'archivio Web in modo che l'array archiviato sia l'array appena creato. Ricordiamoci di utilizzare *JSON.stringify* sull'array prima di aggiungerlo all'archiviazione Web.
Potremmo domandarci perché abbiamo bisogno di un ID per identificare in modo univoco i contatti. In genere, l'aggiunta di una chiave come questa è il modo più semplice per identificare in modo univoco un oggetto perché nessuna delle altre proprietà sull'oggetto è garantita che sia unica. Ad esempio avremmo potuto rendere l'indirizzo e-mail la chiave univoca, tuttavia potrebbe essere condivisa da più persone e quindi esposta.

Utilizzare IndexedDB

L'API IndexedDB è notevolmente più avanzata dell'API di archiviazione Web. Pertanto, se l'API di archiviazione Web soddisfasse tutte le nostre esigenze, si può tranquillamente scegliere di saltare questa lezione.
L'API IndexedDB, tuttavia, offre i seguenti vantaggi rispetto all'API di archiviazione Web:
- Consente di memorizzare vari tipi di dati anziché semplici stringhe. Ad esempio, è possibile memorizzare oggetti JavaScript.
- Consente meccanismi di recupero più sofisticati. Ad esempio, è possibile interrogare IndexedDB per un record specifico, come un singolo contatto, in base al suo ID univoco.
- Consente di includere nelle transazioni le operazioni di creazione, aggiornamento ed eliminazione. Questo significa che potremo eseguire una serie di operazioni e garantire che abbiano tutte successo oppure che falliscano. Questo sarà un

concetto familiare per chi ha già lavorato con i database relazionali.
- È in grado di generare automaticamente chiavi univoche per i record. Questo significa che non sarà necessario fare affidamento su meccanismi come l'ora corrente in millisecondi, come visto nella lezione precedente.
- I browser in genere consentono di archiviare molti più dati in IndexedDB rispetto all'archviazione Web.
Inoltre, a differenza dell'API di archiviazione Web, la specifica di IndexedDB non indica quanto spazio di archiviazione deve essere assegnato a ciascun dominio, e i browser in genere o non porranno limitazioni relative allo spazio consentito.
Sebbene queste funzionalità possano essere estremamente utili in alcune applicazioni Web, complicano l'API. L'API IndexedDB è ulteriormente complicata dal fatto che si basa molto sui *callback* (in pratica una funzione da richiamare in un secondo momento) per praticamente tutte le operazioni: piuttosto che semplicemente invocare un metodo e ricevere una risposta, l'API IndexedDB fa affidamento su di noi per registrare i callback per vari scenari. I callback verranno quindi invocati quando si verifica l'evento specificato.
Il motivo principale per cui l'API si basa sui callback è consentire l'esecuzione delle operazioni in background thread se necessario. Ciò significa che se stai eseguendo un'operazione intensiva, il browser potrebbe essere in grado di eseguire questa operazione senza influire sull'esperienza dell'utente.
Come vedremo, l'uso dei callback complicherà notevolmente il codice che bisognerà scrivere perché sarà spesso necessario assicurarsi che un'operazione sia stata completata prima di eseguire la successiva operazione.

Creiamo un database e inseriamoci i dati

In questo paragrafo faremo riferimento a tutto il codice visto in precedenza. Per iniziare a utilizzare l'API IndexedDB per la prima volta, è necessario creare esplicitamente un database, e comunicare all'API i tipi di dati che andremo ad archiviarci.
Aggiungeremo due distinti insiemi di dati al database. Oltre ai contatti, avremo un elenco di aziende di fornitori. Come

vedremo, si accederà a IndexedDB tramite un oggetto fornito dal browser chiamato indexedDB. In Firefox, questo deve essere attualmente referenziato come *mozIndexedDB*, e in Edge è necessario accedervi tramite msIndexedDB. È abbastanza facile creare il tuo proprio alias a questo oggetto che funzionerà in tutti i browser come segue:

```
   databaseProva = indexedDB || msIndexedDB || mozIndexedDB;
```

Il codice seguente, che utilizzeremo per creare il database, dovrebbe essere posizionato alla fine del init in *prova.js* per assicurarci che venga eseguito ogni volta che viene caricata la pagina, vediamolo:

```
   var richiesta = indexedDB.open('DBprova');
request.onsuccess = function(evento) {
}
  request.onupgradeneeded = function(evento) {
}
```

Ci sarà ancora bisogno di fornire implementazioni per queste funzioni o verrà creato un database vuoto.
Come visto, abbiamo specificato che vorremmo aprire un database con un nome specifico, *DBprova* in questo caso. Se il database è già stato creato all'interno del browser (questa non è la prima volta che l'utente ha effettuato l'accesso all'applicazione web), la funzione di callback registrata per l'evento *onsuccess* lo invocherà.
Se il database non è stato creato in precedenza o se si fornisce un numero di versione opzionale come secondo parametro da aprire, verrà richiamata la funzione *onupgradeneeded*. All'interno di questa funzione è possibile definire la struttura del database.
Iniziamo quindi creando una variabile privata denominata *database* e di seguito il codice con il callback:

```
   var inizialized = false;
var database = null;

   var richiesta = indexedDB.open('DBprova');
request.onsuccess = function(evento) {
  database = request.result;
}
  request.onupgradeneeded = function(evento) {
  database = event.target.result;
                    var          contattiDB          =
database.createObjectStore("contatti",
```

```
  {keyPath: "id", autoIncrement: true });
  var aziendeDB = database.createObjectStore("aziende",
  {keyPath: "id", autoIncrement: true });
}
```

La funzione richiamata in caso di *onsuccess* è ragionevolmente semplice. Infatti registra semplicemente un riferimento al database, che inizieremo a utilizzare a breve.
Il callback relativo a *onupgradeneeded* è un po' più complicato. In questa richiamata, inizieremo a fare riferimento al database e quindi a creare due "oggetti archivio" nel database. Ognuno di questi oggetti dovrà ricevere un nome univoco e, inoltre dovranno anche generare ID univoci per i record tramite la proprietà *autoIncrement*.

Infine, da notare che il database deve avere un numero di versione. Se avremo bisogno di cambiarne la struttura, sarà necessario fornire un numero di versione diverso. Ciò ovviamente garantirà la richiamata con *onupgradeneeded*.

Andiamo avanti, nel file *prova.js,* subito dopo il metodo di salvataggio, aggiungiamo questa porzione di codice:

```
   store: function(contatto) {
  var tx = database.transaction(["contatti"],
"readwrite");
  var oggettiDB = tx.oggettiDB("contatti");
  var richiesta = oggettiDB.put(contatto);
  richiesta.onsuccess = function(evento) {
  console.log("Contatto aggiunto " +
evento.target.result);
  }
}
```

Qui vediamo come si inizia indicando al database che vorremo creare una transazione, per iniziare la procedura di salvataggio dei contatti, specificando che i dati saranno letti e scritti.

Tutti gli accessi ai dati con *IndexedDB* devono essere eseguiti nel contesto di una transazione, che potrà consistere in una o più richieste. In questo caso, alla transazione viene

aggiunta un'unica richiesta, che aggiunge semplicemente il contatto all'archivio oggetti pertinente.

A questo punto sarà utile apprendere che le transazioni IndexedDB presentano quattro proprietà identificate con l'acronimo *ACID*:

Atomic: tutte le operazioni nella transazione hanno esito positivo (commit) o tutte le le operazioni falliscono (rollback).

Consistent: il database rimarrà in uno stato valido al termine della transazione come defi nito dalle regole della banca dati.

Isolation: le modifiche apportate dalla transazione sono isolate dalle altre transazioni fino a quando tutte le modifiche non sono state salvate correttamente.

Durable: una volta che le modifiche sono state impegnate, rimangono impegnate, anche se il il database si arresta in modo anomalo subito dopo.

Come possiamo vedere, potremo registrare una richiamata in caso di successo con la richiesta. Questa richiamata viene semplicemente generata dall'ID che è stato assegnato al contatto appena memorizzato.

Potremo anche aggiungere un callback in caso di successo alla transazione stessa, e questo verrà invocato dopo che tutte le richieste nella transazione sono state completate. È importante notare che i record aggiunti non saranno disponibili per altre transazioni (piuttosto che a una richiesta individuale) finchè non è andata a buon fine.

Inoltre, è sempre possibile registrare una richiamata in caso di errore insieme a una richiamata in caso di successo. Questo ci offre l'opportunità di gestire eventuali eventi imprevisti.

Una volta definito il metodo *store*, è necessario richiamarlo durante l'operazione di salvataggio, proprio come abbiamo visto nella lezione precedente. Per esempio:

```
    riga = bind(riga, contatto);
this.store(contatto);
```

Salvando un contatto, nella console dovrebbe essere visualizzato il seguente messaggio: *Contatto aggiunto con ID = 1*
Ogni volta che ne aggiungeremo un altro, l'ID assegnato automaticamente aumenterà di 1.

Lettura e cancellazione dei dati dal database

Ora che abbiamo archiviato i dati, potremo aggiungere funzionalità per leggerli ogni volta che l'applicazione Web verrà aperta. Questa funzionalità eseguirà esattamente le stesse operazioni del metodo *loadContacts* nella lezione precedente, tranne per il fatto che ovviamente leggerà i dati da *IndexedDB*.
La lettura dei dati da IndexedDB introduce un nuovo concetto chiamato *cursor* (cursore). Un cursore è un meccanismo per tenere una serie di record. È quindi possibile scorrere attraverso *cursor* un record alla volta ed elaborarne il contenuto. Iniziamo subito aggiungendo il seguente metodo a *prova.js* subito dopo il metodo *store*:

```
    caricaContatti: function() {
var tx = database.transaction("contatti");
var oggettiDB = tx.oggettiDB("contatti");
oggettiDB.openCursor().onsuccess = function(evento) {
 var cursore = evento.target.result;
 if (cursore) {
  var contatto = cursore.value;
     var   parte   =   $(screen).find('#rigaContatti'   )
[0].content.
  cloneNode(true);
  var riga = $('<tr>');
  riga.data().id = contact.id;
  riga.append(parte);
  riga = bind(riga, contatto);
  $(riga).find('time').setTime();
  $(screen).find('table tbody').append(riga);
  cursore.continue();
 }
}
}
```

Notiamo che questo metodo inizia specificando che vorremmo creare una transazione, ma questa volta non specifichiamo che dobbiamo scrivere i dati. Per impostazione predefinita, le transazioni sono sempre in grado di leggere dati. Una volta creata la transazione, si può semplicemente aprire il relativo archivio oggetti e un cursore dove sono memorizzati i dati. Per impostazione predefinita, questo cursore fornirà un

set di risultati contenente tutti i record nel file relativo. Aggiungeremo un callback in caso di successo del cursore. Questo verrà automaticamente superato il primo record nel cursore, a cui è possibile accedere da *cursore.value* ed elaborare.
Al termine dell'elaborazione del record, si richiama *cursore.continue()*. Questo automaticamente fa in modo che il callback *onsuccess* venga richiamata di nuovo con il record successivo nel set di risultati. Quando il valore di cursore è nullo, sai che tutti i record sono stati elaborati. È necessario fare attenzione a richiamare *loadContacts* solo dopo che il database è stato aperto. In modo da ottenere ciò, aggiungeremo il codice seguente alla logica che apre il database.

```
   var richiesta = indexedDB.open('DBprova');
richiesta.onsuccess = function(evento) {
database = richiesta.result;
this.caricaContatti();
}.bind(this);
```

Si noti che si tenta di leggere i dati dal database solo dopo che IndexedDB lo ha confermato stato aperto con successo.
Se desideri accedere a un record specifico dall'object store, puoi ottenerlo senza elaborazione tutti i record in un cursore. Ad esempio, il seguente codice troverà il contatto con l'ID 4:

```
   var richiesta = objectStore.get(4);
richiesta.onsuccess = function(evento) {
var contatto = evento.target.result;
}
```

Essenzialmente ciò ci garantisce un ottima gestione del database e un buon aumento delle prestazioni generali.
Adesso, per concludere, vedremo come eliminare dei dati. Lo faremo tramite il loro ID, che, come ricorderemo, è reso disponibile attraverso un attributo *data* sull'elemento *tr*. Vediamolo:

```
   delete: function(prova) {
var idContatti =
$(prova.target).parents('tr').data().id;
$(prova.target).parents('tr').remove();
this.updateTableCount();
var tx = database.transaction("contatti", "readwrite");
var oggettiDB = tx.oggettiDB("contatti");
var richiesta = oggettiDB.delete(contactId);
}
```

Questo frammento di codice inizia ottenendo un riferimento all'ID del contatto che viene eliminato. Esso quindi esegue le operazioni familiari di creare una transazione (che deve essere readwrite) e accedere all'archivio oggetti pertinente.
Una volta ottenuto un riferimento all'archivio oggetti, è possibile richiamare il metodo delete e passare il ID pertinente.
Con questo abbiamo solo intuito le enormi possibilità di queste funzionalità, e ovviamente ci sono tantissime altre sfumature che dovranno essere approfondite sui vari siti di divulgazione.

Ora vedremo come potremo lavorare offline con alcuni dati. Grazie al file *manifest* di HTML5, sarà un compito semplice.

Definizione di una cache con manifest

Il file *manifest* contiene un elenco di tutti i file lato client dell'applicazione Web che devono esistere nella cache del browser client per funzionare offline. Ogni file a cui farà riferimento l'applicazione deve essere elencato qui affinché le cose funzionino correttamente. L'unica eccezione è che il file che include il manifest non deve essere elencato; è memorizzato nella cache implicitamente. Adesso proviamolo sul campo, creiamo un file chiamato *notes.manifest*, il suo contenuto sarà:

```
   CACHE MANIFEST
# v = 1.2

   /stile.css

   /javascripts/notes.js

   /javascripts/jquery.min.js
```

Il commento sulla versione in questo file ci fornisce qualcosa che possiamo cambiare in modo che i browser sappiano che dovrebbero recuperare nuove versioni dei nostri file. Quando cambieremo qualcosa, dovremo anche aggiornare la versione del manifest.

Inoltre, abbiamo consentito a Google di ospitare jQuery per noi, ma non funzionerà se vogliamo che la nostra applicazione funzioni offline, quindi dobbiamo scaricare jQuery e modificare il nostro tag script per caricare jQuery dalla nostra cartella javascripts.

```
   <script type="text/javascript"
  charset="utf-8"
  src="javascripts/jquery.min.js" >

   </script>
```

Successivamente, dobbiamo collegare il file manifest al nostro documento HTML. Lo faremo inserendo questa riga di codice:

```
   <html manifest="notes.manifest">
```

Questo è tutto ciò che dobbiamo fare. C'è solo un piccolo problema: il file manifest deve essere servito da un server

Web, perché il manifest deve essere servito utilizzando il tipo MIME text/cache-manifest. Se stai usando Apache, puoi impostare il tipo MIME in un .htaccess come questo:

```
AddType text/cache-manifest .manifest
```

Dopo aver richiesto la nostra applicazione per le note per la prima volta, i file elencati nel manifest vengono scaricati e memorizzati nella cache. Possiamo quindi disconnetterci dalla rete e utilizzare questa applicazione offline tutte le volte che vogliamo.

Assicuriamoci di esaminare le specifiche. Il file manifest ha anche opzioni più complesse da offrire. Ad esempio, potremo specificare che alcune cose non devono essere memorizzate nella cache e non dovrebbero mai essere accessibili offline, il che è utile per ignorare ad esempio video di grandi dimensioni.

Quando lavoreremo con la nostra applicazione in modalità di sviluppo, vorremo disabilitare qualsiasi memorizzazione nella cache nel nostro server Web. Per impostazione predefinita, vengono memorizzati nella cache i file quando indicheremo al browser di non recuperarne una nuova copia per un determinato periodo di tempo. Questo può creare dei problemi mentre aggiungiamo qualcosa al nostro file manifest.

Se utilizziamo Apache, potremo disabilitare la memorizzazione nella cache aggiungendo questo al nostro file *.htaccess*:

```
ExpiresActive On
ExpiresDefault "access"
```

Ciò disabilita la memorizzazione nella cache dell'intera directory, quindi non lo lasceremo attivo per sempre. Tuttavia questo assicurerà che il nostro browser richieda sempre una nuova versione del file manifest.

Come già detto, se dovessimo cambiare un file elencato nel manifest, sarà meglio anche aggiornare la versione indicata all'interno.

Quiz & esercizi

1) Per lavorare con Web Storage e localStorage avremo bisogno di Javascript, di MySQL o di entrambi?
2) Web SQL e MySQL sono la stessa cosa?
3) Completa questo codice per archiviare una stringa qualsiasi con l'API di archiviazione Web: localStorage.
4) Se volessi archiviare un oggetto software cosa dovrei utilizzare?

5) A cosa serve la proprietà *autoincrement*?

6) Per servirsi di una funzione di *callback* servirà una *transazione*?

7) Creare un database con IndexedDB con i calciatori della tua squadra prefetita.

8) La *transazione* potrà può consistere in più richieste?

9) L'acronimo *ACID* è riferito al metodo per caricare dati dal databse?

10) Perché non si può più utilizzare *Web Storage SQL*?

11) Creare un *manifest* di un listato utilizzato nei precedenti esercizi

Riassunto

Nel capitolo più cospiquo del libro si è trattato di archiviazione di dati, sostanzialmente con due metodi, uno più semplice, l'altro più complesso e completo, parliamo di *archiviazione Web* e *IndexedDB*. Abbiamo parlato dei due argomenti in maniera abbastanza approfondita, condendo il tutto con alcuni cenni a JSON, XML e al tag template, utili allo scopo. La conclusione in crescendo con l'utilissima funzione per permettere ad una pagina Web di lavorare offline con *manifest*.

 # 7 - Giocare con le API

History

Cross-document Messaging

Web Sockets e geolocalizzazione

Le API (Application Programming Interface, cioè interfaccia di programmazione per applicazioni) sono nate per risolvere problemi di comunicazione tra computer, ed è proprio con HTML5 che finalmente diventano parte della specifica e sono state infine amalgamate al codice. Altre API sono utilizzate così spesso con HTML5 che a volte è difficile per gli sviluppatori (e anche per gli autori) ricordarsi che non sono parte del linguaggio. In questo capitolo parleremo proprio di questo. Dedicheremo un po' di tempo a lavorare con l'API della cronologia HTML5, quindi faremo in modo che le pagine su server diversi parlino con la messaggistica incrociata, quindi esamineremo Web Sockets e Geolocation, due API molto potenti che possono aiutarci a creare applicazioni ancora più interattive. Per questo scopo utilizzeremo:

- *History*, gestisce la cronologia del browser.

- *Cross-document Messaging*, invia messaggi tra finestre con contenuto caricato su domini diversi.

- *Web Sockets*, creano una connessione tra un browser e un server.

- *Geolocalizzazione*, ottiene latitudine e longitudine dal browser del client.

Andiamo subito a parlare di come funziona la prima dell'elenco.

Come detto, la specifica HTML5 introduce un'API per gestire la cronologia del browser. Quando nel capitolo uno abbiamo creato la pagina del Ristorante Mario non ci siamo preoccupati di aggiungere al codice il supporto del pulsante indietro (back) del browser. Lo faremo adesso servendoci dell'API *History*. Vediamo:

```
function prova(){
```

```
  return !!(window.history &&
window.history.pushState);
    }
```

Useremo questo metodo ogni volta che lavoreremo con *History*.

Quando un visitatore visualizza una nuova pagina Web, il browser aggiunge quella pagina alla sua cronologia. Quando un utente apre una nuova scheda, dobbiamo aggiungere noi stessi la nuova scheda alla cronologia, in questo modo:

```
    $("nav ul" ).click(function(evento){
target = $(evento.target);
  if(target.is("a")){
  evento.preventDefault();
  if( $(target.attr("href")).hasClass("hidden" ) ){
  if (prova()){
  var tab = $(target).attr("href");
  var statoOgg = {tab: tab};
  window.history.pushState(statoOgg, tab);
  };
  $(".hidden").removeClass("visible").addClass("hidden"
).hide();
  $(target.attr("href"
)).removeClass("hidden").addClass("visible").show();
  };
};
    });
    });
```

Catturiamo l'ID dell'elemento visibile e quindi aggiungiamo uno stato della cronologia al browser. Il primo parametro del metodo *pushstate()* è un oggetto con cui potremo interagire in seguito. Lo useremo per memorizzare l'ID della scheda che vogliamo visualizzare quando il nostro utente tornerà a quel punto. Ad esempio, quando l'utente fa clic sulla scheda *Contatti*, memorizzeremo *#contatti* nello stato dell'oggetto.

Il secondo parametro è un titolo che possiamo usare per identificare lo stato nella nostra storia. Non ha nulla a che fare con l'elemento del titolo della pagina; è solo un modo per identificare la voce nella cronologia del browser. Useremo di nuovo l'ID della scheda. Sebbene questo aggiunga uno stato della cronologia, non abbiamo ancora scritto il codice per gestire la modifica dello stato della cronologia. Quando l'utente farà clic sul pulsante indietro, l'evento *window.onpopstate()* verrà attivato. Ci servirà per visualizzare la scheda che abbiamo memorizzato nell'oggetto *stat*e:

```
  if(prova()){
window.onpopstate =function(evento) {
  if(evento.state){
    var tab = (evento.state["tab"]);
    $(".visible")
      .removeClass("visible")
    .addClass("hidden")
    .hide();
    $(tab)
    .removeClass("hidden")
    .addClass("visible")
    .show();
  }
};
  };
```

Recuperiamo il nome della scheda e quindi utilizziamo jQuery per individuare l'elemento da nascondere tramite il suo ID. Il codice che nasconde e mostra le schede viene ripetuto qui dal codice originale. Dovremmo rifattorizzare questo per rimuovere la duplicazione.

Quando apriamo per la prima volta la nostra pagina, lo stato della nostra cronologia sarà nullo, quindi dovremo impostarlo da soli. Possiamo farlo proprio sopra dove abbiamo definito il nostro metodo *window.onpopstate()*.

```
  if(prova()){
window.history.pushState({tab:          "#benvenuto"},
```

```
'#benvenuto');
   window.onpopstate =function(evento) {
     if(evento.state){
       var tab = (evento.state["tab"]);
       $(".visible")
          .removeClass("visible")
         .addClass("hidden")
         .hide();
        $(tab)
         .removeClass("hidden")
      .addClass("visible")
         .show();
      }
  };
};
```

Ora, quando apriamo la pagina, possiamo scorrere le nostre schede utilizzando la cronologia del browser.
Alle applicazioni Web lato client è sempre stato impedito di comunicare direttamente con gli script su altri domini, una restrizione progettata per proteggere gli utenti. Esistono numerosi modi intelligenti per aggirare questa restrizione, incluso l'uso di un proxy lato server e hack degli URL. Ma ora c'è un modo migliore.
La specifica di HTML5 ha introdotto *Cross-document Messaging*, un'API che consente agli script ospitati su domini diversi di ricevere ed inviare messaggi.
Ed è proprio per questo che lo chef Mario vorrebbe aggiungere una pagina di supporto al sito, la quale, per esigenze tecniche avrà bisogno di un altro dominio. Il nuovo sito di supporto di Mario avrà un modulo di contatto e il responsabile del supporto vorrebbe elencare tutti i contatti di supporto e i loro indirizzi e-mail accanto al modulo di contatto. I contatti di supporto proverranno da un sistema di gestione dei contenuti posto su un altro server, quindi potremo incorporare l'elenco dei contatti accanto al modulo utilizzando un *iframe*. Il problema è che il responsabile del supporto apprezzerebbe se potessimo consentire agli utenti di fare clic su un nome dall'elenco dei contatti e aggiungere automaticamente l'e-mail al nostro modulo.

Possiamo farlo abbastanza facilmente, ma dovremo usare i Web server per testare tutto correttamente sulla nostra

configurazione. Quindi gli esempi su cui stiamo lavorando adesso non funzionano in tutti i browser a meno che non utilizzeremo un server.

L'elenco dei contatti

Creiamo prima l'elenco dei contatti. Il nostro markup di base sarà simile a questo:

```
   <ul id="contatti">
<li>
  <h2>Il capo</h2>
  <p classe="nome">Tony Chan</p>
  <p class="e-mail">tonychan@prova.it</p>
</li>
   <li>
<h2>Il secondo</h2>
  <p classe="nome">Mario Rossi</p>
  <p class="e-mail">mariorossi@prova.it</p>
</li>
   <li>
<h2>Il webmaster</h2>
  <p classe="nome">Ciccio Salciccio</p>
  <p class="e-mail">cicciosalciccio@prova.it</p>
   </li>
   </ul>
```

In quella pagina caricheremo anche sia la libreria jQuery che il nostro file *applicazione.js* personalizzato e un semplice foglio di stile. Lo metteremo nella nostra sezione principale:

```
   <script type="text/javascript"
                    charset="utf-8"
src="http://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js">
   </script>
   <script type="text/javascript"
  src="javascripts/applicazione.js">
   </script>
   <link        rel="stylesheet"        href="style.css"
type="text/css" media="screen">
```

Il foglio di stile per l'elenco dei contatti è simile al seguente:

```
 ul{
list-style: none;
 }
 ul h2, ul p{margin: 0;}
 ul li{margin-bottom: 20px;}
```

Sono solo un paio di piccole modifiche per rendere l’elenco un po’ più pulito.

Inserimento del messaggio

Quando un utente fa clic su una voce nel nostro elenco di contatti, prendiamo l'e-mail dall'elemento dell'elenco e pubblichiamo un messaggio nella finestra principale. Il metodo *postMessage()* accetta due parametri: il messaggio stesso e l'origine della finestra di destinazione. Ecco come appare l'intero gestore eventi:

```
   $(function(){
  $("#contatti li" ).click(function(evento){
    var email = ($(this).find(.email".html());
    var origine =
"http://192.168.1.200:4321/index.html";
      window.parent.postMessage(email, origine);
  });
   });
```

Dovremo cambiare l'origine, poiché deve corrispondere all'URL della finestra padre.

Ora dobbiamo implementare la pagina che conterrà questo frame e riceverà i suoi messaggi.

Il sito di supporto

La struttura del sito di supporto apparirà molto simile, ma per mantenere le cose separate, dovremmo lavorare in una cartella diversa, soprattutto perché questo sito dovrà essere posizionato su un server web diverso. Per assicurarci di includere collegamenti a un foglio di stile avremo bisogno di jQuery e di un nuovo file *applicazione.js*.

La nostra pagina di supporto ha bisogno di un modulo di contatto e di un iframe che punti al nostro elenco di contatti. Faremo qualcosa del genere:

```
    <div id="modulo">
  <form id="modulosupp">
  <fieldset>
    <ol>
      <li>
        <label for="a">A</label>
        <input type="email" name="a" id="a">
      </li>
    <li>
        <label for="modulo">Modulo</label>
        <input type="text" name="modulo" id="form">
    </li>
    <li>
    <label for="messaggio">Messaggio</label>
    <textarea      name="messaggio"      id="messaggio">
</textarea>
    </li>
  </ol>
  <input type="submit" value="Invia">
   </fieldset>
   </form>
   </div>
   <div id="contatti">
   <iframe   src="http://192.168.1.200:1234/index.html">
</iframe>
```

```
</div>
```

Lo modelleremo con questo CSS che aggiungiamo a style.css:

```
#modulo{
  width: 520px;
  float: left;
}
#contatti{
  width: 260px;
  float: left;
}
#contatti iframe{
  border: none;
  height: 520px;
}
fieldset{
width: 520px;
 border: none;
}
fieldset legend{
  background-color: #ddd;
  padding: 0 64px 0 2px;
}
fieldset>ol{
  list-style: none;
  padding: 0;
  margin: 2px;
}
fieldset>ol>li{
margin: 0 0 9px 0;
  padding: 0;
}
input fieldset, fieldset textarea{
  display:block;
 width: 500px;
}
fieldset input[type=submit]{
  width: 500px;
}
```

```
  fieldset textarea{
height: 150px;
  }
```

Questo affianca il modulo e l'iframe e modifica il modulo.

Ricezione dei messaggi

L'evento *onmessage* si attiva ogni volta che la finestra corrente riceve un messaggio. Il messaggio ritorna come proprietà dell'evento. Registreremo questo evento usando il metodo *bind()* di jQuery in modo che funzioni allo stesso modo in tutti i browser:

```
   $(function(){
$(window).bind("messaggio",function(evento){
    $("#to").val(evento.originalEvent.data);
});
});
```

Il metodo *bind()* di jQuery esegue il *wrapping* (essenzialmente l'incapsulamento) dell'evento e quindi non rende visibili tutte le proprietà. Possiamo ottenere ciò di cui abbiamo bisogno accedendovi tramite la proprietà *originalEvent*.

L'interazione in tempo reale è stata qualcosa che gli sviluppatori Web hanno cercato di fare per molti anni, ma la maggior parte delle implementazioni prevedeva l'utilizzo di JavaScript per contattare periodicamente il server remoto verificando la presenza di modifiche. HTTP è un protocollo *stateless*, quindi un browser Web effettua una connessione a un server, riceve una risposta e si disconnette, al contrario di un protocollo *stateful*, dove la connessione è continua. Fare qualsiasi tipo di lavoro in tempo reale su un protocollo *stateless* può essere piuttosto difficile. La specifica HTML5 ha introdotto i *Web Socket*, che consentono al browser di stabilire una connessione con stato a un server remoto (per tutte le info: https://websockets.spec.whatwg.org).

Possiamo utilizzare i *Web Socket* per creare tutti i tipi di grandi applicazioni. Uno dei modi migliori per avere un'idea di come funzionano è scrivere un client di chat, che, per coincidenza, Mario desidera per il suo sito di supporto.

Il nostro chef ha intenzione di creare una semplice interfaccia

di chat sul proprio sito che consentirà ai membri del personale addetto di comunicare internamente. Useremo i *Web Sockets* per implementare l'interfaccia web per il server di chat. Gli utenti potranno connettersi e inviare un messaggio al server. Ogni utente connesso vedrà il messaggio. I nostri visitatori possono assegnarsi un soprannome inviando un messaggio come "/nick Paolo", simile al protocollo di chat IRC. In pratica stiamo cercando di creare un'interfaccia chat molto semplice, con un form per modificare il nickname dell'utente, un'ampia area dove appariranno i messaggi e, infine, un form per postare un messaggio in chat.

In una nuova pagina HTML5, aggiungeremo il markup per l'interfaccia della chat, che consiste in due form e un div che conterrà i messaggi della chat:

```
    <div id="intefaccia_chat">
    <h2>Mario risponde</h2>
<form id="nut" action="#" method="post" accept-charset="utf-8">
  <p>
    <label>Utente
    <input id="vsnome" type="text" value="Inserire un nome"/>
  </label>
  <input type="submit" value="Cambia">
  </p>
   </form>
    <div id="chat">collegamento…</div>
    <form id="chat" action="#" method="post" accept-charset="utf-8">
<p>
  <label>Messaggio
  <input id="messaggio" type="testo" />
  </label>
<input type="submit" valore="Invia">
    </p>
    </form>
    </div>
```

Dovremo anche aggiungere collegamenti a un foglio di stile e un file JavaScript che conterrà il nostro codice per comunicare con il nostro server *Web Sockets*:

```
    <script src='Mariochat.js' type='text/javascript'>
</script>
    <link rel="stylesheet" href="style.css"
media="screen">
```

Il nostro foglio di stile conterrà queste specifiche (modificabili se non di vostro gradimento):

```
    #interfaccia_chat{
  width: 510px;
  height: 660px;
  background-color: #ddd;
  padding: 12px;
}
    #interfaccia_chat h2{
  margin: 0;
}
    #chat{
  width: 490px;
  height: 490px;
  overflow: auto;
  background-color: #fff;
  padding: 12px;
}
```

In questo foglio di stile imposteremo la proprietà di *overflow* nell'area dei messaggi della chat in modo che la sua altezza sia fissa e qualsiasi testo che non si adatta dovrebbe essere nascosto, anche se visualizzabile con le barre di scorrimento.

Con la nostra interfaccia in atto, potremo lavorare sul codice JavaScript che lo farà dialogare con il nostro server di chat.

Indipendentemente dal server *Web Sockets* con cui lavoreremo, utilizzeremo lo stesso schema più e più volte.

Effettueremo una connessione al server, quindi ascolteremo gli eventi dal server e risponderemo in modo appropriato, vediamo le possibili risposte ottenute:

- *onclose()* attivato quando non vi è più la connessione con il server

- *onopen()*attivato alla connessione con il server

- *onmessage()* attivato quando il server invia un messaggio

All'interno del file *Mariochat.js*, inseriremo prima la connessione al nostro server Web Sockets:

```
    var connessione = new
WebSocket('ws://localhost:1234/');
```

Quando ci connettiamo al server, dovremmo informare l'utente. Definiamo il metodo *onopen()* in questo modo:

```
    webSocket.onopen = function(evento){
   $('#chat').append('<br>Collegato al server');
};
```

Quando il browser apre la connessione al server, inseriamo un messaggio nella finestra della chat. Successivamente, dobbiamo visualizzare i messaggi inviati al server di chat. Lo facciamo definendo il metodo *onmessage()*:

```
    webSocket.onmessage = function(evento){
  $('#chat').append("<br>" + evento.data);
  $('#chat').animate({scrollTop: $('#chat').height()});
};
```

Il messaggio ci ritorna tramite la proprietà dei dati dell'oggetto evento. Lo aggiungiamo semplicemente alla nostra finestra di chat. Anticiperemo una pausa in modo che ogni risposta rientri nella propria riga, ma potremo contrassegnarla come preferiamo.
Successivamente ci occuperemo delle disconnessioni. Il metodo *onclose()* si attiverà ogni volta che la connessione sarà terminata:

```
    webSocket.onclose = function(evento){
$("#chat" ).append('<br>Connessione terminata');
};
```

Ora dobbiamo solo collegare l'area di testo per il modulo di chat in modo da poter inviare i nostri messaggi al server di chat:

```
   $(function(){
 $("form#chat" ).submit(function(a){a.preventDefault();
  var areatesto = $("#messagge");
  webSocket.send(areatesto.val());areatesto.val("");
 });
})
```

Ci collegheremo all'evento di invio del modulo, prenderemo il valore del campo del modulo e lo invieremo al server di chat usando il metodo *send()*.
Implementiamo la funzione di modifica del nickname allo stesso modo, tranne per il prefisso "/nick" del messaggio che stiamo inviando. Il server di chat lo vedrà e cambierà il nome dell'utente.

```
   $("form#vsnome" ).submit(function(a)
{a.preventDefault();
   var areatesto = $("#vsnome");webSocket.send("/nick "
+ areatesto.val());
});
```

I server di chat sono solo l'inizio. Con i *Web Sockets* disponiamo di un modo solido e semplice per inviare i dati ai browser dei nostri visitatori.
E ora vedremo come acquisire altre informazioni dai nostri visitatori.

Geolocalizzazione

In caso dovessimo, per una qualsiasi motivazione, anche solo statistica, tracciare la la posizione attuale di un visitatore del nostro sito Web su una mappa lo faremo utilizzando la nostra latitudine e longitudine. Possiamo chiedere al browser di acquisire la latitudine e la longitudine del nostro visitatore, in questo modo:

```
    navigator.geolocation.getCurrentPosition(function(po
siz) {
showLocation(posiz.coords.latitude,
posiz.coords.longitude);

    });
```

Questo metodo richiede all'utente di fornirci le sue coordinate. Se il visitatore ci consente di utilizzare le informazioni sulla sua posizione, chiamiamo il metodo *showLocation()*.

Il metodo *showLocation()* prende la latitudine e la longitudine e ricostruisce l'immagine, sostituendo la sorgente immagine esistente con quella nuova. Ecco come implementiamo questo metodo:

```
    var posizione = function(lat, lng){

  var fr = "&markers=color:red|color:red|label:Y|" +
lat + "," + lng;

  var immagine = $("#map");

  var sorg = immagine.attr("src") + fr;

  sorg = sorg.replace("sensor=false", "sensor=true");

  immagine.attr("src", sorg);

    };
```

Invece di duplicare l'intero codice sorgente dell'immagine, aggiungeremo la latitudine e la longitudine della nostra posizione all'origine dell'immagine esistente.

Prima di riassegnare la sorgente dell'immagine modificata al documento, è necessario modificare il parametro del sensore da false a true. Lo faremo con il metodo *replace()* nella penultima riga.

Quando lo apriremo nel nostro browser, vedremo la nostra posizione, contrassegnata da una “Y”.

Quiz & esercizi

1) Per attivare l’API History avremo bisogno di codice diverso dal HTML?
2) Quale metodo attiva History?
3) Completa questo codice:

```
webSocket.onopen =
```

4) Come si attiverà l’evento *onopen()* ?

5) *Cross-document Messaging* funzionerà offline?

6) Scrivere un listato che mostri la nostra posizione geografica in una pagina Web.

Riassunto

Questo breve capitolo dedicato ad alcune API ci ha introdotto in un mondo creato apposta per permetterci di arricchire le nostre pagine Web. Sarà nostra cura quella di aggiornarci e scoprire l’API che fa per noi.

 # 8 - Funzioni avanzate

Le transizioni

Web Workers, Drag & Drop

Il futuro: HTML6

La maggior parte di questo libro si concentra sulle cose che puoi fare adesso, ma ci sono altre cose che potrai iniziare a usare molto presto che renderanno lo sviluppo web basato su standard ancora più interessante, dal supporto 3D canvas con WebGL alle nuove API di archiviazione , transizioni CSS3 e supporto nativo del trascinamento della selezione. Questo capitolo discute alcune delle cose all'orizzonte, così puoi farti un'idea di cosa aspettarti. Parleremo di cose che potresti essere in grado di utilizzare in almeno un browser ma non disponi di soluzioni di riserva sufficienti o sono troppo indefinite per iniziare a lavorare in questo momento:

Le transizioni con CSS3

Gli inviti all'interazione sono importanti per una buona progettazione dell'esperienza utente e CSS supporta da tempo la pseudoclasse *:hover* in modo da poter eseguire alcuni segnali di interazione di base sui nostri elementi. Ecco alcuni markup CSS che stilizzano un collegamento in modo che assomigli a un pulsante:

```
 x.button{
padding: 12px;
border: 1px solid #000;
text-decoration: none;
 }
 x.button:hover{
background-color: #bbb;
color: #fff
 }
```

Quando posizioniamo il cursore sul pulsante, lo sfondo cambia da bianco a grigio e il testo cambia da nero a bianco. È una transizione istantanea. Le transizioni CSS33 ci consentono di fare un po' di più, incluse semplici animazioni che erano possibili solo con JavaScript. Ad esempio, possiamo rendere questa transizione in dissolvenza incrociata aggiungendo il seguente codice evidenziato alla definizione di stile:

```
 x.button{
padding: 12px;
border: 1px solid #000;
text-decoration: none;
-webkit-transition-property: background-color, color;
-webkit-transition-duration: 1s;
-webkit-transition-timing-function: ease-out;
 }
 x.button: hover{
background-color: #bbb;
color: #fff
 }
```

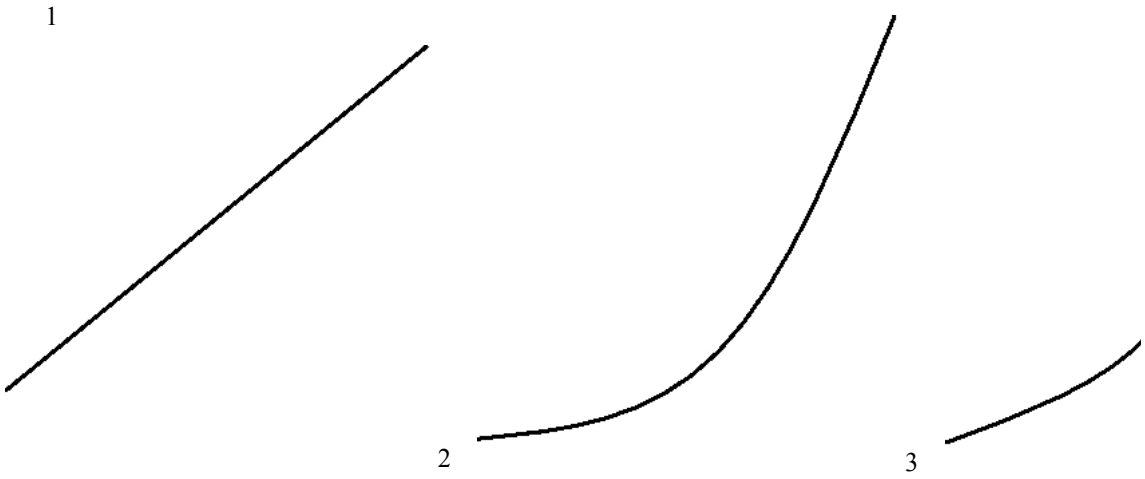
Nelle righe dove abbiamo l'istruzuone *-webkit* specifichiamo a quali proprietà viene applicata la transizione. In questo caso, stiamo cambiando i colori di sfondo e di primo piano. Specifichiamo la durata dell'animazione e specifichiamo la funzione di temporizzazione della transizione.

Gestire il passaggio del tempo

La proprietà di *transition-timing-function* descrive come avvengono le transizioni nel tempo in relazione alla durata che avremo impostato. Specifichiamo questa funzione di temporizzazione usando una curva di *Bezier cubica*, che è definita da quattro punti di controllo su un grafico. Ogni punto ha un valore X e un valore Y, da 0 a 1. Il primo e l'ultimo punto di controllo sono sempre impostati su (0.0,0.0) e (1.0,1.0) e i due punti centrali determinano la forma della curva.

Una curva lineare ha i suoi punti di controllo impostati sui due punti finali, che crea una linea retta con un angolo di 45 gradi. I quattro punti per una curva lineare sono ((0.0, 0.0), (0.0,0.0), (1.0, 1.0), (1.0, 1.0)), e si presenta come la numero uno della tabella a fondo pagina.

Una curva più complessa, con punti ((0.0, 0.0), (0.42,0.0), (1.0, 1.0), (1.0, 1.0)), chiamata curva di andamento, sarà come la numero due.



Questa volta, è cambiato solo il secondo punto, che è ciò che fa curvare la parte in basso a sinistra della linea.
Ancora più complessa è la curva di entrata e uscita, che ha una curva in basso e in alto, come la numero tre della tabella appena vista.
I punti per questa curva sono ((0.0, 0.0), (0.42,0.0), (0.58, 1.0), (1.0,1.0)). Possiamo specificare questi punti direttamente nella proprietà CSS, oppure possiamo usarne alcuni predefiniti come abbiamo fatto nel nostro esempio. Le possibilità sono *default*, *ease-in*, *ease-out*, *ease-in-out*, *ease-out-in*, e *cubic-bezier*, in cui imposteremo noi stessi i punti della curva.
Se vogliamo che la velocità sia costante, useremo *linear*. Altrimenti, se desideriamo che l'animazione inizi lentamente e poi acceleri, useremo la *ease-in*. Questi che abbiamo visto sono solo degli esempi, ci sono molte altre possibilità, basterà approfondire l'argomento sui vari siti specializzati.
Ricordiamoci bene che giocare con le transizioni sarà anche divertente, ma bisogna tenere presente che la nostra interfaccia dovrà essere prima di tutto utilizzabile e poi carina. Non creiamo transizioni che frustrano l'utente, come cose che sfarfallano o impiegano troppo tempo per l'animazione.
Concludendo, con CSS3,5 avremo molte altre possibilità per modificare le proprietà CSS usufruendo di transizioni.

Eseguire codice in background (Web Workers)

I cosiddetti *Web Workers* non sono parte di HTML5, ma potremmo trovarli utili se avessimo bisogno di eseguire alcune elaborazioni in background dal lato client, quindi vale la pena darci un occhiata.
Utilizziamo JavaScript per tutta la nostra codifica lato client, ma questo è un linguaggio a thread singolo: in pratica può succedere solo una cosa alla volta. Se un'attività richiederà molto tempo, si costringe l'utente ad attendere fino al suo termine. I Web Workers risolvono questo problema creando un modo semplice per scrivere programmi simultanei, vediamone un esempio concreto:

```
<!DOCTYPE html>
<html>
```

```
    <body>
    <p>Contatore: <output id="risultati"></output></p>
    <button onclick="startWorker()">Worker Attivo</button>
    <button onclick="stopWorker()">Ferma il Worker</button>
    <script>
    var lav;
    function startWorker() {
  if (typeof(Worker) !== "undefined") {
  if (typeof(lav) == "undefined") {
    lav = new Worker("prova.js");
  }
  lav.onmessage = function(evento) {
    document.getElementById("risultati").innerHTML =
evento.data;
  };
  } else {
  document.getElementById("risultati").innerHTML = "Il browser è obsoleto e non supporta
I Web Worker!";
  }
}
    function stopWorker() {
  lav.terminate();
  lav = undefined;
}
</script>
    </body>
</html>
```

Inanzitutto dovremo avere uno script chiamato *prova.js* che esegue alcune elaborazioni, lo abbiamo così:

```
    lav = new Worker("prova.js");
```

Qualsiasi file JavaScript può essere avviato come *worker*, ma, per la sua indipendenza, lo script di lavoro non potrà accedere al DOM. Ciò significa che non riusciremo a manipolarne gli elementi direttamente.
A parte ciò un *Web Worker* è del codice JavaScript eseguito in background, senza influire sulle prestazioni della pagina.
Infatti, quando si eseguirà uno script nella pagina HTML, la pagina non risponderà fino al termine dello script, mentre un Web Worker, come detto è del codice JavaScript che viene eseguito in background, indipendentemente da altri script, senza influire sulle prestazioni della pagina. Si potrà continuare a fare quello che vogliamo: cliccare, selezionare cose, e così via, mentre il Web Worker continuerà ad essere eseguito in background.

Quindi il nostro codice esterno "prova.js" conterrà:

```
    var z = 0;
    function contatore() {
  z = z + 1;
  postMessage(z);
  setTimeout("contatore()",520);
    }
    contatore();
```

La parte importante del codice sopra è il metodo *postMessage(),* che viene utilizzato per inviare un messaggio alla pagina HTML.

Ovviamente di solito i web worker non vengono utilizzati per script così semplici, ma per attività più impegnative per la CPU, ma questo è un esempio e quindi andrà bene.

Ora che abbiamo il file di lavoro Web, dobbiamo chiamarlo da una pagina HTML. Le righe seguenti controllano se il lavoratore esiste già, in caso contrario, crea un nuovo oggetto Web Worker ed esegue il codice in "prova.js":

```
if (typeof(lav) == "undefined") {
lav = new Worker("prova.js");
}
```

Con questo possiamo inviare e ricevere messaggi dal Web Worker, dopo gli aggiungeremo anche un *listener* (ascoltatore) di eventi, in questo caso *onmessage*:

```
lav.onmessage = function(evento){
document.getElementById("risultati").innerHTML = evento.data;
};
```

Quando il Web Worker pubblica un messaggio, viene eseguito il codice all'interno del listener di eventi. I dati verranno archiviati in *evento.data*.

Quando un oggetto Web Worker viene creato, continuerà ad ascoltare i messaggi (anche dopo che lo script esterno è terminato) fino alla sua chiusura. Per terminarlo e liberare risorse del browser/computer, utilizzeremo il metodo *terminate():*

```
lav.terminate();
```

Impostando la variabile di lavoro su *undefined*, dopo che è stata terminata, potremo riutilizzare il codice:

```
lav = undefined;
```

Questa API funziona proprio come quella per la messaggistica tra domini, di cui abbiamo parlato in *Talking Across Domains*, qui. Quindi se stiamo cercando di svolgere un lavoro pesante sul lato client e non vogliamo restare impantanati, sarà utile approfondire questo aspetto.

Trascina e rilascia (drag & drop)

Consentire agli utenti di trascinare e rilasciare gli elementi dell'interfaccia è qualcosa che siamo stati in grado di fare con le librerie JavaScript per un po' di tempo, ma il W3C ha adottato l'implementazione *drag and drop* come parte della specifica HTML5, vediamone un esempio:

```
    <head>
<script>
  function permettiDrop(xy) {
  xy.preventDefault();
   }
  function drag(xy) {
  xy.dataTransfer.setData("text", xy.target.id);
   }
  function drop(xy) {
  xy.preventDefault();
  var dati = xy.dataTransfer.getData("text");
  xy.target.appendChild(document.getElementById(dati));
   }
   </script>
   </head>
   <body>
   <div id="div1" ondrop="drop(evento)"
ondragover="allowDrop(evento)"></div>
   <img id="prendi" src="prova.jpg" draggable="true"
ondragstart="drag(evento)" width="500" height="250">
   </body>
```

L'implementazione è abbastanza semplice. Prima di tutto: per rendere un elemento trascinabile, impostare l'attributo *draggable* su true:

```
   <img draggable="true">
```

Quindi, specificare cosa dovrebbe accadere quando l'elemento viene trascinato con *ondragstart* e *setData()*.
Nell'esempio precedente, l'attributo *ondragstart* chiama una funzione, *drag(evento)* , che specifica quali dati dovranno

essere trascinati.
Il metodo *dataTransfer. setData()* imposta il tipo e il valore dei dati trascinati:

```
    function drag(xy) {
   xy.dataTransfer. setData("text", xy.target.id);
}
```

In questo caso, il tipo di dati è "text" e il valore è l'id dell'elemento trascinabile ("prendi") .
Una volta che avremo preso il nostro oggetto, con il metodo *ondragover* dovremo specificare dove i dati trascinati potranno essere lasciati.
Per impostazione predefinita, i dati /elementi non possono essere lasciati in altri elementi. Per consentire il rilascio (drop), dobbiamo evitare la gestione predefinita dell'elemento. Questo viene fatto chiamando il metodo *event. preventDefault*() per l'evento *ondragove*r:

```
    event. preventDefault()
```

Quando i dati trascinati vengono rilasciati, si verifica, ovviamente, un evento di rilascio. Nell'esempio precedente, l'attributo *ondrop* chiama una funzione, *drop(evento)*:

```
    function drop(xy) {
   xy. preventDefault();
   var dati = xy.dataTransfer. getData("text");
    xy.target. appendChild(document.getElementById(dati)
);
}
```

Lo spostamento di testo ed elementi nella pagina è solo l'inizio. La specifica consente agli sviluppatori di creare interfacce in grado di ricevere file dal computer dell'utente. Caricare una foto o allegare un file è facile come trascinare il file su una destinazione specificata.

Convalida del modulo lato client

La specifica HTML5 elenca diversi attributi che possiamo utilizzare per convalidare l'input dell'utente sul lato client, in modo da poter rilevare semplici errori di input prima che l'utente invii le richieste al server. E' stato fatto per anni utilizzando JavaScript, ma i moduli di HTML5 possono usufruire di attributi specifici utili in tal senso.
Possiamo assicurarci che un utente abbia un campo obbligatorio da riempire, come ad esempio l'età in caso vi

siano dei contenuti vietati ai minori, aggiungendo l'attributo *required* in questo modo:

```
    <label for="età">Età</label>
<input type="text" name="età" autofocus required
id="età">
```

I browser possono quindi impedire l'invio del modulo e visualizzare un messaggio di errore e non è necessario scrivere una singola riga di convalida in codice JavaScript.
Ciò consentirà agli utenti di sbagliare in anticipo, senza quindi attendere una risposta del server per scoprire se è stato commesso un errore. Tuttavia questo comportamento potrebbe essere disabilitato o non disponibile o semplicemente non implementato correttamente, quindi dovremo comunque assicurarci di avere una strategia lato server per la convalida dei dati. È sicuramente qualcosa con cui dovremo iniziare a ragionare in fase di sviluppo, dato che potremo individuare facilmente i campi richiesti e modellarci l'interfaccia con il CSS in modo che questi si distinguano dal resto.
Oltre a *required* abbiamo anche l'attributo *pattern*, che ci consentirà di specificare un'espressione regolare per assicurarci che il contenuto soddisfi i nostri criteri, nell'esempio potremo inserire solo tre lettere, maiuscole o minuscole:

```
    Nazione: <input type="text" name="nazione"
pattern="[A-Za-z]{3}"  title="La  sigla  della  vostra
nazionalità (es. ITA)">
<input type="submit">
</form>
```

Anche questo è un esempio discretamente utile. E ora facciamo un salto nel mondo della grafica 3D.

Grafica 3D

In questo libro abbiamo parlato del contesto 2D dell'elemento *canvas*, ma è in corso un'altra specifica che descrive come lavorare con gli oggetti 3D. Parliamo, ad esempio della fantastica libreria per Javascript *Three.js*.
Three.js (https://threejs.org/) è una libreria estremamente potente perché ci permetterà, utilizzando Javascript, di realizzare qualsiasi progetto attraverso la grafica tridimensionale fotorealistica, che girerà semplicemente su qualunque computer e smartphone, nel nostro browser. Il suo

utilizzo è relativamente facile, richiede ovviamente una conoscenza di base di Javascript e, altrettanto ovviamente ha una sua sintassi che dovremo imparare. Nello specifico avremo bisogno del solito browser, di un editor di testo come quelli elencati all'inizio del libro, un Web server per permettere il funzionamento delle nostre app, e di un'API Javascript, come WebGL che ci permettera di disegnare grafica 3D sulle pagine web (anche 2D).
Lavorare con la grafica 3D va ben oltre lo scopo di questo libro, ma il sito https://webglfundamentals.org/ offre ottimi esempi ed esercitazioni, anche se in inglese, comunque è possibile utilizzare il traduttore.
Adesso invece scruteremo il futuro con le anticipazioni su HTML6.

Il futuro: HTML6

Sappiamo già che HTML5 è una versione avanzata in esecuzione sul mercato prima che fosse solo HTML. Come abbiamo raccontato in questo libro ha rivoluzionato il Web con alcuni aggiornamenti tra cui l'incorporamento di opzioni video e audio. Gli sviluppatori stanno lavorando su una nuova versione la quale probabilmente si chiamerà semplicemente HTML e che sarà presto rilasciata con diverse novità, di cui alcune avanzate, che andremo adesso ad elencare.
Sappiamo bene che HTML5 è dotato di alcune funzionalità, come supporto audio e video, archiviazione locale offline, capacità di creare e progettare siti Web ottimizzati per dispositivi mobili e libertà da alcuni vecchi tag, che lo hanno reso più popolare rispetto a tutti gli altri concorrenti sul mercato. Ogni sviluppatore sceglie il miglior linguaggio che offre tutto in un unico pacchetto ove possibile e HTML offre questo pacchetto tenendosi sempre aggiornato secondo la domanda e le esigenze. In pratica in questi anni ha sbaragliato il mercato e, la nuova versione vuole mantenere quella leadership.
Per questo motivo HTML6 non si presenterà come un timido upgrade e alcuni nuovi tag, ma con svariate funzionalità avanzate, tutte nuove come:
- *Express Tags*, verranno aggiunti o implementati alcuni tags, come il tag <*logo*></*logo*> per assegnare il logo nelle pagine Web, il tag sidebar (barra laterale) <*sidebar*></*sidebar*>, il

tag navigation per la barra di navigazione <*navigation*> <*/navigation*> e un tag tipo <*div id=”container”*> diventerà più semplicemente <*container*>.
- *Spazi dei nomi (namespaces) simili a XML,* gli sviluppatori si aspettano una struttura simile a XML. Questo ci aiuterà a utilizzare di nuovo lo stesso tag senza entrare in conflitto con un altro. Inoltre avremo altre funzionalità avanzate, ad esempio*, html:title, html:meta,* e molti altri di questo tipo di tag, attiveranno le funzionalità del browser, il primo elemento farà cambiare la barra del titolo del nostro browser e l’elemento multimediale apparirà direttamente sullo schermo del browser.
- *Librerie dedicate,* ampliamente sfruttate in Javascript (jQuery, Three.js) ma, come rovescio della medaglia, un po’ pesanti ed ingombranti, le librerie memorizzate e disponibili nella nostra cache potrebbero essere una realtà nella nuova versione di HTML. Farebbero anche risparmiare del tempo agli sviluppatori.
- *L’annotazione,* che è l’informazione extra (meglio conosciuta come metadati) che abbiamo con le immagini, i video, ecc. verrà aggiunta in HTML6.
- *Ridimensionamento automatico delle immagini,* verrà implementato il sistema di gestione delle immagini e del loro ridimensionamento in base al display utilizzato (aggiornamento del tag <img> e SRC).
- *Integrazione della fotocamera*, oramai svolgono un ruolo fondamentale nella vita di sviluppatori e utenti oggi per riunioni online, interviste, videochiamate ecc. Si prevede che HTML6 avrà un maggiore controllo sulle funzionalità della fotocamera attraverso browser, moduli, ecc. acquisizione di immagini, filtraggio al volo, HDR e visione panoramica, inoltre ci saranno degli effetti avanzati derivati dall’hardware delle nuove fotocamere dedicate.
- *Autenticazione forte*, gli sviluppatori stanno lavorando per rendere i siti più sicuri, cercando di prevenire gli attacchi di hacker e persone non autorizzate, sempre più frequenti, e introducendo alcuni passaggi di autenticazione aggiuntivi come la sostituzione dei cookie con *sign token* e chiavi incorporate che potranno essere archiviati direttamente in un chip hardware. L’API potrebbe essere aggiunta ai browser per

migliorare la sicurezza del sito Web. Ciò impedirà (in teoria) che i dati privati delle persone possano essere hackerati o rubati.
- *Microformati (microformats)*, In HTML6 i numeri di telefono, l'indirizzo e tutte quelle informazioni che potremo definire generali potranno essere definite con i microformati. Le funzionalità più attese sono codici telefonici specifici per paese, la modifica del formato della data, che in questo momento è basato sull'ora degli Stati Uniti. Oltre a questo i microformati permetteranno ai motori di ricerca di trovare meglio le nostre informazioni (se lo vorremmo ovviamente).

- *Pre-processori,* questa tecnologia, già funzionante, permette di rendere omogeneo il codice HTML, con la nuova versione avremo delle implementazioni e sicuramente gli sviluppatori ne trarranno tutti i vantaggi.

- *Privacy e dati personali al sicuro*, si prevede che, attraverso Javascript, nella nuova versione di HTML si potrà impedire ad esempio alle persone di tagliare e incollare informazioni personali, specialmente nella versione mobile. In pratica ci sarà molta più attenzione a questi temi, cercando di rendere il più sicuro possibile tutto ciò che non vogliamo rendere pubblico.

Oltre a questo gli sviluppatori stanno lavorando con alcune funzioni interessanti come fornire una struttura di localizzazione utilizzando il GPS in un browser per telefoni cellulari, incluso il supporto Bluetooth, la protezione da malware integrata e la capacità di trasferire file.
In ogni caso vedremo tutto ciò relativamente presto, sicuramente entro il 2025, almeno così si spera.

Ti ringrazio per aver acquistato questo libro, spero sia stato utile e ti sia piaciuto.

Se vorrai lasciare una recensione su Amazon sarà molto gradita, grazie mille, ti voglio bene.

Tony Chan.

Bibliografia

I sottoelencati siti internet hanno fornito ispirazione per diversi argomenti trattati nel libro:

https://it.wikipedia.org

https://www.codingcreativo.it/

https://www.html.it/

https://www.w3schools.com/

https://stackoverflow.com/

http://www.w3bai.com/it

http://www.codewigs.com/

http://www.xml.com/

http://www.luigisabbetti.it/

http://www.html5italia.com/

E anche il testo: ***Javascript, la guida definitiva*** *di Tony Chan*