Inventario Hardware

+1 Introduzione 3

1.1 Informazioni sul progetto 3

1.2 Abstract 3

1.3 Scopo 3

2 Analisi 4

2.1 Analisi del dominio 4

2.2 Analisi e specifica dei requisiti 4

2.3 Use case 7

2.4 Pianificazione 8

2.5 Analisi dei mezzi 10

2.5.1 Software 10

2.5.2 Hardware 10

3 Progettazione 11

3.1 Design dell’architettura del sistema 11

3.1.1 Model View Controller 11

3.1.2 Struttura delle cartelle 12

3.2 Design dei dati e database 13

3.2.1 Diagramma UML 13

3.2.2 Descrizione diagramma UML 14

3.3 Design delle interfacce 15

3.3.1 Home 15

3.3.2 Gestione utenti 16

3.3.3 Lista inventario 17

3.3.4 Aggiunta componente 18

3.3.5 Impostazione della soglia 19

4 Implementazione 20

4.1 Database 20

4.1.1 Implementazione database 20

4.1.2 Popolazione database 22

4.1.3 Account di accesso 22

4.2 Applicativo Web 23

4.2.1 Struttura 23

4.2.2 Config 24

4.2.3 Controller 24

4.2.4 Libs 32

4.2.5 Models 34

5 Test 42

5.1 Protocollo di test 42

5.2 Risultati test 46

5.3 Mancanze/limitazioni conosciute 54

6 Consuntivo 55

7 Conclusioni 56

7.1 Sviluppi futuri 56

7.2 Considerazioni personali 56

8 Glossario 57

9 Sitografia 58

10 Allegati 59

# Introduzione

## Informazioni sul progetto

**Titolo**: Gestione inventario Hardware

**Allievo** coinvolto nel progetto: Andrea Curti, [andrea.curti@samtrevano.ch](mailto:andrea.curti@samtrevano.ch)

**Classe**: I3AC Scuola Arti e Mestieri Trevano, Informatica

**Formatore**: Guido Montalbetti, [guido.montalbetti@samtrevano.ch](mailto:guido.montalbetti@samtrevano.ch)

**Data** **inizio**: 16.09.2021

**Data** **fine**: 23.12.2021

## Abstract

Da ormai un po’ di tempo, i computer sono entrati nella nostra vita quotidiana, non solo in quella privata, ma anche nella vita aziendale e scolastica.

Per l’utilizzo di questa tecnologia, però, c’è bisogno di computer, insieme a tutti i componenti complementari, come tastiere, mouse, schermi, cavi…

In grandi realtà come le scuole, la gestione di questi oggetti risulta complicata se in numeri elevati; per facilitare, è stata sviluppata la seguente applicazione, Gestione inventario Hardware, che permette il controllo di ogni singolo componente, partendo dalle sue caratteristiche tecniche, all’aula d’impiego.

## Scopo

Lo scopo del progetto “Gestione inventario Hardware” è quello di semplificare e velocizzare proprio la gestione dei componenti hardware all’interno della Scuola d’Arti e Mestieri di Trevano.

L’applicativo permette di conoscere velocemente le informazioni di ogni componente della scuola, come per esempio l’aula in cui si trova al momento, per avere sempre il controllo ed evitare il rischio di confusione e perdere componenti all’interno di un edificio scolastico con più di 100 aule.

# Analisi

## Analisi del dominio

L’applicativo verrà utilizzato principalmente dai sistemisti della scuola.

Serve per la gestione di computer, tastiere, mouse, monitor, lavagne multimediali e tutti gli altri componenti informatici che la scuola dispone. In questo modo semplifica il lavoro permettendo ai sistemisti di cercare e trovare l’aula d’impiego di un componente, così da risparmiare tempo. L’applicativo è semplice da utilizzare, nella schermata principale ci sono le varie categorie di componenti e la possibilità di filtrarli.

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| ID: REQ-001 | |
| Nome | Accesso |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Solo gli amministratori e gli utenti autorizzati possono accedere. |
| Sotto requisiti | |
| 001 | Interagire con il database per consultare i dati di accesso |
| 002 | Distinguere le diverse tipologie di utente |

|  |  |
| --- | --- |
| ID: REQ-002 | |
| Nome | Visualizzazione elenco componenti |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Si devono vedere tutti i componenti con le relative informazioni |
| Sotto requisiti | |
| 001 | Creare le varie categorie dei componenti |
| 002 | Popolare il database con i componenti |

|  |  |
| --- | --- |
| ID: REQ-003 | |
| Nome | Aggiungere un componente |
| Priorità | 1 |
| Versione | 1.0 |
| Note | L’utente può aggiungere un nuovo componente all’inventario |
| Sotto requisiti | |
| 001 | Accedere con un utente |
| 002 | Aggiornare i dati nel database |
| 003 | Impostare la categoria e caratteristiche del componente |

|  |  |
| --- | --- |
| ID: REQ-004 | |
| Nome | Eliminare un componente |
| Priorità | 1 |
| Versione | 1.0 |
| Note | L’utente può eliminare un componente dall’inventario |
| Sotto requisiti | |
| 001 | Accedere con un utente |
| 002 | Aggiornare i dati nel database |

|  |  |
| --- | --- |
| ID: REQ-005 | |
| Nome | Modificare un componente |
| Priorità | 1 |
| Versione | 1.0 |
| Note | L’amministratore può eliminare un componente dall’inventario |
| Sotto requisiti | |
| 001 | Accedere con un utente amministratore |
| 002 | Aggiornare i dati nel database |

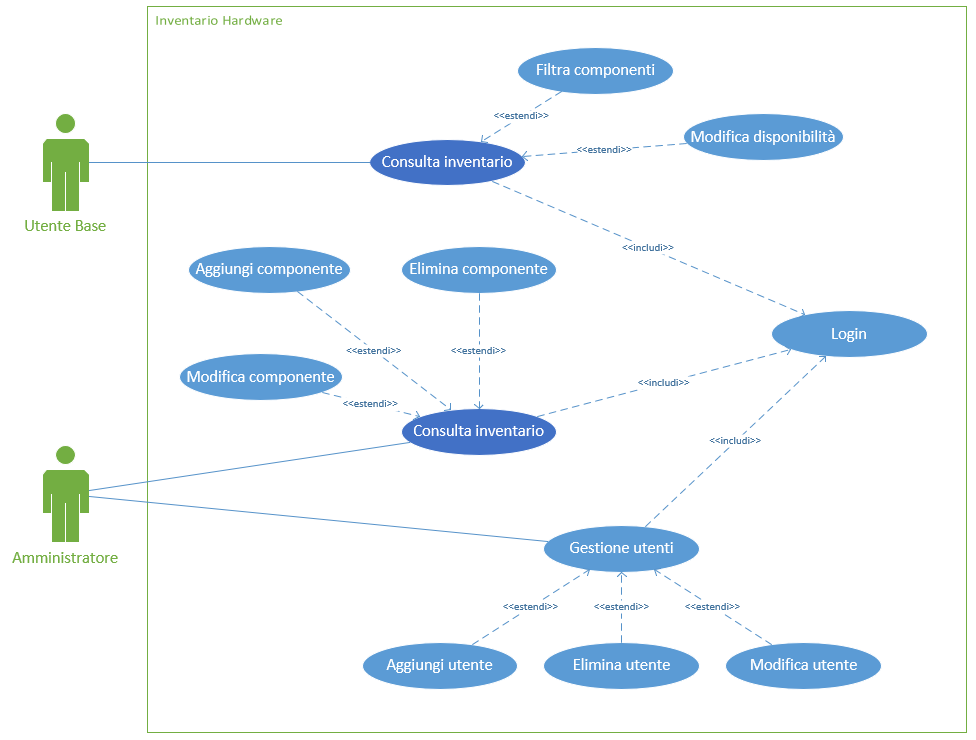
|  |  |
| --- | --- |
| ID: REQ-006 | |
| Nome | Aggiungere un utente |
| Priorità | 1 |
| Versione | 1.0 |
| Note | L’amministratore può aggiungere un utente |
| Sotto requisiti | |
| 001 | Accedere con un utente amministratore |
| 002 | Inserire le informazioni del nuovo utente nel database |

|  |  |
| --- | --- |
| ID: REQ-007 | |
| Nome | Eliminare un utente |
| Priorità | 1 |
| Versione | 1.0 |
| Note | L’amministratore può eliminare un utente |
| Sotto requisiti | |
| 001 | Accedere con un utente amministratore |
| 002 | Aggiornare i dati dell’utente nel database |

|  |  |
| --- | --- |
| ID: REQ-008 | |
| Nome | Modificare un utente |
| Priorità | 1 |
| Versione | 1.0 |
| Note | L’amministratore può modificare le credenziali di un utente |
| Sotto requisiti | |
| 001 | Accedere con un utente amministratore |
| 002 | Inserire le nuove informazioni dell’utente |
| 003 | Aggiornare i dati nel database |

|  |  |
| --- | --- |
| ID: REQ-009 | |
| Nome | Percentuale soglia componenti liberi |
| Priorità | 1 |
| Versione | 1.0 |
| Note | La percentuale di componenti liberi di una categoria non deve scendere sotto una soglia, altrimenti lo notifica |
| Sotto requisiti | |
| 001 | Calcolo del totale dei componenti di una categoria |

## Use case



L’applicativo potrà essere utilizzato da 2 tipi diversi di utente:

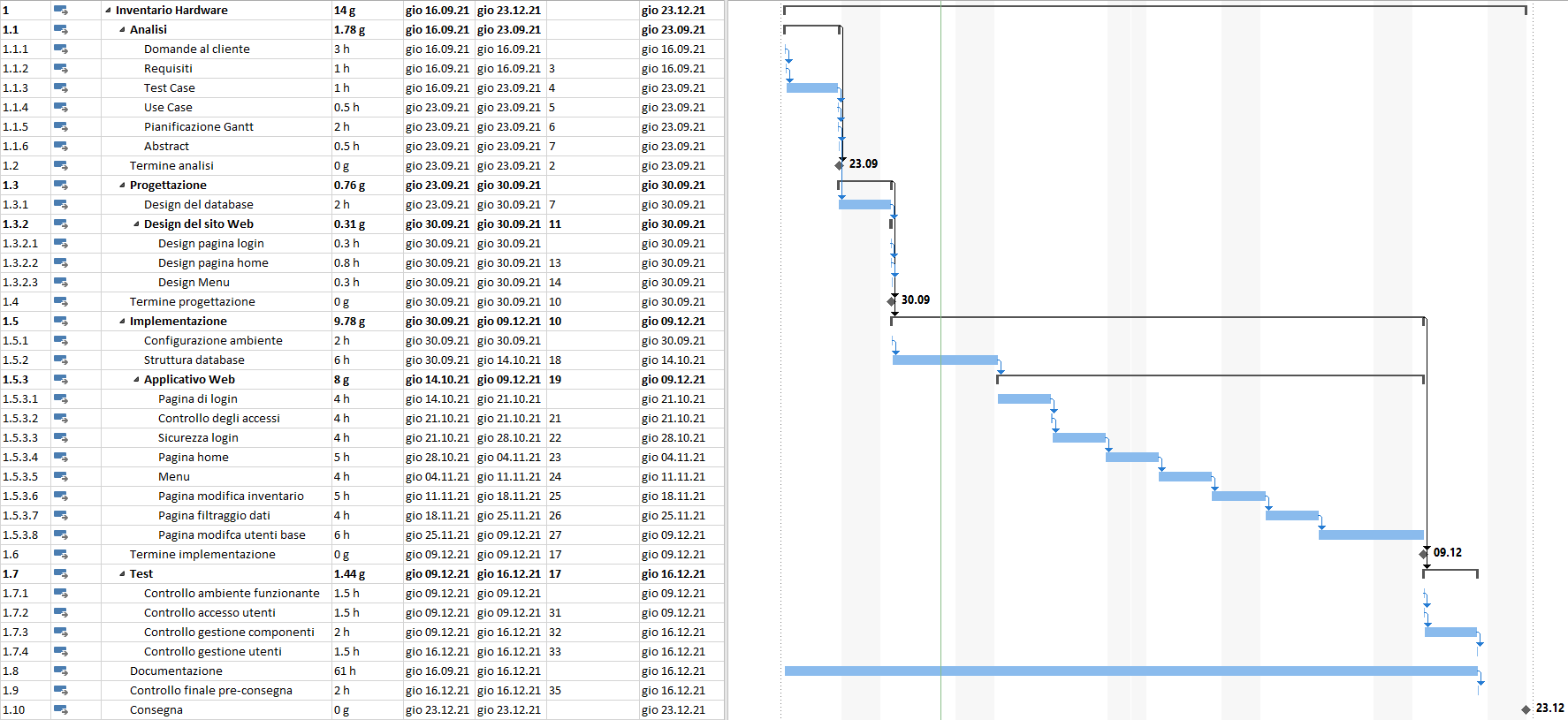
* **Utente base:** L’unica interazione che gli utenti base avranno è consultare l’inventario con le varie funzioni, ovvero il filtraggio dei componenti in base ad un criterio dato, e la modifica delle disponibilità e aula di un componente.
* **Amministratore:** Questo utente, invece, ha il controllo completo sull’inventario, ovvero oltre a consultarlo, ha i permessi per aggiungere, eliminare o modificare un componente dalla lista. Un’altra funzione di cui dispone è la gestione degli utenti base, infatti può aggiungere, eliminare o modificare un utente.

## Pianificazione

Il progetto è stato pianificato con un tempo di durata di una settimana inferiore al termine di consegna, così da lasciare un margine di errore in caso che una delle attività duri più del previsto, a causa di qualche imprevisto. Così facendo sarà più difficile superare il termine della consegna.

Il Gantt ha un totale di 34 righe e contiene varie categorie di attività:

1. **Analisi:** Qua sono raggruppate tutte le attività riguardante l’analisi del progetto necessaria per facilitare in seguito la progettazione, l’implementazione e i test.
2. **Progettazione:** Questa categoria contiene la progettazione del database e del sito Web, è fondamentale per la buona riuscita del progetto, perché permette di arrivare all’implementazione preparato e sapere già come sviluppare l’applicativo.
   1. **Design del sito Web:** Sono raggruppate le progettazioni della della pagina login, home e dei menu.
3. **Implementazione:** In questa categoria inizia il lavoro pratico, ovvero lo sviluppo della struttura del database e del sito Web attraverso la scrittura del codice.
   1. **Applicativo Web:** Con l’utilizzo della progettazione, si sviluppa l’interfaccia grafica Web dell’applicativo.
4. **Test:** Una delle parti finali del progetto, ma non per importanza, infatti serve per il controllo dell’applicativo e delle funzioni implementate, così da consegnare al richiedente un prodotto che rispetti e implementi le sue richieste.



## Analisi dei mezzi

### Software

I software utilizzati per la realizzazione del progetto sono:

* HTML 5
* CSS 3
* Javascript 1.8.5
* PHP 7.4.12
* Sublime Text 3.2.2
* MySQL
* Apache 2.4

Le libre utilizzate:

* Bootstrap 4.3.1 (<https://getbootstrap.com/>)

### Hardware

Il progetto è stato sviluppato su un computer fisso le cui specifiche sono:

* 16 GB di RAM
* Intel Xeon E3-1240

L’applicativo potrà essere utilizzato su macchine anche con basse potenze di calcolo, è consigliato almeno:

* 512 MB di RAM
* 2 GB di spazio sul disco

# Progettazione

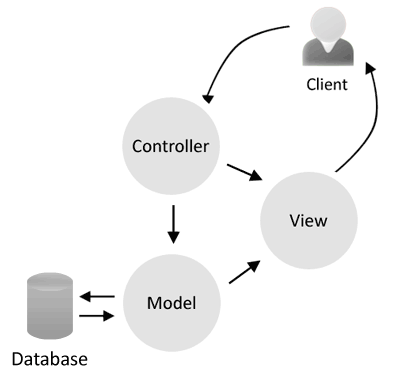
## Design dell’architettura del sistema

### Model View Controller

Model-View-Controller (MVC) è un pattern utilizzato in programmazione per dividere il codice in blocchi dalle funzionalità ben distinte. Sono divise in 3 categorie:

* **Model**: sono le classi che si occupano di dialogare con il database per l’accesso ai dati.
* **View**: sono le interfacce grafiche che vengono mostrate all’utente e gestiscono le interazioni fra quest’ultimo e i controller.
* **Controller**: sono le classi che ricevono i comandi dell’utente attraverso le View, eseguendo delle operazioni con l’utilizzo dei Model e generalmente portano al cambiamento di stato della View

riceve i comandi dell’utente attraverso il View e reagisce eseguendo delle operazioni che possono interessare il Model e che portano generalmente ad un cambiamento di stato del View.



### Struttura delle cartelle

Immagine che contiene tavolo

Descrizione generata automaticamente

La struttura delle cartelle si divide nel seguente modo:

**config**: verranno inseriti i file di configurazione che contengono importanti operazioni per il funzionamento dell’applicativo.

**controller**: qui ci saranno i controller che uniscono le pagine della cartella “views” con i models.

**libs**: tutte le librerie o classi di aiuto per effettuare operazioni o calcoli ripetuti in più models.

**log**: in questa cartella ci saranno i file log di ogni evento o errore generato dall’applicativo.

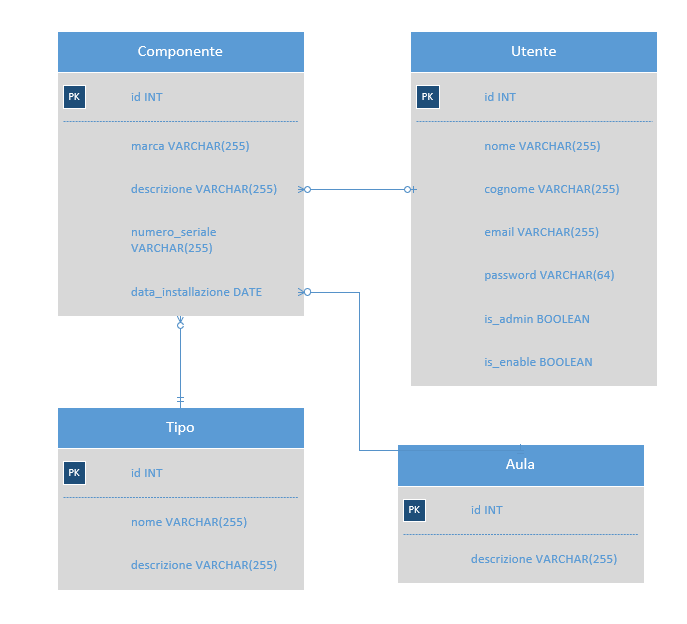
**models**: si trovano tutti i models che servono per effettuare operazioni e modifiche sul database.

**public**: verranno inseriti file .css per lo stile del sito web, creati da me o librerie come Bootstrap.

**views**:tutte le pagine HTML del sito Web saranno in questa cartella.

## Design dei dati e database

### Diagramma UML



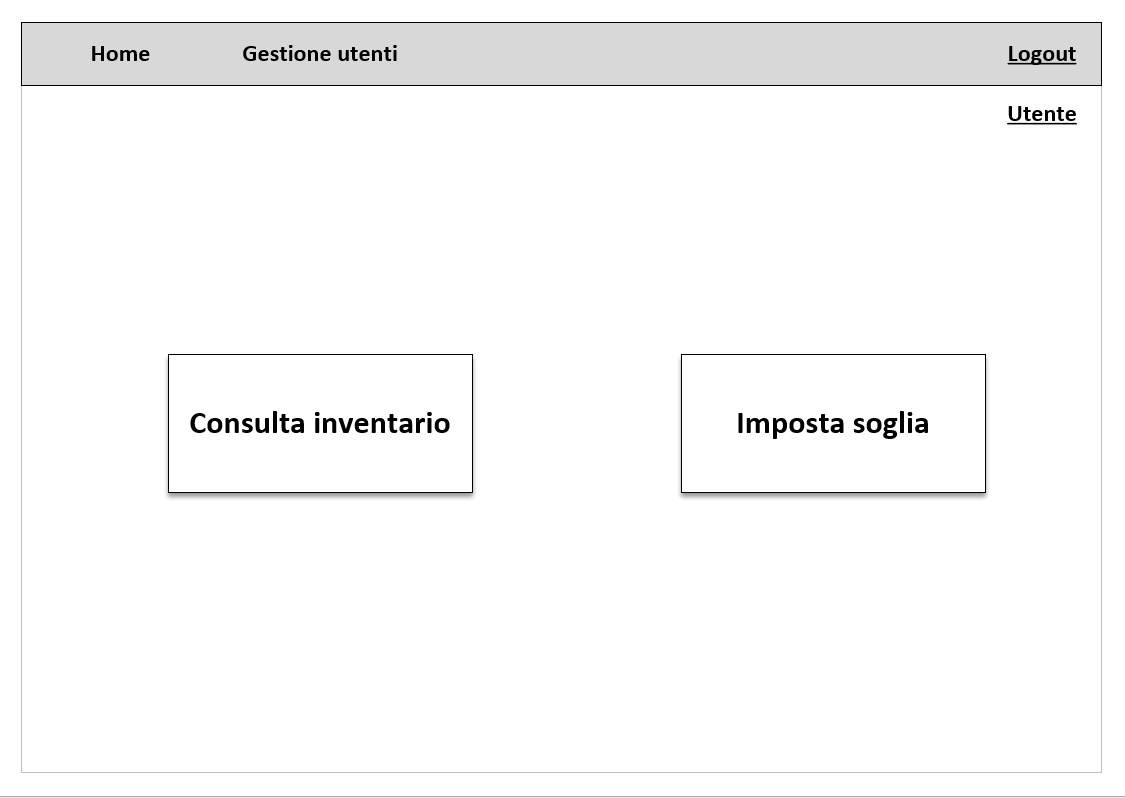
### Descrizione diagramma UML

La struttura del database si basa su 4 tabelle principali, queste sono tutte quelle necessarie all’utilizzo del software come richiesto sul QDC. Il database contiene le seguenti tabelle:

1. **Utente:** in questa tabella sono salvate tutte le informazioni sugli utenti, ovvero nome, cognome, l’email, la password criptata in sha256 e is\_admin, un booleano che permette di capire chi è amministratore del sistema. Come ultimo attributo c’è “is\_enable”, serve per capire se un utente è stato eliminato, ovvero archiviato, oppure no.
2. **Componente:** qua vengono salvate tutte le informazioni di ogni componente, cioè la marca, una breve descrizione di esso, il numero seriale per identificarlo, la data dell’installazione, l’aula in cui si trova e la tipologia del componente, esempio tastiera, mouse, monitor e altro.
3. **Tipo:** questa tabella contiene il nome del tipo di componente e una breve descrizione della tipologia.
4. **Aula:** vengono salvate le informazioni di tutte le aule della scuola, infatti l’id corrisponde al numero dell’aula, in più ha anche una breve descrizione per sapere di che tipo di aula si tratti.

## Design delle interfacce

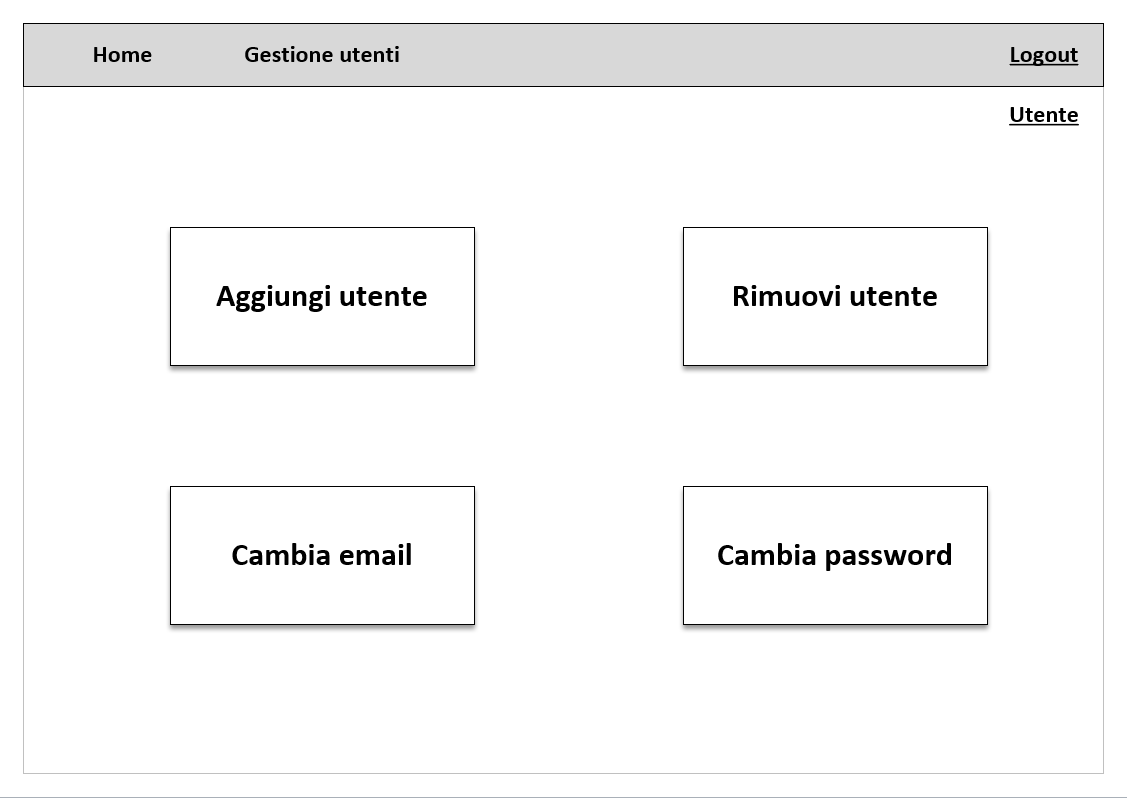
### Home



Questa è l’interfaccia home per gli amministratori, che si ottiene subito dopo aver effettuato il login con un account, appunto, amministratore. La versione dell’utente base sarà quasi identica a questa, a parte nella barra del menù in alto dove avrà solo la possibilità di andare alla home e fare il log out, senza l’opzione per la gestione degli utenti.

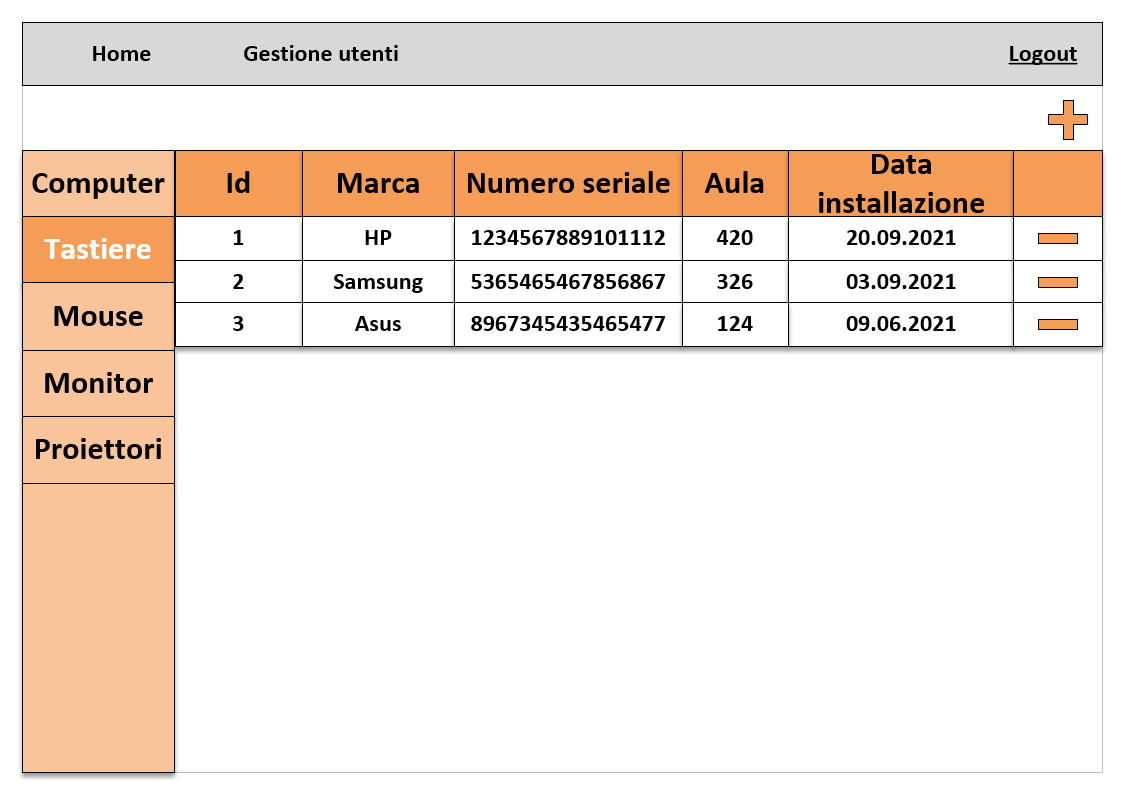
Naturalmente la scritta “Utente” in alto a destra verrà modificata con l’email dell’utente che ha effettuato l’accesso.

### Gestione utenti



In questa pagina, visualizzabile solo dagli amministratori, ci sono varie opzioni per la gestione degli utenti: si può aggiungere o rimuovere un utente, modificare l’email o la password.

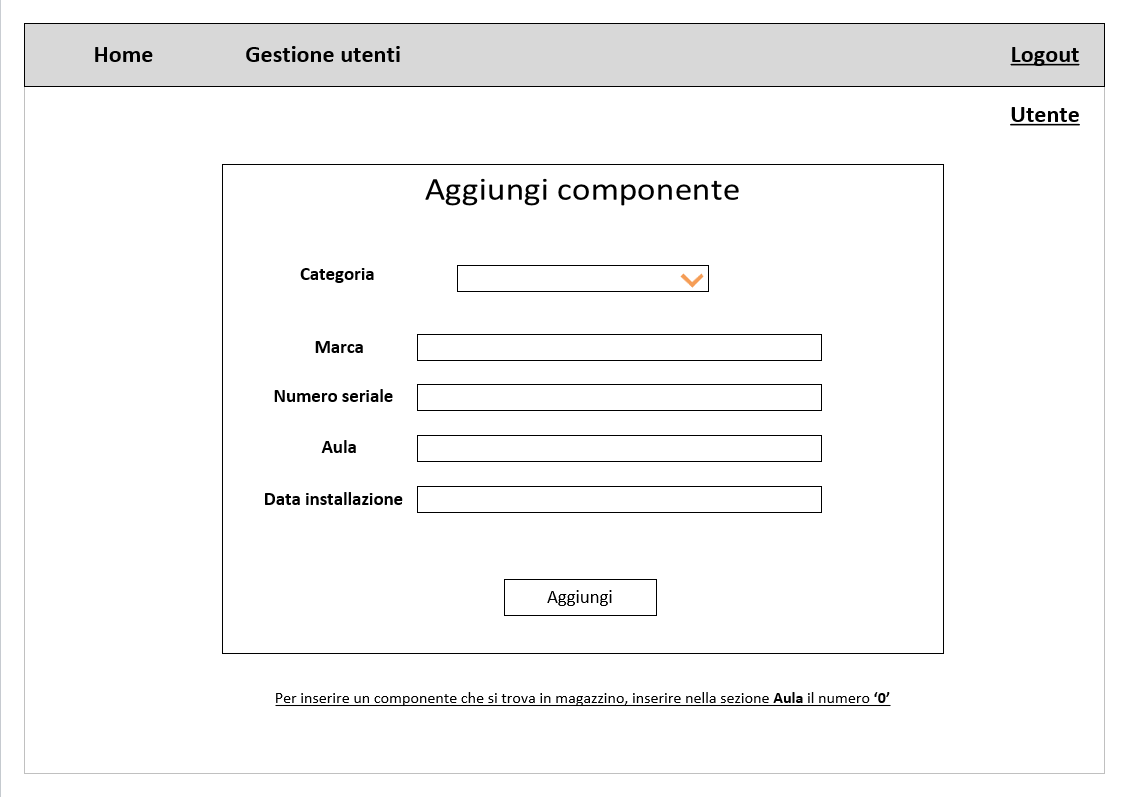
### Lista inventario



Questa pagina è visualizzabile sia dagli utenti base, che quello amministratori, e si ottiene una volta cliccato sul bottone “Consulta inventario” dalla pagina home. A sinistra si possono scegliere le varie categorie, e cliccando su una di esse, in centro comparirà la lista dei componenti di quella categoria con le sue relative informazioni. In alto a destra sotto il logout c’è 1 bottone: il segno più, che serve per aggiungere un nuovo componente. L’ultima colonna della lista contiene il un bottone per eliminare il componente di quella riga.

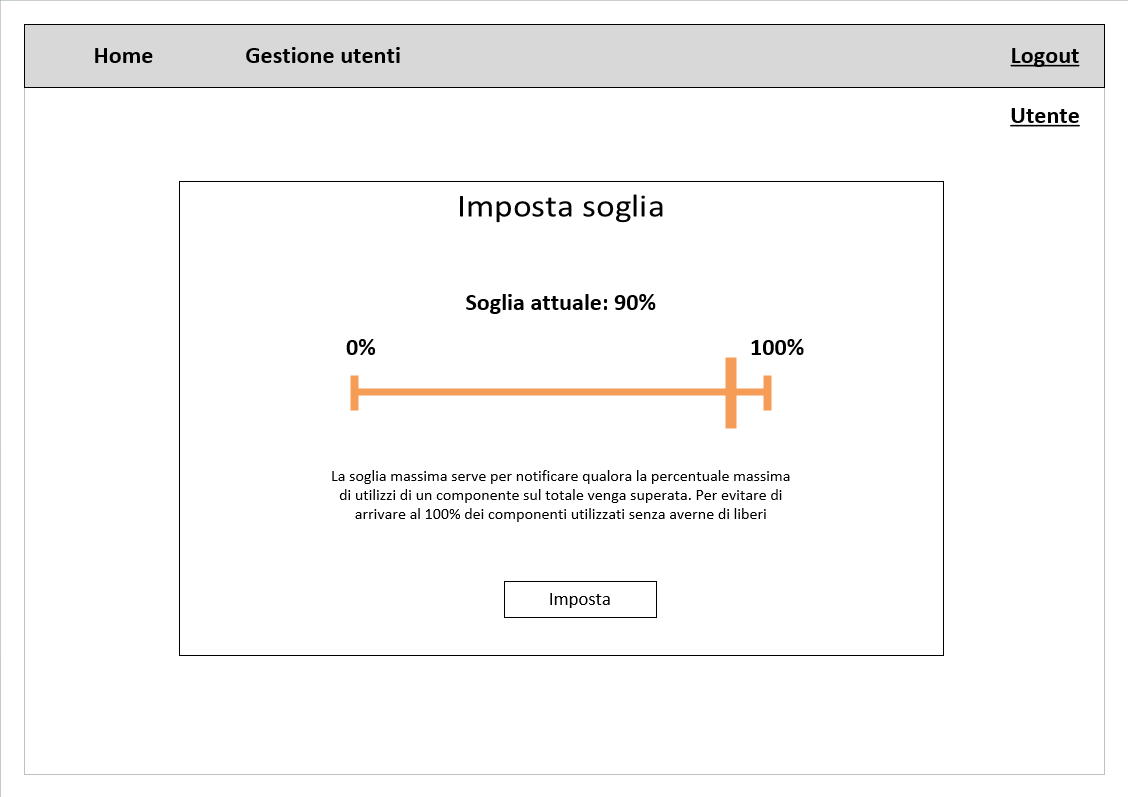
In alto, oltre alla barra per la gestione degli utenti, c’è il bottone a forma di casa per tornare alla home.

### Aggiunta componente



Questa pagina è raggiungibile da tutti gli utenti tramite il bottone a forma di “+” nella pagina della lista dei componenti. Per aggiungere un nuovo componente bisogna completare tutti i campi presenti: selezionare una delle categorie presenti nel menù a tendina, poi inserire la marca, numero seriale (che dovrà essere univoco), l’aula in cui si trova il componente e la data d’installazione, ovvero da quando si trova in quell’aula.

### Impostazione della soglia



Questa pagina è sempre visualizzabile da tutti gli utenti, e serve per impostare la soglia massima di componenti utilizzati prima che venga notificato. Questo è utile per informare gli utenti che una categoria ha quasi terminato i componenti “in magazzino”, così da avere sempre delle scorte di riserva in magazzino.

# Implementazione

## Database

### Implementazione database

|  |
| --- |
| CREATE TABLE utente(  id INT PRIMARY KEY AUTO\_INCREMENT,  nome VARCHAR(255),  cognome VARCHAR(255),  email VARCHAR(255),  password VARCHAR(64),  is\_admin BOOLEAN,  is\_enable BOOLEAN); |

Questo è il codice MySQL per aggiungere la tabella utente, che è identica a quella della progettazione. Molto importanti sono gli ultimi due attributi: is\_admin serve per identificare se l’utente è un amministratore oppure un utente base, per avere più o meno autorizzazioni sul sito.

Invece is\_enable serve per sapere se l’utente è attivo oppure è stato eliminato, ovvero archiviato, così da avere ancora i suoi riferimenti in altre tabelle.

|  |
| --- |
| CREATE TABLE aula(  id INT PRIMARY KEY,  descrizione VARCHAR(255)); |

Qua viene creata la tabella per contenere le aule con il suo id univoco e una breve descrizione del tipo di aula.

|  |
| --- |
| CREATE TABLE tipo(  id INT PRIMARY KEY AUTO\_INCREMENT,  nome VARCHAR(255),  descrizione VARCHAR(255)); |

La tabella tipo contiene l’id di ogni categoria di componente, il nome della categoria e una breve descrizione di essa. Il sito web è progettato per avere 6 categorie diverse: computer, tastiera, mouse, monitor, proiettore e archiviato, ovvero tutti i componenti eliminati

|  |
| --- |
| CREATE TABLE componente(  id INT PRIMARY KEY AUTO\_INCREMENT,  marca VARCHAR(255),  descrizione VARCHAR(255),  numero\_seriale VARCHAR(255),  data\_installazione DATE,  utente\_id INT,  tipo\_id INT,  aula\_id INT,  is\_enable BOOLEAN,  FOREIGN KEY (utente\_id) REFERENCES utente(id),  FOREIGN KEY (tipo\_id) REFERENCES tipo(id),  FOREIGN KEY (aula\_id) REFERENCES aula(id)); |

La tabella componente fa da ponte a tutte le altre 3 tabelle, infatti contiene le chiavi esterne della tabella utente, per sapere che utente ha spostato per ultimo il componente, la chiave di tipo per riconoscere di che categoria è il componente e infine la chiave per la tabella aula, per conoscere in che aula si trova il componente.

### Popolazione database

|  |
| --- |
| INSERT INTO utente(nome, cognome, email, password, is\_admin, is\_enable) VALUES('Andrea','Curti','andrea.curti@samtrevano.ch',  'f526440862c83b4fbe550f700e1cd11f38a09b1dabf62401f6491e04b3c76460', TRUE, TRUE);  INSERT INTO tipo(nome, descrizione) VALUES('Computer','PC and Mac');  INSERT INTO tipo(nome, descrizione) VALUES('Tastiera','Tastiere per Windows e Mac');  INSERT INTO tipo(nome, descrizione) VALUES('Mouse','Mouse per Windows e Mac');  INSERT INTO tipo(nome, descrizione) VALUES('Monitor','Monitor di varie marche');  INSERT INTO tipo(nome, descrizione) VALUES('Proiettore','Proiettore DLP o LCD');  INSERT INTO aula(id, descrizione) VALUES(1,'Magazzino');  INSERT INTO aula(id, descrizione) VALUES(420,'Aula informatica');  INSERT INTO aula(id, descrizione) VALUES(428,'Aula informatica');  INSERT INTO aula(id, descrizione) VALUES(327,'Aula maturita');  INSERT INTO componente(marca, descrizione, numero\_seriale, data\_installazione,  utente\_id, tipo\_id, aula\_id, is\_enable) VALUES('HP','PC da 8GB Ram con i5', 'CZC6518DJ9', CURDATE(), 1, 1, 1, TRUE);  INSERT INTO componente(marca, descrizione, numero\_seriale, data\_installazione,  utente\_id, tipo\_id, aula\_id, is\_enable) VALUES('Samsung','PC da 16GB Ram con i7', 'DZF7431GL8', CURDATE(), 1, 1, 1, TRUE);  INSERT INTO componente(marca, descrizione, numero\_seriale, data\_installazione,  utente\_id, tipo\_id, aula\_id, is\_enable) VALUES('MAC','PC da 4GB Ram con i3', 'GHC1819LQ6', CURDATE(), 1, 1, 1, TRUE);  INSERT INTO componente(marca, descrizione, numero\_seriale, data\_installazione,  utente\_id, tipo\_id, aula\_id, is\_enable) VALUES('HP','Tastiera', 'WSAQ722MC9', CURDATE(), 1, 2, 1, TRUE);  INSERT INTO componente(marca, descrizione, numero\_seriale, data\_installazione,  utente\_id, tipo\_id, aula\_id, is\_enable) VALUES('HP','Tastiera', 'FSGA013BT7', CURDATE(), 1, 2, 1, FALSE);  INSERT INTO componente(marca, descrizione, numero\_seriale, data\_installazione,  utente\_id, tipo\_id, aula\_id, is\_enable) VALUES('HP','Mouse belo', 'SKJA0159E8', CURDATE(), 1, 3, 1, FALSE); |

Vengono inseriti anche dei dati di default per facilitare l’utilizzo iniziale dell’applicativo, con un utente amministratore già impostato, aule, tipi e componenti.

### Account di accesso

Il database ha un solo utente, ovvero un amministratore che ha il controllo completo sul database, quindi modificare la struttura e i dati.

## Applicativo Web

### Struttura

Immagine che contiene tavolo

Descrizione generata automaticamente

La struttura è identica a quella della progettazione. E le cartelle contengono rispettivamente:

**config**: questa cartella contiene il file “config.php”, per configurazioni necessarie, come trovare l’indirizzo host del sito, oppure costanti che vengono utilizzate in tutto il progetto.

**controller**: qui ci sono tutti i controller per le pagine del sito internet. Ognuno di questi controller utilizza un model per effettuare calcoli o modifiche sui dati.

**libs**: vengono raccolte delle classi o librerie utilizzate dai model per effettuare calcoli o operazioni ripetute. Esempio c’è la classe “Hash.php” che serve per calcolare l’hash in sha256 di una password con una salt.

**log**: in questa cartella ci sono tutti i log delle modifiche o errori registrati durante l’utilizzo dell’applicativo.

**models**: si trovano tutti i models richiamati dai controller per svolgere varie funzioni, come l’aggiunta di un componente o utente. Le classi in questa cartella lavorano direttamente sui dati del database.

**public**: in questa cartella sono contenuti 1 file “default.css” e una sottocartella “Bootstrap”. Il file contiene delle configurazioni per lo stile css del sito. La cartella Boostrap contiene a sua volta altre 3 cartelle:

* **css:** ci sono le classi di bootstrap con tutti gli stili css
* **icons:** si trovano le icone per i bottoni offerte sempre da Bootstrap
* **js:** questa cartella contiene i file javascript di Bootstrap.

**views**:tutte le pagine contenenti del codice HTML sono raggruppate in questa cartella. Ci sono tutte le pagine del sito web

### Config

#### config.php

|  |
| --- |
| $actual\_link = (isset($\_SERVER['HTTPS']) && $\_SERVER['HTTPS'] === 'on' ? "https" : "http") . "://$\_SERVER[HTTP\_HOST]";  $documentRoot = $\_SERVER['DOCUMENT\_ROOT'];  $dir = str\_replace('\\','/',getcwd().'/');  $final = $actual\_link.str\_replace($documentRoot,'',$dir);  define('URL', $final); |

Questo codice serve per trovare il percorso host del sito, così da poter salvare il tutto in una costante “URL” e utilizzarla in tutte le classi per caricare o richiamarne un’altra.

Come prima cosa viene controllato se il controllo è https o http, poi viene tornata una stringa $final che contiene il protocollo dell’applicativo con la directory del progetto.

Il codice è stato scritto da Massimo Sartori.

### Controller

#### AddComponent.php

|  |
| --- |
| public function index($error = "", $marca = "", $desc = "", $nSeriale = "", $aula = ""){      if(!empty($\_SESSION['id'])){        require\_once 'application/models/component\_model.php';        $this->view->categories = Component::getCategories();        $this->view->errorMessage = $error;        $this->view->marca = $marca;        $this->view->desc = $desc;        $this->view->nSeriale = $nSeriale;        $this->view->aula = $aula;        $this->view->render('addComponent/index.php');      }else{        $this->view->render('Login/index.php');      }    } |

Il metodo index() serve per caricare la pagina per l’aggiunta di un componente. I parametri, in caso non venissero passati sono di default vuoti. Viene controllato che l’utente sia loggato, poi viene caricata la pagina con i rispettivi parametri da inserire nei vari input. Questo permette che, in caso di errore durante l’aggiunta di un componente, la pagina venga ricaricata con tutte le informazioni inserite precedentemente, così da non svuotare i vari input e obbligare l’utente a reinserire i dati.

|  |
| --- |
| public function add(){      require\_once 'application/models/component\_model.php';      if($\_SERVER["REQUEST\_METHOD"] == "POST"){        try{          if(Component::addComponent($\_POST["marca"], $\_POST["desc"],          $\_POST["nSeriale"], $\_POST["aula"], $\_POST["categoria"])){            header('Location: '.URL."ListComponents/index/".$\_SESSION["idCategoria"]);            $this->writeLog("(AddComponent) New component added");          }        }catch(Exception $e){          $this->writeErrorLog("(AddComponent) Error while adding component");          $this->index($e->getMessage(), $\_POST["marca"], $\_POST["desc"],          $\_POST["nSeriale"], $\_POST["aula"]);        }      }    } |

In caso che venga effettuata una richiesta, viene richiamato il metodo “addComponent” del model “component\_model”, a cui vengono passati i parametri necessari per l’aggiunta di un nuovo componente al database. Se il componente viene aggiunto, allora richiama il metodo “index” del controller “ListComponents” per caricare la pagina della lista dei componenti, altrimenti se c’è un errore, viene richiamato il metodo index e come parametri vengono passate le informazioni inserire dall’utente negli input.

#### AddUser.php

|  |
| --- |
| public function index($message = ""){          if(!empty($\_SESSION['id'])){              if($\_SESSION['isAdmin'] == 1){                  $this->view->errorMessage = $message;                  $this->view->render('addUser/index.php');              }else{                  $this->view->render('Home/index.php');              }          }else{              $this->view->render('Login/index.php');          }      } |

Il metodo index() serve per caricare la pagina per l’aggiunta di un nuovo utente. Vengono fatti i controlli per verificare che l’utente sia loggato e abbia i permessi d’amministratore.

|  |
| --- |
| public function registerUser(){          require\_once 'application/models/addUser\_model.php';          try{              $user = new AddUserClass($\_POST["nome"],                      $\_POST["cognome"], $\_POST["email"],                      $\_POST["password"], $\_POST["confPassword"],                      isset($\_POST["isAdmin"]) ? 1 : 0);              $user->createUser();              $this->writeLog("(AddUser) New user added");              $this->view->render('Home/index.php');          }catch(Exception $e){              $this->writeErrorLog("(AddUser) Error while adding user");              $this->index($e->getMessage());          }  } |

Questa funzione crea un nuovo oggetto AddUserClass richiamando il model adatto. Se il model solleva degli errori, vengono catturati in questa funzione, ed in seguito viene ricaricata la pagina con la descrizione dell’errore. In caso invece vada tutto liscio, viene aggiunto l’utente al database.

#### ChangeEmail.php

|  |
| --- |
| public function index($message = ""){      if(!empty($\_SESSION['id'])){        if($\_SESSION['isAdmin'] == 1){          $this->view->errorMessage = $message;          $this->view->render('ChangeEmail/index.php');        }else{          $this->view->render('Home/index.php');        }      }else{        $this->view->render('Login/index.php');      }    } |

Il metodo index() carica la pagina HTML per la modifica dell’email. Ovviamente dopo aver verificato che l’utente sia loggato e sia un amministratore.

|  |
| --- |
| public function change(){      require\_once 'application/models/changeEmail\_model.php';      try{        if($\_SERVER["REQUEST\_METHOD"] == "POST"){          $user = new ChangeEmailClass($\_POST["oldEmail"],          $\_POST["newEmail"], $\_POST["confEmail"], $\_POST["confPass"]);          $user->changeEmail();          $this->writeLog("(ChangeEmail) Email changed");          $this->view->render('Home/index.php');        }      }catch(Exception $e){        $this->writeErrorLog("(ChangeEmail) Email not changed");        $this->index($e->getMessage());      }    } |

Il metodo change(), utilizzando il model “changeEmail\_model.php”, crea un oggetto di tipo ChangeEmailClass e gli passa tutte le informazioni inserite dall’utente negli input. Se i controlli all’interno vanno bene, viene modificata l’email e poi l’utente viene reindirizzato alla pagina home, altrimenti se vengono sollevate eccezioni, viene ricaricata la pagina.

#### ChangePassword.php

|  |
| --- |
| public function index($message = ""){      if(!empty($\_SESSION['id'])){        if($\_SESSION['isAdmin'] == 1){          $this->view->errorMessage = $message;          $this->view->render('ChangePassword/index.php');        }else{          $this->view->render('Home/index.php');        }      }else{          $this->view->render('Login/index.php');      }    } |

Questa funzione carica la pagina per la modifica della password, dopo che abbia verificato che l’utente sia loggato e un amministratore. In caso contrario ricarica la pagina home se non è amministratore, oppure la pagina di login se non è loggato.

|  |
| --- |
| public function change(){      require\_once 'application/models/changePassword\_model.php';      try{        if($\_SERVER["REQUEST\_METHOD"] == "POST"){          $user = new ChangePasswordClass($\_POST["email"], $\_POST["oldPassword"],                  $\_POST["newPass"], $\_POST["confPass"]);          if($user->changePassword()){            $this->writeLog("(ChangePassword) Password changed");            $this->view->render('Home/index.php');          }        }      }catch(Exception $e){        $this->writeErrorLog("(ChangePassword) Password not changed");        $this->index($e->getMessage());      }    } |

Il metodo change() attraverso i metodi del model “changePassword\_model”, modifica la password di un utente attraverso le informazioni inserite negli input dall’utente. In caso di errore, come al solito, la pagina viene ricaricata con un messaggio di errore.

#### DeleteUser.php

|  |
| --- |
| public function index($message = ""){      if(!empty($\_SESSION['id'])){        if($\_SESSION['isAdmin'] == 1){          $this->view->errorMessage = $message;          $this->view->render('deleteUser/index.php');        }else{          $this->view->render('Home/index.php');        }      }else{        $this->view->render('Login/index.php');      }    } |

Carica la pagina per l’eliminazione (archiviazione) di un utente. Il tutto se l’utente ha i permessi per accedere ed è loggato.

|  |
| --- |
| public function delete(){      require\_once 'application/models/deleteUser\_model.php';      try{        if($\_SERVER["REQUEST\_METHOD"] == "POST"){          $user = new DeleteUserClass($\_POST["email"],                $\_POST["confEmail"]);          $user->deleteUser();          $this->writeLog("(DeleteUser) User deleted");          $this->view->render('Home/index.php');        }      }catch(Exception $e){        $this->writeErrorLog("(DeleteUser) User not deleted");        $this->index($e->getMessage());      }    } |

Il metodo crea un oggetto “DeleteUserClass” della classe model “deleteUser\_model” e gli passa le informazioni necessarie. In seguito viene richiamato il metodo dell’oggetto deleteUser(), che se i controlli vanno a buon fine, elimina, ovvero archivia, l’utente. Altrimenti ricarica la pagina.

#### Home.php

|  |
| --- |
| public function load(){      if(isset($\_POST['listComponents'])){          header('Location: '.URL."ListComponents");      }else if(isset($\_POST['users'])){          $this->view->render("Home/manageUsers.php");      }else if(isset($\_POST['add'])){          header('Location: '.URL."AddUser");      }else if(isset($\_POST['remove'])){          header('Location: '.URL."DeleteUser");      }else if(isset($\_POST['changeEmail'])){          header('Location: '.URL."ChangeEmail");      }else if(isset($\_POST['changePass'])){          header('Location: '.URL."ChangePassword");      }    } |

Questo metodo serve per individuare quale dei bottoni della pagina home è stato cliccato, e in base al quale, carica la pagina richiesta.

#### ListComponents.php

|  |
| --- |
| public function index($categoria = 0){      if(!empty($\_SESSION['id'])){        $\_SESSION["components"] = $this->listAll($categoria);        if($categoria == 0){          $this->view->errorMessage = "Selezionare una categoria";        }        $\_SESSION["idCategoria"] = $categoria;        $this->view->render('ListComponents/index.php');      }else{          $this->view->render('Login/index.php');      }    } |

Se l’utente è loggato, salva in una variabile tutte le informazioni dei componenti della categoria selezionata. In caso non fosse stata ancora selezionata una categoria, viene mostrato il messaggio “Selezionare una categoria”. Poi viene salvata la categoria in una variabile di sessione da utilizzare negli altri controller. Poi viene caricata la pagina che conterrà tutti i componenti della categoria selezionata.

|  |
| --- |
| public function askConfirm($id, $idCategoria){      $this->view->idComponente = $id;      $this->view->idCategoria = $idCategoria;      $this->view->render('ListComponents/confirmDelete.php');    } |

Questo metodo viene richiamato prima di eliminare un componente, serve per caricare la pagina di conferma prima dell’eliminazione di un componente.

#### Login

|  |
| --- |
| public function index($message = ""){      session\_unset();      $this->view->errorMessage = $message;      $this->view->render('login/index.php');    } |

All’inizio viene sempre annullata la sessione corrente, così da eliminare tutte le variabili di sessione prima di un altro login. È molto importante perché, per esempio se si effettua il logout, tutte le variabili di sessione contenenti l’id e i permessi devono assolutamente essere eliminate, così da non lasciare il permesso di caricare pagine nonostante non si sia effettuato ancora il login.

#### ModifyComponent.php

|  |
| --- |
| public function changeComponent($id){      require\_once 'application/models/component\_model.php';      if($\_SERVER["REQUEST\_METHOD"] == "POST"){        $component = new Component($id);        if($component->modifyDescAndAula($\_POST["descrizione"], $\_POST["aula"])){          header('Location: '.URL."ListComponents/index/".$\_SESSION["idCategoria"]);          $this->writeLog("(ModifyComponent) Component modified");        }else{          $this->view->idComponente = $id;          $this->view->component = $this->takeInfos($id);          writeErrorLog("(ModifyComponent) Classroom not existing");          $this->view->errorMessage = "Aula non esistente";          $this->view->render('modifyComponent/index.php');        }      }    } |

Questo metodo aggiorna nel database le modifiche del componente apportate dall’utente, e poi ricarica la lista dei componenti modificata. In caso contrario invece, ricarica la pagina con tutte le informazioni modificate dall’utente già negli input

### Libs

#### Email.php

|  |
| --- |
| function isValid(){              return filter\_var($this->fullEmail, FILTER\_VALIDATE\_EMAIL);          } |

Una funzione importante di questa classe è che ha il metodo isValid() per controllare che l’email rispetti i criteri per essere considerata un’email, ovvero almeno una chiocciola, dei punti etc…

#### Password.php

|  |
| --- |
| function isValid(){              $pass = $this->password;              $uppercase = preg\_match('@[A-Z]@', $pass);              $lowercase = preg\_match('@[a-z]@', $pass);              $number = preg\_match('@[0-9]@', $pass);              $specialChars = preg\_match('@[^\w]@', $pass);              if(!$uppercase || !$lowercase || !$number                  || !$specialChars || strlen($pass) < 8) {                      return false;              }              return true;          } |

Anche in questa classe c’è il metodo isValid(), che controlla che la password rispetti dei criteri: abbia una maiuscola, una minuscola, un numero, un carattere speciale e che sia lunga almeno 8 caratteri.

#### Connection.php

|  |
| --- |
| $servername = "localhost";  $username = "root";  $password = "root";  $dbname = "inventariohw";  $conn = new mysqli($servername, $username, $password, $dbname);  if($conn->connect\_error){     die("Connection failed: ".$conn->connect\_error);  } |

Questa classe viene richiamata da ogni metodo che deve interrogare o comunicare col database; infatti, stabilisce una connessione con il database, effettuando l’accesso.

#### Controller.php

|  |
| --- |
| function writeLog($msg){          if($puntatore = fopen('application/logs/log.log', "a")){              $user = isset($\_SESSION['id']) ? $\_SESSION['id'] : "undefined";              $str = date("Y/m/d H:i:s") . " user\_id=" . $user . ": " . $msg;              fwrite($puntatore, $str . PHP\_EOL);          }      } |

Il metodo scrive all’interno di un file log e serve per registrare qualsiasi modifica del database, con orario della modifica, id dell’utente che l’ha effettuata e il messaggio passato come parametro.

Esiste anche il metodo writeErrorLog() che è identico a questo, ma salva gli errori in un altro file.

#### Hash.php

|  |
| --- |
| function doHash($salt){  $this->hashedText = hash('sha256', $salt . hash('sha256', $this->plainText));  } |

Viene richiamato qualora si volesse fare l’hash di una password in sha256. C’è bisogno di passare come parametro la salt, ovvero l’email. Questo serve per creare una password sicura e non salvare sul database i dati in chiaro.

### Models

#### AddUser\_model.php

|  |
| --- |
| public function createUser(){      require 'application/libs/connection.php';      require 'application/libs/Register/email.php';      require 'application/libs/Register/password.php';      $emailUser = new Email($this->email);      if($emailUser->isValid()){          $passUser = new Password($this->password);          if($passUser->isValid()){              if($this->password == $this->confPass){                  $this->getHashedPass();                  $sql = "SELECT \* FROM utente WHERE email='$this->email'";                  $result = $conn->query($sql);                  if ($result->num\_rows == 0) {                      $sql = "INSERT INTO utente(nome, cognome, email, password,  is\_admin, is\_enable)                      VALUES('$this->name','$this->surname','$this->email',  '$this->hashedPassword', $this->isAdmin, TRUE)";                      $conn->query($sql);                      return TRUE;                  }else{                      throw new Exception("Utente già esistente");                  }              }else{                  throw new Exception("Le password non corrispondono");              }          }else{              throw new Exception("La password deve contenere almeno:<br>- 8 Caratteri<br>-1 Maiuscola<br>-1 Minuscola<br>-1 Cifra<br>-1 Carattere speciale");          }      }else{          throw new Exception("Email invalida");      }  } |

In questo model vengono importate le varie classi per la connessione al database e per controllare la password ed email. In caso i controlli vadano a buon fine, viene calcolata la password in sha256 e poi aggiunto il nuovo utente all’interno del database.

#### ChangeEmail\_model.php

|  |
| --- |
| function changeEmail(){      require 'application/libs/connection.php';      require 'application/libs/Register/email.php';      if($this->equals()){          $emailUser = new Email($this->newEmail);          if($emailUser->isValid()){              $oldHashedPass = $this->getHashedPass($this->oldEmail);              $sql = "SELECT \* FROM utente WHERE email='$this->oldEmail' &&  password='$oldHashedPass'";              $result = $conn->query($sql);              if ($result->num\_rows > 0) {                  $this->hashedPass = $this->getHashedPass($this->newEmail);                  $sql = "UPDATE utente SET email='$this->newEmail', password='$this-  >hashedPass' WHERE email='$this->oldEmail' AND is\_enable=1";                  $conn->query($sql);                  if(mysqli\_affected\_rows($conn) > 0 ){                      return TRUE;                  }                  throw new Exception("La nuova email deve essere diversa dalla  vecchia");              }else{                  throw new Exception("Email o password errata");              }          }else{              throw new Exception("Nuova email non valida");          }      }else{          throw new Exception("Le due email non coincidono");      }  } |

Questo metodo serve per cambiare l’email di un utente. Vengono effettuati vari controlli, come che la nuova email e la ripetizione di essa siano uguali, e che rispetti i criteri dell’email. Poi viene cercato nel database l’utente che ha la vecchia email e password. In caso venisse trovato un utente, cambia la sua email con la nuova e ricalcola la password con la nuova salt.

#### ChangePassword\_model.php

|  |
| --- |
| function changePassword(){      require 'application/libs/connection.php';      if($this->equals()){          $oldHashedPass = $this->getHashedPass($this->email, $this->oldPass);          $sql = "SELECT \* FROM utente WHERE email='$this->email' &&  password='$oldHashedPass'";          $result = $conn->query($sql);          if ($result->num\_rows > 0) {              $this->newHashedPass = $this->getHashedPass($this->email,  $this->newPass);              $sql = "UPDATE utente SET password='$this->newHashedPass' WHERE  email='$this->email' AND is\_enable=1";              $result = $conn->query($sql);              if(mysqli\_affected\_rows($conn) > 0){                  return TRUE;              }              throw new Exception("La nuova password deve essere diversa da quella  vecchia");          }else{              throw new Exception("Email o password errata");          }      }else{          throw new Exception("Le due password non coincidono");      }  } |

Viene controllato che la password nuova e quella di conferma siano identiche, poi viene calcolato l’hash della nuova pasword ed in seguito viene cercato nel database l’utente con l’email e vecchia password. In caso venga trovato, gli viene modificata la password inserendo il nuovo hash. Invece in caso che 1 dei controlli non vada bene, viene sollevata un’eccezione con un messaggio per spiegare il motivo dell’errore.

#### Component\_model.php

|  |
| --- |
| static function getComponents($idCategoria){      require 'application/libs/connection.php';      $sql = "";      if($idCategoria != 6){          $sql = "SELECT \* FROM componente WHERE tipo\_id='$idCategoria' &&  is\_enable=1";      }else{          $sql = "SELECT \* FROM componente WHERE is\_enable=0";      }      $result = $conn->query($sql);      $out = array();      if ($result->num\_rows > 0) {          foreach ($result as $row) {              unset($row['utente\_id']);              unset($row['tipo\_id']);              unset($row['is\_enable']);              $out[] = $row;          }          return $out;      }else{          throw new Exception();      }  } |

Questo metodo torna tutti i componenti e le sue informazioni di una specifica categoria passata come parametro. In caso che l’id della categoria sia 6, allora vengono ritornati tutti i componenti ormai archiviati. Nella selezione delle informazioni vengono eliminate alcune colonne che contengono informazioni non utili per la stampa dei componenti. Ciò viene effettuato con unset() e l’indice della colonna da eliminare.

|  |
| --- |
| function delete(){      require 'application/libs/connection.php';      $sql = "UPDATE componente SET is\_enable=0 WHERE id = $this->id";      $result = $conn->query($sql);      if ($result) {          return TRUE;      }      return FALSE;  } |

Per eliminare un componente, viene ricercato nel database l’id identico a quello dell’attributo. In caso ci sia all’interno della lista, viene settato “is\_enable” a 0, ovvero come che venisse archiviato.

|  |
| --- |
| function modifyDescAndAula($desc, $aula){      require 'application/libs/connection.php';      $aula = intval($aula);      if(!Component::checkAula($aula)){          $a = 1;      }      $sql = "UPDATE componente SET descrizione='$desc', aula\_id=$aula WHERE id =  $this->id";      $result = $conn->query($sql);      if ($result) {          return TRUE;      }      return FALSE;  } |

Vengono passati come parametri la descrizione e l’aula del componente. Poi viene controllato che l’aula sia con un numero da 1 a 500 ed in seguito viene cercato nel database il componente con l’id pari a quello dell’attributo, e in caso esista nel database, modifica la descrizione e l’aula inserendo quelle nuove.

|  |
| --- |
| static function addComponent($marca, $desc, $nSeriale, $aula, $categoria){      require 'application/libs/connection.php';      if(!empty($marca) && !empty($desc) && !empty($nSeriale) && !empty($aula) && !empty($categoria)){          $aula = intval($aula);          if(Component::checkAula($aula)){              $idUser = $\_SESSION['id'];              $sql = "INSERT INTO componente(marca, descrizione, numero\_seriale, data\_installazione, utente\_id, tipo\_id, aula\_id, is\_enable)              VALUES('$marca', '$desc', '$nSeriale', CURDATE(), $idUser, $categoria, $aula, TRUE)";              $result = $conn->query($sql);              if ($result) {                  return TRUE;              }              throw new Exception("Aula non esistente");          }else{              throw new Exception("Aula non esistente");          }      }else{          throw new Exception("Completare tutti i campi");      }  } |

La funzione richiede vari parametri: la marca, descrizione, numero seriale, aula e categoria di un componente. Poi controlla che tutte le informazioni non siano vuote e in seguito passa al controllo dell’aula. Se tutti questi controlli vanno a buon fine, viene aggiunto nel database un nuovo componente con le informazioni passate come parametri. Poi viene controllato che l’aggiunta sia andata buon fine, in caso contrario ritorna false. Quest’ultima parte serve per il controller per sapere se ricaricare la pagina oppure no.

#### DeleteUser\_model.php

|  |
| --- |
| function deleteUser(){      require 'application/libs/connection.php';      if($this->equals()){          $sql = "SELECT \* FROM utente WHERE email='$this->email'";          $result = $conn->query($sql);          if ($result->num\_rows > 0) {              $sql = "UPDATE utente SET is\_enable = FALSE WHERE  email='$this->email' AND is\_enable=1";              $conn->query($sql);              if(mysqli\_affected\_rows($conn) > 0 ){                  return TRUE;              }              throw new Exception("Utente non trovato");          }else{              throw new Exception("Utente non esistente");          }      }else{          throw new Exception("Le due email non coincidono");      }  } |

Come prima cosa viene controllato che le due email siano identiche. Se sono uguali, allora viene cercato nel database che l’utente sia presente, in caso contrario solleva un’eccezione. Se l’utente esiste, né modifica l’attributo “is\_enable” a false, così da archiviare l’utente senza eliminarlo, per non perdere i suoi riferimenti nelle altre tabelle.

#### Login\_model.php

|  |
| --- |
| function doLogin(){      require 'application/libs/connection.php';      $this->getHashedPass();      $sql = "SELECT \* FROM utente WHERE email='$this->email' AND  password='$this->hashedPassword' AND is\_enable=1";      $result = $conn->query($sql);      $row = $result->fetch\_assoc();      if ($result->num\_rows > 0) {          if($row['is\_enable'] == 1){              $\_SESSION['id'] = $row['id'];              $\_SESSION['isAdmin'] = $row['is\_admin'];              return true;          }          session\_destroy();          throw new Exception("Utente disabilitato");      }else{          session\_destroy();          throw new Exception("Email o password non valida");      }  } |

Prima di tutto viene fatto l’hash della password e poi salvata in un attributo della classe. Poi viene cercato nel database l’utente con le credenziali identiche a quelle inserite dall’utente. Se viene trovato, aggiunge 2 variabili di sessione che contengono l’id dell’utente e se quest’ultimo è un admin oppure no. Se invece l’utente viene trovato, ma è un utente archiviato, viene chiusa la sessione e sollevato un errore.

# Test

## Protocollo di test

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-01 | **Nome:** | Accesso utente base |
| **Descrizione:** | Si proverà a fare l’accesso con un utente base | | |
| **Prerequisiti:** | Aver terminato la pagina di login | | |
| **Procedura:** | * Inserire nel database un utente base * Aprire la pagina di login del sito * Inserire le credenziali d’accesso dell’utente base | | |
| **Risultati attesi:** | L’accesso viene effettuato e si apre la pagina home per gli utenti base | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-002  REQ-01 | **Nome:** | Accesso utente amministratore |
| **Descrizione:** | Si proverà a fare l’accesso con un utente amministratore | | |
| **Prerequisiti:** | Aver terminato la pagina di login | | |
| **Procedura:** | * Inserire nel database un utente amministratore * Aprire la pagina di login del sito * Inserire le credenziali d’accesso dell’utente amministratore | | |
| **Risultati attesi:** | L’accesso viene effettuato e si apre la pagina home per gli utenti amministratori | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-003  REQ-02 | **Nome:** | Visualizza elenco componenti di una categoria |
| **Descrizione:** | Si proverà a visualizzare l’elenco dei componenti di una categoria | | |
| **Prerequisiti:** | Aver terminato la pagina per consultare l’inventario | | |
| **Procedura:** | * Fare l’accesso con un utente * Aprire la pagina per consultare l’inventario | | |
| **Risultati attesi:** | La lista viene mostrata con tutti i componenti di quella categoria | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-004  REQ-03 | **Nome:** | Aggiungere un componente |
| **Descrizione:** | Provare ad aggiungere un componente all’inventario | | |
| **Prerequisiti:** | Terminato la pagina web | | |
| **Procedura:** | * Fare il login con un utente * Andare in “Consulta inventario” * In alto a destra cliccare “+” e aggiungere i dati per il nuovo componente | | |
| **Risultati attesi:** | Il componente viene aggiunto alla lista | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-005  REQ-04 | **Nome:** | Eliminare un componente |
| **Descrizione:** | Si proverà ad eliminare un componente dall’inventario | | |
| **Prerequisiti:** | Terminato la pagina web | | |
| **Procedura:** | * Fare il login con un utente * Andare in “Consulta inventario” * A destra di un componente cliccare “-“ | | |
| **Risultati attesi:** | Il componente viene eliminato dalla lista | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-006  REQ-05 | **Nome:** | Modificare un componente |
| **Descrizione:** | Si proverà a modificare un componente dall’inventario | | |
| **Prerequisiti:** | Terminato la pagina web | | |
| **Procedura:** | * Fare il login con un utente * Andare in “Consulta inventario” * A destra di un componente cliccare “modifica” e modificare le informazioni | | |
| **Risultati attesi:** | Il componente viene mostrato nella lista con le informazioni modificate | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-007  REQ-06 | **Nome:** | Aggiunta di un utente base |
| **Descrizione:** | Aggiungere un nuovo utente base con i propri dati d’accesso | | |
| **Prerequisiti:** | Aver terminato la pagina di login e home | | |
| **Procedura:** | * Fare il login con un utente amministratore * Andare in “Gestione utenti” > “Aggiungi utente” * Inserire le informazioni necessarie per creare un nuovo utente * Fare il logout dall’utente amministratore * Provare a fare il login con l’utente appena creato | | |
| **Risultati attesi:** | Dopo aver effettuato il login, verrà aperta la pagina home | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-008  REQ-06 | **Nome:** | Aggiunta di un utente amministratore |
| **Descrizione:** | Aggiungere un nuovo utente amministratore con i propri dati d’accesso | | |
| **Prerequisiti:** | Aver terminato la pagina di login e home | | |
| **Procedura:** | * Fare il login con un utente amministratore * Andare in “Gestione utenti” > “Aggiungi utente” * Inserire le informazioni necessarie per creare un nuovo utente * Fare il logout dall’utente attuale * Provare a fare il login con l’utente appena creato | | |
| **Risultati attesi:** | Dopo aver effettuato il login, verrà aperta la pagina home con anche la possibilità di gestire gli utenti | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-09  REQ-07 | **Nome:** | Eliminare un utente |
| **Descrizione:** | Eliminare un utente | | |
| **Prerequisiti:** | Aver terminato la pagina web | | |
| **Procedura:** | * Fare il login con un utente amministratore * Andare in “Gestione utenti” > “Elimina utente” * Inserire le informazioni necessarie per eliminare un utente * Fare il logout dall’utente amministratore * Provare a fare il login con l’utente appena eliminato | | |
| **Risultati attesi:** | Una volta eliminato un utente, provare a fare il login ed aspettarsi un errore | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-010  REQ-08 | **Nome:** | Modificare un utente |
| **Descrizione:** | Modificare un utente | | |
| **Prerequisiti:** | Aver terminato la pagina web | | |
| **Procedura:** | * Fare il login con un utente amministratore * Andare in “Gestione utenti” > “Modifica utente” * Inserire le informazioni necessarie per modificare un utente * Fare il logout dall’utente amministratore * Provare a fare il login con l’utente appena modificato | | |
| **Risultati attesi:** | Una volta modificato un utente, provare a fare il login con le nuove credenziali | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-011  REQ-009 | **Nome:** | Percentuale soglia |
| **Descrizione:** | Inserire una soglia minima dei componenti liberi | | |
| **Prerequisiti:** | Terminato la pagina web | | |
| **Procedura:** | * Fare il login con un utente * Andare in “Imposta soglia” * Modificare lo slider della percentuale | | |
| **Risultati attesi:** | Quando la percentuale di componenti usati supera la soglia, verrà notificato | | |

## Risultati test

|  |  |  |  |
| --- | --- | --- | --- |
| **Test case** | **Esito** | **Risultati** | **Data** |
| TC-001 | **PASSATO** | Viene mostrata la pagina home dell’utente base | 23.12.2021 |
| TC-002 | **PASSATO** | L’utente viene reindirizzato alla pagina home d’amministratore | 23.12.2021 |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-003 | **PASSATO** | Vengono mostrati tutti i componenti di una categoria | 16.12.2021 |
| TC-004 | **NON PASSATO** | L’elemento non viene aggiunto alla lista | 16.12.2021 |
| TC-004 | **PASSATO** | È stato aggiunto un nuovo componente alla lista | 16.12.2021 |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-005 | **PASSATO** | Il componente eliminato viene aggiunto alla lista di quelli archiviati | 16.21.2021 |
| TC-006 | **PASSATO** | Il componente è stato modificato e la lista aggiornata con la nuova descrizione e aula | 16.21.2021 |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-007 | **PASSATO** | Una volta aggiunto un nuovo utente base ed aver fatto il login con esso, viene caricata la sua home | 16.12.2021 |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-008 | **PASSATO** | Una volta aggiunto un nuovo utente amministratore ed aver fatto il login con esso, viene caricata la sua home | 16.12.2021 |

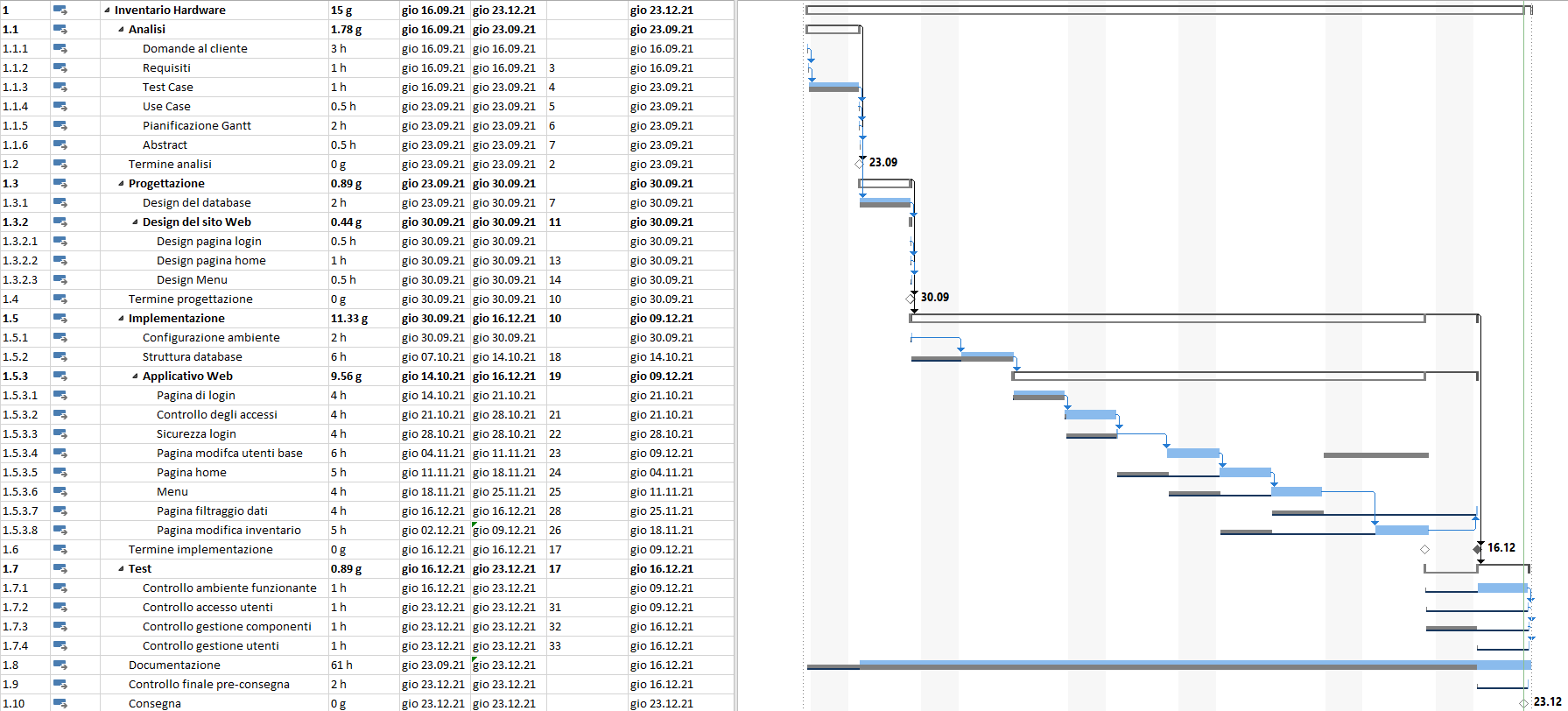
|  |  |  |  |
| --- | --- | --- | --- |
| TC-009 | **PASSATO** | Una volta eliminato l’utente, provando ad effettuare il login con le credenziali di esso, verrà mostrato un errore | 16.12.2021 |

|  |  |  |  |
| --- | --- | --- | --- |
| TC-010 | **PASSATO** | Dopo aver cambiato la passwod di un utente, si effettua l’accesso con esso e viene mostrata la pagina home | 16.12.2021 |
| TC-011 | **NON PASSATO** | Non sono riuscito ad implementare questa funzione | 23.12.2021 |

## Mancanze/limitazioni conosciute

La mancanza più grande è sicuramente la possibilità di impostare la soglia che notifichi all’utente quanti componenti sono rimasti in magazzino. Questo purtroppo è dovuto dal poco tempo disponibile e dalle mie conoscenze iniziali di PHP molto limiate, che mi hanno rallentato durante l’implementazione dell’applicativo.

# Consuntivo



# Conclusioni

## Sviluppi futuri

Per migliorare il prodotto si potrebbe sicuramente aggiungere la possibilità di filtrare i componenti in base a criteri per velocizzare la ricerca. Oppure implementare la soglia, così da ricevere una notifica qualora una tipologia di componente si stia esaurendo dal magazzino.

Molto utile sarebbe anche aggiornare la grafica rendendola più semplice e pulita.

## Considerazioni personali

Per concludere posso dire che il progetto sia stato molto utile e mi ha insegnato a lavora in modo più organizzato e ordinato, per rispettare i tempi di consegna e le specifiche date dal cliente. In più ho imparato a redigere un diario di lavoro a fine di ogni giornata, così da tenere traccia di cosa ho fatto la giornata di lavoro precedente.

Ho imparato ad utilizzare il linguaggio di PHP e MySQL assieme, per collegare il database al sito web, migliorando le mie conoscenze in questo campo.

Il risultato finale del progetto mi soddisfa, soprattutto perché sono riuscito a realizzarlo nonostante le mie limitate conoscenze in questo campo. L’unica cosa che mi spiace è non essere riuscito a terminare le specifiche del quaderno dei compiti, ovvero implementare la possibilità di impostare una soglia.

Per realizzare la parte grafica, ho imparato ad utilizzare anche Bootstrap.

# Glossario

|  |  |
| --- | --- |
| **Nome** | **Definizione** |
| **Applicativo WEB** | Il sito web che verrà visualizzato a progetto terminato |
| **Componente** | Si intende o un monitor, un proiettore, un computer o una sua periferica, come mouse e tastiera |
| **Controller** | È una classe che collega la parte grafica con la parte di modifica dei dati di un database. |
| **Database** | Struttura preparata su tabelle che permette lo storage di informazioni riguardanti un applicativo o progetto. |
| **Date** | Tipo di dato che salva unicamente la data in SQL |
| **Gantt** | Diagramma per pianificare un progetto dividendolo in varie attività, per avere un lavoro più organizzato |
| **Log** | File che contiene tutte le informazioni per l’analisi e il troubleshooting |
| **MVC** | Pattern utilizzato per dividere le funzionalità ben distinte di un applicativo |
| **Soglia** | Limite massimo che può raggiungere il totale di componenti, di una categoria, in utilizzo. |

# Sitografia

* <https://httpd.apache.org/>, 30.09.2021
* <https://stackoverflow.com/>, 09.12.2021
* <https://www.php.net/>, 16.12.2021
* <https://www.w3schools.com/>, 16.12.2021
* <https://www.html.it/pag/18299/il-pattern-mcv/>, *Pattern MVC*, 16.12.2021

# Allegati

Allegati:

* Diari di lavoro
* Codici sorgente presente su GitHub
* Quaderno dei compiti
* Manuale di sistema
* Gantt