Università degli Studi di Perugia

Laurea triennale in Informatica

Progetto di Sistemi Aperti a Distributi

a.a 2017/2018

Luca Bartoli

Introduzione

Stressful è una piattaforma di assessment, cioè un sito Web il cui scopo è testare le competenze degli utenti riguardo vari campi del sapere, offrendo un'interfaccia e un insieme di funzionalità molto elementari.
L'applicazione implementa il paradigma CRUD, e utilizza il formato REST/JSON, classificandosi come RESTful Web Service (da cui, per un gioco di parole, il nome Stressful; tralasciando il fatto che il quiz è un agente ansiogeno per antonomasia).

Analisi dei Requisiti

In primo luogo va considerato il diverso tipo di utenti che andranno ad utilizzare tale servizio, gli erogatori, o amministratori, e i fruitori, o clienti o utenti ordinari. A questi corrisponde un diverso livello di privilegi:

- I clienti potranno oltre che reperire e svolgere test, anche modificare le proprie informazioni personali.
- Gli erogatori potranno accedere ai servizi di creazione, aggiornamento e cancellazione di test (vedremo poi, anche di categorie), oltre che poter usufruire dei servizi offerti anche agli utenti ordinati

Ovviamente vanno forniti opportuni meccanismi di autenticazione e di controllo.
Ciò premesso dopo un'attenta analisi delle richieste si ritiene che i seguenti punti siano i fondamentali.

Semplicità

L'applicazione dovrà presentare un buon livello di utlizzabilità: in quanto sito internet, è opportuno che rispetti la forma ben collaudata a cui tutti gli utenti sono abituati.

Ogni test dovrà essere accessibile, raggiungibile con pochi click, così le informazioni specifiche dell'utente, e per gli erogatori vanno forniti componenti per provvedere alla gestione della piattaforma, rapidi ed intuitivi.

Estensibilità

L'insieme di funzionalità iniziale dovrà costituire lo scheletro dell'applicazione, senza quindi eccedere con la sofisticazione, come descritto nel paragrafo precedente. Ma va posta particolare attenzione alla possibilità di aggiungere funzionalità all'occorrenza, in base alle esigenze degli utenti.

Sicurezza

Le informazioni immagazzinate dovranno essere protette da eventuali attaccanti o utenti maliziosi, e soprattutto si deve evitare che gli utenti ordinari riescano a compiere operazioni per cui non hanno il diritto.

Design delle Informazioni

Descriviamo le varie fasi di utilizzo:

- L'autenticazione si effettua con una pagina tradizionale di *login*, e se l'utente non è registrato lo può fare con un solo passaggio con la pagina di *signup*.
- Una volta acceduti la prima pagina è la *Home*, che mostra i test: l'organizzazione scelta per questi ultimi è quella delle *categorie*, che li raggruppano in base a nuclei concettuali. Si possono quindi navigare le categorie e ad un solo livello di profondità, accedere ai test.
- Una volta cliccato un test si può svolgere, e alla fine inviare i risultati o abbandonare.

- La seconda schermata della pagina è la *Carriera* dove sono mostrati tutti i test svolti e inviati, con i relativi risultati.
- Infine la pagina *Profilo* dove si può prendere visione delle informazioni personali, ed eventualmente modificare con un apposito form simile a quello si *signup*.

Per quanto riguarda gli amministratori:

- Le categorie possono essere create (basta specificarne il nome), modificate (cioè rinominate) e cancellate. La cancellazione di una categoria si propaga ai test contenuti. Le categorie possono anche essere vuote.
- Dentro ogni categoria si trovano dei test, i quali possono essere creati attraverso una schermata specifica, in cui vanno specificate regole sul punteggio, e l'elenco delle domande (con le relative risposte), modificati attraverso una schermata quasi identica ed eliminati.

Scelte Architetturali

L'applicazione è stata implementata in *PHP* lato server, con associato un database relazionale *MYSQL*, e lato client si è utilizzato, oltre che ovviamente *html* e *css*, una serie di script in *Javascript*, con l'ausilio del framework *JQuery*.
Il server può essere schematizzato come:

- *Web Server*, che implementa solo poca logica, e che serve per costruire l'*html*, cioè la presentazione del sito.
- *Data Server*, che contiene il database ed offre delle *API*, che sfruttano il formato REST/JSON.

Si noti che il server è in realtà uno solo, ma che internamente ha una suddivisione di questo tipo.

Perciò l'interazione tra client e server avviene nella seguente maniera: il client richiede una pagina html allo Web Server, fatto questo se ci sono delle informazioni da caricare, effettua una chiamata *POST* per invocare le *API* del Data Server, il quale le riceve e interpreta, per poi rispondere con una stringa in formato *JSON* con informazioni sullo stato ed eventuali messaggi d'errore. Il client, ricevute le informazioni le incorpora nella pagina, o in altri frangenti, richiede allo Web Server un'altra pagina.

Implementazione

Il codice è organizzato secondo una struttura ben consolidata e consigliata nei manuali:

- public_html
  - img
  - css
  - js
  - *home.php*
  - *…*
- resources
  - framework
  - library
    - *stressful_api.php*
    - *...*
  - templates
  - *config.php*
  - *config_local.php*

Tutta questa struttura si appoggia sui due file di configurazione *config* e *config_local*, i quali specificano informazioni globali e costanti che servono per far comunicare le varie parti del codice.
public_html contiene tutto ciò che concerne il lato client e la presentazione del sito, cioè fogli css, immagini, file javascript, e pagine *PHP*, che corrispondono alle schermate diverse che si vedono nel sito (*Home, Career, Profile, testadmin*…). Il codice PHP qua contenuto implementa solo pochissima logica, non si interfaccia al database e

riceve chiamate POST solo in alcuni casi particolari.
Questa scelta ci permetterebbe, con un opportuno file *.htaccess* di impedire l'accesso alle risorse e di esporre solo public_html, ad esempio.
Nella cartella risorse troviamo i frameworks, cioè *JQuery* e *Bootstrap*, rispettivamente per Javascript e CSS.
La cartella library, contiene il codice delle API, in particolare nel file *stressful_api.php*. Tali procedure interpretano le chiamate POST e si interfacciano al database, per poi codificare i valori di ritorno nel formato opportuno.
Tralasciando la struttura del database, di cui si parla nella presentazione, si descrive come avviene l'interfacciamento con il database.
Si è creata una famiglia di classi *singletons*, una per tabella, che offrono una serie di metodi per creare, reperire, aggiornare e cancellare le informazioni, secondo il paradigma alla base dell'immagazzinamento permanente dei dati CRUD.
In questo modo ad ogni operazione corrisponde un metodo che richiede una serie di parametri in cui si svolgono tutti gli opportuni controlli di consistenza e di sicurezza.
Eventuali anomalie vengono riportate attraverso eccezioni specifiche, definite apposta per la libreria, che possono essere stampate in JSON e inviate al client.
Le API quindi ricevono chiamate POST e si interfacciano al DB, sono cioè completamente indipendenti dalla presentazione, si potrebbe quindi programmare un client diverso il quale riuscirebbe comunque a dialogare con il server.
Come detto in precedenza, il client invoca direttamente le API, tramite un metodo apposito di JQuery, e riceve il JSON. In questo modo non si occupa inutilmente banda con il codice html, e non si costringe il browser a effettuare il parsing ad ogni chiamata POST, guadagnando in efficienza.
Perciò parte della logica è implementata proprio in Javascript/JQuery, e questo permette all'applicazione di eseguire alcune funzionalità in modo più rapido.
Infine la cartella templates contiene codice html riutilizzabile, il quale viene incluso da appositi metodi in un file *build.php*. L'idea di base è quella di semplificare il più possibile il Web Server, mantenendo i componenti separati in una locazione apposita, e quella di evitare di ripetere lunghi stralci di codice uguali per tutte le pagine, come i preamboli ed alcuni contenitori.

Conclusioni

L’insegnamento principale acquisito per quanto riguarda la programmazione è l’importanza della modularità, della corretta organizzazione della repository e della produzione razionale del codice. Per quanto riguarda la disciplina invece, una più profonda comprensione dei vari paradigmi su cui si basano i servizi di rete.