CPT Meeting Room Scheduler

[1 Introduzione 3](#_Toc480933422)

[1.1 Informazioni sul progetto 3](#_Toc1637425310)

[1.2 Abstract 3](#_Toc57283201)

[1.3 Scopo 3](#_Toc1747372838)

[Analisi 4](#_Toc1191930074)

[1.4 Analisi del dominio 4](#_Toc842511629)

[1.5 Analisi e specifica dei requisiti 4](#_Toc636897841)

[1.6 Use case 8](#_Toc1021644958)

[1.7 Pianfiicazione 1](#_Toc1653911648)

[1.7.1 Spiegazione diagramma Gantt 1](#_Toc1938727864)

[1.8 Analisi dei mezzi 4](#_Toc1460356270)

[1.8.1 Software 4](#_Toc1790563891)

[1.8.2 Hardware 4](#_Toc192596608)

[2 Progettazione 5](#_Toc408451789)

[2.1 Design dell’architettura del sistema 5](#_Toc750465559)

[2.2 Design dei dati e database 6](#_Toc1926658076)

[2.2.1 Schema logico 7](#_Toc1537951779)

[2.2.2 Tipi di dati 7](#_Toc1853835088)

[2.3 Design delle interfacce 9](#_Toc816106319)

[2.4 Design procedurale 11](#_Toc234347785)

[3 Implementazione 12](#_Toc136658767)

[3.1 Gestione permessi 12](#_Toc1242538687)

[3.1.1 Classe PermissionModel e classe Permissions 12](#_Toc1278274491)

[3.2 Pannello admin 14](#_Toc744392361)

[3.2.1 Gestione utenti 14](#_Toc1094143941)

[3.2.2 Gestione prenotazioni 16](#_Toc172493935)

[4 Test 16](#_Toc1096488927)

[4.1 Protocollo di test 16](#_Toc836677935)

[4.2 Risultati test 17](#_Toc1747270241)

[4.3 Mancanze/limitazioni conosciute 17](#_Toc1664753119)

[5 Consuntivo 17](#_Toc2131440854)

[6 Conclusioni 17](#_Toc80720015)

[6.1 Sviluppi futuri 17](#_Toc1154694781)

[6.2 Considerazioni personali 17](#_Toc41240407)

[7 Bibliografia 17](#_Toc1828092853)

[7.1 Bibliografia per articoli di riviste: 17](#_Toc199141207)

[7.2 Bibliografia per libri 18](#_Toc883752037)

[7.3 Sitografia 18](#_Toc317507046)

[8 Allegati 18](#_Toc1220786165)

# Introduzione

## Informazioni sul progetto

Allievo coinvolto: Luca Di Bello

Classe: Informatica 4AC presso la Scuola di Arti e Mestieri a Trevano

Docenti responsabili: Fabrizio Valsangiacomo

Data inizio: 03 / 09 / 2019

Data fine: 20 / 12 / 2019

## Abstract

*Paper has been used for many centuries in the past and is still used for different purposes; with the passing of time we have realized that*

*it can become chaotic to the point of losing valuable information.*

*At the moment there are many meeting rooms that are reserved using an old piece of paper risking to blow plans*

*of months and months.*

*With this product you have a much cheaper, more ecological and safer substitute!*

*All the information is saved in a database and everything is accessible and modifiable through a*

*simple and intuitive web page, reachable from any device, whether mobile or not.*

## Scopo

Lo scopo di questo progetto è quello di implementare un sistema utile per gestire le prenotazioni dell’aula riunioni della sezione informatica. Il tutto sarà controllato da remoto tramite delle interfacce web. Verrà posto un monitor collegato ad un Raspberry Pi il quale permetterà di leggere quando la sala riunioni è occupata.

Questo progetto verrà anche utilizzato per prepararci all’esame di maturità che dovremo affrontare alla fine dell’anno.

## Analisi

## Analisi del dominio

Fino ad ora i professori utilizzavano un foglio di carta appeso alla porta con scritto chi, quando e per quanto tempo verrà utilizzata l’aula, dunque si cerca un modo per rendere tutto più chiaro e semplice utilizzando un sistema informatico.

## Analisi e specifica dei requisiti

|  |  |
| --- | --- |
| **ID: REQ-001** | |
| **Nome** | Creazione banca dati |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | - |
| **Sotto requisiti** | |
| **001** | Si necessita una tabella che tiene in memoria tutti gli utenti ed i permessi che hanno |
| **002** | Si necessita una tabella che tiene in memoria quali riservazioni sono state fatte |
| **003** | Il database dev’essere normalizzato fino al livello BCNF per semplificare modifiche future |
| **004** | Utente aggiuntivo con permessi limitati da utilizzare per leggere i dati dal sito web |

|  |  |
| --- | --- |
| **ID: REQ-002** | |
| **Nome** | Login con LDAP |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Si necessitano i permessi per accedere al servizio di *active directory* di scuola |
| **Sotto requisiti** | |
| **001** | Pagina web con form che permette all’utente di inserire le proprie credenziali, quindi username e password |
| **002** | Pagina responsive e funzionante su qualunque dispositivo (mobile e non) |

|  |  |
| --- | --- |
| **ID: REQ-003** | |
| **Nome** | Permessi degli utenti |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | - |
| **Sotto requisiti** | |
| **001** | Livello utente normale: può vedere ed inserire una riservazione. Inoltre, può eliminare delle riservazioni fittizie |
| **002** | Livello utente avanzato: ha gli stessi permessi di un utente normale ma può inserire e cancellare riservazioni di un altro utente |
| **003** | Livello utente amministratore: ha tutti i premessi disponibili |

|  |  |
| --- | --- |
| **ID: REQ-004** | |
| **Nome** | Pagina gestione utenti |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | - |
| **Sotto requisiti** | |
| **001** | Si necessita una pagina web dove un utente amministratore può creare, cancellare modificare agli utenti |
| **002** | L’amministratore deve poter creare degli utenti normali ed utenti avanzati (con permessi aggiuntivi) assegnando una password di default. |
| **003** | Al primo login di un utente gli deve essere chiesto di cambiare la password di default con una personale |
| **004** | L’amministratore deve poter cambiare i permessi di un utente in ogni momento. |
| **005** | Alla creazione di un nuovo utente la password generata verrà inviata per e-mail all’utente in questione |
| **006** | Ci deve sempre essere almeno un utente amministratore. |
| **006** | Non deve essere possibile la registrazione di un utente con un’e-mail esterna a quella scolastica (gmail, Sunrise, ..) |

|  |  |
| --- | --- |
| **ID: REQ-005** | |
| **Nome** | Gestione prenotazioni |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | - |
| **Sotto requisiti** | |
| **001** | Deve essere presente un form che permette di aggiungere una nuova prenotazione. I campi necessari sono: data, l’orario di inizio e di fine. Anche il nome, il cognome e la mail di chi fa la riservazione devono essere mostrati. Dev’essere presente un campo utile per aggiungere delle osservazioni. |
| **002** | Controllo sulla data ed ora: non devono essere passati e non dev’esserci già una prenotazione |
| **003** | L’orario della prenotazione dev’essere a blocchi di 15 minuti |
| **004** | Alla creazione e cancellazione di una riservazione verrà inviata un’e-mail di conferma al relativo utente |

|  |  |
| --- | --- |
| **ID: REQ-006** | |
| **Nome** | Schermo per visualizzazione prenotazioni |
| **Priorità** | 1 |
| **Versione** | 1.0 |
| **Note** | Verrà posto (tramite un supporto in alluminio) davanti all’aula riservata ai colloqui. |
| **Sotto requisiti** | |
| **001** | Non si dovrà vedere nessuna pagina web |
| **002** | Devono essere mostrate le date, l’orario di inizio e fine della riservazione, e le informazioni relative al docente che ha eseguito la riservazione (nome e cognome) |
| **003** | Deve mostrare le osservazioni se aggiunte dal docente durante la riservazione |
| **004** | Il sistema verrà controllato tramite un Raspberry Pi Model B fornito dalla scuola |

|  |  |
| --- | --- |
| **ID: REQ-007** | |
| **Nome** | Generazione di report |
| **Priorità** | 2 |
| **Versione** | 1.0 |
| **Note** | - |
| **Sotto requisiti** | |
| **001** | I report devono essere in formato PDF |
| **002** | I file di report devono essere generati automaticamente dalla pagina web |
| **003** | 3 tipi di report: giornalieri, settimanali e mensili. |

## Use case

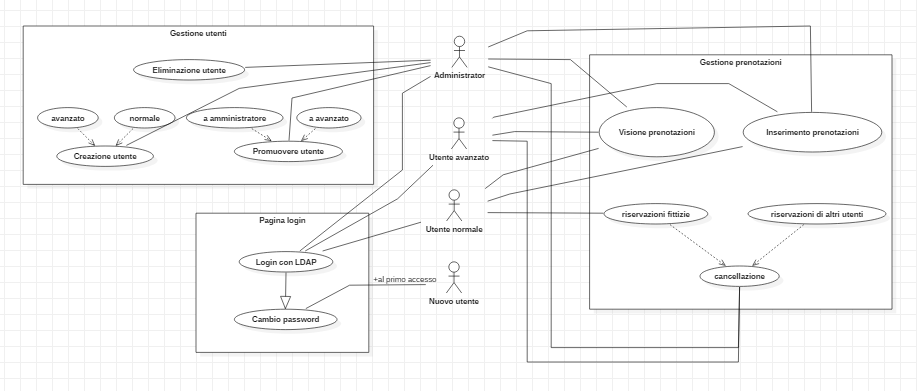


Figure 1 - Diagramma *Use Case*

Come si può vedere molto bene dall’immagine sovrastante ho deciso di dividere il diagramma in 3 macro-gruppi (gestione utenti, gestione prenotazioni e pagina login) per semplificare sia la lettura che la stesura del diagramma.

L’utente con i permessi di amministratore ha tutti i permessi sia nella gestione utenti sia nella gestione delle prenotazioni. L’utente avanzato invece ha tutti i permessi nella gestione delle prenotazioni però non può eliminare, creare o promuovere utenti.

L’utente normale invece può soltanto creare prenotazioni, visionare le prenotazioni fatte dagli altri professori ed eliminare le prenotazioni create da lui (non quelle di altri utenti).

Il “nuovo utente” invece come si può vedere, ha soltanto i permessi per cambiare la password. Questo perché al primo login di un utente il sistema richiede di cambiare la password di default con una password personale. Una volta cambiata la password l’utente viene promosso automaticamente ad “Utente normale”.

## Pianfiicazione

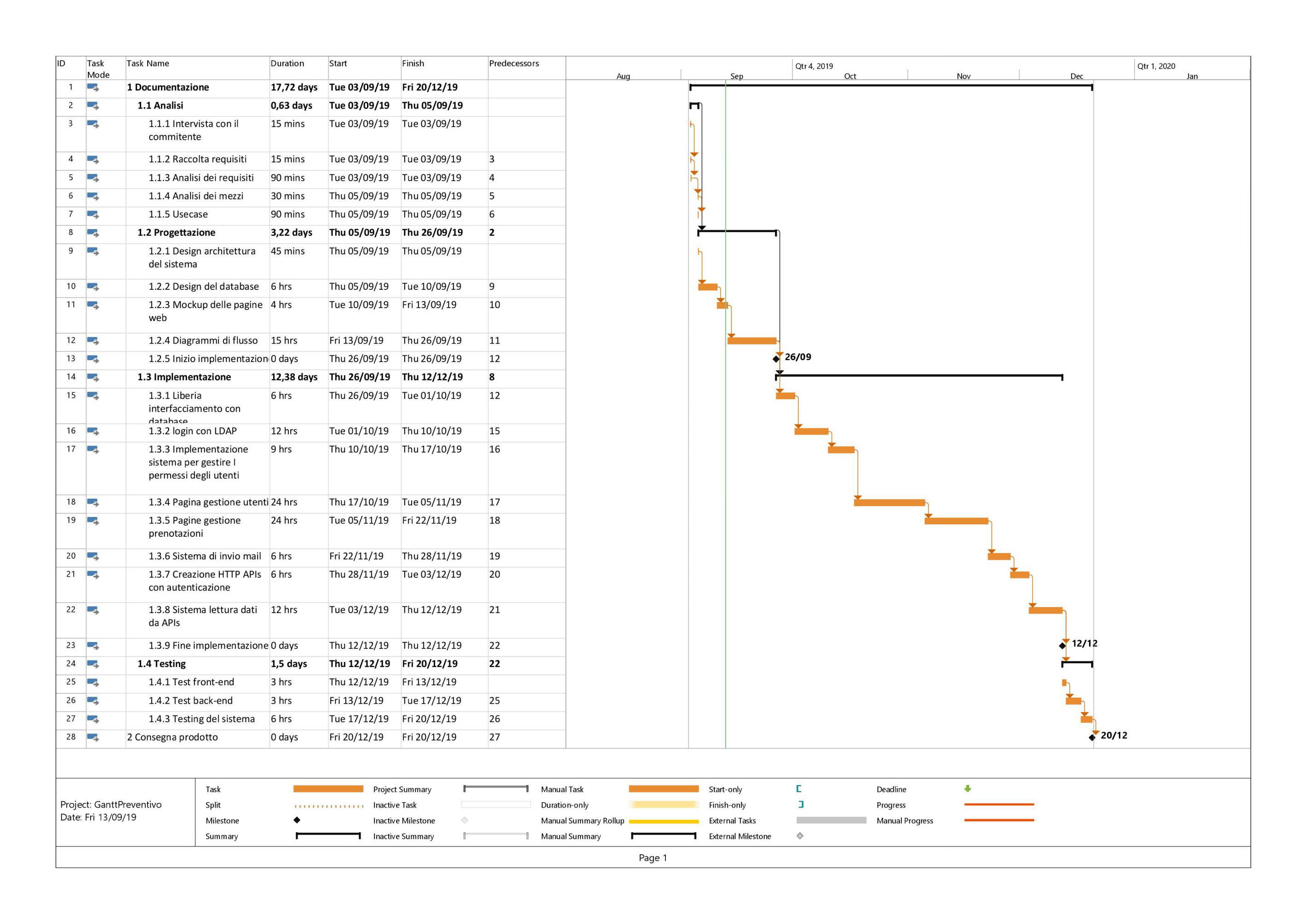


Figure 2 - Diagramma Gantt preventivo

### Spiegazione diagramma Gantt

Nei vari sotto capitoli spiegherò nello specifico cosa avverrà all’interno di ogni attività. Molte di queste sono state riprese dalla documentazione.

#### Analisi

Figure 3 - Gantt attività analisi

Ho deciso di suddividere il capitolo di analisi in 5 attività:

* Intervista con il committente
  + Intervista con il committente, utilizzata per chiarire i dubbi relativi alla consegna
* Raccolta dei requisiti
  + Raccolta dei requisiti tramite la lettura approfondita del QdC (Quaderno Dei Compiti)
* Analisi dei requisiti
  + Riflessione riguardo alle richieste raccolte, analizzandole e suddividendole in modo sensato
* Analisi dei mezzi
  + Riflessione riguardo ai mezzi da utilizzare per il progetto (sia Software che Hardware)
* Use case
  + Stesura del diagramma Use Case

#### Progettazione

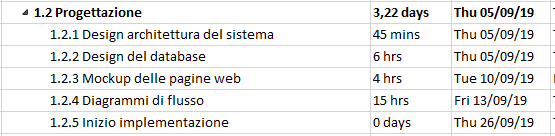


Figure 4 - Gantt attività progettazione

Ho deciso di suddividere il capitolo riguardante alla progettazione del sistema in altre 4 attività:

* Design architettura del sistema
  + Schema che mostra a livello grafico come i vari componenti del sistema interagiscono tra loro
* Design del database
  + Diagramma *Entity Relation* (ER) e schema logico che illustra la struttura del database
* Mockup delle pagine web
  + Sketch delle pagine web
* Diagrammi di flusso
  + Diagrammi di flusso che mostrano il funzionamento di ogni componente del sistema

#### Implementazione

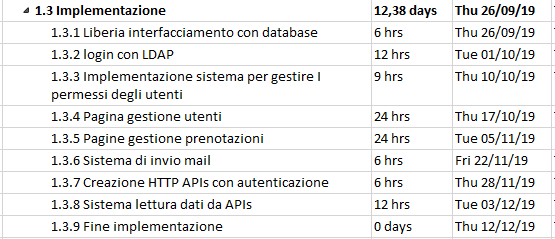


Figure 5 - Gantt attività implementazione

Ho deciso di suddividere il capitolo riguardante all’implementazione del sistema in 8 attività:

* Libreria interfacciamento con database
  + Implementazione di una libreria che permetterà la connessione e l’invio di query al database
* Login con LDAP
  + Sistema di login utilizzando LDAP.
* Implementazione sistema per gestire i permessi degli utenti
  + Implementazione di un sistema che controlla i permessi di un utente prima di eseguire qualsiasi operazione
* Pagina gestione utenti
  + Pagina che permette di gestire gli utenti come da requisiti
* Pagina gestione prenotazioni
  + Pagina che permette di gestire le prenotazioni come da requisiti
* Sistema di invio mail
  + Sistema che permette l’invio di mail di conferma ad un determinato utente
* Creazione http APIs con autenticazione
  + Creazione di Api http con autenticazione che permettono la lettura di dati del database. Esse verranno utilizzate dal Raspberry per mostrare i dati su schermo
* Sistema lettura dati da APIs
  + Sistema di lettura e formattazione dei dati utilizzando le API fornite dal sito web

#### Testing

Figure 6 - Gantt attività testing

Ho deciso di suddividere il capitolo riguardante al testing del sistema in 3 attività:

* Test front-end
  + Testing delle pagine web nella loro interezza
* Test back-end
  + Testing di tutto il back-end
* Testing del sistema
  + Testing approfondito dell’intero sistema

## Analisi dei mezzi

### Software

Per lo sviluppo di questo sistema ho utilizzato questi software:

* PHP 7.2.10: Linguaggio utilizzato per gestire la logica del sito da back-end
* Apache 2.4.27: Web server utilizzato per sviluppare il sito dalla mia macchina
* PhpMailer 6.0.7: Libreria PHP utilizzata per inviare mail agli utenti del sito.
* MDB 4.8.3: Acronimo di *Material Design for Bootstrap*. Esso è un framework front-end basato su *Boostrap 4* ma con una grafica che segue il concetto di casa Google chiamato *“Material Design”*.
* XDebug: Modulo aggiuntivo per PHP che fornisce di avere delle informazioni aggiuntive utili durante il debug
* PHPStorm
* MySQL Workbench 6.3: Applicativo che permette la gestione completa di uno o più database a livello grafico (tramite GUI)
* MeekroDB 2.3: Framework per database in PHP. Esso permette di semplificare l’iterazione tra applicazione e database MySQL

### Hardware

L’intero progetto è stato sviluppato sul mio laptop personale. Queste sono le specifiche:

* Nome: Asus ROG gl702vm
* Ram: 16GB
* CPU: Intel Core i7-7700HQ 2.80GHz
* Gpu: Nvidia GTX 1060 6GB
* Durante lo sviluppo di questo progetto ho utilizzato i seguenti sistemi operativi (OS):
  + Windows 10 Home versione 1809
  + PopOS! 18.04 LTS/19.10

Per controllare il monitor invece ho utilizzato un Raspberry Pi Model B, queste sono le sue specifiche:

* Nome: Raspberry Pi Model B
* Ram: 1GB LPDDR2 (900 MHz)
* CPU: 4× ARM Cortex-A53, 1.2GHz
* OS: Raspbian

Come monitor ho utilizzato un HP Compaq LA1051g fornito dalla scuola.

# Progettazione

## Design dell’architettura del sistema

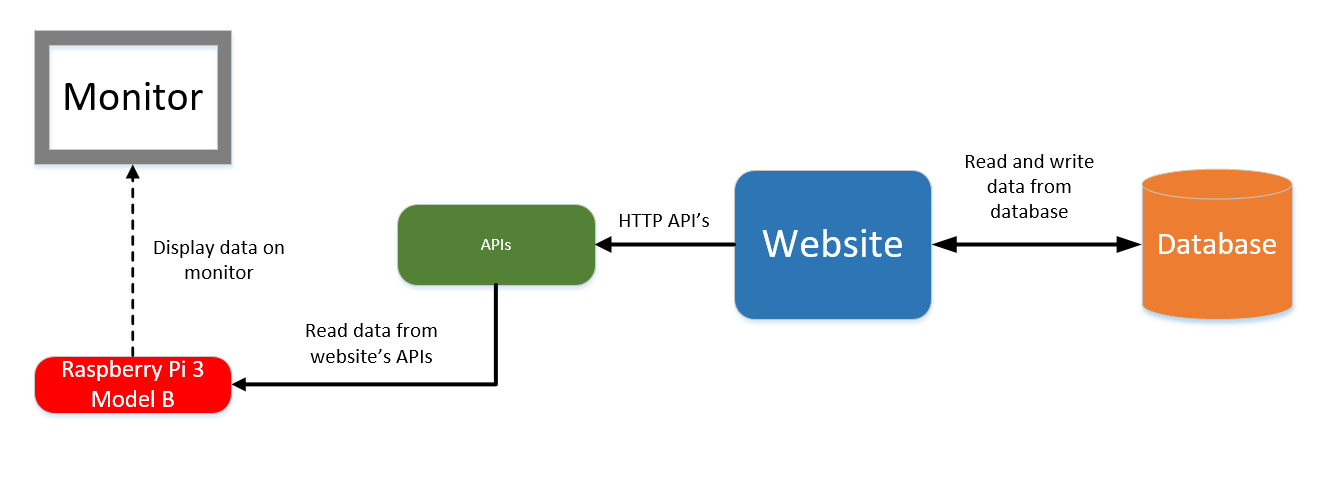


Figure 7 - Struttura del programma

Lo schema sovrastante è composto da 8 parti principali collegate tra loro in questo modo: Il sito web si interfaccerà con il database per la lettura e scrittura dei dati e fornirà delle API HTTP che verranno utilizzate dal Raspberry Pi per leggere i dati i quali verranno parsati (formattati) e mostrati all’utente finale tramite un Monitor.

## Design dei dati e database

Questo è il diagramma *entity relation* del database della mia applicazione:

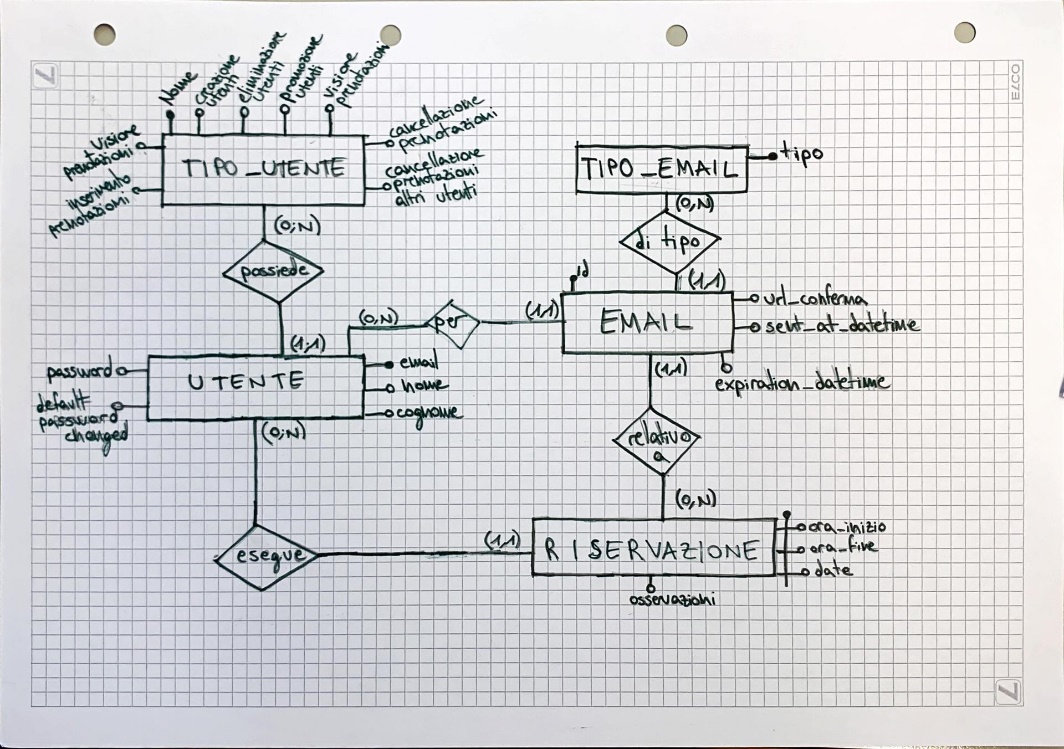


Figure 8 - Diagramma ER database

Lo schema è composto di 5 entità principali, di quali due non sono altro che generalizzazioni (*tipo\_utente* e *tipo\_email*).

L’entità *tipo\_utente* possiede molti attributi; ognuno di questi attributi è un permesso dell’utente. Ho deciso di farlo in questo modo per rendere l’applicazione “aggiornabile”: se in futuro si avrà la necessità di creare un nuovo tipo di utente in questa maniera non ci sarà la necessità di cambiare codice ma bisognerà soltanto aggiungere un record all’interno della tabella. Mi sono ispirato al funzionamento dei permessi di MySQL.

L’entità *utente* come si può intuire conterrà i dati dell’utente che utilizzerà l’applicazione tra questi anche il tipo di permessi che ha (indicato tramite l’associazione intitolata “*possiede”*). L’attributo *default\_password\_changed* servirà per tenere in memoria se l’utente ha cambiato la password al primo accesso o meno.

L’entità *riservazione* conterrà tutti i dati utili per l’identificazione della riservazione, quindi chi ha riservato e quando ha riservato.

L’entità *email* è collegata a ben 3 entità: utente, riservazione e *tipo\_email*. Questo perché un’e-mail è inviata ad un *utente* preciso nello specifico ed è riguardante una specifica *riservazione*. L’entità *tipo\_mail* è utilizzata per specificare se è un’e-mail di conferma di eliminazione o di creazione.

Gli attributi dell’entità *mail* servono per tenere in memoria l’url di conferma e il tempo di vita dell’url (24 ore solitamente).

### Schema logico

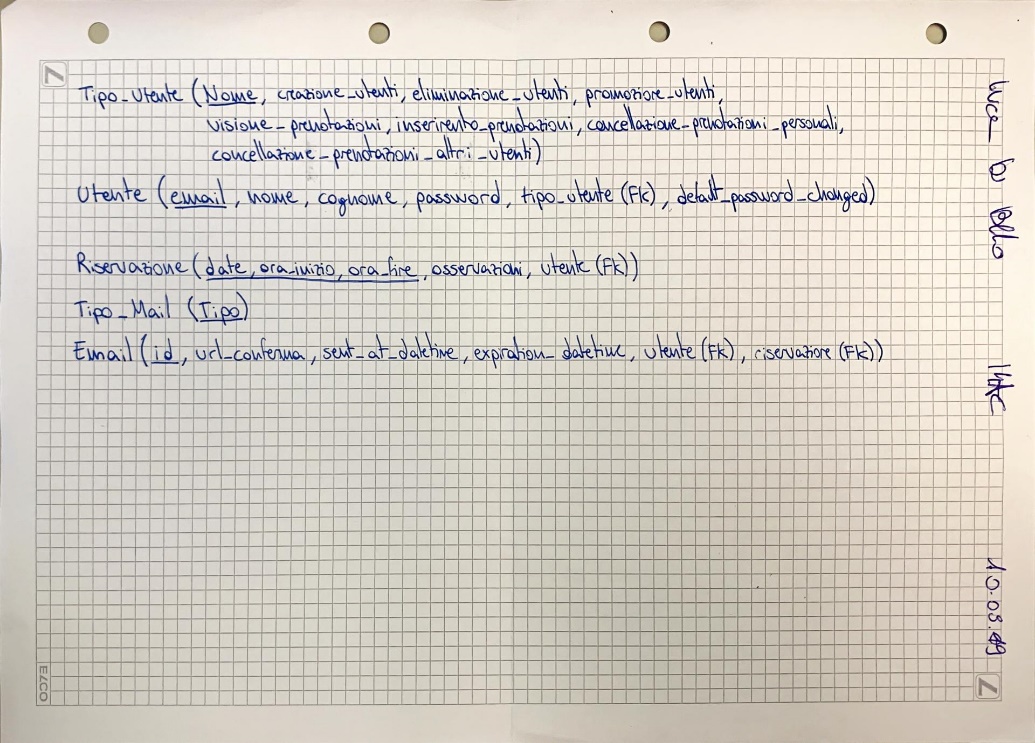
Questo è lo schema logico del database prodotto tramite il diagramma ER disegnato in precedenza:

Figure 9 - Schema logico database

### Tipi di dati

|  |  |
| --- | --- |
| Tipo\_Utente | |
| Nome attributo | Tipo attributo |
| Nome | *Primary Key, Varchar(50)* |
| Creazione\_utenti | *Bit, default 0* |
| Eliminazione\_utenti | *Bit, default 0* |
| Promozione\_utenti | *Bit, default 0* |
| Visione\_prenotazioni | *Bit, default 0* |
| Inserimento\_prenotazioni | *Bit, default 0* |
| Cancellazione\_prenotazioni\_personali | *Bit, default 0* |
| Cancellazione\_prenotazioni\_altri\_utenti | *Bit, default 0* |

|  |  |
| --- | --- |
| Utente | |
| Nome attributo | Tipo attributo |
| Email | *Primary Key, Varchar(255)* |
| Password | *Varchar(255)* |
| Nome | *Varchar(100)* |
| Cognome | *Varchar(100)* |
| Tipo\_utente | *Foreign Key, Varchar(50)* |
| Default\_password\_changed | *Bit, default 0* |

|  |  |
| --- | --- |
| Riservazione | |
| Nome attributo | Tipo attributo |
| Id | *Primary Key, Int(11), Auto Increment* |
| Date | *Primary Key, Datetime* |
| Ora\_inizio | *Time* |
| Ora\_fine | *Time* |
| Osservazioni | *Varchar(512)* |
| Utente | *Foreign Key, varchar(255)* |

|  |  |
| --- | --- |
| Tipo\_mail | |
| Nome attributo | Tipo attributo |
| Nome | *Primary Key, Varchar(50)* |

|  |  |
| --- | --- |
| Email | |
| Nome attributo | Tipo attributo |
| Id | *Primary Key, Int(11), Auto Increment* |
| Url\_conferma | *Varchar(512)* |
| Sent\_at\_datetime | *Datetime* |
| Exipiration\_datetime | *Datetime* |
| Utente | *Foreign Key, varchar(255)* |
| Riservazione | *Foreign Key, Int(11)* |

ERRORE: riservazione non ha un ID, dovrei esportare 3 foreign key 🡪 Mettere apposto lo schema logico ed ER

## Design delle interfacce

ho iniziato a fare i mockup delle pagine web utilizzando un sito web chiamato

Ho realizzato 4 mockup utilizzato un sito web chiamato *mockflow.com:* pagina di login, pagina gestione utenti, pagina gestione prenotazioni e pagina visualizzazione prenotazioni (che verrà creata dal raspberry Pi).

Questo è il mockup che raffigura la pagina di gestione degli utenti:

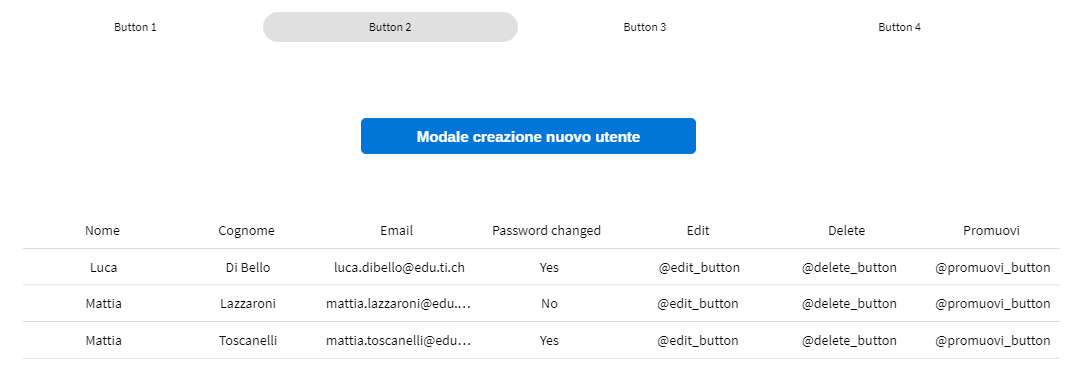


Figure 10 - Pagina gestione utenti

Questa è il mockup che raffigura la pagina di gestione delle prenotazioni:



Figure 11 - Pagina gestione prenotazioni

Questo è il mockup che raffigura la pagina di login, utilizzata per accedere al sito web:

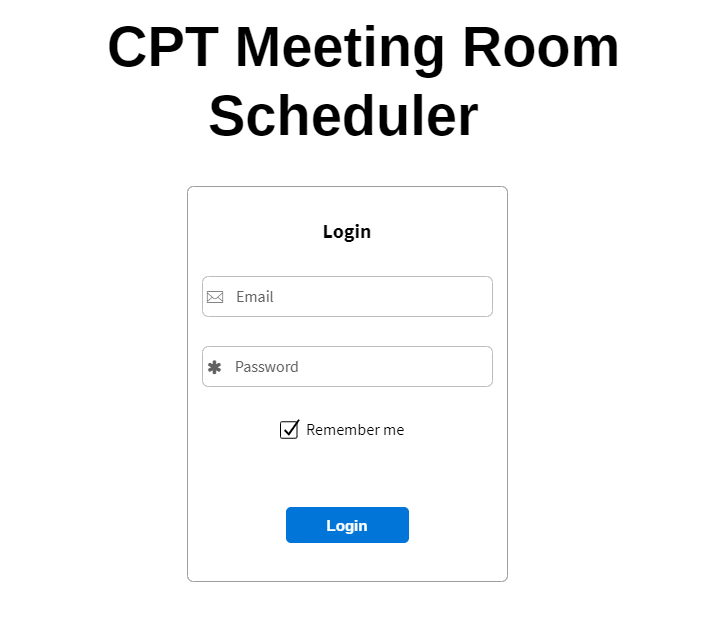


Figure 12 - Pagina di login

Ed infine questo è il mockup che raffigura la pagina di visualizzazione delle prenotazioni che verrà mostrata sul monitor:



Figure 13 - Visualizzazione prenotazioni su monitor

## Design procedurale

Descrive i concetti dettagliati dell’architettura/sviluppo utilizzando ad esempio:

* Diagrammi di flusso e Nassi.
* Tabelle.
* Classi e metodi.
* Tabelle di routing
* Diritti di accesso a condivisioni …

Questi documenti permetteranno di rappresentare i dettagli procedurali per la realizzazione del prodotto.

# Implementazione

## Gestione permessi

La gestione dei permessi è stata implementando utilizzando 2 classi principali ed una di supporto:

* Classe *PermissionModel*
* Classe *Permissions*
* Classe *PermissionManager* (classe di supporto)

Nei capitoli successivi sarà descritta ogni singola classe, spiegando sia la sua struttura (metodi e/o attributi) sia l’utilizzo di essa all’interno del sistema.

Questa è la struttura delle tre classi:

### Classe PermissionModel e classe Permissions

La classe PermissionModel contiene svariate funzioni utilizzate per la lettura dei permessi salvati all’interno del database. Queste classi sono fondamentali per la gestione dei permessi del sistema. Nei vari sottocapitoli successivi andrò a spiegare l’utilizzo ed il funzionamento di tutti i metodi della classe aggiungendo degli spezzoni di codice nelle parti critiche e/o di difficile comprensione.

#### getLocalPermission

Metodo wrapper che permette di caricare i permessi di un utente tramite la chiave del permesso inserite come parametro nel costruttore (esempio di chiave: *admin, user)*. Questo viene fatto leggendo i dati contenuti nella tabella *tipo\_utente* presente all’interno del database *cptmrs*. Tutti i dati letti vengono parsati ed inseriti all’interno di un oggetto *Permission* per rendere più semplice i controlli su cosa può o non può fare un determinato utente.

Questo metodo viene utilizzato dal sistema

Definisco questo metodo *wrapper* dato che non fa altro che richiamare il metodo *read\_data\_from\_database* utilizzando come parametro la chiave inserita nel costruttore e passare il risultato al metodo *parse\_data*. Quest’ultimo metodo non fa altro che fare un parsing dei dati letti dal database inserendoli in un oggetto di tipo *Permission*. Ecco il codice del metodo:

*// Permissions for local users*

**public** **function** getLocalPermissions(): Permissions{

$data = $this->read\_from\_database($this->PERM\_KEY);

return $this->parse\_data($data, $this->PERM\_KEY);

}

#### getLdapPermission

Metodo wrapper che permette di caricare i permessi di default di un utente che effettua il login utilizzando il protocollo *LDAP*. Questo viene fatto utilizzando sempre il metodo *read\_data\_from\_database* ma passando come chiave una definita nella classe *config.php.* Questa costante è chiamata ‘DEFAULT\_USER\_PERMISSION\_GROUP’ ed ha come valore di default *user*.

Questo metodo viene richiamato quando il sistema rileva un login utilizzando *LDAP*.

Proprio come per *getLocalPermission* anche questo metodo è definito *wrapper* proprio perché non fa altro che chiamare il metodo *read\_data\_from\_database* utilizzando la chiave salvata all’interno della costante *DEFAULT\_USER\_PERMISSION\_GROUP.* I dati ritornati dal metodo vengono passati nuovamente al metodo *parse\_data* che genera un oggetto *Permissions* utilizzandoli.

#### getUserPermissionGroup

Questo metodo è utilizzato per ricavare il nome del permesso assegnato ad un utente, il quale username è passato come parametro al metodo, se per qualche motivo non viene trovato nessun permesso il metodo ritorna il valore salvato all’interno della costante *DEFAULT\_USER\_PERMISSION\_GROUP* (che di default il valore “user”).

#### read\_data\_from\_database

Quest’ultimo metodo è sicuramente il fulcro di entrambe le classi. Esso è una funzione privata e viene utilizzato per leggere i dati relativi ad un permesso.

Si passa come parametro il nome del permesso (es: admin, user, ...) ed il metodo ritorna un array che contiene tutti i dati relativi ad esso.

Questo array si può passare al metodo ***parse\_data*** per convertire l’array in un oggetto *Permissions.*

## Pannello admin

L’applicazione presenta un pannello admin che permette la completa gestione degli utenti e delle prenotazioni. Esso è accessibile soltanto agli utenti con gli utenti amministratori (permessi di tipo “*admin”*).

Quando si accede al controller admin (quindi quando si digita l’url “..../admin”) il sistema controlla se i permessi dell’utente siano corretti. Se l’utente dispone dei permessi necessari il sistema carica la pagina utilizzando la libreria *ViewLoader*, altrimenti viene rimandato alla homepage.

Il pannello admin presenta una logica che permette di nascondere le funzionalità dove l’utente non ha permessi: Se un utente non ha nessun permesso per la modifica degli utenti, la pagina oltre a diventare inaccessibile, sparisce il link dalla barra di navigazione. Esegue la stessa cosa per la gestione delle prenotazioni.

Se invece un utente (per esempio) possiede i permessi per leggere e modificare gli utenti presenti nel sistema ma non ha quelli necessari per la creazione e l’eliminazione di essi i due tasti relativi a queste azioni vengono nascosti automaticamente.

Ho scelto di sviluppare questo sistema per rendere invisibili a determinati utenti determinate azioni e quindi limitare possibili problemi di sicurezza e *bugs,* i quali, se a livello di pannello admin, potrebbero risultare molto dannosi e rischiano di compromettere il corretto funzionamento dell’applicazione.

### Gestione utenti

Il pannello admin presenta una sezione dedicata per la gestione di ogni singolo utente registrato nel database locale dell’applicazione. All’interno di questa sezione si può eliminare, modificare, creare e promuovere utenti. Come ho detto in precedenza, questa pagina è dinamica. Essa cambia a seconda dei permessi dell’utente connesso in quel momento (vedi capitolo 3.2: Pannello admin).

Nei vari sottocapitoli successivi spiegherò ogni singola sezione della pagina, andando a spiegare sia il perché di certe scelte di design/strutturali che il come ogni componente opera con il sistema.

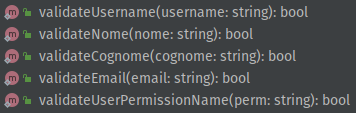
#### UserValidator

Ho deciso di descrivere per primo il sistema di validazione dati dato che viene utilizzato da le funzioni più importanti/critiche della pagina: aggiunta e modifica degli utenti.

Ritengo che questa classe sia la più importante tra quelle utilizzate a livello di gestione degli utenti. Essa permette e controlla che i dati siano sempre corretti e precisi, senza permettere nessun tipo di attacco su di essi: XSS Injection (<https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)>) e SQL Injection (<https://www.owasp.org/index.php/SQL_Injection>).

Questi attacchi potrebbero, se utilizzati da un utente malintenzionato, compromettere il corretto funzionamento del sistema e mettere a rischio i dati degli utenti.

La classe *UserValidator* fornisce metodi statici che permettono di validare ogni dato relativo agli utenti*.* Questa classe infatti viene utilizzata per controllare i dati prima dell’inserimento/modifica di quelli presenti nel database. Questa è la sua struttura:

Spiegazione dettagliata del funzionamento di ogni metodo di validazione:

* *validateUsername*

Questo metodo riceve una stringa (username da validare) come parametro. Questa stringa viene controllata utilizzando la seguente *regular expression* (RegEx): /^[a-zA-Z0-9.]{1,255}$/

Questo RegEx controlla che la stringa sia alfanumerica (lettere e numeri) e permette di contenere i punti. Ho deciso di permettere i punti all’interno degli *username* per permettere la creazione di nomi utenti di questo tipo: *luca.dibello, pinco.pallino, ecc.*

Questa espressione regolare controlla anche che la lunghezza della stringa sia compresa tra 1 e 255 caratteri.

* *validateNome*

Questo metodo riceve una stringa (*nome* da validare) come parametro. Questa stringa viene controllata utilizzando il metodo *ctype\_apha* già presente in PHP. Questo è il codice che gestisce la validazione del nome:

setlocale(LC\_ALL, 'it\_CH.utf8');

return ctype\_alpha($nome);

La prima stringa di codice imposta il tipo di caratteri da validare. È molto importante impostare i caratteri per la gestione degli accenti: è, ö, é, ì, ecc.

La seconda stringa invece effettua la chiamata al metodo che va a fare la vera e propria validazione.

* *validateCognome*

Questo metodo riceve una stringa (cognome da validare) come parametro. Questa stringa viene controllata utilizzando lo stesso metodo della validazione del nome (*validateNome*) ma con una piccola modifica nel codice, la quale permette di avere dei cognomi composti (es: Di Bello, De Maria, ...)

Questa è la stringa che è stata modificata:

return ctype\_alpha(str\_replace(" " ,"", $cognome));

Come si può vedere dal codice il cognome viene passato nel metodo *str\_replace* il quale rimuoverà tutti gli spazi contenuti nel cognome. Quindi il cognome *Di Bello* verrà validato come se fosse il cognome *DiBello.*

Ho scelto di utilizzare questo modo per la gestione dei cognomi composti dato che il carattere spazio non può provocare nessun danno/problema di sicurezza nel sistema.

* *validateEmail*

Questo metodo riceve una stringa (email da validare) come parametro. Questa stringa viene poi controllata in due *step*:

1. Controllo formattazione email (controllo classico)
2. Controllo dominio

Il primo step controlla il formato della mail utilizzando un metodo già presente in PHP chiamato *filter\_var* il quale, se impostato con un flag, permette di controllare se un email è formattata correttamente o meno.

Il secondo step invece controlla se il dominio dell’email da validare (quindi la parte dopo la ‘@’: [luca.dibello@](mailto:luca.dibello@samtrevano.ch))**[samtrevano.ch](mailto:luca.dibello@samtrevano.ch))**) è uguale a quello impostato nella configurazione dell’applicativo (file di config: *config/config.php*).

Se entrambi i controlli vengono passati l’email è valida, altrimenti ritornerà *false*.

* *validateUserPermissionName*

Questo metodo riceve il nome di un gruppo di permessi (*es: admin, user, ...)* e controlla se effettivamente è un gruppo esistente all’interno del sistema o meno. Questo lo fa appoggiandosi al metodo *getUniquePermissionTypes* della classe *PermissionModel.*

Questo metodo (come dice anche il suo nome) ritorna una lista che contiene i nomi di tutti i gruppi presenti nel sistema.

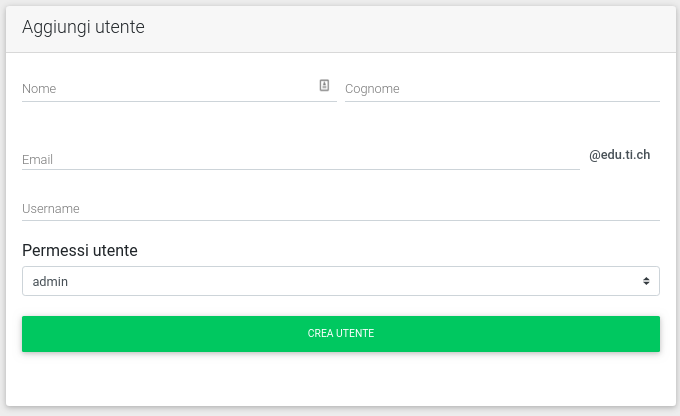
Una volta aver interrogato il database e ricevuta la lista, il metodo controlla se il nome passato come parametro è contenuto nella lista:

return in\_array($perm, PermissionModel::getUniquePermissionTypes());

Se il nome è contenuto significa che è un nome valido, altrimenti il metodo ritornerà *false.*

#### Aggiunta utenti

Questa sezione della pagina permette l’inserimento di un nuovo utente tramite un modulo (*form*). Questo è un piccolo screenshot che mostra com’è strutturato e che dati necessita:

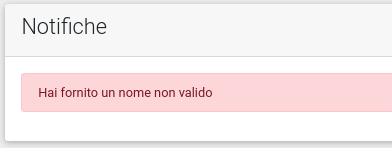
Come si può vedere dall’immagine il *form* richiede all’amministratore di inserire il proprio nome, cognome la parte iniziale della mail (p.e: **[luca.dibello](mailto:luca.dibello@edu.ti.ch))**[@edu.ti.ch)](mailto:luca.dibello@edu.ti.ch)) ed i permessi che avrà l’utente all’interno dell’applicazione.

I permessi disponibili vengono caricati automaticamente andando a leggere i record presenti nella tabella *tipo\_utente* del database *cptmrs*. Questo viene fatto utilizzando la classe *PermissionModel* (vedi capitolo 3.1.1).

I dati vengono controllati da back-end tramite la classe *UserValidator,* la quale è stata creata ad hoc proprio per la gestione degli utenti.

Se i dati inseriti sono validi, essi vengono inseriti nel database e quindi l’utente viene creato correttamente. Se invece i dati risultano non validi l’operazione relativa alla creazione dell’utente viene annullata e viene mostrato un errore nella sezione *Notifiche* posta in cima alla pagina.

Questo è un esempio di notifica la quale è stata generata inserendo un nome non valido:



#### Visione utenti locali

In questa sezione si possono eseguire molteplici operazioni relative al

PERMESSI + OPERAZIONI DISPONIBILI

### Gestione prenotazioni

# Test

## Protocollo di test

Definire in modo accurato tutti i test che devono essere realizzati per garantire l’adempimento delle richieste formulate nei requisiti. I test fungono da garanzia di qualità del prodotto. Ogni test deve essere ripetibile alle stesse condizioni.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case:**  **Riferimento**: | TC-001  REQ-012 | **Nome:** | Import a card with KIC, KID and KIK keys, but not shown with the GUI |
| **Descrizione:** | Import a card with KIC, KID and KIK keys with no obfuscation, but not shown with the GUI | | |
| **Prerequisiti:** | Store on local PC: Profile\_1.2.001.xml (appendix n\_n) and Cards\_1.2.001.txt (appendix n\_n).  PIN (OTA\_VIEW\_PIN\_PUK\_KEY) and ADM (OTA\_VIEW\_ADM\_KEY) user right not set. | | |
| **Procedura:** | 1. Go to “Cards manager” menu,  in main page click “Import Profiles” link, Select the “1.2.001.xml” file, Import the Profile 2. Go to “Cards manager” menu,  in main page click “Import Cards” link, Select the “1.2.001.txt” file, Delete the cards,  Select the “1.2.001.txt” file, Import the cards 3. Research the “41795924770” Card, Click the imsi card link Check the card details 4. Execute the SQL: SELECT imsi, dir, keyset, cntr, rawtohex(kickey), rawtohex(kidkey), rawtohex(kikkey), rawtohex(chv), rawtohex(dap)FROM otacardkey a where imsi='340041795924770' ORDER BY keyset; | | |
| **Risultati attesi:** | Keys visible in the DB (OtaCardKey) but not visible in the GUI (Card details) | | |

## Risultati test

Tabella riassuntiva in cui si inseriscono i test riusciti e non del prodotto finale. Se un test non riesce e viene corretto l’errore, questo dovrà risultare nel documento finale come riuscito (la procedura della correzione apparirà nel diario), altrimenti dovrà essere descritto l’errore con eventuali ipotesi di correzione.

## Mancanze/limitazioni conosciute

Descrizione con motivazione di eventuali elementi mancanti o non completamente implementati, al di fuori dei test case. Non devono essere riportati gli errori e i problemi riscontrati e poi risolti durante il progetto.

# Consuntivo

Consuntivo del tempo di lavoro effettivo e considerazioni riguardo le differenze rispetto alla pianificazione (cap 1.7) (ad esempio Gannt consuntivo).

# Conclusioni

Quali sono le implicazioni della mia soluzione? Che impatto avrà? Cambierà il mondo? È un successo importante? È solo un’aggiunta marginale o è semplicemente servita per scoprire che questo percorso è stato una perdita di tempo? I risultati ottenuti sono generali, facilmente generalizzabili o sono specifici di un caso particolare? ecc

## Sviluppi futuri

Migliorie o estensioni che possono essere sviluppate sul prodotto.

## Considerazioni personali

Cosa ho imparato in questo progetto? ecc

# Bibliografia

## Bibliografia per articoli di riviste:

1. Cognome e nome (o iniziali) dell’autore o degli autori, o nome dell’organizzazione,
2. Titolo dell’articolo (tra virgolette),
3. Titolo della rivista (in italico),
4. Anno e numero
5. Pagina iniziale dell’articolo,

## Bibliografia per libri

1. Cognome e nome (o iniziali) dell’autore o degli autori, o nome dell’organizzazione,
2. Titolo del libro (in italico),
3. ev. Numero di edizione,
4. Nome dell’editore,
5. Anno di pubblicazione,
6. ISBN.

## Sitografia

1. URL del sito (se troppo lungo solo dominio, evt completo nel diario),
2. Eventuale titolo della pagina (in italico),
3. Data di consultazione (GG-MM-AAAA).

**Esempio:**

* http://standards.ieee.org/guides/style/section7.html, *IEEE Standards Style Manual*, 07-06-2008.

# Allegati

Elenco degli allegati, esempio:

* Diari di lavoro
* Codici sorgente/documentazione macchine virtuali
* Istruzioni di installazione del prodotto (con credenziali di accesso) e/o di eventuali prodotti terzi
* Documentazione di prodotti di terzi
* Eventuali guide utente / Manuali di utilizzo
* Mandato e/o Qdc
* Prodotto
* …