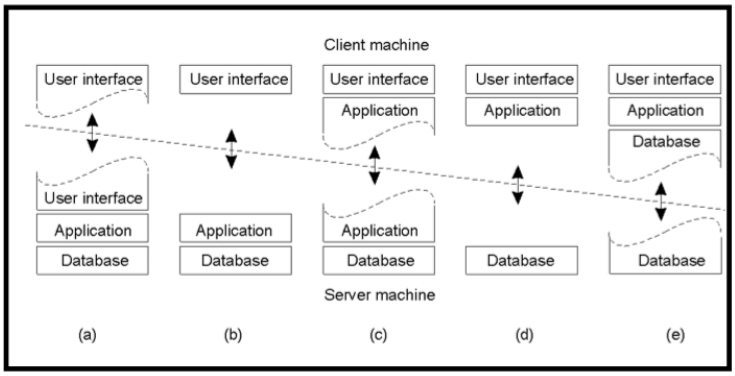
# Applicazioni web

Monday, 8 May 2023 08:38

- Multitier application
  - Abbiamo un middleware che ci fà credere che abbiamo 1 singolo ambiente
    Lo abbiamo visto in:
    - RMI
    - Come la rete funziona (user process-tcp-ip-hardware)
  - Noi qui abbiamo dei strati, ed ogni strato ha un compito preciso
    Un esempio è fatto da MVC:
    - Presentation
    - Logic (Business)
    - Data (access)

    Ed ognuno è connesso da interfacce astratte (api) ed appoggiato tutto da una infratuttura di computer distribuiti
    E siccome è basato sulle API:
    - L'utente fa una richiesta al server, il server fa la richiesta al database, e poi si ritorna
    - Un problema di questo è decidere quanto accesso il client ha del server
      E qui abbiamo 5 casi:

      Application = Prepara i dati [Servlet]
      User interface = Organizza i dati, qui il client decide come visualizzare i dati tipo, ed il server lo mostra
      Database = Acquisizione dei dati [DB]
      Quando sono condivisi, il server invia dati grezzi che il client elabora

- RIA (Rich ich application)

- RIA (Rich ich application)
    - Noi vogliamo poter una connessione delle informazioni, passare al web 3.0
    - Il web lo conoscono tutti, è facile da aprire da tutti, e quindi l'idea è di far si che tutti possono accedere a qualunque applicazione, e qui nascono le web applicazioni -> javascript
- JS
    - Linguaggio di scripting a oggetti non tipizzato (Yey!)
        - E' interpretato da un engie
        - Permette creazione pagine dinamiche
        - Permette effetualizzazione di richieste HTTP
    - NOTA da ora in poi nei miei appunti mancheranno tanti pezzetti, e la motivazione è che io tante cose già le so. Se notate un pò di fragmentazzione, questa è la motivazione. Detto questo, vedrete ciò che io non sapevo principalmente
    - Tipo variabili:
        - Var -> Usarlo solo per variabili globali, e fa riferimento a varibili globali
          Es:
          ```
          var x = 10;
                {
                   var x = 4;
                   console.log(x) -> 4
                }
                console.log(x) -> 4
          ======
          var x = 10;
                {
                   let x = 4;
                   console.log(x)
                }
                console.log(x)
          ```
        - Let -> Definire variabili sono nel blocco dichiarato
        - Const -> Una let che non permette di cambiare il contentiore
          Non si possono modificare variabili semplici
          Aka, const x = 2; x=4 -> errore
          Però con gli array, aka varibili composte, possono cambiare
          ```
          const ages = [15,2]
          ages[0] = 0
          ages[10] = 10
          for (const agesKey in ages) {
          console.log(ages[agesKey]) -> 0, 2, 10
          ```

}

Nota: con **in** ottiamo gli index, se invece lo avremmo sostituito con **of** avremmo ottenuto gli elementi

- Array hanno la funzione "shift", che toglie il primo elemento, ed unshift, che aggiunge un qualcosa come primo elemento
- Map, è una funzione degli array, praticamente noi applichiamo una funzione ad ogni elemento dell'array, sostituendolo.

```
const numbers1 = [45, 4, 9, 16, 25];
const numbers2 = numbers1.map(myFunction);

function myFunction(value, index, array) { // multiplies each array value by 2
  return value * 2;
} // resulting array: [90, 8, 18, 32, 50]
```

- Con Filter invece possiamo togliere delle persone attraverso un true/false

```
const numbers = [45, 4, 9, 16, 25];
const over18 = numbers.filter(myFunction);

function myFunction(value, index, array) {
  // creates a new array from elements with a value larger than 18
  return value > 18;
} // resulting array: [45, 25]
```

Note: The example us
the value parameter.
The other two can be

- Con reduce rendiamo 1 array in 1 singola variabile

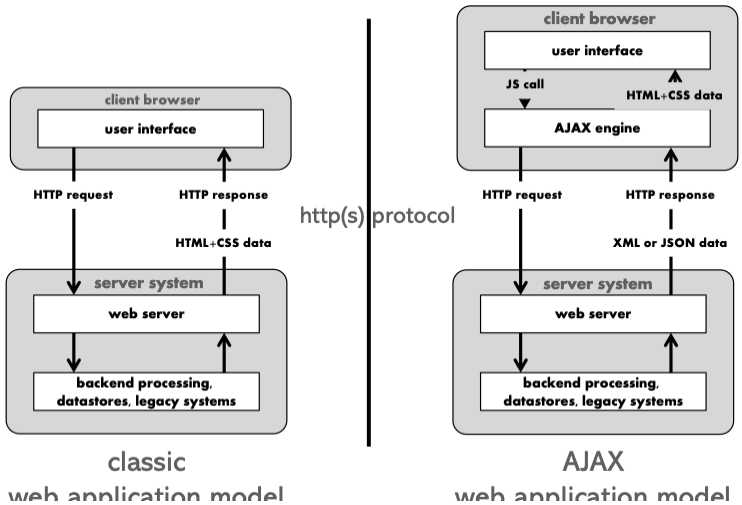
```
const numbers = [45, 4, 9, 16, 25];
let sum = numbers.reduce(myFunction);

function myFunction(total, value, index, array) {  // sums all numbers in an array
  return total + value;
} // resulting value: 99
```

- Tipi funzioni:
  - Function f(x, y) {return x*y}
  - Var f = function(x, y) {return x*y}
  - Const f = (x, y) => x*y
  - Const f = (x, y) => { return x*y }

  Gli ultimi 3 possono essere passati come parametri a funzioni

- Ajax
  - In uno schema tradizionale, ci sono multiple richieste-risposte tra client e server, e qui viene usato AJAX



- E' asincrono, e quindi possiamo modificare la pagina anche se non abbiamo ancora ricevuto la risposta del server
  Però siccome è asincrono, si aggiungono dei problemi:
  - Problematiche di concorrenza
  - Difficile debugging
  - Lo stato diventa complicato
  - Sono operazioni pesanti
- La pagina può cambiare di sua iniziativa facendo delle richieste al server, senza reload della pagina
  - L'applicazione è puramente attiva -> Eventi
  - L'applicazione si limita ad iniziallizare l'applicazione, e poi determinati eventi vengono chiamati