Stressful

Piattaforma di assessment

Luca Bartoli

Analisi dei Requisiti

Da un’attenta valutazione delle richieste si sono estrapolate tre caratteristiche essenziali:

1. **Semplicità:** va fornita un’interfaccia elementare che permetta di organizzare, creare, cercare ed effettuare test in maniera intuitiva ed efficace

2. **Estensibilità:** è opportuno progettare la piattaforma con features essenziali e funzionanti in prima battuta, per poi aggiungere funzionalità in base alla nuove esigenza

3. **Sicurezza:** è importante impedire che avvengano accessi o modifiche abusive ai dati riguardanti test e utenti

Analisi dei Requisiti

L’architettura della piattaforma dovrà prevedere due tipi di utenti:

- Fruitori (o clienti)
- Erogatori (o amministratori)

Dovrà quindi essere garantito:

- Un modo per registrarsi (signup)
- Un meccanismo di autenticazione (login)
- Un insieme di funzionalità più esteso agli amministratori
- Minori privilegi agli utenti ordinari

Cenni sul Design

- La prima cosa da vedere dovrà essere la *Home* da cui partire per trovare i test
- Questi ultimi dovranno essere suddivisi in categorie che possono essere navigate
- Ogni test potrà essere effettuato e sottomesso
- I risultati dei test saranno elencati nella voce *Carriera* del sito
- Le varie informazioni riguardo l'utente saranno visibili nella voce *Profilo* del sito

Scelte Architetturali

I linguaggi scelti:

- Il lato server della piattaforma è programmato in *PHP,* con database relazionale *MYSQL*.
- Il lato client, oltre che ovviamente incorporare *html* e *css*, sfrutta *Javascript* con i framework *JQuery*

Il server si suddivide in:

- *Web Server* che implementa solo poca logica, e per lo più costruisce lo scheletro delle varie pagine
- *Data Server* che contiene il database ed offre delle *API*, sfruttando il formato *REST/JSON*

CRUD

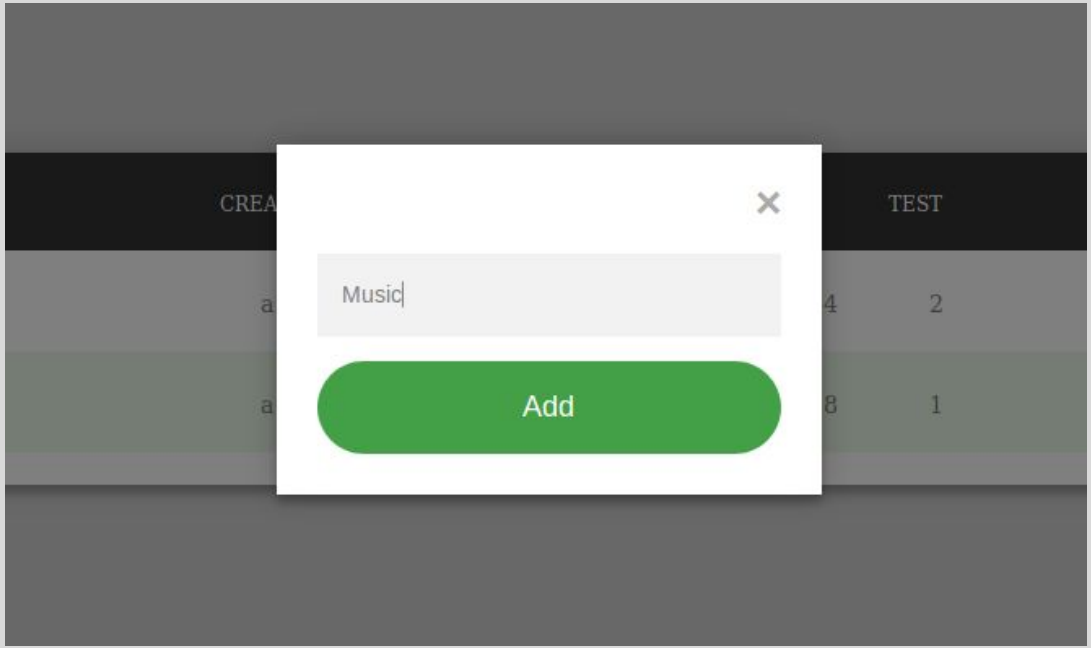
Schematizzando, il dialogo tra server

- Il client, con delle chiamate *POST* invoca delle procedure sul server
- Queste permettono di creare, leggere, aggiornare ed eliminare informazioni (secondo il paradigma *CRUD*)
- Il server risponde in formato *JSON* comunicando informazioni sullo stato ed eventuali messaggi d'errore
- Il client aggiorna localmente il contenuto della pagina sulla base del risultato delle chiamate (quando può, altrimenti richiede altre pagine al server)

Se quindi si volesse cambiare la parte che riguarda la presentazione questo non influirebbe sull'interfaccia alle informazioni. Di seguito alcuni esempi:

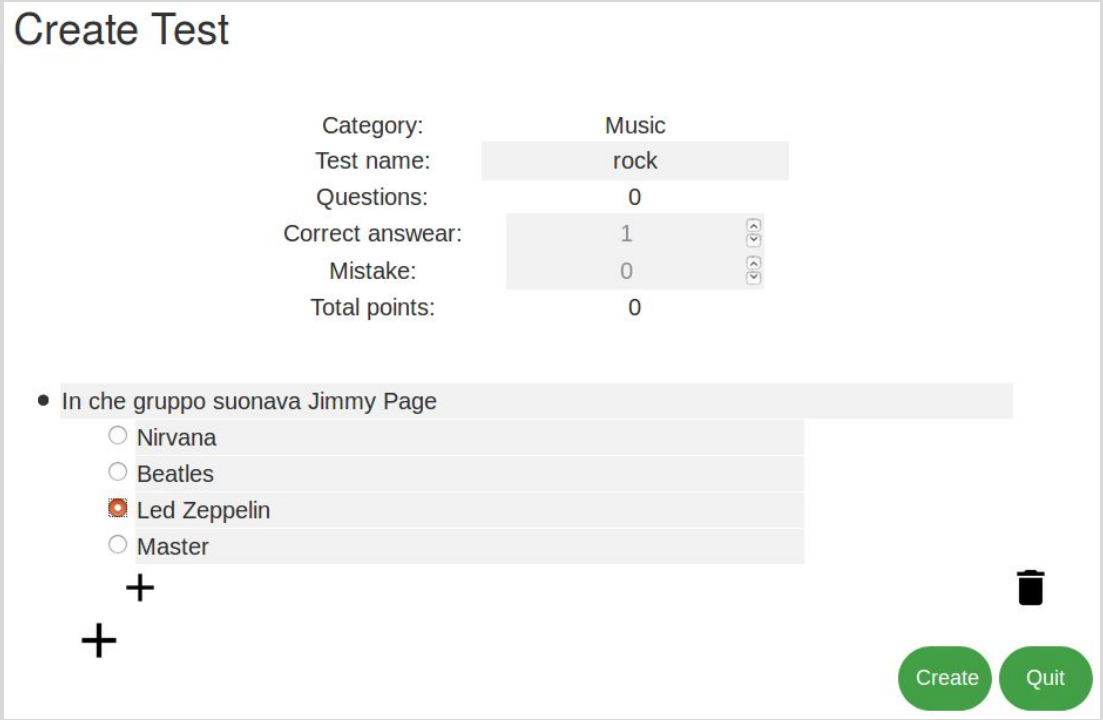
CRUD

- Nuove categorie



CRUD

- Nuovi test



CRUD

- Lettura informazioni carriera

	CATEGORY	RESULT	DATE
Relazioni	Math	100%	2018-06-16 15:44:31
Grammatica	Ita	66%	2018-06-16 15:50:52

- Lettura informazioni profilo

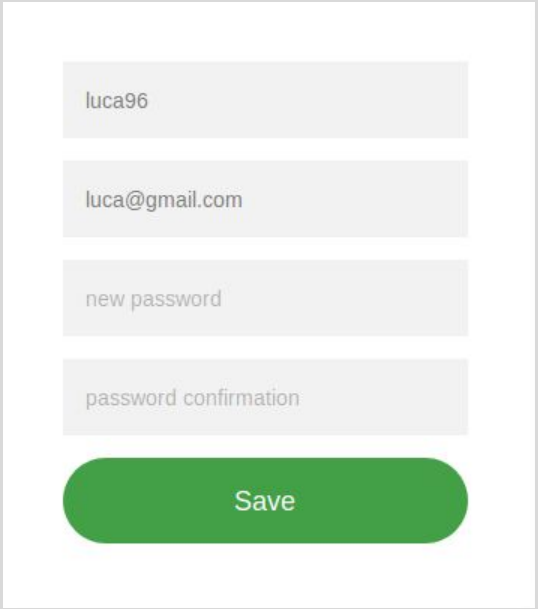
Username: luca96

Email: luca@gmail.com

Since: 2018-06-16 15:58:11

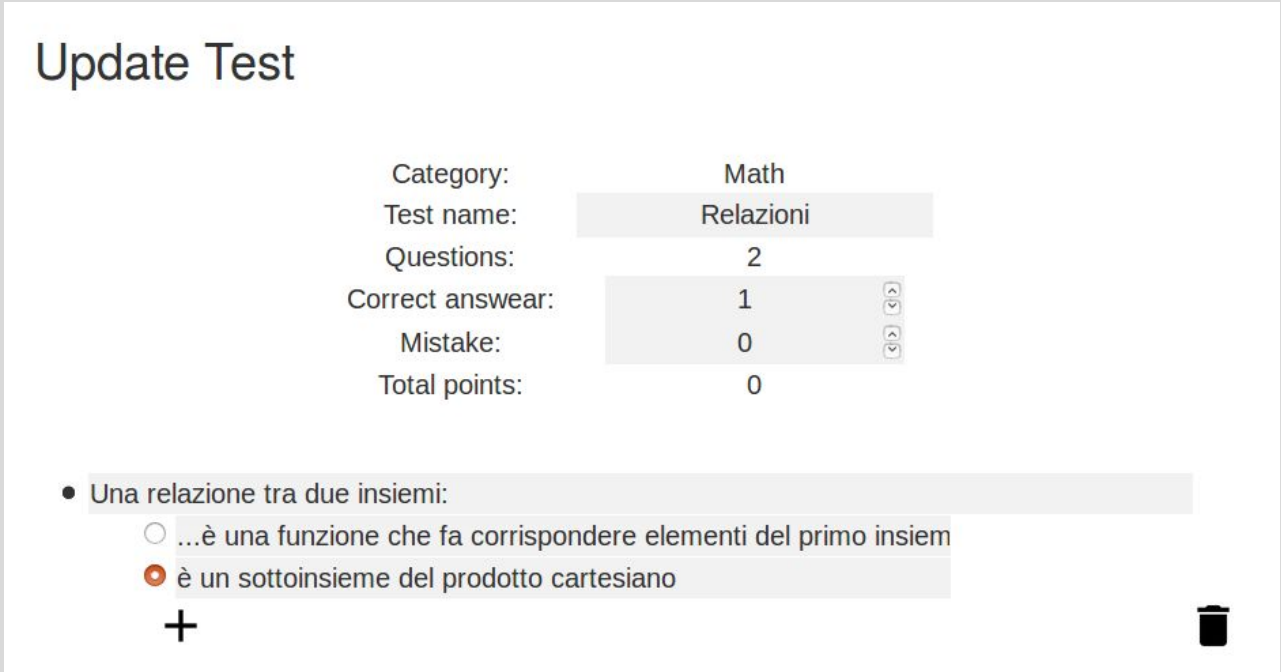
CRUD

- Modifica informazioni profilo



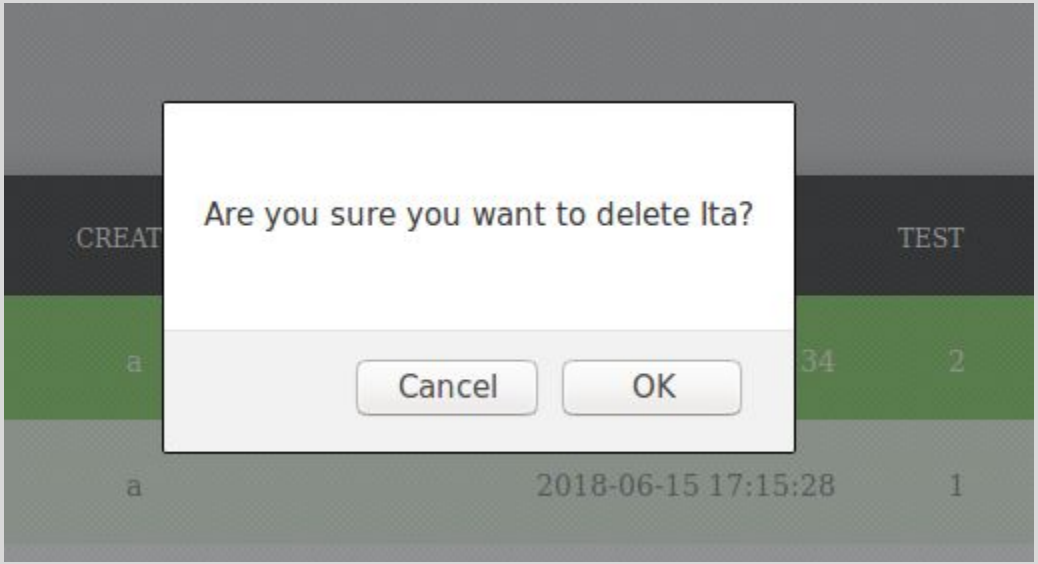
CRUD

- Modifica test



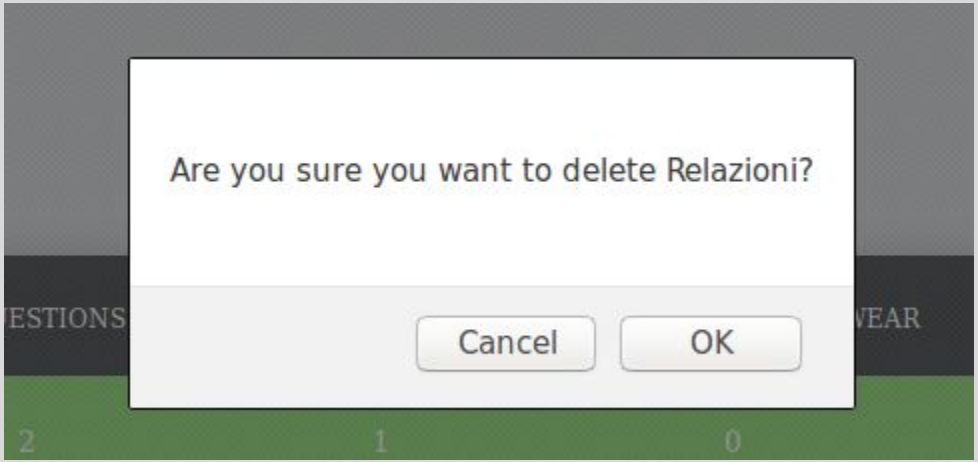
CRUD

- Cancellazione categorie



CRUD

- Cancellazione test



Cenni sull'Implementazione

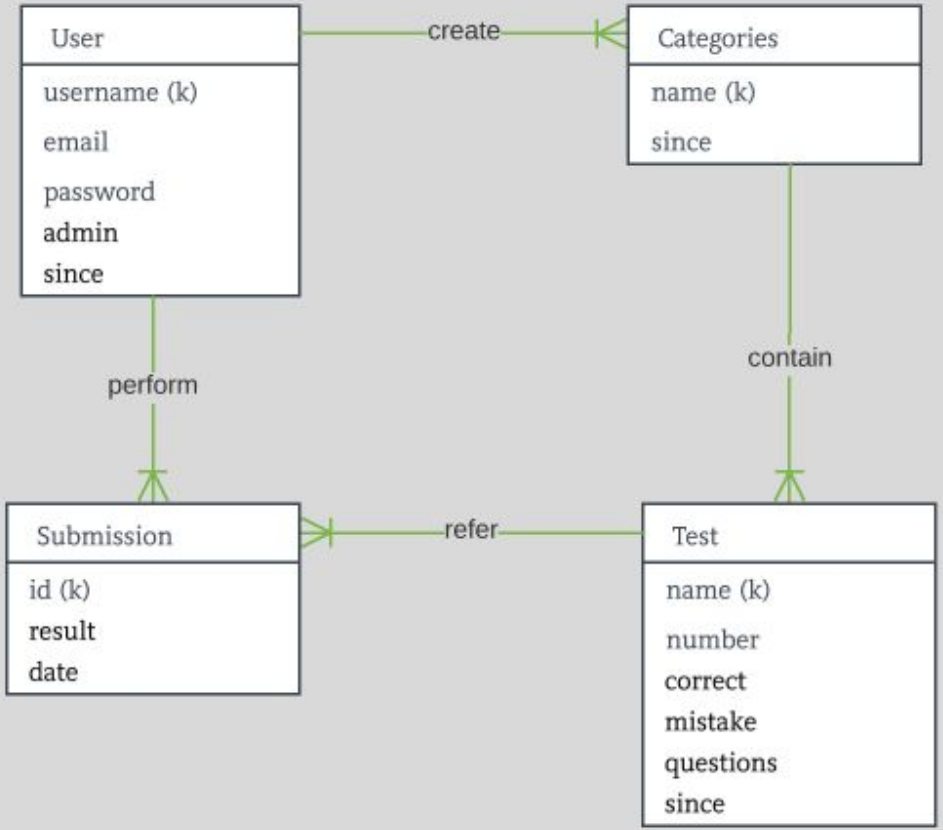
Il codice si suddivide in:

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

E il tutto si appoggia sui file di configurazione:

```php
require_once("config_local.php");

$config = array(
    "paths" => array(
        "resources" => "/Stressful/resources"
    ),
    "info" => array(
        "topbar" => array('Home', 'Career', 'Profile'),
    )
);

$config = array_merge($config, $config_local);


defined("PUBLIC_HTML_PATH")
    or define("PUBLIC_HTML_PATH", "http://localhost/Stressful/public_html");

defined("LIBRARY_PATH")
    or define("LIBRARY_PATH", realpath(dirname(__FILE__) . '/library'));

defined("FRAMEWORK_PATH")
    or define("FRAMEWORK_PATH", '/Stressful/resources/library');

defined("TEMPLATES_PATH")
    or define("TEMPLATES_PATH", realpath(dirname(__FILE__) . '/templates'));
.
```

Struttura Base di Dati



Interfaccia DB

Il database non è direttamente esposto, ma acceduto tramite una famiglia di classi *singletons*, una per ciascuna tabella, che offrono un'interfaccia più sicura.

Ad ogni classe corrisponde una particolare eccezione, che riguarda duplicati nel database, o violazione dei privilegi.

```php
class Connection {

    private static $conn = null;

    public static function get() {

        global $config;

        if ( self::$conn === null) {
            self::$conn = new mysqli(
                $config['db']['host'],
                $config['db']['username'],
                $config['db']['password'],
                $config['db']['database']
            );
        }

        return self::$conn;

    }
}
```

Esempio: class User

```
class User {

    private static $READ = "SELECT * FROM user WHERE username='%s' LIMIT 1";
    private static $CHECK = "SELECT * FROM user WHERE %s='%s' LIMIT 1";
    private static $UPDATE = "UPDATE user SET %s='%s' WHERE %s='%s' LIMIT 1";
    private static $SIGNUP = "INSERT INTO user (username, email, password) VALUES ('%s', '%s', '%s')";
    private static $LOGIN = "SELECT * FROM user WHERE username='%s' LIMIT 1";

    private $user = null;

    private function __construct() { }

    public static function get() {

        Session::start();

        if ( !isset($_SESSION[__CLASS__]) ) {
            $_SESSION[__CLASS__] = new static();
        }

        return $_SESSION[__CLASS__];

    }
 ...
}
```

Esempio: login method

```php
public function login($username, $password) {

    $db = Connection::get();

    $username = $db->real_escape_string($username);
    $password = md5($db->real_escape_string($password));

    $user = $db->query(sprintf(self::$LOGIN, $username))->fetch_assoc();

    $err = new UserException("Can't login User");

    if( !$user ) {
        $err->push('username', 'No user with given name');
    } else {
        if ($user['password'] != $password) {
            $err->push('password', 'Wrong password');
        }
    }

    if( $err->is_set() ) {
        throw $err;
    } else {
        $this->logged = true;
        $this->update_user($username);
    }
}
```

Esempio: class UserException

```
class UserException extends Exception {
    private $stack;

    public function __construct($message) {
        parent::__construct($message);
        $this->stack = null;
    }

    ...

    public function to_json() {

        $err = array(
            "error" => $this->stack
        );

        return json_encode($err);
    }
}
```

API

- Le *API* che costituiscono il servizio sono raccolte nel file *stressful_api.php*, possono essere invocate tramite chiamate *POST*, assegnando opportuni valori alle variabili.

- Mentre le pagine php contenute in *public_html* sono specifiche del sito, *stressful_api* potrebbe funzionare anche con un altro tipo di sistema informatica (es. un altro server, un client android).

Esempio: procedura login/signup

```php
if ( !$user->is_logged() ) {

    try {
        if(isset($_POST['login'])) {
            User::get()->login($_POST['username'], $_POST['password']);
        } else if(isset($_POST['signup'])) {
            User::get()->signup($_POST['username'], $_POST['email'], $_POST['password']);
        }
        echo '{"login" : true }';
    } catch ( UserException $err ) {
        echo $err->to_json();
    }

}
```

Esempio: login/signup client

```
var params = getParams();

if ( params !== null ) {

  $.post(STRESSFUL_API, params)
    .done(function(data) {

    var rep = JSON.parse(data);

    if ( rep.error ) {

      $("input").each(function() {

        var name = $(this).attr('name');

        if( rep.error[name] ) {
          $(this).after(errMsg(rep.error[name]))
        }

      });

    } else {
      setLocation('home');
    }

  });
```

Grazie!