Corso di Programmazione Web e Mobile
A.A. 2017-2018

Pomidoro Timer

◖ Franco Piras, 896773 ◗

1. Introduzione

Pomidoro Timer è un app di time management creata sulla falsa riga dell'originale pomodoro technique inventata nei primi anni 80 da Francesco Cirillo.

E' basata sulla suddivisione del tempo dedicato ad un lavoro in 4 sessioni da 25 min. di lavoro intervallate da pause brevi e seguite da una pausa lunga dopo la quarta sessione.

E' un metodo utilissimo per chi ha problemi di gestione del tempo ed è focalizzata al mantenimento dell'attenzione.

Oltre ad essere un time manager in senso stretto, lavora anche tracciatore del tempo dedicato ad ogni singola attività, consentendo la programmazione del tempo delle singole sessioni di lavoro e pausa personalizzate per ognuna.

L'app è concepita per lavorare totalmente senza connessione dopo il primo download della pagina ed è web based, così da poter essere sempre raggiungibile usando un comunissimo browser.

I contenuti elaborati nel corso dell'utilizzo sono salvati nella memoria locale del browser e consentono una persistenza degli stessi anche a seguito di un refresh della pagina.

I medesimi contenuti possono essere salvati sul server e recuperati da un altro browser, a patto di conoscere il nome della sessione salvata.

1.1. Breve analisi dei requisiti

1.1.1. Destinatari

Capacità e possibilità tecniche.

L'utente tipo per quest'app può essere chiunque abbia necessità di uno strumento utile per gestire il proprio tempo lavorando su più progetti contemporaneamente, come lavoratori in ambito IT o studenti.

Autore: Franco Piras

Parliamo quindi di soggetti che prediligono una gestione organizzata del loro tempo, ma necessitano di strumenti esterni per farlo con profitto.

Questi fanno a priori stime di lavoro su determinati task, ma che non hanno modo di poter gestire sempre il loro tempo in modo opportuno a causa di influenze esterne o interne (la distrazione è sempre in agguato!).

L'utente conosce quasi certamente questa tipologia di tools e arriverà all'applicazione dopo una ricerca mirata ed un'accurata selezione delle caratteristiche proposte da questa versione custom della classica app Pomodoro base.

Linguaggio.

Il linguaggio dell'app, fatto salvo per la guida iniziale è prettamente legato ad immagini (timer) ed elenchi (elenco dei task).

L'impatto visivo è immediato e consente di focalizzare la propria attenzione al progetto/i definiti.

Il cuore di tutto è il timer centrale che occupa buona parte della parte alta della home page.

Questo aspetto è stato scelto con attenzione, poichè l'utente dovrà focalizzarsi solo su questo, dopo aver definito i suoi obiettivi.

Essendo il suo aspetto volutamente minimale, ogni button implementa un popup di aiuto che ne descrive la funzione, senza essere invasivo, ed ogni casella di input suggerisce cosa inserire con medesime modalità.

I contenuti dell'app sono in inglese, lingua oramai nota ai più e, certamente nota alla tipologia di utenza che ne farà uso, per poterle dare accesso ad un'utenza internazionale.

La terminologia usata è semplice e non dà luogo ad incomprensioni.

Motivazione.

Tipo di motivazione: L'app, a dispetto dei classici pomodoro timer, è task based.

Per utilizzarla è quindi necessario creare almeno una categoria con uno o più task all'interno.

La categoria può essere vista come un mero contenitore per organizzare i vari task per tipologia o come la radice di una lista di task facenti capo allo stesso progetto.

Ogni singolo task può essere personalizzato nei tempi di lavoro e pausa e può essere marcato come completato prima della fine di tutto il ciclo di un pomodoro time.

I task possono essere interrotti in favore di altri e ripresi successivamente.

Queste caratteristiche ne fanno uno strumento perfertto per i project manager o per chi segue più progetti complessi contemporaneamente, oltre che per il classico utente in cerca di un timer.

Livello di consapevolezza: L'app non ha di per se contenuti iniziali, salvo una descrizione delle motivazioni che mi hanno portato a crearla posta in un link dedicato ed una mini guida iniziale che appare al centro della main page al primo utilizzo, in assenza di dati.

1.1.2. Modello di valore

Il valore dell'applicazione risiede nella sua visione task based.

Rispetto ad altre app simili, infatti, consente di lavorare su più task contemporaneamente.

L'acquisizione dei dati sull'avvio dei task nel tempo, la loro categorizzazione in sottogruppi ed i tempi stimati dagli utenti per ogni task, sono dati sempre appetibili per le aziende che acquistano da siti terzi dati staticici.

Nelle successive versioni, inoltre, verrà creata una modalità con login, advertise free e con possibilità di salvare le proprie sessioni in una base dati dedicata ed accessibile solo all'utente loggato e non a chiunque conosca il nome della sessione.

1.1.3. Flusso dei dati

L'applicazione è inizialmente priva di contenuti.

Salvo l'utente non sia solito usarla e ne abbia salvato copia sul server.

Ottenere i contenuti.

I contenuti dell'app sono personali e possono essere reperiti dall'utente con un solo semplicissimo passaggio dall'app stessa.

Se nel frattempo l'utente ha già inserito altri dati in locale, questi verranno integrati con i dati provenienti dal server.

Archiviare e organizzare i contenuti

L'app ha due livelli di archiviazione entrambi basati su json per futura integrazione con db no sql come MongoDB:

- Locale, basato su localstorage del browser e volatile (a seconda delle impostazioni del browser stesso).
  Su localstorage è possibile salvare unicamente stringhe di testo e quindi, in teoria, dati dalla struttura poco complessa.

  Per bypassare questa limitazione, l'app organizza tali dati secondo un modello json convertito in stringa prima del salvataggio.

  Sono stati previste due basi dati:

  - Time: tiene traccia dei valori di default dell'app e dello stato del task attivo.
  - Panel: memorizza l'alberatura delle categorie/task e tutti i loro dettagli.

  Tutti i dati sono aggiornati runtime ad ogni modifica dello stato corrente e salvati su localstorage.
- Su server. Il salvataggio è manuale, a discrezione dell'utente.
  E' estremamente facile da attuare e si basa unicamente sulla scelta di un nome per la sessione da salvare.

  Se questa sessione è presente sul server, verrà aggiornata.

  In caso contrario verrà creata.

  Il reperimento di tali sessioni è altrettanto facile e necessita solo della conoscenza del nome di sessione da parte dell'utente.

  Il salvataggio è reso possibile da una chiamata API a cui viene passata una stringa di testo/ascii in formato "application/x-www-form-urlencoded" in cui verrà concatenato il nome dlela sessione al contenuto del record panel (sempre in formato stringa) prelevato dal localstorage.

  Il recupero dei dati dal server in locale, opera l'azione opposta, sovrascrivendo il record panel sul localstorage dopo opportuni controlli di merge dei record.

1.1.4. Aspetti tecnologici

L'app è ospitata su un server aruba "raw" in affitto al più basso prezzo sul mercato italiano (1€). L'ip pubblico utilizzato per l'app è stato fornito di risoluzione DNS utilizzando il servizio gratuito no-ip.org che permette, in modalità gratuita, il mapping di 3 host diversi. Il server è costruito pacchetto per pacchetto, firewall compreso, è predisposto per l'uso di node.js e di database mongodb e mariaDb.

Trattandosi di un app concepita per lavorare anche offline, la maggior parte delle tecnologie utilizzate fa capo a questa parte.
Nello specifico, il frontend usa un mix di javascript vanilla per la maggior parte delle operazioni più pesanti, ma semplici nel loro complesso, con qualche richiamo di funzioni jquery per le funzionalità più complesse o comunque in cui jquery non introduceva latenza.
Jquery è una delle librerie maggiormente utilizzate indirettamente o direttamente nel web. Conta oltre un milione e mezzo di download settimanali da npm. Ad esempio nel framework UI Bootstrap viene utilizzata per fornire animazioni e aumentare le funzionalità dei componenti.
Ha avuto dalla fondazione l'obiettivo di semplificare attraverso la realizzazione di metodi ad hoc la gestione di vari eventi complessi.
Questo, se da una parte porta ad una notevole semplificazione del codice, in alcuni casi ne appesantisce l'esecuzione rispetto a javascript vanilla.
E' quindi un dato da tenere in estrema considerazione nella scelta d'uso o meno, se i propri obbiettivi prevedono un sito snello.
Nel nostro caso, per esempio, tutto il mapping di id e classi richiamate poi dal codice javascript per modifiche all'alberatura html esistente o per crearne di nuova, non usano chiamate funzionalità jquery, ma funzioni javascript vanilla (queryselector).
Analogamente, la function che si occupa della gestione delle chiamate ajax da e verso il server, è stata interamente scritta in javascript vanilla, anche per poterne controllare alcune peculiarità.
Questa, infatti, con una semplice chiamata a funzione, consente l'inclusione di altre pagine html e/o oggetti (immagini, suoni, pdf) in precisi punti dell'albetura html con una semplice chiamata a funzione.

Come framework a supporto di HTML5 vanilla, è stato utilizzato Bootstrap, framework sviluppato da Twitter ed oramai standard internazionale de facto per i framework frontend. Oltre che per la sua fama e facilità d'uso, bootstrap fornisce notevoli agevolazioni nella gestione di pagine responsitive per cui è quindi previsto un uso su display con formati diversi.

Lato backend, l'app si avvale di un server node.js costruito secondo il paradigma mvc:

● Node.js: E' una piattaforma server-side che grazie al motore V8, sviluppato da Google ma vi sono anche altri engine per altri browser, permette la realizzazione di applicazioni Javascript.
Nodejs è un'architettura basata sugli event-driven capace di gestire applicazioni asincrone. Queste qualità ne hanno decretato il successo, poiché all'aumentare delle richieste, permette scalabilità e un maggiore throughput di dati, oltre a qualità come la leggerezza e velocità.
Nodejs opera in single thread, utilizzando la tecnologia non-blocking I/O permettendo lo sviluppo in parallelo senza incorrere nel costo di thread context-switching.
Questo porta un "occhio non attento", a pensare che venga utilizzata la multi

programmazione. Le sue dipendenze vengono sviluppate su npm, il più grande ecosistema di librerie in questo ambito, dove è possibile reperire librerie che svolgono specifiche funzionalità già realizzate da altre persone velocizzando così la realizzazione di applicazioni.

Express.js: E' un framework web per Node.js rilasciato con licenza MIT ed ha lo scopo di realizzare applicazioni web e API.
Negli anni ha assunto una maggiore importanza diventando grazie alla sua semplicità lo standard de facto in ambiente Node.js.

Nel nostro caso, il server espone pubblicamente tutte le cartelle contenti le pagine html, i file css e javascript, gestendo con specifiche chiamate get e post l'accesso alla cartella che contiene le sessioni salvate.

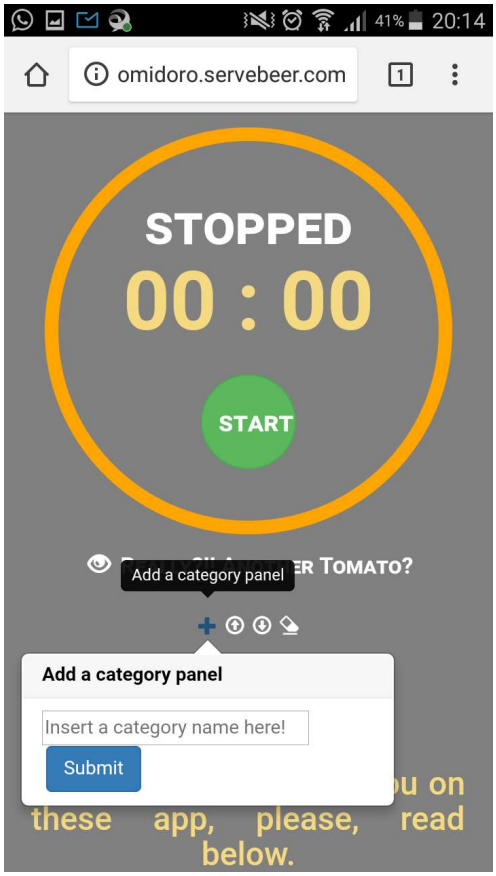
Scambio dati: JSON (Javascript Object Notation).
È un formato largamente utilizzato nelle applicazioni tra client-server per comunicare informazioni.
Esso è indipendente dal linguaggio utilizzato, si basa su una logica di associazione chiave-valore, rendendolo il suo utilizzo semplice e immediatamente capibile.
I tipi di dato supportati sono booleani, interi, array, stringhe, null e altri oggetti.
In pomidoro gli oggetti json vengono trasformati in stringa in fase di trasporto/salvataggio.

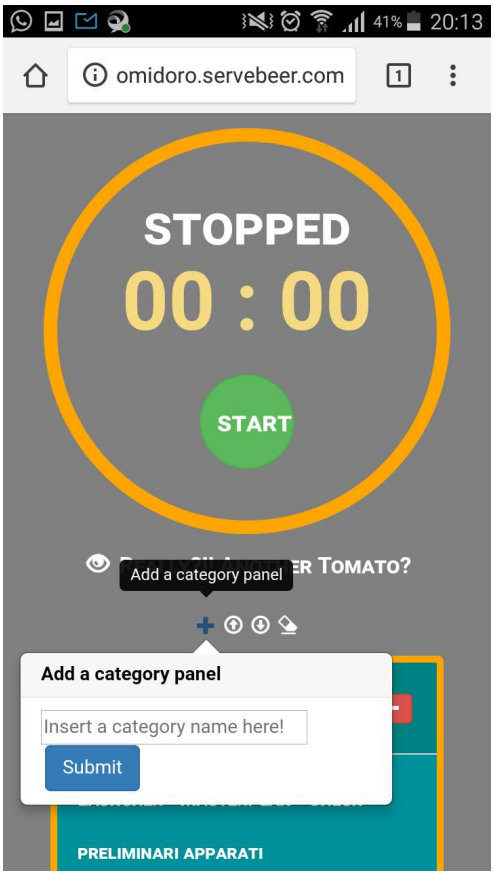
Architettura: REST (Restful State Transfer).
Uno dei tipi di architettura maggiormente diffuse e famose nel web basate su HTTP con lo scopo di ottenere miglioramenti lato performance e scalabilità.
Prevede dei vincoli, lasciando libertà nella loro implementazione.
Primo, client-server: viene applicata una separazione tra i due livelli che hanno responsabilità e compiti diversi.
La seconda è stateless dove ogni richiesta del cliente contiene tutte le informazioni necessarie per la richiesta e la sessione.

IDE: Per lo sviluppo è stato usato l'IDE Webstorm, della Jetbrain.
Oltre alle classiche comodità dei comuni IDE di sviluppo, in questo caso è stato collegato realtime al server ed un repository git.
Grazie a ciò, ogni modifica fatta al codice su ambiente molto più comodo del classico terminale era immediatamente disponibile sul server, mentre git si occupava del versioning del codice.

2. Interfacce

L'interfaccia è molto semplice e totalmente grafica, fatto salvo per una descrizione iniziale e dei tooltip per ogni tasto, come da esempio:





Con l'apposito tasto funzione si può creare una o più categorie, che daranno vita ad un pannello richiudibile per ogni categoria.

Cliccando sul nome di ogni categoria il corpo del pannello, in cui sono contenuti i task, viene nascosto.

E' stato inoltre implementato il cosidetto "accordion behavior". In pratica un solo pannello alla voltsa resta visibile, così da rendere l'app più leggibile.

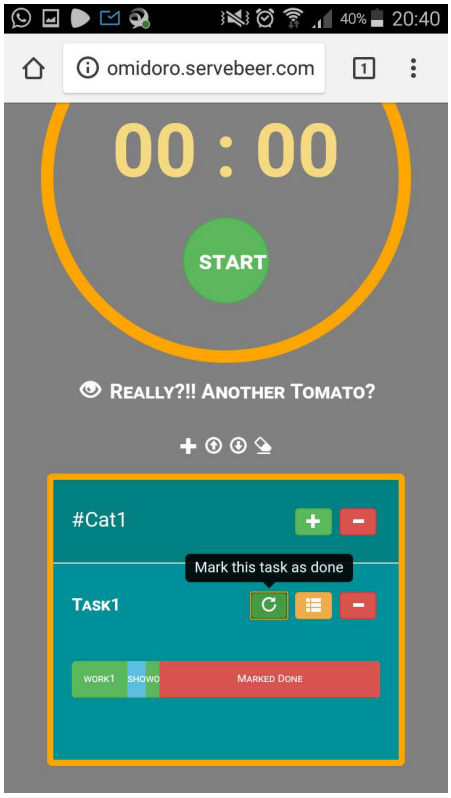
Cliccando su un pannello per espanderlo quello aperto viene automaticamente nascosto.

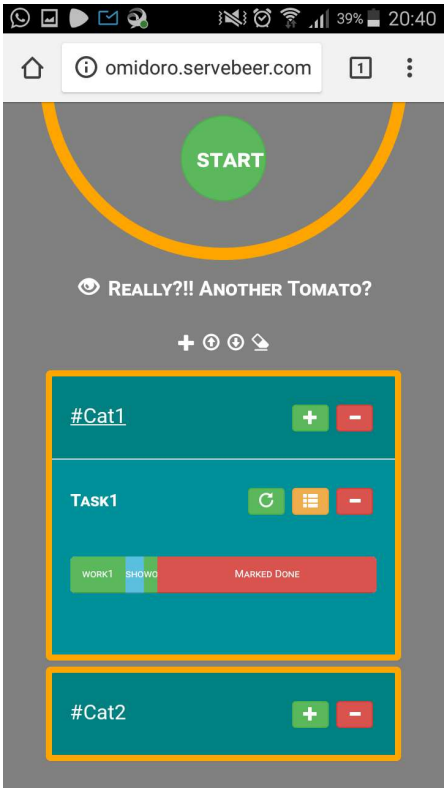
Ogni categoria mette a disposizione un tasto per la creazione dei task ed un tasto per la completa rimozione della categoria.

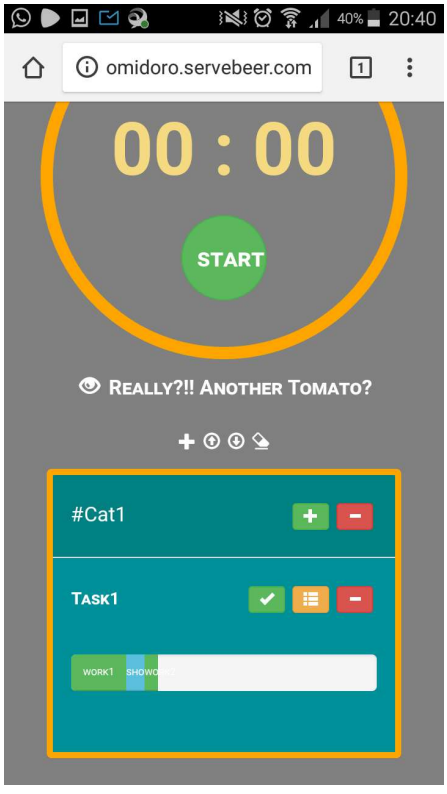
A sua volta, creando un task, questo mette a disposizione un tasto per la sua rimozione un tasto per personalizzarne i tempi ed un tasto per completare/ripristinare un task.

Cliccando invece sul nome del task, il suo timer inizia/riprende la sessione, interrompendo quella di un eventuale task in corso, se già presente.

Il task in corso viene visualizzato in un'apposita riga:





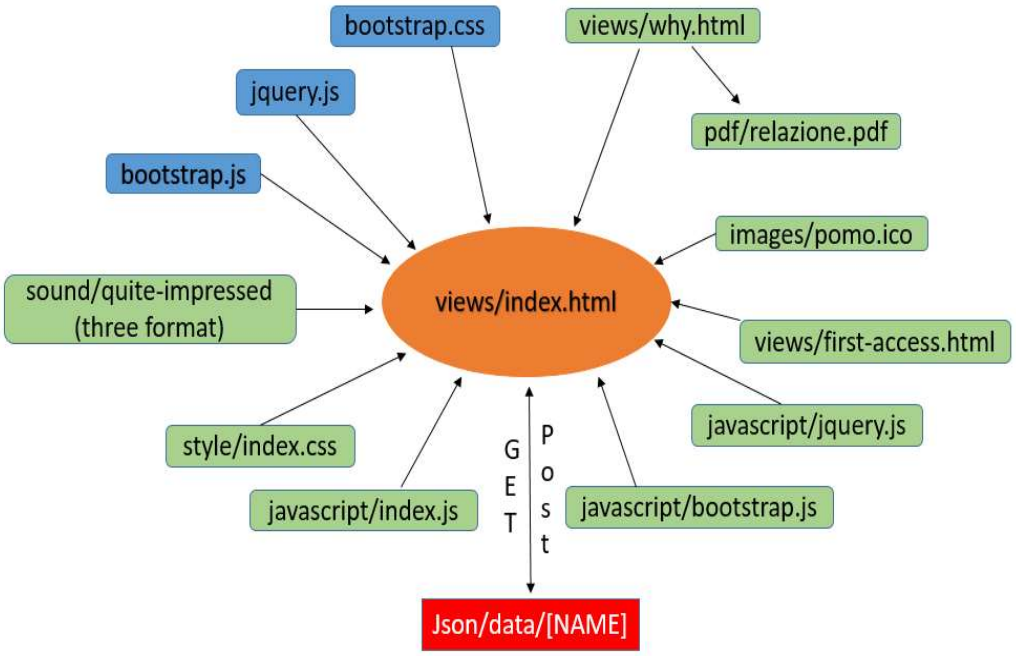


3. Architettura

3.1. Diagramma dell'ordine gerarchico delle risorse

Tutte le pagine dell'app sono inglobate nella pagina principale al primo download.

Per ovviare a problemi di caricamento di bootstrap e jquery dai repository ufficiali, una copia probabilmente meno aggiornata è stata caricata sul server e viene scaricata in parallelo per garantirne sempre la presenza:



L'accesso alle pagine secondarie viene garantito mediante un'apposita funzione che prende in pasto gli id dei DIV prescelti, nascondendo il secondo e rendendo visibile il primo.

Nel caso delle pagine secondarie, queste vengono caricate su opportuni DIV in modalità hidden da un'altra funzione, che si occupa anche di creare la struttura dei DIV, associandola ad un DIV bersaglio di default già presente.

3.2. Descrizione delle risorse

- Bootstrap.min.js: libreria bootstrap, in locale e da repository remoto.
- Jquery.min.js: libreria jquery, in locale e da repository remoto.
- Index.js: libreria principale di gestione dell'app
- Why.html: pagina in cui spiego il perchè dell'app e che porta al link della relazione.
- First-access.html: la pagina è visibile solo in assenza di categorie già create.

3.3. Altri diagrammi

Schema Json dei panel:

```
{"cat1":
      {"name":"cat1",
       "task1":{"name":"task1",
            "time":{"def_work":25,
                          "def_short":5,
                          "def_long":15,
                          "curr_phase":"",
                          "curr_min":"",
                          "curr_sec":"",
                          "count_work":"",
                          "count_short":"",
                          "count_long":"","done":""
          },
          "progress":{"work1":0,
                        "short1":0,
                        "work2":0,
                        "short2":0,
                        "work3":0,
                        "short3":0,
                        "work4":0,
                        "long1":0
        }
      }
   }
}
```

Schema Json del time current:


```
{"clock":
        {"work":25,
         "short":5,
         "long":15,
         "sec":0,
         "display_min":"",
         "display_sec":""
        },
        "count":{"work":0,
        "short":0,
         "long":0
        },
        "state_of_art":{"session":"stopped",
                        "clockrun":"",
                        "progress":"false",
                        "current_title":"none",
                        "current_task":"none",
                        "current_cat":"none"
        }
  }
```

4. Codice

4.1. HTML

Tasti popover che governano l'app.

```
5.          <div class="row functions">
              <div class="col-sm-4"></div>
              <div class="col-sm-4 flexbox">
                <span data-toggle="tooltip" title="Add a category panel" data-placement="top">
                  <a href="#" data-toggle="popover" id="cat" title="Add a category panel" data-placement="bottom">
                              <span class="glyphicon glyphicon-plus"></span>
                          </a>
                </span>
                 <div id="popover-content-cat" class="hide">
                  <input type="text" id="cat_name" placeholder="Insert a category name here!">
                  <button type="submit" class="btn btn-primary" onclick="addCat('none')">Submit</button>
                     </div>
                     <span data-toggle="tooltip" title="Upload session to server" data-placement="top">
                   <a href="#" data-toggle="popover" id="upload" title="Choose a name for this session" data-place-
            ment="bottom">
                              <span class="glyphicon glyphicon-upload"></span>
                     </a>
                    </span>
                    <div id="popover-content-upload" class="hide">
                      <input type="text" id="upload_name" placeholder="Insert a file name here">
                      <button type="submit" class="btn btn-primary" onclick="ajaxRequest('api/sessions/', 'up-
            load_name', 'post')">Submit</button>
                    </div>
                    <span data-toggle="tooltip" title="Download a saved session from server" data-placement="top">
                      <a href="#" data-toggle="popover" id="download" title="Choose a name file to retrieve" data-
            placement="bottom">
                        <span class="glyphicon glyphicon-download"></span>
                      </a>
                    </span>
                    <div id="popover-content-download" class="hide">
                      <input type="text" id="download_name" placeholder="Insert a file name here">
                      <button type="submit" class="btn btn-primary" onclick="ajaxRequest('api/sessions/', 'down-
            load_name', 'get', 'jsession')">Submit</button>
                    </div>
                <span>
                   <a href="#" data-toggle="tooltip" title="Erase all saved data" data-placement="top" on-
            click="erase()">
                    <span class="glyphicon glyphicon-erase"></span>
                  </a>
                </span>
              </div>
              <div class="col-sm-4"></div>
            </div>
```

5.1. CSS

Codice CSS dedicato ad adattare alcune impostazioni a seconda della larghezza del display di visualizzazione dell'app.

```
6.  @media(max-width: 260px){
      .container{
        margin-left:20px;
        margin-right:20px;
      }
      #timer_button{
        text-align: left;
      }
      #timer{
        margin-left:10px;
      }
      #first_access{
        width:280px;
        margin-left:20px;
        margin-right:20px;
      }
    }

    @media(max-width: 414px){
      .container{
        margin-left:20px;
        margin-right:20px;
      }
      #timer_button{
        text-align: left;
      }
      #timer{
        margin-left:10px;
      }
      #first_access{
        width:280px;
        margin-left:20px;
        margin-right:20px;
      }
    }
```

6.1. API

API lato server get (download) e post (salvataggio) delle sessioni locali.

7.
```
app.post('/api/sessions',function(req,res){
  var jsession_name = req.body.user;
  var jsession_data = req.body.data;

  if(!jsession_name || jsession_name === ""){
     console.log("not written file" + jsession_name + ".json");
     res.end("unsaved");
  }
  else{

      var contents = file.writeFile("data/json/" + jsession_name + ".json", jsession_data,
       function(err){
          if(err){
                console.log(err);
            res.end("error_occurred");
            next(err);
          }
          else {
                 console.log("written file" + jsession_name + ".json");
                  res.end("saved");
              }
       }
     );
  }
});

app.get('/api/sessions/:name', function (req, res, next) {

  var options = {root: __dirname + '/data/json/', dotfiles: 'deny', headers: {'x-timestamp': Date.now(),'x-sent':
true}};

  var fileName = req.params.name + '.json';

  res.sendFile(fileName, options, function (err){
     if(err){
        next(err);
     }
     else {
        console.log('Sent:', fileName);
     }
  });

});
```

7.1. Node.js

Server node.js

```
8.  var express = require('express');
    var app = express();
    var bodyParser = require('body-parser')
    var file = require("fs");

    var rootpath = {root: './public/views'};
    var path = require('path');

    app.use(express.static(path.join(__dirname, 'public')));

    // Tell express to use the body-parser middleware and to not parse extended bodies

    app.use(bodyParser.urlencoded({ extended: false }))

    app.use(bodyParser.json());


    // Route that receives a POST request to /sessions

    app.post('/api/sessions',function(req,res){
      var jsession_name = req.body.user;
      var jsession_data = req.body.data;

      if(!jsession_name || jsession_name === ""){
         console.log("not written file" + jsession_name + ".json");
         res.end("unsaved");
      }
      else{

          var contents = file.writeFile("data/json/" + jsession_name + ".json", jsession_data,
           function(err){
              if(err){
                    console.log(err);
                res.end("error_occurred");
                next(err);
              }
              else {
                     console.log("written file" + jsession_name + ".json");
                      res.end("saved");
                  }
           }
         );
      }
    });

    app.get('/api/sessions/:name', function (req, res, next) {

      var options = {root: __dirname + '/data/json/', dotfiles: 'deny', headers: {'x-timestamp': Date.now(),'x-sent':
    true}};

      var fileName = req.params.name + '.json';

      res.sendFile(fileName, options, function (err){
         if(err){
            next(err);
         }
         else {
            console.log('Sent:', fileName);
```

```
            }
        });

    });


    app.get('/', function(req, res){res.sendFile('index.html', rootpath);});

    app.listen(80, function(){console.log("Con Ok");});
```

Codice javascript lato frontend:

Codice di gestione del caricamento delle pagine secondarie e delle chiamate API.

```
function ajaxRequest(url,id_data_string, direction, get_type) {
   /* It's a customized ajax Request function with multiple purposes explained below */
  var xhttp;
  var final_url = url; // this is normally the url, but it could change with some specitif call.
   if (window.XMLHttpRequest) {
      // code for modern browsers
      xhttp = new XMLHttpRequest();
   }
   else {
      // code for IE6, IE5
      xhttp = new ActiveXObject("Microsoft.XMLHTTP");
   }
  xhttp.onreadystatechange = function() {
      /* this is a callback function that manage something from xhttp send action with a specific ready state */
     if (this.readyState == 4 && this.status == 200) {
        if(direction === "get" && !get_type) {
              /* default for get method without get_type spec. It put the response into a gived DIV tag */
                document.getElementById(id_data_string).innerHTML = this.responseText;
        }
           else if(direction === "get" && get_type === "normal_get") {
              /* get data are only passed to the call function without do anything else. */
             return this.responseText;
          }
          else if(direction === "get" && get_type === "jsession"){
             /*
                This get it's for data session stored on the server.
                The there is a call to another function that merge this data with current local storage, if present.
             */
             console.log("response: " + this.responseText);
             merge(this.responseText);
          }
       else if(direction === "post" && this.responseText === "saved"){
          /* when data are charged with a positive result on the server a popup show up */
          pophide();
          showPage("notify","up","Session saved on server!", false);
       }
          else if(direction === "post" && this.responseText !== "saved"){
             /* when data are not charged on the server a popup show up */
          showPage("notify","up","Error saving data on the server!", false);
          }
          else {
             /* if anything match a specific console log it's generated */
             console.log("ajax request without scope");
          }
     }
```

```
    };
    if(get_type === "jsession" && direction !== "post") {
        /*
            For the get call of a session stored, file name is from an input and not nested in the call.
            Url should be with sintax expected by the get route defined on the server.
         */
        var name_file = $("#" + id_data_string).val();
        final_url = url + name_file ;
    }

    console.log(final_url); // log of url called
    xhttp.open(direction,final_url, true); // open the request connection. True parameter is for make the call async

    if(direction === "post") {
        /*
            For the post call of a session stored, file name is from an input and not nested in the call.
            A Request header  and a data string to post should be set and included into send request.
         */
        var name_file = $("#" + id_data_string).val();
        xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
        string_to_post = "user=" + name_file + "&data=" + localStorage.panel;
        console.log(string_to_post);
        xhttp.send(string_to_post);
    }
    else{
        /* normal send call for get calls */
        xhttp.send();
    }
}
function addPageToDom(url, id, id_parent, media){
    /*

    */
    if(!media) {
        var spare_div1 = document.createElement('div');
        spare_div1.setAttribute('class', 'col-sm-2');
        var spare_div2 = document.createElement('div');
        spare_div2.setAttribute('class', 'col-sm-2');
        var row = document.createElement('div');
        row.setAttribute('class', 'row');
        var new_page_div = document.createElement('div');
        new_page_div.setAttribute('class', 'col-sm-8 flexbox');
        new_page_div.setAttribute('id', id);
        row.appendChild(spare_div1);
        row.appendChild(new_page_div);
        row.appendChild(spare_div2);
        $('#' + id_parent).append(row);
        ajaxRequest(url, id, "get");
    }
    else{
        var obj_tag = document.createElement('object');
        obj_tag.setAttribute('type', 'application/pdf');
        obj_tag.setAttribute('data', url);
        obj_tag.setAttribute('width', '100%');
        obj_tag.setAttribute('height', '800%');
        //$('#' + id_parent).append(obj_tag);
        $(body).append(obj_tag);
    }
}
function merge(data_get){
    /*
        Elaborate data session received by the get call to the jsession route on the server.
        If something is present on the local storage a merge is maked.
```

```
        If local storage is empty it do a simple charge.
    */
    console.log("received data: " + data_get);
    var panel_receiver = JSON.parse(data_get); // create a json object from the string received
    var merged = false; // default merge status. If there isn't a localstorage should be false.

    if("panel" in localStorage && localStorage.panel !== ""){ // Only if local panel storage is present
        console.log(localStorage.panel);
        var merged = true; // something is present.
        var panelJ = JSON.parse(localStorage.panel); // create a json object from the local storage string
        var cat,task; // instance of cat and task variables used below

        for(cat in panel_receiver){ // iterate cat list
              console.log("cat: " + cat);
              if (!panelJ[cat]) { // check if cat is present on local storage and merge if present
               console.log("aggiungo cat");
               panelJ[cat] = panel_receiver[cat];
               addCat("local", cat, "in"); // create cat panel
            }

           for(task in panel_receiver[cat]){ // iterate all tasks of category
                console.log("task: " + task);
              if (!panelJ[cat][task]) { // check if task is present on local storage and merge if present
                  console.log("aggiungo task");
                  panelJ[cat][task] = panel_receiver[cat][task];
                  addTask(cat, "local", task); // add the task under category panel.
              }
            }
        }
        localStorage.setItem("panel", JSON.stringify(panelJ)); //delete actual local storage and write new merged one.
    }
    else{ // Else localstorage not present is populated directly with data received
       localStorage.setItem("panel", JSON.stringify(panel_receiver));
        populateDomPanel(); // populate the dom with new sessions downloaded. similar to merge flow, but is all in one
function
    }

    if(merged){ // put a popup message with charge result.
   showPage("notify","up","Requested session merged correctly with local session!", false);
    }
    else{
        showPage("notify","up","Requested session correctly added!", false);
    }
}
function unhide(id,side){
    /*
        This site act as a single page app.
        Does it means that all is charged locally and server calls are minimal.
        To do it, all is charged with first load and the DIV hidden.
        This function permit hide/unhide management of a generic DIV
    */
    var page_to_unhide = document.querySelector("#" + id);

    switch(side){
        case "up":
          page_to_unhide.style.display = "inline";
          break;
        case "down":
            page_to_unhide.style.display = "none";
            break;
    }
}
function showPage(id_up,id_down,text,music){
```

```
    if(id_down !=="up" && id_down !=="down"){
        unhide(id_up,"up");
        unhide(id_down,"down");
    }
    else{
      var page_to_show = document.querySelector("#" + id_up);

      switch(id_down){
        case "up":
          var pan = document.querySelector("#notify_pan");
          var button = document.createElement('button');
              button.setAttribute('type', 'button');
              button.setAttribute('class', 'btn btn-xs btn-danger pull-right');
              button.setAttribute('onclick', 'showPage(\'notify\',\'down\')');
              button.innerText = "Close";
          pan.innerText = text;
          pan.appendChild(button);
          page_to_show.style.display = "inline";

          if(music){
              playSound();
              }
          break;
        case "down":
          page_to_show.style.display = "none";
          break;
      }
    }
}
```

9. Conclusioni

Sono abbastanza soddisfatto dal progetto svolto, essendo anche il mio primo progetto web.

Il progetto è stato volutamente sviluppato con molto codice javascript vanilla anche per evitare di utilizzare codice già pronto per le singole problematiche affrontate.

Spesso, avevo la soluzione a portata di mano ma ho comunque provato a crearne una tutta mia, nonostante la sicura perdita di tempo rispetto alla soluzione pronta.

In termini di progettazione di un sito questo si potrebbe considerare una pessima progettazione e sviluppo.

Nel mio caso, lo considero tempo spesso molto profiquamente poichè mi ha permesso di migliorare notevolmente nella scrittura del codice.

Essendo in generale un pessimo organizzatore nelle tempistiche, sto testando personalmente con profitto l'app, e ciò mi ha permesso in poco tempo di individuare punti morti o sottostimati nel mio lavoro.

10. Nota bibliografica e sitografica

https://www.w3schools.com

https://stackoverflow.com/