Dario Olianas A.A 2013/2014

Appunti di Sviluppo di Applicazioni Web (SAW)

1. HTML
   1.1 Tabelle ed elementi complessi
      1.1.1 Form
   1.2 HTML5
2. CSS
3. JavaScript
   3.1 Document Object Model (DOM)
4. Programmazione lato server
   4.1 PHP
   4.2 Cookies, controllo degli accessi e controllo di sessione
      4.2.1 Cookies
      4.2.2 Controllo degli accessi con HTTP Basic Authentication
      4.2.3 Session control
5. Database
   5.1 Interazione via PHP
   5.2 Funzionalità avanzate: prepared statement e transazioni
6. XML
7. AJAX
   7.1 JSON
8. REST
9. Vulnerabilità
   9.1 Preparazione all'attacco: profiling
   9.2 Tecniche di attacco
   9.3 Difesa
10. Usabilità

1. HTML

HyperText Markup Language, introdotto da Tim Berners-Lee negli anni 90. È un **linguaggio di markup**, ovvero un linguaggio per codificare informazioni sulla formattazione di un documento: dato un testo, dice come deve essere visualizzato. Non è quindi un linguaggio di programmazione. Con un editor WYSIWYG (what you see is what you get) si agisce direttamente sul layout della pagina e sarà l'editor a produrre il codice, invece con un editor che segue un approccio WYSIWYM (what you see is what you mean) siamo noi a scrivere direttamente in linguaggio di markup. È questo l'approccio seguito anche da LaTeX. Un documento in un linguaggio di markup è costituito da due parti:

- il contenuto vero e proprio
- i marcatori, ovvero le istruzioni che dicono al browser come il contenuto deve essere visualizzato

I documenti HTML sono file di testo ASCII con estensione .html o .htm (estensione a 3 caratteri per retrocompatibilità DOS). Iniziano sempre con una dichiarazione DOCTYPE che specifica la versione di HTML utilizzata: può essere la 4.0, la 5 (non ancora standardizzata) o l'XHTML. L'HTML 5 offre una serie di funzionalità che permettono di usare JavaScript il meno possibile, però non è ancora standardizzata; l'XHTML invece è la versione compatibile con XML (raccomandata dal W3C). Il doctype va copiaincollato dai template disponibili su w3c.org. La specifica del doctype serve sia al browser per sapere con che versione di HTML ha a che fare, sia per il controllo delle pagine con un validatore: un programma che, presa in input una pagina, controlla se rispetta tutte le specifiche della versione di HTML utilizzata. Ce n'è uno online su validator.w3c.org. La sintassi di HTML è: **<comando>** `informazioni` **</comando>**, dove il comando è qualcosa che dice come interpretare le informazioni racchiuse tra i due marker di inizio e fine. Dopo la dichiarazione del doctype, c'è sempre un tag **<html>** che viene chiuso a fine documento con **</html>**. HTML è case insensitive, XHTML no. Le informazioni visualizzate dal browser sono quelle racchiuse tra i tag **<body>** e **</body>.** Nella parte prima, **<head>** sono contenute informazioni come il titolo della pagina (**<title>**)e i metadati (autore, tag per la ricerca ecc.). È una parte opzionale.
Nel tag body possono essere specificati una serie di attributi come sfondo della pagina, colore del testo, dei link ecc. con la sintassi `nomeAttributo="valore"` (vedi slides), ma è deprecato: oggi queste informazioni sono contenute nel foglio di stile: questo per permettere di modificare le proprietà di più pagine contemporaneamente (i fogli di stile sono condivisi da più pagine).
I colori sono rappresentati in codifica RGB: un byte per il rosso, uno per il verde e uno per il blu, codificati in esadecimale (da #000000 a #FFFFFF). In questo modo si possono codificare circa 16 milioni di colori. Molti programmi di grafica (tra cui GIMP) quando si usa la palette dei colori fanno vedere anche la codifica esadecimale per l'HTML. Nel testo del documento, all'interno del body, per i titoli si usano i tag **<hN>** `titolo N` **</hN>** con N che va da 1 a 6, per scegliere la dimensione del titolo. Per i paragrafi si usa ** ** `testo` ** **. Grassetto e corsivo si specificano rispettivamente con **<b>** o **** e **<i>** o **.** I tag per colore e dimensione del font sono deprecati in favore di CSS.
L'allineamento si fa con ** ... **
Sintassi per elenchi puntati e numerati:
Puntati:

```
<ul>
<li>first element</li>
<li>second element</li>
</ul>
```

Numerati:

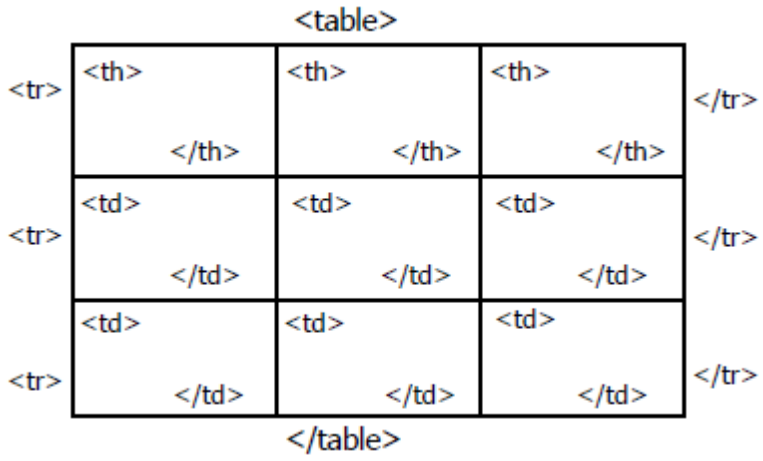
```
<ol>
<li>first element</li>
<li>second element</li>
</ol>
```

L'inserimento di immagini si fa con `<img src="filename/>`, più vari tag per specificare opzioni quali allineamento e dimensione (tutti elencati in slides).I tag height e width non vanno usati per ridimensionare l'immagine ma per dare informazioni al browser sulla sua dimensione. L'immagine va già preparata della dimensione corretta, in particolare quando è piccola: sarebbe infatti è inutile far scaricare all'utente un'immagine di grandi dimensioni per poi farla ridimensionare dal browser, meglio prepararla da subito della dimensione appropriata. Molto importante il tag alt, che permette di specificare un testo alternativo all'immagine: i browser vocali leggono questo. L'immagine va inclusa da un file esterno, e va fatto con un pathname relativo altrimenti quello che funzionava sulla nostra macchina non funzionerà sul server.

I link si fanno con il tag <a>, con parametro href: **`<a href="http://www.example.com">`**`testo da mostrare`**`</a>.` Per link esterni va SEMPRE incluso il protocollo (http://) altrimenti il link non va; per link interni al sito invece posso usare percorsi relativi. Si può scegliere se aprire la pagina nella scheda corrente o aprirne una nuova: generalmente se il link è interno al nostro sito si preferisce aprire la pagina nella stessa scheda, se è un link ad un sito esterno si preferisce aprirne una nuova. I pop-up invece sono deprecati, perché molti utenti li disabilitano. I link possono riferirsi anche ad un altro punto della stessa pagina, molto utile per documenti lunghi che necessitano di indice: si mettono le ancore nei punti della pagina che voglio linkare: la sezione inizia col tag **`<a name="art1">`**`titolo sezione`**`</a>:`** li linkerò nell'indice poi con **`<a href=#art1>`**`articolo 1`**`</a>`**

1.1 Tabelle ed elementi complessi

Le tabelle sono state molto usate in passato per creare i layout delle pagine, allineando gli elementi. Da quanto è stato introdotto il CSS questo utilizzo delle tabelle è deprecato, adesso vanno usate solo per rappresentare dati da mettere in tabella. La sintassi prevede tag di inizio e fine tabella, inizio e fine riga ed inizio e fine cella, in questo modo:



Il tag th indica che la cella è un header, ed il contenuto sarà quindi evidenziato.
Un tempo era possibile impedire la visualizzazione di una pagina di forum postando del codice HTML con una tabella non chiusa dal tag finale `</table>`: questo impediva la corretta visualizzazione della pagina. All'interno di una cella di tabella è possibile inserire un'altra tabella. Il tag table ha molti attributi, oggi deprecati in favore del CSS. L'attributo width permette di dare una

dimensione della tabella, non solo in pixel ma anche in percentuale rispetto alle dimensioni della pagina: in questo modo il browser adatterà la tabella perché occupi sempre la percentuale specificata della pagina. Le pagine che usano questa tecnica sono dette **pagine elastiche**. Gli attributi cellpadding e cellspacing indicano rispettivamente la spaziatura dentro la cella e la spaziatura tra le celle. Anche i tag <th> e <td> (table header per le instestazioni e inizio cella) hanno attributi: width definisce la dimensione della cella (anche questo sia pixel che percentuale). L'attributo colspan si usa per fare il merge tra più colonne.
Quando si usano le tabelle per layout, si imposta il border a 0. Esempi pratici nei link delle slides a w3schools.com.

Frame
Il meccanismo dei frame (oggi deprecato) permetteva di aprire altre risorse all'interno di una stessa pagina, ognuna in un frame. Deprecatissimi per vari motivi, uno dei quali è che l'indirizzo mostrato è quello della pagina principale che contiene il frame: mascheravano la struttura delle pagine. Inoltre non erano accessibili da browser vocali. Un uso frequentissimo era quello di mettere i link alle varie pagine del sito in un frame, che sarebbero poi state aperte in un altro frame.

1.1.1 Form
Un elemento complesso che si usa ancora è invece il **form**, lo strumento che permette di costruire moduli in cui l'utente ci invia dei dati. È l'oggetto che permette di avere il web in cui anche gli utenti fanno contenuto.
Sintassi: si usa il tag `<form action="..." method="..." name"...">`. L'attributo action è il nome del file sul server che riceverà i dati immessi nel form (generalmente uno script PHP), l'attributo name indica il nome, l'attributo method assume i valori POST e GET, come i corrispondnti metodi HTTP. Con il metodo GET i dati immessi nel form vengono inviati nell'header della richiesta HTTP di GET. Viene usato per le stringhe di ricerca dei motori (infatti quello che scriviamo lo vediamo poi nell'URL dei risultati della ricerca) mentre è assoultamente da evitare per i login (se non vogliamo far vedere username e password in chiaro nella barra degli indirizzi).
Con il metodo POST invece i dati vengono inviati nel body della richiesta.
Dopo il `<form>` sono elencati i vari campi del modulo, e alla fine ci sono i pulsanti, definiti dal tag **`<input type="submit" value="Invia"/>`**. Esiste anche un pulsante per cancellare i dati dai campi (con input type reset) ma va controllato tramite JavaScript per chiedere conferma: non vogliamo che l'utente possa cancellare per sbaglio tutti i dati immessi. I campi in cui immettere testo sono definiti da **`<input type="text">`** per le password il type è "password" (visualizza gli asterischi). Si possono avere anche dei campi nascosti (type="hidden"), un tempo utilizzati per mantenere informazioni sulla sessione ma abbandonati perché poco sicuri (oggi si usano i cookies). Ci sono altri input type per avere checkbox, menu a tendina, pulsanti di selezione rotondi a scelta unica. Si possono anche caricare file, con input type "file": ci appare una casella di testo in cui verrà immesso il pathname e un pulsante per sfogliare il nostro filesystem locale e trovare il file da inviare.
Per convenzione, i campi obbligatori si indicano con l'asterisco. Si può obbligare lato client l'utente a immettere tutti i campi obbligatori, ma il controllo va poi ripetuto sul server perché il client può aver disabilitato JavaScript o, nel caso in cui venga usato, non essere compatibile con HTML5.
Questo vale in generale per qualunque controllo fatto lato client: viene fatto solo per migliorare la user experience (l'utente non deve attendere che venga inviata una richiesta al server e che venga ricevuta la risposta solo per farsi dire che i dati non andavano bene), ma per motivi di sicurezza ogni controllo lato client va ripetuto sul server perché non posso sapere se i controlli client siano

effettivamente avvenuti (potrebbero essere falliti per incompatibilità del browser con HTML5 o disabilitazione di JavaScript).

1.2 HTML5

L'HTML5 ha una serie di nuovi tag per stare al passo con l'evoluzione del web evitando il più possibile l'esecuzione di codice JavaScript da parte del browser: ad esempio permette di includere un video nella pagina semplicemente con un tag, e i form supportano il campo "email" che controlla da solo che l'input sia un indirizzo ben formato, senza dover smanettare con JavaScript ed espressioni regolari. Tramite la libreria TinyMCE si può includere un form in cui l'utente può immettere contenuti che saranno converititi in codice HTML da postare nella pagina: approccio WYSIWYG per cui non si produce direttamente il codice.
L'attributo placeholder permette di impostare il testo predefinito per un certo campo, con l'attributo required si obbliga l'utente a compilare quel campi altrimenti il modulo non verrà inviato (vedi discorso di prima su controlli lato client). L'attributo **pattern** permette di definire, tramite espressioni regolari, la formattazione che il testo immesso in un campo deve avere (caratteri consentiti, lunghezza massima e minima e altre impostazioni di formattazione). Ai campi di tipo numerico posso assegnare gli attributi min e max.
Ci sono anche attributi che definiscono quale tastiera deve essere mostrata nella navigazione da smartphone: con "tel" ci viene mostrato il tastierino numerico, con "url" la tastiera QWERTY completa di .com, / e www.
C'è pure la possibilità di scegliere un colore dalla palette, ma non fuzniona su tutti i browser.
Il problema dell'HTML5 è che non è ancora standardizzato, quindi può funzionare in modi diversi (o non funzionare proprio) a seconda del browser.

2. CSS

Il CSS (Cascading Style Sheets) è un linguaggio che definisce formattazione, layout ed altri aspetti visivi di pagine Web, separando quindi il contenuto dalla formattazione. Permette di definire uno stesso stile per più pagine e di rendere più pulito il codice HTML, poiché permette di non utilizzare molti attributi. Velocizza anche il caricamento delle pagine, perché il foglio CSS viene scaricato una volta sola per tutte le pagine che lo condividono. L'ultima versione di CSS è la 3 (non ancora standard). Il CSS consente il responsive web design: avere pagine elastiche che adattano automaticamente le dimensioni a seconda di dove sono visualizzate.
Un documento HTML è suddiviso in blocchi, i pezzi di codice compresi tra un tag di inizio e un tag di fine: sono quindi annidati, e la struttura di questo annidamento può essere vista come un albero (che navigheremo con JavaScript). Il CSS agisce sui blocchi definendo come ogni blocco va visualizzato. I tag ** ** e **** servono solo a definire nuovi blocchi, con l'unica differenza che il browser va a capo ogni volta che incontra un ** **. E' possibile definire informazioni di stile direttamente nei tag con `<div style= 'proprietà:valore'>`. Va però riservato a casi eccezionali poiché viola la separazione contenuto-formattazione.

Un foglio di stile definisce un insieme di regole stilistiche, costituite da un selettore a cui è associato un elenco di coppie proprietà : valore. Il nome del selettore indica di quale blocco stiamo definendo le proprietà visive. Se usiamo i nomi dei tag, poi ogni volta che troviamo quel tag nel documento sarà visualizzato con le stesse regole. Possiamo allora definire degli identificatori: nella pagina HTML lo definiamo con **<p id="my_par">** e useremo come selettore nel foglio di stile **#my_par { proprietà : valore }**. Gli identificatori si usano per un solo elemento, per definire

uno stile che vogliamo usare su più elementi usiamo le classi: definite nell'HTML con **`<p class="my_par">`** e richiamate nel foglio di stile con **`.my_par { proprietà:valore }`**. Le pseudo classi si usano per associare effetti speciali ad alcuni selettori, ad esempio i link che cambiano colore quando ci si passa sopra col mouse. Una pessima funzionalità da usare è quella di far ingrandire il testo di un link quando ci si passa sopra col mouse: si spostano tutti gli altri elementi della pagina.
Le proprietà definibili col CSS sono un bel po' ed è impossibile impararle tutte a memoria: alcune sono nelle slides. Tutte le proprietà con valore numerico sono specificabili sia con varie unità di misura sia con le percentuali: ideali per pagine elastiche.
Si possono definire anche i font, e dal CSS 3 in poi possiamo fornire ai client font esterni che non avevano già nel loro sistema. Non è però una buona abitudine usare font strani nelle pagine. Lo standard per il web sono i font Sans-Serif, quelli senza grazie, perché più facili da leggere (Arial, Helvetica ecc. sono senza grazie, Times New Roman è con grazie). Scrivendo più font separati da virgola fornisco delle alternative: se il sistema del client non ha il primo font verrà usato il secondo, se non ha neanche il secondo verrà utilizzato il terzo e così via.
Firefox ha una funzionalità fighissima che fa vedere, in rilievo in 3D, gli elementi della pagina generati da ogni blocco.
Ci sono tre modi di definire uno stile per le pagine: internal, external e inline. Internal e inline si usano per proprietà specifiche di una sola pagina, mentre l'external (l'inclusione di un file esterno) è la più usata perché permette di usare lo stesso stile per diverse pagine. Si fa con:

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

Esiste una specie di ereditarietà a cascata per gli stili, che in passato provocava incompatibilità.
Oltre alle proprietà visive singolari come colori, font ecc. il CSS permette di definire anche il posizionamento dei vari elementi all'interno della pagina. Ci sono tag che definiscono elementi **block** che vanno a capo automaticamente quando finiscono ed elementi **inline** che vengono visualizzati sulla stessa linea (se non specificato diversamente). Tramite la proprietà position possiamo definire la posizione dei blocchi all'interno della pagina.
Con la proprietà z-index si può definire la "profondità" dei vari elementi, chi verrà visualizzato sopra chi.
La proprietà float permette di organizzare l'allineamento degli elementi, quello per cui prima dell'introduzione di CSS si usavano le tabelle.
Si può anche definire cosa succede quando si passa col mouse sopra un'immagine, ad esempio un cambio di opacità.
Con la direttiva @media si possono definire diverse visualizzazioni a seconda del dispositivo in uso: è possibile ad esempio definire un carattere diverso per visualizzazione su schermo e stampa.
Oggi i siti seguono quasi tutti una specie di standard non scritto per il layout: una sezione in alto detta header, una in basso detta footer e un menu, generalmente laterale. Una volta questo veniva implementato in HTML con le tabelle.

Responsive Web Design

Il responsive web design consiste nell'offrire uno stile diverso per le pagine a seconda della dimensione dello schermo su cui sono visualizzate. Le pagine elastiche fatte usando le misure percentuali del CSS standard infatti non funzionano bene su schermi di smart TV o di smartphone. Questo viene fatto introducendo nel foglio di stile dei resolution breakpoints, che ci dicono, per ogni risoluzione, quale interfaccia offrire. Prima dell'introduzione del CSS 3 il responsive web

design poteva essere fatto lato client tramite JavaScript oppure lato server che leggendo lo user agent della richiesta può sapere il dispositivo del client e quindi la sua risoluzione. Nel CSS 3 i resoluton breakpoints vengono realizzati tramite gli attributi media query, che iniziano con @media.

3. JavaScript

JavaScript (definito nello standard ECMA-262) è l'unico linguaggio utilizzato per lo scripting sul Web: all'inizio c'era un po' di concorrenza ma oggi si usa solo quello. Quando si parla di JavaScript si parla di **programmazione lato client** perché il codice viene eseguito dal browser. Può quindi essere disabilitato dall'utente, anche se oggi disattivandolo non funziona praticamente nulla. Il nome JavaScript è fuorviante perché con Java c'entra poco e niente, hanno solo qualche similitudine sintattica.
L'ambiente di esecuzione JavaScript è composto da:

- **core language**: il linguaggio propriamente detto: interpretato, debolmente object oriented e debolmente tipizzato, con molti costrutti tipici del C e di tutti i linguaggi di alto livello.
- **host,** per interagire con il mondo esterno, che può essere:
    - **client side**: per interagire con il browser del client attraverso l'interfaccia DOM (Document Object Model)
    - **server side**: per la programmazione lato server, prodotta dalla Netscape e adesso non più utilizzata poiché richiedeva web server speciali (Apache non lo supportava).

Uno degli usi principali di JavaScript è quello di verificare la correttezza sintattica degli input di un form, tramite espressioni regolari. L'uso di JavaScript per il controllo degli indirizzi email verrà soppiantato da HTML5, che dispone di un campo email che fa proprio questo lavoro. Si usa inoltre per aprire nuove finestre e per effetti speciali (realizzabili anche con CSS 3).
Permette anche di modificare la pagina corrente senza ricaricarla, ad esempio in un form che richiede un numero variabile di numeri di telefono: all'inizio ci da solo un campo, e un pulsante per aggiungerne altri: quando li aggiungiamo non viene ricaricata tutta la pagina.
Se ci sono errori nel codice JavaScript, il browser non visualizza la pagina: il debug si fa da una console che il browser mette a disposizione (su Chrome vi si accede con Ispeziona elemento -> Console, su Firefox Tools -> Web Developer -> Error Console).
Come per i fogli di stile, gli script possono essere internal, external o inline.

- **internal**: Gli script vengono racchiusi tra i tag **`<script> </script>`**. Una volta nel tag di apertura era richiesto anche il MIME type "text/javascript", ma ora non è più necessario dato che JavaScript è il default. Il posizionamento è importante, perché uno script vede solo gli elementi che sono già stati dichiarati.
- **external**: lo script viene importato da un file .js con la sintassi **`<script src="myfile.js"> </script>`**
- **inline**: si inserisce il codice direttamente nei tag HTML (simile al **`<div style=...>`** del CSS)

Caricamento ed esecuzione
Quando il browser carica una pagina con codice JavaScript, se trova sequenze di istruzioni le esegue, se trova definzioni di funzioni ne verifica la correttezza sintattica. Le funzioni possono

essere associate ad eventi, che ne determinano l'esecuzione: ad esempio in un campi di form in cui posso inserire un numero massimo di caratteri, ci sarà una funzione JavaScript che decrementa il contatore dei caratteri rimanenti associata all'evento pressione di un tasto nel form.

Core language
È un linguaggio di scripting, quindi più flessibile di un normale linguaggio di programmazione: consente alcuni errori e cattive abitudini di programmazione tipo omettere il punto e virgola tra un'istruzione e l'altra (ma solo se si va a capo). Supporta sia commenti C like (`//commento inline`, `/*commento su più righe*/`) che HTML like (`<!-- commento -->`). Gli identificatori possono iniziare con una lettera, $ o _ ma non con un numero. È ovviamente impossibile usare nomi di keyword per gli identificatori. JavaScript è case sensitive. È **debolmente tipato**: la stessa variabile può essere usata per memorizzare dati di tipo diverso e non è obbligatorio dichiarare il tipo di una variabile. Supporta tipi intero (che può assumere il valore NaN in caso di errori), stringa e array: questi ultimi però a differenza degli array C/Java sono oggetti che possono contenere variabili di tipi diversi tra loro, e sono dinamici. Gli oggetti possono essere creati con lo standard **JSON (JavaScript Object Notation)**, con la sintassi `var object { campo1 : val1, campo2:val2..... };`, e ai campi si accede in dot notation o con la notazione degli array associativi/hash table (**`object_name["property name"]`**).
Il linguaggio fornisce oggetti built-in come Date per lavorare con tempi e date e Math che offre una serie di funzioni matematiche e costanti (PI, e).
Non è necessario dichiarare le variabili: se utilizziamo una variabile non dichiarata verrà allocata automaticamente nello scope globale, quindi se vogliamo una variabile visibile solo all'interno di una data funzione dobbiamo obbligatoriamente dichiararla con la keyword **var**.

Eventi
JavaScript è un linguaggio orientato agli eventi: l'esecuzione del codice è condizionata al verificarsi di un evento, che può essere il caricamento della pagina, il caricamento di un'immagine, click del mouse, pressione di tasti ecc. Ad esempio, il codice HTML

```
<img src="dummy.jpg"
  onmouseover = "js code"
  onmouseout = "js code"
  onclick = "js code" />
```

associa codice JavaScript agli eventi passaggio del mouse sull'immagine, mouse che se ne va dall'immagine e click sull'immagine. Un po' di nomi di eventi sono nelle slides. Con gli eventi è possibile modificare il comportamento di default degli elementi HTML, ad esempio far aprire un link in una nuova finestra associando al link il gestire di evento onclick.
Generalmente il codice associato ad un evento non viene scritto direttamente nel tag, ma si usano **funzioni** da invocare, ad esempio `<form ... onsubmit="return check_input()">` dove `check_input()` è una funzione booleana che controlla la correttezza sintattica dei dati nel form: se ritorna false i dati non verranno inviati.

JavaScript non ha costrutti propri di I/O, come del resto neanche C e C++. Ma mentre questi si appoggiano a librerie standard (stdio, iostream) JavaScript si appoggia a librerie del client.
Se vogliamo fare la somma di due numeri presi in input vanno castati ad int, altrimenti l'operazione viene scambiata per una concatenazione di stringhe. Per la moltiplicazione invece il cast è automatico perché non esiste un operatore di prodotto tra stringhe, quindi l'interprete capisce cosa volevo fare.

Via JS si possono chiudere finestre, ma solo quelle aperte da JavaScript, per motivi di sicurezza: altrimenti potrei chiudere il browser.

3.1 DOM (Document Object Model)

Il Document Object Model è uno standard W3C per manipolare gli oggetti di una pagina indipendentemente dal linguaggio utilizzato.
Un documento HTML in una finestra del browser è un oggetto Document con proprietà, metodi ed eventi, ed è accessibile come window.document (anche se window può essere omesso).
A proprietà e metodi dell'oggetto document si accede in dot notation: attraverso le proprietà possiamo vedere informazioni come i cookies della pagina (document.cookie) o da dove sono arrivato a quella pagina (document.referrer).
Ad ogni pagina HTML viene associato un albero, che ha come radice il documento e come nodi tutti i suoi elementi come blocchi, immagini, link, script ecc. organizzati ad albero a seconda di come sono annidati. Quest'albero può essere navigato via JavaScript per modificare elementi "a runtime", durante la visualizzazione della pagina.
Le proprietà di Document permettono di accedere e modificare caratteristiche del documento come colore dello sfondo etc. Molte proprietà restituiscono un array (HTMLcollection) che può contenere ad esempio tutti i link del documento, tutti gli script, tutti i form ecc. Alcune proprietà (come la "name" degli oggetti image) non sono più supportate in HTML5
I metodi open, write, writeln e close dell'oggetto document servono a generare il codice HTML delle pagine dinamiche. Il metodo write, per scrivere il codice HTML della pagina dinamica, non va usato quando il caricamento è terminato altrimenti la pagina sparisce (diventa bianca). Il metodo getElementById permette di accedere agli elementi del documento attraverso il loro id.
Un tipo particolare di oggetto nell'albero DOM è il form, perché è a sua volta un albero che ha come radice form e come figli gli elementi del form (caselle di testo, pulsanti ecc.).
Il DOM dispone gli oggetti di una pagina su un albero, la cui radice è l'elemento <html>. Ogni nodo ha una serie di proprietà come nodeType, nodeName, nodeValue e childNodes. Per il type in particolare, text è il valore di default quindi jon è detto che quando otteniamo un nodeType=text il contenuto sia necessariamente testo. L'albero del DOM può essere navigato graficamente con il DOM Inspector di Firefox.
Il metodo DOM getElementById può essere utilizzato per eliminare completamente il codice js dalle pagine, anche quello dei gestori di eventi. Supponendo di avere un bottone con id B1, invece di mettere nel tag il codice onclick=handlerB1, possiamo scrivere nel file .js una funzione start che trova il nodo con getElementById("B1") e poi con nodo.onclick(handlerB1) (supponendo di aver assegnato il valore dir itorno di getelementbyid ad una variabile chiamata nodo) gli associa il suo handler da eseguire quando viene cliccato.
Con il DOM possiamo non solo accedere alle proprietà degli elementi della pagina, ma anche modificarle dinamicamente con il metodo setAttribute.

Il DOM dispone gli oggetti di una pagina su un albero, la cui radice è l'elemento <html>. Ogni nodo ha una serie di proprietà come nodeType, nodeName, nodeValue e childNodes. Per il type in particolare, text è il valore di default quindi jon è detto che quando otteniamo un nodeType=text il contenuto sia necessariamente testo. L'albero del DOM può essere navigato graficamente con il DOM Inspector di Firefox.
Il metodo DOM getElementById può essere utilizzato per eliminare completamente il codice js dalle pagine, anche quello dei gestori di eventi. Supponendo di avere un bottone con id B1, invece di mettere nel tag il codice onclick=handlerB1, possiamo scrivere nel file .js una funzione start che trova il nodo con getElementById("B1") e poi con nodo.onclick(handlerB1) (supponendo di aver

assegnato il valore dir itorno di getelementbyid ad una variabile chiamata nodo) gli associa il suo handler da eseguire quando viene cliccato.
Con il DOM possiamo non solo accedere alle proprietà degli elementi della pagina, ma anche modificarle dinamicamente con il metodo setAttribute. Possiamo anche creare nuovi elementi come link, immagini, paragrafi ecc. con il metodo createElement, e definirne gli attributi con setAttribute.

Il Web lato client si suddivide in tre livelli: la **struttura** (definita dall'html) lo **stile** (definito dalCSS) e il **comportamento** (JavaScript e DOM)

4. Programmazione lato server

Ad un vero server Web non basta il web server propriamente detto (Apache, IIS ecc.): tipicamente sono presenti anche un DBMS e un interprete PHP per la logica applicativa.
Lavoreremo su webdev.disi.unige.it, che utilizza Apache e offre a supporto MySQL e PHP. Come abbiamo visto a SET, il web server deve offrire accesso pubblico solo ad alcune directory: quella di default è /var/www, ma siccome webdev è condiviso offre anche le cartelle public_html contenute nelle home directory degli utenti. Per accedere alla public html di un utente si usa webdev.disi.unige.it/~utente, altrimenti va su /var/www.
Lavorando in locale, se voglio aprire una pagina dinamica .php non posso aprirla direttamente col browser perché tenterebbe di fare il parsing di codice PHP come se fosse direttamente visualizzabile: bisogna invece passare attraverso un web server, che dobbiamo avere installato sulla nostra macchina e a cui accederemo con http://localhost.

4.1 PHP

Creato nel 1994 da Rasmus Lerdorf, PHP è il più utilizzato linguaggio di programmazione server side. È un linguaggio di scripting interpretato e open source. Una pagina .php contiene sia codice HTML/JavaScript sia codice PHP: quando il server incontra HTML/JS lo invia al client, quando incontra PHP lo esegue. C'è chi per avere codice più pulito preferisce separare HTML/JS dal PHP, facendo scrivere tutto l'HTML da script PHP attraverso l'istruzione echo. Le pagine generate dinamicamente da PHP non sono salvabili in cache, visto che non hanno data di creazione ma nascono su richiesta.
L'inclusione di codice nelle pagine avviene con la sintassi XML **`<?php`** `codice` **`?>`**. Due modi deprecati di includerlo sono **`<?`** `codice` **`?>`** e **`<script language="php"> </script>.`**
Per motivi di sicurezza, la visualizzazione degli errori è disabilitata per default: se vogliamo installare un inteprete in locale bisogna cambiare questa impostazione o includere manualmente in ogni file un'istruzione che fa visualizzare gli errori.

Gli identificatori devono iniziare col carattere $. Come per JavaScript, non richiede la dichiarazione delle variabili: una variabile inizia ad esistere dal momento in cui viene assegnato un valore ad un identificatore. Ci sono quattro scope:

- **local**: variabili locali; visibili solo nella funzione dove vengono usate, cessano di esistere (e se ne perde il valore) quando la funzione termina
- **global**: visibili in ogni scope
- **static**: variabili locali ad una funzione come visibilità, ma che non si perdono quando la funzione termina e che ritroveremo con lo stesso valore quando la funzione viene richiamata. Utile ad esempio per una funzione che conta i visitatori: solo lei deve poter vedere il contatore, ma il suo valore non si vede perdere.
- **parameter**

Le **variabili superglobal** sono variabili con scope globale, generate automaticamente, che contengono informazioni relative al server Web, alla gestione delle sessioni, alle richieste del client e cose simili: è in queste variabili che verranno memorizzati lo user agent, le opzioni dell'header, i cookie, informazioni inviate tramite form, sulle impostazioni del server ecc. Sono array associativi a cui si accede con la sintassi **`nome_array[nome_campo]`**.
PHP è debolmente tipato: non è necessario specificare il tipo delle variabili, che può cambiare dinamicamente (anche se non è una buona abitudine di programmazione assegnare, nel corso del programma, valori di tipo diverso ad una stessa variabile, poiché riduce la leggibilità). Si possono definire costanti con **`define("NOMECOSTANTE", valore)`**. Per vedere i valori delle costanti predefinite si usa la funzione **`phpinfo()`** (che su un server reale è bene disabilitare per sicurezza, visto che fornisce un sacco di informazioni sulla configurazione della macchina).
La maggior parte delle istruzioni sono simili a C/Java; ci sono due delimitatori di stringa ' e ": l'apice doppio agisce tipo una stringa di formattazione per la printf, se ci scriviamo dentro nomi di variabili stampabili le stampa, mente con l'apice singolo non possiamo scrivere i nomi delle variabili dentro la stringa ma dobbiamo concatenarla. La concatenazione di stringhe si fa col punto: **`stringa 1 . stringa2`**. La stampa in output si fa con **`echo`**, **`print`** e **`print_r`**. Quando si scrive in output dobbiamo stare attenti al ritorno a capo: se stampiamo con **`\n`** il browser non andrà a capo, bisogna usare **`<br>`**. Si può fare output multilinea con l'operatore <<< di `echo` (esempio nelle slides).
Nei casi in cui dovessimo ripetere lo stesso codice su più pagine possiamo metterlo in un file a parte ed includerlo con **`<?php include("filename.php"); ?>`** Per evitare di includere due volte lo stesso file possiamo usare **`require_once`**. Questi file da includere **NON** devono contenere i tag `<html>` `<head>` ecc. perché sono parti che vanno incluse in una pagina, ed una pagina HTML valida i tag di header ce li ha una volta sola.
Tramite PHP riceviamo i dati immessi in un form dal client, che vedremo nelle variabili superglobali **`$_GET`** o **`$_POST`**, a seconda del metodo usato dal form per l'invio. Queste variabili sono array associativi, e i nomi delle celle saranno i nomi dei campi del form: per questo motivo nei form HTML non vanno messi due input con stesso name, altrimenti otteniamo solo il valore dell'ultimo. Per i radio button non c'è problema perché sono a selezione singola: avremo una posizione con il nome condiviso dai tutti i radio button per una stessa scelta, e il valore dipenderà da quale è stato selezionato. L'identificazione tramite id invece che name non è supportata.

Con la direttiva di configurazione **`register_globals`** si può creare una variabile globale per ogni campo di form ricevuto invece di usare gli array associativi **`$_GET`** ed **`$_POST`**, in modo da usare lo "short style": accedo col nome del campo invece che con la sintassi di accessi degli array. È una funzionalità deprecata per motivi di sicurezza.
Programmando in PHP bisogna stare estremamente attenti alla sicurezza, poiché uno dei compiti più frequenti è quello di ricevere input dai client, e tra i client ci sono pure gli attaccanti: quindi **ogni input è malevolo fino a prova contraria**. Gli input andranno controllati attentamente

per evitare SQL injection e altri attacchi basati sull'invio di codice malevolo via PHP. Quindi PHP fornisce una serie di funzioni per ripulire i dati in input, ad esempio per eliminare gli spazi (trim), sostituire i \n con i
 (per produrre contenuti HTML validi), convertire i caratteri speciali nelle entità HTML corrispondenti (per evitare JS injection) e per inserire slash di escape (per evitare SQL injection).
PHP supporta le espressioni regolari con la sintassi del PERL.
I dati possono essere inviati sia tramite GET che tramite POST: il GET è sconsigliato non solo per le password, per ovvi motivi di sicurezza (ci troveremmo la password in chiaro nella barra degli indirizzi) ma anche per input di testo di lungezza arbitraria: i dati inviati via GET infatti hanno una lunghezza massima, superata la quale vengono troncati: anche in questo caso meglio usare POST.
Per ripulire l'input alcune delle funzioni più importanti sono htmlentities, che converte ogni simbolo parte del linguaggio HTML (maggiore e minore ecc.) nell'entità HTML corrispondente. Può servire ad evitare iniezioni di codice nei commenti: un rudimentale attacco verso un forum infatti era di postare codice HTML o JavaScript nei commenti. In questo modo invece non si riescono ad inserire i tag. La funzione addslashes invece introduce sequenze di escape nelle stringhe davanti ai caratteris peciali: fondamentale per prevenire le SQL injection. Ci sono anche una serie di funzione per il cast (tutte nelle slides).
Ci sono due modi di salvare i file sul server: in un database o in un normale file. La gestione dei file è molto simile al C: abbiamo i file pointer che corrispondono ai file descriptor e una serie di funzioni per manipolarli (fopen, fclose, fread ecc.).
Per motivi di sicurezza, quando salviamo dati su un file questo deve essere al di fuori della document root del web server: altrimenti, indovinandone il nome, chiunque potrebbe accedervi.
Bisogna anche assicurarsi che l'utente che esegue il aebs erver (solitamente www-data) abbia i permessi di scrittura sul file in questione: questo va fatto tramite il sistema operativo.
Il server potrebbe anche ricevere più richieste contemporanee di accessi allo stesso file: va quindi gestita la concorrenza, attraverso la funzione flock. Per la scrittura su database invece il problema non si pone perché la serializzazione delle richieste è gestita in automatico dal DBMS.

4.2 Cookies , controllo degli accessi e controllo di sessione

HTTP è un protocollo stateless: il web server non ha memoria delle richieste precedenti di uno stesso client. Per poter riconoscere uno stesso client, sono state introdotte funzionalità come i cookies, il controllo degli accessi e il controllo di sessione.

4.2.1 Cookies

Un cookie è una porzione di testo contenente informazioni utili ad identificare univocamente un client. Il web server rilascia i cookies attraverso l'header Set-Cookie della risposta HTTP. È suddiviso in questi campi

- name
- expires
- path
- domain
- secure

di cui solo name è obbligatorio. Expires memorizza la data di scadenza e domain il dominio di chi lo ha rilasciato: ogni volta che si ritorna su quel dominio il browser rispedisce il cookie al server nella richiesta HTTP.

Ci sono alcune limitazioni: un cookie può pesare al massimo 4 KB, se ne possono memorizzare al massimo 300 in tutto e 20 per dominio (queste restrizioni possono variare da un browser all'altro). Per cancellare un cookie, gli si può assegnare una data di scadenza passata o un valore nullo.
In PHP i cookie si possono creare con la funzione **`setcookie(string name, string value, int expire, string path, string domain, int secure)`**, e quelli ricevuti dai client vengono messi nella variabile superglobale `$_COOKIE` . La data di scadenza può essere creata con la funzione PHP `mktime`.
Per cancellare un cookie, gli si può assegnare una data di scadenza passata o un valore nullo.
Un errore frequente è quello in cui viene inviato dell'output prima che sia finita la preparazione degli header, a causa di codice mal posizionato: in questo caso gli header vengono automaticamente chiusi non appena si arriva all'istruzione che invia l'output, e quando si arriva alla parte di codice che manipola l'header si ottiene un errore. Oltre che riposizionando il codice, questo problema può essere corretto con la bufferizzazione dell'output: un'impostazione del server PHP che attende che l'header sia pronto prima di inviare i dati, anche se nel codice le due cose avvengono in ordine sbagliato, ma questa feature è disabilitata sul nostro server webdev.
Quando memorizziamo le password sul server, è opportuno usare due codifiche, solitamente SHA1 e MD5, per evitare attacchi a dizionario in cui l'attaccante dispone della codifica MD5 di tutte le parole di senso compiuto e delle password di uso più frequente. Se usiamo due cifrature gli complichiamo la vita di parecchio.

4.2.2 Controllo degli accessi con HTTP Baisc Authentication

Essendo HTTP un protocollo stateless, bisogna usare degli accorgimenti per mantenere l'informazione che un utente autorizzato si è già loggato senza bisogno di chiedergli l'autenticazione ad ogni pagina. Tecniche di controllo degli accessi sono usare un URL nascosto (pessima perché l'unica protezione che fornisce è che per trovare la pagina bisogna conoscerne l'URL), controlli basati sull'indirizzo IP (va bene nelle intranet, ma sarebbe problematica per un sito pubblico dove i client possono avere IP dinamici) oppure verificare l'identità dell'utente (la tecnica giusta per il web).
HTTP fornisce un controllo degli accessi, la **Basic Authentication** che mostra al client una finestra di dialogo in cui inserire le credenziali. Viene impostata a livello di configurazione del web server, non c'è nulla da programmare: basta mettere un file .htaccess nelle directory che vogliamo proteggere, in cui metteremo tre linee AuthType (codifica usata per le credenziali) AuthName e AuthUserFile, che contiene il pathname assoluto del file che contiene le credenziali degli utenti. Altrimenti, l'informazione sulle directory da proteggere può essere messa direttamente nel file di configurazione di Apache (la sintassi precisa è nelle slides, ma potrebbe contenere errori) La sintassi per il file di configurazione Apache si trova anche su http://webdev.disi.unige.it/phpexamples/week7/protected/ (login marina:marina). Il file con le password si crea col comando htpasswd, e va salvato al di fuori della document root. Nel file .htaccess possiamo anche configurare messaggi di errore personalizzati.
Questo metodo di autenticazione è inefficiente quando si ha un grande numero di utenti.

4.2.3 Session control

Corrisponde al livello Session dello stack ISO/OSI. È fatto via PHP lato server, quindi è abbastanza sicuro (a meno che qualcuno non stia sniffando il traffico). Si serve di un identificatore univoco, il **session ID**: una stringa abbastanza lunga che identifica l'interazione corrente e che, a differenza di un cookie, viene perso quando chiudiamo il browser. Le informazioni di sessione vengono inviate attraverso i cookies: se sono disabilitati verranno inviate nell'URL.
Fasi di session control:

- inizio sessione
- registrazione delle variabili di sessione dopo aver chiesto l'autenticazione al client (se necessario)
- usare le variabili di sessione
- distruggere le variabili di sessione e chiudere la sessione

Le sessioni vengono gestite tramite funzioni PHP. La funzione `session_start()` verifica se l'utente ha già un session ID: se non ce l'ha ne genera uno, altrimenti rende accessibili le variabili di sessione già create per quell'utente. Le variabili di sessione vengono memorizzate nella superglobale `$_SESSION`. Le pagine che hanno bisogno di sapere se l'utente è loggato o no avranno un `if(isset($_SESSION["myvar"]) {..} else {...}`. Per cancellare un identificatore di sessione si usa `session_destroy()`.

5. Database

MySQL ha due motori, che differiscono a livello fisico: MyISAM (default) e InnoDB. MyISAM è il più efficiente, ma non supporta le transazioni: nei casi in cui ci servono dobbiamo usare InnoDB. Può gestire più utenti con diversi privilegi, tra cui un utente root da usare solo per l'amministrazione. Una buona norma di sicurezza è di definire due utenti per ogni applicazione: un power user che ha il permesso di modificare le tabelle, e un utente normale che può fare solo interrogazioni.
Il controllo dell'accesso di MySQL si compone in due fasi

- verifica delle credenziali dell'utente
- per ogni operazione richiesta, verificare se l'utente ha i privilegi per eseguirla

I privilegi degli utenti vengono memorizzati nelle grant tables, tabelle di sistema che memorizzano per ogni utente che privilegi ha.
Esistono quattro livelli di privilegi: Global, Database, Table, Column, e vengono assegnati e revocati tramite i comandi GRANT e REVOKE di MySQL. Per sicurezza vanno assegnati secondo il principio di Denning del minimo privilegio: ogni utente deve avere solo i privilegi per fare quello che deve fare, e nulla di più.

5.1 Interazione via PHP

L'interazione tra il server web e il database MySQL di backend si compone in sei fasi:

1. Controllare e filtrare i dati ricevuti dall'utenti (prevenzione delle SQL injection)
2. Stabilire una connessione e selezionare un database
3. Interrogare/modificare il database
4. Ricevere il risultato
5. Preparare il risultato e inviarlo all'utente
6. Chiusura della connessione

I passi 2, 3, 4 e 6 possono essere eseguiti da funzioni di libreria PHP, che iniziano in `mysqli_` per l'interazione con MySQL e `pg_` per l'interazione con PostgreSQL. Sul web potremmo trovare funzioni analoghe che iniziano con prefissi diversi, sono deprecate. Esistono anche dei driver, librerie che offrono funzioni ad un livello di astrazione più alto per l'interazione con ogni DBMS:

hanno un parametro in più, una stringa in cui specifico quale DBMS sto usando e sarà compito della libreria invocare la funzione di livello inferiore per l'interazione con quel DBMS. Sono utili perché permettono di cambiare DBMS semplicemente modificando una strigna, senza dover cambiare tutto il codice.
Il controllo dei dati in arrivo lo abbiamo già visto: esistono funzioni da chiamare sulle stringhe che aggiungono caratteri di escape ecc. Il controllo dei dati può essere automatizzato tramite le direttive "magic quotes" del file php.ini, ma è buona norma farlo sempre a mano, altrimenti avremmo del codice che dipende dalla configurazione dell'interprete PHP, e questo non va bene perché non sempre abbiamo i permessi per modificarla.
La connessione avviene con la funzione `mysqli_connect(host, username, password, database)`. Il valore di ritorno va assegnato ad una variabile, su cui invocheremo la `mysql_connect_errno` per controllare se la connessione è andata a buon fine. La variante `mysqli_pconnect` apre una connessione persistente, rischiosa per server con molto traffico perché espone ad attacchi DoS (l'attaccante mi apre un sacco di connessioni a vuoto, rimangono attive perché sono persistenti, esaurisco la banda disponibile e i client legittimi non riescono a collegarsi).
Dopo la connessione, in caso ci siano più database sul server se ne seleziona uno con `mysqli_select_db`. Ai diversi database dovremmo accedere con lo stesso utente però, non possiamo specificarne un altro: se l'utente utilizzato nella connect non ha i permessi per accedere al database selezionato successivamente con select_db, la chiamata fallisce.
Al terzo step si prepara la query, unendo i dati ricevuti, tramite la concatenazione tra stringhe, ai comandi SQL necessari.
Quando la query è pronta, va inviata al server con la funzione `mysqli_query($con,$query)`, dove `$con` è l'handler della connessione (il valore ritornato dalla connect) e `$query` è la stringa contenente la query. Il valore di ritorno sarà false in caso di errore, e un'identificatore di risorsa che permette di accedere ai dati restituiti dalla query.
Al contenuto della risposta si accede con le funzioni `mysqli_fetch_*`, che restituiscono riga per riga il risultato memorizzato in diverse strutture dati:

- `mysqli_fetch_assoc` restituisce un array associativo, in cui i nomi delle celle sono i nomi degli attributi del risultato
- `mysqli_fetch_array` restituisce un array
- `mysqli_fetch_object` restituisce un record, i cui campi hanno il nome degli attributi

A questo punto va prodotto il codice di una pagina HTML contenente il risultato: anche questo viene fatto via PHP.

In alcune situazioni potrebbe essere utile fornire moduli precompilati: per farlo si può aggiungere un campo hidden al form, al cui value assegneremo un <?php echo"$row" ?>, dove $row è il risultato di una query dei dati che vogliamo mettere nel form.

5.2 Funzionalità avanzate: prepared statement e transazioni

Se dobbiamo fare tante query dello stesso tipo, ma con valori diversi, è possibile fornire al DBMS un template di query, in modo da dover passargli solo i valori. In PHP il template può essere preparato con **`mysqli_prepare($con, $stmt)`** dove `$con` è un handler di connessione e `$stmt` la stringa del template, con un punto interrogativo al posto dei parametri. Bisogna poi specificare i

tipi dei parametri, con **`mysql_stmt_bind_param`**, a cui va passato lo statement, la variabile che conterrà il parametro e il tipo, definito da un carattere (vedi slides). Per l'esecuzione della query si usa **`mysqli_stmt_execute($stmt)`**. Sembra più lungo dal punto di vista del codice, ma l'esecuzione è più efficiente perché il DBMS lavorerà più velocemente, ed è anche più sicuro perché aiuta a prevenire le SQL injection.

Le transazioni sono un insieme di comandi SQL atomici: o vanno tutti a buon fine o non viene eseguito nulla. L'esempio tipico è quello del trasferimento di denaro: se fallisce il prelievo deve fallire anche l'accredito, altrimenti avrei creato soldi dal nulla. Per fare questo è necessario dire al DBMS dove inizia e dove finisce la transazione: se si arriva alla fine senza errori si esegue il **commit**, che conferma la transazione, altrimenti si esegue il **rollback**: annulla la transazione riportando il database in uno stato di consistenza.

6. XML

XML è il linguaggio di markup standard per lo scambio di dati tra applicazioni. Come l'HTML prevede dei tag, che a differenza dell'HTML non sono predefiniti: è un linguaggio di markup libero, lo standard definisce solo la struttura che il documento deve avere ma i nomi dei tag sono a scelta dell'utente. Nasce dall'esigenza di dover rappresentare informazioni strutturate o semi-strutturate. Ultimamente, soprattutto nell'ambito del web 2.0, sta venendo soppiantato dallo standard JSON, più semplice.
Ci sono due classi di applicazioni dell'XML: document-centric, per rappresentare documenti strutturati o semi-strutturati come manuali, cataloghi e documenti legali, e data-centric, per rappresentare dati strutturati come record di un database relazionale o informazioni su transazioni finanziarie.
Un documento XML si compone di struttura, contenuto e presentazione: la struttura è definita dallo schema, il contenuto dal documento, la presentazione dall'XML transformation.

Contenuto
Un documento XML è un documento di testo con estensione xml e mimetype `application/xml`. Inizia con un prologo, che definisce la versione di XML e il set di caratteri utilizzato. Se il documento andrà scambiato su protocolli come l'SMTP basati sula codifica ASCII a 7 bit, va usato il set utf-8. All'interno del documento, le informazioni possono essere strutturate gerarchicamente in elementi e sotto-elementi oppure a contenuto misto (mixed content), Come già detto i tag sono definiti dall'utente, possibilmente con nomi significativi per il contesto applicativo. Nel documento possiamo trovare sia tag semplici che con attributi, con la stessa sintassi dell'HTML. A differenza dell'HTML la sintassi è molto rigida: se dimentico le virgolette agli attributi, per esempio, il documento non è considerato valido e il parser mi darà errore. I caratteri > e < sono parte del linguaggio e non possono essere usati, vanno sostituiti con le entità HTML `&lt` e `&gt`. Anche & è riservato e va sostituito con `&amp`. I nomi degli attributi per uno stesso elemento devono essere diversi.
I documenti XML possono essere composti tra loro per creare nuovi documenti, anche se questo può causare collisioni dei nomi dei tag. Per evitare ciò si usano i namespace: ogni volta che definiamo un documento con un suo set di tag e attributi definiamo un namespace che li identifica (in modo simile ai namespace del C++ che evitano collisioni tra funzioni di librerie diverse), in

modo che se anche quel documento venisse unito ad un altro tutti i tag e gli attributi continuerebbero ad essere univoci.

Schema
L'XML schema è un meta linguaggio che usa XML per definire la struttura di un documento XML. Viene specificato ad inizio documento dal tag `<xs:schema xmlns:xs="url dello schema">`. Siccome è complicato definire uno schema, generalmente viene fatto da un software (XMLSpy).

L'XML, a differenza dell'HTML, non viene renderizzato dai browser che ci fanno invece vedere il sorgente. Se vogliamo visualizzare il contenuto di un documento XML, e non il sorgente, dobbiamo trasformare il markup XML in markup di presentazione tramite software, fogli CSS (molto poco potenti) o trasformazioni XSLT.
Esiste una libreria, SimpleXML, che traduce documenti XML in oggetti.

7. AJAX

AJAX è una tecnica di programmazione che permette di sviluppare pagine interattive, che si modificano dinamicamente senza bisogno di essere ricaricate, grazie ad interazioni asincrone con il server. AJAX permette di resittuire al client più tipi di dato in una sola risposta. La novità più grande che ha introdotto è quella di poter caricare dinamicamente porzioni di pagina, senza dover ricaricare tutto (se ad esempio carico un'immagine che verrà poi mostrata nella pagina, può essere caricata e integrata senza dover ricaricare tutto). Modifica l'albero DOM aggiungendo o togliendo nodi dinamicamente. È il metodo usato da Google Maps per caricare i pezzi di mappa.
Prima di AJAX si usavano le applet Java, i frame e l'**XMLHttpRequest** di Microsoft: un oggetto che permette di eseguire richieste HTTP via Javascript. Sarà proprio questo XHR, iniziato ad usare in maniera non convenzionale a partire dal 2002, a dare origine alla tecnica di sviluppo AJAX.

Abbiamo già visto un modo per modificare dinamicamente le pagine: JavaScript. Ma questo ha una serie di limitazioni, come l'impossibilità di interagire col server e il fatto che, essendo client side, il codice JavaScript è sempre visibile al client: dato che è interpretato direttamente dal browser non possiamo in nessun modo nasconderlo all'utente, mentre a volte vorremmo che lato client fosse impossibile vedere la logica applicativa che sta dietro al nostro sito.
La tecnica AJAX invece permette a Javascript di eseguire richieste HTTP senza che l'utente debba fare nulla: da Javascript vediamo un oggetto XMLHttpRequest messo a disposizione dal browser, che permette di eseguire chiamate HTTP. Essendo messo a disposizione dal browser, ci sono istruzioni diverse per crearne un'istanza a seconda di che browser stiamo usando. Una volta creato ne possiamo usare i metodi, come ad esempio open per aprire una connessione. Per open in particolare, il parametro `async` va messo (generalmente) a true, in modo da avere chiamate

asincrone: altrimenti con una chiamata sincrona il browser si pianta ad aspettare la risposta e non fa nulla finché non arriva.
La funzione send invece invia la richiesta: se è con metodo POST ha un parametro data in cui mettere i dati da inviare, se è con metodo GET i dati sono nella URL (con sintassi ?nomeParametro1=valore&nomeParametro2=valore). Se eseguiamo una chiamata tramite POST bisogna inoltre impostare gli header, attraverso il metodo setRequestHeader dell'oggetto XHR, per specificare che tipo di dati stiamo inviando: per coppie nome-valore va impostato l'header Content-Type a **`application/x-www-form-urlencoded`** : in questo modo lato server troveremo le coppie nell'array superglobale **`$_POST`** come se fossero state inviate da un form.
Essendo un oggetto, XHR non ha solo funzioni ma anche proprietà che permettono di conoscere lo stato della richiesta. In particolare readyState ci dice lo stato della richiesta (un intero da 0 a 4: 0 richiesta non inizializzata, 1 richiesta inizializzata, 2 richiesta inviata, 3, in attesa di risposta dal server, 4 risposta ricevuta), responseText e responseXML contengono il contenuto della risposta a seconda che sia in XML o di altro tipo, status contiene il codice di stato della risposta HTTP (200, 404 ecc.). Per le chiamate asincrone la proprietà onreadystatechange dice quale funzione va eseguita quando lo stato della richiesta cambia: questa funzione, chiamata funzione di callback, dovrà controllare le proprietà readyState e status per verificare se la richiesta è andata a buon fine.
Uno degli usi di AJAX può essere di controllare se l'e-mail (o altro campo univoco) inserita dall'utente in registrazione è già presente nel database.
La risposta può arrivare in XML, JSON o testo semplice. Le risposte XML finiscono nel campo responseXML, le altre nel campi responseText. Nel caso la risposta sia in JSON la dobbiamo interpretare con JSON.parse, se invece è in XML può essere trasformato in HTML tramite operazioni sul DOM.

Ma AJAX non ha solo pregi: se il server deve fare grossi calcoli, o se la rete è lenta, avremo dei ritardi nella visualizzazione della pagina; il pulsante Indietro perde il suo significato (perché torna alla pagina precedente, non annulla le modifiche effettuate da AJAX: l'utente potrebbe non saperlo e meravigliarsi quandov a più indietro di quello che pensava), più alcuni problemi di sicurezza e compatibilità.
Ci sono un sacco di framework per Javascript/AJAX, tra cui jQuery è il più utilizzato. Permette di selezionare elementi del DOM ed eseguire azioni su di essi, con la sintassi $(selector).action().
Un altro framework è DataTable, per la visualizzazione di tabelle risultato di query SQL. Ce le suddivide già in pagine, ci permette di riordinare il risultato cliccando sui nomi delle colonne ecc.

7.1 JSON

La notazione JSON (JavaScript Object Notation) permette l'interscambio di dati occupando molto meno spazio dell'XML. Ogni proprietà è rappresentata come nome:"valore". Ha i tipi base number, string, boolean, array, object e null.
Quando un client riceve dei dati JSON li deve interpretare, e può farlo in due modi:

- eval(): /* DEPRECATA */ che esegue ciò che gli viene passato come parametro. Deprecata perché espone a JS injection

- JSON.parse(): converte una stringa JSON in oggetto

Per trasformare in automatico un'array in stringa si può usare JSON.stringify(). Se il parsing con JSON.parse fallisce, lo script termina.

8. REST

Acronimo di Representational State Transfer, REST è un'architettura software per sistemi distribuiti che vede tutto il web come un'unico documento. È stato presentato nella tesi di dottorato di Roy Fielding, che lavorò anche alle specifiche di HTTP. È una tecnica utile per il mashup di contenuti provenienti da siti diversi su di una stessa pagina (come possono essere i plugin per social network). I servizi REST vengono invocati attraverso richieste HTTP, e i risultati sono generalmente restituiti in XML o JSON

Due principali teniche per sviluppare web services: **SOA** (Service Oriented Architecture) e **WOA** (Web Oriented Architecture). Il SOA comunica via XML, il WOA via JSON. Il SOA è usato principalmente dalle aziende per far comunicare sistemi legacy, il WOA è il più usato dai servizi web per l'utente.

I servizi Web che mettono a disposizione API pubbliche forniscono chiavi agli sviluppatori, per identificarli e soprattutto ler evitare attacchi DoS: avendo ogni sviluppatore identificato da una chiave posso impostare un limite di richieste e/o un intervallo di tempo minimo tra una richiesta e l'altra per uno stesso client: in questo modo le mie API non possono essere bombardate di richieste.

Il mashup di contenuti provenienti da servizi diversi richiede una comunicazione server-to-server, che in PHP può essere fatta con le librerie cURL.

9. Vulnerabilità

La sicurezza di un'applicazione Web è su due livelli:

- infrastruttura (server Web, SO del server, DBMS, versione dei linguaggi utilizzati ecc.)
- codice (bug nel codice, colpa del programmatore)

Molte applicazioni per server arrivano con password di default (MySQL ecc.): vanno cambiate immediatamente perché saranno la prima cosa che un attaccante proverà.
Una volta che l'attaccante ha qualche informazione sul nostro server (principalmente web server usato e DBMS usato) potrà tentare un attacco mirato. Il tipo di attacco più diffuso è l'**injection**: tentare di inviare codice malevolo al server, con vari trucchi (ad esempio scrivere delle query SQL in un form di login sperando che lo sviluppatore non faccia l'escape delle stringhe ricevute).
Gli attacchi alle applicazioni web sono particolarmente diffusi perché particolarmente facili: le applicazioni web sono ovunque e le tecniche sono relativamente semplici, ulteriormente semplificate dal fatto che spesso gli sviluppatori Web hanno poca esperienza; inoltre è facile

anonimizzarsi dietro proxy o VPN. Anche il protocollo HTTP semplifica il lavoro agli attaccanti: non offre tecniche per riconoscere gli utenti unici. Anche il fatto che persone diverse modifichino una stessa applicazione è pericoloso: modificando il codice di un'applicazione senza conoscerla a fondo potrebbe introdurre dei bug. Infine, il guadagno è una forte spinta a tentare di attaccare un'applicazione Web, soprattutto se ci girano sopra soldi.

Gli attacchi possono essere di vario tipo: può essere un attacco al database per cercare di rubare informazioni, uno sniffing per intercettare la comunicazione o un DoS per buttare giù il server (tante richieste contemporanee fino a saturare la banda). Le tecniche possono sfruttare la manipolazione delle richieste e risposte HTTP/HTTPS attraverso il browser, estensioni del browser o tool a linea di comando. Ad esempio, modificando lo user agent posso capire se il server invia codice personalizzato a seconda del client.

9.1 Preparazione all'attacco: profiling

Prima di sferrare un attacco è necessario un profiling del sito, per cercare di ottenere più informazioni possibili su infrastruttura e codice: cercare di capire quali software e librerie vengono utilizzate sul server, e soprattutto quali versioni dei software usano, in modo da sapere se si possono sfruttare vulnerabilità note. Le librerie, i CMS e tutto quello che produce pagine dinamiche, infatti, lasciano tracce nella struttura delle pagine che creano (ad esempio nei nomi utilizzati, nella struttura della pagina ecc.).
Attraverso il banner grabbing si può chiedere tipo e versione del web server: ma il server potrebbe bloccare queste richieste o fornire informazioni errate. A questo punto entra in gioco l'HTTP fingerprinting: anche se il server non vuole dirmi chi è, posso scoprirlo da come sono organizzati gli header.
I grossi siti fanno uso di load balancer: hanno più server per uno stesso indirizzo, per fare in modo di non venire inondati di richieste (impossibile pensare che sia un solo server a rispondere a tutte le richieste per Google). In questi casi, è possibile che uno dei server non sia ancora aggiornato con le ultime patch o che abbia ancora directory, file o configurazioni provvisorie per lo sviluppo/debug: tutte cose che rendono più facile l'attacco.
Il profiling continua analizzando minuziosamente struttura e funzionalità di un'applicazione, oltre che all'HTML (che potrebbe contenere commenti dimenticati dallo sviluppatore). Bisogna anche cercare il nome dello sviluppatore: potrebbe aver avuto problemi nello sviluppo e aver chiesto aiuto sui forum: saprò che problemi ha avuto e quindi quali attacchi saranno più efficaci.
Si può anche controllare se nella document root esistono directory non linkate, che magari contengono tool di amministrazione o file da includere nel codice: esistono tool che provano tutti i nomi più comuni.
Altri controlli vanno fatti sulle estensioni dei file, che danno informazioni sui linguaggi utilizzati, e file predefiniti delle applicazioni come README, COPYING, EULA ecc. che possono darci informazioni sulle applicazioni utlizzate sul server.
I nomi dei campi dei form, invece, ci danno informazioni su nomi di attributi e tabelle del database: spesso e volentieri infatti lo sviluppatore usa gli stessi nomi per entrambi, per comodità.
I form espongono ad attacchi di tipo file upload, in cui il client ci invia un file maggiore della dimensione massima consentita: succede quando facciamo i controlli solo lato client.

9.2 Tecniche di attacco

Session hijacking

Il session hijacking invece consiste nello scoprire un'identificatore di sessione valido sniffando il traffico (attacco man in the middle), attaccando il client via JavaScript, XSS, malware vario ecc. oppure indovinarlo se il session ID è predicibile (bug di PHP)

SQL injection
Uno degli attacchi più noti ed efficaci: invio query SQL nei form per cercare di farmi autenticare e/o di scaricare informazioni. È pericolosissima la stringa -- perché è il commento: ignora tutto quello che c'è scritto dopo: se nello username scrivo ad esempio Rossi' OR 1=1-- al database arriverà una query del tipo: SELECT * FROM Users WHERE username='Rossi' OR 1=1. La condizione sarà quindi sempre vera.

Cross-site scripting XSS
Consiste nell'inviare su un forum che non fa controlli sull'input del codice Javascript malevolo: quando qualcuno visualizza il mio messaggio il codice verrà eseguito. Se i controlli sono fatti solo lato client, all'utente potrebbe bastare disabilitare JavaScript per riuscire ad inviare il codice. Un esempio tipico di codice che vorremmo far eseguire è qualcosa che ci invia tutti i cookie del client, in modo da poter usare dei session ID validi.

Denial of Service
Di base, consiste nel consumare le risorse di un'applicazione al punto di non farla più rispondere ai client legittimi. Proteggersi è complesso perché bisogna trovare un compromesso tra sicurezza ed usabilità: principalmente bisogna cercare di poter sempre identificare gli utenti in modo da limitare le risorse assegnate ad ognuno.

9.3 Difesa
La difesa dagli attacchi va fatta seguendo i principi di Denning. Serve ridondanza: è buona norma chiedere nuovamente la password ad un utente già autenticato quando si svolgono operazioni critiche. Anche il principio del minimo privilegio torna utile: mai fornire privilegi non necessari. Senza dimenticare che l'anello debole è quasi sempre l'uomo: sarà più efficace tentare un attacco di ingegneria sociale, tentando di convincere, con vari trucchetti, di farci dare le autorizzazioni che vogliamo.

10. Usabilità

"Un oggetto è usabile quando non servono etichette (o libretti di istruzioni) per usarlo"
. www.nngroup.com : sito del Nielsen-Norman Group, guru dell'usabilità.
La usability engineering definisce i criteri che devono essere adottati per valutare l'usabilità di un prodotto: questo parte dlle fasi iniziali dello sviluppo del software.
Tre misure dell'usabilità:

- efficacia: numero di successi nell'uso rispetto al numero di tentativi
- efficienza: carico di lavoro richiesto per completare l'operazione
- soddisfazione: numero di volte in cui viene espressa una preferenza per il prodotto.

Un'interfaccia user friendly deve richiedere all'utente di riconoscere piuttosto che di ricordare: Una shell mi richiede di **ricordare** i comandi da utilizzare, un'interfaccia grafica invece mi chiede

di **ricordare** le icone e i pulsanti per eseguire determinate azioni. Questa è una delle linee guida principali: sempre mantenere ben visibili gli oggetti richiesti per le azioni più comuni.
Il cervello può memorizzare 7 (+/-2) informazioni nella memoria percettiva: quando la nostra interfaccia ha molti elementi bisogna fare chunking delle informazioni: raggrupparle.
È importante anche avere un corretto mapping dell'interfaccia: deve essere chiaro da subito cosa succede se clicco qualcosa. Un esempio di mapping errato sono i comandi dei fornelli: guardando le manopole non so quale apre quale fornello.
Bisogna anche stare attenti a non usare sinonimi (confondono l'utente inesperto) e a mantenere la consistenza. Bisogna anche usare metafore corrette: su alcuni vecchi Mac per espellere il floppy andava trascinata l'icona del floppy sul cestino: assolutamente antiintuitivo, a nessuno verrebbe in mente di farlo perché al cestino è associata l'eliminazione, non l'espulsione di un disco.

Vanno inoltre fornite informazioni persistenti: per notificare all'utente l'arrivo di un messaggio non basta una suoneria, perché l'utente potrebbe non sentirla: va anche cambiato l'aspetto dell'icona.