Gestione e backup via web dei DB dei siti internet

Indice

[1 Introduzione 3](#_Toc19884237)

[1.1 Informazioni sul progetto 3](#_Toc19884238)

[1.2 Abstract 3](#_Toc19884239)

[1.3 Scopo 3](#_Toc19884240)

[1.4 Analisi del dominio 3](#_Toc19884241)

[1.5 Analisi e specifica dei requisiti 3](#_Toc19884242)

[1.6 Pianificazione 6](#_Toc19884243)

[1.7 Analisi dei mezzi 9](#_Toc19884244)

[1.7.1 Software 9](#_Toc19884245)

[1.7.2 Hardware 9](#_Toc19884246)

[2 Progettazione 9](#_Toc19884247)

[2.1 Design dell’architettura del sistema 9](#_Toc19884248)

[2.2 Design delle interfacce 9](#_Toc19884249)

[2.3 Design procedurale 13](#_Toc19884250)

[2.4 Design Database 13](#_Toc19884251)

[3 Implementazione 13](#_Toc19884252)

[3.1 Implementazione Frontend 13](#_Toc19884253)

[3.1.1 Pagina di Login 14](#_Toc19884254)

[3.1.2 Pagine principali 16](#_Toc19884255)

[3.2 Implementazione backend 17](#_Toc19884256)

[3.2.1 Gestione login e accessi 17](#_Toc19884257)

[4 Test 18](#_Toc19884258)

[4.1 Protocollo di test 18](#_Toc19884259)

[4.2 Risultati test 19](#_Toc19884260)

[4.3 Mancanze/limitazioni conosciute 19](#_Toc19884261)

[5 Consuntivo 19](#_Toc19884262)

[6 Conclusioni 20](#_Toc19884263)

[6.1 Sviluppi futuri 20](#_Toc19884264)

[6.2 Considerazioni personali 20](#_Toc19884265)

[7 Bibliografia 20](#_Toc19884266)

[7.1 Sitografia 20](#_Toc19884267)

[8 Allegati 20](#_Toc19884268)

# Introduzione

## Informazioni sul progetto

Allievo coinvolto: Paolo Guebeli

Classe: Informatica 4AA Presso la Scuola Arti e Mestieri Trevano

Docente responsabile: Fabrizio Valsangiacomo

Data inizio: 3-09-19

Data fine: 20-12-19

## Abstract

## Scopo

**Analisi**

## Analisi del dominio

Al momento è difficile trovare un prodotto del genere quindi non c’è una competizione per questo programma. C’è molta competizione nel campo dei programmi di backup ma raramente si trova una soluzione web che permette la gestione dei backup da qualsiasi posto in cui ci si trova. In generale si possono trovare altre soluzioni al problema che risolve questo progetto ma nessuna sarà personalizzata perfettamente per questo caso come questo progetto.

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-01** | |
| **Nome** | Pagina di login |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Pagina di login che ci permette di accedere al sito web |
| **Sotto requisiti** | |
| **01** | Pagina creazione account |
| **02** | Verifica creazione da parte di un admin |

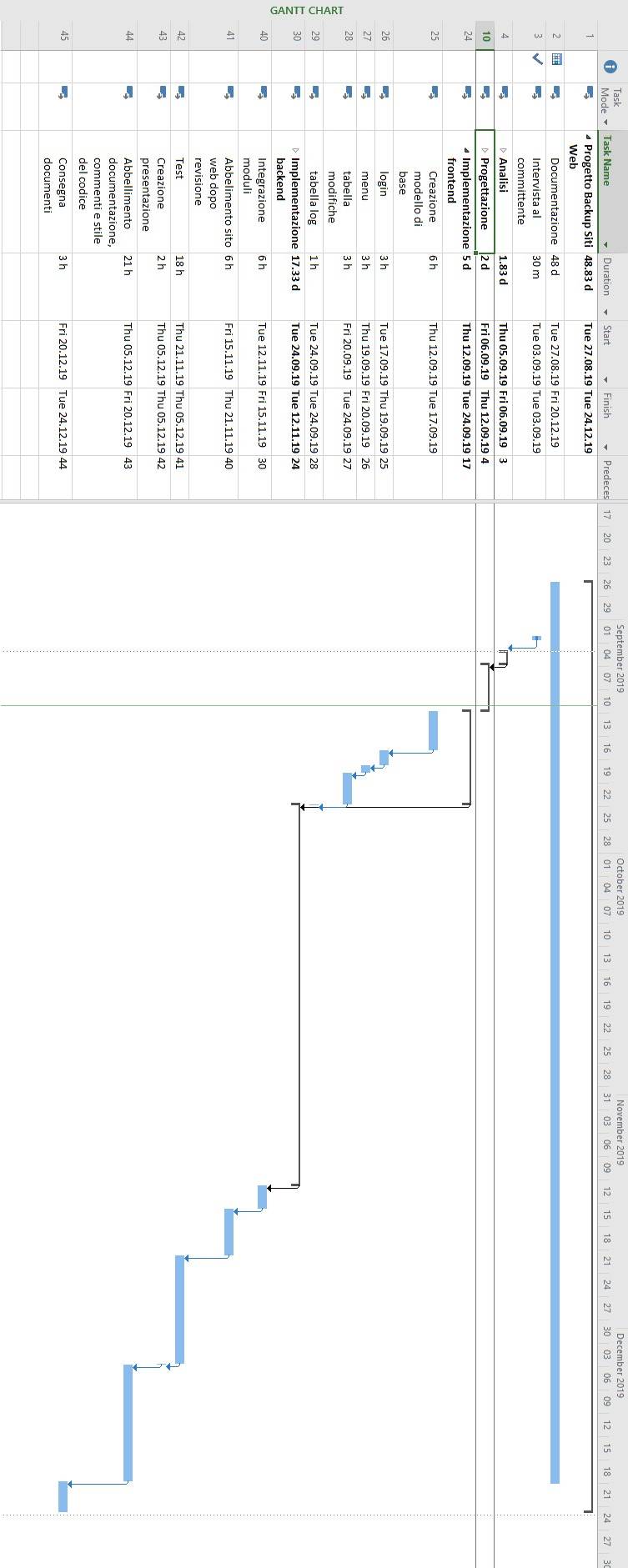
|  |  |
| --- | --- |
| **ID: REQ-02** | |
| **Nome** | Pagina home |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Pagina che mostra i dati generali dei backup, e in caso di problemi li notifica |
| **Sotto requisiti** | |
| **01** | Un menu che permette la navigazione attraverso il sito |
| **02** | Una parte dedicata alla gestione del proprio account |
| **03** | Interfaccia user friendly |

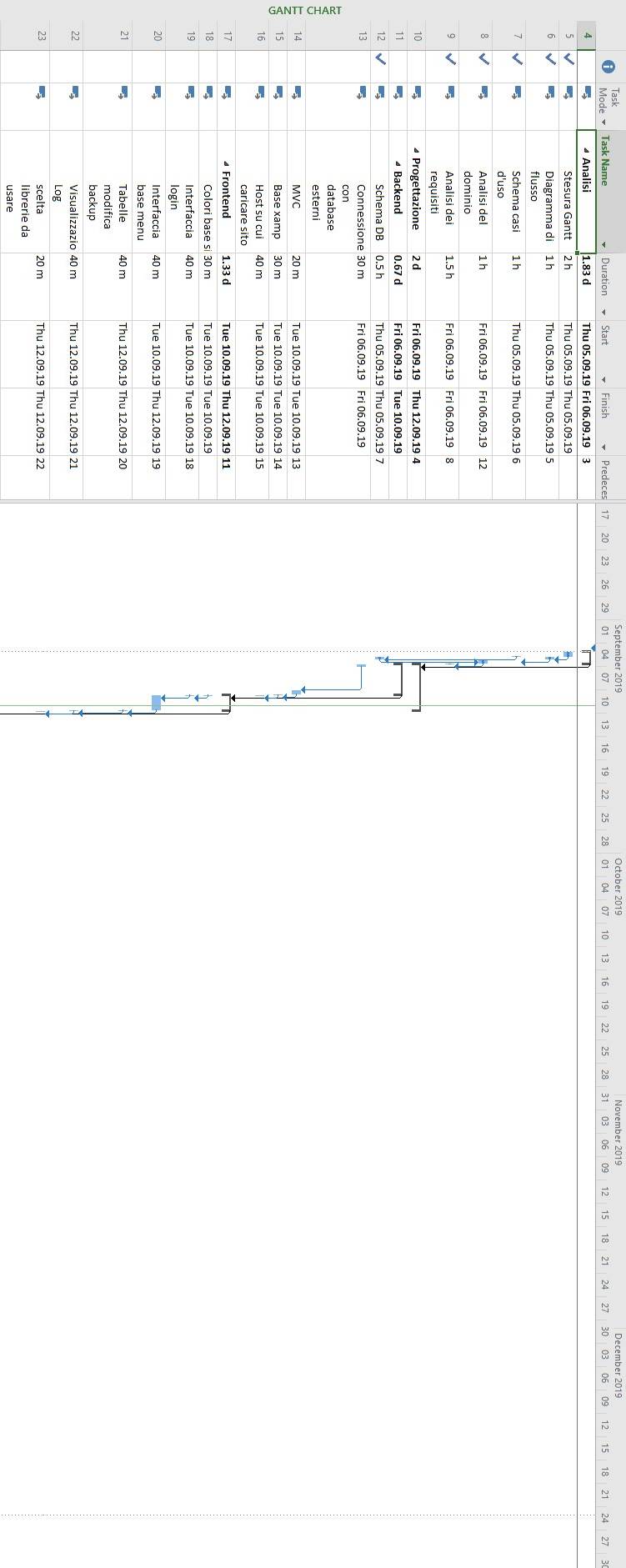
|  |  |
| --- | --- |
| **ID: REQ-03** | |
| **Nome** | Pagina gestione backup |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Pagina che mostra i backup in una tabella e i diversi tool di gestione |
| **Sotto requisiti** | |
| **01** | Possibilità di eliminare un backup |
| **02** | Possibilità di cambiare le impostazioni del backup |
| **03** | Possibilità di gestire i log del backup |
| **04** | Possibilità di fare un backup manuale |
| **05** | Possibilità di vedere i dati principali dei backup (grandezza, data inizio, impostazioni) |

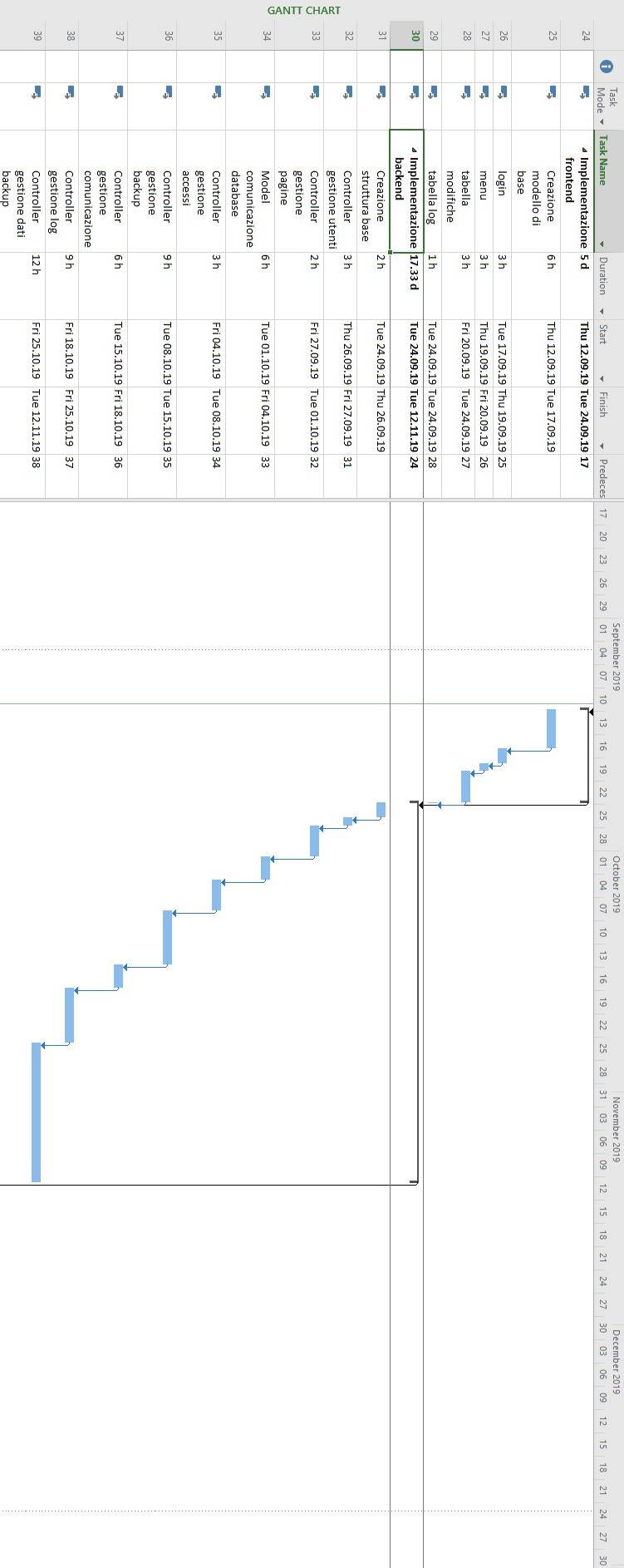
|  |  |
| --- | --- |
| **ID: REQ-04** | |
| **Nome** | Salvataggio database di backup |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | I database verranno salvati come file .sql |
| **Sotto requisiti** | |
| **01** | Si dovrà richiedere i dati di accesso per connettersi ai database esterni |
| **02** | I backup dovranno essere eseguiti a orari e date specifici |
| **03** | Il recupero dei dati del database dev’essere facile e veloce |

|  |  |
| --- | --- |
| **ID: REQ-05** | |
| **Nome** |  |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | Si necessita di una tabella che prende tutti i dati salvati nel file csv “Registrazione\_aaaa\_mm\_gg.csv” dello stesso giorno e li mostri |
| **Sotto requisiti** | |
| **01** | Si necessita di visualizzare tutti i dati salvati del giorno in una tabella |
| **02** | Si necessita del bottone “home” che al suo click ci riporta alla pagina iniziale |

## Pianificazione







## Analisi dei mezzi

### Software

Per la realizzazione di questo progetto ho usato come software:

* XAMP: Software che mi permette di gestire database e webserver in contemporanea.
* PHP : Interprete che si occupa di permettere al server di lavorare con file scritti in PHP.
* JetBrains phpStorm 2019.1: Software per l’implementazione della parte PHP, HTML, CSS e JS del progetto.

Inoltre ho usato le seguenti librerie:

* Font Awesome 5.3.1: Libreria che fornisce le varie icone come quelle presenti nei form.
* Jquery 3.4.1: Libreria che abbrevia la scrittura del codice Javascript

### Hardware

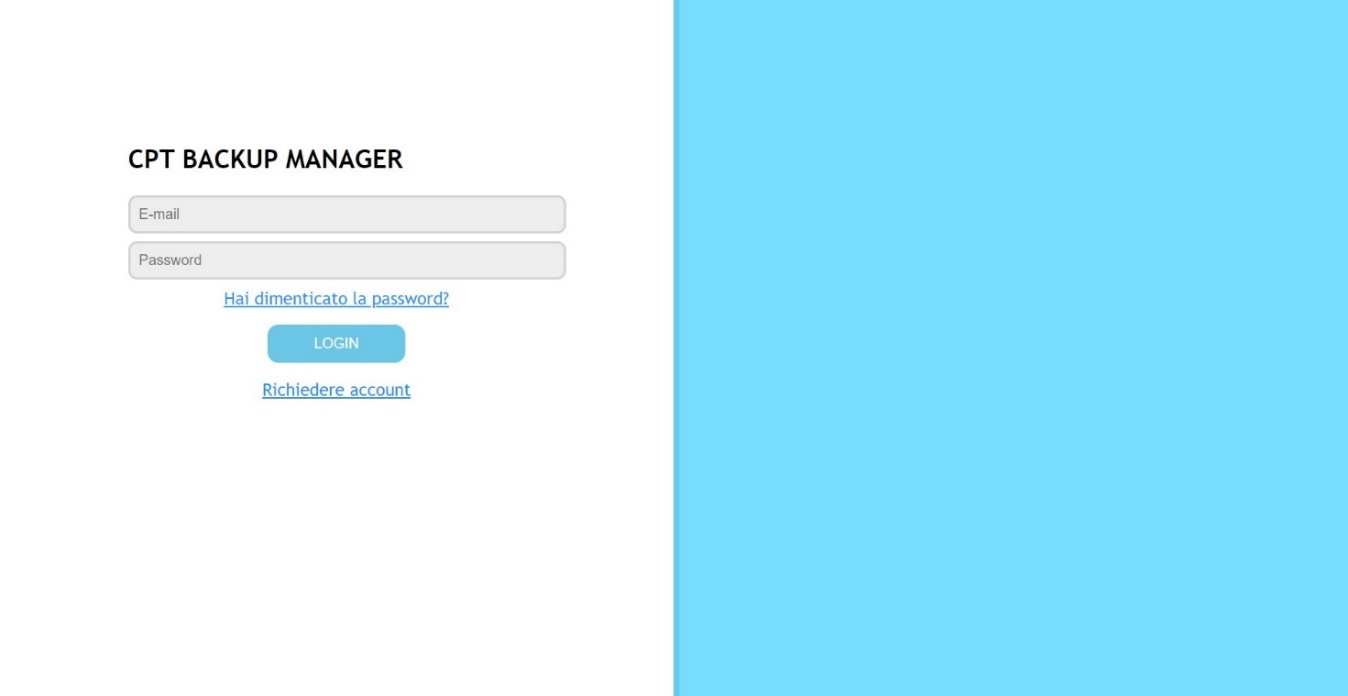
Per questo progetto non ho avuto hardware particolari e ho solo usato il mio portatile:

HP Omen 17 i7-6700HQ CPU 2.6GHz RAM 16GB

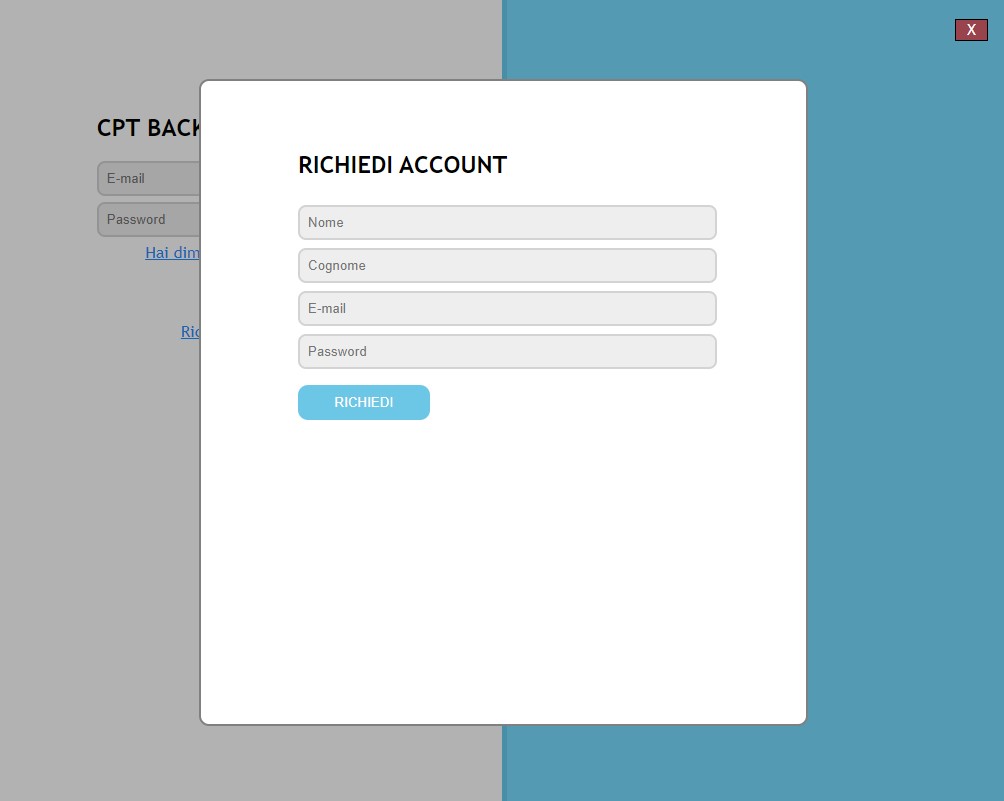
# Progettazione

## Design dell’architettura del sistema

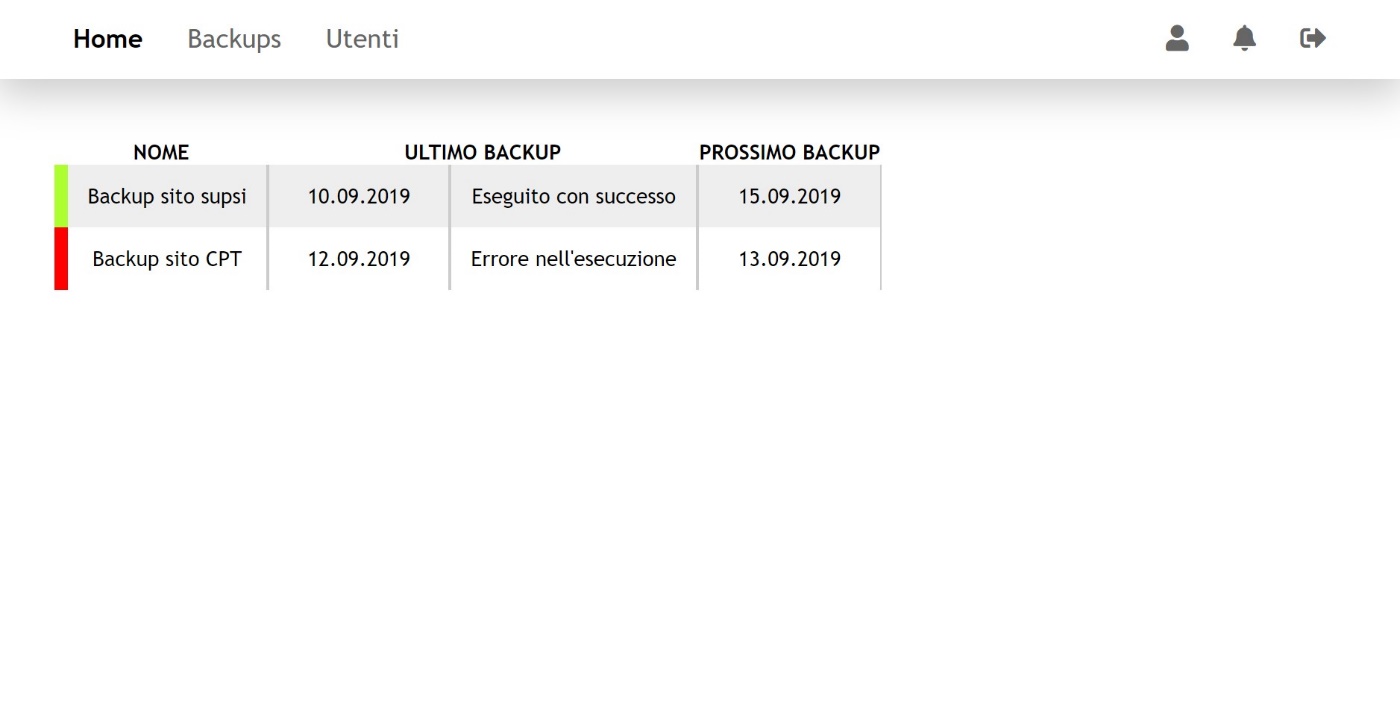
## Design delle interfacce



Come si può vedere il design è molto semplice anche perché il sito web dev’essere più user friendly possibile. La parte destra della pagina verrà lasciata libera in caso si voglio aggiungere qualcosa alla pagina di benvenuto. La parte di login sulla sinistra occuperà il 25% della pagina in modalità desktop, in modalità tablet il 50% e in modalità smartphone/mobile il 100%. Questo per rendere a seconda del dispositivo tutte le scritte leggibili. Il titolo è ancora da discutere con il committente al momento è solo per avere un esempio del font dei titoli.



In caso che venga richiesto un account per accedere comparirà questo form sovrapposto alla pagina. Il form come la pagina si adatterà a seconda della grandezza della pagina. Per uscire da questo form si può o cliccare al di fuori del form o sul pulsante in alto a destra. Il pulsante a destra è presente in caso che non fosse possibile cliccare al di fuori del form. Un form simile comparirà in caso si voglia cambiare la password del proprio account. Ma in quel caso conterrà solo il campo e-mail.



Il sito di base è molto semplice, ho deciso di suddividere la barra di navigazione in due parti:

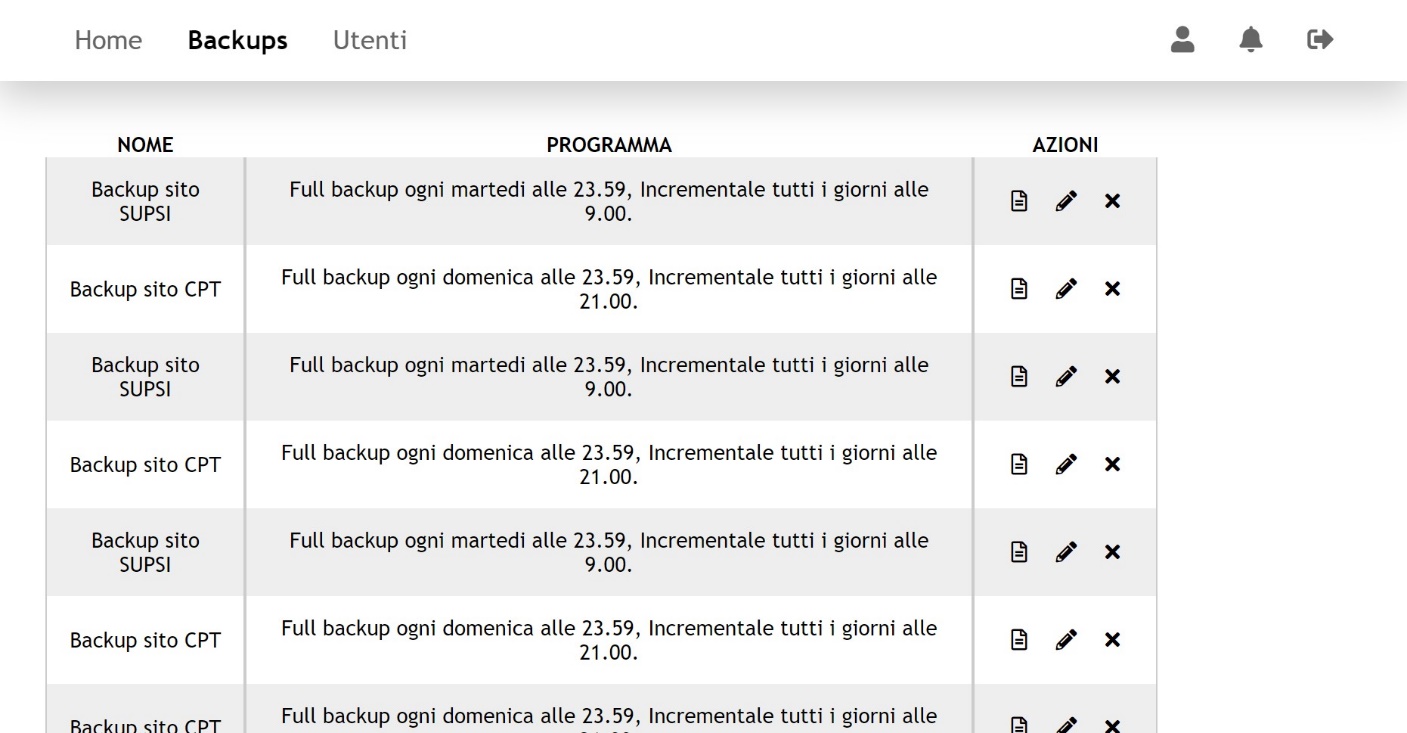
* A sinistra ci saranno le diverse pagine in cui si può andare, ovviamente il numero di pagine dipende dal livello di acceso che si ha.
* A destra c’è la gestione del proprio account le notifiche (sarà di colore rosso se ce ne sono) e il Logout che ovviamente permette all’utente di uscire dal proprio account.

Nella parte principale della pagina ci sarà una tabella contenente tutti i database con dei dati generali. In questo caso ho scelto dei dati semplici quindi:

* Nome del database per renderne possibile l’identificazione.
* L’ultimo backup e come è andato quindi successo, errore o problemi ma completato.
* E infine il prossimo backup pianificato così da sapere quando verranno aggiornati i dati.

La barra di navigazione seguirà la pagina se si trascina verso il basso. In più in modalità mobile diventerà un’ icona in alto a sinistra che si potrà aprire per aver un menu che esce dalla sinistra. La tabella si ridimensiona insieme alla pagina.

Se si clicca sull’icona delle notifiche ci sarà un piccolo pop up che mostra le notifiche che potranno essere eliminate.

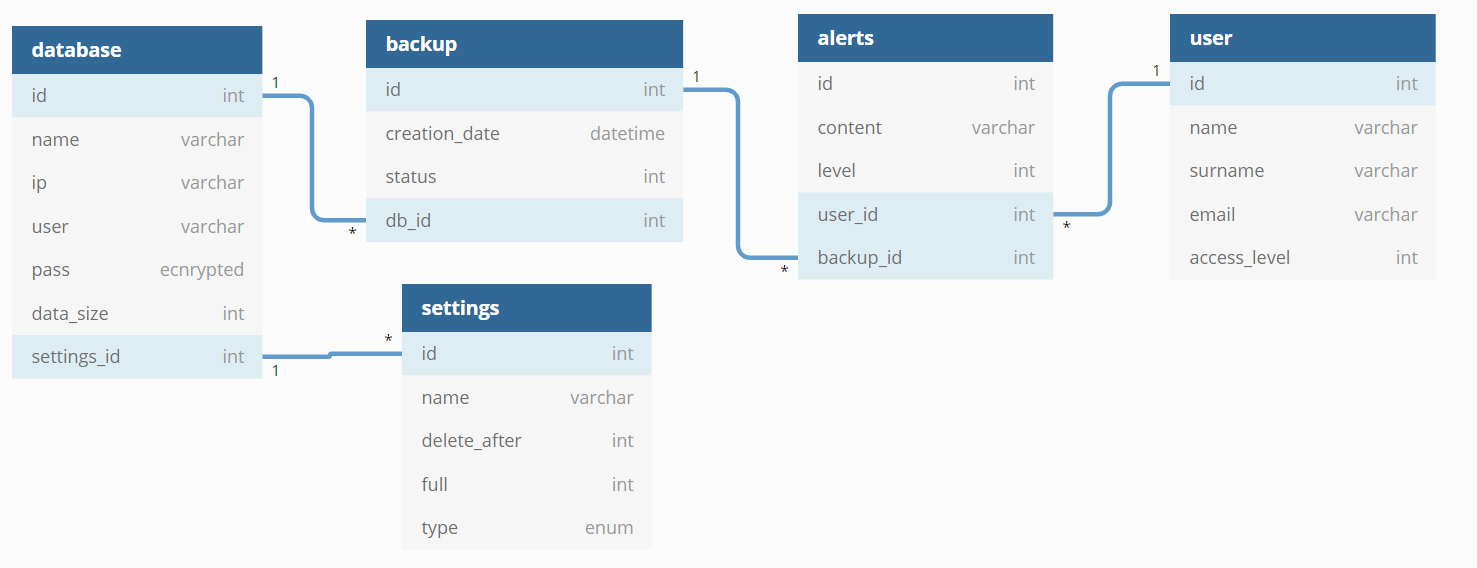


La pagina di gestione dei backup comprende il menu di base e una tabella contenente tutti i database a cui bisogna fare i backup. La tabella ha tre colonne:

* Nome: nome del database di cui si fa il backup.
* Programma: contiene quando vengono fatti quali backup e di che tipo sono.
* Azioni: contiene le azioni possibili da fare sul backup

## Design procedurale

## Design Database

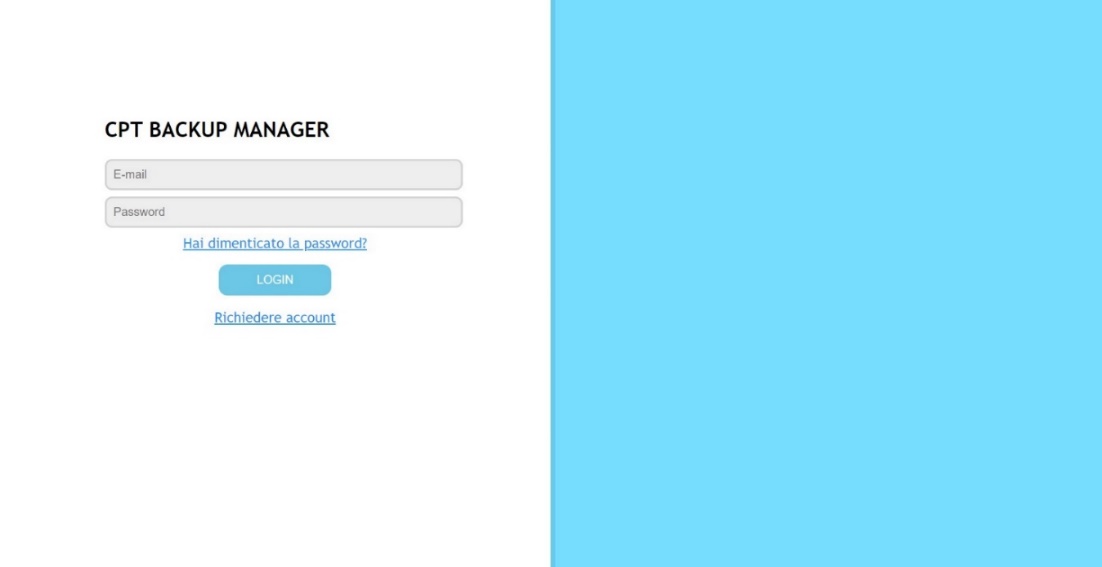


# Implementazione

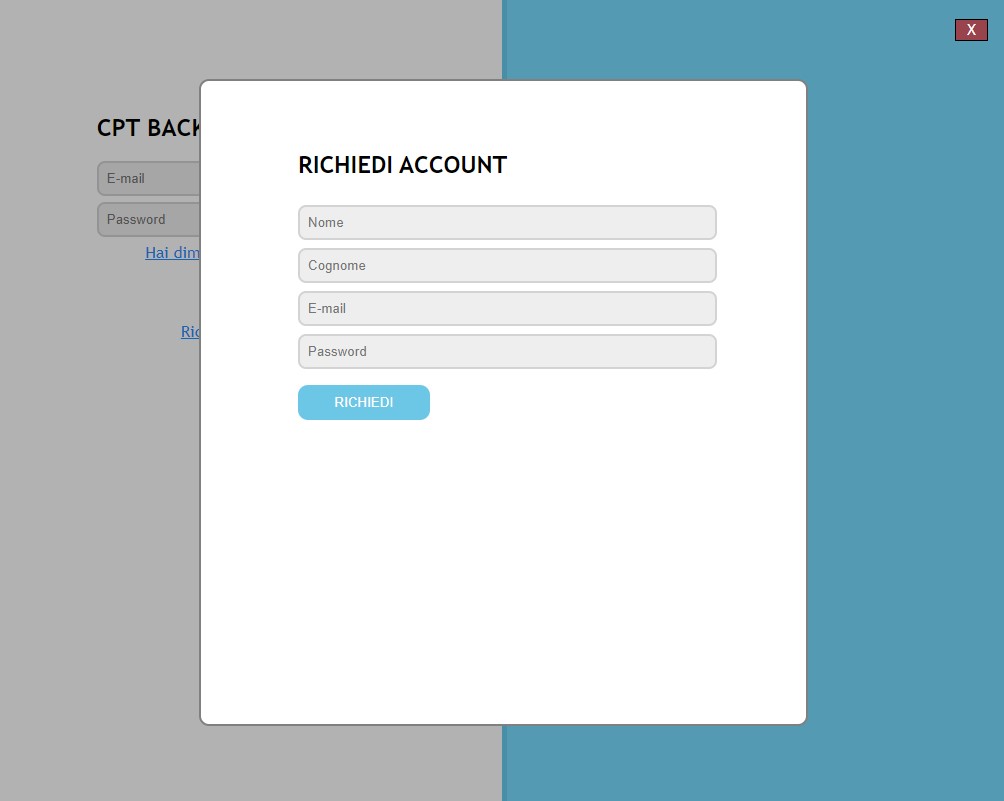
## Implementazione Frontend

Questa parte di implementazione comprende il codice HTML e CSS delle pagine web quindi tutto quello che è la gestione dell’estetica del progetto. Principalmente ho usato uno stile semplice ed intuitivo che quindi tutti i tipi di utenti dovrebbero essere in grado di usarlo.

### Pagina di Login



Per implementare questa pagina ho semplicemente diviso il contenuto con 2 tag div uno al momento vuoto in cui si può inserire dei dati d’interesse. Invece l’altro contiene un tag form con due input di testo, quando si preme il pulsante login i dati inseriti verranno inviati a una funzione in PHP che li verificherà e se sono validi si potrà accedere alla pagina home. Se le credenziali non sono valide comparirà un messaggio d’errore sotto agli input. I messaggi d’errore possibili sono “Inserire credenziali”, “username o password non sono validi” e “account non verificato”.



Quando si richiede un account comparirà un form che quando si preme il pulsante “RICHIEDI” questo invierà i dati del nuovo account a una funzione in PHP che li verificherà e li inserirà nel database. In seguito comparirà una scritta che conferma la creazione o segnala se c’è stato un errore di qualche tipo.

<**body**>  
<**div class="cover" id="creationTab"**>  
 <**div id="closeButton"**> X </**div**>  
 <**form method="post" action="<?php echo *URL*?>users/addUser" class="creationForm" id="creationForm"**>  
 <**div class="row spacing-top-lg spacing-left-xl col-8"**>  
 <**h2 class="title" style="text-align**: **left"**>RICHIEDI ACCOUNT</**h2**>  
 <**input class="spacing-top-sm" type="text" placeholder="Nome" name="name"**>  
 <**input class="spacing-top-sm" type="text" placeholder="Cognome" name="lastname"**>  
 <**input class="spacing-top-sm" type="text" placeholder="E-mail" name="email"**>  
 <**input class="spacing-top-sm" type="password" placeholder="Password" name="password"**>  
 <**div class="row spacing-top-md"**>  
 <**input class="col-4" type="submit" value="RICHIEDI"**>  
 </**div**>  
 **<?php if**(**isset**($\_POST[**'creationError'**])): **?>**<**p class="error"**>**<?php echo** $\_POST[**'creationError'**] **?>**</**p**>**<?php endif**; **?>** </**div**>  
 </**form**>  
</**div**>  
<**div class="row" id="container"**>  
 <**div class="side" id="login"**>  
 <**div class="row"**>  
 <**center**>  
 <**form action="<?php echo *URL*?>login/verify" method="post"**>  
 <**div class="col-8 spacing-top-xl"**>  
 <**h2 class="title"**>  
 CPT BACKUP MANAGER  
 </**h2**>  
 <**input type="text" placeholder="E-mail" name="email"**>  
 <**input class="spacing-top-sm" type="password" placeholder="Password" name="pass"**>  
 **<?php if**(**isset**($\_POST[**'error'**])): **?>**<**p class="error"**>**<?php echo** $\_POST[**'error'**] **?>**</**p**>**<?php endif**; **?>** <**div class="linked spacing-top-sm" id="passReset"**>Hai dimenticato la password?</**div**>  
 <**div class="row spacing-top-md"**>  
 <**input class="col-4" type="submit" value="LOGIN"**>  
 </**div**>  
 <**div class="linked spacing-top-md" id="new"**>Richiedere account</**div**>  
 </**div**>  
 </**form**>  
 </**center**>  
 </**div**>  
 </**div**>  
</**div**>  
</**body**>

Questo è il body della pagina di login, si può suddividere in due la parte di creazione del account e la parte di login.

<**div class="cover" id="creationTab"**>  
 <**div id="closeButton"**> X </**div**>  
 <**form method="post" action="<?php echo *URL*?>users/addUser" class="creationForm" id="creationForm"**>  
 <**div class="row spacing-top-lg spacing-left-xl col-8"**>  
 <**h2 class="title" style="text-align**: **left"**>RICHIEDI ACCOUNT</**h2**>  
 <**input class="spacing-top-sm" type="text" placeholder="Nome" name="name"**>  
 <**input class="spacing-top-sm" type="text" placeholder="Cognome" name="lastname"**>  
 <**input class="spacing-top-sm" type="text" placeholder="E-mail" name="email"**>  
 <**input class="spacing-top-sm" type="password" placeholder="Password" name="password"**>  
 <**div class="row spacing-top-md"**>  
 <**input class="col-4" type="submit" value="RICHIEDI"**>  
 </**div**>  
 **<?php if**(**isset**($\_POST[**'creationError'**])): **?>**<**p class="error"**>**<?php echo** $\_POST[**'creationError'**] **?>**</**p**>**<?php endif**; **?>** </**div**>  
 </**form**>  
</**div**>

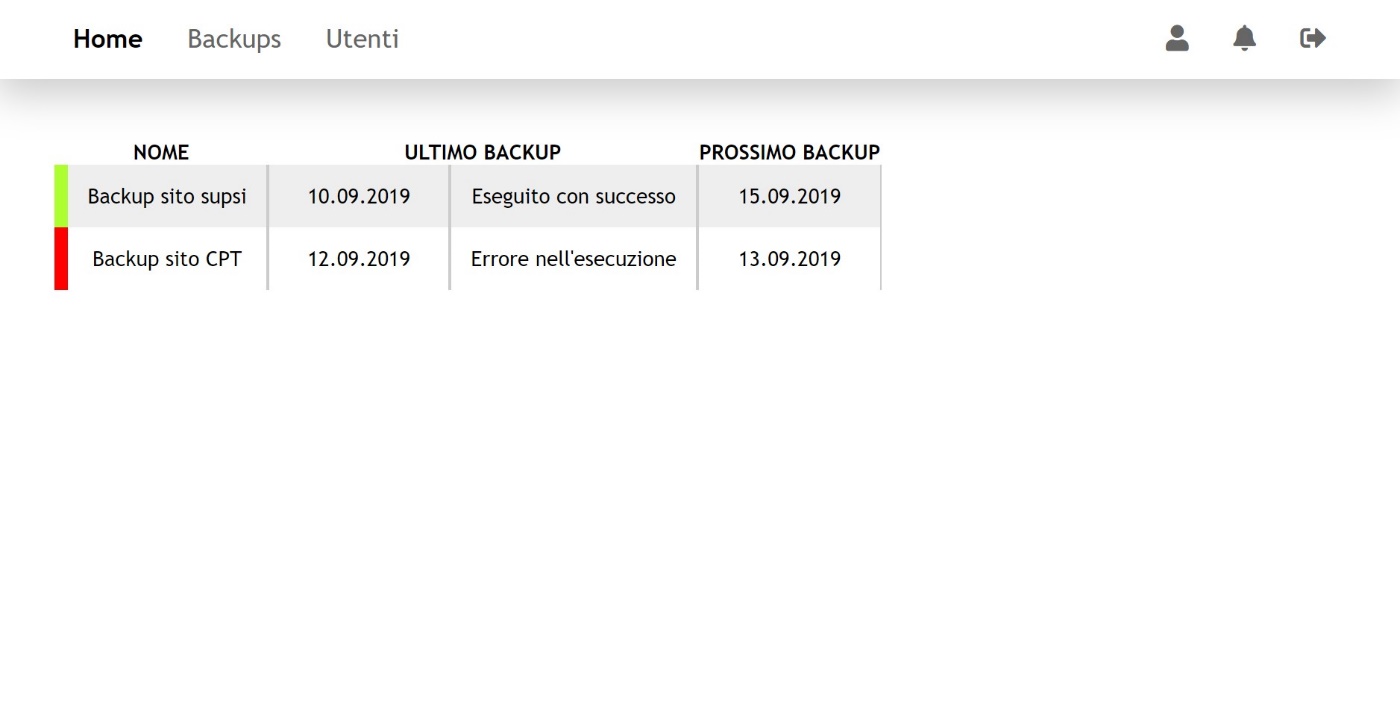
In questa parte di codice si possono vedere tutti gli input del form di creazione del account.

Se c’è stato un errore verrà segnalato nel paragrafo alla fine del form. Ovviamente per poter avere dei messaggi personalizzati devo usare php e non HTML o CSS.

<**div class="row" id="container"**>  
 <**div class="side" id="login"**>  
 <**div class="row"**>  
 <**center**>  
 <**form action="<?php echo *URL*?>login/verify" method="post"**>  
 <**div class="col-8 spacing-top-xl"**>  
 <**h2 class="title"**>  
 CPT BACKUP MANAGER  
 </**h2**>  
 <**input type="text" placeholder="E-mail" name="email"**>  
 <**input class="spacing-top-sm" type="password" placeholder="Password" name="pass"**>  
 **<?php if**(**isset**($\_POST[**'error'**])): **?>**<**p class="error"**>**<?php echo** $\_POST[**'error'**] **?>**</**p**>**<?php endif**; **?>** <**div class="linked spacing-top-sm" id="passReset"**>Hai dimenticato la password?</**div**>  
 <**div class="row spacing-top-md"**>  
 <**input class="col-4" type="submit" value="LOGIN"**>  
 </**div**>  
 <**div class="linked spacing-top-md" id="new"**>Richiedere account</**div**>  
 </**div**>  
 </**form**>  
 </**center**>  
 </**div**>  
 </**div**>  
</**div**>

Nella parte di login si può notare che anche qui i errori sono gestiti tramite PHP e che i margini e le dimensioni sono gestite tramite classi CSS. Per esempio spacing-top-sm crea un margine in alto piccolo (xs=extrasmall, sm = small, md=medium, lg=large, xl=extralarge).

### Pagine principali



Le pagine principali hanno tutte la stessa struttura, hanno la barra in alto che permette di navigare attraverso il sito e le icone a destra che sono:

* Gestione del account
* Campanella delle notifiche che diventa rossa se c’è qualcosa da fare
* E il Logout

Queste sono secondo me le funzionalità di base che ci devono essere in ognuno dei siti principali.

Nella parte del contenuto della pagina hanno tutte una tabella:

* La pagina home ha una tabella che contiene l’ultimo backup di ogni database
* La pagina dei backup contiene una tabella che permette di gestire i backup. Si possono vedere i log dei backup, si può modificare il programma dei backup (quando farli e di che tipo) e si può eliminare il processo(non elimina i backup solo il programma).
* La pagina degli utenti contiene tutti gli utenti che possono essere modificati (tutti i campi tranne la password), se non sono ancora stati abilitati possono essere abilitati e possono essere eliminati basta che rimane un amministratore.

## Implementazione backend

Per il backend ho cercato di suddividere tutto nelle classi apposite così da rendere tutto il codice più pulito.

### Gestione login e accessi

**public function** index()  
 {  
 **if**($this->check()){  
 **require 'views/home.php'**;  
 }**else**{  
 **require 'views/login.php'**;  
 }  
 }

**public function** check(){  
 **if**(**isset**($\_COOKIE[**'backup\_site'**])) {  
 $results = db\_manager::*executeQuery*(**"SELECT** *\** **FROM users where pass ='"**.$\_COOKIE[**'backup\_site'**].**"'"**);  
 **if** ($results !== **NULL**){  
 **if**($results[0][**'access'**] > 0){  
 **return true**;  
 }**else**{  
 $\_POST[**'error'**] = **'account non verificato'**;  
 **return false**;  
 }  
 }**else**{  
 **return false**;  
 }  
 }**else**{  
 **return false**;  
 }  
 }

La funzione index() serve a richiamare la home del sito, per verificare se l’utente ha accesso alla home usa la funzione check() che controlla i dati del utente. Inizia a controllare se c’è un cookie per l’accesso, se c’è controlla se il valore che contiene è presente nel database da lì poi sa chi è l’utente e quindi sa a quali pagine può accedere e a quali non può accedere.

**public function** admin(){  
 $results = db\_manager::*executeQuery*(**"SELECT** *\** **FROM users where pass ='"**.$\_COOKIE[**'backup\_site'**].**"'"**);  
 **if**($results !== **NULL**) {  
 **if** ($results[0][**'access'**] > 1) {  
 **return true**;  
 } **else** {  
 **return false**;  
 }  
 }**else**{  
 **return false**;  
 }  
 }

La funzione admin() semplicemente ritorna se l’utente è amministratore quindi se può accedere a funzionalità avanzate o no.

**public function** verify()  
 {  
 **if**($\_POST[**'email'**] != **null**){  
 $results = db\_manager::*executeQuery*(**"SELECT** *\** **from users where email ='"**.$\_POST[**'email'**].**"'"**);  
 $hash = $results[0][**'pass'**];  
 **if** (*password\_verify*($\_POST[**'pass'**], $hash)) {  
 *setcookie*(**'backup\_site'**, $hash, *time*() + (86400 \* 30), **"/"**);  
 $first = **true**;  
 **require 'views/home.php'**;  
 } **else** {  
 $\_POST[**'error'**] = **'username o password non validi'**;  
 $this->index();  
 }  
 }  
 }  
}

Questa funzione serve esclusivamente al login, infatti controlla se i dati di accesso inseriti nel form sono corretti e validi. Se i dati inseriti corrispondo ad un account esistente crea un cookie contente l’hash code unico per ogni utente che verrà usato come identificativo.

### Gestione utenti

La gestione degli utenti è solo accessibile per gli amministratori. Si devono poter verificare gli utenti, modificarli ed eliminarli. Però non si può eliminare l’ultimo amministratore, io per ovviare al problema semplicemente ho messo che non si può eliminare il proprio utente o cambiargli il livello di accesso.

**public function** addUser()  
{  
 **if** (**isset**($\_POST[**'name'**]) && **isset**($\_POST[**'lastname'**]) && **isset**($\_POST[**'email'**]) && **isset**($\_POST[**'password'**])) {  
 $name = validate::*secureString*($\_POST[**'name'**]);  
 $lastname = validate::*secureString*($\_POST[**'lastname'**]);  
 **if** (validate::*checkEmail*($\_POST[**'email'**])) {  
 **if** (db\_manager::*executeQuery*(**"SELECT** *\** **from users where email ='"** . $\_POST[**'email'**] . **"'"**) == **NULL**) {  
 $hash = *password\_hash*($\_POST[**'password'**], ***PASSWORD\_DEFAULT***) . **"\n"**;

db\_manager::*execute*(  
 **"INSERT into users (name,lastname,email,pass,access) values ('"** . $name . **"','"** . $lastname . **"','"** . $\_POST[**'email'**] . **"','"** . $hash . **"',0)"** );

$user = db\_manager::*executeQuery*(**"SELECT** *\** **from users where email ='"** . $\_POST[**'email'**] . **"'"**);

db\_manager::*execute*(  
 **"INSERT into alerts (content,level,user\_id) values ('"** . $name . **" "** . $lastname . **" richiede un account','2','"** . $user[0][**'id'**] . **"')"** );

$\_POST[**'creationError'**] = **'L\'account è stato creato, verrà verificato nei prossimi giorni'**;  
 **require**(**'views/login.php'**);  
 } **else** {  
 $\_POST[**'creationError'**] = **'Esiste già un account con questa email'**;  
 **require**(**'views/login.php'**);  
 }  
 } **else** {  
 $\_POST[**'creationError'**] = **'Email non valida'**;  
 **require**(**'views/login.php'**);  
 }  
 }  
}

Questa parte di codice è la funzione per la creazione degli utenti, nella parte principale dopo i controlli dei dati inserisco nel database i dati. Le due cose principali che faccio è creare l’utente assegnandoli il livello di accesso 0 che corrisponde a account non verificato, in seguito inserisco nella tabella delle notifiche che un utente aspetta per la verifica da parte di un amministratore.

**public function** verify($id)  
{  
 **if** (validate::*checkInt*($id)) {  
 db\_manager::*execute*(  
 **"UPDATE users set access = 1 where id = "** . $id  
 );  
 db\_manager::*execute*(  
 **"DELETE from alerts where user\_id = "** . $id  
 );  
 }  
 $this->home();  
}

Questo metodo serve a verificare un utente, quindi cambiare il suo livello da non verificato (0) a responsabile (1). Prima di cambiare il dato nel database controllo se l’id passato è un int così da evitare dati fasulli.

**public function** modify($id)  
{  
 $name = validate::*secureString*($\_POST[**'mName'**]);  
 **if** ($name != **NULL**) {  
 db\_manager::*execute*(  
 **"UPDATE users set name = '"** . $name . **"' where id = "** . $id  
 );  
 }

$lastname = validate::*secureString*($\_POST[**'mLastname'**]);  
 **if** ($lastname != **NULL**) {  
 db\_manager::*execute*(  
 **"UPDATE users set lastname = '"** . $lastname . **"' where id = "** . $id  
 );  
 }

**if** ($\_POST[**'mEmail'**] != **NULL** && validate::*checkEmail*($\_POST[**'mEmail'**])) {  
 db\_manager::*execute*(  
 **"UPDATE users set email = '"** . $\_POST[**'mEmail'**] . **"' where id = "** . $id  
 );  
 }

**if** ($\_POST[**'mAccess'**] != **NULL** && validate::*checkInt*($\_POST[**'mAccess'**])) {  
 db\_manager::*execute*(  
 **"UPDATE users set access = "** . $\_POST[**'mAccess'**] . **" where id = "** . $id  
 );  
 }

$this->home();  
}

Questo metodo serve a modificare i dati di un utente, tutti i dati possono essere modificati indipendentemente dagli altri. Per riuscire a gestire la modifica multipla ho dovuto semplicemente suddividere tutti i dati in degli if singoli. Così facendo cambio i dati solo di quello che viene modificato. Ovviamente per evitare problemi di sicurezza ogni dato viene verificato prima di venir inserito nel database.

**public function** delete($id)  
{  
 **if** (validate::*checkInt*($id)) {  
 db\_manager::*execute*(  
 **"DELETE from users where id = "** . $id  
 );  
 }  
 $this->home();  
}

Questo metodo semplicemente verifica l’id e poi elimina l’account corrispondente tramite una query sul database.

### Gestione Database

Per gestire le query sul database ho creato due classi di base database e db\_manager, la classe database si occupa delle connessioni ai database e i dump invece db\_manager si occupa delle query usate spesso e di ritornare un array invece di un oggetto.

**<INSERIRE CODICE D’ESEMPIO database.php>**

Come si può vedere questa classe passati i diversi parametri della connessione al database ritorna la connessione oppure si può creare un file con il dump mysql del database.

**<INSERIRE CODICE D’ESEMPIO db\_manager.php>**

Manca descrizione codice perché non è finito.

### Gestione Validazioni

Tutte le funzioni per controllare gli accessi e le variabili inseriti sono presenti nella classe validate.

*/\*\*  
 \* Questo metodo controlla se l'email passata è valida  
 \** ***@param*** *$email String Email da controllare  
 \** ***@return*** *bool Email valida  
 \*/* **public static function** checkEmail($email){  
 $email = *filter\_var*($email, ***FILTER\_SANITIZE\_EMAIL***);  
 **if**(*filter\_var*($email, ***FILTER\_VALIDATE\_EMAIL***)) {  
 **return true**;  
 }  
 **return false**;  
 }  
  
*/\*\*  
 \* Questo metodo mette in sicurezza la stringa passata  
 \** ***@param*** *$string String Sringa da mettere in sicurezza  
 \** ***@return*** *mixed Stringa sicura  
 \*/* **public static function** secureString($string){  
 **return** *filter\_var*($string, ***FILTER\_SANITIZE\_STRIPPED***);  
 }  
  
*/\*\*  
 \* Questo metodo controlla se il valore passato è un int  
 \** ***@param*** *$int String Valore da controllare  
 \** ***@return*** *bool È un int  
 \*/* **public static function** checkInt($int){  
 **if**(*filter\_var*($int, ***FILTER\_VALIDATE\_INT***)){  
 **return true**;  
 }  
 **return false**;  
 }

Queste tre funzioni servono a controllare i dati inseriti. La prima controlla se il valore inserito è un’email, per verificare nel minimo dettaglio che è un’email usa filter\_var(<valore>, FILTER\_VALIDATE\_EMAIL).

La seconda invece “purifica” una variabile di testo che viene passata e poi la ritorna questo per evitare principalmente Sql Injections.

L’ultima controlla se il dato passato è un numero tramite filter\_var(<valore>, FILTER\_VALIDATE\_INT) infine ritorna true o false.

*/\*\*  
 \* Questa funzione controlla se l'utente possiede un cookie valido  
 \** ***@return*** *bool $  
 \*/***public static function** check(){  
  
 */\*\*  
 \* Controllo se inanzitutto possiede un cookie  
 \*/* **if**(**isset**($\_COOKIE[**'backup\_site'**])) {  
 */\*\*  
 \* Metto in sicurezza la stringa del cookie e controllo se è valido  
 \*/* $backup\_site = validate::*secureString*($\_COOKIE[**'backup\_site'**]);  
 $results = db\_manager::*executeQuery*(**"SELECT** *\** **FROM users where pass ='"**.$backup\_site.**"'"**);  
 **if** ($results !== **NULL**){  
  
 */\*\*  
 \* Controllo se ha un livello di accesso sufficiente per accedere  
 \*/* **if**($results[0][**'access'**] > 0){  
 **return true**;  
 }**else**{  
 $\_POST[**'error'**] = **'account non verificato'**;  
 **return false**;  
 }  
 }**else**{  
 **return false**;  
 }  
 }**else**{  
 **return false**;  
 }  
}

La funzione check controlla ogni parametro del utente che è connesso così da essere sicuri che il suo cookie è valido. Quindi prima di tutto controlla se ha un cookie, se ce l’ha cerca il codice all’interno nel database. Se si trova una corrispondenza si controlla come ultima cosa se l’account è abilitato se anche questo controllo va a buon fine ritorna true.

*/\*\*  
 \* Questo metodo ritorna true se l'utente collegato è un amministratore  
 \** ***@return*** *bool È un amministratore  
 \*/* **public static function** admin(){  
  
 */\*\*  
 \* Controllo se c'è qualcuno collegato  
 \*/* **if**(**isset**($\_COOKIE[**'backup\_site'**])) {  
  
 */\*\*  
 \* Prendo i dati del utente collegato  
 \*/* $results = db\_manager::*executeQuery*(**"SELECT** *\** **FROM users where pass ='"** . $\_COOKIE[**'backup\_site'**] . **"'"**);  
  
 */\*\*  
 \* Se l'utente è presente nel database controlla il suo livello d'accesso  
 \*/* **if** ($results !== **NULL**) {  
 **if** ($results[0][**'access'**] > 1) {  
 **return true**;  
 } **else** {  
 **return false**;  
 }  
 } **else** {  
 **return false**;  
 }  
 }  
 }

Questa funzione usa il cookie per ricavare i dati del utente in particolare il suo livello d’accesso che usa per ritornare se l’utente è amministratore o no. Per fare questo viene cercato l’utente all’interno del database e se viene trovata una corrispondenza viene controllato se è amministratore quindi il suo livello d’accesso è pari a 2.

### Gestione backup

La parte più importante della gestione backup è effettivamente la creazione del backup che viene fatta tramite la funzione startBackup presente nella classe backup.

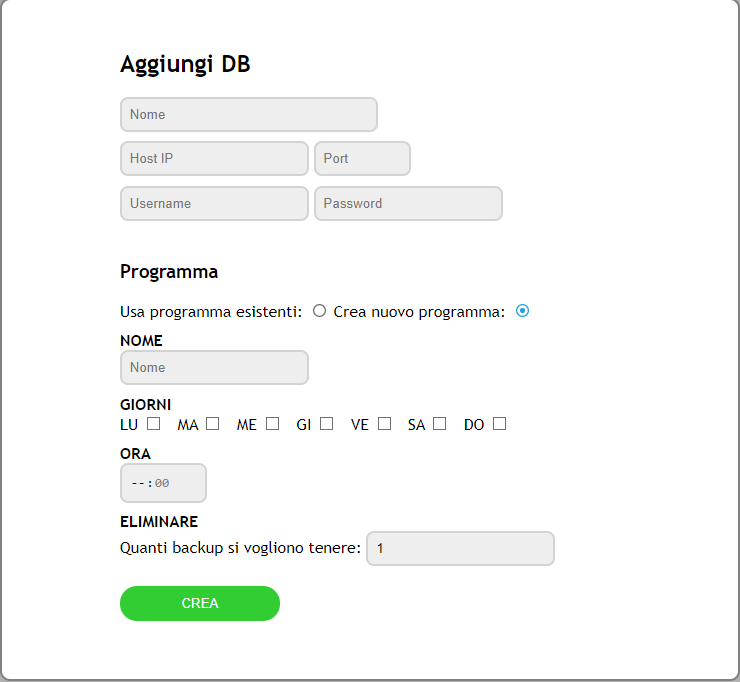
*/\*\*  
 \* Questa funzione crea il backup del database passato.  
 \** ***@param*** *$id int Id del database  
 \*/***public function** startBackup($id)  
{  
  
 */\*\*  
 \* Controllo se l'id è un int.  
 \*/* **if** (validate::*checkInt*($id)) {  
  
 */\*\*  
 \* Prendo le informazioni del database di cui devo fare il backup.  
 \*/* $db\_data = db\_manager::*executeQuery*(**'SELECT** *\** **from database\_data where id = '** . $id);  
 **if** (**isset**($db\_data[0]) || $db\_data[0] != **NULL**) {  
 $db\_data = $db\_data[0];  
  
 */\*\*  
 \* Salvo la data dell'esecuzione.  
 \*/* $now = *date*(**'Y-m-d H:i:s'**);  
 $sql\_file = $db\_data[**'name'**] . **'\_'** . *date*(**"Y-m-d\_H.i.s"**) . **'.sql'**;  
  
 */\*\*  
 \* Controllo se il file esiste già.  
 \*/* **if** (!*file\_exists*(**'./backups/'** . $sql\_file)) {  
  
 */\*\*  
 \* Creo il file sql.  
 \*/* $handle = *fopen*($sql\_file, **'w'**) **or die**(**"Unable to open file!"**);  
 *fclose*($handle);  
 $dir = **'./backups/'** . $sql\_file;  
  
 */\*\*  
 \* Faccio il dump del database  
 \*/* $output = database::*dump*($db\_data[**'ip'**], $db\_data[**'user'**], $db\_data[**'pass'**], $db\_data[**'name'**], $dir);  
 **if** (**isset**($output[1])) {  
  
 */\*\*  
 \* Se va male inserisco il backup come fallito. E aggiungo la notifica.  
 \*/* db\_manager::*execute*(**"Insert into backup (creation\_date,status, db\_id) values('"** . $now . **"','2','')"**);  
 $bc = db\_manager::*executeQuery*(**"Select id from backup where creation\_date ='"** . $now . **"'"**);  
 db\_manager::*execute*(  
 **"INSERT into alerts (content,level,backup\_id) values ('Il backup del database"** . $db\_data[**'name'**] . **" è fallito','1','"** . $bc[0][**'id'**] . **"')"** );  
 } **else** {  
  
 */\*\*  
 \* Se tutto va bene inserisco il backup come successo.  
 \*/* db\_manager::*execute*(**"Insert into backup (creation\_date,status, db\_id) values('"** . $now . **"','1','**{$id}**')"**);  
 }  
 }  
  
 */\*\*  
 \* Richiamo la pagina di backup.  
 \*/* $this->home();  
 }  
 }  
}

In questa funzione faccio 4 controlli principali. Il primo è se mi viene passato un valore intero, in seguito controllo se è l’id di un database e quindi se è valido. Invece per la creazione del backup controllo se un file con lo stesso nome esiste già per evitare duplicati. Infine controllo se il backup è andato bene o se ci sono stati errori, in caso che ci sono stati errori aggiungo una notifica agli utenti così che possono controllare.

### Gestione DB dei siti

La funzione più importante per la gestione dei DB dei siti web esterni è quella per l’aggiunta di un nuovo database. Questa funzione viene richiamata da un form con l’inserimento dei dati principali che ci sono bisogno per connettersi al database e farne il backup. Il form è suddiviso in 2 parti:

* la prima con i dati del database
* la seconda per il programma di backup del database in cui si può scegliere se usare un programma esistente o di crearne uno.

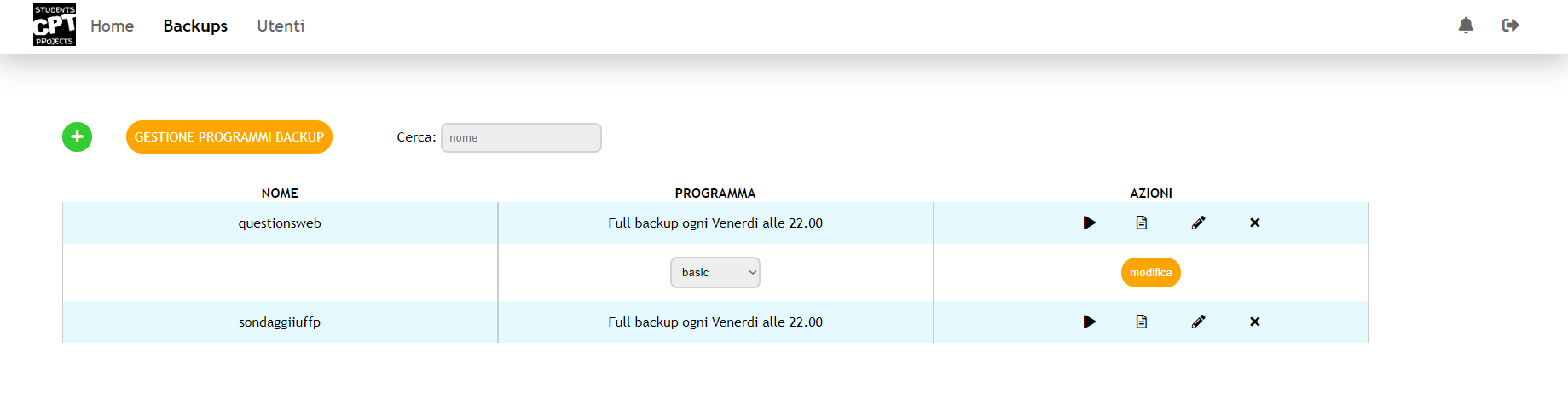


*/\*\*  
 \* Questo metodo aggiunge i dati di connessione di un DB al database  
 \*/***public function** addDB()  
{  
  
 */\*\*  
 \* Controllo se il nome non è vuoto  
 \*/* **if** (validate::*checkNotEmpty*($\_POST[**'sname'**])) {  
  
 */\*\*  
 \* Creo il numero complesso dei giorni e controllo che non sia vuoto  
 \*/* $days = $this->getDays();  
 **if** ($days != **''**) {  
  
 */\*\*  
 \* Controllo se è stato inserito un orario  
 \*/* **if** (validate::*checkNotEmpty*($\_POST[**'sethour'**])) {  
  
 */\*\*  
 \* Trasformo l'ora da una stringa ad un int e controllo che sia effettivamente un intero  
 \*/* $hour = (int)*substr*($\_POST[**'sethour'**], 0, 2);  
 **if** (validate::*checkInt*($hour)) {  
  
 */\*\*  
 \* Controllo che i backup da tenere sia almeno 1 e che sia un numero.  
 \*/* **if** (validate::*checkInt*($\_POST[**'da'**]) && $\_POST[**'da'**] >= 1) {  
  
 */\*\*  
 \* Creo il nuovo programma.  
 \*/* $this->createSettings(validate::*secureString*($\_POST[**'sname'**]), $\_POST[**'da'**], $days, $hour);  
  
 }  
 }  
 }  
 }  
 }  
  
 **if** (!validate::*checkNotEmpty*($\_POST[**'port'**])) {  
 $port = 3306;  
 } **else** {  
  
 $port = $\_POST[**'port'**];  
 }  
 $name = validate::*secureString*($\_POST[**'name'**]);  
 $username = validate::*secureString*(($\_POST[**'username'**]));  
 $password = validate::*secureString*($\_POST[**'password'**]);  
  
 **if** (validate::*checkNotEmpty*($name) && validate::*checkNotEmpty*($username) && *filter\_var*($\_POST[**'ip'**], ***FILTER\_VALIDATE\_IP***) && validate::*checkInt*($port) && $port < 65536) {  
 **if** (db\_manager::*executeQuery*(**"SELECT** *\** **from database\_data where name ='"** . $name . **"'"**) == **NULL**) {  
  
  
 */\*\*  
 \* Inserisco tutti i dati nel database  
 \*/* db\_manager::*execute*(  
 **"INSERT into database\_data (name,ip,port,user,pass,settings\_id) values ('"** . $name . **"','"** . $\_POST[**'ip'**] . **"','"** . $port . **"','"** . $username . **"','"** . $password . **"',1)"** );  
  
  
 } **else** {  
  
 */\*\*  
 \* Ritorno un messaggio di feedback  
 \*/* $\_POST[**'creationError'**] = **'Esiste già un database con questo nome'**;  
 }  
 } **else** {  
 */\*\*  
 \* Ritorno un messaggio di errore  
 \*/* **if**($\_POST[**'createSettings'**] == **'true'**) {  
 $\_POST[**'creationError'**] = **'Uno dei valori inseriti non è valido'**;  
 }  
 }  
  
 $this->home();  
}

MANCA DESCRIZIONE PERCHÉ POTREBBE ANCORA CAMBIARE

*/\*\*  
 \* Questo metodo modifica le impostazioni di un programma  
 \** ***@param*** *$id Int Id del programma da modificare  
 \*/***public function** updateSettings($id){  
 **if**(validate::*checkInt*($id)){  
  
 */\*\*  
 \* Metto in sicurezza il nome prima di inserirlo nella query  
 \*/* $name = validate::*secureString*($\_POST[**'smName'**]);  
 **if** ($name != **NULL**) {  
 db\_manager::*execute*(  
 **"UPDATE settings set name = '"** . $name . **"' where id = "** . $id  
 );  
 }  
  
 */\*\*  
 \* Controllo se è stato selezionato almeno un giorno  
 \*/* $days = $this->getDays();  
 **if**($days != **''**){  
 db\_manager::*execute*(  
 **"UPDATE settings set day = '"** . $days . **"' where id = "** . $id  
 );  
 }  
  
 */\*\*  
 \* Trasformo l'ora da una stringa ad un int e controllo che sia effettivamente un intero  
 \*/* $hour = (int)*substr*($\_POST[**'smhour'**], 0, 2);  
 **if** (validate::*checkInt*($hour)) {  
 db\_manager::*execute*(  
 **"UPDATE settings set hour = '"** . $hour . **"' where id = "** . $id  
 );  
 }  
  
 */\*\*  
 \* Controllo che i backup da tenere sia almeno 1 e che sia un numero.  
 \*/* **if** (validate::*checkInt*($\_POST[**'smda'**]) && $\_POST[**'smda'**] >= 1) {  
 db\_manager::*execute*(  
 **"UPDATE settings set delete\_after = '"** . $\_POST[**'smda'**] . **"' where id = "** . $id  
 );  
 }  
 }  
  
 */\*\*  
 \* Ritorno alla pagina dei database  
 \*/* $this->home();  
}

Questo metodo è suddiviso nei diversi input a cui vengono fatti i controlli e di cui poi viene immesso il valore all’interno del database.

Nel frontend come si vede si può solo modificare il programma assegnato al database.

*/\*\*  
 \* Questa funzione cambia il programma di un database  
 \** ***@param*** *$id Int Id del database da modificare  
 \*/***public function** modify($id){  
  
 */\*\*  
 \* Controllo che è stato passato è un id  
 \*/* **if**(validate::*checkInt*($id)) {  
  
 */\*\*  
 \* Controllo che l'impostazione che è stata passata è un id  
 \*/* **if** (validate::*checkInt*($\_POST[**'dmSetting'**])) {  
  
 */\*\*  
 \* Controllo che ci sia un programma con quel id  
 \*/* $results = db\_manager::*executeQuery*(**"SELECT** *\** **from settings where id ='"** . $\_POST[**'dmSetting'**] . **"'"**);  
 **if**($results !== **NULL**) {  
   
 db\_manager::*execute*(  
 **"UPDATE database\_data set settings\_id = '"** . $\_POST[**'dmSetting'**] . **"' where id = "** . $id  
 );  
   
 }  
 }  
 }  
  
 $this->home();  
}

La funzione come quella di prima è molto semplice e in questa visto che c’è solo un input è semplicemente un’azione sul database, che viene eseguita dopo aver controllato che il valore passato sia valido e che sia un programma esistente.

# Test

## Protocollo di test

ESEMPIO

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-001 sub01 | **Nome:** | Funzionamento bottone “registrati”. |
| **Descrizione:** | Prova del funzionamento corretto del bottone “registrati” che sta al centro della pagina sotto la scritta “Benvenuto!”. | | |
| **Prerequisiti:** | Pagina di benvenuto funzionante.  Bottone “registrati” esistente.  Codice di background del bottone registrati che ci porta alla pagina di registrazione. | | |
| **Procedura:** | 1. Andare sulla pagina di benvenuto 2. Cliccare il bottone “registrati” | | |
| **Risultati attesi:** | Al click del bottone veniamo portati alla pagina “Registrazioni.html”. | | |

## Risultati test

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## Mancanze/limitazioni conosciute

# Consuntivo

# Conclusioni

## Sviluppi futuri

## Considerazioni personali

# Bibliografia

## Sitografia

# Allegati

* Diari di lavoro
* Codici sorgente
* Quaderno dei compiti
* Prodotto